СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Пособие для самостоятельной работы студентов составлено для студентов специиальности 09.02.03 «Программирование в компьютерных системах». Представлены задачи и упражнения по языку Си и программированию на нем. Рассматриваемая версия Си соответствует международному и ANSI-стандарту этого языка.

Сборник задач и упражнений может быть использован как на занятиях во время закрепления изученного материала, та к и для самостоятельной внеаудиторной работы студентов.

В сборнике помимо упражнений по основным темам курса, приведено краткое описание заданий практикума, которые рекомендуется выполнить для закрепления пройденного материала и получения навыков в написании законченных программ.

Значительную часть сборника составляют приложения, где описана библиотека стандартных функций языка Си, некоторые системные функции ОС UNIX и фрагменты стандарта языка Си, связанные с правилами приведения типов и адресной арифметикой.

ТИПЫ, ОПЕРАЦИИ, ВЫРАЖЕНИЯ

2.1. Верно ли записаны константы, представляющие целочисленные значения? Для верно записанных констант определить их значение, тип.

123	1E6	123456789LU -5	0XFUL	
‘0’	058	‘\x7’	0X-1AD	‘\122’
00123	0xffffffL	01A	-‘x’	″x″
‘a’U	0731UL	‘\n’	+0xaf	0X0

2.2. Верно ли записаны константы с плавающей точкой? Для верно записанных констант определить их значение, тип.

1.71	1E-6	0.314159E1F	.005	0051E-04
5.E+2	0e0	0x1A1.5	05.5	0
0X1E6	0F	1234.56789L	1.0E-10D	3.1415U
1e-2f	-12.3E-6	+10e6	123456L	E-6

2.3. Верно ли записаны выражения? Для верно записанных выражений вычислить их значения (операции + - * / % =):

```
int a, b, c, d, e;
a = 2;  b = 13;  c = 7;  d = 19;  e = -4;
```

b / a / c	d / a % c	c % d-e	-e % a + b / a *-5+5
b % e	7-d%+(3-a)	b % - e * c	9 / c - - 20 / d

2.4. Верно ли решена задача: «значение целочисленной переменной с увеличить на 1; целочисленной переменной а присвоить значение, равное удвоенному значению переменной с ».

```
int a, c;  c = 5;
```

a). c ++ ; a = 2 * c;	b). a = 2 * c++ ;	c). c += 1; a = c + c;	d). a = c++ + c;

e). ++c; f). a = ++ c + c; g). a = c += 1 + c; h). a = (c+=1)+c;
a = c + c;

2.5. Верно ли решена задача: «значение целочисленной переменной с уменьшить на 1; целочисленной переменной а присвоить значение, равное частному от деления переменной с на 2».

int a, c; c = 5;

a). -- c ; b). a = -- c / 2; c). c -= 1; d). a = c -- / 2;
a = c / 2; a = c % 2;

e). a = c -= 1/2; f). a = (c = c - 1)/2; g). a = (c -= 1)/2; h). a=(c-= 1)/2.0;

2.6. Эквивалентны ли выражения?

a) E1 op= E2 и E1 = E1 op E2
b) E1 op= E2 и E1 = E1 op (E2)

Замечание: здесь E1, E2 - выражения допустимого в этом случае типа ; op - операция (одна из + - * / % >> << & ^ |).

2.7. Верно ли записаны выражения? Для верно записанных выражений вычислить их значения (операции + - * / ++ - - операции присваивания):

int a, b, c; a = 2; b = 6; c = 3;

- - - a	-- - a	b-- - a	a += a ++	++ b / a ++ * --c
a --- b	- a-- -b a ++ = b	a = a ++	b++ / ++a * c --	
- --a	a- --c	a ++ = a	++ a = b	a = (b + 1) ++

2.8. Верно ли записаны выражения? Для верно записанных выражений вычислить их значения, определить тип результата (операции + - * / % ++ операции отношения, операции присваивания):

int i, j, k, m; char c, d; i = 1; j = 2; k = -7; m = 0; c = 'w';

d = 'a'+1 < c	m = - i - 5 * j >= k+1	i + j++ + k = = -2*j
m = 3 < j < 5	m = 3 = = j < 5	m = = c = 'w'
m = c != 87	m = c = ! 87	m = ! c = 87
m = !c+87	! m = =c + 87	m ! = c + 87
k = = j - 9 = = i	k *= 3 + j	i + j = !k
i += ++ j + 3	k %= m = 1 + n / 2	1 + 3 * n += 7 / 5
1 + 3 * (n += 7) / 5	c + i < c - 'x'+10	i - k = = '0'+9 < 10

2.9. В логике справедливы утверждения:

not (not x) = x
x and true = x

Верны ли соответствующие утверждения для операций ! и && в Си? Ответ обосновать.

2.10. При любом вещественном y > 0 x < x + y математически верно. Верно ли подобное утверждение для выражения на Си?

2.11. Написать эквивалентное выражение, не содержащее операции **!**

! (a>b) ! (2∗a == b+4) ! (a<b && c<d)
! (a<2 || a>5) ! (a<1 || b<2 && c<3)

2.12. Пусть
char c; short s; int i; unsigned u; signed char sc;
float f; double d; long lng; unsigned short us; long double ld;
Определить тип выражений:
c - s / i u ∗ 3 - 3.0 ∗ u - i u - us ∗ i (sc + d) ∗ ld
(5 ∗ lng - ‘a’) ∗ (s + u / 2) (f + 3) / (2.5f - s ∗ 3.14)

2.13. Допустимо ли в Си? Если "да" - опишите семантику этих действий; если "нет" - объясните почему.

a). . . .
```
int  i;
i = (1 || 2) % (1 | 2 );
printf ( ″ i = %d\n″, i);
```

b). . . .
```
int  a, b, m, n, z;
m = n = 5;
z = a = b = 0;
z--, ( a = b ) = z + ( m != n );
printf (″%d %d %d %d %d\n″,
        a, b, m, n, z);
```

c). . . .
```
int  i = 1;
i = i << i | i;
printf ( ″ i = %d\n″, i);
```

d). . . .
```
double  x = 1.9; int a;
double  b = 3.7;
a = b += (1 && 2 || 3) != (int)x;
printf (″%f %d %f\n″, x, a, b);
```

e). . . .
```
int x;
x = 5; ++ x =10;
printf ("%d\n", x);
```

f). . . .
```
int i, x, y;  x = 5; y = 10; i = 15;
x = ( y = 0, i = 1);
printf("%d %d %d\n", i, x ,y);
( x = y == 0) , i=1;
printf("%d %d %d\n", i, x, y);
```

g). . . .
```
int x, y;
x = 5; y = x && ++ x;
printf("%d %d\n", x, y);
```

h). . . .
```
int x = 2, y, z;
x ∗= 3+2; x ∗= y = z = 4;
printf ("%d %d %d\n", x, y, z);
x = y == z; x == ( y = z );
printf ("%d %d %d\n", x, y, z);
```

i). . . .
```
int x = 2, y = 1, z = 0;
y = x && y || z;
x = x || !y && z;
z = x / ++x;
printf(" %d %d %d\n", x, y, z);
```

j). . . .
```
int x = 03, y = 02, z = 01;
printf("%d\n", x | y & -z);
printf("%d\n", x ^ y & -z);
printf("%d\n", x & y && z);
printf("%d\n", x<<3);
```

k). . . .
```
int x, y, z;  x = y = z = 1;
x += y += z;
printf("%d\n", x < y ? y++ : x++);
printf("%d\n", z+=x<y ? ++x : y--);
printf("%d %d %d\n", x, y, z);
```

l). . . .
```
int x, y, z, i;  x = y = z = 1;
i = ++x || ++y && ++z;
printf("%d%d%d%d\n", x,y,z,i);
i = x++ <= --y || ++z >= i;
printf("%d%d%d%d\n", x,y,z,i);
```

```
printf("%d\n", z>=y && y>=x);
```

