СОДЕРЖАНИЕ

Практическая работа № 1. Создание классов

Теоретические сведения

Класс есть расширение понятия структуры языка С++. Он позволяет создавать типы и определять функции, которые задают поведение типа. Каждый представитель класса называется объектом.

Определение класса

Определение класса напоминает определение структуры в С++, за исключением того, что

-оно обычно содержит одну или несколько спецификаций доступа(public, protected, private);

-вместо ключевого слова struct используется слово class;

-оно обычно включает в себя функции (функции-элементы или методы) на ряду с элементами данных;

-обычно в нем имеются некоторые специальные функции, такие как конструктор(функция с тем же именем, что и сам класс) и деструктор (функция, именем которой является имя класса с префиксом-тильдой ~).

Пример определения класса.

```
class str{
        char *s;    //элемент-данное
            public:     //спецификатор публичного доступа
str(char *word);        //функция-элемент: конструктор
~str();                 //функция-элемент: деструктор
void write();           //функция-элемент: метод печати
   };
```

Управление доступом

В С++ можно ограничить видимость данных и функций класса при помощи меток public, protected, private. Метка -спецификатор доступа применяется ко всем элементам класса, следующим за ней, пока не встретится другая метка или кончится определение класса.

Метка-спецификатор private используется, если элементы-данные и функции-элементы должны быть доступны только для функций-элементов данного класса.

Метка-спецификатор public используется тогда, когда элементы-данные и функции-элементы класса доступны для функций-элементов и других функций программы, в которой имеется представитель класса.

Метка-спецификатор protected используется в том случае, когда элементы данных и функции-элементы доступны для функций-элементов данного класса и классов производных от него .

В классе элементы имеют по умолчанию доступ private.

Элементы класса

Элементы класса делятся на две основные категории:

-данные, называемые элементами-данными;

-код, называемый элементами-функциями или методами.

Элементы данных

Элементы-данные классов С++ такие же, как и элементы структур языка С++ с некоторыми дополнениями:

-элементами-данными могут быть перечислимые типы, битовые поля или представители ранее объявленного класса;

-элемент-данное класса может быть указателем или ссылкой на представитель этого класса.

Элементы функции

Элемент-функция является функцией, объявленной (описанной) внутри определения

класса. Тело функции может также определяться внутри определения класса, в этом случае функция называется встроенной (inline) функцией-элементом. Когда тело функции определяется вне тела класса, перед именем функции ставится префикс из имени класса и операции разрешения видимости (::). Например:

```
class  str {
      char *s;                                          //  указатель на строку
      public:
str(char *word){ s=new char[strlen(word)+1];    // встроегнный конструктор
                    strcpy(s,word);
                  };
~str(){ delete [ ]s; };                            // встроенный деструктор
void write();                                      // функция-элемент
             };
void str::write()
/*    определение функции-элемента            */
{
          puts("");
           puts(s);
}
```

Доступ к элементам-данным

Функции-элементы находятся в области действия класса, в котором они определены. Т.о. они могут обращаться к любому элементу класса Б, используя просто имя переменной. Обычные функции или функции-элементы другого класса могут получить доступ к элементам-данным с помощью операции . или ->, применяемых к представителю или указателю на представитель класса.

Пример.

```
class   coord{
                  public:int x,y; // координаты x  и y
          };
void main()
{
  coord   org;                    // представитель класса координат
       coord *orgptr=&org;              // указатель на представитель класса
  org.x=0;                        // задание значения координаты x
  orgptr->y=0;                    // задание значения координаты y
}
```

Вызов функций-элементов

Функции-элементы класса могут вызывать другие функции-элементы того же класса, используя имя функции. Обычные функции или функции-элементы других классов могут вызывать элементы класса с помощью операций . и ->, применяемых к представителю или указателю на представитель класса.

Пример.

```
class coord{

          int x,y;                // координаты x и y
public:
void setcoord(int _x, int _y)    // функция задания значений координат
      {   x=_x; y=_y;};
void getcoord(int &_x,int &_y)         // функция полуцения значений координат
      {  _x=x; _y=y;};
                };
void main()
```

```
{
      coord   org;                    // представитель класса координат
coord *org*ptr=&org;          // указатель на представитель класса
org.setcoord(10.10);          // вызов функции-элемента задания
                              // значений
int col,row;
orgptr->getcoord(col,row);          // вызов функции-элемента получения
                              // значений координат
}
```

Взаимосвязь классов и структур

В языке C++ расширены возможности структуры, если сравнивать их с языком C. В C++ классы и структуры тесно взаимосвязаны. Фактически, за одним исключением, они взаимозаменяемы, поскольку структура в C++ может включать как данные, так и код, который может манипулировать этими данными таким же образом, как и класс. Структуры также могут содержать конструкторы и деструкторы. Единственное отличие между ними связано с тем, что по умолчанию члены класса имеют в качестве спецификатора доступа private, тогда как спецификатором доступа членов структуры служит public. Согласно формальному синтаксису C++, ключевое слово struct определяет тип класса. В качестве примера рассмотрим программу:

```
#include <iostream.h>
struct cl {
int get_i(); // публичные
void put_i(int j); // по умолчанию
private:
int i;
};
int cl::get_i()
{
return i;
}
void cl::put_i(int j)
{
i = j;
}
int main()
{
cl s;
s.put_i(10);
cout << s.get_i();
return 0;
}
```

В этой простой программе определен тип структуры с именем cl, в которой функции get_i() и put_i() являются соответственно public и private. Обратим внимание, что структура использует ключевое слово private для введения частных членов структуры.

Ниже приведена эквивалентная программа, использующая класс вместо структуры.

```
#include <iostream.h>
class cl {
int i; // по умолчанию частный
public:
int get_i();
```

```
void put_i(int j);
};
int cl::get_i()
{
return i;
}
void cl::put_i(int j)
{
i = j;
}
int main()
{
cl s;
s.put i(10);
cout << s.get_i();
return 0;
}
```

В большинстве случаев программисты, работающие на языке C++, используют классы для определения объектов, содержащих и данные и код. Они используют структуры для определения объектов, содержащих только данные. Это означает, что структуры используются обычно точно в том же стиле, что и структуры в языке C. Тем не менее время от времени встречается код на языке C++, который использует расширенные возможности структур.

Задание 1

Откомпилировать пример. Получить результаты.

Программа с использованием классов и их функций-членов
Составить класс для исследования оценок студента в сессии на качество (на «4» и «5»)и неуспеваемость (наличие хотя бы одной оценки «2»). Используя этот класс, составить программу нахождения количества студентов группы, успевающих на «4» и «5» и неуспевающих, а также вычислить процентное соотношение для каждого из этих показателей. Студенты в сессию сдавали 4 экзамена.
Для реализации этого алгоритма составим класс Student, содержащий закрытую секцию с массивом оценок одного студента s[4], и открытую секцию с функциями-членами ввода этого массива, его вывода, исследования на наличие только оценок 4 и 5, наличие хотя бы одной оценки 2.

```
#include <iostream.h>
class Student {
public:
void cinr();
void coutr();
int r45();
int r2();
private:
int s[4];};  //Массив оценок студента
void Student :: cinr() //Функция ввода оценок студента
{ cout<<”Введите оценки студента”;
for (int k=0; k<4;k++) cin>>s[k];
}
```

```
void Student :: coutr() //Функция вывода оценок студента
{ cout<<" Оценки студента: \n";
for (int k=0; k<4;k++) cout<<s[k];
}
int Student :: r45() //Функция исследования  оценок студента на «4» и «5»
{ int p=1;
for (int k=0; k<4;k++)
 if(s[k]<4) p=0;
return p;
}
int Student :: r2() //Функция исследования  оценок студента на «2»
{ int p=0;
for (int k=0; k<4;k++)
 if(s[k]==2) p=1;
return p;
}
void main()
{ int n; //Количество студентов в группе
cin>>n;
int rx45=0;
int rx2=0;
Student sr;
for (int k=1; k<=n;k++)
{
sr.cinr();
sr.coutr();
rx45= rx45+sr.r45();
rx2+=sr.r2();
}
cout<<"На 4 и 5 сдали "<<rx45<<" студентов или"<<rx45*100.0/n<<"%";
cout<<"Хотя бы одну 2 имеют "<<rx2<<" студентов или"<<rx2*100.0/n<<"%";
}
```

Задание 2

Разработать классы для описанных ниже объектов. Включить в класс методы set (…), get (…), show (…). Определить другие методы. Написать программу, демонстрирующую работу с этим классом. Класс соответствует индивидуальному варианту.

1. **Student**: Фамилия, Имя, Отчество, Дата рождения, Адрес, Средний балл, Факультет, Курс. Определить является ли студент «двоечником», «троечником», «хорошистом», «отличником».
2. **Abiturient**: Фамилия, Имя, Отчество, Адрес, Оценки. Задать проходной балл для поступления и определить, поступил ли абитуриент в ВУЗ.
3. **Aeroflot**: Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели. Определить осуществляет ли самолет рейсы на выходных.
4. **Worker**: Фамилия и инициалы, Должность, Год поступления на работу, Зарплата. Определить стаж работы сотрудника.
5. **Train**: Пункт назначения, Номер поезда, Время отправления,

Число общих мест, Купейных, Плацкартных. Определить процентное соотношение купейных и плацкартных мест в поезде.

6. **Product**: Наименование, Производитель, Цена, Срок хранения, Дата выпуска. Определить пригодность продукта к использованию.
7. **Patient**: Фамилия, Имя, Отчество, Адрес, Номер медицинской карты, Показатель

температуры. Определить отклонения температуры пациента от нормального показателя.

8. **Bus**: Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег. Определить срок эксплуатации автомобиля.

9. **Customer**: Фамилия, Имя, Отчество, Адрес, Телефон, Номер кредитной карточки, Номер банковского счета. Определить находиться ли номер кредитной карточки в заданном интервале.

10. **File**: Имя файла, Размер, Дата создания, Количество обращений. Определить возможно ли записать файл на CD-диск.

11. **House**: Адрес, Этаж, Количество комнат, Площадь. Определить количество квартир, имеющих площадь, превосходящую заданную.

12. **Phone**: Фамилия, Имя, Отчество, Адрес, Номер, Время внутригородских разговоров, Время междугородних разговоров. Определить процент внутригородских и междугородних разговоров абонента.

13. **Person**: Фамилия, Имя, Отчество, Адрес, Пол, Образование, Год рождения. Определить является ли человек совершеннолетним.

Задание 3

Разработать эквивалентную программу (варианты задания 2), но используя структуры, а не классы.

Контрольные вопросы.

1. Что представляет собой класс?
2. Какие спецификации доступа используются при описании класса?
3. Что является элементами класса?
4. Как осуществляется доступ к элементам класса?

Содержание отчёта:

- ✓ Постановка задачи.
- ✓ Текст программы.
- ✓ Комментарии к введённым обозначениям и описание используемых функций.
- ✓ Результаты работы программы.

Практическая работа № 2.Работа с конструкторами и деструкторами

Теоретические сведения

Указатель this

Каждая нестатическая функция-элемент имеет доступ к объекту для которого вызвана, через ключевое слово this.

Типом this является: тип_класса *.

Пример.

```
class simple
{
  public:
  simple();
void greet()  { printf(“ Hello!”);};
};
simple::simple()
{
  greet();          // вызов
  this->greet();            // функции
  (*this).greet();          // greet()
}
```

Т.к. функции-элементы могут обращаться ко всем элементам класса просто по имени, в основном указатель this используется для возврата указателя (return this) или ссылки (return *this) на подразумеваемый объект.

Конструктор

Конструктор инициализирует представитель класса (объект) и является функцией-элементом с тем же именем, что и класс. Конструктор вызывается компилятором всегда, когда создается представитель класса.

Для конструкторов выполняются следующие правила:

-для конструктора не указывается возвращаемый тип;

-конструктор не может возвращать значение;

-конструктор не наследуется.

Деструктор

Деструктор является дополнением конструктора. Он имеет то же имя, что и класс, но с префексом-тильдой (~). Он вызывается всякий раз, когда уничтожается представитель класса. Для деструктора существуют следующие правила:

-деструктор не может иметь аргументов;

-деструктор не может возвращать значения;

-деструктор не наследуется.

Конструктор копирования

Многие программы для своей работы требуют обязательного наличия специального конструктора копирования.

Этот конструктор, как и остальные, вызывается неявным образом, когда происходит копирование одного объекта в поле другого. Последнее имеет место в следующих случаях:

- При описании нового объекта с инициализацией другим объектом;
- при передаче фактического объекта в функцию, когда формальным параметром является объект;
- при возврате значения функции типа класса.

Во всех указанных случаях в поле одного объекта копируется другой. При этом, если класс имеет в качестве данных-членов указатели, то копируются и эти указатели. Последнее

может приводить к ошибкам, так как указатели разных объектов будут указывать на один и тот же адрес, чего не должно быть. Более того, в случае формального объекта при выходе из функции происходит уничтожение поля формального объекта, что может приводить и к изменению указателя фактического параметра. Автоматическое создание нового указателя при копировании одного объекта в другой осуществляется с помощью конструктора копирования.

Конструктор копирования имеет только один формальный параметр, который задаётся в виде константной ссылки типа данного класса. Так для класса Student он имеет вид

Student:: Student(const Student& st) {Тело конструктора}

С помощью этого конструктора функция создаёт новые поля, куда копирует указательные элементы формального параметра-объекта. Например, класс

```
class Student{
private:
int n;
int *s;
public:
Student();
Student(int n);
Student(const Student& st)
{n=st.n;
If(st.s) {s=new int[n];
for (int k=0; k<n;k++) s[k]=st.s[k];
}
else s=0;
}
~Student();
void cinr();
void coutr();
};
```

Дружественные классы

Разрешить элементам другого класса полный доступ к элементам данного класса, объявленным как private или protected, можно включив в определение данного класса описание friend.

Пример.

```
class myclass
{
  friend class another_class;
};
```

Задание 1

Откомпилировать примеры и получить результаты.
Программа с использованием конструкторов и деструкторов.
Пример 1.

```
# include <iostream>
 using namespace std;
// описание класса Point
 class Point {
int x, y; // координаты точки, по умолчанию private
public:
```

```
// конструктор присваивает переменным класса x и y
// начальные значения соответственно x0 и y0
Point(int x0, int y0) { x = x0; y = y0; }
// функция вывода координат точки на экран
 void ShowPoint()
{ cout << "\nx = " << x; cout << "\ny = " << y; }
};
void main()
{ Point A(1,3); // создаем объект A типа Point
A.ShowPoint(); // выводим координаты точки A на экран }
```

Пример 2.

Составить класс для исследования оценок студента в сессии на качество (на «4» и «5»)и неуспеваемость (наличие хотя бы одной оценки «2»). Используя этот класс, составить программу нахождения количества студентов группы, успевающих на «4» и «5» и неуспевающих. Вычислить процентное соотношение для каждого из этих показателей. Студенты в сессию сдавали m экзаменов.

Для реализации этого алгоритма составим класс Student, содержащий закрытую секцию с массивами фамилии fam, имени name, отчества otch, оценок одного студента s, переменных cfam, cname, cotch, cs – размеров указанных массивов, а также открытую секцию с конструкторами и деструктором, с функциями-членами заказа памяти под массивы fam, name, otch, s, ввода данных студента и их вывода, исследования на наличие у студента только оценок 4 и 5, наличие хотя бы одной оценки 2. При этом размеры указанных массивов задаются в виде параметров задачи pfam, pname, potch, ps соответственно.

Программа реализации приведённого алгоритма для группы из n студентов представлена в листинге ниже.

```
#include <iostream.h>
#include <string.h>

class Student {
public:
Student (int pfam=20,  int pname=15,  int potch=20,  int ps=4);
~Student();
void cinr();
void coutr();
int r45();
int r2();
private:
int cfam,  cname,  cotch,  cs;
char *fam;
char *name;
char *otch;
int  *s; //Массив оценок студента
};
// Конструктор с параметрами
Student ::Student (int pfam,  int pname,  int potch,  int ps)
{
cfam=pfam;
cname= pname;
 cotch=potch;
 cs=ps;
```

```
fam=new char [cfam];
name=new char [cname];
otch=new char [cotch];
s=new int [cs];
}
//Деструктор
Student ::~Student ()
{
delete[]fam;
delete[]name;
delete[]otch;
delete[]s;
}
void Student :: cinr() //Функция ввода данных студента
{ cout<<”Введите данные студента: ”;
cin>>fam;
cin>>name;
cin>>otch;
for (int k=0; k<cs;k++) cin>>s[k];
}
void Student :: coutr() //Функция вывода данных студента
{ cout<<” Данные студента: ”;
cout<<’\n’<<fam;
cout<<’\n’<< name;
cout<<’\n’<< otch<<’\n’;
for (int k=0; k<cs;k++) cout<<s[k]<< ’\n’;
}
//Получение информации об учёбе на качество
int Student :: r45()
{ for (int k=0; k< cs;k++)
 if(s[k]<4) return  0;
return 1;
}
//Получение информации о наличии хотя бы одной оценки «2»
int Student :: r2()
{ for (int k=0; k<cs;k++)
 if(s[k]==2)  return  1;
return 0;
}
void main()
{ int n; //Количество студентов в группе
int pfam,  pname,  potch,  ps;
cin>>n>> pfam>> pname>>potch>>ps;
Student sr(pfam,  pname,  potch,  ps);
int rx45=0;
int rx2=0;
for (int k=1; k<=n;k++)
{
sr.cinr();
sr.coutr();
rx45+=sr.r45();
rx2+=sr.r2();
```

```
}
cout<<”На 4 и 5 сдали ”<<rx45<<” студентов или”<<rx45*100.0/n<<”%”;
cout<<”Хотя бы одну 2 имеют ”<<rx2<<” студентов или”<<rx2*100.0/n<<”%”;
}
```

Пример 3

Создать класс, который осуществляет работу со строками: инициализация, реализация функций ввода-вывода и сортировка.

```
#include <iostream>
#include <string.h>

using namespace std;

class string_
{
private:
	// Строка
	char* S;

	// Длина строки
	int len;
public:
	// Конструктор по умолчанию
	// без параметров
	string_();

	// Перегруженный конструктор
	// с параметром
	string_(char* s);

	// Конструктор копирования
	string_(const string_& s);

	// Деструктор
	~string_(){
		delete [] S;
	}

	// Метод сортировки
	void Sort(string_ s[], int n);

	// Константный метод
	// возвращающий содержимое
	// строки
	const char*GetStr()const
	{
		return S;
	}
	// метод позволяющий изменить содержимое
	// с помощью пользователя
```

```
	void SetStr()
	{
		// если строка не пустая - очистить
		if(S!=NULL)
			delete[]S;

		// создаем массив
		// и запрашиваем у пользователя данные
		char a[256];
		cin.getline(a,256);

		// просчитываем размер
		len=strlen(a)+1;

		// выделяем память
		S = new char[len];

		// переписываем в объект
		// введенную строку
		strcpy(S,a);
	}

	// метод позволяющий изменить содержимое
	// с помощью параметра
	void SetStr2(char*str)
	{
		// если строка не пустая - очистить
		if(S!=NULL)
			delete[]S;

		// просчитываем размер
		len=strlen(str)+1;

		// выделяем память
		S = new char[len];

		// переписываем в объект
		// строку из параметра
		strcpy(S, str);
	}
};

string_::string_()
{
	// Начальная инициализация
	S = NULL;
	len = 0;
}

string_::string_(char* s)
```

```
{
	len = strlen(s);
	S = new char[len + 1];
	// Инициализация строкой,
	// переданной пользователем
	strcpy(S, s);
}

string_::string_(const string_& s)
{
	len = s.len;
	// Безопасное копирование
	S = new char[len + 1];
	strcpy(S, s.S);
}

void string_::Sort(string_ s[], int n)
{
	//  Сортировка строк
	//  Методом пузырька

	string_ temp;

	for(int i=0;i<n-1;i++)
	{
		for(int j=n-1;j>i;j--)
		{
			// сравнение двух строк
			if(strcmp(s[j].S,s[j-1].S)<0)
			{
				// запись стороки s[j] в temp
				temp.SetStr2(s[j].S);

				// запись стороки s[j-1] в s[j]
				s[j].SetStr2(s[j-1].S);

				// запись стороки temp в s[j-1]
				s[j-1].SetStr2(temp.S);
			}
		}
	}
}

void main()
{
	int n,i;

	// Вводим количество строк
	cout << "Input the number of string s:\t";
	cin >> n;
```

```
    if(n < 0)
    {
            cout << "Error number:\t" << n << endl;
            return;
    }

    // Забираем из потока символ Enter ("\n")
    char c[2];
    cin.getline(c, 2);

    // Создаем массив из n строк
    string_ *s = new string_[n];

    // Ввод строк с клавиатуры
    for(i = 0; i < n; i++)
            s[i].SetStr();

    // Сортировка строк
    // Вызов через указатель,
    // так как функция работает
    // для группы объектов,
    // а не для одного конкретного
    s->Sort(s, n);

    // Вывод отсортированных строк
    for(i = 0; i < n; i++)
            cout<<"\n"<<s[i].GetStr()<<"\n";

    // Удаление массива строк
    delete [] s;
}
```

Задание 2

1. Объявить класс по приведенному ниже заданию в соответствии с номером варианта и определить для него конструктор по умолчанию, конструктор инициализации и конструктор преобразования.

2. Определить функции-члены класса для ввода и вывода членов-данных внутри объявления класса.

3. Составить программу, которая определяет четыре объекта класса и выводит их на экран. Первый объект должен инициализироваться по умолчанию, второй использовать конструктор инициализации, третий - преобразование типа своего параметра к типу своего класса, а четвертый - функцию ввода данных.

4. Выполнить программу и проверить соответствие заданных и полученных данных.

5. Определить функции ввода и вывода вне объявления класса и повторить выполнение программы.

6. Объявить функции ввода и вывода как "друзей" класса с параметром - объектом класса, передаваемым по значению. Изменить определение этих функций и повторить выполнение программы.

7. Объявить функции ввода и вывода как "друзей" класса с параметром - объектом класса, передаваемым по ссылке. Изменить определение этих функций и повторить выполнение

программы.

Варианты.

1. Класс "Комплексное число" с данными действительная и мнимая части.
2. Класс "Вектор на плоскости" с данными проекция вектора на оси X и Y.
3. Класс "Дробь" с данными числитель и знаменатель.
4. Класс "Окружность" с данными центр и радиус окружности.
5. Класс "Номер телефона" с данными код города, код станции и номер линии.
6. Класс "Квадрат" с данными центр квадрата и его сторона.
7. Класс "Прямоугольник" с данными верхняя левая и правая нижняя точки.
8. Класс "Вектор на плоскости" с данными модуль вектора и угол между вектором и положительным направлением оси X.
9. Класс "Дата" с данными число, номер месяца и две последние цифры года.
10. Класс "Время" с данными часы, минуты и секунды.
11. Класс "Карта" с данными масть в виде первой буквы масти и значения в виде одного из символов: 2, ..10, В, Д, К, Т.
12. Класс «Проект» с данными номер проекта, сумма, дата исполнения
13. Класс «Здание» с данными количество этажей, подъездов и квартир.

Контрольные вопросы.

1. Для чего используется указатель this?
2. Что такое конструктор?
3. Что такое деструктор?
4. Каково назначение конструктора копирования?
5. Какие классы и функции называются дружественными?
6. В чем отличие дружественных функций от функций-членов класса?
7. Приведите примеры дружественной функции, дружественного класса.

Содержание отчёта:

1. Постановка задачи.
2. Текст программы.
3. Комментарии к ведённым обозначениям и описание используемых функций.
4. Результаты работы программы.

Практическая работа №3. Дружественные функции и перегрузка операций. Преобразование данных

План

1. Преобразования, определяемые классом
2. Перегрузка и выбор функций
3. Дружественные функции
4. Перегружаемые операции
5. Перегрузка унарного оператора
6. Перегрузка бинарного оператора
7. Перегрузка оператора присваивания и индексирования
8. Перегрузка операций new, delete, ->

Полиморфизм- это средство для придания различных значений одному и тому же сообщению в зависимости от типа обрабатываемых данных. *Преобразование* - это явное или неявное изменение значения в зависимости от типа. Преобразование обеспечивает форму полиморфизма. Перегрузка функций дает одному и тому же имени функции различные значения. Одно и то же имя имеет различные интерпретации, которые зависят от выбора функции. Выбранная функция удовлетворяет алгоритму соответствия сигнатур для C++. Такая форма полиморфизма называется *специальный полиморфизм.*

Операторы перегружаются и выбираются на основании алгоритма соответствия сигнатур. Перегрузка операторов придает им новые значения. Например, выражение **a+b** имеет различные значения, в зависимости от типов переменных **a** и **b.** Перегрузка оператор + для типов, определяемых пользователем, позволяет использовать их в дополнительных выражениях в большинстве случаев так же, как и встроенные типы. При определении функции преобразования допустимы также выражения смешанного типа.

Один из принципов ООП состоит в том, что определяемые пользователем типы должны иметь те же привилегии, что и встроенные типы. Типы, встроенные в базовый язык, могут смешиваться в выражениях, поскольку это удобно, но, с другой стороны, при этом обременительно определять последовательность необходимых преобразований.

Преобразование, определяемые классом

Операция преобразования типов предоставляет механизм явного преобразования объектов данного класса в другой тип.

Явное преобразование типов выражения применяется тогда, когда неявное преобразование нежелательно или когда без него выражение недопустимо. Одна из задач C++ - интеграция Абстрактного Типа Данных (АТД) и встроенных типов. Чтобы достигнуть этого существует механизм функции-члена, обеспечивающей явное преобразование. Функциональная запись:

Имя типа (выражение)

эквивалентна приведению. Тип должен быть выражаем как идентификатор. Таким образом два выражения
x=(float)i;
x=float(i);
эквивалентны. Выражение
p=(int *)x; // допустимое приведение

не может непосредственно функционально выражаться как
p=int *(x);
Однако, для этого может использоваться typedef
typedef int * int_ptr;
p=int_ptr(x);
Конструктор с одним аргументом фактически представляет собой преобразование из типа аргумента к типу конструируемого класса.

string(const char *p) { len=strlen(p); s=new char[len+1]; strcpy(s,p); }

Представленное выражение - автоматическое преобразование типов от char * к string. Оно доступно как явно, так и неявно. Явно оно используется или как операция преобразования, или в приведении, или в функциональной форме. Таким образом, возможны два рабочих варианта кода:
string s;
char * logo=" Hello";
s=string(logo); // выполняет преобразование, затем присвоение
и
s=logo; // неявный вызов преобразования
Данный код - преобразование из уже определенного типа к типу, определяемому пользователем. Однако, пользователь не может добавлять конструкторы встроенных типов, таких как int. Может возникнуть необходимость в преобразовании из string в char *, что может быть выполнено с помощью определения специальной функции преобразования внутри класса string следующим образом:

operator char * () { return s; }

Общая форма записи такой функции-члена

operator тип() {...}

Такая функция преобразования должна быть нестатической функцией-членом без возвращаемого типа и с пустым списком аргументов. Преобразование происходит неявно в выражениях присвоения, при передаче параметров функциям, и в значениях, возвращаемых функциями.
Преобразующая функция-член в форме A::operator B() и конструктор в форме B::B(const A&) обеспечивает преобразование из типов объекта A в тип объекта B.
Неявное преобразование типов происходит тогда, когда имеется конструктор преобразования данного типа в объект класса:
Int int1,int2; int2=int1+6;
Если класс содержит конструктор с одним параметром вида:
Int::Int(int a) { b=a; },
то код будет работать нормально, если же нет, то будет ошибка.
Int int1,int2; int2=6+int1;
Запись такого вида недопустима. Она означает:. передать объекту 6 сообщение +с параметром int1. Возможно ли, обеспечить такую запись. Да, если написать перегруженный оператор + как дружественные функции.
Рассмотрим преобразование объектов одного класса в объекты другого класса. Если мы хотим преобразовать объект класса OldType в объект класса NewType, то класс NewType должен иметь конструктор с параметром типа OldType или OldType& :
class OldType{
};

```
class NewType {
public:
NewType(void) {}
NewType(OldType &old) { }
};
void main() {
OldType oldtype;
NewType newtype=oldtype;
}
```
либо с использованием операции преобразования типов
```
operator NewType(void);
class NewType {
public:
NewType(void) {}
};
class OldType {
publci:
operator NewType() { return NewType(); }
};
void main() {
OldType oldtype;
NewType newtype=oldtype;
}
```
Использование одновременно конструктора и перегрузку операции приведение невозможно. Так как, операции преобразования не имеют ни явных аргументов, ни возвращаемого значения, то они не могут быть перегружены. Может быть несколько операторов преобразования для различных типов. Операции преобразования наследуются и могут быть виртуальными.

Перегрузка и выбор функции

Перегруженные функции - важная особенность языка C++. Перегружаемая функция выбирается в соответствии со списком параметров при обращении к функции и при объявлении функции. При вызове перегруженных функций компилятор должен иметь алгоритм для выбора соответствующей функции. Этот алгоритм зависит от того, какие преобразования типов доступны. Наилучшее соответствие должно быть уникально. Оно должно быть лучше всех, по крайне мере, для одного параметра, а для всех остальных параметров так же хорошо, как любое другое соответствие.

Алгоритм соответствия для каждого параметра следующий:

1. Использовать точное соответствие, если оно найдено
2. Проверить поддержку стандартных типов
3. Проверить стандартные преобразования типов
4. Проверить преобразования, определяемые пользователем
5. Использовать соответствие для аргументов, если оно найдено

Рассмотрим пример.

```
class complex

{
        double real,imag;
public:
        complex(double r) { real=r; imag=0; }
        void assign(double r,double i) { real=r; imag=i; }
        void print() { cout << real << “+”<<imag<<”i”; }
```

```
                // обеспечивает преобразование complex в double
                operator double() { return sqrt(real*real+imag*imag); }
};
inline int greater(int i,int j) { return (i > j) ? i:j; }
inline double greater(double i,double j) { return (i > j) ? i:j; }
inline complex greater(complex i,complex j) { return (i > j) ? i:j; }
void main() {
                int i=5,j=10;
                float x=7;
                double y=14;
                complex w(0),z(0);
                w.assign(x,y); // требует преобразования
                float x в double
                z.assign(i,j); // оба параметра преобразуются в
                double
                // выбирает первое определение
                greater по правилу соответствия
                cout << greater(i,j) << endl;
                // Выбирает второе определение
                greater, так как использовано стандартное расширяющее преобразование из float
                в double
                cout << greater(x,y) << endl;
                // Второе определение
                greater выбирается из-за точного соответствия //правилу.
                cout << greater(y,double(z)) << endl;
                // точно соответствует третьему определению
                cout << greater(w,z) << endl;
}
```

Функция-член преобразования **operator double** требуется для оценки условия **w<z. Complex** переменные **w z** преобразуются в **double**. Для возвращаемого типа не необходимости в преобразовании.
Для избежания неоднозначности необходимо явное преобразование **double(z)**. Если вместо этого будет использоваться вызов функции:
greater(y,z); // error
то он будет иметь два доступных преобразования, достигших соответствия. Преобразования параметра **y** из **double** в **complex**, определяемое пользователем, соответствует третьему определению. Преобразование **z** из **complex** в **double**, также определяемое пользователем, соответствует второму определению. Но второе определение функции имеет лучшее соответствие для первого параметра, между тем как третья функция имеет лучшее соответствие для второго параметра. Это нарушает требование о том, чтобы “Максимальное соответствие должно быть уникально. Оно должно быть лучше всего для, по крайне мере, одного параметра и одинаково хорошо для всех остальных”.

Дружественные функции

Класс может предоставлять особые привилегии определенным внешним функциям или функциям-членам другого класса. Эти функции получили названия дружественных. Если функция или класс объявлены как дружественные данному классу, то такие функции или функции-члены такого класса могут осуществлять непосредственный доступ ко всем полям класса, для которого они дружественны. Дружественные функции и классы могут осуществлять прямой доступ к закрытым полям класса без использования функций-членов этого класса.

Ключевое слово **friend** - спецификатор функции, который дает функции- не члену класса доступ к скрытым членам класса. Он используется для того, чтобы выйти за строгие рамки типизации и сокрытия данных в C++.

Одна из причин их использования состоит втом, что некоторые функции нуждаются в привилегированном доступе более, чем к одному классу. Вторая причина - **friend**-функция передает все параметры через список параметров, и значение каждого из них подчинено преобразованию, совместимому с назначением. Такие преобразования применяются к явно переданным аргументам-классам и поэтому особенно полезны в случаях перегрузки оператора.

Объявление **friend** функции должно появляться внутри объявления класса, которому она дружественна. Имени функции предшествует ключевое слово **friend**, и ее объявление может находится как в **public** так и в **private** части класса, что не повлияет на значение. Функция-член одного класса может быть **friend**-функцией другого класса. Это происходит тогда, когда функция-член объявлена в **friend** классе с использованием оператора разрешения контекста для определения имени функции дружественного класса. Если все функции-члены одного класса являются **friend**-функциями другого класса, то это можно определить записью:

```
freind class имя класса
class t1 {
friend void a(); // friend-функция
int b(); // функция-член
};
class t2 {
friend int t1::b(); // функция-член класса t1 имеет доступ ко всем скрытым полям класса t2
};
class t3 {
friend class t1; // все функции-члены класса t1 имеют доступ ко всем полям класса t3
};
```

Рассмотрим класс **matrix** и класс **vector**. Функция умножения вектора на матрицу должна иметь доступ к **private-членам** обоих классов. Эта функция будет **friend** для обоих классов.

```
class matix;
class vect {
int *p;
int size;
friend vect mpy(const vect &,const matix &);
public:
};
class matrix {
int **base;
int row,column;
friend vect mpy(constr vect&,const matirx &);
};
vect mpy(const vect &v,const matrix &m) {
if(v.size!=m.row) { exit(1); }
vect ans(column);
//.....
return ans;
}
```

Второстепенное значение требует предварительного описания класса **matrix**. Оно необходимо потому, что функция **mpy** должна появляться в обоих классах, и использует каждый класс как тип аргумента.

Friend-функции можно рассматривать как часть общего интерфейса класса. Существует ряд ситуаций, в которых они могут быть альтернативой функциям-членам. Использование **freind**-функций спорно, потому что они нарушают инкапсуляцию, окружающую **private** члены классов. Парадигма ООП утверждает, что объекты (в С++ они - переменные класса) доступны через их **public** члены. Только функции-члены должны иметь доступ к скрытой реализации АТД. Это ясный и строгий принцип проектирования. **Friend**-функция находится на самой его границе, поскольку имеет доступ к **private** членам, сама не являясь функцией-членом. С ее помощью можно организовать быстрый код для доступа к подробностям реализации класса.

Перегружаемые операции

Для перегрузки встроенных операторов С++ можно использовать ключевое слово **operator.** Ему может быть присвоен ряд значений, зависящих от параметров. Таким образом оператору типа + можно присвоить дополнительные значения. Это облегчает написание программ и делает их более читабельными.

vect oprator *(const vect &,const matix &)

где * является двухместным оператором умножения.

matix t; vect s,r; s=t*r; // s=mpy(r,t);
Перегруженный оператор можно вызвать и использованием функциональной формы записи:
s=operator*(t,r);
Хотя операторам и могут добавляться новые значения, но их приоритет остается прежним. Например, операция умножения имеет более высокий приоритет, чем сложение. Перегружены могут все операторы. Нельзя использовать аргументы по умолчанию.

Доступны все арифметические, логические операторы, операторы сравнения, равенство, присвоение, операторы поразрядных операций, префиксные и постфиксные формы операторов приращения и декремента. могут быть перегружены операторы индексации “[]” и обращения к функции “()”. Также могут быть перегружены оператор указателя класса “->“ и оператор указателя на член “->*”. Возможна перегрузка **new, delete.**

Перегрузка унарного оператора

Одноместные операторы типа “!”, “++”, “~” “[]”.

```
class clock
{
                unsigned long sec;
public:
                clock(unsigned long s):sec(s) { }
                void tick() { sec++; }
                clock operator++() { tick(); return *this; }
};
void main() { clock t(100); ++t; }
```

Этот класс перегружает префиксный оператор приращения ++. Перегруженный оператор представляет собой функцию-член. Перегруженный
operator++() также обновляет неявную переменную **clock** и возвращает обновленное значение. Префиксную операцию ++ можно перегрузить, используя
friend-функцию.

frien clock operator++(clock &s1) { s1.tick(); return s1; }
Так как переменная
clock должна увеличиваться мы передаем ее по ссылке. Решение о выборе между представлением friend и функцией-членом обычно зависит от того, насколько необходимы и доступны операторы неявного преобразования. Явная передача аргумента, как в friend-функции, позволяет автоматическое его приведение.

Операции инкремента и декримента могут быть перегружены как для префиксной, так и для постфиксной формы записи.
Префиксная форма объявляется как операция-член класса, которая не имеет аргументов, либо как дружественная операция, которая имеет один аргумент, представляющий собой ссылку на объект данного класса.
Постфиксная форма объявляется как операция-член класса, которая имеет один аргумент типа int, либо как дружественная операция, которая имеет два аргумента: ссылку на объект данного класса и аргумент типа int. Аргумент типа int на самом деле не используется. Фактически он имеет нулевое значение.

```
class X {
public:
X & operator ++(); //Префиксная форма
X & operator ++(int); // Постфиксная форма
};
class Y {
public:
friend Y & operator ++(Y &); //Префиксная форма
friend Y & operator ++(Y &,int); // Постфиксная форма
};
```
@x интерпретируется как x.opertor@(), если operator @ - функция-член и operator@(x), если operator @ - дружественная функция.

Перегрузка бинарного оператора

Если бинарный оператор перегружается с помощью функции-члена, то в качестве своего первого аргумента он получает неявно переданную переменную класса, а второго - единственный из списка аргументов. При объявлении **friend**-функции или обычной функции в списке параметров определяют оба аргумента.

```
class clock {
friend clock operator+(const clock &c1,const clock &c2) ;
};
clock operator+(const clock &c1,const clock &c2) {
clock temp(c1.sec+c2.sec); return temp;
}
```

Оба параметра явно определяются и являются кандидатами для преобразования назначением. Используя это определение имеем:

```
int i=5; clock c(100);
c+i; // допустимо: i - преобразуется в clock
i+c; // допустимо: i - преобразуется в clock
```
В противоположность этому, перегрузим двухместный - функцией-членом.

```
class clock {
clock operator -(const clock &c) { clock temp(sec-c.sec); return temp; }
};
```

Существует первый неявный аргумент. В него попадает некоторый используемый параметр. Это может вызвать асимметричное поведение двухместных операторов.

```
int i=5; clock c(100);
c-i; // допустимо i преобразуется в clock; c.operator-(i)
i-c; // недопустимо i не относится к типу clock; i.operator-(c)
```

Определим операцию умножения: long и clock

```
clock operator *(long m, const clock &c) {
clock temp(m*c.sec); return temp;
```

Такая реализация вынуждает операцию умножения иметь фиксированный порядок выполнения, зависящий от типа. Для избежания этого обычно пишется второй перегруженный функциональный оператор.

```
clock operator *(const clock &c,long m) {
                clock temp(m*c.sec); return temp;
}
```

y@x интерпретируется как y.opertor@(x), если operator @ - функция-член и operator@(y,x), если operator @ - дружественная функция.

Операция вызова функции

Операция вызова функции должна быть объявлена как нестатическая функция-член класса. Она позволяет пользователю определять число операндов.

```
class X {
int a,b,c;
public:
X(int a1, int b1, int c1): a(a1),b(b1),c(c1) {}
void operator() { int i,int j, int k);
};
X ex(8,20,17); // Вызывается конструктор
ex(10,15,57); // Вызывается операция ()
```

Перегруженные операции - функции-члены против дружественных функций

Если перегруженная операция реализована как функция-член, то ей либо вообще не передаются явно параметры, либо передается один параметр. Дружественной функции - перегруженной операции - передается один, либо два параметра.

Если бинарная операция перегружена как функция-член, то ее первым операндом является объект, который принимает сообщение. Следовательно, явно передается только один параметр.

Если унарная перегруженная операция реализована как функция-член, то ее операндом является принимающий сообщение объект. Таким образом, эта функция-член не имеет явных параметров.

Перегруженная операция не может иметь более двух параметров.

Фунция-член

```
Class X {
X operator-() const; // Унарный минус
X operator&() const; // Вычисление адреса
X operator^() const; // Ошибка: операция ^ - бинарная
        };
```

```
Class X {
        X operator-(const X&) const; // бинрный
```

Дружественная функция

```
class Y {
        friend Y operator-(const Y&);
friend Y operator&(const Y&);
        freind Y operator^(const Y&);
};
```

```
class Y {
        friend Y operator-(const Y&,const
```

```
        минус                                                   Y&);
X operator&( const X&) const; // побитовое И          friend Y operator&(const Y&, const Y&);
        };                                                      };
```

Перегружаемые оператор присваивания и индексированные операторы

```
class vect {
private:
                int *p;
                int size;
public:
                vect(void); // создает массив из 10 элементов
                vect(int mysize); // создает массив из mysize элеметов
                vect(int *a,int n); // инициализируется массивом
                vect(const vect &v); // инициализируется vect
                ~vect() { delete p; }
                int & operator[](int i);
                vect& operator=(const vect &v); // перегруженное присваивание
                vect operator+(const vect &v); // перегруженное сложение
};
vect::vect(int n) { // преобразует обычный массив
                p=new int[n]; size=n;
                for(int index=0; index < size; index++) p[i]=index;
}
vect::vect(int *a,int n) { // преобразует обычный массив
                p=new int[n];
                size=n;
                for(int index=0; index < size; index++) p[i]=a[i];
}
vect::vect(const vect &v) { // конструктор копии
                p=new int[v.size]; size=v.size;
                for(int index=0; index < size; index++) p[i]=v.p[i];
}
```

Перегруженный оператор индексации берет целый аргумент и проверяет, находится ли значение в пределах диапазона. Если да, он использует это значение для того, чтобы возвратить адрес индексированного элемента.

```
int &vect::operator[](int i) {
if(i < 0 || i > =size) { cerr << “”;}
return p[i];
}
```

Перегруженный оператор индексации имеет возвращаемый тип и один параметр. Он должен быть нестатической функцией-членом.

Вопросы:

1. Что если перегруженная операция [] будет иметь тип возвращаемого значения int, а не int&?

Ответы:

Произойдет ошибка при компиляции. Использование ссылки в качестве возвращаемого значения позволяет использовать операцию индексирования как с правой, так и слевой стороны от операции присваивания.

Оператор присваивания

В C++ версии 2.0 имеется операция присваивания по умолчанию, которая выполняет рекурсивное по членное копирование одного объекта в другой. В более ранних версиях использовался механизм побитового копирования.

```
vect first(200),second(200);
second=first; cout << second[10]; // =9
first[10]=-50; cout << second[10]; //=-50
```

Когда назначение не перегруженно, оно определяется по умолчанию с семантикой, представляющей собой присвоения значения. Это называется “семантикой поверхностного копирования” и не всегда допустимо. Перегрузка назначения желательна, а иногда и необходима, особенно, если класс содержит указатели динамически распределяемой памяти:

```
vect &vect::operator=(const vect &) {
int s=(size<v.size) ? size : v.size;
for(int i=0; i<s; i++) p[i]=v.p[i]; return *this; }
```

Если необходимо запретить операцию присваивания для объектов данного класса, то ее нужно просто объявить в закрытой части объявления.

```
Class IntStack {
int *v, size,top;
public:
IntStack &operator=(const IntStack&);
};
IntStack & IntStack::operator=(IntStack &from) {
if(this != &from) { // Проверка на from=from
delete [] v;
v= new int [size=from.size]; for(int i=0; i<size; i++) v[i]=from.v[i];
top=from.top;
}
return *this; // Позволяет выполнять множественное присваивание
}
IntStack a(100),b,c;
b=c=a;
```

Присваивание против инициализации

Для того, чтобы операция = обозначала присваивание, в левой части выражения, содержащего эту операцию, должен находиться объект (т.е. для него должна быть выделена память) операция присваивания должна выглядеть следующим образом:
C& C::operator=(C&);
Так как операция присваивания возвращает ссылку на объект, она может использоваться в цепочке таких операций.
Инициализация происходит либо при по членном копировании, либо при использовании конструктора следующего вида:
C::
C(&C);
Желательно включение конструктора инициализации.
Инициализация происходит в следующих случаях:
Объявление переменной vect second=first;

Возврат автоматической переменной из функции. Возвращаемое значение - это новый объект, который инициализируется при помощи конструктора.
Передача значения функции в качестве параметра. Параметр действует как локальная переменная и инициализируется при помощи конструктора.
Рассмотрим примеры инициализации:

```
class demo {
        int i;
public:
        demo(void) {i=0; }
        demo(int i1):i(i1) {}
        demo(demo &a) { i=a.i; }
        demo &operator=(demo &a) { i=a.i; return *this; }
};
demo copydemo(demo a) {
        demo temp; // конструктор без параметров
        temp=a; // оператор присваивания
        return temp; // вызов конструктора инициализации, временная копия
}
void main(void) {
        demo *ptr=new demo(5); // вызывается конструктор с одним параметром
        demo demo2=demo1; // вызов инициализирующего конструктора
        demo demo3; // вызов конструктора без параметров
        demo3=demo2=demo1; // перегруженная операция присваивания вызывается 2
        раза
        demo4=copydemo(demo1); // вызывается конструктор инициализации для
        передачи аргумента, по выходе оператор присваивания.
}
```

Конструктор копирования должен быть определен, если класс содержит поля , являющиеся указателями на динамически распределяемую память. При отсутствии конструктора копирования, конструктор копирования по умолчанию сделает так, что указатель b объекта v и указатель b объекта a будут иметь одинаковые значения. По той же причине необходимо перегружать операцию присваивания, если конструктор копирования перегружен пользователем. Инициализация объекта из другого объекта того же класса может быть запрещена, если конструктор копирования просто объявить в закрытой части класса.

```
Int a(100);
Int b(a); // Объявить b, которое равно a
Int c=a; // То же в альтернативной форме
```

Перегруженные операции new delete, ->

Операции new и delete предоставляют классу механизм управления собственной памятью. Тип возвращаемого значения для new - void *, для delete - void. Неявно они являются статическими функциями-членами и, следовательно, не могут быть не константными не виртуальными.

```
class X {
public:
void *operator new(size_t sz) { return malloc(sz); }
void operator delete(void *p) { free(p); }
};
```

Если операции new и delete определены для класса, не могут быть вызваны при выделении и освобождении памяти для массивов объектов данного класса. Операция new может быть перегружена, чтобы принимать дополнительные аргументы. Передаваемые аргументы, заключенные в скобки, помещаются при вызове после ключевого слова new.

```
class T {
public:
// Добавить дополнительный аргумент buf , который будет использоваться для
// выделения памяти для T по указанному адресу
void * operator new(size_t size,void *buf) { /* Разместить T по адресу buf */}
};
char buffer[1000];
T *p=new (buffer) T;
```

Для объектов класса X вместо операций new, delete будет использоваться X::operator new, X::operator delete.

Перегруженная операция new с дополнительными аргументами скрывает глобальную операцию ::operator new. Теперь при обращении к new с обычным синтаксисом произойдет ошибка.

```
T *p=new T; // ошибка: отсутствует аргумент
```

Чтобы снова использовать обычный синтаксис оператора new, для класса должна быть предусмотрена функция-член new, имеющая только один аргумент - size_t size. Она явно вызывает глобальную операцию ::operatot new.

```
class T {
public:
void * operator new(size_t size,void *buf) { /* Разместить T по адресу buf */}
void * operator new(size_t size) { return ::operator new(size); }
};
```

Перегруженная операция -> обычно используется в классах, которые содержат в качестве поля указатель на другой класс.

```
struct A { int *intptr; };
class B { A *a;
public:
B(void) { a=new A; } A *operator->(void) { return a; }
};
B b; *(b->inptr)=15; // = b.a->inptr;
```

Примеры:

1. Перегрузка оператора + относительно класса **coord**.

```
// Перегрузка оператора + относительно класса coord
#include <iostream>
using namespace std;

class coord {
   int x,y; // значения координат
public:
   coord() { x = 0; y=0; }
   coord(int i, int j) { x = i; y = j; }
   void get_xy(int &i, int &j) { i = x; j = y; }
   coord operator+(coord ob2);
};

// Перегрузка оператора + относительно класса coord
coord coord::operator+(coord ob2)
{
   coord temp;
   temp.x = x +ob2.x;
   temp.y = y +ob2.y;

   return temp;
}

int main()
{
   coord o1(10, 10), o2(5, 3), o3;
   int x, y;

   o3 = o1 + o2; // сложение двух объектов - вызов функции operator+()

   o3.get_xy(x, y);
   cout << "(o1 + o2) X: " << x << ", Y: " << y << "\n";

   return 0;
}
```

2. Перегрузка операторов == и &&

```
// Перегрузка операторов == и && относительно класса coord
#include <iostream>
using namespace std;
class coord {
   int x, y; // значения координат
public:
   coord() { x = 0; y=0; }
   coord(int i, int j) { x = i; y = j; }
   void get_xy(int &i, int &j) { i = x; j = y; }
   int operator==(coord ob2);
   int operator&&(coord ob2);
};
// Перегрузка оператора == для класса coord
int coord::operator==(coord ob2)
{
   return x==ob2.x && y==ob2.y;
}
// Перегрузка оператора && для класса coord
int coord::operator&&(coord ob2)
{
   return (x && ob2.x) && (y && ob2.y);
}
int main()
{
   coord o1(10, 10), o2(5, 3), o3(10, 10), o4(0, 0);
   if(o1==o2) cout << "o1 равно o2\n";
   else cout << "o1 не равно o2\n";
   if(o1==o3) cout << "o1 равно o3\n";
   else cout << "o1 не равно o3\n";
   if(o1&&o2) cout << "o1 && o2 равно истина\n";
   else cout << "o1 && o2 равно ложь\n";
   if(o1&&o4) cout << "o1 && o4 равно истина\n";
   else cout << "o1 && o4 равно ложь\n";
   return 0;
}
```

3. Относительно класса перегружается оператор инкримента(++)

```
// Перегрузка оператора ++ относительно класса coord
#include <iostream>
using namespace std;

class coord {
   int x, y; // значения координат
public:
   coord() { x = 0; y=0; }
   coord(int i, int j) { x = i; y = j; }
   void get_xy(int &i, int &j) { i = x; j = y; }
   coord operator++();
};

// Перегрузка оператора ++ для класса coord
coord coord::operator++()
{
   x++;
   y++;

   return *this;
}

int main()
{
   coord o1(10, 10);
   int x, y;

   ++o1; // инкремент объекта
   o1.get_xy(x, y);
   cout << "(++o1) X: " << x << ", Y: " << y << "\n";

   return 0;
}
```

4. Перегрузка функции operator+() перегружается для класса **coord**

```
/ Перегрузка оператора + относительно класса coord
// с использованием дружественной функции
#include <iostream>
using namespace std;
class coord {
   int x, y; // значения координат
public:
   coord() { x = 0; y=0; }
   coord(int i, int j) { x = i; y = j; }
   void get_xy(int &i, int &j) { i = x; j = y; }
   friend coord operator+(coord ob1, coord ob2);
};
// Перегрузка оператора + с использованием дружественной функции
coord operator+(coord ob1, coord ob2)
{
   coord temp;

   temp.x = ob1.x + ob2.x;
   temp.y = ob1.y + ob2.y;

   return temp;
}
int main()
{
   coord o1(10, 10), o2(5, 3), o3;
   int x, y;
   o3 = o1 + o2; // сложение двух объектов
            // вызов функции operator+()
   o3.get_xy(x, y);
   cout << "(o1 + o2) X: " << x << ", Y: " << y << "\n";

   return 0;
}
```

5. Объявляется состоящий из пяти целых элементов массив **arraytype.** Каждый элемент массива инициализируется конструктором. Перегруженная функция operator[]() возвращает элемент, заданный ее параметром.

```
#include <iostream>
using namespace std;

const int SIZE = 5;

class arraytype {
   int a[SIZE];
public:
   arraytype() {
      int i;
      for (i=o; i<SIZE; i++) a[i] = i;
   }
   int operator[](int i) { return a[i]; }
};

int main()
{
   arraytype ob;
   int i;

   for(i=0; i<SIZE; i++)
      cout << ob[i] << " ";

   return 0;
}
```

Вариант 1. Перегрузка операторов

Задание 1 Унарная операция
Создать класс целых чисел. Определить оператор ++, как функцию-член и -- как дружественную функцию.
Задание 2. Бинарная операция
Создать класс целых чисел. Определить оператор +, как функцию-член и - как дружественную функцию.
Задание 3.
Создать класс вектор, содержащий ссылку на int, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым числом операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа int. Перегрузить операторы вывода и ввода в поток.
Задание 4
Создать класс матриц, содержащий ссылку на int, число строк и столбцов и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с целым и с вектором, определенном в задании 10. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток.

Вариант 2. Перегрузка операторов

Задание 1 Унарная операция
Создать класс целых чисел. Определить оператор --, как функцию-член и ++ как дружественную функцию.
Задание 2. Бинарная операция
Создать класс координат. Определить оператор +, как функцию-член и - как дружественную функцию. Сложить и вычесть координаты с друг другом и с числом. Присвоить координаты, сравнить координаты (==, !=).
Задание 3
Создать класс вектор, содержащий ссылку на long, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с числом типа long, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа long. Перегрузить операторы вывода и ввода в поток.
Задание 4
Создать класс матриц, содержащий ссылку на long, число строк и столбцов и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с long и с вектором, определенном в задании 10. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток.

Вариант 3. Перегрузка операторов

Задание 1. Унарная операция
Создать класс вещественных чисел. Определить оператор ++, как функцию-член и -- как дружественную функцию.
Задание 2. Бинарная операция
Создать класс целых чисел. Определить оператор -, как функцию-член и + как дружественную функцию.
Задание 3
Создать класс вектор, содержащий ссылку на float, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым числом операторы ++ и --. Определить операторы =, +, - , *, +=, -=, *= с вещественным числом, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа float. Перегрузить операторы вывода и ввода в поток.
Задание 4
Создать класс матриц, содержащий ссылку на float, число строк и столбцов и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором, определенном в задании 10. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток.

Вариант 4. Перегрузка операторов

Задание 1. Унарная операция
Создать объект типа очередь. Перегрузить оператор ++ как функцию член и -- как дружественную функцию. (Как постфиксными так префиксными). ++ добавляет элемент в очередь (пустой элемент, например int i=0), -- вытаскивает элемент из очереди. Оператор ! проверяет очередь на пустоту.
Задание 2. Бинарная операция
Создать объект типа стек. Перегрузить оператор + как функцию член и * как дружественную функцию. + складывает элемент в новый стек, * умножает верхушку стека на параметр. Стеки можно присваивать, проверять на равенство == или !=, вводить и выводить в поток, добавлять += элемент в стек.
Задание 3
Создать объект - двунаправленный список, в котором определены операции, + - добавляет в конец списка, += добавляет в этот же список в конец списка. - удаляет указанный элемент из списка (номер элемента через параметр), = - присвоение списков, сравнение списков ==, !=, >, <, >=, <=, [] получение элемента списка, ++ - устанавливает указатель на следующий элемент, -- устанавливает указатель на предыдущий элемент. () выдать подсписок от первого до второго элемента.
Задание 4
Создать класс матриц и вектор, содержащие ссылку на float, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /. Проверки. Операторы * И *= должны быть определены для умножения вектора и матрицы.
Вариант 5. Перегрузка операторов

Задание 1 Унарная операция
Создать объект типа стек. Перегрузить оператор ++ как функцию член и -- как дружественную функцию. (Как постфиксными так префиксными). ++ добавляет элемент новый в стек, -- удаляет верхушку стека. Оператор ! проверяет стек на пустоту.
Задание 2. Бинарная операция
Создать объект типа очередь. Перегрузить оператор + как функцию член и * как дружественную функцию. + добавляет элемент в очередь, * умножает элемент в очереди. Вытаскивает элемент из очереди --. Очереди можно присваивать, проверять на равенство == или !=, вводить и выводить в поток, добавлять += элемент в очередь.
Задание 3
Создать объект - однонаправленный список, в котором определены операции, + - добавляет в конец списка, += добавляет в этот же список в конец списка. - удаляет указанный элемент из списка (номер элемента через параметр), = - присвоение списков, сравнение списков ==, !=, >, <, >=, <=, [] получение элемента списка, ++ - устанавливает указатель на следующий элемент. () выдать подсписок от первого до второго элемента.
Задание 4
Создать класс матриц и вектор, содержащие ссылку на double, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /. Проверки. Операторы * И *= должны быть определены для умножения вектора и матрицы.

Вариант 6. Перегрузка операторов

Задание 1 Унарная операция
Создать объект - связный двунаправленный список, с перегруженными унарными операциями ++, --, как движение по списку. (Как постфиксными так префиксными).
Задание 2 Бинарная операция
Создать объект динамический стек. Перегрузить операции +, +=, -= (с извлечением элемента).
Задание 3
Создать объект динамическая очередь. Перегрузив операции +, --, +=, -=, =, !=, ==, >=, <=, >, <, ввода, вывода в поток, получить под-очередь ().
Задание 4
Создать класс матриц и вектор, содержащие ссылку на long, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /. Проверки. Операторы * И *= должны быть определены для умножения вектора и матрицы.

Вариант 7. Перегрузка операторов

Задание 1. Унарная операция
Создать объект стек, перегрузив ++ и --. (Как постфиксными так префиксными). ++ Добавляет элемент в стек. -- извлекает элемент из стека.
Задание 2. Бинарная операция
Создать объект очередь с перегруженными +, +=, добавление элемента в очередь и сложение очередей, -- для извлечения из очереди, - для вычитания очередей.
Задание 3
Определить класс список однонаправленный с перегруженными операциями ++ вперед по списку, -- удалить элемент, на котором стоит указатель, += с другим списком и с новым элементом, - унарный удаляет с конца списка, =, ==, !=, >, <, <=, >=. Ввод, вывод в поток. () - выдает подсписок.
Задание 4
Создать класс матриц и вектор, содержащие ссылку на int, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /. Проверки. Операторы * И *= должны быть определены для умножения вектора и матрицы.

Вариант 8. Перегрузка операторов

Задание 1. Унарная операция
Создать класс - координаты с унарным ++ и --, -. ++ и -- постфиксная и префиксная. - меняет знак у обеих координат. ++ как функция-член, -- как дружественная функция.
Задание 2. Бинарная операция
Создать класс целых чисел (long). Определить оператор -, как функцию-член и + как дружественную функцию. Оператор присвоения, и сравнений.
Задание 3
Создать класс вектор, содержащий ссылку на double, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа double. Перегрузить операторы вывода и ввода в поток.
Задание 4
Создать класс матриц, содержащий ссылку на double, число строк и столбцов и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с double и с вектором, определенном в задании 10. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток.

Вариант 9. Перегрузка операторов

Задание 1. Унарная операция
Создать класс вещественных чисел (double). Определить оператор ++, как функцию-член и -- как дружественную функцию.
Задание 2. Бинарная операция
Создать класс целых чисел (long). Определить оператор +, как функцию-член и - как дружественную функцию.
Задание 3
Создать класс вектор, содержащий ссылку на unsigned , размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым без знаковым числом, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа unsigned. Перегрузить операторы вывода и ввода в поток.
Задание 4
Создать класс матриц, содержащий ссылку на unsigned, число строк и столбцов и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с unsigned и с вектором, определенном в задании 10. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток.

Вариант 10. Перегрузка операторов

Задание 1. Унарная операция
Создать класс вещественных чисел (double). Определить оператор --, как функцию-член и ++ как дружественную функцию.
Задание 2. Бинарная операция
Создать класс вещественных чисел (double). Определить оператор -, как функцию-член и + как дружественную функцию.
Задание 3
Создать класс вектор, содержащий ссылку на unsigned long, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с типа unsigned long, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа unsigned long. Перегрузить операторы вывода и ввода в поток.
Задание 4
Создать класс матриц, содержащий ссылку на unsigned long, число строк и столбцов и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с unsigned long и с вектором, определенном в задании 10. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток.

Вариант 11. Перегрузка операторов

Задание 1. Унарная операция
Создать класс целых чисел (long). Определить оператор ++, как функцию-член и -- как дружественную функцию.
Задание 2. Бинарная операция
Создать класс вещественных чисел (double). Определить оператор +, как функцию-член и - как дружественную функцию.
Задание 3
Создать класс вектор, содержащий ссылку на unsigned char, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с числом типа unsigned char, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа unsigned char. Перегрузить операторы вывода и ввода в поток.
Задание 4
Создать класс матриц, содержащий ссылку на unsigned char, число строк и столбцов и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с unsigned char и с вектором, определенном в задании 10. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток.

Вариант 12. Перегрузка операторов

Задание 1. Унарная операция
Создать класс целых чисел (long). Определить оператор --, как функцию-член и ++ как дружественную функцию.
Задание 2. Бинарная операция
Создать класс вещественных чисел. Определить оператор -, как функцию-член и + как дружественную функцию.
Задание 3
Создать класс вектор, содержащий ссылку на long double, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым числом операторы ++ и --. Определить операторы =, +, - , *, +=, -=, *= с числом типа long double, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа long double. Перегрузить операторы вывода и ввода в поток.
Задание 4
Создать класс матриц, содержащий ссылку на long double, число строк и столбцов и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с long double и с вектором, определенном в задании 10. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток.

Вариант 13. Перегрузка операторов

Задание 1. Унарная операция
Создать объект - очередь с перегруженными операциями ++ как функциями-членами, -- как дружественными функциями. (Как постфиксными так префиксными).
Задание 2. Бинарная операция
Создать объект - однонаправленный список, в котором определены операции, + - добавляет в конец списка, += добавляет в этот же список в конец списка. - удаляет указанный элемент из списка (номер элемента через параметр), = - присвоение списков, сравнение списков ==, !=, >, <, >=, <=, [] получение элемента списка, ++ - устанавливает указатель на следующий элемент. () выдать подсписок от первого до второго элемента.
Задание 3
Создать объект типа стек. Перегрузить оператор ++, --, !, !=, ==, >, <, >=, <=, +,. Ввод, вывод в поток.
Задание 4
Создать класс матриц и вектор, содержащие ссылку на float, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /. Проверки. Операторы * И *= должны быть определены для умножения вектора и матрицы.

Вариант 14. Перегрузка операторов

Задание 1. Унарная операция
Создать объект - однонаправленный список, в котором определены операции, ++ - добавляет в конец списка, -- удаляет элемент из списка. (Как постфиксными так префиксными).
Задание 2. Бинарная операция
Создать объект стек, с перегруженными операциями +, *, =, +=, и для вытаскивания из стека --. () - выдает под-стек.
Задание 3
Определить класс - комплексные числа, перегрузив различные операторы, +, -, ++, --, +=, -=, *, ./, *=, /=, !, !=, ==, >,<, >=, <=. Ввода, вывода в поток. Сложение и вычитание должно производиться как с элементами данного класса так и со встроенными float.
Задание 4
Создать класс матриц и вектор, содержащие ссылку на double, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /. Проверки. Операторы * И *= должны быть определены для умножения вектора и матрицы.

Практическая работа № 4. Использование перегрузки.

Теоретические сведения

Перегрузка функций-элементов

Функции-элементы класса могут быть перегружены. Две или несколько функций-элементов могут иметь одно и тоже имя, при условии, что списки их аргументов не совпадают.
Операторы-функции. Используются для введения операций над
объектами, связываемых с символами:
+ , - , * , / , % , ^ , & , | , ~ , ! , = , < , > ,+= , [] , -> , ( ) , new, delete.

Оператор-функция является членом класса или дружественной (***friend***) классу. Общая форма оператор-функции-члена класса:
***возвращаемый_типимя_класса*::operator#(список_аргум)**
{/*тело функции*/}

После этого вместо ***operator#(a,b)*** можно писать ***a#b***. Здесь # представляет один из введенных выше символов. Примерами являются операторы >>и <<– перегружаемые операторы ввода- вывода. Отметим, что при перегрузке нельзя менять приоритет операторов и число операндов. Если оператор-функция-член класса перегружает бинарный оператор, то у функции будет только один параметр-объект, стоящий справа от знака оператора. Объект слева вызывает оператор-функцию и передается неявно с помощью указателя ***this***. В дружественную функцию указатель ***this***не передается, поэтому унарный оператор имеет один параметр, а бинарный – два.

Оператор присваивания не может быть дружественной функцией, а только членом класса.

Пример.

```
class Time
{
     chartimestr[30];
public:
            Time();                  // перегрузка конструкторов
            Time(char *str);
      };
```

Перегрузка операций

Язык С++ позволяет определять и применять к классам обозначения операций. Эта особенность, называемая перегрузкой операций дает классам возможность вести себя подобно встроенному типу данных.

Операции, допускающие перегрузку:

```
+  -  *  /  %  ^  &  |  ~  !  =  <>  +=  -=  *=  /=  %=  ^=  &=
|=  <<>><<=  >>=  ==  !=  <=  >=  &&  ||  ++  --  ->*  ->  ()  [ ]
```

Функции-операции и перегрузка операций подчиняются следующим правилам:

-приоритеты операций и правила ассоциации, принятые для встроенных типов данных, остаются неизменными при оценке выражений с перегруженными функциями-операциями;

-функция-операция не может изменить поведение операции по отношению к встроенным типам данных;

-функция-операция должна быть либо элементом класса, либо воспринимать один или

несколько аргументов, имеющих тип класса;

-функция-операция не может иметь аргументов по умолчанию;

-за исключением operator=() функции-операции наследуются.

Задание

Задание 1

Отладить программу с примером перегруженных операций

Составить класс Matr для работы с матрицами, предусмотрев в нем функции-члены сложения (+) и вычитания (-) матриц, а также дружественные функции ввода и вывода, умножения (*) матриц, умножения на число слева (*), умножения на число справа (*). Используя этот класс, вычислить

c=z*(a-b+u)*d*x, где a, b, u, d — матрицы, z, x — числа.

В этом примере перегрузку знаков операций проведем с помощью дружественных функций.

```
#include<iostream.h>
classmatr {
public:
    int n, m;                  // Размерымассиван×m
    double **M;
    matr (intnt=3, intmt=3); // Конструктор с параметрами по умолчанию
    matr (constmatr&y); // Конструктор копирования
    ~matr();
    matr operator + (matr&M2);
    matr operator - (matr&M2);
    friendmatr operator * (matr&M1, matr&M2);
    friendmatr operator * (matr&M1, double x);    // Умнож. матр. начисло
    friendmatr operator * (double z, matr&M2);    // Умнож. числанаматр.
    friendistream& operator>>(istream& in, matr&mt);
    friendostream& operator<<(ostream& out, matr&mt);
    };
matr:: matr (intnt, intmt)
{n=nt;
m=mt;
  M=new double *[n];
  for(int k=0; k<n; k++)
  M[k]=new double [m];
  }
matr:: matr (constmatr& y)
{n=y.n;
m=y.m;
  M=new double *[n];
  for(int k=0; k<n; k++)
 M[k]=new double [m];
 for(k=0; k<n; k++)
 for(int p=0; p<m; p++)
       M[k][p]=y.M[k][p];
 }
matr:: ~matr()
{for(int k=0; k<n; k++)
delete [ ] M[k];
delete [ ] M;
}
```

```
matrmatr:: operator + (matr&M2)
{int k, p, nt, mt;
nt=M2.n;
mt=M2.m;
   matr r(nt, mt), r0(0, 0);
   if ((n==nt) &&(m==mt))
    {for(k=0; k<nt;k++)
        for(p=0; p<mt; p++)
          r.M[k][p]=M[k][p] + M2.M[k][p];
          returnr;
    }
   else
    { cout«'"\n Сложение матриц невозможно";
    return r0;
    }
}
matrmatr:: operator - (matr&M2)
{int k, p, nt, mt;
nt=M2.n;
mt=M2.m;
   matr r(nt, mt), r0(0, 0);
   if ((n==nt) &&(m==mt))
   {for(k=0; k<nt; k++)
    for(p=0; p<mt; p++)
         r.M[k][p]=M[k][p] - M2.M[k][p];
         returnr;
   }
   eise
     {cout<<"\n Вычитание матриц невозможно";
     return r0;
     }
}
matr operator * (matr&M1, matr&M2)
{int k, p;
 double s;
   matr r(M1.n, M2.m), r0(0, 0);
   if(M1.m!=M2.n)
    {cout<<"\n Умножение матриц невозможно";
    return r0;
    };
   for(k=0;k<M1.n; k++)
   for(p=0; p<M2.m; p++)
          { s=0;
         for(int j=0; j<M1.m; j++)
           s=s+M1.M[k][j]*M2.M[j][p];
           r.M[k][p]=s;
           }
   return r;
   }
matr operator * (matr&M1, double x)
{int k, p, n, m;
   n=M1.n;
```

```
   m=M1.m;
   matr r(n, m);
   for(k=0; k<n; k++)
   for(p=0; p<m; p++)
         r.M[k][p]=M1.M[k][p]*x;
   return r;
   }
matr operator * (double z, matr&M2)
{int k, p, n, m;
n=M2.n;
m=M2.m;
matr r(n, m);
for(k=0; k<n; k++)
for(p=0; p<m;p++)
       r.M[k][p]=z*M2.M[k][p];
return r;
}
istream& operator>>(istream& in, matr&mt)
{for(int k=0; k<mt.n; k++)
  for(int p=0; p<mt.m; p++)
       in>>mt.M[k][p];
       return in;
 }
ostream& operator<<(ostream& out, matr&mt)
{for(int k=0; k<mt.n; k++)
 {cout<<"\n";
 for(int p=0; p<mt.m; p++)
 out<<" ";
}
return out;
}
void main ()
{intnl, ml, n2, m2;
   double  x, z;
   cout<<"Введите размеры матриц n1, ml, n2, m2 и x, z \n";
   cin>>n1>>m1>>n2>>m2>>x>>z;
   matr a(n1, ml), b(n2, m2), r0(0,0);
   cout<<"Введите матрицы a, b";
   cin>>a>>b;
   cout<<"\n Суммаматрица+b"<<(a+b);
   cout<<"\n Разность матриц a-b"<<(a-b);
   cout<<"\n Произведение матрица*b"<<(a*b);
   cout<<"\n Произведение на константу z слева z*a"<<(z*a);
cout<<"\n Произведение на константу x справа a*x"<<(a*x); }
```

Задание 2

Для каждого варианта написать программу, демонстрирующую работу класса

Вариант 1

а) Создать класс целых чисел. Определить операторы "++" и "+", как методы класса, а "- -" и "-" как дружественные функции. Операторы должны позволять осуществления операций, как с переменными данного класса, так и с переменными встроенного целого.

b) Создать класс *Set* – множество целых чисел, используя динамическую память. Определить операторы работы с множествами: "+" – объединение, "*" – пересечение, "-" вычитание, как дружественные функции, а "+=" – включение нового элемента в множество, "==" – сравнения на равенство, и др. как методы класса. Определить операторы "<<" и ">>". Также определить функцию определения принадлежности элемента множеству.

c) «**Комплексное число**» – **Complex**. Класс должен содержать несколько конструкторов и операции для сложения, вычитания, умножения, деления, присваивания. Создать два вектора размерности *n* изкомплексных координат. Передать их в функцию, которая выполняетсложение комплексных векторов.

Вариант 2

a) Создать класс 2-D координат. Определить операторы "+" и "-" как дружественные функции, а операторы присваивания и сравнения как методы класса. Должны быть возможность осуществления операций, как между координатами, так и между координатами и обычными числами.

b) Создать класс Stack – стек, используя динамическую память. Определить операторы "+" – сложения стеков, "=" – присваивания, "()" – выдачи нового стека содержащего последние n элементов - как методы класса. Определить операторы сравнения "==", "!=", "<", ">", как дружественные функции. Для реализации последних двух операторов определить функцию, вычисляющую норму элементов стека. Определить операторы ввода/вывода в поток. Класс должен быть полностью функционален, то есть содержать все необходимые конструкторы и деструктор.

c) Определить класс «**Дробь**» – **Fraction**в виде пары (*m*,*n*). Классдолжен содержать несколько конструкторов. Реализовать методы длясложения, вычитания, умножения и деления дробей. Перегрузить операции сложения, вычитания, умножения, деления, присваивания иоперации отношения.

Вариант 3

a) Создать класс 3-D координат. Определить операторы "+", "-", "=" как методы класса, а операторы сравнения как дружественные функции. Должны быть возможность осуществления операций, как между координатами, так и между координатами и обычными числами.

b) Создать класс *Queue* - очередь, используя динамическую память. Определить операторы "+" – сложения стеков, "=" – присваивания как методы класса. Определить операторы сравнения "==", "!=", "<", ">", как дружественные функции. Для реализации последних двух операторов определить функцию, вычисляющую норму элементов очереди. Перегрузить операторы "<<" и ">>" для ввода/вывода в поток, так и для вставки/извлечения элементов в/из очереди.

c) Разработать класс «**Вектор**» – **Vector**размерности *n* . Определить несколько конструкторов, в том числе конструктор копирования.Реализовать методы для вычисления модуля вектора, скалярного произведения, сложения, вычитания, умножения на константу. Перегрузить операции сложения, вычитания, умножения, инкремента, декремента, индексирования, присваивания для данного класса.