2.14. Что будет напечатано в результате выполнения следующего фрагмента программы?

```
. . .
double d; float f; long lng; int i; short s;
s = i = lng = f = d = 100/3;
printf("s = %hd i = %d lng = %ld f = %f d = %f\n", s, i, lng, f, d);
d = f = lng = i = s =100/3;
printf("s = %hd i = %d lng = %ld f = %f d = %f\n", s, i, lng, f, d);
s = i = lng = f = d = 1000000/3;
printf("s = %hd i = %d lng = %ld f = %f d = %f\n", s, i, lng, f, d);
d = f = lng = i = s =1000000/3;
printf("s = %hd i = %d lng = %ld f = %f d = %f\n", s, i, lng, f, d);
lng = s = f = i = d =100/3;
printf("s = %hd i = %d lng = %ld f = %f d = %f\n", s, i, lng, f, d);
f = s = d = lng = i = (double)100/3;
printf("s = %hd i = %d lng = %ld f = %f d = %f\n", s, i, lng, f, d);
s = i = lng = f = d = 100/(double)3;
printf("s = %hd i = %d lng = %ld f = %f d = %f\n", s, i, lng, f, d);
f = s = d = lng = i = (double)100/3;
printf("s = %hd i = %d lng = %ld f = %f d = %f\n", s, i, lng, f, d);
i = s = lng = d = f = (double)(100/3);
printf("s = %hd i = %d lng = %ld f = %f d = %f\n", s, i, lng, f, d);
```

2.15. Что будет напечатано в результате выполнения следующего фрагмента программы?

```
double d = 3.2, x; int i = 2, y;
x = ( y = d / i ) * 2; printf ("x = %f ;y = %d\n", x, y);
x = ( y = d / i ) * 2; printf ("x = %d ;y = %f\n", x, y);
y = ( x = d / i ) * 2; printf ("x = %f ;y = %d\n", x, y);
y = d * ( x = 2.5 / d); printf ("x = %f; y = %d\n", x, y);
x = d * ( y = ( (int)2.9 + 1.1) / d; printf ("x = %d y = %f\n", x, y);
```

2.16. Дано вещественное число x . Не пользуясь никакими операциями, кроме умножения, сложения и вычитания, вычислить

$$2x^4-3x^3+4x^2-5x+6.$$

Разрешается использовать не более четырех умножений и четырех сложений и вычитаний.

2.17. Целой переменной k присвоить значение, равное третьей от конца цифре в записи целого положительного числа x.

2.18. Целой переменной k присвоить значение, равное сумме цифр в записи целого положительного трехзначного числа x.

2.19. Целой переменной k присвоить значение, равное первой цифре дробной части в записи вещественного положительного числа x.

2.20. Определить число, полученное выписыванием в обратном порядке цифр заданного целого трехзначного числа.

2.21. Идет n-ая секунда суток. Определить, сколько полных часов и полных минут прошло к этому моменту.

2.22. Дано вещественное число x . Не пользуясь никакими операциями, кроме умножения, получить

a) x^{21} за шесть операций
b) x^3 и x^{10} за четыре операции
c) x^5 и x^{13} за пять операций
d) x^2, x^5 и x^{17} за шесть операций
e) x^4, x^{12} и x^{28} за шесть операций

2.23. Выражения, соединенные операциями && и || , по правилам Си вычисляются слева направо; вычисления прекращаются, как только становится известна истинность или ложность результата. В других языках программирования, например в Паскале, вычисляются все части выражения в любом случае. Приведите «за» и «против» каждого из этих решений.

2.24. Почему в Си не допускается, чтобы один и тот же литерал-перечислитель входил в два различных перечислимых типа? Могут ли совпадать имена литералов-перечислителей и имена обычных переменных в одной области видимости? Могут ли разные литералы-перечислители иметь одинаковые значения?

2.25. «Упаковать» четыре символа в беззнаковое целое. Длина беззнакового целого равна 4.

2.26. «Распаковать» беззнаковое целое число в четыре символа. Длина беззнакового целого равна 4.

2.27. Заменить в целочисленной переменной x n бит, начиная с позиции p, n старшими инвертированными битами целочисленной переменной y.

2.28. Циклически сдвинуть значение целочисленной величины на n позиций вправо.

2.29. Циклически сдвинуть значение целочисленной величины на n позиций влево.

2.30. Выясните некоторые свойства и особенности поведения доступного Вам транслятора Си:

a) выяснить, сколько байт отведено для хранения данных типа short, int, long, float, double и long double;

b) выяснить способ представления типа char (signed- или unsigned- вариант);

c) проконтролировать, все ли способы записи констант допустимы:

- целых (обычная форма записи, u/U, l/L, их комбинации; запись констант в восьмеричной и шестнадцатиричной системах счисления)

- вещественных (обычная форма записи, в экспоненциальном виде, f/F, l/L, e/E)
- символьных (обычная форма записи, с помощью эскейп-последовательности) и строковых (в частности, происходит ли конкатенация рядом расположенных строковых констант)

d) выяснить, как упорядочены коды символов '0' - '9', 'a' - 'z', 'A' - 'Z', пробел (между собой и относительно друг друга);

e) проконтролировать, происходит ли инициализация переменных по умолчанию;

f) проверить, реагирует ли транслятор на попытку изменить константу;

g) исследовать особенности выполнения операции % с отрицательными операндами;

h) проверьте, действительно ли операции отношения == и != имеют более низкий приоритет, чем все другие операции отношения;

i) проверьте, действительно ли выполняется правило "ленивых вычислений" выражений в Си, т.е. прекращается ли вычисление выражений с логическими операциями, если возможно "досрочно" установить значение результата;

j) проверьте, все ли виды операнда операции sizeof (X), определяемые стандартом для арифметических типов, допускаются компилятором; действительно ли выражение X не вычисляется.

УПРАВЛЕНИЕ

3.1 Синтаксис и семантика операторов языка Си

3.1. Перечислить все ситуации, когда в программах на Си используется составной оператор.

3.2. В Си точка с запятой используется в качестве признака конца оператора; в Паскале - в качестве разделителя операторов. Сравните эти решения, сформулируйте возможные «за» и «против».

3.3. Эквивалентны ли следующие фрагменты программы:

```
if (e1) if (e2) S1; else S2;
if (e1) { if (e2) S1; else S2; }
if (e1) { if (e2) S1; } else S2;
if (e1)  if (e2) S1; else ; else S2;
if (e1)  if (e2) S1; else S2; else ;
```

Замечание: здесь e1 и e2 - выражения допустимого в этом случае типа; S1 и S2 - произвольные операторы.

3.4. Описать в виде блок-схемы семантику каждого оператора цикла в Си (с учетом операторов break и continue, которые, возможно, содержатся в теле цикла).

3.5. Может ли быть определено число итераций цикла for до начала его выполнения?

a) в Паскале
b) в Си

3.6. Верно ли решена задача: «найти сумму первых 100 натуральных чисел»?

a) i = 1; sum = 0;
   for (; i <= 100; i++) sum += i;
b) sum = 0;
   for (i = 1; i <= 100;) sum += i++;
c) for (i = 1, sum = 0; i <= 100; sum += i+, i++);
d) for (i = 1, sum = 0; i <= 100; sum += i++);
e) for (i = 0, sum = 0; i++, i <= 100; sum += i);

3.7. Выразить семантику цикла for с помощью цикла while. Эквивалентны ли полученные фрагменты программ?