Вариант 4

a) Создать класс *Date* – дата, содержащая поля: день, месяц, год. Определить операторы "+" и "-", как методы класса, а "++" и "--" в обеих формах (префиксная и постфиксная) как дружественные функции. Оператор "+" должен позволять осуществление операции только с переменными встроенного*int*. (x=y+5;). Должна быть предусмотрена корректная работа с

високосными годами.

b) Создать класс *List* - очередь. Определить операторы "+" – сложения списков, "=" – присваивания как методы класса. Определить операторы сравнения "==", "!=", "<", ">", как дружественные функции. Перегрузить операторы "<<" и ">>" для ввода/вывода в поток, так и для вставки/извлечения элементов в/из очереди. Класс должен быть полностью функционален, то есть содержать все необходимые конструкторы и деструктор. Для упрощения работы используйте класс либо структуру *ListItem* для представления элементов списка, на которые и ссылается *List*.

c) Определить класс «**Квадратная матрица**» – **Matrix**. Класс должен содержать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для сложения, вычитания, умноженияматриц; вычисления нормы матрицы. Перегрузить операции сложения, вычитания, умножения и присваивания для данного класса.

Вариант 5

a) Создать класс действительных чисел. Определить операторы "++" и "+", как методы класса, а "- -" и "-" как дружественные функции. Операторы должны позволять осуществления операций, как с переменными данного класса, так и с переменными встроенного double.

b) Создать класс *Vector* – вектор, используя динамическую память. Определить операторы "+" – поэлементное сложения векторов, "-" – поэлементное вычитание векторов, "=" – присваивания, как методы класса. Определить операторы сравнения "==", "!=", "<", ">", как дружественные функции. Для реализации последних двух операторов определить функцию, вычисляющую норму элементов вектора. Перегрузить операторы "<<" и ">>" для ввода/вывода в поток. Класс должен содержать все необходимые конструкторы и деструктор.

c) Разработать класс **«Многочлен» – Polynom**степени *n* . Написать несколько конструкторов, в том числе конструктор копирования.Реализовать методы для вычисления значения полинома; сложения,вычитания и умножения полиномов. Перегрузить операции сложения,вычитания, умножения, инкремента, декремента, индексирования,присваивания.

Вариант 6

a) Создать класс целых чисел большой величины. Определить операторы "+" и "*", как методы класса, а "-" и "/" как дружественные функции. Перегрузить операторы инкремента и декремента в обеих формах (префиксная и постфиксная). Операторы должны позволять осуществления операций, как с переменными данного класса, так и с переменными встроенного long.

b) Создать класс *Matrix* – матрица, используя динамическую память. Определить операторы "+" –сложение матриц, "-" – вычитание матриц, "=" – присваивания, как методы класса. Определить операторы сравнения "==", "!=", "<", ">", как дружественные функции. Для реализации последних двух операторов определить функцию, вычисляющую норму элементов матрицы. Определить оператор "[]" так, чтобы обращение *[][]* к элементам имело смысл. Перегрузить операторы "<<" и ">>" для ввода/вывода в поток.

c) Построить классы для описания плоских фигур: круг, квадрат,прямоугольник. Включить методы для изменения объектов, перемещения на плоскости, вращения. Перегрузить операции, реализующиете же действия.

Вариант 7

a) Создать класс *Bool* – логические переменные. Определить операторы "+" – логическое ИЛИ, "*" – логическое И "^" – ИСКЛЮЧИТЕЛЬНОЕ ИЛИ, как методы класса, а операторы "==" и

"!=" как дружественные функции. Операторы должны позволять осуществления операций, как с переменными данного класса, так и с переменными встроенного int. (Если целое число отлично от нуля, считается что переменная истинна, в противном случае ложна).

b) Создать класс *String* – строку, используя динамическую память. Определить операторы "+" – сложение строк, "=" и "+=" – присваивания, как методы класса. Определить операторы сравнения "==", "!=", "<", ">", как дружественные функции. Операторы должны работать как со *String*, так и с *char**. Определить оператор "*[]*" для доступа к каждому символу в отдельности. Перегрузить операторы ввода/вывода в поток.

c) Определить класс «**Строка**» – **String**длины *n*. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для выполнения конкатенации строк, извлечения символа из заданной позиции, сравнения строк. Перегрузить операциисложения, индексирования, отношения, добавления(+ =) , присваивания для данного класса.

Вариант 8

a) Создать класс Stack – стек, используя динамическую память. Определить операторы "+" – сложения стеков, "=" – присваивания, "()" – выдачи нового стека содержащего последние n элементов - как методы класса. Определить операторы сравнения - "==", "!=", "<", ">", как дружественные функции. Для реализации последних двух операторов определить функцию, вычисляющую норму элементов стека. Определить операторы ввода/вывода в поток.

b) Определить класс Complex – комплексные числа. Определить все односимвольные операторы как дружественные операторы, а двухсимвольные как методы класса. Исключение – оператор присваивания, который может быть только методом класса и операторы вводы/вывода в поток. Сложение и вычитание должно производиться как с комплексными числами, так и со встроеннымdouble.

c) Разработать класс «**Множество (целых чисел, символов, строк**и т. д.)» – **Set**мощности *n*. Написать несколько конструкторов, в томчисле конструктор копирования. Реализовать методы для определенияпринадлежности заданного элемента множеству, пересечения, объединения, разности двух множеств. Перегрузить операции сложения,вычитания, умножения (пересечения), индексирования, присваивания.

Вариант 9

a) Создать класс *Time* – время, содержащая поля: часы, минуты, секунды. Определить операторы "+" и "-", как дружественные функции, а "++" и "--" в обеих формах (префиксная и постфиксная) как методы класса. Операторы должны позволять осуществления операций, как с переменными данного класса, так и с переменными встроенного *int* (обозначает секунды).

b) Создать класс *Queue* - очередь, используя динамическую память. Определить операторы "+" – сложения стеков, "=" – присваивания как методы класса. Определить операторы сравнения "==", "!=", "<", ">", как дружественные функции. Для реализации последних двух операторов определить функцию, вычисляющую норму элементов очереди. Перегрузить операторы "<<" и ">>" для ввода/вывода в поток, так и для вставки/извлечения элементов в/из очереди.

c) Разработать класс для массива строк. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для поэлементной конкатенации двух массивов, а также для вывода на экран всего массива и заданной строки. Перегрузить операции сложения, умножения, индексирования, присваивания для данного класса.

Вариант 10

a) Создать класс *Bool* – логические переменные. Определить операторы "+" – логическое ИЛИ,

"*" – логическое И "^" – ИСКЛЮЧИТЕЛЬНОЕ ИЛИ, как дружественные функции, а операторы "==" и "!=" как методы класса. Операторы должны позволять осуществления операций, как с переменными данного класса, так и с переменными встроенного int. (Если целое число отлично от нуля, считается что переменная истинна, в противном случае ложна.)

b) Создать класс *Set* – множество целых чисел, используя динамическую память. Определить операторы работы с множествами: "+" – объединение, "*" – пересечение, "-" вычитание, как методы класса, а "+=" – включение нового элемента в множество, "==" – сравнения на равенство, и др. как дружественные функции. Определить операторы "<<" и ">>". Также определить функцию определения принадлежности элемента множеству.

c) Составить описания класса, обеспечивающего представлениематрицы заданного размера *n*×*m*и любого минора в ней. Написать несколько конструкторов,в том числе конструктор копирования. Реализовать методы для отображения на экране как матрицы в целом, так и заданного минора, а
также для изменения минора; сложения, вычитания, умножения миноров. Перегрузить операции сложения, вычитания, умножения иприсваивания для данного класса.

Вариант 11

a) Создать класс *Date* – дата, содержащая поля: день, месяц, год. Определить операторы "+" и "-", как методы класса, а "++" и "--" в обеих формах (префиксная и постфиксная) как дружественные функции. Оператор "+" должен позволять осуществление операции только с переменными встроенного*int*. (x=y+5;). Должна быть предусмотрена корректная работа с високосными годами.

b) Создать класс *String* – строку, используя динамическую память. Определить операторы "+" – сложение строк, "=" и "+=" – присваивания, как дружественные функции. Определить операторы сравнения "==", "!=", "<", ">", как методы класса. Операторы должны работать как со *String*, так и с *char**. Определить оператор "[]" для доступа к каждому символу в отдельности. Перегрузить операторы "<<" и ">>" для ввода/вывода в поток.

c) Построить класс «**Булев вектор»** – **BoolVector**размерности *n* .Определить несколько конструкторов, в том числе конструктор копирования. Реализовать методы для выполнения поразрядных конъюнкции, дизъюнкции и отрицания векторов, а также подсчета числа единиц и нулей в векторе. Реализовать те же действия над векторами спомощью перегруженных операций. Перегрузить операции отношения и присваивания для данного класса.

Вариант 12

a) Создать класс 2-D координат. Определить операторы "+" и "-" как дружественные функции, а операторы присваивания и сравнения как методы класса. Должны быть возможность осуществления операций, как между координатами, так и между координатами и обычными числами.

b) Создать класс List - очередь. Определить операторы "+" – сложения списков, "-" – вычитание (как в множестве) как дружественные функции. Определить операторы сравнения "==", "!=", "<", ">", как методы класса. Перегрузить операторы "<<" и ">>" для ввода/вывода в поток, так и для вставки/извлечения элементов в/из очереди. Класс должен быть полностью функционален, то есть содержать все необходимые конструкторы и деструктор. Для упрощения работы используйте класс либо структуру ListItem для представления элементов списка, на которые и ссылается List.

c) Определить класс «**Булева матрица»** – **BoolMatrix**размерности *n*×*m*. Класс должен содержать несколько конструкторов, в томчисле конструктор копирования. Реализовать методы для логическогосложения (дизъюнкции), умножения и инверсии матриц. Реализоватьметоды

для подсчета числа единиц в матрице. Перегрузить операции для логического сложения, умножения и инверсии матриц, а также операцию присваивания.

Вариант 13

a) Создать класс *Date* – дата, содержащая поля: день, месяц, год. Определить операторы "+" и "-", как методы класса, а "++" и "--" в обеих формах (префиксная и постфиксная) как дружественные функции. Оператор "+" должен позволять осуществление операции только с переменными встроенного*int*. (x=y+5;). Должна быть предусмотрена корректная работа с високосными годами.

b) Создать класс *Vector* – вектор, используя динамическую память. Определить операторы "+" – поэлементное сложения векторов, "-" – поэлементное вычитание векторов, "=" – присваивания, как методы класса. Определить операторы сравнения "==", "!=", "<", ">", как дружественные функции. Для реализации последних двух операторов определить функцию, вычисляющую норму элементов вектора. Перегрузить операторы "<<" и ">>" для ввода/вывода в поток. Класс должен содержать все необходимые конструкторы и деструктор.

c) Построить классы для описания плоских фигур: круг, квадрат,прямоугольник. Включить методы для определения периметра и площади фигур. Перегрузить операции, реализующиете же действия.

Контрольные вопросы.

1. Что означает перегрузка операций?
2. Какие общие принципы перегрузки?
3. Какие операции можно, а какие нельзя перегружать?
4. В чём отличие перегрузки операций с помощью функций-членов и с помощью дружественных функций?
5. В чём отличие перегрузки унарных и бинарных операций?

Содержание отчёта:

Постановка задачи.
Текст программы.
Комментарии к ведённым обозначениям и описание используемых функций.
Результаты работы программы.

Практическая работа №5. Одиночное наследование классов.

Цель: научиться реализовывать принцип наследования в классах; научиться реализовывать одиночное наследование.

Подготовительная часть:

Наследование. Одиночное наследование. Уровни доступ к базовому классу. Особенности использования конструкторов и деструкторов в реализации одиночного наследования.

Содержание отчёта:

- ✓ Постановка задачи.
- ✓ Текст программы.
- ✓ Комментарии к ведённым обозначениям и описание используемых функций.
- ✓ Результаты работы программы.

Задания:

Задание №1

Откомпилировать пример.

Определите общий базовый класс **Fruit** описывающий некоторые характеристики фруктов. Также, определите два класса **Apple** и **Orange**, производные от базового класса, которые должны содержить специальную информацию о соответствующем фрукте. Создайте объекты указанных классов, заполните их данными и, с помощью функции, отобразите ее на экране.

```
// Пример наследования классов
#include <iostream>
#include <cstring>
using namespace std;
enum yn {no, yes};
enum color {red, yellow, green, orange};
void out (enum yn x);
char *c[ ] = {"red", "yellow", "green", "orange"};
// Родовой класс фруктов
class fruit {
//В этом базовом классе все элементы открыты
public:
enum yn annual;
enum yn perennial;
enum yn tree;
enum yn tropical;
enum color clr;
char name [40] ;
};
// Производныйклассяблок
class Apple: public fruit {
enum yn cooking;
enum yn crunchy;
enum yn eating;
public:
void seta (char *n, enum color c, enum yn ck, enum yn crchy,enum yn e) ;
void show ( ) ;
};
// Производныйклассапельсинов
class Orange: public fruit {
```

```
enum yn juice;
enum yn sour;
enum yn eating;
public:
void seto(char *n, enum color c, enum yn j, enum yn sr, enum yn e) ;
void show () ;
};
void Apple:: seta (char *n, enum color c, enum yn ck, enum yn crchy, enum yn e)
{
strcpy (name, n) ;
annual = no;
perennial = yes;
tree = yes;
tropical = no;
clr = c;
cooking = ck;
crunchy = crchy;
eating = e;
}
void Orange:: seto ( char *n, enum color c, enum yn j, enum yn sr, enum yn e)
{
strcpy (name, n) ;
annual = no;
perennial = yes;
tree = yes;
tropical = yes;
clr = c;
juice = j;
sour = sr;
eating = e;
}
void Apple :: show ( )
{
cout << name << " яблоко — это: " <<"\n";
cout<< "Однолетнее растение: "; out (annual);
cout<< "Многолетнее растение: "; out (perennial) ;
cout << "Дерево: "; out (tree);
cout <<"Тропическое: "; out (tropical) ;
cout<< "Цвет: " << c [clr] << "\n";
cout<< "Легко приготавливается: "; out (cooking) ;
cout<< "Хрустит на зубах: "; out (crunchy) ;
cout << "Съедобное: "; out (eating) ;
cout << "\n";
}
void Orange :: show ( )
{
cout << name << " апельсин - это: " << "\n";
cout<<"Однолетнее растение: "; out (annual);
cout<<"Многолетнее растение: "; out (perennial) ;
cout<<"Дерево: "; out (tree);
cout<<"Тропическое: "; out (tropical) ;
cout<<"Цвет: " << c[clr] << "\n";
```

```
cout<<"Годится для приготовления сока: "; out (juice);
cout<<"Кислый: "; out (sour);
cout<<"Съедобный: "; out (eating) ;
cout<<"\n";
}
void out (enum yn x)
{
if (x==no) cout<<"нет\n";
else cout<<"да\n";
}
int main ( )
{
Apple al, a2;
Orange ol, o2;
al . seta ( "Краснаяпрелесть", red, no, yes, yes);
a2 .seta ( "Джонатан", red, yes, no, yes);
ol. seto ("Пуп", orange, no, no, yes);
o2 . seto ( "Валенсия" , orange, yes, yes, no);
al.show ( );
a2.show ( );
ol.show ( );
o2.show ( );
return 0;
}
```

Задание №2

В приведенном ниже фрагменте добавьте конструктор для класса **myderived.** Он должен передать указатель на инициализируемую строку конструктору класса mybase. Кроме того, конструктор **myderived()** должен инициализировать переменную **len** длиной строки.

```
#include <iostream>
#include <cstring>
using namespace std;
class mybase {
char str [80] ;
public:
mybase(char *s) { strcpy(str, s) ; }
char *get() { return str; }
class myderived: public mybase {
int len;
public:
// добавьте здесь конструктор myderived ()
int getlen() { return len; }
void show() { cout <<get() <<'\n'; }
};
int main ( )
{
myderived ob ("привет") ;
ob . show ( ) ;
cout<< ob.getlen()<< '\n';
return 0;
}
```

Задание №3

Создайте исходный базовй класс **building** для хранения числа этажей и комнат в здании, а также общую площадь комнат. Создайте класс **house**, который наследует класс **building** и хранит число ванных комнат и число спален. Создайте произвольный класс **office**, который наследует класс **building** и хранит число огнетушителей и телефонов. Создайте объекты указанных классов, заполните их данными и, с помощью функции, отобразите ее на экране.

Индивидуальные задания
Задания

Вариант 1

а) Создать иерархию классов *игра – спортивная игра – волейбол*. Определить конструкторы, деструктор, оператор присваивания и другие необходимые функции. Продемонстрировать работу классов.

b) Создать класс *колесо*, имеющий радиус. Определить конструкторы и методы доступа. Создать класс *автомобиль*, имеющий колеса и строку обозначающую фирму-производителя. Создать производный класс *грузовой автомобиль*, отличающийся грузоподъемностью. Определить конструкторы, деструктор и другие необходимые функции. Продемонстрировать работу классов.

Вариант 2

а) Создать класс *студент*, у которого есть имя, специальность, год обучения и средний балл. Определить функции установки, изменения данных, сравнения. Определить конструкторы, деструктор и другие необходимые функции. Создать производный класс *студент-дипломник*, для которого определена тема дипломной работы. Так же, необходимо определить все необходимые функции. Продемонстрировать работу классов.

b) Создать класс *комната*, имеющая площадь. Определить конструкторы и методы доступа. Создать класс *однокомнатных квартир*, содержащий комнату и кухню (ее площадь), этаж (комната содержится в классе однокомнатная квартира). Определить конструкторы, методы доступа. Определить производный класс однокомнатные квартиры с адресом (дополнительное поле - адрес). Определить конструкторы, деструктор и вывод в поток.

Вариант 3

а) Создать класс *мебель,* содержащий информацию о цене, стиле и области применения (офисная, кухонная и др. мебель). Для задания текстовых полей использовать динамическую память. Определить производные классы: *стол* и *стул*. Определить конструкторы, деструктор, оператор присваивания и другие необходимые функции. Продемонстрировать работу классов.

b) Создать класс *гараж*, имеющий площадь. Определить конструкторы и методы доступа. Создать класс *дом*, содержащий комнаты, кухню (ее площадь) и гараж. Определить производный класс дача (дополнительный параметр – количество земли). Определить конструкторы, деструктор и вывод в поток. Продемонстрировать работу классов.

Вариант 4

а) Создать иерархию классов *человек* и *сотрудник*, занимающий соответствующий пост и получающий соответствующую зарплату. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *карта*, имеющая ранг и масть. Карту можно перевернуть и открыть. Создать класс - *колода карт*, содержащий карты. Создать два производных класса от колоды карт, в одном карты могут извлекаться только по порядку, в другом – случайным образом. Продемонстрировать работу классов.

Вариант 5

a) Создать класс жидкость, имеющий название (указатель на строку), плотность. Определить конструкторы, деструктор и операторы вывода в поток. Создать производный класс - спиртные напитки, имеющий крепость. Определить функции переназначения плотности и крепости. Продемонстрировать работу классов.

b) Создать класс *кнопка*, содержащая некий текст. Определить конструкторы и метод доступа. Создать класс *окно*, содержащий кнопку и координаты окна. Определить конструкторы и деструктор. Определить производный класс *окно с кнопкой и сообщением*. Определить конструкторы, деструкторы и операторы вывода в поток. Продемонстрировать работу классов.

Вариант 6

a) Создать класс *человек*, имеющий имя (указатель на строку), возраст, вес. Определить конструкторы, деструктор и оператор присваивания. Создать производный класс - *совершеннолетний*, имеющий номер паспорта. Определить конструкторы по умолчанию и с разным числом параметров, деструкторы, операторы вывода в поток. Определить функции переназначения возраста и номера паспорта. Продемонстрировать работу классов.

b) Определить класс *корова* состоящее из следующих полей: идентификационный номер – должно быть гарантировано уникально (для чего использовать статический счетчик), средний надой, возраст, кличку и породу. Определить класс *стадо*, состоящее из неограниченного количества коров. Определить методы вставки, удаление, определения среднего надоя по стаду и общий надой, и другие необходимые функции. Продемонстрировать работу классов.

Вариант 7

a) Создать иерархию классов *здание*, *административное здание* и *жилое здание*. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *студент*, у которого есть имя, специальность, год обучения и средний балл. Определить функции установки, изменения данных, сравнения. Определить конструкторы, деструктор и другие функции. Создать класс *группа* содержащий студентов (неограниченное количество). Определить методы вставки, удаление студентов, определения среднего балла по группе, конструкторы, деструкторы и др. необходимые функции. Продемонстрировать работу классов.

Вариант 8

a) Создать иерархию классов: *учебное заведение* – абстрактный базовый класс и *дошкольное у.з.*, *среднее у.з.* и *высшее у.з.* – производные классы. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *двигатель*, у которого есть фирма-производитель, тип, мощность. Определить функции установки, изменения параметров двигателя. Создать иерархию классов: *корабль* – базовый класс и *пассажирский пароход* – производный. Корабль имеет двигатель, грузоподъемность, водоизмещение, название, порт приписки. Продемонстрировать работу классов.

Вариант 9

a) Создать иерархию классов *пресса - газета, журнал* и *электронное издание*. Определить поля: название издания, тираж, подписной индекс, периодичность издания. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса.

b) Определить класс *Processor* - процессор, содержащий информацию о названии процессора, его частоте, используемой технологии производства и размере внутренней памяти. Определить класс *Computer* - компьютер, состоящий из процессора и других компонентов. Определить конструкторы, функции вывода в поток и другие необходимые функции. Продемонстрировать работу классов.

Вариант 10

a) Создать иерархию классов *транспорт – воздушный транспорт – вертолет*. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Определить класс *химический элемент,* содержащий информацию о названии элемента его химических свойствах. Определить класс *медикаменты*, содержащий разное количество хим. элементов и в разном количестве. Определить конструкторы, функции вывода в поток и другие необходимые функции. Продемонстрировать работу классов.

Вариант 11

a) Создать иерархию классов *датчик* – абстрактный базовый класс и датчики температуры, влажности и скорости ветра. Для каждого класса определить свои единицы измерения и способ снятия данных о значениях состояния окружающей среды. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *Устройство сбора информации* о погоде состоящее из датчиков по заданию а. Для снятия значений создать класс генератор значений для каждого датчика. Продемонстрировать работу устройства.

Вариант 12

a) Создать иерархию классов *Шахматная фигура* – абстрактный класс, содержащий поле – цвет. Создать производные классы все фигуры, содержащие свое название и координаты позиции на доске. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *Шахматы,* состоящее из набора фигур из задания а, и шахматной доски – двумерный массив 8 на 8. Должна быть возможность удаления фигур с поля. Определить конструктор динамически создающий фигуры и задающий их позиции в шахматной нотации (E2). Определить конструктор копий и оператор присваивания. Продемонстрировать работу классов.

Вариант 13

a) Создать иерархию классов *здание, административное здание* и *жилое здание*. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *гараж*, имеющий площадь. Определить конструкторы и методы доступа. Создать класс *дом*, содержащий комнаты, кухню (ее площадь) и гараж. Определить производный класс дача (дополнительный параметр – количество земли). Определить конструкторы, деструктор и вывод в поток. Продемонстрировать работу классов.

Контрольные вопросы:

1. Когда базовый класс наследуется производным классом как открытый, что происходит с его открытыми членами? Что происходит с его закрытыми членами?
2. Когда базовый класс наследуется производным классом как закрытый, что происходит с его закрытыми и открытыми членами?
3. Объясните, что означает ключевое слово **protected.** (Рассмотрите два случая: когда оно используется для задания элементов класса и когда оно используется в качестве спецификатора доступа.)
4. При наследовании одного класса другим, когда вызываются конструкторы классов? Когда вызываются их деструкторы?

ПРАКТИЧЕСКАЯ РАБОТА №6. Наследование и композиция

Цели работы:

– изучение наследования, преимущества и недостатки;
– изучение композиции;
– изучение правил определения наследования и композиции;
– изучение форм наследования;
– изучение инициализаторов;
– принцип подстановки;
– наследование и композиция – что выбрать.

Основные понятия

Мотивация

Наследование один из трех основных механизмов объектно-ориентированного наследования. Наследование может быть изучено с двух точек зрения: разработчика и пользователя класса.

С точки зрения разработчика наследование означает, что поведение и свойства производного класса, являются расширением свойств базового класса. А с точки зрения пользователя – наследование обозначает существование ряда частично взаимозаменяемых классов с единым интерфейсом. (Инженеры ИТ и машиностроения оба, в первую очередь являются инженерами).

Преимущества наследования:

- уменьшение объема кода и его повторное использование;
- уменьшение количества ошибок и себестоимости;
- единый интерфейс и подставляемость;
- увеличение скорости разработки;

Издержки наследования: единственной издержкой наследования можно назвать некоторое падение скорости выполнения кода. Все же заботы об эффективности не должны входить в противоречие с преимуществами, так как потери, на самом деле, не существенны, так как частично их можно обойти использованием подставляемых функций.

Определение и использование наследования

Наследование обозначается указанием базового класса после имени производного, через двоеточие. Пример наследования:

```
class Animal{
      int NofLegs;
public:
      void Say(){ cout<<”!!!”; }
};
class Dog: public Animal{  // наследование
...
};
void main(){
 Dog d;
 d.Say();
}
```

Из примера видно, что функция и переменная, определенные в классе *животное*

импортируются и могут быть использованы для объекта класса *собака*. Обстоятельство, усложняющее восприятие данного механизма, состоит в том, что производный класс может переопределить поведение, заданное в базовом классе. (Страус и пингвин не летают, хотя являются птицами, а утконос откладывает яйца, хотя является млекопитающим.) Мы вернемся к данной теме в последующих лабораторных работах.

Принцип подстановки

Кроме всего выше сказанного, наследование предполагает, что объект производного класса может быть использован вместо объекта базового:

```
void main(){
 Animal *ptr = new Dog;
}
```

Из примера видно что имеется указатель базового класса – значит ожидается использование объекта базового, однако, указателю присваивается объект производного.

Формы наследования

В отношении наследования, C++ очень богатый язык, так как существуют пять разных форм наследования. На этой лабораторной работе мы исследуем три из них, те что называются простыми.

- открытое;
- закрытое;
- защищенное;

Разные формы наследования используются, для того чтобы изменить статус доступа для переменных членов класса. Например, при использовании закрытого наследования, все доступные (открытые и защищенные) переменные базового класса становятся закрытыми в производном. А при защищенном наследовании – защищенными. Закрытые переменные базового класса при любой форме наследования становятся недоступными. Кроме того формы наследования, отличаются тем, что открытое наследование создает подтип данных, то есть соответствует принципу подстановки, а защищенное и открытое – нет.

Композиция

Композиция это еще один механизм, связанный с ООП, обозначающий отношение между объектами, тогда как наследование это отношение между классами.

Наследование реализует отношение *быть "is a"*. *Собака* является *млекопитающим*, а *млекопитающее – животным*.

Композиция же реализует отношение *содержать "has a"*. *Автомобиль* содержит *двигатель* и *колеса*.

Определение композиции

На самом деле композиция используется очень широко, так как и встроенные переменные имеют тип и используются при определении класса. Однако при использовании переменных встроенного типа вопросов не возникает, чего не скажешь о пользовательских классах. Само по себе определение не сложно, сложности состоят в использовании конструкторов.

```
class Car{
 Engine e;
};
```

Инициализаторы

Как известно закрытые переменные базового класса становятся недоступными,

следовательно, их невозможно инициализировать в конструкторе производного и, кроме того, это противоречит принципу повторного использования кода. Выход один – вызов конструктора базового класса. Так оно и происходит, но только для конструкторов по умолчанию и копий, сгенерированного компилятором. Все остальные должны быть вызваны вручную. Та же проблема и при композиции, так же может быть использован только конструктор по умолчанию и копий[1]. Для решения этих проблем используются инициализаторы – позволяющие вызывать любые конструкторы и производить любую инициализацию.

```
class Engine{
 int power;
public:
 Engine(int p){power=p;}
};
class Transport{
...
public:
 Transport(char*);
};
class Car:public Transport{ // наследование
 Engine e;                  // композиция
public:
 Car():Transport("automobile"),e(10){}
};
```

Для вызова конструкторы базового класса, после круглых скобок конструктора производного через круглые скобки записывается конструктор базового, при чем можно передавать параметры производного базовому. При композиции дела обстоят так же, но записывается имя переменной а не конструктора.

Что выбрать

Так как и наследование и композиция являются инструментами реутилизации кода, встает вполне осмысленный вопрос: когда использовать наследование и когда композицию. По этому поводу существует много разных рекомендаций, однако, легче всего запомнить следующее правило: нужно поставить себе вопрос: новый класс является ли подтипом (*Dog* ***is an*** *Animal*), если мы получили утвердительный ответ значит должно быть использовано наследование; в противном случае другой вопрос: не является ли новый класс контейнером (*Car* ***has a*** *door*) – в этом случае композиция.

К сожалению, для некоторых случаев данная стратегия бесполезна. К примеру, класс *множество* можно создать на базе класса *список*, но какой механизм использовать. Можно наследование, а можно композицию. В этом случае правила сложнее. Нужно задать себе несколько вопросов:

- будут ли объекты нового класса подставляться в место старого?
- нужно ли переопределить какую-нибудь виртуальную функцию?
- обрабатывает ли новый тип те же сообщения что и старый?
- Не является ли базовый класс абстрактным?

Если ответы положительны, используйте наследование[2].

Задания

Вариант 1

[1] Причина того, что компилятор создает код вызова только для этих конструкторов, состоит в том, что компилятор знает, как их вызывать и что им передавать в параметрах.

[2] Часто для замены композиции наследованием используется закрытое наследование – создающее подкласс, а не подтип.

a) Создать иерархию классов *игра – спортивная игра – волейбол*. Определить конструкторы, деструктор, оператор присваивания и другие необходимые функции. Продемонстрировать работу классов.

b) Создать класс *колесо*, имеющий радиус. Определить конструкторы и методы доступа. Создать класс *автомобиль*, имеющий колеса и строку обозначающую фирму-производителя. Создать производный класс *грузовой автомобиль*, отличающийся грузоподъемностью. Определить конструкторы, деструктор и другие необходимые функции. Продемонстрировать работу классов.

Вариант 2

a) Создать класс *студент*, у которого есть имя, специальность, год обучения и средний балл. Определить функции установки, изменения данных, сравнения. Определить конструкторы, деструктор и другие необходимые функции. Создать производный класс *студент-дипломник*, для которого определена тема дипломной работы. Так же, необходимо определить все необходимые функции. Продемонстрировать работу классов.

b) Создать класс *комната*, имеющая площадь. Определить конструкторы и методы доступа. Создать класс *однокомнатных квартир*, содержащий комнату и кухню (ее площадь), этаж (комната содержится в классе однокомнатная квартира). Определить конструкторы, методы доступа. Определить производный класс однокомнатные квартиры с адресом (дополнительное поле - адрес). Определить конструкторы, деструктор и вывод в поток.

Вариант 3

a) Создать класс *мебель,* содержащий информацию о цене, стиле и области применения (офисная, кухонная и др. мебель). Для задания текстовых полей использовать динамическую память. Определить производные классы: *стол* и *стул*. Определить конструкторы, деструктор, оператор присваивания и другие необходимые функции. Продемонстрировать работу классов.

b) Создать класс *гараж*, имеющий площадь. Определить конструкторы и методы доступа. Создать класс *дом*, содержащий комнаты, кухню (ее площадь) и гараж. Определить производный класс дача (дополнительный параметр – количество земли). Определить конструкторы, деструктор и вывод в поток. Продемонстрировать работу классов.

Вариант 4

a) Создать иерархию классов *человек* и *сотрудник*, занимающий соответствующий пост и получающий соответствующую зарплату. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *карта*, имеющая ранг и масть. Карту можно перевернуть и открыть. Создать класс - *колода карт*, содержащий карты. Создать два производных класса от колоды карт, в одном карты могут извлекаться только по порядку, в другом – случайным образом. Продемонстрировать работу классов.

Вариант 5

a) Создать класс жидкость, имеющий название (указатель на строку), плотность. Определить конструкторы, деструктор и операторы вывода в поток. Создать производный класс - спиртные напитки, имеющий крепость. Определить функции переназначения плотности и крепости. Продемонстрировать работу классов.

b) Создать класс *кнопка*, содержащая некий текст. Определить конструкторы и метод доступа. Создать класс *окно*, содержащий кнопку и координаты окна. Определить конструкторы и деструктор. Определить производный класс *окно с кнопкой и сообщением*. Определить конструкторы, деструкторы и операторы вывода в поток. Продемонстрировать работу классов.

Вариант 6

a) Создать класс *человек*, имеющий имя (указатель на строку), возраст, вес. Определить конструкторы, деструктор и оператор присваивания. Создать производный класс - *совершеннолетний*, имеющий номер паспорта. Определить конструкторы по умолчанию и с разным числом параметров, деструкторы, операторы вывода в поток. Определить функции переназначения возраста и номера паспорта. Продемонстрировать работу классов.

b) Определить класс *корова* состоящее из следующих полей: идентификационный номер – должно быть гарантировано уникально (для чего использовать статический счетчик), средний надой, возраст, кличку и породу. Определить класс *стадо*, состоящее из неограниченного количества коров. Определить методы вставки, удаление, определения среднего надоя по стаду и общий надой, и другие необходимые функции. Продемонстрировать работу классов.

Вариант 7

a) Создать иерархию классов *здание, административное здание* и *жилое здание*. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *студент*, у которого есть имя, специальность, год обучения и средний балл. Определить функции установки, изменения данных, сравнения. Определить конструкторы, деструктор и другие функции. Создать класс *группа* содержащий студентов (неограниченное количество). Определить методы вставки, удаление студентов, определения среднего балла по группе, конструкторы, деструкторы и др. необходимые функции. Продемонстрировать работу классов.

Вариант 8

a) Создать иерархию классов: *учебное заведение* – абстрактный базовый класс и *дошкольное у.з., среднее у.з.* и *высшее у.з.* – производные классы. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *двигатель*, у которого есть фирма-производитель, тип, мощность. Определить функции установки, изменения параметров двигателя. Создать иерархию классов: *корабль* – базовый класс и *пассажирский пароход* – производный. Корабль имеет двигатель, грузоподъемность, водоизмещение, название, порт приписки. Продемонстрировать работу классов.

Вариант 9

a) Создать иерархию классов *пресса - газета, журнал* и *электронное издание*. Определить поля: название издания, тираж, подписной индекс, периодичность издания. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса.

b) Определить класс *Processor* - процессор, содержащий информацию о названии процессора, его частоте, используемой технологии производства и размере внутренней памяти. Определить класс *Computer* - компьютер, состоящий из процессора и других компонентов. Определить конструкторы, функции вывода в поток и другие необходимые функции. Продемонстрировать работу классов.

Вариант 10

a) Создать иерархию классов *транспорт – воздушный транспорт – вертолет*. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Определить класс *химический элемент,* содержащий информацию о названии элемента его химических свойствах. Определить класс *медикаменты*, содержащий разное количество хим.

элементов и в разном количестве. Определить конструкторы, функции вывода в поток и другие необходимые функции. Продемонстрировать работу классов.

Вариант 11

a) Создать иерархию классов *датчик* – абстрактный базовый класс и датчики температуры, влажности и скорости ветра. Для каждого класса определить свои единицы измерения и способ снятия данных о значениях состояния окружающей среды. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *Устройство сбора информации* о погоде состоящее из датчиков по заданию a. Для снятия значений создать класс генератор значений для каждого датчика. Продемонстрировать работу устройства.

Вариант 12

a) Создать иерархию классов *Шахматная фигура* – абстрактный класс, содержащий поле – цвет. Создать производные классы все фигуры, содержащие свое название и координаты позиции на доске. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *Шахматы,* состоящее из набора фигур из задания a, и шахматной доски – двумерный массив 8 на 8. Должна быть возможность удаления фигур с поля. Определить конструктор динамически создающий фигуры и задающий их позиции в шахматной нотации (E2). Определить конструктор копий и оператор присваивания. Продемонстрировать работу классов.

Вариант 13

a) Создать иерархию классов *здание, административное здание* и *жилое здание*. Определить конструктор копирования, оператор присваивания через соответствующие функции базового класса. Продемонстрировать работу классов.

b) Создать класс *гараж*, имеющий площадь. Определить конструкторы и методы доступа. Создать класс *дом*, содержащий комнаты, кухню (ее площадь) и гараж. Определить производный класс дача (дополнительный параметр – количество земли). Определить конструкторы, деструктор и вывод в поток. Продемонстрировать работу классов.

Контрольные вопросы:

1. Чем различаются формы наследования?
2. Что собой представляет композиция?
3. Каковы преимущества наследования?
4. Как работают конструкторы при наследовании?
5. Что такое принцип подстановки?
6. В каких случаях используется наследование, а в каких композиция?
7. Как инициализируются элементы контейнера?

Содержание отчёта:

Постановка задачи.
Текст программы.
Комментарии к ведённым обозначениям и описание используемых функций.
Результаты работы программы.
Ответы на контрольные вопросы

ПРАКТИЧЕСКАЯ РАБОТА №7. Множественное наследование

Цели работы:

– изучение правил определения множественного наследования;
– изучение преимуществ и недостатков множественного наследования;
– проблем связанных с использованием множественного наследования;
– изучение решений проблем;

Основные понятия

Множественное наследование, представляет собой наследование от двух и более классов. Для того, что бы понять, зачем нам необходимо множественное наследование нужно вспомнить, что одиночное наследование не решает всех проблем, так как иногда вынуждает выбирать между двумя подходящими базовыми классами.