3.8. Эквивалентны ли следующие фрагменты программы:

a) for (; e2 ;) S; и while (e2) S;
b) for (; ;) S; и while (1) S;

Замечание: здесь e2 - выражение допустимого в этом случае типа; S - произвольный оператор.

3.9. Можно ли написать фрагмент программы на Си, эквивалентный данному, используя один оператор цикла for с пустым оператором в качестве тела цикла?

```
i = 0; c = getchar();
while (c != ’  ’ && c != ’\n’ && c !=’ \t’ && c != EOF)
   { i++; c = getchar(); }
```

3.10. Сравнить семантику операторов repeat в Паскале и do-while в Си.

3.11. Улучшить стиль (структуру) следующих фрагментов программы на Си:

a)
```
while ( E1 )
      { if ( E2 ) continue; S; }
```
b)
```
do  { if ( E1 ) continue; else S1; S2; }
   while ( E2 );
```

Замечание: здесь E1, E2 - выражения допустимого в этом случае типа; S, S1, S2 - произвольные операторы.

3.12. Что напечатает следующая программа?

```
# include <stdio.h>
main()
  { int x, y, z;
   x = y = 0;
   while ( y < 10 ) ++y; x += y;
   printf ("x = %d y = %d\n", x, y);
   x = y = 0;
   while ( y < 10 ) x += ++ y;
   printf (" x= %d y = %d\n", x, y);
   y = 1;
   while ( y < 10 ) { x = y ++; z = ++y;}
   printf ("x = %d y = %d z = %d\n", x, y, z);
   for ( y =1; y < 10; y++ ) x = y;
   printf (" x= %d y = %d\n", x, y);
```

```
  for ( y = 1; ( x = y ) < 10; y++ );
  printf ("x = %d y = %d\n", x, y);
  for ( x = 0, y = 1000; y > 1; x++, y /= 10 )
  printf ("x = %d y = %d\n", x, y);
}
```

3.13. Сравнить семантику операторов case в Паскале и switch в Си.

3.14. Верны ли следующие утверждения:

a) «любое выражение в Си может быть преобразовано в оператор добавлением к нему точки с запятой (;) »

b) «пустой оператор в Си - это отсутствие каких-либо символов в том месте конструкции, где по синтаксису может находиться оператор»

c) «составной оператор (блок) в Си - это совокупность операторов, заключенная в фигурные скобки { }»

d) «оператор присваивания в Си - это выражение вида

переменная = выражение»

e) «тип выражения в условии в операторе if , в условии завершения цикла в операторах цикла может быть скалярным (т.е. любым целочисленным, любым вещественным либо указателем)»

f) «в блоке описания/объявления и операторы могут располагаться в любом порядке; единственное требование - использование не должно предшествовать описанию/объявлению»

e) «цикл for (; ;) является бесконечным циклом, и поэтому его использование в Си запрещено»

f) «семантика операторов цикла while и do-while в Си различается только тем, что тело цикла while может не выполниться ни разу, а тело цикла do-while выполнится хотя бы один раз.»

g) «каждая ветвь в операторе switch должна быть помечена одной или несколькими различными целочисленными константами или константными выражениями»

h) «если в операторе switch нет ветви default, то значение выражения выбора должно совпадать с одной из констант ветвей case»

i) «в операторе switch ветви case и ветвь default можно располагать в любом порядке»

3.15. Верно ли утверждение: « действие оператора continue; в приведенных ниже примерах эквивалентно действию оператора go to next; ».

```
a) while ( E ) { S; ... continue; ... S; next: ; }
b) do { S; ... continue; ... S; } while ( E ); next: ; ...
c) for ( E1; E2; E3) { S; ... continue; ... S; next: ; }
d) while ( E) { S; ... for (E1;E2;E3) { S; ... continue; ... S; } ... S; next:; }
e) while ( E) { S; ... for (E1;E2;E3) { S; ... continue; ... S; next: ; } ... S;}
f) switch ( E ) { case C1: S;
                  case C2: S; continue;
                  case C3: S; }
   next:; ...
g) switch ( E ) {  case C1: S;
                   case C2: S; continue;
                   case C3: next: S; }
```

h) next: switch (E) { case C1: S;
case C2: S; continue;
case C3: S; }

Замечание: здесь E, E1, E2, E3 - выражения допустимого в этом случае типа ; S - произвольный оператор; C1, C2 C3 - константы подходящего типа.

3.16. Верно ли утверждение: « действие оператора break; в приведенных ниже примерах эквивалентно действию оператора go to next; ».

a) while (E) { S; ... break; ... S; next: ; }
b) while (E) { S; ... break; ... S; } next: ; ...
c) do { S; ... break; ... S; } while (E); next: ; ...
d) for (E1; E2; E3) { S; ... break; ... S; next: ; }
e) while (E) { S; ... for (E1; E2; E3) { S; ... break; ... S; } next: ; ... S; }
f) while (E) { S; ... for (E1; E2; E3) { S; ... break; ... S; } ... S; next: ; }
g) while (E) { S; ... for (E1; E2; E3) { S; ... break; ... S; } ... S; } next: ;
h) switch (E) { case C1: S;
case C2: S; break;
case C3: S; }
next:; ...
i) switch (E) { case C1: S;
case C2: S; break; S;
case C3: next: S; }

Замечание: здесь E, E1, E2, E3 - выражения допустимого в этом случае типа; S - произвольный оператор; C1, C2 C3 - константы подходящего типа.

3.2 Обработка числовых данных

Замечание: при решении некоторых задач этого раздела необходимы минимальные знания о «стандартном» вводе и выводе целых и вещественных чисел.

3.17. Для данных чисел a, b и c определить, сколько корней имеет уравнение $ax^2+bx+c = 0$, и распечатать их. Если уравнение имеет комплексные корни, то распечатать их в виде $v \pm iw$.

3.18. Подсчитать количество натуральных чисел n ($111 \le n \le 999$), в записи которых есть две одинаковые цифры.

3.19. Подсчитать количество натуральных чисел n ($102 \le n \le 987$), в которых все три цифры различны.

3.20. Подсчитать количество натуральных чисел n ($11 \le n \le 999$), являющихся палиндромами, и распечатать их.

3.21. Подсчитать количество цифр в десятичной записи целого неотрицательного числа n.

3.22. Определить, верно ли, что куб суммы цифр натурального числа n равен n^2.

3.23. Определить, является ли натуральное число n степенью числа 3.

3.24. Для данного вещественного числа a среди чисел 1, 1 + (1/2), 1 + (1/2) + (1/3), ... найти первое, большее a.

3.25. Для данного вещественного положительного числа a найти наименьшее целое положительное n такое, что 1 + 1/2 +1/3+ ... + 1/n > a.

3.26. Даны натуральное число n и вещественное число x. Среди чисел $\exp(\cos(x^{2k}))\sin(x^{3k})$ (k = 1, 2, ..., n) найти ближайшее к какому-нибудь целому.

3.27. Дано натуральное число n. Найти значение числа, полученного следующим образом: из записи числа n выбросить цифры 0 и 5, оставив прежним порядок остальных цифр.

3.28. Дано натуральное число n. Получить все такие натуральные q, что n делится на q^2 и не делится на q^3.

3.29. Дано натуральное число n. Получить все его натуральные делители.

3.30. Дано целое число m > 1.Получить наибольшее целое k, при котором $4^k <$ m.

3.31. Дано натуральное число n. Получить наименьшее число вида 2^r, превосходящее n.

3.32. Распечатать первые n простых чисел (p - простое число, если p >= 2 и делится только на 1 и на себя).

3.33. Даны вещественные числа x и y (x > 0, y > 1). Получить целое число k (положительное, отрицательное или равное нулю), удовлетворяющее условию $y^{k-1} \le x < y^k$.

3.34. Распечатать первые n чисел Фибоначчи ($f_0 = 1$; $f_1 = 1$; $f_{k+1} = f_{k-1} + f_k$; k = 1, 2, 3,...)

3.35. Вычислить с точностью eps > 0 значение «золотого сечения» - $0.5*(1+\sqrt{5})$ - предел последовательности $\{ q_i \}$ при $i \to \infty$

$q_i = f_i / f_{i-1}$, i = 2, 3,...где f_i - числа Фибоначчи (см. предыдущую задачу).
Считать, что требуемая точность достигнута, если $| q_i - q_{i+1}| <$ eps.

3.36. Распечатать числа Фибоначчи (см. задачу 3.34), являющиеся простыми числами со значениями меньше n.

3.37. Вычислить с точностью eps > 0 значение числа **e** - предел последовательности $\{ x_i \}$ при $i \to \infty$

$x_i = (1+1/i)^i$, i = 1, 2, ...
Считать, что требуемая точность достигнута, если $| x_i - x_{i+1}| <$ eps.

3.38. Вычислить значение $\sum i!$ для i, изменяющихся от 1 до n. Воспользоваться соотношением $\sum i! = 1 + 1*2 + 1*2*3 +...+ 1*2*3*...*n = 1+2*(1+3*(1+ \ +n*(1)...))$.

3.39. Пусть a_0 и b_0 - положительные вещественные числа. Соотношениями $a_{n+1} = \sqrt{(a_n b_n)}$; $b_{n+1} = (a_n+b_n) / 2$ при n = 0, 1, 2, ... задаются две бесконечные числовые последовательности $\{a_n\}$и $\{b_n\}$, которые сходятся к общему пределу $M(a_0,b_0)$, называемому арифметико-геометрическим средним чисел a_0 и b_0. Найти приближенное значение $M(a_0,b_0)$ с точностью eps > 0. Поскольку при $a_0 < b_0$ $a_i < b_i$ и, более того, $a_0 < a_1 < ... < a_i < ... bi < ... < b_1 < b_0$, то в качестве подходящего критерия прекращения вычислений можно использовать соотношение $| a_i - b_i | <$ eps.

3.40. Вычислить квадратные корни вещественных чисел x = 2.0, 3.0, ... , 100.0. Распечатать значения x, $\sqrt{x}$, количество итераций, необходимых для вычисления корня с точностью eps > 0.

Для a > 0 величина $\sqrt{a}$ вычисляется следующим образом:

$$a_0 = 1; a_{i+1} = 0.5*(a_i+a/a_i) \ i = 0, 1, 2,...$$

Считать, что требуемая точность достигнута, если $| a_i-a_{i+1}| <$ eps.

3.41. Найти приближенное значение числа π с точностью eps > 0. Для этого можно использовать представление числа $2/\pi$ в виде произведения корней $\sqrt{(1/2)} *\sqrt{(1/2+1/2\sqrt{(1/2)})}*\sqrt{(1/2+ 1/2\sqrt{(1/2+1/2\sqrt{(1/2)})})}*...$. Вычисления прекращаются, когда два следующих друг за другом приближения для числа π будут отличаться меньше, чем на eps.

3.42. Для данного вещественного числа x и натурального n вычислить:

a) $\sin x + \sin^2 x + ... + \sin^n x$

b) $\sin x + \sin x^2 + ... + \sin x^n$

c) $\sin x + \sin(\sin x) + ... + \sin (\sin (... \sin(\sin x) ...))$

3.43. Алгоритм Евклида нахождения наибольшего общего делителя (НОД) неотрицательных целых чисел основан на следующих свойствах этой величины: пусть m и n - одновременно не равные нулю целые неотрицательные числа и $m \geq n$. Тогда, если n = 0, то НОД(n, m) = m, а если $n \neq 0$, то для чисел m, n, и r, где r - остаток от деления m на n, выполняется равенство НОД(m, n) = НОД(n, r). Используя алгоритм Евклида, определить наибольший общий делитель неотрицательных целых чисел a и b.

3.44. Вычислить 1 - 1/2 + 1/3 - 1/4 + ...+1/9999 - 1/10000 следующими способами:

a). последовательно слева направо;

b). последовательно справа налево;

c). последовательно слева направо вычисляются 1 +1/3 + 1/5 + ... + 1/9999 и 1/2 + 1/4 + ... + 1/10000, затем второе значение вычитается из первого;

d). последовательно справа налево вычисляются 1 +1/3 + 1/5 + ... + 1/9999 и 1/2 + 1/4 + ... + 1/10000, затем второе значение вычитается из первого.

Сравнить и объяснить полученные результаты.

3.45. Натуральное число называется совершенным, если оно равно сумме всех своих делителей, за исключением самого себя. Дано натуральное число n. Получить все совершенные числа, меньшие n.

3.46. Определить, является ли число простых чисел, меньших 10000, простым числом.

3.47. Если p и q - простые числа и q = p+2, то они называются простыми сдвоенными числами или “близнецами” (twin primes). Например, 3 и 5 - такие простые числа. Распечатать все простые сдвоенные числа, меньшие N.

3.3 Обработка символьных данных

Замечание: при решении некоторых задач этого раздела необходимы минимальные знания о «стандартном» вводе и выводе литер.

3.48. Пусть во входном потоке находится последовательность литер, заканчивающаяся точкой (кодировка ASCII):

a) определить, сколько раз в этой последовательности встречается символ ‘a’;

b) определить, сколько символов ‘e’ предшествует первому вхождению символа ‘u’ (либо сколько всего символов ‘e’ в этой последовательности, если она не содержит символа ‘u’);

c) выяснить, есть ли в данной последовательности хотя бы одна пара символов-соседей ‘n’ и ‘o’, т.е. образующих сочетание ‘n’ ‘o’ либо ‘o’ ‘n’;

d) выяснить, чередуются ли в данной последовательности символы ‘+’ и ‘-‘, и сколько раз каждый из этих символов входит в эту последовательность;

e) выяснить, сколько раз в данную последовательность входит группа подряд идущих символов, образующих слово *C++*;

f) выяснить, есть ли среди символов этой последовательности символы, образующие слово char;

g) выяснить, есть ли в данной последовательности фрагмент из подряд идущих литер, образующий начало латинского алфавита (строчные буквы), и какова его длина. Если таких фрагментов несколько, найти длину наибольшего из них. Если такого фрагмента нет, то считать длину равной нулю;

h) выяснить, есть ли в данной последовательности фрагменты из подряд идущих цифр, изображающие целые числа без знака. Найти значение наибольшего из этих чисел. Если в этой последовательности нет ни одной цифры, то считать, что это значение равно нулю;

i) определить, имеет ли данная последовательность символов структуру, которая может быть описана с помощью следующих правил:

последовательность ::= слагаемое + последовательность | слагаемое
слагаемое ::= идентификатор | целое
идентификатор ::= буква | идентификатор буква | идентификатор цифра
буква ::= A | B | C | D | E | F | G | H | I | J | K
цифра ::= 0 | 1 | 2 | 3 | 4 | 5
целое ::= цифра | целое цифра

3.49. Пусть во входном потоке находится последовательность литер, заканчивающаяся точкой (кодировка ASCII). Вывести в выходной поток последовательность литер, измененную следующим образом:

a) заменить все символы ‘?’ на’!’;

b) удалить все символы ‘-‘ и удвоить все символы ‘&’;

c) удалить все символы, не являющиеся строчными латинскими буквами;

d) заменить все прописные латинские буквы строчными (другие символы копировать в выходной поток без изменения);

e) заменить все строчные латинские буквы прописными (другие символы копировать в выходной поток без изменения);

f) каждую группу рядом стоящих символов ‘+’ заменить одним таким символом;

g) каждую группу из n рядом стоящих символов ‘*’ заменить группой из n/2 рядом стоящих символов ‘+’ (n >= 2); одиночные ‘*’ копировать в выходной поток без изменения;

h) удалить из каждой группы подряд идущих цифр все начальные незначащие нули (если группа состоит только из нулей, то заменить эту группу одним нулем);

i) удалить все комбинации символов the;

j) оставить только те группы цифр, которые составлены из подряд идущих цифр с возрастающими значениями; все остальные цифры и группы цифр удалить (другие символы копировать в выходной поток без изменения);

k) заменить все комбинации символов child комбинациями символов children;

l) удалить группы символов, расположенные между фигурными скобками { и }. Скобки тоже должны быть удалены. Предполагается, что скобки сбалансированы, и внутри каждой пары скобок других фигурных скобок нет.