Интересно высказывание Буча относительно этого механизма: «Множественное наследование - как парашют: как правило, он не нужен, но, когда вдруг он понадобится, будет жаль, если его не окажется под рукой»[3].

Таким образом, данный механизм абсолютно необходим. Однако, он используется далеко не во всех языках, но реализован в C++. Приведем пример:

Необходимо описать класс Окно с кнопкой и заголовком, при чем классы Окно, Окно с кнопкой, Окно с заголовком уже созданы и представляют собой иерархию классов.

Оба класса подходят под определение базового, для нового, еще не созданного класса. В отсутствии множественного наследования пришлось бы выбирать и дописывать недостающую функциональность. Мало того, что пользователь ставится перед выбором, дописывание само по себе противоречит принципу минимизации кода и его повторного использования.

Множественное наследование имеет место и в жизни:

- Практикант является студентом и временным сотрудником. Он получает оценку за свой труд, как студент и, в тоже время, выполняет функции возложенные на сотрудника, а возможно еще и получает зарплату.
- Зубной техник является медработником, так как закончил медицинское учебное заведение; в то же время он выполняет работу более свойственную кузнецу или ювелиру.
- Утконос кормит своих детенышей молоком, однако откладывает яйца.
- Акции являются бумагой и ценной собственностью.
- Класс *iostream* наследует классы: *istream* и *ostream*.

Правда, последний пример не совсем из жизни, а опять-таки из программирования.

Определение

Множественное наследование определяется следующим образом:

```
class Student{
public:
  int mark;
...
};
class Worker{
```

[3]Буч Г. *«Объектно-ориентированный анализ и проектирование с примерами приложений на C++»*. Москва «Бином» 1998 г.

```
public:
  int salary;
};
class Practicant: public Student, public Worker{
};
void PutMark(Student& s, int mark){
 s.mark = mark;
}
void PutSalary(Worker& w, int salary){
 w. salary = salary;
}

void main(){
Practicant p;
        PutMark(p,5);
        PutSalary(p,200);
}
```

Из примера видно, что производный класс импортирует поведение обоих базовых классов, так как, подходит, как параметр обеих глобальных функций, то есть принцип подстановки остается в силе! Конечно же, остается в силе и возможность использования форм наследования, можно открыто наследовать от одного и закрыто – от другого базового класса.

Проблемы

Все было бы хорошо, если бы не одно "но". Как и любой мощный, красивый инструмент, множественное наследование имеет свои недостатки, которые стали причиной исключения множественного наследования из многих современных языков. Проблемы возникают из-за проявляющихся неоднозначностей. Предположим, что в обоих базовых классах существуют поля с одним и тем же именем.

```
class A{
public:
  int x;
};
class B{
public:
  int x;
};

class C: public A, public B{

};
void main(){
 C c;
 c.x = 10;
}
```

Таким образом класс *C* содержит две переменные с одним и тем же именем, так как наследуются обе, потому что для каждого базового класса переменная может иметь свой смысл. Компилятор не может решить, какой из наследованных переменных присвоить новое значение. Такая же ситуация имеет место и с функциями.

Решение данной неоднозначности состоит в использовании уточнения имени переменной.

Имена переменных могут совпадать, но имена классов нет[4]. То есть, для указания какая переменная используется нужно указать класс, от которого наследована переменная:

```
c.A::x = 10;
c.B::x = 5;
```

Еще сложнее обстоят дела в том случае если классы *A* и *B* родственны, то есть происходят от одного и того же класса, так как показано на рисунке 1, хотя может быть и какой-либо более сложный случай. В данной конфигурации существуют две одинаковые переменные, с одним и тем же смыслом. Различать их можно, так как показано выше. Но проблема состоит в том, что они и по смыслу одинаковые, а функции определенные на втором уровне иерархии будут работать с собственной копией переменной, что часто приводит к трудноуловимым семантическим ошибкам. Кроме того, в этом случае, конструктор самого базового класса вызывается дважды. Для решения этой проблемы используется последняя форма наследования: виртуальное.

```
class A{
public:
  int x;
  A(int x){this->x=x;}
};
class B: virtual public A{
public:
  B(int x):A(x){}
};
class C: virtual public A{
public:
  C(int x):A(x){}
};
class D: public B, public C{
public:
  D(int x):A(x),B(x),C(x){}
}
```

Как видно классы *B* и *C* должны оба виртуально наследовать класс *A*. Сразу же решается проблема существования двух идентичных переменных. Кроме того, в этом случае необходимо вызывать конструктор самого базового класса вручную, так как показано в примере. Однако не всегда вся иерархия проектируется одним программистом, а некоторые классы уже возможно откомпилированы, в таких случаях решить проблему практически не возможно.

Задания

Задание 1

Отладить программу.
Следующая программа COMPUTER. CPP порождает класс computer, используя базовые классы computer_screen и mother_board:

```
#include
#include
class computer_screen
{
```

[4] В соответствии с новым стандартом могут существовать классы с одинаковыми именами, но в разных пространствах имен.

```
public:
computer_screen(char *, long, int, int);
void show_screen(void);
private:
char type[32];
long colors;
int x_resolution;
int y_resolution;
};
computer_screen::computer_screen(char *type, long colors, int x_res, int y_res)

{
strcpy(computer_screen::type, type);
computer_screen::colors = colors;
computer_screen::x_resolution = x_res;
computer_screen::y_resolution = y_res;
}
void computer_screen::show_screen(void)
{
cout << "Тип экрана: " << type << endl;
cout << "Цветов: " << colors << endl;
cout << "Разрешение: " << x_resolution << " на " << y_resolution << endl;
}
class mother_board
{
public:
mother_board(int, int, int);
void show_mother_board(void);
private:
int processor;
int speed;
int RAM;
};
mother_board::mother_board(int processor, int speed, int RAM)
{
mother_board::processor = processor;
mother_board::speed = speed;
mother_board::RAM = RAM;
}
void mother_board::show_mother_board(void)
{
cout << "Процессор: " << processor << endl;
cout << "Частота: " << speed << "МГц" << endl;
cout << "ОЗУ: " << RAM << " МБайт" << endl;
}

class computer : public computer_screen, public mother_board
{
public:
computer(char *, int, float, char *, long, int, int, int, int, int);
void show_computer(void);
private:
```

```
char name [64];
int hard_disk;
float floppy;
};

computer::computer(char *name, int hard_disk, float floppy, char *screen, long colors, int x_res, int
y_res, int processor, int speed, int RAM) : computer_screen(screen, colors, x_res, y_res),
mother_board(processor, speed, ram)
{
strcpy(computer::name, name);
computer::hard_disk = hard_disk;
computer::floppy = floppy;
}

void computer::show_computer(void)
{
cout << "Тип: " << name << endl;
cout << "Жесткий диск: " << hard_disk << "MBайт" << endl;
cout << "Гибкий диск: " << floppy << "MBайт" << endl;
show_mother_board();
show_screen();
}
void main(void)
{
computer my_pc("Compaq", 212, 1.44, "SVGA", 16000000, 640, 480, 486, 66, 8);
my_pc.show_computer();
}
```

Задание 2

ПОСТРОЕНИЕ ИЕРАРХИИ КЛАССОВ

При использовании наследования в C++ для порождения одного класса из другого возможны ситуации, когда вы порождаете свой класс из класса, который уже, в свою очередь, является производным от некоторого базового класса. Например, предположим, вам необходимо использовать класс computer базовый для порождения класса workstation, как показано ниже:

```
class work_station : public computer
{
public:
work_station (char *operating_system, char *name, int hard_disk, float floppy, char *screen, long
colors, int x_res, int y_res,int processor, int speed, int RAM);
void show_work_station(void);
private:
char operating_system[64];
};
```

Конструктор класса workstation просто вызывает конструктор класса computer, который в свою очередь вызывает конструкторы классов computer_screen и mother_board:

```
work_station::work_station( char *operating_system, char *name, int hard_disk, float floppy, char
*screen, long colors, int x_res, int y_res, int processor, int speed, int RAM) : computer (name,
```

```
hard_disk, floppy, screen, colors, x_res, y_res, processor, speed, RAM)
{
strcpy(work_station::operating_system, operating_system);
}
```

В данном случае класс computer выступает в роли базового класса. Однако вы знаете, что класс computer был порожден из классов computer_screen и mother_board. В результате класс work_station наследует характеристики всех трех классов.
Добавить в программу COMPUTER. CPP класс work_station. Откомпилировать. Получить результаты.

Задание 3

Для всех вариантов необходимо создать две программы, которые иллюстрировали бы оба приведенных выше примера множественного наследования. Создать динамический массив объектов, инициализировать его. Организовать поиск элемента в массиве по заданному значению.

Вариант 1

a) Создать иерархии наследования: студент, сотрудник - практикант.
b) Создать иерархии наследования: человек - студент, сотрудник - практикант.

Вариант 2

a) Создать иерархии наследования: млекопитающее, пресмыкающееся – утконос.
b) Создать иерархии наследования: животное - млекопитающее, пресмыкающееся – утконос.

Вариант 3

a) Создать иерархии наследования: телевизор, цифровое устройство – монитор.
b) Создать иерархии наследования: электронное устройство - телевизор, цифровое устройство – монитор.

Вариант 4

a) Создать иерархии наследования: авторучка, карандаш – авторучка с грифелем.
b) Создать иерархии наследования: письменные принадлежности - авторучка, карандаш – авторучка с грифелем.

Вариант 5

a) Создать иерархии наследования: катер, мотоцикл – водный мотоцикл.
b) Создать иерархии наследования: транспорт - катер, мотоцикл – водный мотоцикл.

Вариант 6

a) Создать иерархии наследования: диван, кровать – диван-кровать.
b) Создать иерархии наследования: мебель - диван, кровать – диван-кровать.

Вариант 7

a) Создать иерархии наследования: воздушный транспорт, пассажирский транспорт - лайнер *Boing 747*
b) Создать иерархии наследования: транспорт - воздушный транспорт, пассажирский транспорт - лайнер *Boing 747*

Вариант 8

a) Создать иерархии наследования: удочка, телескоп – телескопическая удочка.
b) Создать иерархии наследования: предмет - удочка, телескоп – телескопическая удочка.

Вариант 9

a) Создать иерархии наследования: бумага, собственность – акции.
b) Создать иерархии наследования: предмет - бумага, собственность – акции.

Вариант 10

a) Создать иерархии наследования: легковой автомобиль, грузовой автомобиль – внедорожник.
b) Создать иерархии наследования: автомобиль - легковой автомобиль, грузовой автомобиль – внедорожник.

Вариант 11

a) Создать иерархии наследования: книга, тетрадь – записная книжки.
b) Создать иерархии наследования: бумага - книга, тетрадь – записная книжки.

Вариант 12

a) Создать иерархии наследования: самолет, корабль - водный самолет.
b) Создать иерархии наследования: транспорт - самолет, корабль - водный самолет.

Контрольные вопросы:

1. Каковы преимущества множественного наследования?
2. Каковы сложности реализации множественного наследования?
3. Классифицируйте проблемы связанные с множественным наследованием?
4. Как решаются проблемы связанные с множественным наследованием?
5. Зачем необходимо виртуальное наследование?
6. Как реализовано виртуальное наследование?
7. Как работают конструкторы при множественном и виртуальном наследовании?

Содержание отчёта:

- ✓ Постановка задачи.
- ✓ Текст программы.
- ✓ Комментарии к ведённым обозначениям и описание используемых функций.
- ✓ Результаты работы программы.
- ✓ Ответы на контрольные вопросы

ПРАКТИЧЕСКАЯ РАБОТА №8. Полиморфизм. Виртуальные функции

Цели работы:

– исследование полиморфизма;
– изучение принципов позднего связывания;
– изучение виртуальных функций;
– полиморфизм *ad-hoc*;
– реализация виртуальных функций;
– изучение абстрактных классов.

Основные понятия

Слово полиморфизм греческого происхождения и приблизительно переводится как "*много форм*" (*poly*— много, *morphos*— форма). Слово *morphos*имеет отношение к греческому богу сна Морфею (Morphus), который мог являться спящим людям в любой форме, в какой только пожелает.

В жизни полиморфные виды — это те, которые характеризуются наличием различных форм или характеристик. В химии полиморфные соединения могут кристаллизоваться, по крайней мере, в двух различных формах (например, углерод имеет две кристаллические формы — графита и алмаза). С другой стороны, инженер ИТ воспринимается в первую очередь как человек, а потом как инженер (принцип подстановки).

В языках программирования полиморфный объект — это сущность (переменная, аргумент функции), хранящая, во время выполнения программы, значения различных типов. Полиморфные функции — это те функции, которые имеют полиморфные аргументы.

ВС++ полиморфизм — естественное следствие:

- отношения *"быть экземпляром"*;
- механизма пересылки сообщений;
- наследования;
- принципа подстановки.

Одно из важнейших достоинств объектно-ориентированного подхода состоит в возможности комбинирования этих средств. В результате получается богатый набор технических приемов совместного и многократного использования кода.

Полиморфная переменная многолика: она содержит значения, относящиеся к различным типам данных. Полиморфные переменные реализуют принцип подстановки. Другими словами, хотя для такой переменной имеется ожидаемый тип данных, фактический тип может быть подтипом ожидаемого типа. (Нужно вспомнить понятие подтипа и подкласса). В С++ полиморфные переменные существуют только как указатели и ссылки.

Полиморфные функции — это одна из наиболее мощных объектно-ориентированных техник программирования. Они позволяют единожды писать код на высоком уровне абстрагирования и затем применять его в конкретной ситуации. Обычно программист выполняет подгонку кода с помощью посылки дополнительных сообщений получателю, использующему метод. Эти дополнительные сообщения часто не связаны с классом на уровне абстракции полиморфного метода. Они являются виртуальными методами, которые определяются для классов более низкого уровня.

Когда настоящей переменной (то есть не указателю и не ссылке) присваивается значение типа подкласса, то динамический класс значения вынужденно приводится так, чтобы совпадать со статическим типом переменной.

Однако при использовании указателей или ссылок значение сохраняет свой динамический тип.

Позднее связывание

Под поздним связыванием нужно понимать механизм позволяющий определять динамический тип во время выполнения программы, а не во время компиляции. Похожий механизм это дескрипторы файлов, так как файлы открываются во время выполнения программы, а не во время компиляции. Данный механизм – основа полиморфизма, так как реализуют виртуальные функции.

Виртуальные функции

Позднее связывание решает проблему, однако, оно не имеет непосредственного представления в языке. По этому, для указания его необходимости используются виртуальные функции, которые записываются с использованием зарезервированного слова *virtual*. Виртуальные функции отличаются от обычных только способом вызова. Однако их использование имеет смысл лишь при работе с указателями либо ссылками.

Приведемпример:

```
#include<iostream.h>
class Animal{
public:
    void Say(){ cout<<"!!!\n";}
};
class Dog: public Animal{
public:
    void Say(){ cout<<"GAV\n";}
};
class Cat: public Animal{
public:
    void Say(){ cout<<"MIAU\n";}
};
voidFunSay(Animal a){
a.Say();
}
void main(){
 Animal a;
 Dog    d;
 Cat    c;
FunSay(a);
FunSay(d);
FunSay(c);
}
```

В данном примере глобальная функция не является полиморфной, так как переменная передается по значению, что приводит к потере специфических характеристик производного класса, кроме того, функция *Say* сама по себе не виртуальна. Следовательно, для правильной работы этой программы нужны следующие изменения: функция Say должна быть определена как виртуальная, а параметр глобальной функции должен быть определен как указатель либо ссылка.

Полиморфизм *ad-hoc*

Приводящий в замешательство аспект переопределения методов в языке C++ — это разница между переопределением виртуального и невиртуального методов.

Еще большее замешательство возникает, если программист пытается переопределить виртуальную функцию в подклассе, но при этом указывает (возможно, по ошибке) другой тип аргументов. Например, родительский класс содержит описание:

virtualvoiddisplay (char *, int);

Подкласс пытается переопределить метод:

virtualvoiddisplay (char *, short);

Поскольку списки аргументов различаются, то второе определение не распознается как переопределение, а как перегрузку. Поэтому, например, при вызове в форме родительского типа будет выбираться первый метод, а не второй. Такие ошибки чрезвычайно трудноуловимы, поскольку обе формы записи допустимы, и надеяться на диагностику компилятора не приходится.

Некоторых авторы склонны считать перегрузку, а так же шаблоны полиморфизмом, так как частично подпадают под определение многоформенности. Однако нужно вспомнить, что полиморфизм реализуется поздним связыванием, в то время как проблемы возникающие при использовании перегрузки и шаблонов решаются при компиляции. По этому эти конструкции языка иногда называются полиморфизмом *ad-hoc*.

Абстрактные классы

Отложенный метод (иногда называемый абстрактным методом, а в C++ — чисто виртуальным методом) может рассматриваться как обобщение переопределения. В обоих случаях поведение родительского класса изменяется для потомка. Для отложенного метода, однако, поведение просто не определено. Любая полезная деятельность задается в дочернем классе.

```
classShape{
public:
  ...
  virtualvoiddraw() = 0;
  ...
};
```

Компилятор не разрешает пользователю создавать экземпляр класса, который содержит чисто виртуальные методы, по этому эти классы называются абстрактными. Подклассы должны эти методы переопределять. Переопределение чисто виртуального метода должно произойти при описании его в потомках, для которых создаются реальные объекты.

Задания

Разобрать и откомпилировать примеры

Пример 1

Рассмотрим пример иерархии классов, описывающих неких животных. Для упрощения примера ограничимся в описании каждого животного его кличкой и издаваемым животным звуком. Ну, а основной возможностью программы будет вывод на экран списка кличек животных и представления издаваемых ими звуков.

```
#include <iostream>
#include <string.h>
```

```
using namespace std;
//абстрактный базовый класс
class Animal
{	public:
		char Title[20]; //кличка животного
		//простой конструктор
		Animal(char *t){
			strcpy(Title,t);
		}
		//чисто виртуальная функция
		virtual void speak()=0;
};

//класс лягушка
class Frog: public Animal
{
	public:
		Frog(char *Title): Animal(Title){};
		virtual void speak(){
			cout<<Title<<" say "<<"\'kwa-kwa\'\n";
		}
};

//класс собака
class Dog: public Animal
{
	public:
		Dog(char *Title): Animal(Title){};
		virtual void speak(){
			cout<<Title<<" say "<<"\'gav-gav\'\n";
		}
};

//класс кошка
class Cat: public Animal
{
	public:
		Cat(char *Title): Animal(Title){};
		virtual void speak(){
			cout<<Title<<" say "<<"\'myau-myau\'\n";
		}
};

//класс лев
class Lion: public Cat
{
	public:
		Lion(char *Title): Cat(Title) {};
		/*virtual void speak(){
			cout<<Title<<" say "<<"\'rrr-rrr\'\n";
```

```
		}*/

		/*virtual int speak(){
			cout<<Title<<" say "<<"\'rrr-rrr\'\n";
			return 0;
		}*/

		virtual void speak(int When){
			cout<<Title<<" say "<<"\'rrr-rrr\'\n";
		}
};

void main ()
{
	// объявим массив указателей на базовый класс Animal
	// и сразу его заполним указателями, создавая объекты
	// списокживотных
	Animal *animals[4] = {new Dog("Bob"), new Cat("Murka"),
new Frog("Vasya"),
new Lion("King")};

	for(int k=0; k<4; k++)
		animals[k]->speak();
}
```

Пример 2

В этой программе создается исходный базовый класс area, в котором сохраняются две размерности фигуры. В нем также объявляется виртуальная функция getarea(), которая, при ее подмене в производном классе, возвращает площадь фигуры, вид которой задается в производном классе. В этом случае определение функции getarea() внутри базового класса задает интерфейс. Конкретная реализация остается тем классам, которые наследуют класс area.

В этом примере рассчитывается площадь треугольника и прямоугольника.

```
// Использование виртуальной функции для определения интерфейса

#include <iostream>
using namespace std;
class area {
	double dim1, dim2; // размерыфигуры
	public:
	voidsetarea (double d1, double d2)
	{
	dim1 = d1;
	dim2 = d2;
	}
	voidgetdim(double &d1, double &d2)
	{
	d1 = dim1;
	d2 = dim2;
	}
	virtual double getarea ( )
	{
	cout<< "Вам необходимо подменить эту функцию\n";
```

```
return 0.0;
}
};
class rectangle: public area {
public:
doublegetarea ( )
{
double d1, d2;
getdim(d1, d2) ;
return d1 * d2;
}
};
class triangle: public area {
public:
doublegetarea ( )
{
double d1, d2;
getdim(d1, d2);
return 0.5 * d1 * d2;
}
};
int main ( )
{
area *p;
rectangle r;
triangle t;
r. setarea (3.3, 4.5);
t. setarea (4.0, 5.0);
p = &r;
cout<< "Площадь прямоугольника: " <<p->getarea() « '\n';
p= &t;
cout<< "Площадь треугольника: " <<
p->getarea() << '\n';
return 0;
}
```

Индивидуальное задание

Вариант 1

Создать абстрактный базовый класс *Worker*с виртуальной функцией начисления зарплаты. Создать производные классы *StateWorker, HourlyWorker* и *CommissionWorker,* в которых данная функция переопределена. В функции *main* определить массив указателей на абстрактный класс, которым присваиваются адреса объектов производных классов.

Вариант 2

Создать абстрактный базовый класс *Figure* с виртуальной функцией - площадь. Создать производные классы *Square, Circle, Triangle, Trapeze* в которых данная функция переопределена. В функции main определить массив указателей на абстрактный класс, которым присваиваются адреса различных объектов. Площадь трапеции: $S=(a+b)h/2$.

Вариант 3

Создать абстрактный базовый класс прогрессия с виртуальной функцией - сумма прогрессии. Создать производные классы: арифметическая прогрессия и геометрическая прогрессия. Каждый класс имеет два поля типа double. Первое - первый член прогрессии, второе (double) - постоянная разность (для арифметической) и постоянное отношение (для геометрической). Определить функцию вычисления суммы, где параметром является количество элементов прогрессии.

Арифметическая прогрессия $a_j=a_0+jd$, $j=0,1,2,\ldots$
Сумма арифметической прогрессии: $s_n=(n+1)(a_0+a_n)/2$
Геометрическая прогрессия: $a_j=a_0r^j$, $j=0,1,2,\ldots$
Сумма геометрической прогрессии: $s_n=(a_0-a_nr)/(1-r)$

Вариант 4

Создать абстрактный класс *Mammal* – млекопитающие с виртуальной функцией - описание. Определить производные классы - *Animal* и *Human*. Для животных определить производные классы *Dog* - собака и *Cow* – корова, в которых функция переопределяется.

Вариант 5

Создать абстрактный базовый класс *Lines*с виртуальной функцией *f(x)*. Создать производные классы *StraightLine, Ellipse, hyperbola* в которых данная функция переопределена. В функции *main* определить массив указателей на абстрактный класс, которым присваиваются адреса различных объектов. Уравнение прямой: $y=ax+b$, эллипса: $x^2/a^2+y^2/b^2=1$, гиперболы: $x^2/a^2-y^2/b^2=1$

Вариант 6

Создать абстрактный базовый класс *Figure* с виртуальной функцией - периметр. Создать производные классы *Rectangle, Circle, Triangle, Rhomb* в которых данная функция переопределена. В функции main определить массив указателей на абстрактный класс, которым присваиваются адреса различных объектов.

Вариант 7

Создать абстрактный базовый класс *Container*с виртуальными функциями вставки и извлечения. Создать производные классы *Stack*и *Queue,* в которых данные функция определены. В функции *main* определить массив указателей на абстрактный класс, которым присваиваются адреса объектов производных классов.

Вариант 8

Создать абстрактный базовый класс *Figure* с виртуальной функцией - площадь поверхности. Создать производные классы параллелепипед, тетраэдр, шар в которых данная функция переопределена.
Площадь поверхности параллелепипеда: $S=6xy$.
Площадь поверхности шара: $S=4\pi r^2$.
Площадь поверхности тетраэдра: $S=a^2\sqrt{3}$

Вариант 9

Создать абстрактный базовый класс *Number*с виртуальной функцией - норма. Создать производные классы *Complex, Vector* из 10 элементов, *Matrix*2 на 2, в которых данная функция переопределена. В функции *main* определить массив указателей на абстрактный класс, которым присваиваются адреса различных объектов.

Вариант 10

Создать абстрактный базовый класс *Учебное заведение* с виртуальной функцией - описания. Создать производные классы *дошкольное у.з., среднее у.з.* и *высшее у.з.* в которых данная функция переопределена. В функции *main* определить массив указателей на абстрактный класс, которым присваиваются адреса различных объектов.

Вариант 11

Создать абстрактный базовый класс уравнение с виртуальной функцией - корни уравнения. Создать производные классы линейное уравнение и квадратное уравнение, в которых данная функция переопределена.

Вариант 12

Создать абстрактный базовый класс *Figure*с виртуальной функцией - объем. Создать производные классы параллелепипед, пирамида, тетраэдр и шар, в которых данная функция переопределена. В функции *main* определить массив указателей на абстрактный класс, которым присваиваются адреса различных объектов.
Объем параллелепипеда - *V=xyz*(*x,y,z – стороны).*
Объем пирамиды: *V=xyh* (*x,y,* - стороны, *h - высота).*
Объем тетраэдра: V= a^3 2/12.

Объем шара: *V*=4 r^3/3.

Контрольные вопросы:

1. Какие механизмы подходят под определение полиморфизм?
2. И обычные и виртуальные функции могут быть переопределены. В чем различия?
3. Как физически реализованы виртуальные функции?
4. Что обозначает позднее связывание? Приведите примеры.
5. Каковы условия реализации полиморфизма?
6. Какие переменные полиморфны?
7. Что значит полиморфизм *ad-hoc*?
8. Чем отличаются абстрактные классы от обычных?
9. Как определяется чисто виртуальная функция?

Практическая работа №9. Файловый ввод/вывод.

Цель: научиться использовать различные функции и параметры для ввода/вывода информации, как в текстовом формате, так и в двоичном формате.

Подготовительная часть:
Файловый ввод/вывод. Двоичный файловый ввода/вывода. Манипуляторы ввода/вывода.

Содержание отчёта:
- ✓ Постановка задачи.
- ✓ Текст программы.
- ✓ Комментарии к ведённым обозначениям и описание используемых функций.
- ✓ Результаты работы программы.

Задания:

Задание №1

Создайте манипулятор вывода **sethex()**, который осуществляет вывод в шестнадцатеричной системе счисления и устанавливает флаги **uppercase** и **showbase**. Кроме того, создайте манипулятор вывода **reset()**, который отменяет изменения, сделанные манипулятором **sethex().**

Задание №2

Создайте манипулятор ввода **skipcbar(),** который, поочередно, то считывает, то пропускает каждые десять символов потока ввода.

Задание №3

Напишите программу, в которой копируется текстовый файл и при этом пробелы превращаются в символы |. Для контроля конца файла для ввода используйте функцию **eof().**

Задание №4

Напишите программу для подсчета числа слов в файле. Считайте, что словом является все, имеющее с двух сторон пробелы.

Контрольные вопросы:

1. Укажите особенности файлового ввод/вывод.
2. С помощью каких функций можно открыть и закрыть файл? Укажите их синтаксис.
3. Каково назначение режимов открытия файла? Перечислить существующие режимы.
4. Дайте характеристику режимам открытия файла.
5. Укажите особенности неформатируемого файлого ввода/вывода.
6. Дайте характеристику функциям считывания в неформатируемом вводе/выводе.
7. Дайте характеристику функциям записи в неформатируемом вводе/выводе.

Практическая работа №10. Неформатируемый двоичный ввод/вывод.

Цель: научиться использовать специальные функции, для организации считывания и записи информации в файл в двоичном формате.

Подготовительная часть:
Файловый ввод/вывод. Двоичный файловый ввода/вывода. Манипуляторы ввода/вывода.

Содержание отчёта:
- ✓ Постановка задачи.
- ✓ Текст программы.
- ✓ Комментарии к ведённым обозначениям и описание используемых функций.
- ✓ Результаты работы программы.

Задания:

Задание №1

Напишите программу для копирования текстового файла. Эта программа должна подсчитывать число копируемых символов и выводить на экран полученный результат. Используйте функции **get(), put(), read()** и/или **write()** в случае необходимости.

Задание №2

Напишите программу для подсчета числа слов в файле. Для простоты считайте, что словом является все, имеющее с двух сторон пробелы. Используйте функции **get(), put(), read()** и/или **write()** в случае необходимости.

Задание №3

Дан следующий класс. Напишите программу для вывода содержимого класса в файл. Для этой цели создайте пользовательскую функцию вывода.

```
class account {
        int custnum;
        char name[80];
        double balance;
public:
        account(int c, char *n, double b)
        {
        custnum = c;
        strcpy(name, n);
        balance = b;
        }
        // здесь нужна пользовательская функция вывода
};
```

Контрольные вопросы:

8. Дайте характеристику неформатируемому файловому вводу/выводу.
9. С помощью каких функций можно записать информацию в файл? Укажите их синтаксис.
10. С помощью каких функций можно считать информацию в файл? Укажите их синтаксис.
11. Как осуществляется контроль в системе ввода/вывода?
12. Укажите особенности пользовательского ввода/вывода в файлы.

Практическая работа №11 Список.

Цель: формирование навыков создания списочных структур, применения операций над списками при решении задач обработки списков.

Подготовительная часть:

1. Двусвязный список. Операции над двусвязным списком.
2. СД Многосвязный список. Обработка многосвязного списка. Примеры.

Методические рекомендации:

При формировании списков значения элементов задавать произвольно, каждый этап алгоритма выполнять в виде отдельной процедуры. Предусмотреть вывод на печать всех промежуточных результатов работы.

Задание 1.

Отладить программу, демонстрирующую работу со списками и выполняющую основные операции над односвязным списком.

```
#include <iostream>
using namespace std;

struct Element
{
  // Данные
  char data;
  // Адрес следующего элемента списка
  Element * Next;
};

// Односвязный список
class List
{
  // Адрес головного элемента списка
  Element * Head;
  // Адрес головного элемента списка
  Element * Tail;
  // Количество элементов списка
  int Count;

public:
  // Конструктор
  List();
  // Деструктор
  ~List();

  // Добавление элемента в список
  // (Новый элемент становится последним)
  void Add(char data);

  // Удаление элемента списка
```

```
  // (Удаляется головной элемент)
  void Del();
  // Удаление всего списка
  void DelAll();

  // Распечатка содержимого списка
  // (Распечатка начинается с головного элемента)
  void Print();

  // Получение количества элементов, находящихся в списке
  int GetCount();
};

List::List()
{
  // Изначально список пуст
  Head = Tail = NULL;
  Count = 0;
}

List::~List()
{
  // Вызов функции удаления
  DelAll();
}

int List::GetCount()
{
  // Возвращаем количество элементов
  return Count;
}

void List::Add(char data)
{
  // создание нового элемента
  Element * temp = new Element;

  // заполнение данными
  temp->data = data;
  // следующий элемент отсутствует
  temp->Next = NULL;
  // новый элемент становится последним элементом списка
  // если он не первый добавленный
  if(Head!=NULL){
                        Tail->Next=temp;
                        Tail = temp;
  }
  // новый элемент становится единственным
  // если он первый добавленный
  else{
              Head=Tail=temp;
  }
```

```
}

void List::Del()
{
  // запоминаем адрес головного элемента
  Element * temp = Head;
  // перебрасываем голову на следующий элемент
  Head = Head->Next;
  // удаляем бывший головной элемент
  delete temp;
}

void List::DelAll()
{
  // Пока еще есть элементы
  while(Head != 0)
    // Удаляем элементы по одному
    Del();
}

void List::Print()
{
  // запоминаем адрес головного элемента
  Element * temp = Head;
  // Пока еще есть элементы
  while(temp != 0)
  {
    // Выводим данные
    cout << temp->data << " ";
    // Переходим на следующий элемент
    temp = temp->Next;
  }

  cout << "\n\n";
}

// Тестовый пример
void main()
{
  // Создаем объект класса List
  List lst;

  // Тестовая строка
  char s[] = "Hello, World !!!\n";
  // Выводим строку
  cout << s << "\n\n";
  // Определяем длину строки
  int len = strlen(s);
  // Загоняем строку в список
  for(int i = 0; i < len; i++)
    lst.Add(s[i]);
  // Распечатываем содержимое списка
```

```
  lst.Print();
  // Удаляем три элемента списка
  lst.Del();
  lst.Del();
  lst.Del();
  //Распечатываем содержимое списка
  lst.Print();
     }
```