3.50. Пусть во входном потоке находится последовательность литер, заканчивающаяся маркером конца $ (кодировка ASCII). Вывести в выходной поток последовательность литер, измененную следующим образом:

a) удалить из каждой группы подряд идущих цифр, в которой более двух цифр и которой предшествует точка, все цифры, начиная с третьей (например, a+12.3456-b-0.456789+1.3-45678 преобразуется в a+12.34-b-0.45+1.3-45678);

b) удалить из каждой группы цифр, которой не предшествует точка, все начальные нули (кроме последнего, если за ним идет точка либо в этой группе нет других цифр, кроме нулей ; например, a-000123+bc+0000.0008-0000+0001.07 преобразуется в a-123+bc+0.0008-0+1.07).

ФУНКЦИИ И СТРУКТУРА ПРОГРАММЫ

4.1. Перечислите все существенные изменения, внесенные стандартом ANSI в правила объявления и описания функций. Какова цель этих изменений?

4.2. Перечислить все случаи, когда в Си используется тип void. Дать определение этого типа.

4.3. Перечислить классы памяти, определенные в Си. Что определяет класс памяти? В каких случаях и каким образом класс памяти определяется по умолчанию? Привести примеры.

4.4. Определен ли в Си класс памяти для функций? Если определен, то каким образом; если нет, то почему.

4.5. Допустима ли в Си вложенность функций? Можно ли в Си каким-то образом управлять видимостью функций?

4.6. Объяснить, чем различаются описание (объявление, declaration) и определение (definition) – по терминологии Б. Кернигана и Д. Ритчи [1, см. стр.71]. Привести примеры.

4.7. Эквивалентны ли следующие объявления функций:

a) double f ( );	и	double f (void);
b) char g (int i, char c);	и	char g (int, char);
c) h (double x);	и	int h (double x);
d) void h (int);	и	h (int);
e) extern int q (int);	и	int q (int);
f) static void s (char c);	и	void s (char c);

4.8. Определить, какие конструкции являются определениями, а какие описаниями; где они могут располагаться и что обозначают:

int i;	char c = ‘a’;	extern int f (int, char);
static int j;	register int b;	double g( ) { return 3.141592; };
extern long k;	int h (int i);	static char q(int, double);
auto short n;	s();	static void p(int i) { };

4.9. Верны ли следующие утверждения:

a) «тип выражения в операторе return должен совпадать с типом результата функции»

b) «функция, которая не возвращает результата (тип результата void), может не содержать оператор return;»

c) «функция, которая возвращает результат, может не содержать оператор return E; но вызывает другую функцию, которая содержит такой оператор»

d) «функция, которая возвращает результат, может содержать несколько операторов return E; »

e) «в Си аргументы функции всегда передаются по значению»

f) «в теле одной функции могут находиться два разных оператора, помеченных одинаковыми метками, если эти операторы находятся в разных блоках»; например,

```
void f(void)
  { ... label: S1; ...
       { ... label: S2; ... goto label; ...}
    ... goto label; ...
  }
```

g) «в Си нельзя использовать две (или более) взаимно рекурсивных функций»; например, если есть void f(void){... g();...} и void g(void){...f();...}, то программа, использующая такие функции, будет ошибочной.

h) «в Си можно войти в блок, минуя его заголовок; при этом память под локальные переменные, описанные в этом блоке, будет отведена, но инициализация (если она есть) выполняться не будет»

i) «любая функция, описанная в каком-либо файле, входящем в состав программы, может быть использована в этом и любом другом файле этой программы»

j) «в этом фрагменте программы нет ошибок »

```
#include <stdio.h>
int f(void) { return 100;}
void g(void) { printf("O.K.\n");}
main()
   { int i, j;
     i = f();
     j = g(), f();
     g(); f();
     printf("i=%d, j=%d f=%d\n", i, j, f());
   }
```

4.10. В каких случаях в Си возможна инициализация переменных? Когда она происходит? Как определяется инициализация по умолчанию?

4.11. Допустимо ли в Си? Если "да" - опишите семантику этих действий; если "нет" - объясните почему.

a)
```
#include <stdio.h>
main()
   { int i; int sum = 0;
     for ( i = 1; i <= 3; i++)
         { sum += i;
             { int i;
               for ( i = 1; i <= 5; i++)
                    sum *= i;
             }
         }
     printf("sum=%d\n", sum);
   }
```

b)
```
#include <stdio.h>
main()
   { int i; int sum = 0;
     for ( i = 1; i <= 3; i++)
         { sum += i;
             { for ( i = 1; i <= 5; i++)
                  sum *= i;
             }
         }
     printf("sum=%d\n", sum);
   }
```

c)
```
#include <stdio.h>
main()
   { double x;
     scanf("%f", &x);
     if ( x >= 0 ) goto ok;
         { double y = 5.0;
           x = x ? x : -x;
           ok: y+=sqrt(x);
           printf("y = %f\n",y);}
         }
   }
```

d)
```
#include <stdio.h>
main()
   { double x;
     scanf("%f", &x);
     if ( x >= 0 ) goto ok;
         { double y;
           x = x ? x : -x;
           ok: y = sqrt(x);
           printf("y = %f\n",y);
         }
   }
```

4.12. Допустимо ли в Си? Если "да" - опишите семантику этих действий; если "нет" - объясните почему.

a) файл **f1:**
```
#include<stdio.h>
```

b) файл **f1:**
```
#include<stdio.h>
```

```
main()
   { extern int f(int);
     int i;
     i = f(g(5));
     printf("i = %d\n", i);
   }
int g(int k)
   { k++;
    return f(k);
   }
```

файл **f2:**

```
int f(int i) { return i*i; }
```

```
static int f(void)
   { int k;
     scanf("%d", &k); return k; }
extern int g(int);
main()
   { int i;
     i = f(); j = g(i);
     printf("i=%d,j=%d\n", i,j);
    }
```

файл **f2:**

```
extern int f(void);
int g(int i)
  { return  f()+i; }
```

c) файл **f1:**

```
#include<stdio.h>
 extern int i; extern void f(void);
 main()
    { f();
      printf("i = %d\n", i);
      f();
      printf("i = %d\n", i);
    }
```

файл **f2:**

```
int i = 1;
void f(void) { i++; }
```

d) файл **f1:**

```
#include<stdio.h>
 extern int f(void);
 int i;
 main()
    { printf("res = %d\n", i+f()+f());
     }
```

файл **f2:**

```
int f(void)
   { static int i = 10;
     return i--;
    }
```

4.13. Что напечатает следующая программа?

```
a). #include <stdio.h>
   int i = 0; void f(void);
   main()
       { int i =1;
         printf("i1 = %d\n", i);
         f();
            { int i = 2; printf("i4 = %d\n", i);
                 { i +=1; printf("i5 = %d\n", i); }
              printf("i6 = %d\n", i);
            }
         printf("i7 = %d\n", i);
       }
   void f(void)
      { printf("i2 = %d\n", i);
        i = i + 10; printf("i3 = %d\n", i); }
```

```
b). #include <stdio.h>
  int f( int ); int b = 10;
   main()
      { int c = 3;
        b = f(c);
        printf("c=%d b=%d\n",c,b);
       }
    int f( int b )
      { int c = 5;
        c--; b++;
        printf("c=%d b=%d\n",c,b);
        return c+b;
       }
```

```
c). #include <stdio.h>
   char g ( char c);
   int f ( int i, char c)
```

```
d). #include <stdio.h>
  int abc( int ); int x;
   main()
```

```
    { int k = i+4; char b = g(c) + i;
      printf("c1 = %c k0 = %d\n", c, k);
        { int i = 3; k += i;
          printf("i1 = %d k1 = %d\n", i, k);
          c = g('b'); printf("c2 = %c\n", c);
        }
      i++; printf("i2 = %d k2 = %d\n", i, k);
      return i*(b-c); }
  char g(char c)
     { c = c + 1; return c; }
  main()
      { int k =2; char c = 'a'; k = f (k, c);
        printf("c3 = %c k3 = %d\n", c, k);
      }
```

e).

```
#include <stdio.h>
  int i = 1;
  int reset ( void );
  int next ( int );
  int last ( int );
  int new ( int );
  main()
    { int i, j; i = reset();
      for ( j = 1; j <= 3; j++ )
         { printf("i=%d j=%d\n", i, j);
           printf("%d %d\n", next(i), last(i));
           printf("%d\n", new(i+j));
          } }
int reset ( void ) { return i; }
  int next ( int j ) { return j = i++; }
  int last ( int j )
     { static int i = 10; return j = i--; }
  int new ( int i )
     { int j = 10; return i = j += i; }
```

```
    { int b;
      b = abc(x);
      printf("b1=%d x1=%d\n",b,x);
      x = abc(b);
      printf("b2=%d x2=%d\n",b,x);
      }
 int abc( int b )
   { static int x = 5;
     x += 7; b++;
     printf("b0=%d x0=%d\n",b,x);
     return x+b;
   }
```

f).

```
#include <stdio.h>
int i = 1;
int reset ( void );
int next ( void );
int last ( void );
int new ( int );
main()
   { int i, j; i = reset();
     for ( j = 1; j <= 3; j++ )
       { printf("i=%d j=%d\n", i, j);
         printf("%d\n", next(),);
         printf("%d\n", last());
         printf("%d\n", new(i+j)); } }
```

в файле **f1**:

```
static int i =10;
int next ( void ) { return i += 1; }
int last ( void ) { return i -= 1; }
int new ( int i)
   { static int j = 5; return i=j+=i; }
```

в файле **f2**:

```
extern int i;
int reset ( void ) { return i; }
```

4.14. Программа. Описать рекурсивную функцию вычисления n! - факториала числа n, основанную на соотношении n! = n*(n-1)!. С ее помощью найти факториалы натуральных чисел от1 до 10.

4.15. Программа. Описать рекурсивную функцию вычисления x^n для вещественного x ($x \neq 0$) и целого n:

$$x^n = \begin{cases} 1 & \text{при } n = 0 \\ 1/x^{|n|} & \text{при } n < 0 \\ x * x^{n-1} & \text{при } n > 0 \end{cases}$$

Протестировать эту функцию на подходящих наборах входных данных.

4.16. Программа. Описать рекурсивную функцию вычисления НОД(n,m) - наибольшего общего делителя неотрицательных целых чисел n и m, основанную на соотношении НОД(n,m) = НОД(m,r), где r - остаток от деления n на m (см. задачу 3.43). С ее помощью найти наибольший общий делитель натуральных чисел a и b. Сравнить эффективность рекурсивной и нерекурсивной функций вычисления НОД.

4.17. Программа. Описать рекурсивную функцию вычисления НОД (n_1, n_2, n_3, ..., n_m), воспользовавшись для этого соотношением: НОД (n_1, n_2, n_3, ... , n_k) = НОД (НОД (n_1, n_2, n_3, ... , n_{k-1}), n_k), k = 3, 4, ... , m. С ее помощью найти НОД (a_1, a_2, a_3, ..., a_{10}).

4.18. Программа. Описать рекурсивную функцию вычисления n–ого числа Фибоначчи: $f_0 = 1$; $f_1 = 1$; $f_{j+1} = f_{j-1} + f_j$; j = 1, 2, 3,….С ее помощью вычислить 100-ое число Фибоначчи.

4.19. Программа. Описать рекурсивную функцию вычисления значения A(n,m) - функции Аккермана для неотрицательных целых чисел n и m:

$$A(n,m) = \begin{cases} m+1 & \text{если } n = 0 \\ A(n-1,1) & \text{если } n \neq 0,\ m = 0 \\ A(n-1, A(n, m-1)) & \text{если } n>0,\ m>0 \end{cases}$$

С помощью этой функции найти значение A(5,8).

УКАЗАТЕЛИ И МАССИВЫ

5.1. Допустимо ли в Си? Если "да" - опишите семантику каждого правильного действия (не принимая во внимание ошибочные); если "нет" - объясните почему.

a). . . .

```
int i,* p, j, *q;
p = &i;          q = &p;
j = *p = 1;              q = p-1;        *p += 1;
i = *++q + *p; q -= 1;          i = *q ++ + *q;
printf("i=%d, j=%d, *p=%d, *q=%d \n", i, j, *p, *q);
```

b) . . .

```
int x = 1, y; char c = 'a';
int *pi, *qi; char *pc;
pi = &x;        *pi = 3;        y = *pi;        *pi = c;        qi = pi;
pc = qi;        *qi+=1;         pi++;           *(- - pi) = 5;  y = *qi+1;
pc = &c;        ++*pc;          (*pc)++;        *pc++; *pc+=1;
x = (int)pi;    pi=(int*)pc;    pi=(int*)x;     x = 1+ *pi;     pc=(char*)pi;
c = *pc;        pc = &y;        x = qi – pi;    qi = 0;         qi+=pi;
y = &pi;        y = (int)&pi;   pi = pi +5;     *(pi+1)=0;      pi=&(x+0);
```

5.2. К любому ли объекту в Си можно применять операцию взятия адреса & ?