Отладить программу, демонстрирующую работу со списками и выполняющую основные операции над двусвязным списком.

```
#include <iostream>
using namespace std;

struct Elem
{
  int data; // данные
  Elem * next, * prev;
};

class List
{
  // Голова, хвост
  Elem * Head, * Tail;
  // Количество элементов
  int Count;

public:

  // Конструктор
  List();
  // Конструктор копирования
  List(const List&);
  // Деструктор
  ~List();

  // Получить количество
  int GetCount();
  // Получить элемент списка
  Elem* GetElem(int);

  // Удалить весь список
  void DelAll();
  // Удаление элемента, если параметр не указывается,
  // то функция его запрашивает
  void Del(int pos = 0);
  // Вставка элемента, если параметр не указывается,
  // то функция его запрашивает
  void Insert(int pos = 0);

  // Добавление в конец списка
  void AddTail(int n);
```

```
  // Добавление в начало списка
  void AddHead(int n);

  // Печать списка
  void Print();
  // Печать определенного элемента
  void Print(int pos);

  List& operator = (const List&);
  // сложение двух списков (дописывание)
  List operator + (const List&);

  // сравнение по элементам
  bool operator == (const List&);
  bool operator != (const List&);
  bool operator <= (const List&);
  bool operator >= (const List&);
  bool operator < (const List&);
  bool operator > (const List&);

  // переворачивание списка
  List operator - ();
};

List::List()
{
  // Изначально список пуст
  Head = Tail = NULL;
  Count = 0;
}

List::List(const List & L)
{
  Head = Tail = NULL;
  Count = 0;

  // Голова списка, из которого копируем
  Elem * temp = L.Head;
  // Пока не конец списка
  while(temp != 0)
  {
    // Передираем данные
    AddTail(temp->data);
    temp = temp->next;
  }
}

List::~List()
{
  // Удаляем все элементы
  DelAll();
```

```
}

void List::AddHead(int n)
{
  // новый элемент
  Elem * temp = new Elem;

  // Предыдущего нет
  temp->prev = 0;
  // Заполняем данные
  temp->data = n;
  // Следующий - бывшая голова
  temp->next = Head;

  // Если элементы есть?
  if(Head != 0)
    Head->prev = temp;

  // Если элемент первый, то он одновременно и голова и хвост
  if(Count == 0)
    Head = Tail = temp;
  else
    // иначе новый элемент - головной
    Head = temp;

  Count++;
}

void List::AddTail(int n)
{
  // Создаем новый элемент
  Elem * temp = new Elem;
  // Следующего нет
  temp->next = 0;
  // Заполняем данные
  temp->data = n;
  // Предыдущий - бывший хвост
  temp->prev = Tail;

  // Если элементы есть?
  if(Tail != 0)
    Tail->next = temp;

  // Если элемент первый, то он одновременно и голова и хвост
  if(Count == 0)
    Head = Tail = temp;
  else
    // иначе новый элемент - хвостовой
  Tail = temp;

  Count++;
}
```

```
void List::Insert(int pos)
{
  // если параметр отсутствует или равен 0, то запрашиваем его
  if(pos == 0)
  {
    cout << "Input position: ";
    cin >> pos;
  }

 // Позиция от 1 до Count?
  if(pos < 1 || pos > Count + 1)
  {
    // Неверная позиция
    cout << "Incorrect position !!!\n";
    return;
  }

  // Если вставка в конец списка
  if(pos == Count + 1)
  {
    // Вставляемые данные
    int data;
    cout << "Input new number: ";
    cin >> data;
    // Добавление в конец списка
    AddTail(data);
    return;
  }
  else if(pos == 1)
  {
    // Вставляемые данные
    int data;
    cout << "Input new number: ";
    cin >> data;
    // Добавление в начало списка
    AddHead(data);
    return;
  }

  // Счетчик
  int i = 1;

  // Отсчитываем от головы n - 1 элементов
  Elem * Ins = Head;

  while(i < pos)
  {
    // Доходим до элемента,
    // перед которым вставляемся
    Ins = Ins->next;
    i++;
```

```
    }

    // Доходим до элемента,
    // который предшествует
    Elem * PrevIns = Ins->prev;

    // Создаем новый элемент
    Elem * temp = new Elem;

    // Вводим данные
    cout << "Input new number: ";
    cin >> temp->data;

    // настройка связей
    if(PrevIns != 0 && Count != 1)
      PrevIns->next = temp;

    temp->next = Ins;
    temp->prev = PrevIns;
    Ins->prev = temp;

    Count++;
}

void List::Del(int pos)
{
    // если параметр отсутствует или равен 0, то запрашиваем его
    if(pos == 0)
    {
       cout << "Input position: ";
       cin >> pos;
    }
    // Позиция от 1 до Count?
    if(pos < 1 || pos > Count)
    {
      // Неверная позиция
      cout << "Incorrect position !!!\n";
      return;
    }

    // Счетчик
    int i = 1;

    Elem * Del = Head;

    while(i < pos)
    {
      // Доходим до элемента,
      // который удаляется
      Del = Del->next;
      i++;
    }
```

```
  // Доходим до элемента,
  // который предшествует удаляемому
  Elem * PrevDel = Del->prev;
  // Доходим до элемента, который следует за удаляемым
  Elem * AfterDel = Del->next;

  // Если удаляем не голову
  if(PrevDel != 0 && Count != 1)
    PrevDel->next = AfterDel;
  // Если удаляем не хвост
   if(AfterDel != 0 && Count != 1)
    AfterDel->prev = PrevDel;

  // Удаляются крайние?
  if(pos == 1)
     Head = AfterDel;
  if(pos == Count)
     Tail = PrevDel;

  // Удаление элемента
  delete Del;

  Count--;
}

void List::Print(int pos)
{
  // Позиция от 1 до Count?
  if(pos < 1 || pos > Count)
  {
    // Неверная позиция
    cout << "Incorrect position !!!\n";
    return;
  }

  Elem * temp;

  // Определяем с какой стороны
  // быстрее двигаться
  if(pos <= Count / 2)
  {
    // Отсчет с головы
    temp = Head;
    int i = 1;

    while(i < pos)
    {
      // Двигаемся до нужного элемента
      temp = temp->next;
      i++;
    }
```

```
  }
  else
  {
     // Отсчет с хвоста
    temp = Tail;
    int i = 1;

    while(i <= Count - pos)
    {
      // Двигаемся до нужного элемента
      temp = temp->prev;
      i++;
    }
  }
  // Вывод элемента
  cout << pos << " element: ";
  cout << temp->data << endl;
}

void List::Print()
{
  // Если в списке присутствуют элементы, то пробегаем по нему
  // и печатаем элементы, начиная с головного
  if(Count != 0)
  {
    Elem * temp = Head;
    cout << "( ";
    while(temp->next != 0)
    {
       cout << temp->data << ", ";
       temp = temp->next;
    }

    cout << temp->data << " )\n";
  }
}

void List::DelAll()
{
  // Пока остаются элементы, удаляем по одному с головы
  while(Count != 0)
    Del(1);
}

int List::GetCount()
{
   return Count;
}

Elem * List::GetElem(int pos)
{
  Elem *temp = Head;
```

```
  // Позиция от 1 до Count?
  if(pos < 1 || pos > Count)
  {
    // Неверная позиция
    cout << "Incorrect position !!!\n";
    return 0;
  }

  int i = 1;
  // Ищем нужный нам элемент
  while(i < pos && temp != 0)
  {
    temp = temp->next;
    i++;
  }

  if(temp == 0)
    return 0;
  else
    return temp;
}

List & List::operator = (const List & L)
{
   // Проверка присваивания элемента "самому себе"
   if(this == &L)
     return *this;

  // удаление старого списка
  this->~List(); // DelAll();

  Elem * temp = L.Head;

  // Копируем элементы
  while(temp != 0)
  {
    AddTail(temp->data);
    temp = temp->next;
  }

  return *this;
}
// сложение двух списков
List List::operator + (const List& L)
{
  // Заносим во временный список элементы первого списка
  List Result(*this);
  // List Result = *this;

  Elem * temp = L.Head;
```

```
  // Добавляем во временный список элементы второго списка
  while(temp != 0)
  {
    Result.AddTail(temp->data);
    temp = temp->next;
  }

  return Result;
}

bool List::operator == (const List& L)
{
  // Сравнение по количеству
  if(Count != L.Count)
    return false;

  Elem *t1, *t2;

  t1 = Head;
  t2 = L.Head;

  // Сравнение по содержанию
  while(t1 != 0)
  {
    // Сверяем данные, которые
    // находятся на одинаковых позициях
    if(t1->data != t2->data)
      return false;

    t1 = t1->next;
    t2 = t2->next;
  }

  return true;
}

bool List::operator != (const List& L)
{
  // Используем предыдущую функцию сравнения
  return !(*this == L);
}

bool List::operator >= (const List& L)
{
  // Сравнение по количеству
  if(Count > L.Count)
    return true;
  // Сравнение по содержанию
  if(*this == L)
    return true;

  return false;
```

```
}

bool List::operator <= (const List& L)
{
  // Сравнение по количеству
  if(Count < L.Count)
    return true;
  // Сравнение по содержанию
  if(*this == L)
     return true;

  return false;
}

bool List::operator > (const List& L)
{
  if(Count > L.Count)
    return true;

  return false;
}

bool List::operator < (const List& L)
{
  if(Count < L.Count)
     return true;

  return false;
}

// переворот
List List::operator - ()
{
  List Result;

  Elem * temp = Head;
  // Копируем элементы списка, начиная с головного,
  // в свой путем добавления элементов в голову,
  // таким образом временный список Result будет содержать
  // элементы в обратном порядке
  while(temp != 0)
  {
     Result.AddHead(temp->data);
     temp = temp->next;
  }

  return Result;
}

// Тестовый пример
void main()
{
```

```
List L;

const int n = 10;
int a[n] = {0,1,2,3,4,5,6,7,8,9};

// Добавляем элементы, стоящие на четных индексах, в голову,
// на нечетных - в хвост
for(int i = 0; i < n; i++)
  if(i % 2 == 0)
    L.AddHead(a[i]);
  else
    L.AddTail(a[i]);

// Распечатка списка
cout << "List L:\n";
L.Print();

cout << endl;

// Вставка элемента в список
L.Insert();
// Распечатка списка
cout << "List L:\n";
L.Print();

// Распечатка 2-го и 8-го элементов списка
L.Print(2);
L.Print(8);

List T;

// Копируем список
T = L;
// Распечатка копии
cout << "List T:\n";
T.Print();

// Складываем два списка (первый в перевернутом состоянии)
cout << "List Sum:\n";
List Sum = -L + T;
// Распечатка списка
Sum.Print();
   }
```

Задание 2.

Задание 2а.

Создайте подзаголовочный файл с именем SPIS.H, содержащий функции для работы со списком, демонстрирующий основные операции для работы со списком. Сохраните его как c/c++ Header File. Скопируйте библиотеку в папку C:\Program Files\Microsoft Visual Studio\VC98\Include. Текст библиотеки SPIS.H:

```
#include < iostream.h >
#include < conio.h >
#include < stdlib.h >
#include < time.h >
typedef  long  BT;
struct Zveno{
BT Inf;
Zveno *Next; };

Zveno *V_Nachalo(Zveno *First, BT X)
{           Zveno *Vsp;
Vsp = (Zveno *) malloc(sizeof(Zveno));
Vsp->Inf=X;
Vsp->Next=First;
First=Vsp;
return First;}

Zveno *Iz_Nachala(Zveno *First)
{           Zveno *Vsp;
Vsp=First->Next;
free(First);
return Vsp;}

Zveno *V_Spisok(Zveno *Pred, BT X)
{           Zveno *Vsp;
Vsp = (Zveno *) malloc(sizeof(Zveno));
Vsp->Inf=X;
Vsp->Next=Pred->Next;
Pred->Next=Vsp;
return Vsp;}

BT Iz_Spiska(Zveno *Pred)
{           BT X;     Zveno *Vsp;
Vsp=Pred->Next;
Pred->Next=Pred->Next->Next;
X=Vsp->Inf;         free(Vsp);
return X;}

void Print(Zveno *First)
{           Zveno *Vsp;
Vsp=First;
while (Vsp)
  {cout << Vsp->Inf << ' ';
   Vsp=Vsp->Next;}
cout << "\n";}

int Pust(Zveno *First)
{                       return !First;}

Zveno *Ochistka(Zveno *First)
{
```

```
  while (!Pust(First))
   First=Iz_Nachala(First);
   return First;}
```

Задание 2 б

Составить программу, которая на основе заданного списка формирует два других, помещая в первый из них положительные, а во второй — отрицательные элементы исходного списка.

При реализации алгоритма будем использовать подпрограммы разработанного модуля SPIS.H. Это существенно облегчает решение задачи.

Отладить программу.

```
#include <SPIS.H>

void main()
{Zveno *S1, *S2, *S3, *V1, *V2, *V3;
 BT a; int i, n;
 clrscr();
 srand(time(NULL));
 S1=NULL;
 // создаём первый элемент
 a=-100+rand()%201;
 S1=V_Nachalo(S1, a);
 n=1+rand()%20;
 // формируем список произвольной длины и выводим на печать
 V1=S1;
 for (i=2; i<=n; i++)
 {
   a=-100+rand()%201;
   V1=V_Spisok(V1, a);
 }
 Print(S1);
 V1 = S1;  S2 = NULL; S3 = NULL;
   while (V1)
            {if (V1->Inf > 0)
                 if (!S2)
                          {S2=V_Nachalo(S2, V1->Inf); V2 = S2;}
                  else {V_Spisok(V2, V1->Inf); V2 = V2->Next;};
             if (V1->Inf < 0)
                if (!S3)
                          {S3=V_Nachalo(S3, V1->Inf); V3 = S3;}
                else {V_Spisok(V3, V1->Inf); V3 = V3->Next;};
             V1= V1->Next;}
  cout << "Результирующий список из положительных элементов: \n";
  Print(S2);
  cout << "Результирующий список из отрицательных элементов: \n";
  Print(S3);
  S1=Ochistka(S1); S2=Ochistka(S2); S3=Ochistka(S3);
}
```

Для чтения русскоязычных символов подключите библиотеку russian.h.

Задание 3.

1. Составить функцию нахождения среднего арифметического элементов непустого списка L. Используя данную функцию, найти максимальное среднее арифметическое в списках K, M, N.
2. Составить функцию проверки упорядоченности символьных элементов списка L по алфавиту. Используя данную функцию, проанализировать элементы списков M, N, K.
3. Составить функцию, подсчитывающую количество слов списка, которые начинаются и оканчиваются одной и той же литерой. Используя данную функцию, найти сумму числа слов, начинающихся и оканчивающихся одной и той же литерой в списках M, K, L.
4. Составить функцию, которая помещает в начало списка L количество четных элементов, а в конец списка - количество нечетных элементов. С использованием данной процедуры преобразовать списки M, N и K.
5. Составить функцию, проверяющую на равенство значения элементов списков L1 и L2 и подсчитывающую количество одинаковых элементов в них. Используя функцию, проанализировать пары списков M1и M2, N1 и N2.
6. Составить функцию, определяющую вхождение списка L1 в список L2 и наоборот. Если один из списков длиннее, удалить лишние элементы из его начала. Используя функцию, проанализировать пары списков M1и M2, N1 и N2.
7. Составить функцию, определяющую порядковый номер наибольшего элемента последовательности натуральных чисел. Используя данную функцию, проанализировать последовательности натуральных чисел M и N.
8. Составить функцию вставки элемента E после каждого элемента списка, превышающего некоторое значение P. Подсчитать количество вставленных элементов.
9. В списке натуральных чисел переставить элементы по следующему правилу: если текущий элемент больше некоторого числа P, то поместить следующий за ним элемент в конец цепочки; если текущий элемент меньше или равен числу P, перенести в начало цепочки текущий элемент (первый оставить без изменения).
10. Построить список L1 - копию списка L, расположив элементы в обратном порядке (первый элемент списка L - последний элемент списка L1). Заменить элементы списка L, имеющие четные значения, на элементы спискаL1, имеющие нечетные значения.
11. Построить список L, упорядочив его по возрастанию, из двух неупорядоченных списков L1 и L2.
12. Определить, входит ли элемент E в список L, подсчитать количество вхождений данного элемента в список. Вставить первый элемент цепочки после каждого вхождения E в список.
13. Построить список L, упорядочив его по убыванию, из четных элементов L1 и нечетных элементов L2.

Задание 3.

1. Сформировать список L из элементов, которые входят одновременно в списки L1 и L2. Дописать в начало элементы, которые входят в L1, но не входят в L2, а в конец - элементы, которые входят в L2, но не входят в L1
2. Сформировать список L, включив в него положительные элементы списка L1 и отрицательные элементы списка L2. Список L отсортировать в порядке возрастания абсолютных значений элементов.
3. Определить, является ли список L пустым, если список не пуст, поменять местами первый и последний элементы списка и найти среднее арифметическое значений

элементов.

4. Сформировать списки L1 и L2 из списка L по следующему правилу: в L1 поместить четные положительные элементы списка L, в L2 - нечетные отрицательные элементы списка L. Подсчитать количество компонентов в списках L1 и L2.
5. Сформировать списки L1 и L2 из списка L по следующему правилу: в список L1 занести порядковые номера положительных компонентов, а в список L2 - отрицательных, считая от начала списка L. В начало списка L1 и конец списка L2 добавить порядковые номера нулевых компонентов списка L.
6. Дописать в список L после первого вхождения элемента E список L1 и удалить из списка L все оставшиеся элементы E, если таковые имеются.
7. Дописать после каждого вхождения в список L компонента E элемент, который представляет собой среднее арифметическое положительных элементов списка L.
8. Определить максимальные элементы списков L1 и L2. Поменять местами найденные максимальные элементы в списках. Вывести порядковые номера найденных элементов, считая от начала списка.
9. Дописать в конец списка L1 максимальный четный элемент списка L2, а в начало списка L2 - минимальный нечетный элемент списка L1.
10. Вставить после максимального элемента цепочки L1 компонент, равный среднему арифметическому компонентов списка L2; перед минимальным элементом списка L1 - компонент, равный произведению компонентов списка L2.
11. Сформировать цепочку L из минимальных и максимальных элементов цепочек L1 и L2, поместив минимальные элементы в начало, максимальные - в конец, а среднее арифметическое компонентов L1 и L2 - и середину цепочки L.
12. Заменить минимальный и максимальный элемент списка L компонентом, равным среднему арифметическому компонентов списка L, минимальный элемент дописать в начало цепочки, максимальный - в конец.
13. Переместить отрицательные элементы списка L в его начало, положительные - в конец. В случае равных по абсолютной величине элементов первый заменить нулём. Подсчитать количество положительных, отрицательных и равных элементов.

Контрольные вопросы:

1. Каковы особенности объявления данных динамической структуры?
2. Как определить, является ли данный элемент первым в цепочке данных?
3. Почему динамическим переменным не дают имен?
4. Как определить, что список компонентов является пустым?
5. В чем различие между статическими и динамическими переменными?
6. Какую информацию содержит ссылочная переменная?
7. Какой процедурой создается указанная переменная?
8. Можно ли сформировать указатель, ссылающийся на статическую переменную?
9. Что выполняет операция разыменования?
10. В каких случаях указатель может находиться в неопределенном состоянии?
11. В чем различие между состоянием NULL и неопределенным состоянием?
12. Какие действия выполняют процедуры New и Delete?
13. Что требуется для создания связанных динамических структур данных?
14. В чем состоит особенность описания типов для создания динамических структур данных?

Содержание отчёта:

1. Тема, цель работы. Теоретическая часть.

2. Постановка задачи.
3. Блок-схема алгоритма. Пояснения.
4. Комментированный листинг. Описание входных, выходных данных.
5. Скриншот работы программы. Электронный вариант работы программы.
6. Ответы на контрольные вопросы.

Практическая работа №12 Стек.

Цель: формировать практические умения и навыки описания структуры данных таких, как стек, решения задач его обработки.

Подготовительная часть

Структура данных типа «Дек».

Содержание отчёта:

1. Тема, цель работы.
2. Теоретическая часть.
3. Постановка задачи.
4. Блок-схема с пояснениями.
5. Текст программы.
6. Скриншот работы программы.
7. Электронный вариант (код программы)

ВАРИАНТЫ ЗАДАНИЙ

Задание 1

Задание 1

Реализуем эти операции, используя разработанный ранее модуль для однонаправленных списков (см. материал лаб.раб №1).

```
// файл STACK.H

#include "SPIS.CPP"

Zveno *V_Stack(Zveno *Versh, BT X)
{
 return V_Nachalo(Versh, X);
}

Zveno *Iz_Stack(Zveno *Versh)
{
 return Iz_Nachala(Versh);
}

int SPust(Zveno *Versh)
{
        return !Versh;
}

BT V_Vershine(Zveno *Versh)
{
        return Versh->Inf;
```

```
}

Zveno *Chistka(Zveno *Versh)
{
while (!Pust(Versh)) Versh=Iz_Stack(Versh);
                        return Versh;
}
```

Используя разработанные здесь библиотеки, решить задачу.

Пример. Написать программу, которая вычисляет как целое число значение выражений (без переменных), записанных (без ошибок) в постфиксной форме в текстовом файле. Каждая строка файла содержит ровно одно выражение.

Алгоритм решения. Выражение просматривается слева направо. Если встречается число, то его значение (как целое) заносится в стек, а если встречается знак операции, то из стека извлекаются два последних элемента (это операнды данной операции), над ними выполняется операция и ее результат записывается в стек. В конце в стеке остается только одно число — значение всего выражения.

```
// ST2.CPP

#include "STACK.H"
#include < string.h >
#include < stdio.h >

void main(void)
{
  char S[255];
  FILE *T;
  int I;
  BT X, Y;
  Zveno *NS;
  clrscr();
  cout << "Введите имя файла: ";
  cin >> S;
  T=fopen(S, "r");
  NS = NULL;
  while (!feof(T))
  {
    fgets(S, 255, T);
    I = 0;
    while (I <= strlen(S)-1)
    {
      if (S[I]>='0'&&S[I]<='9')
      {
        X=0;
        while(S[I]>='0'&&S[I]<='9')
        {
          X=X*10+(S[I]-'0');
```

```
        I++;
       }
       NS=V_Stack(NS, X);
     }
     else
       if (S[I]=='+'||S[I]=='-'||S[I]=='/'||S[I]=='*')
       {
         X=V_Vershine(NS);
         NS=Iz_Stack(NS);
         Y=V_Vershine(NS);
         NS=Iz_Stack(NS);
         switch (S[I])
         {
           case '+' : X += Y; break;
           case '-' : X = Y - X; break;
           case '*' : X *= Y; break;
           case '/' : X = Y / X; break;
         }
         NS=V_Stack(NS, X);
       }
     I++;
   }
   X=V_Vershine(NS);
   NS=Iz_Stack(NS);
   cout << S << " => " << X << "\n";
  }
}
```

Задание 2.

Отладить и откомпилировать следующую программу, включающую основные операции над стеком.

Пример. Программная реализация стека

Рассмотрим, как представлять стек с помощью односвязного списка и как в этом случае реализуются операции со стеком. Операция записи в стек означает создание нового элемента списка, присоединение его к списку в качестве первого его элемента и занесение в него заданного значения. Операция чтения из стека реализуется как чтение значения первого элемента списка с удалением этого элемента из списка, при этом обязателен контроль состояния стека. Состояние стека определяется значением его указателя: стек пуст, если указатель имеет значение NULL, в противном случае - стек не пуст.

```
#include <iostream>
#include <string.h>
#include <time.h>
using namespace std;

class Stack
{
        // Нижняя и верхняя границы стека
        enum {EMPTY = -1, FULL = 20};

        // Массив для хранения данных
```

```
	char st[FULL + 1];

	// Указатель на вершину стека
	int top;

public:
	// Конструктор
	Stack();

	// Добавление элемента
	void Push(char c);

	// Извлечение элемента
	char Pop();

	// Очистка стека
	void Clear();

	// Проверка существования элементов в стеке
	bool IsEmpty();
	// Проверка на переполнение стека
	bool IsFull();

	// Количество элементов в стеке
	int GetCount();
};


Stack::Stack()
{
	// Изначально стек пуст
	top = EMPTY;
}

void Stack::Clear()
{
	// Эффективная "очистка" стека
	// (данные в массиве все еще существуют,
	// но функции класса, ориентированные на работу с вершиной стека,
	// будут их игнорировать)
	top = EMPTY;
}

bool Stack::IsEmpty()
{
	// Пуст?
	return top == EMPTY;
}

bool Stack::IsFull()
{
	// Полон?
```

```
	return top == FULL;
}

int Stack::GetCount()
{
	// Количество присутствующих в стеке элементов
	return top + 1;
}

void Stack::Push(char c)
{
	// Если в стеке есть место, то увеличиваем указатель
	// на вершину стека и вставляем новый элемент
	if(!IsFull())
		st[++top] = c;
}

char Stack::Pop()
{
	// Если в стеке есть элементы, то возвращаем верхний и
	// уменьшаем указатель на вершину стека
	if(!IsEmpty())
		return st[top--];
	else // Если в стеке элементов нет
		return 0;
}

void main()
{
	srand(time(0));
	Stack ST;
	char c;
	// пока стек не заполнится
	while(!ST.IsFull()){
      c=rand()%4+2;
		ST.Push(c);
	}
	// пока стек не освободится
	while(c=ST.Pop()){
		cout<<c<<" ";
	}
	cout<<"\n\n";

   }
```

Задание 3

В задании2 структура *«стек»* (stack) моделируется цепочкой связанных узлов-записей типа TNode. Поле Next последнего элемента цепочки равно NULL. *Вершиной стека* (top) считается первый элемент цепочки. Для доступа к стеку используется указатель на его вершину (для пустого стека данный указатель полагается равным NULL). *Значением* элемента стека считается значение его поля Data.

1. Дано число *D* и указатель *P1* на вершину непустого стека. Добавить элемент со значением *D* в стек и вывести адрес *P2* новой вершины стека.
2. Дано число N (>0) и набор из .N чисел. Создать стек, содержащий исходные числа (последнее число будет вершиной стека), и вывести указатель на его вершину.
3. Дан указатель *P1* на вершину непустого стека. Извлечь из стека первый (верхний) элемент и вывести его значение D, а также адрес *P2* новой вершины стека. Если после извлечения элемента стек окажется пустым, то положить P2 = NULL. После извлечения элемента из стека освободить память, занимаемую этим элементом.
4. Дан указатель P1 на вершину стека, содержащего не менее десяти элементов. Извлечь из стека первые девять элементов и вывести их значения. Вывести также адрес новой вершины стека. После извлечения элементов из стека освобождать память, которую они занимали.
5. Дан указатель *P1* на вершину стека (если стек пуст, то *P1* = NULL). Извлечь из стека все элементы и вывести их значения. Вывести также количество извлеченных элементов *N* (для пустого стека вывести 0). После извлечения элементов из стека освобождать память, которую они занимали.
6. Даны указатели *P1* и *P2* на вершины двух непустых стеков. Переместить все элементы из первого стека во второй (в результате элементы первого стека будут располагаться во втором стеке в порядке, обратном исходному) и вывести адрес новой вершины второго стека. Операции выделения и освобождения памяти не использовать.
7. Даны указатели *P1* и *P2* на вершины двух непустых стеков. Перемещать элементы из первого стека во второй, пока значение вершины первого стека не станет четным (перемещенные элементы первого стека будут располагаться во втором стеке в порядке, обратном исходному). Если в первом стеке нет элементов с четными значениями, то переместить из первого стека во второй все элементы. Вывести адреса новых вершин первого и второго стека (если первый стек окажется пустым, то вывести для него константу NULL). Операции выделения и освобождения памяти не использовать.
8. Дан указатель *P1* на вершину непустого стека. Создать два новых стека, переместив в первый из них все элементы исходного стека с четными значениями, а во второй — с нечетными (элементы в новых стеках будут располагаться в порядке, обратном исходному; один из этих стеков может оказаться пустым). Вывести адреса вершин полученных стеков (для пустого стека вывести NULL). Операции выделения и освобождения памяти не использовать.
9. Дан указатель *P1* на вершину стека (если стек пуст, то *P1* = NULL). Также дано число *N* (>0) и набор из *N* чисел. Описать тип TStack -запись с одним полем Top типа PNode (поле указывает на *вершину* стека) — и процедуру Push (S, *D),* которая добавляет в стек *S* новый элемент со значением *D (S* — входной и выходной параметр типа TStack*, D* -входной параметр целого типа). С помощью процедуры Push добавить в исходный стек данный набор чисел (последнее число будет вершиной стека) и вывести адрес новой вершины стека.
10. Дан указатель *P1* на вершину стека, содержащего не менее пяти элементов. Используя тип TStack (см. задание 11), описать функцию Pop(S) целого типа, которая извлекает из стека *S* первый (верхний) элемент, возвращает его значение и освобождает память, которую занимал извлеченный элемент *(S* — входной и выходной параметр типа TStack). С помощью функции Pop извлечь из исходного стека пять элементов и вывести их значения. Вывести также указатель на новую вершину стека (если результирующий стек окажется пустым, то этот указатель должен быть равен NULL).
11. Дан указатель *P1* на вершину стека. Используя тип TStack (см. задание 11), описать функции StackIsEmpty(S) логического типа (возвращает TRUE, если стек *S* пуст, и FALSE в противном случае) и Peek(S) целого типа (возвращает значение вершины непустого стека *S*, не удаляя ее из стека). В обеих функциях переменная *S* является входным параметром типа TStack. С помощью этих функций, а также функции Pop из задания 12, извлечь из

исходного стека пять элементов (или все содержащиеся в нем элементы, если их менее пяти) и вывести их значения. Вывести также значение функции StackIsEmpty для результирующего стека и, если результирующий стек не является пустым, значение и адрес его новой вершины.

Задание 3 (дополнительно)

1. В составе целочисленного стека имеется единственный нулевой элемент, разделяющий его на две части. Переставить эти части местами, изменив лишь информационные связи (указатели) между элементами стека.
2. Заданы два стека *S1* и *S2,* в которых записаны элементы двух разреженных целочисленных матриц. Каждая компонента стека содержит индексы *i, j* и значение $a_{i,j}$ элемента матрицы. Сформировать новый стек *S3,* записав в него элементы суммарной матрицы. Если при сложении элементов $a_{i,j}$ из *S1* и *S2* образуется нулевое значение, такой элемент в стек *S3* не записывать.
3. В текстовом файле записано несколько последовательностей целых чисел, каждая из которых заканчивается нулем. Сформировать типизированный файл, в котором числа каждой последовательности записаны в обратном порядке. В качестве буфера использовать стек, в котором временно размещать числа очередной последовательности.
4. Информационная часть стека содержит неупорядоченные целочисленные значения. Если среди элементов стека есть такие, что *Inf* [*i*] < *b* < *Inf* [*i*+1] (*Inf*[*i*] - информационная часть *i*-го элемента очереди, *b* - произвольное целое число), то вставить значение *b* между этими элементами.

Контрольные вопросы:

1. Определение СД типа «стек».
2. Совокупность операций, определяющих структуру данных типа «стек».
3. Дескриптор структуры данных типа «стек».
4. Области применения СД типа «стек».
5. Области применения СД типа «Дек».

Практическая работа №13 Очередь.

Цель: формировать практические умения и навыки описания структуры данных такой, как очередь, решения задач её обработки.

Подготовительная часть

1. Программная реализация кольцевой очереди.
2. Программная реализация очереди с приоритетами.

Содержание отчёта:

8. Тема, цель работы.
9. Теоретическая часть.
10. Постановка задачи.
11. Блок-схема с пояснениями.
12. Текст программы.
13. Скриншот работы программы.
14. Электронный вариант (код программы)

ВАРИАНТЫ ЗАДАНИЙ

Задание 1.

Отладить и откомпилировать следующие программы, включающие основные операции над стеком и очередью.

Пример 1. Программная реализация очереди

Операции записи в очередь и чтения из очереди реализуются аналогично операциям со стеком. Здесь отличие состоит в том, что добавление и удаление элемента производятся в разные концы очереди, при этом состояние очереди должны контролировать обе операции: если очередь была пуста, то операция записи корректирует и указатель на конец очереди, а если очередь состоит из одного элемента, то операция чтения должна корректировать и указатель на начало очереди (задавать значение NULL при списочном представлении очереди).

```
#include <iostream>
#include <string.h>
#include <time.h>
using namespace std;

class Queue
{
        // Очередь
        int * Wait;
        // Максимальный размер очереди
        int MaxQueueLength;
        // Текущий размер очереди
        int QueueLength;

public:
        // Конструктор
        Queue(int m);
```

```
	//Деструктор
	~Queue();

	// Добавление элемента
	void Add(int c);

	// Извлечение элемента
	int Extract();

	// Очистка очереди
	void Clear();

	// Проверка существования элементов в очереди
	bool IsEmpty();

	// Проверка на переполнение очереди
	bool IsFull();

	// Количество элементов в очереди
	int GetCount();

	//демонстрация очереди
	void Show();
};

void Queue::Show(){
	cout<<"\n-----------------------------------\n";
	//демонстрация очереди
	for(int i=0;i<QueueLength;i++){
		cout<<Wait[i]<<" ";
	}
	cout<<"\n-----------------------------------\n";
}

Queue::~Queue()
{
	//удаление очереди
	delete[]Wait;
}

Queue::Queue(int m)
{
	//получаем размер
	MaxQueueLength=m;
	//создаем очередь
	Wait=new int[MaxQueueLength];
	// Изначально очередь пуста
	QueueLength = 0;
}

void Queue::Clear()
```

```
{
	// Эффективная "очистка" очереди
	QueueLength = 0;
}

bool Queue::IsEmpty()
{
	// Пуст?
	return QueueLength == 0;
}

bool Queue::IsFull()
{
	// Полон?
	return QueueLength == MaxQueueLength;
}

int Queue::GetCount()
{
	// Количество присутствующих в стеке элементов
	return QueueLength;
}

void Queue::Add(int c)
{
	// Если в очереди есть свободное место, то увеличиваем количество
	// значений и вставляем новый элемент
	if(!IsFull())
		Wait[QueueLength++] = c;
}

int Queue::Extract()
{
	// Если в очереди есть элементы, то возвращаем тот,
	// который вошел первым и сдвигаем очередь
	if(!IsEmpty()){
		//запомнить первый
		int temp=Wait[0];

		//сдвинуть все элементы
		for(int i=1;i<QueueLength;i++)
			Wait[i-1]=Wait[i];

		//уменьшить количество
		QueueLength--;

		//вернуть первый(нулевой)
		return temp;
	}

	else // Если в стеке элементов нет
		return -1;
```

```
}

void main()
{
	srand(time(0));

	//создание очереди
	Queue QU(25);

	//заполнение части элементов
	for(int i=0;i<5;i++){
		QU.Add(rand()%50);
	}
	//показ очереди
	QU.Show();

	//извлечение элемента
	QU.Extract();

	//показ очереди
	QU.Show();
}
```

Кольцевая очередь.

Кольцевая очередь очень похожа на простую очередь. Она тоже построенна на идеологии FIFO, напоминаем - *элемент, который добавили в очередь первым, первым ее и покинет*. Разница лишь в том, что элемент, который выходит из начала очереди, будет перемещён в её конец.

В качестве самого простого примера, можно привести известный вам с детства круговорот воды в природе, или трамваи, курсирующие по круговому маршруту.

Пример 2. Реализация - кольцевая очередь.

```
#include <iostream>
#include <string.h>
#include <time.h>
using namespace std;

class QueueRing
{
	// Очередь
	int * Wait;
	// Максимальный размер очереди
	int MaxQueueLength;
	// Текущий размер очереди
	int QueueLength;

public:
	// Конструктор
	QueueRing(int m);

	//Деструктор
	~QueueRing();
```

```
	// Добавление элемента
	void Add(int c);
	// Извлечение элемента
	bool Extract();

	// Очистка очереди
	void Clear();

	// Проверка существования элементов в очереди
	bool IsEmpty();

	// Проверка на переполнение очереди
	bool IsFull();

	// Количество элементов в очереди
	int GetCount();

	//демонстрация очереди
	void Show();
};

void QueueRing::Show(){
	cout<<"\n-------------------------------------\n";
	//демонстрация очереди
	for(int i=0;i<QueueLength;i++){
		cout<<Wait[i]<<" ";
	}
	cout<<"\n-------------------------------------\n";
}

QueueRing::~QueueRing()
{
	//удаление очереди
	delete[]Wait;
}

QueueRing::QueueRing(int m)
{
	//получаем размер
	MaxQueueLength=m;
	//создаем очередь
	Wait=new int[MaxQueueLength];
	// Изначально очередь пуста
	QueueLength = 0;
}

void QueueRing::Clear()
{
	// Эффективная "очистка" очереди
	QueueLength = 0;
}
```

```
bool QueueRing::IsEmpty()
{
	// Пуст?
	return QueueLength == 0;
}

bool QueueRing::IsFull()
{
	// Полон?
	return QueueLength == MaxQueueLength;
}

int QueueRing::GetCount()
{
	// Количество присутствующих в стеке элементов
	return QueueLength;
}

void QueueRing::Add(int c)
{
	// Если в очереди есть свободное место, то увеличиваем количество
	// значений и вставляем новый элемент
	if(!IsFull())
		Wait[QueueLength++] = c;
}

bool QueueRing::Extract()
{
	// Если в очереди есть элементы, то возвращаем тот,
	// который вошел первым и сдвигаем очередь
	if(!IsEmpty()){
		//запомнить первый
		int temp=Wait[0];

		//сдвинуть все элементы
		for(int i=1;i<QueueLength;i++)
			Wait[i-1]=Wait[i];

		//забрасываем первый "вытолкнутый элемент в конец"
		Wait[QueueLength-1]=temp;
		return 1;

	}
	else return 0;
}

void main()
{
	srand(time(0));

	//создание очереди
```

```
	QueueRing QUR(25);

	//заполнение части элементов
	for(int i=0;i<5;i++){
		QUR.Add(rand()%50);
	}
	//показ очереди
	QUR.Show();

	//извлечение элемента
	QUR.Extract();

	//показ очереди
	QUR.Show();
}
```

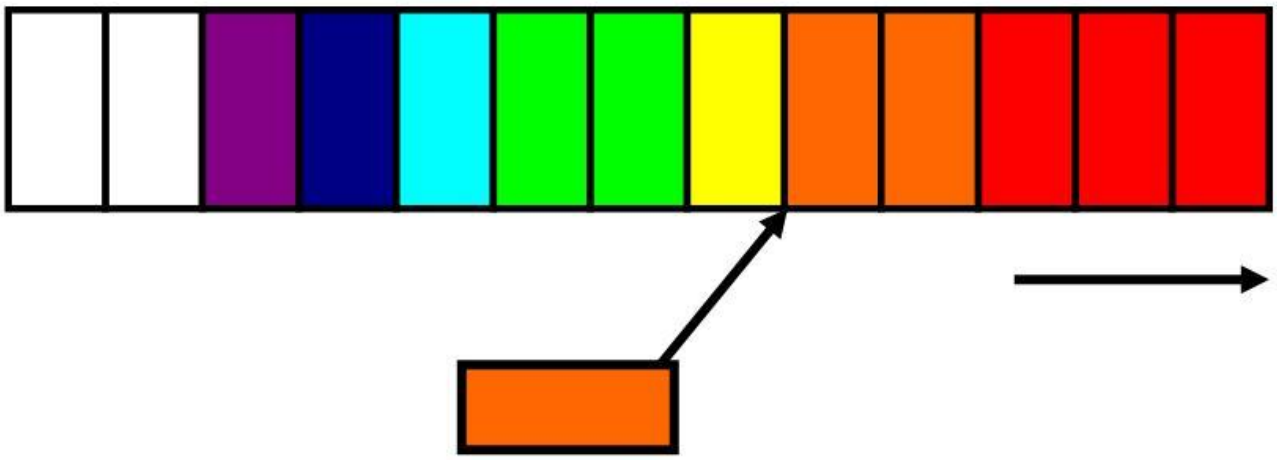
Очередь с приоритетом.

Мы уже познакомились с двумя типами очередей и они оказались достаточно простыми. Однако существует еще одна очередь - **очередь с приоритетом**.