5. 3. Допустимо ли в Си? Если "да" - опишите семантику этих действий; если "нет" - объясните почему.

a) int i = 2; const int j = 5;

```
int *pi;
const int *pci;
int *const cpi;
const int * const cpci;
pi = &i;        pci = &j;        cpi = &i;        cpci = &j;       pci = &i;
pi = (int*)&j;                   i = *pci + *pi;                   *pci = 3;
*pi = 3;        i=*pci++;        *(cpi++)=5;      *cpi++;
```

b)
```
int f(const int i, int j) { j++; return i+j; }
main()
  { int a, b; const int c = 5;
    scanf("%d", &a);
    b = f(c,a); printf("a=%d, b=%d, c=%d \n", a, b, c);
    b = f(c,c); printf("a=%d, b=%d, c=%d \n", a, b, c);
    b = f(a,a); printf("a=%d, b=%d, c=%d \n", a, b, c);
    b = f(a,c); printf("a=%d, b=%d, c=%d \n", a, b, c);
  }
```

5.4. Пусть целочисленный массив a соддержит 100 элементов. Верно ли решена задача: "написать фрагмент программы, выполняющий суммирование всех элементов массива a".

a)
```
int a[100], sum, i;
sum = 0;
for ( i = 0; i < 100; ++i ) sum += a[i];
```

b)
```
int a[100], *p, sum;
sum = 0;
for ( p = a; p < &a[100]; ++p ) sum = sum + *p;
```

c)
```
int a[100], *p, sum;
sum = 0;
for ( p = &a[0]; p < &a[100]; p++ ) sum += *p;
```

d)
```
int a[100], sum, i;
sum = 0;
for ( i = 0; i < 100; ++i ) sum += *(a+i);
```

e)
```
int a[100], sum, i;
sum = 0;
for ( i = 0; i < 100; ++a, ++i ) sum += *a;
```

f)
```
int a[100], *p, sum, i;
sum = 0;
for ( i = 0, p = a; i < 100; ++i ) sum += p[i];
```

g)
```
int a[100], *p, sum, i;
sum = 0;
for ( i = 0, p = a; i < 100; ++i ) sum += *(p+i);
```

5.5. Допустимо ли в Си? Если "да" - опишите семантику каждого правильного действия (не принимая во внимание ошибочные); если "нет" - объясните почему.

a)
```
. . .
int  a[5] = { 1, 2, 3, 4, 5 };
int  *p, x, *q , i;
p = a + 2;        * (p+2) = 7;
```

```
   *a += 3;          q=*&p-1;
   x = ++ p - q ++;          x += ++ *p;     x=*p-- + *p++;
   for (i = 0; i < 5; i++) printf("a [%d]=%d", i, a[ i ] ); printf("\n");
   printf("x=%d, *p=%d, *q=%d \n", x, *p, *q);
```

b) . . .

```
   char *str = "abcdef";
   char *p, *q, *r; int k;
   p = str; q = 0;              p++;
   k = p - str;       r = p+k;
   if ( k && p || q ) q = str + 6;
   p = q ? r : q;               *(p-1) = 'a';    *r = 'x';
   printf("str: %s\n", str);
```

c) . . .

```
   char s[ ] = "0123456";
   int *pi; char *pc1, *pc2;
   pc2 = s;
   pc1 = s + *(s+strlen(s) - 1) - '0';
   pi = ( int* ) pc2;          *pc1-- = '8';
   if ( pc1 - pc2 < 3 ) pc1 = pc2 = pi; else pc1 = ( pc1+pc2 )/2;
   if ( s == pc2 ) *pc1 = *pc2 + 1; else *pc1 = '9';
   printf("s: %s\n", s);
```

d) . . .

```
   int i; char *c; int *pi;
   i = 'a';
   pi = &i;          c = (char*)pi + 3;       printf("c1=%c", *c);
   i <<= 8;          c--;                     printf("c2=%c\n", *c);
```

e) . . .

```
   char c1, c2; short i;
   char *pc; short *ps;
   c1 = '1';         c2 = '2';         ps =&i;
   pc = (char*)ps;          *pc = c1;        pc++;            *pc = c2;
   printf("i = %hd\n", i);
```

5.6. Эквивалентны ли следующие фрагменты программы на Си?

a[i] /= k+m	и	a[i] = a[i]/k+m
a[i] /= k+m	и	a[i] = a[i]/(k+m)
a[i++]+=3	и	a[i++] = a[i++]+3
a[i++]+=3	и	a[i] = a[i++]+3
a[i++]+=3	и	a[i++] = a[i]+3
a[i++]+=3	и	a[i] = a[i]+3; i++;

5.7. Что напечатает следующая программа?

```
#include <stdio.h>
char str[ ] = "SSSWILTECH1\1\11W\1WALLMP1";
main()
   { int i, c;
     for ( i = 2; ( c = str [ i ] ) != '\0'; i++) {
           switch (c) {
```

```
                case ‘a’: putchar(‘i’); continue;
                case ‘1’: break;
                case 1: while ( ( c = str [++ i ] ) != ‘\1’ && c != ‘\0’);
                case 9: putchar(‘S’);
                case ‘E’: case ‘L’: continue;
                default: putchar(c); continue; }
              putchar(‘ ’); }
        putchar(‘\n’);
       }
```

5.8. Что напечатает следующая программа?

```
#include <stdio.h>
int a[ ] = { 0, 1, 2, 3, 4 };
main()
  { int i, *p;
    for ( i = 0; i <= 4; i++ ) printf("a[ i ]=%d  ", a[ i ]);  printf("\n");
    for ( p = &a[0]; p <= &a[4]; p++ ) printf("*p=%d  ", *p);  printf("\n");
   for ( p = &a[0], i = 0 ; i <= 4; i++ ) printf("p[ i ]=%d  ", p[ i ]);  printf("\n");
    for ( p = a, i = 0; p+i <= a+4; i++ ) printf("* (p+i)=%d ", * (p+i));
    printf("\n");
    for ( p = a+4; p >= a; p-- ) printf("*p=%d  ", *p );  printf("\n");
    for ( p = a+4, i=0; i <= 4; i++ ) printf("p[ -i ]=%d  ", p[ -i ]);
    printf("\n");
    for ( p = a+4; p >= a; p -- ) printf("a[ p - a ]=%d  ", a[ p - a ]);
    printf("\n");
  }
```

5.9. Что напечатает следующая программа?

```
#include <stdio.h>
int a[ ] = { 8, 7, 6, 5, 4 };
int  *p[ ] = { a, a+1, a+2, a+3, a+4 };
int  **pp = p;
main()
   { printf("*a=%d  **p=%d  **pp=%d\n", *a, **p, **pp );
     pp++;
     printf("pp-p=%d  *pp-a=%d  **pp=%d\n", pp-p, *pp-a, **pp );
     ++**pp;
     printf("pp-p=%d  *pp-a=%d  **pp=%d\n", pp-p, *pp-a, **pp );
     pp = p;
     ++**pp;
     printf("pp-p=%d  *pp-a=%d  **pp=%d\n", pp-p, *pp-a, **pp );
   }
```

5.10. Что напечатает следующая программа?

```
#include <stdio.h>
int a[ 3 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
int  *pa[ 3 ] = { a[ 0 ], a[ 1 ], a[ 2 ] };
int  *p = a[ 0 ];
main()
```

```
{ int i;
  for ( i = 0; i < 3; i ++ )
        printf(" a[ i ][ 2 – i ]=%d  *a[ i ]=%d  *(*(a+i)+i)=%d\n",
              a[ i ][ 2 – i ], *a[ i ], *(*(a+i)+i));
  for ( i = 0; i < 3; i ++ )
        printf("*pa[ i ]=%d  p[ i ]=%d \n", *pa[ i ], p[ i ] );
}
```

5.11. Что напечатает следующая программа?

```
#include <stdio.h>
char *c[ ] = { "ENTER", "NEW", "POINT", "FIRST" };
char ** cp[ ] = { c+3, c+2, c+1, c };
char ***cpp=cp;
main()
  { printf("%s", **++cpp );
    printf("%s  ", * -- *++cpp+3 );
    printf("%s", *cpp[ -2 ]+3 );
    printf("%s\n", cpp[ -1 ][ -1 ]+1 );
  }
```

5.12. Какие соглашения о конце строки существуют в Си и Паскале? Укажите все «за» и «против» явного указания концов строк с помощью null-литеры ‘\0’.

5.13. В чем заключается проблема «висящей» ссылки? Приведите примеры.

5.14. Нужна ли в Си «сборка мусора»? Почему возникает такая проблема и как она решается в Си?