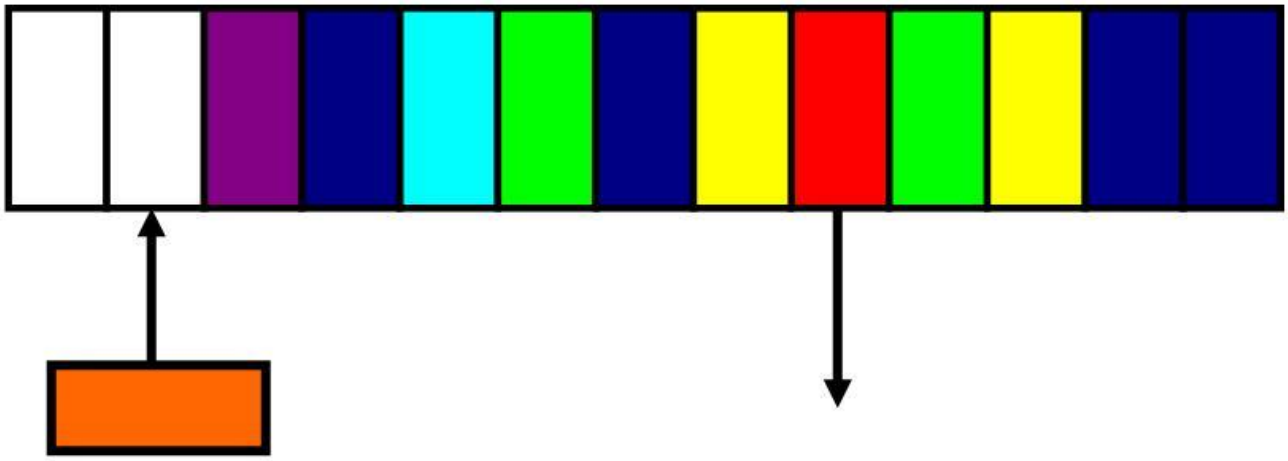
Дело в том, что часто необходимо создать очередь, где (извините за тавтологию) очередность выхода зависит от приоритетов элементов. В качестве приоритета может выступать какое-либо число, временная константа и т. п. В момент извлечения элемента будет выбран тот элемент, который обладает большим приоритетом.

Существует несколько видов приоритетных очередей:

1. Очередь с приоритетным включением - последовательность элементов очереди является строго упорядоченной. Другими словами, каждый элемент при попадании в очередь сразу же располагается согласно своего приоритета. А в момент исключения элемента просто извлекается элемент из начала.



2. Очереди с приоритетным исключением - элемент добавляется в конец очереди, а при извлечении осуществляется самого приоритетного элемента, который впоследствии удаляется из очереди.



Пример 3. Реализация - очередь с приоритетами.

```
#include <iostream>
#include <string.h>
#include <time.h>
using namespace std;

class QueuePriority
{
        // Очередь
        int * Wait;
        // Приоритет
        int * Pri;
        // Максимальный размер очереди
        int MaxQueueLength;
        // Текущий размер очереди
        int QueueLength;

public:
        // Конструктор
        QueuePriority(int m);

        //Деструктор
        ~QueuePriority();

        // Добавление элемента
        void Add(int c,int p);

        // Извлечение элемента
        int Extract();

        // Очистка очереди
        void Clear();

        // Проверка существования элементов в очереди
        bool IsEmpty();

        // Проверка на переполнение очереди
        bool IsFull();
```

```
	// Количество элементов в очереди
	int GetCount();

	//демонстрация очереди
	void Show();
};

void QueuePriority::Show(){
	cout<<"\n-----------------------------------\n";
	//демонстрация очереди
	for(int i=0;i<QueueLength;i++){
		cout<<Wait[i]<<" - "<<Pri[i]<<"\n\n";
	}
	cout<<"\n-----------------------------------\n";
}

QueuePriority::~QueuePriority()
{
	//удаление очереди
	delete[]Wait;
	delete[]Pri;
}

QueuePriority::QueuePriority(int m)
{
	//получаем размер
	MaxQueueLength=m;
	//создаем очередь
	Wait=new int[MaxQueueLength];
	Pri=new int[MaxQueueLength];
	// Изначально очередь пуста
	QueueLength = 0;
}

void QueuePriority::Clear()
{
	// Эффективная "очистка" очереди
	QueueLength = 0;
}

bool QueuePriority::IsEmpty()
{
	// Пуст?
	return QueueLength == 0;
}

bool QueuePriority::IsFull()
{
	// Полон?
	return QueueLength == MaxQueueLength;
}
```

```
int QueuePriority::GetCount()
{
	// Количество присутствующих в стеке элементов
	return QueueLength;
}

void QueuePriority::Add(int c,int p)
{
	// Если в очереди есть свободное место, то увеличиваем количество
	// значений и вставляем новый элемент
	if(!IsFull()){
		Wait[QueueLength] = c;
		Pri[QueueLength] = p;
		QueueLength++;
	}
}

int QueuePriority::Extract()
{
	// Если в очереди есть элементы, то возвращаем тот,
	// у которого наивысший приоритет и сдвигаем очередь
	if(!IsEmpty()){


		//пусть приоритетный элемент - нулевой
		int max_pri=Pri[0];
		//а приоритетный индекс = 0
		int pos_max_pri=0;

		//ищем приоритет
		for(int i=1;i<QueueLength;i++)
			//если встречен более приоритетный элемент
			if(max_pri<Pri[i]){
				max_pri=Pri[i];
				pos_max_pri=i;
			}

		//вытаскиваем приоритетный элемент
		int temp1=Wait[pos_max_pri];
		int temp2=Pri[pos_max_pri];

		//сдвинуть все элементы
		for(int i=pos_max_pri;i<QueueLength-1;i++){
			Wait[i]=Wait[i+1];
			Pri[i]=Pri[i+1];
		}
		//уменьшаем количество
		QueueLength--;
		// возврат извлеченного элемента
		return temp1;

	}
```

```
        else return -1;
}

void main()
{
        srand(time(0));

        //создание очереди
        QueuePriority QUP(25);

        //заполнение части элементов
        for(int i=0;i<5;i++){

                // значения от 0 до 99 (включительно)
                // и приоритет от 0 до 11 (включительно)
                QUP.Add(rand()%100,rand()%12);
        }
        //показ очереди
        QUP.Show();

        //извлечение элемента
        QUP.Extract();

        //показ очереди
        QUP.Show();
}
```

Задание 2

1. В задании 2 структура *«очередь»* (queue) моделируется цепочкой связанных узлов-записей типа TNode. Поле Next последнего элемента цепочки равно NULL. *Началом очереди* («головой», head) считается первый элемент цепочки, *концом* («хвостом», tail) — ее последний элемент. Для возможности быстрого добавления в конец очереди нового элемента удобно хранить, помимо указателя на начало очереди, также и указатель на ее конец. В случае пустой очереди указатели на ее начало и конец полагаются равными NULL. Как и для стека, *значением* элемента очереди считается значение его поля Data.
2. Дан набор из 10 чисел. Создать очередь, содержащую данные числа в указанном порядке (первое число будет размещаться в начале очереди, последнее — в конце), и вывести указатели *P1* и *P2* на начало и конец очереди.
3. Дан набор из 10 чисел. Создать две очереди: первая должна содержать числа из исходного набора с нечетными номерами (1,3,..., 9), а вторая — с четными (2, 4, ..., 10); порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе. Вывести указатели на начало и конец первой, а затем второй очереди.
4. Дан набор из 10 чисел. Создать две очереди: первая должна содержать все нечетные, а вторая — все четные числа из исходного набора (порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе). Вывести указатели на начало и конец первой, а затем второй очереди (одна из очередей может оказаться пустой; в этом случае вывести для нее две константы NULL**).**
5. Дано число *D* и указатели *P1* и *P2* на начало и конец очереди (если очередь является пустой, то *P1* = *P2* = NULL). Добавить элемент со значением *D* в конец очереди и вывести новые адреса начала и конца очереди.
6. Дано число *D* и указатели *P1* и *P2* на начало и конец очереди, содержащей не менее двух

элементов. Добавить элемент со значением *D* в конец очереди и извлечь из очереди первый (начальный) элемент. Вывести значение извлеченного элемента и новые адреса начала и конца очереди. После извлечения элемента из очереди освободить память, занимаемую этим элементом.

7. Дано число *N* (>0) и указатели *P1* и *P2* на начало и конец непустой очереди. Извлечь из очереди *N* начальных элементов и вывести их значения (если очередь содержит менее *N* элементов, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести NULL). После извлечения элементов из очереди освобождать память, которую они занимали.
8. Даны указатели *P1* и *P2* на начало и конец непустой очереди. Извлекать из очереди элементы, пока значение начального элемента очереди не станет четным, и выводить значения извлеченных элементов (если очередь не содержит элементов с четными значениями, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести NULL). После извлечения элементов из очереди освобождать память, которую они занимали.
9. Даны две очереди; адреса начала и конца первой равны *P1* и *P2,* а второй – *P3 и P4* (если очередь является пустой, то соответствующие адреса равны NULL). Переместить все элементы первой очереди (в порядке от начала к концу) в конец второй очереди и вывести новые адреса начала и конца второй очереди. Операции выделения и освобождения памяти не использовать.
10. Дано число *N* (>0) и две непустые очереди; адреса начала и конца первой равны *P1* и P2, а второй — *P3* и *P4.* Переместить *N* начальных элементов первой очереди в конец второй очереди. Если первая очередь содержит менее N элементов, то переместить из первой очереди во вторую все элементы. Вывести новые адреса начала и конца первой, а затем второй очереди (для пустой очереди дважды вывести NULL). Операции выделения и освобождения памяти не использовать.
11. Даны две непустые очереди; адреса начала и конца первой равны *P1* и P2, а второй — P3 и *P4* Перемещать элементы из начала первой очереди в конец второй, пока значение начального элемента первой очереди не станет четным (если первая очередь не содержит четных элементов, то переместить из первой очереди во вторую все элементы). Вывести новые адреса начала и конца первой, а затем второй очереди (для пустой очереди дважды вывести NULL). Операции выделения и освобождения памяти не использовать.
12. Даны две непустые очереди; адреса начала и конца первой равны *P1* и *P2,* а второй — *P3* и P4. Очереди содержат одинаковое количество элементов. Объединить очереди в одну, в которой элементы исходных очередей чередуются (начиная с первого элемента первой очереди). Вывести указатели на начало и конец полученной очереди. Операции выделения и освобождения памяти не использовать.
13. Даны две непустые очереди; адреса начала и конца первой равны P1 и P2, а второй — P3 и *P4.* Элементы каждой из очередей упорядочены по возрастанию (в направлении от начала очереди к концу). Объединить очереди в одну с сохранением упорядоченности элементов. Вывести указатели на начало и конец полученной очереди. Операции выделения и освобождения памяти не использовать, поля Data не изменять.
14. Даны указатели *P1* и P2 на начало и конец очереди (если очередь является пустой, то *P1* = P2 = NULL). Также дано число *N* (>0) и набор из *N* чисел. Описать тип TQueue — запись с двумя полями Head и Tail типа PNode (поля указывают на *начало* и *конец* очереди) — и процедуру Enqueue(Q, *D),* которая добавляет в конец очереди *Q* новый элемент со значением *D (Q* — входной и выходной параметр типа TQueue, *D* — входной параметр целого типа). С помощью процедуры Enqueue добавить в исходную очередь данный набор чисел и вывести новые адреса ее начала и конца.
15. Даны указатели *P1* и P2 на начало и конец очереди, содержащей не менее пяти элементов. Используя тип TQueue (см. задание 14), описать функцию Dequeue(Q) целого типа, которая извлекает из очереди первый (начальный) элемент, возвращает его значение и освобождает

память, занимаемую извлеченным элементом *(Q* — входной и выходной параметр типа TQueue). С помощью функции Dequeue извлечь из исходной очереди пять начальных элементов и вывести их значения. Вывести также адреса начала и конца результирующей очереди (если очередь окажется пустой, то эти адреса должны быть равны NULL).

16. Даны указатели *P1* и *P2* на начало и конец очереди. Используя тип TQueue (см. задание 14), описать функцию QueueIsEmpty(Q) логического типа, которая возвращает true, если очередь *Q* пуста, и FALSE в противном случае *(Q* — входной параметр типа TQueue). Используя эту функцию для проверки состояния очереди, а также функцию Dequeue из задания 15, извлечь из исходной очереди пять начальных элементов (или все содержащиеся в ней элементы, если их менее пяти) и вывести их значения. Вывести также значение функции QueueIsEmpty для полученной очереди и новые адреса ее начала и конца.

Задание 3 (Дополнительно)

5. В составе целочисленной очереди имеется единственный нулевой элемент, разделяющий ее на две части. Сформировать две новые очереди, записав в них соответственно первую и вторую части исходной очереди.
6. Элементы очереди - целые числа. Сформировать новую очередь, в которую переписать сначала все положительные числа в их исходном относительном порядке, а затем - отрицательные числа в обратном порядке. Нулевые числа в новую очередь не включать.
7. В циклической очереди последовательно записаны числа 1, 2, 3, ..., n . Задано также значение $m > 1$. Начиная отсчет от начала очереди и двигаясь по часовой стрелке, удалить m-ый элемент.

Примечание. В частном случае может быть $m > n$. В этом случае целесообразно исходную очередь временно сделать циклической.

8. В каждой строке текстового файла записано малыми буквами одно слово анализируемого текста. Длина слова не превышает 15 символов. При однократном чтении файла сформировать очередь, каждый элемент которой содержит две переменные: значение слова и количество его повторений в тексте. Слова в очереди должны быть расположены в алфавитном порядке. Группировку очереди не производить.
9. Элементы вещественного массива $X = (x_1, x_2, ..., x_n)$ содержат большое количество нулевых элементов. Ненулевые элементы x_i записаны в очередь в порядке увеличения их индексов. Компонента очереди содержит индекс элемента i и его значение x_i. Требуется элементу x_k присвоить новое значение b. Если элемент x_k отсутствует в очереди, его нужно в нее добавить.
10. Элементы вещественного массива $X = (x_1, x_2, ..., x_n)$ содержат большое количество нулевых элементов. Ненулевые элементы x_i записаны в очередь в порядке увеличения их индексов. Компонента очереди содержит индекс элемента i и его значение x_i. Требуется элемент x_k удалить из очереди, если он в нее был записан.
11. В текстовом файле записаны элементы полинома по убывающим степеням (значение коэффициента и значение степени). Элементы с нулевым коэффициентом в файл не включаются. Сформировать очередь, содержащую коэффициенты полинома (в том числе нулевые), после чего по схеме Горнера вычислить его значение, введя с клавиатуры аргумент полинома.
12. Каждый элемент очереди содержит два вещественных числа, определяющих координаты точки на плоскости. Найти среди элементов очереди точку, наиболее близкую к первой точке, и удалить ее из состава очереди.
13. Элементом очереди является строка с объявленной длиной 80 символов. В частном случае строка может быть пустой. Удалить из очереди все пустые строки.

14. Элементами очереди являются целочисленные переменные, упорядоченные в порядке возрастания или убывания. Первый элемент может повторяться несколько раз. Изменить очередь таким образом, чтобы в ее начале было ровно два одинаковых элемента.

Контрольные вопросы:

1. Определения СД типа «очередь».
2. Совокупность операций, определяющих структуру данных типа «очередь».
3. Дескриптор структуры данных типа «очередь».
4. Области применения СД типа «очередь».

Практическая работа №14 Бинарное дерево.

Цель: формировать практические умения и навыки описания и работы со структурой данных бинарное дерево, решения задач обработки двоичных деревьев.

Подготовительная часть

1. Представление бинарного дерева матрицей смежности.
2. Сбалансированное дерево.
3. Представление арифметического выражения с помощью бинарного дерева.

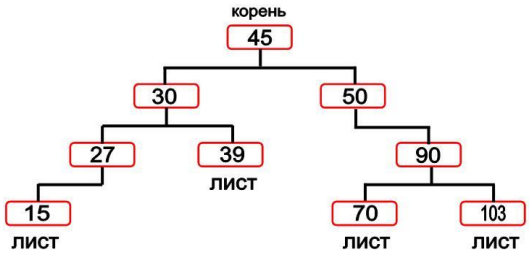
Содержание отчёта:

1. Тема, цель работы.
2. Теоретическая часть.
3. Постановка задачи.
4. Блок-схема с пояснениями.
5. Текст программы.
6. Скриншот работы программы.
7. Электронный вариант (код программы)

Теоретические сведения

Бинарное дерево.

Бинарное дерево (binary tree) - это упорядоченная древовидная динамическая структура. Каждый элемент (узел) дерева имеет не более двух элементов следующих за ним (потомков) и не более одного предыдущего (родителя). Рассмотрим схематичное изображение бинарного дерева:



Комментарии к изображению дерева. Терминология.

1. Самый главный принцип бинарного дерева заключается в том, что для каждого узла выполняется правило: **в левой ветке содержатся только те ключи, которые имеют значения, меньшие, чем значение данного узла. В правой же ветке содержатся ключи, имеющие значения, большие, чем значение данного узла.**
2. Каждый узел может иметь два, одного или ни одного потомка.
3. **Лист** - узел, не имеющий потомков.
4. Узел является родительским для своих потомков и дочерним для своего предка.
5. **Левый потомок** - дочерний узел слева от текущего узла.
6. **Правый потомок** - дочерний узел справа от текущего узла.
7. **Корень** - основной узел, не имеющий родителей.
8. Каждый узел состоит из четырех частей:
   - Значение.
   - Указатель на родителя.
   - Указатель на левого потомка.
   - Указатель на правого потомка.

Примечание: Бинарное дерево является рекурсивной структурой, поскольку каждое его поддерево само является бинарным деревом и, следовательно, каждый его узел в свою очередь является корнем самостоятельного дерева.

Организация работы с деревом.

При работе с деревьями обычно используются рекурсивные алгоритмы. Использование рекурсивных функций менее эффективно, поскольку многократный вызов функции расходует системные ресурсы. Тем не менее, в данном случае использование рекурсивных функций является оправданным, поскольку нерекурсивные функции для работы с деревьями гораздо сложнее и для написания, и для восприятия кода программы. Здесь мы приведем схематичные алгоритмы работы с деревом. А, осмысленный, практический пример смотрите в следующем разделе урока. Вот некоторые значения, которые мы будем использовать в схемах:

- x - вершина бинарного дерева
- left[x] - левое поддерево
- right[x] - правое поддерево
- key[x] - ключ
- p[x] - родитель вершины

Обход всего дерева.

Печать(x)
Начало

1. если x не равен NULL
2. тогда Печать(left[x])
3. напечатать key[x]
4. Печать(right[x])

Конец

Поиск значения.

Поиск(x,k)
Начало

1. Пока x не равен NULL и k не равно key[x]
2. Начало
3. если k меньше key[x]
4. тогда x равно left[x]
5. иначе x равно right[x]
6. Конец
7. Вернуть x

Конец

Нахождение минимума и максимума.

Минимум(x)
Начало

1. Пока left[x] не равен NULL
2. Начало
3. x=left[x]
4. Конец

5. Вернуть x

Конец

Максимум(x)

Начало

1. Пока right[x] не равен NULL
2. Начало
3. x=right[x]
4. Конец
5. Вернуть x

Конец

Получение следующего и предыдущего элементов.

ПолучитьСледующийЭлемент(x)

Начало

1. если right[x] не равен NULL, тогда вернуть Минимум(right[x])
2. y равно p[x]
3. пока y не равно NULL и x равно right[x]
4. Начало
5. x равно y
6. y равно p[y]
7. Вернуть y

Конец

ПолучитьПредыдущийЭлемент(x)

Начало

1. если left[x] не равен NULL, тогда вернуть Максимум(left[x])
2. y равно p[x]
3. пока y не равно NULL и x равно left[x]
4. Начало
5. x равно y
6. y равно p[y]
7. Вернуть y

Конец

Добавление значения.

Вставка(T,z)

Начало

1. y равно NULL
2. x равно root[T]
3. пока x не равно NULL
4. Начало
5. y равно x
6. если key[z] меньше key[x], тогда x равно left[x]
7. иначе x равно right[x]
8. Конец
9. p[z] равно y
10. если y равно NULL, тогда root[T] равно z
11. иначе если key[z] меньше key[y], тогда left[y] равно z
12. иначе right[y] равно z

Конец

Удаление значения.

Удаление(T,z)

Начало

1. если left[z] равно NULL или right[z]=NULL, тогда y равно z
2. иначе y равно ПолучитьСледующийЭлемент(z)

3. если left[y] не равно NULL, тогда x равно left[y]
4. иначе x равно right[y]
5. если x не равно NULL, тогда p[x] равно p[y]
6. если p[y] равно NULL, тогда root[T] равно x
7. иначе если y равно left[p[y]], тогда left[p[y]] равно x
8. иначе right[p[y]] равно x
9. если y не равен z, тогда key[z] равно key[y]
10. Копируем дополнительные данные связанные с y
11. Удалить y

Конец

Применение.

В заключение, отметим, что организация данных с помощью бинарных деревьев часто позволяет значительно сократить время поиска нужного элемента. Поиск элемента в линейных структурах данных обычно осуществляется путем последовательного перебора всех элементов, присутствующих в данной структуре. Поиск по дереву не требует перебора всех элементов, поэтому занимает значительно меньше времени. Максимальное число шагов при поиске по дереву равно высоте данного дерева, т.е. количеству уровней в иерархической структуре дерева.

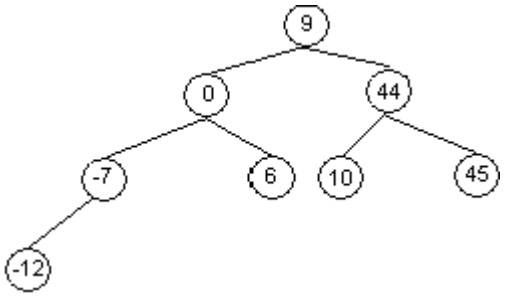
ВАРИАНТЫ ЗАДАНИЙ

Задание 1

Двоичное дерево поиска может быть либо пустым, либо оно обладает таким свойством, что корневой элемент имеет большее значение узла, чем любой элемент в левом поддереве, и меньшее или равное, чем элементы в правом поддереве. Указанное свойство называется *характеристическим свойством двоичного дерева* поиска и выполняется для любого узла такого дерева, включая корень. Далее будем рассматривать только двоичные деревья поиска. Такое название двоичные деревья поиска получили по той причине, что скорость

Пример. Для набора данных 9, 44, 0, –7, 10, 6, –12, 45 построить двоичное дерево поиска.

Согласно определению двоичного дерева поиска число 9 помещаем в корень, все значения, меньшие его — на левое поддерево, большие или равные — на правое. В каждом поддереве очередной элемент можно рассматривать как корень и действовать по тому же алгоритму. В итоге получаем



Выделим типовые операции над двоичными деревьями поиска:

- добавление элемента в дерево;
- удаление элемента из дерева;
- обход дерева (для печати элементов и т.д.);
- поиск в дереве.

Поскольку определение двоичного дерева рекурсивно, то все указанные типовые операции могут быть реализованы в виде рекурсивных подпрограмм (на практике именно такой вариант чаще всего и применяется). Отметим лишь, что использование рекурсии замедляет работу программы и расходует лишнюю память при её выполнении.

Пусть двоичное дерево поиска описывается следующим типом

```
        struct BinTree{
    BT inf;
    BinTree *L; BinTree *R;
          };
```

Покажем два варианта добавления элемента в дерево: итеративный и рекурсивный.

```
/* Итеративный вариант добавления элемента в дерево*/

BinTree* InsIteration(BinTree *T, BT x)
{ BinTree *vsp, *A;
  A = (BinTree *) malloc(sizeof(BinTree));
  A->inf=x; A->L=0; A->R=0;
  if (!T) T=A;
  else {vsp = T;
          while (vsp)
          {if (A->inf < vsp->inf)
             if (!vsp->L) {vsp->L=A; vsp=A->L;}
             else vsp=vsp->L;
           else
             if (!vsp->R) {vsp->R=A; vsp=A->R;}
             else vsp=vsp->R;
          }
          }
return T;
}

/* Рекурсивный вариант добавления элемента в дерево*/

BinTree* InsRec(BinTree *Tree, BT x)
{
  if (!Tree) {Tree = (BinTree *) malloc(sizeof(BinTree));
               Tree->inf=x; Tree->L=0; Tree->R=0;
              }
  else if (x < Tree->inf) Tree->L=InsRec(Tree->L, x);
      else Tree->R=InsRec(Tree->R, x);
  return Tree;
}
```

Существует несколько способов обхода (прохождения) всех узлов дерева. Три наиболее часто используемых из них называются *обход в прямом (префиксном) порядке*, *обход в обратном (постфиксном) порядке* и *обход во внутреннем порядке (или симметричный обход)*. Каждый из обходов реализуется с использованием рекурсии.

Ниже приведены подпрограммы печати элементов дерева с использованием обхода двоичного дерева поиска в обратном порядке.

```
void PrintTree(BinTree *T)
{
  if (T) {PrintTree(T->L); cout << T->inf<< " "; PrintTree(T->R);}
}
```

Реализуем функцию, возвращающую 1, если элемент присутствует в дереве, и 0 — в противном случае.

```
int Find(BinTree *Tree, BT x)
{ if (!Tree) return 0;
  else if (Tree->inf==x) return 1;
      else if (x < Tree->inf) return Find(Tree->L, x);
          else return Find(Tree->R, x);
}
```

По сравнению с предыдущими задача удаления узла из дерева реализуется несколько сложнее. Можно выделить два случая удаления элемента *x* (случай отсутствия элемента в дереве является вырожденным):

1) узел, содержащий элемент *x*, имеет степень не более 1 (степень узла — число поддеревьев, выходящих из этого узла);

2) узел, содержащий элемент *x*, имеет степень 2.

Случай 1 не представляет сложности. Предыдущий узел соединяется либо с единственным поддеревом удаляемого узла (если степень удаляемого узла равна 1), либо не будет иметь поддерева совсем (если степень узла равна 0).

Намного сложнее, если удаляемый узел имеет два поддерева. В этом случае нужно заменить удаляемый элемент самым правым элементом из его левого поддерева.

```
BinTree * Delete(BinTree *Tree, BT x)
{ BinTree* P, *v;
  if (!Tree) cout << "такого элемента в дереве нет!" << endl;
  else if (x < Tree->inf) Tree->L = Delete(Tree->L, x);
      else if (x > Tree-> inf) Tree->R = Delete(Tree->R, x);
              else {P = Tree;
                    if (!Tree->R) Tree = Tree->L; // случай 1
                    else if (!Tree->L) Tree = Tree->R; // случай 1
                        else // случай 2
                  { v = Tree->L;
                    if (v->R)
                    {
                                  while (v->R->R) v = v->R;
                                  Tree->inf = v->R->inf;
                                  P = v->R; v->R = v->R->L;
```

```
                    }
                    else
                    {
                     Tree->inf = v->inf;
                     P = v;
                     Tree->L=Tree->L->L;
                    }
                                }
                      free(P);
                     }
 return Tree;
}
```

Примечание. Если элемент повторяется в дереве несколько раз, то удаляется только первое его вхождение.

Задание 2.
Отладить и откомпилировать следующую программу, включающую работу с двоичным деревом.

```
Использования бинарного дерева для хранения результатов игр команд английского чемпионата по футболу.
#include <iostream>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

using namespace std;
struct Elem
{
   int  OwnerPoints;                    // Очки хозяина
   int  OppPoints;              // Очки соперника
   char Match[10];              // Счет
   char Name[20];               // Команда
   char Opponent[20];                   // Соперник

   Elem * left, * right, * parent;
};

class Tree
{
   // корень
   Elem * root;
public:
   Tree();
   ~Tree();
   // печать от указанного узла
   void Print(Elem * Node);
   // поиск от указанного узла
```

```
  Elem * Search(Elem * Node, char * key);
  // min от указанного узла
  Elem * Min(Elem * Node);
  // max от указанного узла
  Elem * Max(Elem * Node);
  // следующий для указанного узла
  Elem * Next(Elem * Node);
  // предыдущий для указанного узла
  Elem * Previous(Elem * Node);
  // вставка узла
  void Insert(Elem * z);
  // удаление ветки для указанного узла,
  // 0 - удаление всего дерева
  void Del(Elem * z = 0);
  // получить корень
  Elem * GetRoot();
};

Tree::Tree()
{
  root = NULL;
}

Tree::~Tree()
{
  Del();
}

// Рекурсивный обход дерева
void Tree::Print(Elem * Node)
{
  if(Node != 0)
  {
    Print(Node->left);
    cout << Node->Name
         << Node->Match
         << Node->Opponent
         << endl;
    Print(Node->right);
  }
}

Elem * Tree::Search(Elem * Node, char * k)
{
  // Пока есть узлы и ключи не совпадают
  while(Node != 0 && strcmp(k, Node->Name) != 0)
  {
    if(strcmp(k, Node->Name) < 0)
      Node = Node->left;
    else
      Node = Node->right;
  }
```

```
  return Node;
}

Elem * Tree::Min(Elem * Node)
{
  // Поиск самого "левого" узла
  if(Node != 0)
    while(Node->left != 0)
      Node = Node->left;

  return Node;
}

Elem * Tree::Max(Elem * Node)
{
  // Поиск самого "правого" узла
  if(Node != 0)
    while(Node->right != 0)
      Node = Node->right;

  return Node;
}

Elem * Tree::Next(Elem * Node)
{
  Elem * y = 0;
  if(Node != 0)
  {
    // если есть правый потомок
    if(Node->right != 0)
      return Min(Node->right);

    // родитель узла
    y = Node->parent;
    // если Node не корень и Node справа
    while(y != 0 && Node == y->right)
    {
      // Движемся вверх
      Node = y;
      y = y->parent;
    }
  }
  return y;
}

Elem * Tree::Previous(Elem * Node)
{
  Elem * y = 0;
  if(Node != 0)
  {
    // если есть левый потомок
    if(Node->left != 0)
```

```
        return Max(Node->left);

      // родитель узла
      y = Node->parent;
      // если Node не корень и Node слева
      while(y != 0 && Node == y->left)
      {
        // Движемся вверх
        Node = y;
        y = y->parent;
      }
    }
    return y;
}

Elem * Tree::GetRoot()
{
    return root;
}

void Tree::Insert(Elem * z)
{
    // потомков нет
    z->left = NULL;
    z->right = NULL;

    Elem * y = NULL;
    Elem * Node = root;

    // поиск места
    while(Node != 0)
    {
      // будущий родитель
      y = Node;
      if(strcmp(z->Name, Node->Name) < 0)
        Node = Node->left;
      else
        Node = Node->right;
    }
    // заполняем родителя
    z->parent = y;

    if(y == 0) // элемент первый (единственный)
      root = z;
    // чей ключ больше?
    else if(strcmp(z->Name, y->Name) < 0)
      y->left = z;
    else
      y->right = z;
}

void Tree::Del(Elem * z)
```

```
{
  // удаление куста
  if(z != 0)
  {
    Elem * Node, * y;

    // не 2 ребенка
    if(z->left == 0 || z->right == 0)
      y = z;
    else
      y = Next(z);

    if(y->left != 0)
      Node = y->left;
    else
      Node = y->right;

    if(Node != 0)
      Node->parent = y->parent;
    // Удаляется корневой узел?
    if(y->parent == 0)
      root = Node;
    else if(y == y->parent->left) // слева от родителя?
      y->parent->left = Node;
    else
      y->parent->right = Node;  // справа от родителя?

    if(y != z)
    {
      // Копирование данных узла
      strcpy(z->Name, y->Name);
      strcpy(z->Opponent, y->Opponent);
      strcpy(z->Match, y->Match);
      z->OppPoints = y->OppPoints;
      z->OwnerPoints = y->OwnerPoints;
    }

    delete y;
  }
  else // удаление всего дерева
    while(root != 0)
      Del(root);
}


// Турнирная таблица
Tree tournament;

void Game(char Commands[][20], int N)
{
  int i, j;
  int p1, p2; // Счет
```

```
  // Каждая команда играет с каждой по 2 раза - дома и в гостях
  int k;

  Elem * temp;
  for(k = 0; k < 2; k++)
    for(i = 0; i < N - 1; i++)
    {
      for(j = i + 1; j < N; j++)
      {
        temp = new Elem;
        if(k == 0)
        {
          // 1 игра
          strcpy(temp->Name, Commands[i]);
          strcpy(temp->Opponent, Commands[j]);
        }
        else
        {
          // 2 игра
          strcpy(temp->Name, Commands[j]);
          strcpy(temp->Opponent, Commands[i]);
        }

        p1 = rand() % 6;
        p2 = rand() % 6;

        if(p1 > p2)
        {
          temp->OwnerPoints = 3;
          temp->OppPoints = 0;
        }
        else if(p1 == p2)
        {
          temp->OwnerPoints = 1;
          temp->OppPoints = 1;
        }
        else
        {
          temp->OwnerPoints = 0;
          temp->OppPoints = 3;
        }

        // Запись счета
        sprintf(temp->Match, " %d : %d ", p1, p2);
        // Добавление записи
        tournament.Insert(temp);
      }
    }
}

void main()
```

```
{
  srand(time(0));

  const int N = 4;
  char Commands[4][20] =
  {
    "Arsenal",
    "Liverpool",
    "Lids United",
    "Manchester United"
  };

  // Игра
  Game(Commands, N);
  // Вывод результатов
  tournament.Print(tournament.GetRoot());
}
```

Задание 3

1. Реализуйте процедуру ввода двоичного дерева.
2. Подсчитайте число листьев (терминальных элементов) в заданном двоичном дереве.
3. Определите высоту заданного двоичного дерева.
4. В заданном двоичном дереве подсчитайте число элементов, равных максимальному.
5. Замените в заданном двоичном дереве все отрицательные элементы их абсолютными величинами.
6. Поменяйте местами максимальный и минимальный элементы заданного двоичного дерева, все элементы которого различны.
7. Пусть для дерева А была сделана копия – дерево В, и из дерева В были удалены некоторые из ветвей. Необходимо сформировать список указателей на удалённые ветви дерева А.
8. Проверьте, является ли заданное двоичное дерево сбалансированным.
9. Пусть задано дерево двоичного поиска. Реализуйте подпрограмму добавления элемента в дерево двоичного поиска.
10. Пусть задано дерево двоичного поиска. Реализуйте подпрограмму удаления элемента из дерева двоичного поиска.
11. Пусть задано дерево двоичного поиска. Реализуйте подпрограмму поиска элемента в дереве двоичного поиска.
12. Пусть задано дерево двоичного поиска. Реализуйте подпрограмму преобразования данного дерева в сбалансированное дерево двоичного поиска.
13. Пусть задано дерево двоичного поиска. Реализуйте подпрограмму подсчёта числа элементов, больших заданного.

Контрольные вопросы:

1. Перечислите составляющие дерева.
2. Дайте определение бинарного дерева.
3. Перечислите операции над деревьями.
4. Использование бинарных деревьев для поиска данных.
5. Красно-чёрное дерево. Основные свойства.
6. Понятие сбалансированного дерева. Виды сбалансированных деревьев.

Практическая работа №15 Шаблоны классов

Цель: закрепить умения и навыки по созданию функций-шаблонов и шаблонных классов; продемонстрировать возможности родовых функций и родовых классов для существующих типов данных.

Содержание отчёта:

- ✓ Постановка задачи.
- ✓ Текст программы.
- ✓ Комментарии к ведённым обозначениям и описание используемых функций.
- ✓ Результаты работы программы.

Задания:

Задание 1

Откомпилировать пример (использованием шаблонной функции).

Ниже приведен короткий пример, в котором создается функция-шаблон, имеющая два параметра. Эта функция меняет между собой величины значений этих параметров. Поскольку общий процесс обмена значениями между двумя переменными не зависит от их типа, то он естественным способом может быть реализован с помощью функции-шаблона.

```
// пример шаблона функции
#include <iostream.h>
// шаблон функции
template <class X>

void swap(X &a, X &b)
{
X temp;
temp = a;
a = b;
b = temp;
}
int main()
{
int i=10, j = 20;
float x=10.1, y= 23.3;
char a='x', b='z';
cout << "Original i, j: " << i << ' ' << j << endl;
cout << "Original x, y: " << x << ' ' << y << endl;
cout << "Original a, b: " << a << ' ' << b << endl;
swap(i, j); // обмен целых
swap(x, y); // обмен вещественных значений
swap(a, b); // обмен символов
cout << "Swapped i, j : " << i << ' ' << j << endl;
cout << "Swapped x, y: " << x << ' ' << y << endl;
cout << "Swapped a, b: " << a << ' ' << b << endl;
return 0;
}
```

Задание 2

Откомпилировать пример (использование шаблона класса)

```
// демонстрация класса-шаблона stack
#include <iostream.h>
const int SIZE = 100;
// создание класса-шаблона stack
template <class SType> class stack {
SType stck[SIZE];
int tos;
public:
stack();
~stack();
void push(SType i);
SType pop();
};
// функция-конструктор stack
template <class SType> stack<SType>::stack()
{
tos = 0;
cout << "Stack Initialized\n";
}
/* функция-деструктор stack
This function is not required. It is included for illustration only. */
template <class SType> stack<SType>::~stack()
{
cout << "Stack Destroyed\n";
}
// помещение объекта в стек
template <class SType> void stack<SType>::push(SType i)
{
if (tos==SIZE) {
cout << "Stack is full. \n";
return;
}
stck[tos] = i;
tos++;
}
// извлечение объекта из стека
template <class SType> SType stack<SType>::pop()
{
if(tos==0) {
cout << "Stack underflow.\n";
return 0;
}
tos --;
return stck[tos];
}
int main()
{
stack<int> a; // создание целочисленного стека
stack<double> b; // создание вещественного стека
stack<char> c; //создание символьного стека
int i;
// использование целого и вещественного стеков
```

```
a.push (1);
b.push (99.3);
a.push(2);
b.push(-12.23);
cout << a.pop() << " ";
cout << a.pop() <<  " ";
cout << b.pop() << " ";
cout << b.pop() << "\n";
// демонстрация символьного стека
for (i=0; i<10; i++) c.push ( (char) 'A'+i);
for (i=0; i<10; i+ + ) cout << c.pop();
cout << "\n";
return 0;
}
```

Задание 3.

Вариант 1
1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить сумму отрицательных элементов массива;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных между максимальным и минимальным элементами. Продемонстрировать работу методов класса.

Вариант 2
1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить сумму положительных элементов массива;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета произведения элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами. Продемонстрировать работу методов класса.

Вариант 3
1) В одномерном массиве, состоящем из n целых элементов, использую родовые функции, вычислить произведение элементов массива с четными номерами;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных между первым и последним нулевыми элементами. Продемонстрировать работу методов класса.

Вариант 4
1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить сумму элементов массива с нечетными номерами;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных между первым и последним отрицательными элементами. Продемонстрировать работу методов класса.

Вариант 5
1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить минимальный элемент массива;

2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных между первым и последним положительными элементами. Продемонстрировать работу методов класса.

Вариант 6
1) В одномерном массиве, состоящем из n целых элементов, использую родовые функции, вычислить номер максимального элемента массива;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета произведения элементов массива, расположенных между первым и вторым нулевыми элементами. Продемонстрировать работу методов класса.

Вариант 7
1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить номер минимального элемента массива;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных между первым и вторым отрицательными элементами. Продемонстрировать работу методов класса.

Вариант 8
1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить максимальный по модулю элемент массива;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных между первым и вторым положительными элементами. Продемонстрировать работу методов класса.

Вариант 9
1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить номер максимального по модулю элемента массива;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных после первого положительного элемента. Продемонстрировать работу методов класса.

Вариант 10
1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить количество элементов массива, лежащих в диапазоне от A до B;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных после максимального элемента. Продемонстрировать работу методов класса.

Вариант 11
1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить количество элементов массива, равных 0;
2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных после минимального элемента. Продемонстрировать работу методов класса.

Вариант 12

1) В одномерном массиве, состоящем из n вещественных элементов, использую родовые функции, вычислить количество элементов массива, больших C;

2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета произведения элементов массива, расположенных после максимального по модулю элемента. Продемонстрировать работу методов класса.

Вариант 13

1) В одномерном массиве, состоящем из n целых элементов, использую родовые функции, вычислить количество положительных элементов массива;

2) Создать класс-шаблон, содержащий поля для хранения одномерного массива и количества элементов в массиве. Описать методы для инициализации и вывода элементов массива на экран, а так же для подсчета суммы элементов массива, расположенных после последнего элемента, равного нулю. Продемонстрировать работу методов класса.

Контрольные вопросы:

1. В каких случаях используются перегруженные функции?
2. Правила описания перегруженных функций.
3. Как задается шаблон функции?
4. Какие основные свойства параметров шаблона?
5. Когда применяются шаблоны функций?
6. Дайте характеристику родовым классам.
7. Как задается родовой класс?
8. Какие классы входят в стандартную библиотеку шаблонов в C++?

Практическая работа №16. Обработка исключительных ситуаций

Система обработки исключительных ситуаций встроена в C++ и позволяет работать с ошибками, которые возникают во время работы программы, заранее предусмотренным и управляемым образом. С помощью системы обработки исключительных ситуаций C++ ваша программа при возникновении ошибки может автоматически вызвать процедуру ее обработки. Принципиальное преимущество обработки исключительных ситуаций состоит в том, что она автоматически в зависимости от ситуации запускает одну из множества подпрограмм обработки ошибок, которые предварительно "вручную" встраиваются в основную программу. Должным образом запрограммированная обработка исключительных ситуаций помогает создавать действительно отказоустойчивые программы.

Обработка исключительных ситуаций

C++ обеспечивает встроенный механизм обработки ошибок, называемый *обработкой исключительных ситуаций (exception handling).* Благодаря обработке исключительных ситуаций можно упростить управление и реакцию на ошибки во время выполнения программ. Обработка исключительных ситуаций в C++ организуется с помощью трех ключевых слов: **try, catch и throw.**

В самых общих словах, инструкции программы, во время выполнения которых вы хотите обеспечить обработку исключительных ситуаций, располагаются в блоке try. Если исключительная ситуация (т. е. ошибка) имеет место внутри блока try, она возбуждается (ключевое слово **throw),** перехватывается (ключевое слово **catch)** и обрабатывается. Ниже поясняется приведенное здесь общее описание.

Как уже отмечалось, любая инструкция, которая возбуждает исключительную ситуацию, должна выполняться внутри блока try. (Функции, которые вызываются из блока **try** также могут возбуждать исключительную ситуацию.) Любая исключительная ситуация должна перехватываться инструкцией **catch,** которая располагается непосредственно за блоком try, возбуждающем исклю

чительную ситуацию. Далее представлена основная форма инструкций **try** и **catch:**

```
try {
// блох возбуждения исключительной ситуации
}
catch (type1 arg) {
II блок перехвата исключительной ситуации
}
catch (type2 arg) {
II блок перехвата исключительной ситуации
}
catch (type3 arg) {
// блок перехвата исключительной ситуации
catch (typeN arg) {
II блок перехвата исключительной ситуации
}
```

Блок **try** должен содержать ту часть вашей программы, в который вы хотите отслеживать ошибки. Это могут быть как несколько инструкций внутри одной функции, так и все инструкции внутри функции **main()** (что естественно ведет к отслеживанию ошибок во всей программе).

После того как исключительная ситуация возбуждена, она перехватывается соответствующей этой конкретной исключительной ситуации инструкцией **catch,** которая ее обрабатывает. С блоком **try** может быть связано более одной инструкции **catch.** То, какая именно инструкция **catch** используется, зависит от типа исключительнойситуации. То есть,

если тип данных, указанный в инструкции **catch,** соответствует типу исключительной ситуации, выполняется данная инструкция **catch.** При этом все оставшиеся инструкции блока **try** игнорируются (т. е. сразу после того, как какая-то инструкция в блоке **try** вызвала появление исключительной ситуации, управление передается соответствующей инструкции **catch,** минуя оставшиеся инструкции блока try, — *примеч. пер.).* Если исключительная ситуация перехвачена, аргумент ***arg*** получает ее значение. Если вам не нужен доступ к самой исключительной ситуации, можно в инструкции **catch** указать только ее тип ***type***, аргумент ***arg*** указывать не обязательно. Можно перехватывать любые типы данных, включая и типы создаваемых вами классов. Фактически в качестве исключительных ситуаций часто используются именно типы классов.

Далее представлена основная форма инструкции **throw:**

throw *исключительная_ситуация;*

Инструкция **throw** должна выполняться либо внутри блока try, либо в любой функции, которую этот блок вызывает (прямо или косвенно). Здесь *исключительная_ситуация* — это возбуждаемая инструкцией **throw** исключительная ситуация.

Если вы возбуждаете исключительную ситуацию, для которой нет соответствующей инструкции **catch,** может произойти ненормальное завершение программы. Если ваш компилятор функционирует в соответствии со стандартом Standard C++, то возбуждение необрабатываемой исключительной ситуации приводит к вызову стандартной библиотечной функции **terminate().** По умолчанию для завершения программы функция **terminate()** вызывает функцию **abort()**, однако при желании можно задать собственную процедуру завершения программы. За подробностями обращайтесь к справочной документации вашего компилятора.

Примеры.

1. В следующем очень простом примере показано, как в C++ функционирует система обработки исключительных ситуаций:

```
// Простой пример обработки исключительной ситуации
#include <iostream>
using namespace std;

int main()
{
    cout << "начало\n";

    try {            // начало блока try
        cout << "Внутри блока try\n";
        throw 10;    // возбуждение ошибки
        cout << "Эта инструкция выполнена не будет";
    }
    catch (int i) { // перехват ошибки
        cout << "перехвачена ошибка номер: ";
        cout << i << "\n";
    }

    cout << "конец";

    return 0;
}
```

После выполнения программы на экран будет выведено следующее:

начало

Внутри блока try

Перехвачена ошибка номер: 10

конец

Внимательно изучите программу. Как видите, здесь имеется блок **try,** содержащий три инструкции, и инструкция **catch (int i),** обрабатывающая исключительную ситуацию целого типа. Внутри блока **try** будут выполнены только две из трех инструкций — инструкции **cout и throw.** После того как исключительная ситуация возбуждена, управление передается

выражению **catch** и выполнение блока **try** завершается. Таким образом, инструкция **catch** вызывается *не* явно, управление выполнением программы просто передается этой инструкции. (Для этого стек автоматически сбрасывается.) Следовательно, следующая за инструкцией **throw** инструкция **cout** не будет выполнена никогда. После выполнения инструкции **catch,** управление программы передается следующей за ней инструкции. Тем не менее, обычно блок **catch** заканчивают вызовом функции **exit(), abort()** или какой-либо другой функции, принудительно завершающей программу, поскольку, как правило, система обработки исключительных ситуаций предназначена для обработки катастрофических ошибок.

2. Как уже упоминалось, тип исключительной ситуации должен соответствовать типу, заданному в инструкции **catch.** Например, в предыдущем примере, если изменить тип данных в инструкции **catch** на **double,** то исключительная ситуация не будет перехвачена и будет иметь место ненормальное завершение программы. Это продемонстрировано в следующем фрагменте:

```
// Этот пример работать не будет
#include <iostream>
using namespace std;

int main()
{
    cout << "начало\n";

    try {                // начало блока try
        cout << "Внутри блока try\n";
        throw 10;        // возбуждение ошибки
        cout << "Эта инструкция выполнена не будет";
    }
    catch (double i) { // Эта инструкция не будет работать
                       // с исключительной ситуацией целого типа
        cout << "перехвачена ошибка номер: ";
        cout << i << "\n";
    }

    cout << "конец";

    return 0;
}
```

Поскольку исключительная ситуация целого типа не будет перехвачена инструкцией catch типа double, на экран программа выведет следующее:

начало
Внутри блока try
Abnormal program termination

3. Исключительная ситуация может быть возбуждена не входящей в блок try инструкцией, если сама эта инструкция входит в функцию, которая вызывается из блока try. Например, ниже представлена совершенно правильная программа:

```
/* Возбуждение исключительной ситуации из функции, находящейся вне блока try
*/
#include <iostream>
using namespace std;

void Xtest(int test)
{
    cout << "Внутри функции Xtest, test равно: " << test << "\n";
    if(test) throw test;
}

int main()
{
    cout << "начало\n";

    try {             // начало блока try
        cout << "Внутри блока try\n";
        Xtest(0);
        Xtest(1);
        Xtest(2);
    }
    catch (int i) { // перехват ошибки
        cout << "перехвачена ошибка номер: ";
        cout << i << "\n";
    }

    cout << "конец";

    return 0;
}
```

На экран программа выводит следующее:

начало
Внутри блока try
Внутри функции Xtest, test равно: О
Внутри функции Xtest, test равно: 1
Перехвачена ошибка номер: 1
конец

4. Блок **try** можно располагать внутри функции. В этом случае при каждом входе в функцию обработчик исключительной ситуации устанавливается снова. Например, рассмотрим следующую программу:

```
#include <iostream>
using namespace std;

// Блоки try и catch могут находиться не только в функции main()
void Xhandler(int test)
{
     try {
         if(test) throw test;
     }
     catch(int i) {
         cout << "перехвачена ошибка номер: " << i << '\n';
     }
}

int main()
{
     cout << "начало\n";

     Xhandler(1);
     Xhandler(2);
     Xhandler(0);
     Xhandler(3);

     cout << "конец";

     return 0;
}
```

На экран программа выводит следующее:

начало
Перехвачена ошибка номер: 1

Перехвачена ошибка номер: 2
Перехвачена ошибка номер: 3
конец

Как видите, обработаны три исключительные ситуации. После вызова каждой исключительной ситуации функция возвращает свое значение. При повторном вызове функции обработчик исключительной ситуации устанавливается вновь.

5. Как упоминалось ранее, с блоком **try** можно связать более одной инструкции **catch.** Как правило, так и делается. При этом каждая инструкция **catch** предназначена для перехвата своего типа исключительной ситуации. Например, в следующей программе перехватываются две исключительных ситуации, одна для целых и одна для строки:

```
#include <iostream>
using namespace std;

// Можно перехватывать разные типы исключительных ситуаций
void Xhandler(int test)
{
    try {
        if(test) throw test;
        else throw "Значение равно нулю";
    }
    catch(int i) {
        cout << "Перехвачена ошибка номер: " << i << '\n';
    }
    catch(char *str) {
        cout << "Перехвачена стока: ";
        cout << str << '\n';
    }
}

int main()
{
    cout << "начало\n";

    Xhandler(1);
    Xhandler(2);
    Xhandler(0);
    Xhandler(3);

    cout << "конец";

    return 0;
}
```

На экран программа выводит следующее:
начало
Перехвачена ошибка номер: 1
Перехвачена ошибка номер: *2*
Перехвачена строка: Значение равно нулю
Перехвачена ошибка номер: 3
конец

Как видите, каждая инструкция **catch** перехватывает только исключительные ситуации соответствующего ей типа.

Обычно выражения инструкций **catch** проверяются *в* том порядке, в котором они появляются в программе. Выполняется только та инструкция, которая совпадает по типу данных с исключительной ситуацией. Все остальные блоки **catch** игнорируются.

Дополнительная информация об обработке исключительных ситуаций

В системе обработки исключительных ситуаций имеется несколько дополнительных аспектов и нюансов, которые могут сделать ее понятнее и удобней для применения.

В некоторых случаях необходимо настроить систему так, чтобы перехватывать все исключительные ситуации, независимо от их типа. Сделать это достаточно просто. Для этого используйте следующую форму инструкции **catch:**

catch (. . .) {

// *обработка* ***всех*** *исключительных ситуаций*

Здесь многоточие соответствует любому типу данных. Для функции, вызываемой из блока **try,** вы можете ограничить число типовисключительных ситуаций, которые способна возбудить такая функция. Фактически можно даже запретить функции вообще возбуждать какую бы то ни было исключительную ситуацию. Для этого необходимо добавить в определение функции ключевое слово **throw.** Здесь представлена основная форма такого определения:

возвращаемый_тип имя_функции (список_аргументов) throw (*список_типов)*

Здесь в поле *спшок_типов* перечисляются через запятые только те типы данных исключительных ситуаций, которые могут быть возбуждены функцией. Возбуждение любого другого типа исключительной ситуации приведет к аварийному завершению программы. Если вы хотите, чтобы функция не возбуждала никаких исключительных ситуаций, то оставьте поле ***список_типов*** пустым.

Если ваш компилятор работает в соответствии с современным стандартом Standard C++, то попытка возбуждения неподдерживаемой исключительной ситуации приведет к вызову стандартной библиотечной функции unexpected ().

По умолчанию функция **unexpected ()** вызывает функцию abortQ, что ведет к варийному завершению программы. Однако при желаниивы можете задать собственную процедуру завершения программы. За подробностями обращайтесь к справочной документации вашего компилятора.

Если вы хотите в процедуре обработки исключительной ситуации возбудить повторную исключительную ситуацию, можно просто использовать инструкцию **throw** без параметров. Это приводит к тому, что текущая исключительная ситуация передается внешней последовательности инструкций **try/catch.**

Примеры

1. В следующей программе иллюстрируется использование инструкции **catch(...):**

```
// В этой программе перехватываются все типы исключительных ситуаций
#include <iostream>
using namespace std;

void Xhandler(int test)
{
    try {
        // возбуждение исключительной ситуации типа int
        if(test==0) throw test;
        // возбуждение исключительной ситуации типа char
        if(test==1) throw 'a';
        // возбуждение исключительной ситуации типа double
        if(test==2) throw 123.23;
    }
    catch(...) { // перехват исключительных ситуаций всех типов
        cout << "Перехвачена ошибка!\n";
    }
}

int main()
{
    cout << "начало\n";

    Xhandler(0);
    Xhandler(1);
    Xhandler(2);

    cout << "конец";

    return 0;
}
```

На экран программа выводит следующее:

начало
Перехвачена ошибка !
Перехвачена ошибка !
Перехвачена ошибка !
конец

Как видите, во всех трех случаях возбуждения исключительной ситуации в инструкции throw, она перехватывается с помощью единственной инструкции catch.

2. Очень удобно инструкцию catch(...) использовать в качестве последней в группе инструкций catch. В этом случае инструкция catch(...) по умолчанию становится инструкцией, которая "перехватывает все". Например, далее представлена слегка измененная версия предыдущей программы, где исключительные ситуации целого типа перехватываются явно, а все другие — с помощью инструкции catch(...):

```
/* В этом примере инструкция catch(...) по умолчанию перехватывает все типы исключительных ситуаций
*/
#include <iostream>
using namespace std;

void Xhandler(int test)
{
try {
        // возбуждение исключительной ситуации типа int
        if(test==0) throw test;
        // возбуждение исключительной ситуации типа char
        if(test==1) throw 'a';
        // возбуждение исключительной ситуации типа double
        if(test==2) throw 123.23;
    }
    catch(int i) { // перехват исключительной ситуации типа int
        cout << "Перехвачен " << i << '\n';
    }
    catch(...) { // перехват исключительных ситуаций остальных типов
        cout << "Перехвачена ошибка!\n";
    }
}

int main()
{
    cout << "начало\n";

    Xhandler(0);
    Xhandler(1);
    Xhandler(2);

    cout << "конец";

    return 0;
}
```

На экран программа выводит следующее:
начало
Перехвачен 1
Перехвачена ошибка 2
Перехвачена ошибка 3
конец

Как показано в этом примере, использование инструкции catch(...) таким образом —это хороший способ перехватывать те исключительные ситуации, которые вы не хотите обрабатывать явно. Кроме этого, путем перехвата всех исключительных ситуаций вы предотвращаете аварийное завершение программы из-за случайно необработанной исключительной ситуации.

3. В следующей программе показано, как ограничить число типов исключительных ситуаций, которые возбуждаются функцией:

```
/* Ограничение числа возбуждаемых функцией типов исключительных ситуаций
*/
#include <iostream>
using namespace std;

// Этой функцией могут возбуждаться только
// исключительные ситуации типов int, char и double
void Xhandler(int test) throw(int, char, double)
{
        // возбуждение исключительной ситуации типа int
        if(test==0) throw test;
        // возбуждение исключительной ситуации типа char
        if(test==1) throw 'a';
        // возбуждение исключительной ситуации типа double
        if(test==2) throw 123.23;
}

int main()
{
    cout << "начало\n";

    try {
        Xhandler(0); // попробуйте также передать в
                     // функцию Xhandler() значения 1 и 2
    }
    catch(int i) {
        cout << "Перехват int\n";
    }
    catch(char c) {
        cout << "Перехват char\n";
    }
    catch(double d) {
        cout << "Перехват double\n";
    }

    cout << "конец";

    return 0;
}
```

В этой программе функция **Xhandler()** может возбуждать только исключительные ситуации типа **int, char и double.** При попытке возбудить исключительную ситуацию другого типа произойдет аварийное завершение программы. (То есть будет вызвана функция **unexpected().** Чтобы убедиться в этом, удалите из списка допустимых исключительных ситуаций тип int и повторите запуск программы. Важно понимать, что ограничить типы возбуждаемых исключительных ситуаций можно только после того, как функция вызвана из блока **try.** То есть *внутри* функции блок **try** может возбудить любой тип исключительной ситуации, коль скоро она перехватывается *внутри* этой функции. Ограничения вступают в силу только тогда, когда исключительная ситуация не перехвачена функцией.

4. Следующее небольшое изменение в функции Xhandler() запрещает возбуждение любой исключительной ситуации:

```
// Эта функция НЕ может вызывать никаких исключительных ситуаций
void Xhandler (int test) throw ()
{
/*Следующие инструкции больше не работают. Наоборот, попытка их
выполнения ведет к ненормальному завершению программы
*/
// возбуждение исключительной ситуации типа int
if (test==0) throw test;
// возбуждение исключительной ситуации типа char
if(test==l) throw 'a';
// возбуждение исключительной ситуации типа double
if(test==2) throw 123.23;
```

5. Вы уже знаете, что можно повторно возбудить исключительную ситуацию. Смысл этого в том, чтобы предоставить возможность обработки исключительной ситуации нескольким

процедурам. Например, предположим, что одна процедура обрабатывает один аспект исключительной ситуации, а вторая — другой. Повторно исключительная ситуация может быть возбуждена только внутри блока **catch** (или любой функцией, которая вызывается из этого блока). Когда вы повторно возбуждаете исключительную ситуацию, она перехватывается не той же инструкцией **catch,** а переходит к другой, внешней к данной инструкции. В следующей программе иллюстрируется повторное возбуждение исключительной ситуации: возбуждается исключительная ситуация типа **char *.**

```
/* Пример повторного возбуждения исключительной ситуации одного и того же типа
*/
#include <iostream>
using namespace std;

void Xhandler()
{
    try {
        // возбуждение исключительной ситуации типа char *
        throw "привет";
    }
    // перехват исключительной ситуации типа char *
    catch(char *) {
        cout << "Перехват char * внутри функции Xhandler()\n";
        // повторное возбуждение исключительной ситуации
        // типа char *, но теперь уже не в функции Xhandler()
        throw;
    }
}

int main()
{
    cout << "начало\n";

    try {
        Xhandler();
    }
    catch(char *) {
        cout << " Перехват char * внутри функции main()\n";
    }

    cout << "конец";

    return 0;
}
```

На экран программа выводит следующее:
начало
Перехват char * внутри функции Xhandler()
Перехват char * внутри функции main()
Конец

Упражнения

Задание на «3»

1. Далее представлен каркас функции divide ().

```
double divide (double a, double b)
{
// добавьте обработку ошибок
return a/b;
}
```

Эта функция возвращает результат деления a на b. Добавьте в функцию процедуру обработки исключительных ситуаций, а конкретно предусмотрите обработку ошибки деления на ноль. Покажите, что ваша процедура работает.

Задание на «4,5»

1. Написать программу вычисления арифметических действий, в которой перехватываются исключения типа int, char *. Сгенерировать исключительную ситуацию.

Проверка усвоения

Практическая работа №17 Последовательные контейнеры.

Цель: закрепить умения и навыки по использованию последовательных классов из стандартной библиотеки C++ для реализации хранения и обработки информации.

Содержание отчёта:

1. Постановка задачи.
2. Текст программы.
3. Комментарии к ведённым обозначениям и описание используемых функций.
4. Результаты работы программы.

Задание 1
Разобрать примеры
Откомпилировать программы

Если не получится разработать свой собственный класс, хотя бы примените стандартный контейнер STL. [Павловская С/С++: учебник, гл.12–14]. Лучше сделать и то, и другое (решить задачу двумя способами).

Пример выполнения:
Описать класс, реализующий стек. Написать программу, использующую этот класс для моделирования Т-образного сортировочного узла на железной дороге. Программа должна разделять на два направления состав, состоящий из вагонов двух типов (на каждое направление формируется состав из вагонов одного типа). Предусмотреть возможность формирования состава из файла и с клавиатуры.

```
// Вариант 1: собственный класс
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

typedef struct {int num, typ; } vagon; // номеритипвагона (1/2)

class mstack{
  enum { EMPTY=-1 };
  vagon* s;
  int maxn,stk;
public:
  mstack(int n) { s=new vagon[n]; maxn=n; stk=EMPTY; } // конструктор
  ~mstack() { delete []s; }
  void push(vagon x);
  vagon pop();
  vagon top() const { return s[stk]; } ;
  bool empty() const { return stk==EMPTY; }
  bool full() const { return stk==maxn-1; }
  void reset() { stk=EMPTY; }
}; //-class mstack

void mstack::push(vagon x){  s[++stk]=x; }
vagon mstack::pop(){  return s[stk--]; }
```

```
#define N 20 // max vagon of sostav
int main(){
   vagon v;
   mstack s1(N),s2(N),a(N);

   ifstream in("a101.txt");
// читаем из каждой строки файла номер и тип вагона
while(!in.eof()){
     in>>v.num>>v.typ;
     if (in.eof()) break;
     cout<<v.num<<"   "<<v.typ<<'\n';
a.push(v);  // помещаем их в стек
}
   in.close();

   while (!a.empty()){ // пока стек не пуст
v=a.pop();  // берем последний элемент (вагон)
                                    // и помещаем его на один из 2 путей (в другие стеки)
if (v.typ==1) s1.push(v); else s2.push(v);
   }
// выводим составы
   cout<<"Sostav 1:\n";
   while (!s1.empty()){
        cout<<(s1.pop()).num<<" ";
   }
   cout<<"\n";
   cout<<"Sostav 2:\n";
   while (!s2.empty()){
        cout<<(s2.pop()).num<<" ";
   }
   cout<<"\n";
system("pause");
   return 0;
}

// Вариант 2: Используем шаблонный классstack из STL
// особенность в STL – метод извлечения из стека не возвращает результат
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <stack>

using namespace std;

typedef struct {int num, typ; } vagon;

#define N 20 // max vagon of sostav
int main(){
   vagon v;
stack<vagon, vector<vagon>>s1,s2,a; // строим стеки на основе векторов
```

```
ifstream in("a101.txt");
// читаем из каждой строки файла номер и тип вагона
while(!in.eof()){
     in>>v.num>>v.typ;
     if (in.eof()) break;
     cout<<v.num<<"   "<<v.typ<<'\n';
a.push(v);  // и помещаем их в стек
}
   in.close();

   while (!a.empty()){
v=a.top();  // Сначала нужно получить значение из вершины стека
a.pop();    // затем извлечь элемент (в никуда – только для изменения  указателя вершины стека)
                  // помещаем извлеченный вагон на один из двух путей
if (v.typ==1) s1.push(v); else s2.push(v);
   }
// вывод составов – почти как в первом варианте
cout<<"Sostav 1:\n";
   while (!s1.empty()){
        cout<<(s1.top()).num<<" ";  s1.pop();
   }
   cout<<"\n";
   cout<<"Sostav 2:\n";
   while (!s2.empty()){
        cout<<(s2.top()).num<<" ";  s2.pop();
   }
   cout<<"\n";
   system("pause");
   return 0;
}
```

ВАРИАНТЫ

Вар.	Задание 2: Разработка своего класса и/или использование контейнеров STL Требуется выводить содержимое всех контейнеров на каждом этапе: начальное состояние, состояние после преобразований. Вывод должен осуществляться отдельно от обработки данных.
1	Определить класс «строка» (последовательность символов), операции сцепления, лексикографического сравнения, ввода/вывода в поток. Обеспечить автоматическое приведение из/в тип **char***.
2	Определить класс «двунаправленный список», операции добавления и исключения элемента перед или после текущего, вывода списка в поток (без изменения состояния списка), очистки списка.
3	Определить класс «рациональное число» (простая дробь), позволяющий выполнять арифметические действия над ним без потери точности. Обеспечить приведение к типу double, ввод/вывод в поток, операции сравнения (<,>,==), арифметические операции (+,-,*,/).
4	Определить класс «треугольная матрица», операции матричного умножения, сложения, вычитания, доступ к элементу матрицы по его индексам, вывод в поток.
5	Определить класс «очередь координат», операции постановки в очередь и выборки очередного элемента, очистки, вывода очереди в поток (без изменения ее состояния), вычисления текущей длины.

	Использовать данный класс для решения следующей задачи: в матрице заданы числа от 0 до 3. При выборе некоторой ячейки матрицы ее значение увеличивается на 1. При достижении в ячейке числа 4, оно распадается на 4 единицы, которые добавляются к соседним ячейкам или уходят за пределы матрицы. В самой ячейке получается ноль. При этом возможна цепная реакция, если в соседних ячейках были тройки.
6	Определить класс «таблица имен (идентификаторов)», где каждому уникальному имени соответствует его тип. Например уникальным переменным A, B, Score может соответствовать тип int, . Реализовать операции добавления, исключения элемента, проверки принадлежности элемента таблице, вывода упорядоченной таблицы в поток, получения типа по заданному имени. Считать, что все идентификаторы состоят не более чем из ***N*** символов и ***N*** известно заранее.
7	Определить класс, моделирующий какую-либо стратегию управления распределением свободной памяти. Обеспечить операции выделения, освобождения участка памяти, вывод информации о состоянии «кучи» в поток, вычисление размера свободной памяти и размера максимального доступного блока. При необходимости реализовать «сбор мусора».
8	Определить тип данных «множество целых чисел», операции добавления, исключения элемента, вычисления пересечения, объединения, симметрической разности, проверки принадлежности элемента множеству, вывода множества в поток.
9	Разработать класс стек координат. Лабиринт представляется в виде матрицы, состоящей из квадратов. Каждый квадрат либо открыт, либо закрыт. Вход в закрытый квадрат запрещен. Если квадрат открыт, то вход в него возможен со стороны, но не с угла. Программа находит проход через лабиринт, двигаясь от заданного входа. После отыскания прохода программа выводит найденный путь в виде координат квадратов. Задайте лабиринт в виде текстового файла. Используйте символы: # – закрытый квадрат (стена); . – открытый квадрат (свободное место); A – вход, B – выход
10	Разработать класс «двухсвязный динамический список» для представления каталога файловой системы. Написать программу, моделирующую управление каталогом в файловой системе. Для каждого файла в каталоге содержатся следующие сведения: имя файла, дата создания, количество обращений к файлу. Программа должна обеспечивать: • начальное формирование каталога файлов; • вывод каталога файлов; • удаление файлов, дата создания которых раньше заданной; • выборку файла с наибольшим количеством обращений. Выбор моделируемой функции должен осуществляться с помощью меню.
11	Написать программу моделирования работы автобусного парка. Сведения о каждом автобусе содержат: номер автобуса, фамилию и инициалы водителя, номер маршрута. Программа должна обеспечивать выбор с помощью меню и выполнение одной из следующих функций: • начальное формирование данных о всех автобусах в парке в виде списка (ввод с клавиатуры или из файла); • имитация выезда автобуса из парка: вводится номер автобуса; программа удаляет данные об этом автобусе из списка автобусов, находящихся в парке, и записывает эти данные в список автобусов, находящихся на маршруте;

	• имитация въезда автобуса в парк: вводится номер автобуса; программа удаляет данные об этом автобусе из списка автобусов, находящихся на маршруте, и записывает эти данные в список автобусов, находящихся в парке; • вывод сведений об автобусах, находящихся в парке, и об автобусах, находящихся на маршруте. Для представления необходимых списков использовать класс «динамический список»
12	Написать программу учета заявок на авиабилеты. Каждая заявка содержит: пункт назначения, номер рейса, фамилию и инициалы пассажира, желаемую дату вылета. Программа должна обеспечивать выбор с помощью меню и выполнение одной из следующих функций: • добавление заявок в список; • удаление заявок; • вывод заявок по заданному номеру рейса и дате вылета; • вывод всех заявок Для представления необходимых списков использовать класс «динамический список»
13	Написать программу учета книг в библиотеке. Сведения о книгах содержат: фамилию и инициалы автора, название, год издания, количество экземпляров данной книги в библиотеке. Программа должна обеспечивать выбор с помощью меню и выполнение одной из следующих функций: • добавление данных о книгах, вновь поступающих в библиотеку; • удаление данных о списываемых книгах; • выдача сведений о всех книгах, упорядоченных по фамилиям авторов; • выдача сведений о всех книгах, упорядоченных по годам издания. Для хранения данных разработайте класс Хранение данных организовать с применением класса «бинарное дерево», в качестве ключа использовать «фамилию и инициалы автора». [Павловская С/С++: учебник, с.122 – организация бинарных деревьев]
14	Описать класс, реализующий бинарное дерево, обладающее возможностью добавления новых элементов, удаления существующих, поиска элемента по ключу, а также последовательного доступа ко всем элементам. Написать программу, использующую этот класс для представления англо-русского словаря. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса. Предусмотреть возможность формирования словаря из файла и с клавиатуры. [Павловская С/С++: учебник, с.122 – организация бинарных деревьев]
15	Разработать класс «стек» для хранения скобок. Проверить баланс скобок в выражении, используя стек. Скобки могут быть 3 видов: круглые, квадратные, фигурные. Пример: 3+(x+2*y*sin(5*x)–{2*ln[t]+sqrt{8}})/3
16	Составить описание класса для объектов-векторов, задаваемых координатами концов в трехмерном пространстве. Обеспечить операции сложения и вычитания векторов с получением нового вектора (суммы или разности), вычисления скалярного произведения двух векторов, длины вектора, косинуса угла между векторами. Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.
17	Составить описание класса для определения одномерных массивов целых чисел (векторов). Предусмотреть возможность обращения к отдельному элементу массива с контролем выхода за пределы массива, возможность задания произвольных границ индексов при создании объекта, возможность выполнения операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов,

	умножения и деления всех элементов массива на скаляр, вывода на экран элемента массива по заданному индексу, вывода на экран всего массрша. Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.
18	Составить описание класса многочленов от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Предусмотреть методы для вычисления значения многочлена для заданного аргумента, операции сложения, вычитания и умножения многочленов с получением нового объекта-многочлена, вывод на экран описания многочлена. Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.
19	Написать класс для эффективной работы со строками, позволяющий форматировать и сравнивать строки, хранить в строках числовые значения и извлекать их. Для этого необходимо реализовать: • перегруженные операции присваивания и конкатенации (сцепления строк); • операции сравнения и приведения типов; • преобразование в число любого типа; • форматный вывод строки. Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.
20	Описать класс «записная книжка». Предусмотреть возможность работы с произвольным числом записей, поиска записи по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона), добавления и удаления записей, сортировки по разным полям. Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Задания для выполнения

Задание 3.

Ниже представлен пример класса **Coord.** Напишите программу для хранения объектов типа **Coord,**используя последовательные контейнеры, соответствующие Вашему варианту.

```
class Coord {
public:
int x, y;
Coord() { x = y = 0; }
Coord(int a, int b) { x = a; y = b; }
};
```

Варианты:

1. Вектор и стек
2. Вектор и очередь
3. Вектор и двусторонняя очередь
4. Вектор и список
5. Вектор и очередь с приоритетами
6. Стек и очередь
7. Стек и двусторонняя очередь
8. Стек и список
9. Стек и очередь с приоритетами

10. Очередь и список.
11. Двусторонняя очередь и очередь с приоритетами.
12. Очередь и вектор
13. Очередь и стек.

Задание 4.

Используя программу из задания 3, реализуйте в главной функции программы, следующие действия:

- добавление и удаление элементов контейнеров по указанному номеру элемента;
- редактирование элементов по заданному номеру;
- поиск элементов по заданному номеру;
- сортировка элементов контейнера;
- определить количество элементов в контейнере;
- установить максимальное значение элементов в контейнере.

Задание 5.

Используя программу из задания 1, создать объекты-контейнеры obj1 и obj2. Обменять местами их элементы.

Создатьобъект-контейнерobj3 и сохраните в него элементы из obj1в обратном порядке.

Контрольные вопросы:

1. Дайте определение контейнерным классам, и приведите их примеры.
2. Чем последовательные контейнеры отличаются от ассоциативных?
3. Укажите заголовочные файлы и соответствующие им контейнерные классы?
4. Что такое итераторы? Укажите методы для доступа к элементам контейнерных классов.
5. Чем отличается очередь от двусторонней очереди и очереди с приоритетами?
6. На базе какого последовательного контейнера по умолчанию определен стек в STL?
7. Укажите особенности использования операции присваивания и логических операторов для объектов контейнерных классов.

Практическая работа №18. Итераторы

Цели работы:

- изучение необходимости итераторов;
- изучение правил определения и использования итераторов;
- изучение использования итераторов для работы с элементами списков;

Основные понятия

Итераторы можно условно разделить на две категории: основные и вспомогательные. Но прежде чем перейти к подробному описанию и тех и других, остановимся на двух важных правилах работы с итераторами: получения итераторов и отслеживания значения "за пределом". Многие функции и методы классов STL возвращают итераторы, вместо того чтобы производить действия над обычными указателями Си++. Классы STL, хранящие информацию, возвратят итератор, указывающий на первый элемент данных, если вызвать их метод begin(). Напротив, вызов метода end() приводит к возврату значения "за пределом" (past-the-end). Так называется значение итератора, при котором он указывает на элемент, следующий за последним разрешенным для ссылки элементом. Начиная со значения "за пределом" находится область данных, из которой нельзя читать и куда нельзя записывать. Нарушение этого запрета приведет к непредсказуемому результату. Представьте себе массив, например, из 10 элементов. Если итератор ссылается на некий одиннадцатый элемент, то говорят, что он ссылается на значение "за пределом". Разумеется, попытка записать в несуществующий элемент данные или оперировать данными этого элемента, как достоверными, приведет к неопределенному результату. Если бы вы попытались проделать такую операцию с блоком памяти в среде Windows, то немедленно получили бы исключение выхода за предел, что, скорее всего, окончилось бы аварийным завершением программы. Вот и судите сами, насколько важно следить за значением "за пределом".

Основные итераторы

Основные итераторы используются наиболее часто. И вы будете сталкиваться с ними постоянно. Поэтому с их рассмотрения мы и начнем.

Основные итераторы взаимозаменяемы.

Итераторы ввода

Итераторы ввода (input iterator) стоят в самом низу иерархии итераторов. Это наиболее простые из всех итераторов STL, и доступны они только для чтения. Итератор ввода может быть сравнен с другими итераторами на предмет равенства или неравенства, чтобы узнать, не указывают ли два разных итератора на один и тот же объект. Вы можете использовать оператор разыменовывания (*) для прочтения содержимого объекта, на который итератор указывает. Перемещаться от первого элемента, на который указывает итератор ввода, к следующему элементу можно с помощью оператора инкремента (++). Итераторы ввода возвращает только шаблонный класс istream_iterator. Однако, несмотря на то что итераторы ввода возвращаются единственным классом, ссылки на него присутствуют повсеместно. Это связано с тем, что вместо итератора ввода может подставляться любой из основных итераторов, за исключением итератора вывода, назначение которого прямо противоположно итератору ввода.

Проиллюстрируем это на примере алгоритма for_each. Если вы использовали ранее библиотеку контейнеров Borland BIDS, то операция for_each вам уже знакома. В STL она реализована следующим алгоритмом:

```
template <class InputIterator, class Function>
Function for_each (InputIterator first, InputIterator last, Function f)
{
   while (first != last) f(*first++);
   return f;
}
```

В этом примере итератор ввода выступает как первый параметр, указывающий на начало цепочки объектов, а второй параметр - также итератор ввода - это значение "за пределом" для этой цепочки. Тело алгоритма выполняет переход от объекта к объекту, вызывая для каждого значения, на которое указывает итератор ввода first, функцию. Указатель на нее передается в третьем параметре. Здесь задействованы все три перегруженных оператора, допустимые для итераторов ввода: сравнения (!=), инкремента (++) и разыменовывания (*). Так что лучше примера и не найти. В качестве первого и второго аргументов алгоритма сгодятся даже обычные указатели, о чем уже было сказано. Проиллюстрируем это. Пусть у нас имеется массив чисел, которые нужно распечатать. Тогда программа печати с использованием for_each может выглядеть так:

```
#include <iostream>
#include <algorithm>
#pragma warning (disable: 4550)
using namespace std;
void printValue(int num)
{
  cout << num << "\n";
}
main(void)
{
  int init[] = {1, 2, 3, 4, 5};
  for_each(init, init + 5, printValue);
```

```
}
```

Программа чрезвычайно проста. Есть массив init с пятью числами. Вызывается алгоритм for_each, ему в качестве входных итераторов передаются указатели на начало массива и на адрес, следующий за концом массива, т. е. на значение "за пределами". Третьим параметром является указатель на функцию printValue, которая печатает элементы массива. Чтобы включить в программу возможность использования потоков, добавляется включаемый файл iostream, а для описания прототипа алгоритма for_each в программу включается заголовочный файл algorithm (algorith для продуктов Borland). Обязательным при использовании STL является использование директивы:

```
using namespace std,
```

включающей пространство имен библиотеки STL. Поскольку пример изготовлен в среде компилятора Microsoft Visual C++ 5.0, была добавлена следующая директива:

```
#pragma warning (disable: 4550)
```

Она отключает назойливое предупреждение об отсутствии параметров, что, впрочем, ни на скорость (ни на зарплату!), конечно, не влияет, так что эту строку вполне можно опустить.

Итераторы вывода

Если итератор ввода предназначен для чтения данных, то итератор вывода (output iterator) служит для ссылки на область памяти, куда выводятся данные. Итераторы вывода можно встретить повсюду, где происходит хоть какая-то обработка информации средствами STL. Это могут быть алгоритмы (алгоритмы STL - специальные блоки кода, выполняющие определенные операции по обработке данных) копирования, склейки и т. п. Для данного итератора определены операторы присвоения (=), разыменовывания (*) и инкремента (++). Однако следует помнить, что первые два оператора предполагают, что итератор вывода располагается в левой части выражений, т. е. во время присвоения он должен быть целевым итератором, которому присваиваются значения. Разыменовывание нужно делать лишь для того, чтобы присвоить некое значение объекту, на который итератор ссылается. Итераторы ввода могут быть возвращены итераторами потоков вывода (ostream_iterator) и итераторами вставки inserter, front_inserter и back_inserter (описаны в разделе "Итераторы вставки").

Ниже приведен типичный пример использования итераторов вывода:

```
#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

main(void)
```

```
{
  int init1[] = {1, 2, 3, 4, 5};
  int init2[] = {6, 7, 8, 9, 10};
  vector<int> v(10);
  merge(init1, init1 + 5, init2, init2 + 5, v.begin());
  copy(v.begin(), v.end(), ostream_iterator<int>(cout, "\n"));
}
```

В отличие от предыдущего примера, здесь, помимо потоков и алгоритмов, использован контейнер (контейнеры STL - структуры данных для хранения информации определенным способом) типа "вектор", т. е. одномерный массив. У него имеются специальные методы begin() и end(), задача которых - возвращать итератор, указывающий на начало вектора и значение "за пределом" соответственно. В приведенном нами примере создаются и инициализируются два массива - init1 и init2. Далее их значения соединяются вместе алгоритмом merge и записываются в вектор. А для проверки полученного результата мы пересылаем данные из вектора в поток вывода, для чего вызываем алгоритм копирования copy и специальный итератор потока вывода ostream_iterator. Он перешлет данные в поток cout, разделив каждое пересылаемое значение символом окончания строки. Для шаблонного класса ostream_iterator требуется указать тип выводимых значений. В нашем случае это int.

С этим примером вы можете проделать эксперимент, чтобы узнать, к чему приводит обращение к значению по адресу "за пределом". Для этого достаточно сделать вектор размером не 10, а 9 элементов:

```
vector<int> v(9);
```

Однонаправленные итераторы

Если соединить итераторы ввода и вывода, то получится однонаправленный итератор (forward iterator), который может перемещаться по цепочке объектов в одном направлении, за что и получил такое название. Для такого перемещения в итераторе определена операция инкремента (++). И разумеется, в однонаправленном итераторе есть операторы сравнения (== и !=), присвоения (=) и разыменовывания (*). Все эти операторы можно увидеть, если посмотреть, как реализован, например, алгоритм replace, заменяющий одно определенное значение на другое:

```
template <class ForwardIterator, class T>
void replace (ForwardIterator first, ForwardIterator last, const T& old_value,
              const T& new_value)
{
```

```
    while (first != last)
    {
        if (*first == old_value) *first = new_value;
        ++first;
    }
}
```

Чтобы убедиться в правильности работы всех операторов однонаправленных итераторов, составим программу, заменяющую в исходном массиве все единицы на нули и наоборот, т. е. произведем инверсию. С этой целью все нули изначально заменяются на некоторое нейтральное значение, например на двойку, затем все единицы обнуляются, а все двойки становятся единицами:

```
#include <algorithm>
#include <iostream>
using namespace std;
main(void)
{
  replace(init, init + 5, 0, 2);
  replace(init, init + 5, 1, 0);
  replace(init, init + 5, 2, 1);
  copy(init, init + 5, ostream_iterator<int>(cout, "\n"));
}
```

Как видите, алгоритм replace, умело используя однонаправленные итераторы, читает значения, заменяет их и перемещается от одного к другому.

Двунаправленные итераторы

Двунаправленный итератор (bidirectional iterator) аналогичен однонаправленному итератору. В отличие от последнего двунаправленный итератор может перемещаться не только из начала в конец цепочки объектов, но и наоборот. Это становится возможным благодаря наличию оператора декремента (-). На двунаправленных алгоритмах базируются различные алгоритмы, выполняющие реверсивные операции, например reverse. Этот алгоритм меняет местами все объекты в цепочке, на которую ссылаются переданные ему итераторы. Следующий пример был бы невозможен без двунаправленных итераторов:

```
#include <algorithm>
#include <iostream>
using namespace std;
main(void)
{
  int init[] = {1, 2, 3, 4, 5};
  reverse(init, init + 5);
  copy(init, init + 5, ostream_iterator<int>(cout, "\n"));
}
```

Итераторы двунаправленного доступа возвращаются несколькими контейнерами STL: list, set, multiset, map и multimap.

Итераторы произвольного доступа

Итераторы произвольного доступа - самые "умелые" из основных итераторов. Они не только реализуют все функции, свойственные итераторам более низкого уровня, но и обладают большими возможностями. Глядя на исходные тексты, в которых используются итераторы произвольного доступа, можно подумать, что имеешь дело с арифметикой указателей языка C++. Реализованы такие операции, как сокращенное сложение и вычитание (+= и -=), сравнение итераторов (<, >, <= и >=), операция обращения к заданному элементу массива ([]), а также и некоторые другие операции.

Как правило, все сложные алгоритмы, требующие расширенных вычислений, оперируют итераторами произвольного доступа. Ниже приводится пример, в котором мы используем практически все операции, допустимые для них. Исходный текст разбит на части, к каждой из которых мы приводим комментарии. Сначала нужно включить требуемые заголовочные файлы и определить константу пробела^:

```
#include <algorithm>
#include <iostream>
#include <vector>
#define space " "
```

Затем включить использование STL:

```
using namespace std;
```

В функции main мы описываем массив числовых констант и вектор из пяти элементов:

```
int main(void)
{
  const int init[] = {1, 2, 3, 4, 5};
  vector<int> v(5);
```

Создаем переменную типа "итератор произвольного доступа". Для этого берем итератор и на его основе создаем другой, более удобный:

```
 typedef vector<int>::iterator vectItr;
  vectItr itr ;
```

Инициализируем вектор значениями из массива констант и присваиваем адрес его первого элемента итератору произвольного доступа:

```
  copy(init, init + 5, itr = v.begin());
```

Отсюда начинается самое интересное. Воспользовавшись разнообразными доступными операторами, мы последовательно читаем элементы вектора, начиная с конца, и выводим их на экран:

```
  cout << *( itr + 4 ) << endl;
  cout << *( itr += 3 ) << endl;
  cout << *( itr -= 1) << endl;
  cout << *( itr = itr - 1) << endl;
  cout << *( -itr ) << endl;
```

После этого итератор, претерпев несколько изменений, снова указывает на первый элемент вектора. А чтобы убедиться, что значения в векторе не были повреждены, и проверить оператор доступа ([]), выведем в цикле значения вектора на экран:

```
 for(int i = 0; i < (v.end() - v.begin()); i++)
    cout << itr[i] << space;
  cout << endl;
}
```

Как видите, операции с итераторами произвольного доступа реализованы так, чтобы программист не чувствовал разницы между использованием обычных указателей и итераторов.

Итераторы произвольного доступа возвращают такие контейнеры, как vector и deque.

Вспомогательные итераторы

Вспомогательные итераторы названы так, потому что они выполняют вспомогательные операции по отношению к основным. Часто можно встретить выражения, в которых основные итераторы и вспомогательные работают "рука об руку".

Реверсивные итераторы

Некоторые классы-контейнеры спроектированы так, что по хранимым в них элементам данных можно перемещаться в заданном направлении. В одних контейнерах это направление от первого элемента к последнему, а в других - от элемента с самым большим значением к элементу, имеющему наименьшее значение. Однако знайте, что существует специальный вид итераторов, называемых реверсивными. Такие итераторы работают "с точностью до наоборот": т. е. если в контейнере итератор ссылается на первый элемент данных, то реверсивный итератор ссылается на последний. Получить реверсивный итератор для контейнера можно вызовом метода rbegin(), а реверсивное значение "за пределом" возвращается методом rend(). Следующий пример использует нормальный итератор для вывода значений от 1 до 5 и реверсивный итератор для вывода этих же значений, но в обратном порядке:

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
main(void)
{
  const int init[] = {1, 2, 3, 4, 5};
  vector<int> v(5);
  copy(init, init + 5, v.begin());
  copy(v.begin(), v.end(), ostream_iterator<int>(cout, " "));
  copy(v.rbegin(), v.rend(), ostream_iterator<int>(cout, " "));
}
```

Итераторы потоков

Важную роль в STL играют итераторы потоков, которые делятся на итераторы потоков ввода и вывода. Практически во всех показанных в этой статье примерах вы найдете

итератор потока вывода для отображения данных на экране. Суть применения потоковых итераторов в том, что они превращают любой поток в итератор, используемый точно так же, как и прочие итераторы: перемещаясь по цепочке данных, считывает значения объектов и присваивает им другие значения.

Итератор потока ввода - это удобный программный интерфейс, обеспечивающий доступ к любому потоку, из которого требуется считать данные. Конструктор итератора имеет единственный параметр - поток ввода. А поскольку итератор потока ввода представляет собой шаблон, то ему передается тип вводимых данных. Вообще-то должно передаваться четыре типа, но последние три имеют значения по умолчанию, и вряд ли вы решитесь их изменять прежде, чем досконально изучите STL. Каждый раз, когда вам нужно ввести очередной элемент информации, используйте оператор ++ точно так же, как с основными итераторами. Считанные данные можно узнать, если применить разыменовывание (*).

Итератор потока вывода весьма схож с итератором потока ввода, но у его конструктора имеется дополнительный параметр, которым указывают строку-разделитель, добавляемую в поток после каждого выведенного элемента. Ниже приведен пример программы, читающей из стандартного потока cin числа, вводимые пользователем и дублирующие их на экране, завершая сообщение строкой " - last entered value". Работа программы заканчивается, как только пользователь введет число 666:

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
main(void)
{
  istream_iterator<int> is(cin);
  ostream_iterator<int> os(cout, " - last entered value\n");
  int input;
  while((input = *is) != 666)
  {
    *os++ = input;
    is++ ;
  }
}
```

Потоковые итераторы имеют одно существенное ограничение - в них нельзя возвратиться к предыдущему элементу. Единственный способ сделать это - заново создать итератор потока.

Итераторы вставки

Появление итераторов вставки (insert iterator) было продиктовано необходимостью. Без них просто невозможно добавить значения к цепочке объектов. Так, если в массив чисел, на которые ссылается итератор вывода, вы попытаетесь добавить пару новых значений, то итератор вывода попросту запишет новые значения на место старых, и они будут потеряны. Любой итератор вставки: front_inserter, back_inserter или inserter выполнит ту же операцию вполне корректно. Первый из них добавляет объекты в начало цепочки, второй - в конец. Третий итератор вставляет объекты в заданное место цепочки. При всем удобстве итераторы вставки имеют довольно жесткие ограничения. Так, front_inserter и back_inserter не могут работать с наборами (set) и картами (map), а front_inserter к тому же не умеет добавлять данные в начало векторов (vector). Зато итератор вставки inserter добавляет объекты в любой контейнер. Одной интересной странностью обладает front_inserter: данные, которые он вставляет в цепочку объектов, должны передаваться ему в обратном порядке. Очень неудобно, однако можно привыкнуть.

Приведем пример, как всегда разбитый на части для удобства комментирования. В нем создается список (list) с двумя значениями, равными нулю. После этого в начало и конец списка добавляются значения 1, 2, 3. Третья последовательность 1-1-1 вставляется в середину списка между нулями. Итак, после описания заголовочных файлов мы создаем массивы, необходимые для работы, и контейнер типа "список" из двух элементов:

```
#include <algorithm>
#include <iostream>
#include <list>
using namespace std;
main(void)
{
  int init[]  = {0, 0};
  int init1[] = {3, 2, 1};
  int init2[] = {1, 2, 3};
  int init3[] = {1, 1, 1};
  list<int> l(2);
```

Затем список инициализируется нулями из массива init и его значения отображаются на экране:

```
  copy(init, init + 2, l.begin());
  copy(l.begin(), l.end(), ostream_iterator<int>(cout, " "));
  cout << " - before front_inserter" << endl;
```

Итератором вставки в начало списка в обратном порядке добавляются значения массива init1, и производится повторный показ данных из списка на экране:

```
  copy(init1, init1 + 3, front_inserter(l));
  copy(l.begin(), l.end(), ostream_iterator<int>(cout, " "));
  cout << " - before back_inserter" << endl;
```

Теперь итератор вставки в конец добавит элементы массива init2 в "хвост" списка:

```
  copy(init2, init2 + 3, back_inserter(l));
  copy(l.begin(), l.end(), ostream_iterator<int>(cout, " "));
  cout << " - before inserter" << endl;
```

Сложнее всего обстоит дело с итератором insertes. Для него, кроме ссылки на сам контейнер, нужен итератор, указывающий на тот объект в контейнере, за которым будет произведена вставка единичек из массива init3. С этой целью мы создаем переменную типа "итератор", инициализируя ее итератором, указывающим на начало списка:

```
  list<int>::iterator& itr = l.begin();
```

Теперь специальной операцией advance делаем приращение переменной итератора так, чтобы она указывала на четвертый объект в цепочке данных списка:

```
  advance(itr, 4);
```

Остается добавить данные в цепочку посредством insertes и показать итог нашей работы на дисплее:

```
  copy(init3, init3 + 3, inserter(l, itr));
  copy(l.begin(), l.end(), ostream_iterator<int>(cout, " "));
  cout << " - the end!" << endl;
}
```

Константный итератор

Последний итератор, который мы рассмотрим, - константный (constant iterator). Он образуется путем модификации основного итератора. Константный итератор не допускает изменения данных, на которые он ссылается. Можно считать константный итератор указателем на константу. Чтобы получить константный итератор, можно воспользоваться типом const_iterator, предопределенным в различных контейнерах. К примеру, так можно описать переменную типа константный итератор на список:

```
list<int>::const_iterator c_itr;
```

Операции с итераторами

Существуют две важных операции для манипуляции ими. С одной из них - advance - вы познакомились в последнем примере. Это просто удобная форма инкрементирования итератора iter на определенное число n:

```
void advance (InputIterator& iter, Distance& n);
```

Вторая операция измеряет расстояние между итераторами first и second, возвращая полученное число через ссылку n:

```
void distance(InputIterator& first, InputIterator& second, Distance& n);
```

Задания

Выполнить задание не используя библиотеку стандартных шаблонов.

1. Разработать класс «Упорядоченный список», тип которого определяется заданием в п. a), предусмотрев в нем конструкторы инициализации и копирования, деструктор, функции вставки и удаления элемента, просмотра доступного элемента, функцию проверки наличия элементов.

2. Разработать класс «Итератор», который должен содержать конструктор и функции просмотра текущего элемента, перехода к следующему элементу, перехода в начало списка, перехода по заданному адресу, выдачу текущего адреса.

3. Используя разработанные классы, решить задачу из пункта b). Содержимым элемента списка является целое число.

Вариант 1

a) Простой однонаправленный упорядоченный по возрастанию.

b) Из списков s1 и s2 образовать список s3, содержащий только одинаковые элементы s1 и s2.

Вариант 2

a) Простой однонаправленный упорядоченный по убыванию.

b) Из списка s образовать два списка s1 и s2. В s1 должны войти все элементы, содержимое которых меньше заданного числа k, а в s2 - остальные.

Вариант 3

a) Простой однонаправленный упорядоченный по возрастанию.

b) Из списков s1 и s2 образовать список s3, включив в него элементы из s1, не содержащиеся в s2 и элементы из s2, не содержащиеся в s1.

Вариант 4

a) Простой однонаправленный упорядоченный по убыванию.

b) Удалить из списка s все элементы с значением содержимого, превышающим заданное число k.

Вариант 5

a) Простой однонаправленный упорядоченный по возрастанию.

b) Из списков s1 и s2 образовать список s3, являющийся объединением s1 и s2, т.е. одинаковые элементы s1 и s2 должны войти в s1 один раз.

Вариант 6

a) Простой однонаправленный упорядоченный по убыванию.

b) Включить в список s1 все элементы списка s2, содержимое которых меньше заданного числа k.

Вариант 7

a) Простой однонаправленный упорядоченный по возрастанию.

b) Из списка s образовать два списка s1 и s2. В s1 должны войти все элементы, содержимое которых меньше заданного числа k, а в s2 - остальные.

Вариант 8

a) Простой однонаправленный упорядоченный по убыванию.

b) Из списка s1 образовать список s2, в который включены все элементы s1, превышающие заданное число k.

Вариант 9

a) Простой однонаправленный упорядоченный по возрастанию.

b) Из списка s1 образовать список s2, в котором из элементов в s1, имеющих одинаковое содержимое, присутствует только один.

Вариант 10

a) Простой однонаправленный упорядоченный по убыванию.

b) Из списков s1 и s2 удалить все элементы с одинаковым содержимым.

Вариант 11

a) Простой однонаправленный упорядоченный по возрастанию.

b) Из списков s1 и s2 образовать общий список s, включающий все элементы как списка s1 так и s2.

Вариант 12

a) Простой однонаправленный упорядоченный по возрастанию.

b) Из списков s1 и s2 образовать список s3, содержащий только одинаковые элементы s1 и s2.

Контрольные вопросы:

1. Что собой представляют итераторы?
2. Перечислите основные виды итераторов.
3. Есть ли отличия в использовании итераторов для контейнерных классов?
4. Может ли итератор быть унаследован?
5. Какие методы можно использовать для работы с итераторами?

Содержание отчёта:

✓ Постановка задачи.
✓ Текст программы.
✓ Комментарии к ведённым обозначениям и описание используемых функций.
✓ Результаты работы программы.
✓ Ответы на контрольные вопросы

Практическая работа №19. Ассоциативные контейнеры.

1. Работа с примерами.

Для каждого из следующих примеров указать название ассоциативного класса, выписать методы этих классов, которые используются в программе, и назначение перечисленных методов.

Пример 1.

В следующей программе на примере ассоциативного списка, предназначенного для хранения десяти пар ключ/значение, иллюстрируются основы использования ассоциативных списков. Ключом здесь является символ, а значением — целое. Когда пользователь набирает на клавиатуре ключ (т. е. одну из букв от A до J), программа выводит на экран соответствующее этому ключу значение.

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
   map<char, int> m;
   int i;
   // размещение пар в ассоциативном списке
   for(i=0; i<10; i++) {
      m.insert(pair<char, int>('A' + i, i));
   }
   char ch;
   cout << "Введите ключ: ";
   cin >> ch;
   map<char, int>::iterator p;
   // поиск значения по заданному ключу
   p = m.find(ch);
   if(p != m.end())
      cout << p->second;
   else
      cout << "Такого ключа в ассоциативном списке нет\n";
   return 0;
}
```

Обратите внимание на использование класса-шаблона **pair** для образования пар ключ/значение. Типы данных, указанные в классе -шаблоне **pair,** должны соответствовать типам данных, хранящимся в ассоциативном списке.

После того как ассоциативный список инициализирован парами ключ/значение, найти нужное значение по заданному ключу можно с помощью функции find(). Функция **find()** возвращает итератор соответствующего ключу элемента или итератор конца ассоциативного списка, если указанный ключ не найден. Когда соответствующее ключу значение найдено, оно сохраняется в качестве второго члена класса-шаблона **pair.**

Пример 2.

В предыдущем примере типы пар ключ/значение были указаны явно в конструкции **pair<char, int>.** Хотя такой подход совершенно правилен, часто проще использовать функцию **make_pair(),** которая создает пары объектов на основе типов данных своих параметров.

```
#include <iostream>
#include <map>
using namespace std;
```

```
int main()
{
   map<char, int> m;
   int i;
   // размещение пар в ассоциативном списке
   for(i=0; i<10; i++) {
      m.insert(make_pair(char)('A' + i, i));
   }
   char ch;
   cout << "Введите ключ: ";
   cin >> ch;
   map<char, int>::iterator p;
   // поиск значения по заданному ключу
   p = m.find(ch);
   if(p != m.end())
      cout << p->second;
   else
      cout << "Такого ключа в ассоциативном списке нет\n";
   return 0;
}
```

Данный пример отличается от предыдущего только строкойm.insert(make_pair(char)('A' + i, i));

В данном случае, чтобы в операции сложения 'A' + i не допустить автоматического преобразования в тип int, используется приведение к типу **char.** Во всем остальном процесс определения типов объектов выполняется автоматически.

Пример 3.

Так же как и в других контейнерах, в ассоциативных списках можно хранить создаваемые вами типы данных. Например, в представленной ниже программе создается ассоциативный список для хранения слов с соответствующими словам антонимами. С этой целью используются два класса: **word** (слово) и **opposite** (антоним). Поскольку для ассоциативных списков поддерживается отсортированный список ключей, в программе для объектов типа **word** определяется оператор <. Как правило, оператор < необходимо перегружать для всех классов, объекты которых предполагается использовать в качестве ключей. (Помимо оператора < в некоторых компиляторах может потребоваться определить дополнительные операторы сравнения.)

```
// Ассоциативный список слов и антонимов
#include <iostream>
#include <map>
#include <cstring>
using namespace std;
class word {
   char str[20];
public:
   word() { strcpy(str, ""); }
   word(char *s) { strcpy(str, s); }
   char *get() { return str; }
};
// для объектов типа word следует определить оператор < (меньше)
bool operator<(word a, word b)
{
```

```
    return strcmp(a.get(), b.get()) < 0;
}
class opposite {
    char str[20];
public:
    opposite() { strcmp(str, ""); }
    opposite(char *s) { strcpy(str, s); }
    char *get() { return str; }
};
int main()
{
    map<word, opposite> m;

    // размещение в ассоциативном списке слов и антонимов
    m.insert(pair<word, opposite>
        (word("да"),opposite("нет")));
    m.insert(pair<word, opposite>
        (word("хорошо"),opposite("плохо")));
    m.insert(pair<word, opposite>
        (word("влево"),opposite("вправо")));
    m.insert(pair<word, opposite>
        (word("вверх"),opposite("вниз")));
    // поиск антонима по заданному слову
    char str[80];
    cout << "Введите слово: ";
    cin >> str;
    map<word, opposite>::iterator p;
    p = m.find(word(str));
    if(p != m.end())
        cout << "Антоним: " << p->second.get();
    else
        cout << "Такого слова в ассоциативном списке нет\n";
    cout << "ob1: " << ob1.geta() << endl;
    cout << "ob2: " << ob2.geta() << endl;
    return 0;
}
```

В данном примере любой объект, который вводится в ассоциативный список, представляет собой символьный массив для хранения заканчивающейся нулем строки. Далее в этой главе будет показано, как упростить эту программу, используя стандартный тип данных **string.**

2. Реализация ассоциативных классов

Составить программу, которая содержит текущую информацию о книгах в библиотеке.
Сведения о книгах содержат:

- номер УДК;
- фамилию и инициалы автора;
- название;
- год издания;
- количество экземпляров данной книги в библиотеке.

Программа должна обеспечивать:

- начальное формирование данных о всех книгах в библиотеке в виде двоичного дерева;
- добавление данных о книгах, вновь поступающих в библиотеку;

• удаление данных о списываемых книгах;
• по запросу выдаются сведения о наличии книг в библиотеке, упорядоченные по годам издания.

Для реализации хранения информации о книгах используйте один из ассоциативных классов.

Практическая работа №20. Алгоритмы.

Цель: закрепить умения и навыки по использованию элементов библиотеки STL для обработки различных последовательностей.

Содержание отчёта:

Постановка задачи.

Текст программы.

Комментарии к ведённым обозначениям и описание используемых функций. Результаты работы программы.

Задание на «3»

Наберите следующие программы. Исправьте ошибки в программном коде, откомпилируйте и проверьте работу программ.

1. **Определить количество четных элементов в векторе.**

```
#include <iostream>
#include <vector>
#include <algoritm>
using namespace std;

bool even(int x)
{
   return !(x%2);
}

int main()
{
   vector<int> v;
   int i;

   for(i=0; i<20; i++) {
      if(i%2) v.push_back(1);
      else v.push_back(2);
   }

   cout << "Последовательность: ";
   for(i=0; i<v.size(); i++) cout << v[i] << " ";
   cout << endl;


  int n;
  n = count(v.begin(), v.end(), 1);
  cout << n << " элементов равно 1\n";

  n = count_if(v.begin(), v.end(), even);
  cout << n << " четных элементов\n";

  return 0;
}
```

2. **Поиск элемента в списке, соответствующий введенному пользователем числу**

```
include <algoritm>
#include <lst>
#include <iostream>
int main()
{
  int search_value;
  int ia[ 6 ] = { 27, 210, 12, 47, 109, 83 };
  list<int> ilist( ia, ia+6 );

  cout << "enter search value: ";
  cin >> search_value;

  list<int>::iteratar presult;
  presult = find( ilist.began(), ilist.end(), search_value );

  cout << "The value "<<search_value
    <<( presult = ilist.end(
        ? " is not present" : " is present" )
 << endl;
}
```

3. Поиск заданного элемента в векторе

```
#include <iostream>
#include <algorithm> //используем алгоритм search
#include <vectors> //используем вектор

using namespace std; //нужно пространство имен std

bool mypredicate ( int i, int j) //это предикат
{
    return (i=j); //если i не равно j вернет ноль, иначе вернет 1
}

int main ()
{
    vector< int > v; //наш вектор v
    vector< int >::iteator it; //наш итератор it

    // С помощью цикла записываем в вектор значения: 10 20 30 40 50 60 70 80 90
    for ( int i=1; i<10; i++) v.push_back(i*10);
     copy(v.begin(),v.end(), ostream_iterator<int>(cout," ")); //вывод на экран элементов
вектора
    cout<<endl<<endl;

    // сейчас используем обычный способ поиска последовательности:
    int match1[] = {40,50,60,70}; //это последовательность, которую ищем
```

Задание на «4-5»
Выполнить задания с использованием алгоритмов библиотеки STL.
1. Обработка одномерных массивов

Вариант 1
В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:
1) номер минимального по модулю элемента массива;
2) сумму модулей элементов массива, расположенных после первого отрицательного элемента.
Сжать массив, удалив из него все элементы, величина которых находится в интервале [a,b]. Освободившиеся в конце массива элементы заполнить нулями.
Вариант 2
в одномерном массиве, состоящем из *n* вещественных элементов, вычислить:
1) номер максимального по модулю элемента массива;
2) сумму элементов массива, расположенных после первого положительного элемента.
Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых лежит в интервале [a,b], а потом — все остальные.
Вариант 3
в одномерном массиве, состоящем из *n* вещественных элементов, вычислить:
1) количество элементов массива, лежащих в диапазоне от A до B;
2) сумму элементов массива, расположенных после максимального элемента.
Упорядочить элементы массива по убыванию модулей элементов.
Вариант 4
в одномерном массиве, состоящем из *n* вещественных элементов, вычислить:
1) количество элементов массива, равных 0;
2) сумму элементов массива, расположенных после минимального элемента.
Упорядочить элементы массива по возрастанию модулей элементов.
Вариант 5
в одномерном массиве, состоящем из *n* вещественных элементов, вычислить:
1) количество элементов массива, больших C;
2) произведение элементов массива, расположенных после максимального по модулю элемента.
Преобразовать массив таким образом, чтобы сначала располагались все отрицательные элементы, а потом — все положительные (элементы, равные O, считать положительными).
Вариант 6
в одномерном массиве, состоящем из *n* вещественных элементов, вычислить:
1) количество отрицательных элементов массива;
2) сумму модулей элементов массива, расположенных после минимального по модулю элемента.
Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.
Вариант 7
в одномерном массиве, состоящем из *n* целых элементов, вычислить:
1) количество положительных элементов массива;
2) сумму элементов массива, расположенных после последнего элемента, равного нулю.
Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых не превышает 1, а потом — все остальные.
Вариант 8
в одномерном массиве, состоящем из *n* вещественных элементов, вычислить:
1) количество элементов массива, меньших C;

2) сумму целых частей элементов массива, расположенных после последнего отрицательного элемента.
Преобразовать массив таким образом, чтобы сначала располагались все элементы, отличающиеся от максимального не более чем на 20%, а потом — все остальные.

Вариант 9

в одномерном массиве, состоящем из *n* вещественных элементов, вычислить:
1) произведение отрицательных элементов массива;
2) сумму положительных элементов массива, расположенных до максимального элемента.
Изменить порядок следования элементов в массиве на обратный.

Вариант 10

в одномерном массиве, состоящем из *n* вещественных элементов, вычислить:
1) произведение положительных элементов массива;
2) сумму элементов массива, расположенных до минимального элемента.
Упорядочить по возрастанию отдельно элементы, стоящие на четных местах, и элементы, стоящие на нечетных местах.

2. Обработка последовательностей структур.

Вариант 1

1. Описать структуру с именем STUDENT, содержащую следующие поля:
• фамилия и инициалы;
• номер группы;
• успеваемость (массив из пяти элементов).
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT;
• подсчитать количество всех студентов, включенных в массив, средний балл которых больше 4,0;
• если таких студентов нет, вывести соответствующее сообщение.

Вариант 2

1. Описать структуру с именем STUDENT, содержащую следующие поля:
• фамилия и инициалы;
• номер группы;
• успеваемость (массив из пяти элементов).
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT;
• подсчитать количество всех студентов, имеющих оценки 4 и 5;
• если таких студентов нет, вывести соответствующее сообщение.

Вариант 3

1. Описать структуру с именем STUDENT, содержащую следующие поля:
• фамилия и инициалы;
• помер группы;
• успеваемость (массив из пяти элементов).
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT;
• подсчитать количество всех студентов, имеющих хотя бы одну оценку 2;
• если таких студентов нет, вывести соответствующее сообщение.

Вариант 4

1. Описать структуру с именем AEROFLOT, содержащую следующие поля:
• название пункта назначения рейса;
• номер рейса;
• тип самолета.
2. Написать программу, выполняющую следующие действия:

• ввод с клавиатуры данных в массив, состоящий из семи элементов типа AEROFLOT;
• подсчитать количество записей из массива, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры;
• если таких рейсов нет, выдать на дисплей соответствующее сообщение.

Вариант 5

1. Описать структуру с именем AEROFLOT, содержащую следующие ноля:
• название пункта назначения рейса;
• номер рейса;
• тип самолета.
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из семи элементов типа AEROFLOT;
• подсчитать количество записей из массива, обслуживаемых самолетом, тип которого введен с клавиатуры;
• если таких рейсов нет, выдать на дисплей соответствующее сообщение.

Вариант 6

1, Описать структуру с именем WORKER, содержащую следующие поля:
• фамилия и инициалы работника;
• название занимаемой должности;
• год поступления на работу.
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из десяти структур типа WORKER;
• подсчитать количество работников из массива, чей стаж работы в организации превышает значение, введенное с клавиатуры;
• если таких работников нет, вывести на дисплей соответствующее сообщение.

Вариант 7

1. Описать структуру с именем TRAIN, содержащую следующие поля:
• название пункта назначения;
• номер поезда;
• время отправления.
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из восьми элементов типа TRAIN;
• подсчитать количество записей из массива о поездах, отправляющихся после введенного с клавиатуры времени;
• если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 8

1. Описать структуру с именем TRAIN, содержащую следующие поля:
• название пункта назначения;
• номер поезда;
• время отправления.
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из шести элементов типа TRAIN;
• подсчитать количество записей из массива о поездах, направляющихся в пункт, название которого введено с клавиатуры;
• если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 9

1. Описать структуру с именем TRAIN, содержащую следующие поля:
• название пункта назначения;
• номер поезда;
• время отправления.
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из восьми элементов типа TRAIN;
• вывод на экран информации о поезде, номер которого введен с клавиатуры;

• если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 10

1. Описать структуру с именем MARSH, содержащую следующие поля:
• название начального пункта маршрута;
• название конечного пункта маршрута;
• номер маршрута.
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из восьми элементов типа MARSH;
• подсчитать количество записей из массива о маршрутах, которые начинаются в пункте, название которого введено с клавиатуры;
• если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

Вариант 11

1. Описать структуру с именем MARSH, содержащую следующие поля:
• название начального пункта маршрута;
• название конечного пункта маршрута;
• номер маршрута.
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из восьми элементов типа MARSH
• подсчитать количество записей из массива о маршрутах, которые кончаются в пункте, название которого введено с клавиатуры;
• если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

Вариант 12

1. Описать структуру с именем NOTE, содержащую следующие поля:
• фамилия, имя;
• номер телефона;
• день рождения (массив из трех чисел).
2. Написать программу, выполняющую следующие действия:
• ввод с клавиатуры данных в массив, состоящий из восьми элементов типа NOTE;
• подсчитать количество записей из массива, у которых дата дня рождения совпадает с датой введенной с клавиатуры;
• если такого нет, выдать на дисплей соответствующее сообщение.

Контрольные вопросы:

1. Какие типы алгоритмов существуют в библиотеке STL?
2. Перечислите структуры для хранения данных, для которых можно применять алгоритмы STL.
3. Перечислите не модифицирующие алгоритмы, которые используют в качестве параметра функциональный объект.
4. Перечислите не модифицирующие алгоритмы, которые позволяют проводить анализ и обработку нескольких последовательностей.
5. Какие виды последовательностей можно обрабатывать не модифицирующими алгоритмами?