5.15. Прочитайте следующие описания и определения:

```
int *ip, f( ), *fip( ), (*pfi)( );          char *str[10];          char * (*cp)[5];
int (*r) ( );                               double (*k)(double,int*);
float * (* (*x) [6] )( );                   double (* (* ( y( ) )[ ] )( );
int * (*const *name[9])(void);              char * const p;
```

5.16. Определите переменную x как массив указателей на функцию, имеющую два параметра типа int и возвращающую результат типа указатель на double.

5.17. Определите переменную y как указатель на массив указателей на функцию без параметров, возвращающую результат типа указатель на функцию с одним параметром типа int и результатом типа float.

5.18. Что будет напечатано? Объяснить, почему результат будет таким.

a)
```
#include <stdio.h>
int  try_to_change_it(int);
main()
   { int i = 4, j;
     j = try_to_change_it(i);
     printf("i=%d, j=%d\n", i, j);
   }
int  try_to_change_it(int k)
```

b)
```
#include <stdio.h>
void compare (int , int *);
main()
  { int i = 4, j = 5;
    compare(i, &j);
    printf("i=%d, j=%d\n", i, j);
  }
void compare (int k, int *m)
```

```
{ printf("k1=%d\n", k);
  k+=33;
  printf("k2=%d\n", k);
  return k;
}
```

```
{ printf("k1=%d,*m1=%d\n",k, *m);
    k++; (*m)++;
  printf("k2=%d,*m2=%d\n", k, *m);
 }
```

5.19. Верно ли решена задача: « Описать функцию, меняющую местами значения двух переменных символьного типа. Использовать эту функцию для изменения значений символьных переменных a и b.»

a)
```
void swap ( char x, char y)
    { char t; t = x; x = y; y = t;}
main()
    { char a,b;
      scanf("%c%c", &a, &b);
      swap(a,b);
      printf("a=%c,b=%c\n",a,b);
    }
```

b)
```
void swap ( char *x, char *y)
    { char *t; t = x; x = y; y = t;}
main()
    { char a,b;
      scanf("%c%c", &a, &b);
      swap(&a, &b);
      printf("a=%c,b=%c\n",a,b);
    }
```

c)
```
void swap ( char *x, char *y)
    { char t; t = *x; *x = *y; *y = t;}
main()
    { char a,b;
      scanf("%c%c", &a, &b);
      swap(a,b);
      printf("a=%c,b=%c\n",a,b);
    }
```

d)
```
void swap ( char *x, char *y)
    { char t; t = *x; *x = *y; *y = t;}
main()
    { char a,b;
      scanf("%c%c", &a, &b);
      swap(&a, &b);
      printf("a=%c,b=%c\n",a,b);
    }
```

e)
```
void swap ( char x, char y)
    { char *t; t = &x; &x = &y; &y = t;}
main()
    { char a,b;
      scanf("%c%c", &a, &b);
      swap(&a, &b);
      printf("a=%c,b=%c\n",a,b);
    }
```

f)
```
void swap ( char &x, char &y)
    { char t; t = x; x = y; y = t;}
main()
    { char a,b;
      scanf("%c%c", &a, &b);
      swap(a, b);
      printf("a=%c,b=%c\n",a,b);
    }
```

5.20. Допустимо ли в Си? Если "да" - опишите семантику этих действий; если "нет" - объясните почему.

```
int ques ( char *s1, char *s2)
    { while (*s1 && *s2 && *s1++ == *s2++ );
      return *--s1 - *--s2;
     }
```

5.21. Допустимо ли в Си? Если "да" - опишите семантику этих действий; если "нет" - объясните почему.

```
void ques ( char *s1, char *s2, int n)
     { while (*s1 && *s2 && n-- && (*s1 ++ = *s2 ++ ) ); }
```

5.22. Описать функцию, определяющую упорядочены ли строго по возрастанию элементы целочисленного массива из n элементов.

5.23. Описать функцию, определяющую индекс первого элемента целочисленного массива из n элементов, значение которого равно заданному числу x. Если такого элемента в массиве нет, то считать номер равным –1.

5.24. Описать функцию, вычисляющую значение $x_0 + x_0*x_1 + x_0*x_1*x_2 + \ldots + x_0*x_1*x_2 *\ldots *x_m$, где x_i - элементы вещественного массива x из n элементов, m - индекс первого отрицательного элемента этого массива либо число n-1, если такого элемента в массиве нет.

5.25. Описать функцию, вычисляющую значение max($x_0 + x_{n-1}$, $x_1 + x_{n-2}$, $x_2 + x_{n-3}$,…, $x_{(n-1)/2} + x_{n/2}$), где x_i - элементы вещественного массива x из n элементов.

5.26. Описать функцию, вычисляющую значение min($x_0 * x_1$, $x_1 * x_2$, $x_2 * x_3$,…, $x_{n-3} * x_{n-2}$, $x_{n-2} * x_{n-1}$), где x_i - элементы вещественного массива x из n элементов.

5.27. Описать функцию, вычисляющую значение $x_0*y_0+x_1*y_1+ \ldots + x_k*y_k$, где x_i – отрицательные элементы вещественного массива a из n элементов, взятые в порядке их следования; y_i – положительные элементы этого массива, взятые в обратном порядке; k = min(p,q), где p – количество положительных элементов массива a, q – количество отрицательных элементов этого массива.

5.28. Описать функцию, которая упорядочивает элементы целочисленного массива по неубыванию, используя следующий алгоритм сортировки:

a) сортировка выбором: находится максимальный элемент массива и переносится в его конец; затем этот метод применяется ко всем элементам массива, кроме последнего (т.к. он уже находится на своем месте), и т.д.

b) сортировка обменом (метод пузырька): последовательно сравни-ваются пары соседних элементов x_k и x_{k+1} (k = 0, 1, … ,n-2) и, если $x_k > x_{k+1}$, то они переставляются; в результате наибольший элемент окажется на своем месте в конце массива; затем этот метод применяется ко всем элементам, кроме последнего, и т.д.

c) сортировка вставками: пусть первые k элементов массива (от 0 до k-1) уже упорядочены по неубыванию; тогда берется x_k и размещается среди первых k элементов так, чтобы упорядоченными оказались уже k+1 первых элементов; этот метод повторяется при k от 1 до n-1.

5.29. Описать функцию, определяющую индекс первого элемента целочисленного массива из n элементов, значение которого равно заданному числу x. Если такого элемента в массиве нет, то считать номер равным –1. Элементы массива упорядочены по возрастанию; использовать метод двоичного (бинарного) поиска.

5.30. Программа. Описать функцию f(a, n, p), определяющую, чередуются ли положительные и отрицательные элементы в целочисленном массиве a из n элементов и вычисляющую целочисленное значение p. Если элементы чередуются, то p - это сумма положительных элементов, иначе p - это произведение отрицательных элементов. С помощью этой функции провести анализ целочисленного массива x [50].

5.31. Программа. Описать функцию f(a, n, p), определяющую, упорядочены ли строго по возрастанию элементы в целочисленном массиве a из n элементов, и вычисляющую целочисленное значение p. Если элементы упорядочены, то p - это

произведение разностей рядом стоящих элементов, иначе p - это количество нарушений порядка в массиве a. С помощью этой функции провести анализ целочисленного массива b [60].

5.32. Программа. Описать функцию f (s, n, x), определяющую, какой символ чаще других встречается в строке s и сколько раз он в нее входит. Если таких символов несколько, то взять первый из них по алфавиту. С помощью этой функции провести анализ строки str.

5.33. Программа. Описать функцию f(s, n, x), определяющую, какой символ реже других (но не нуль раз) встречается в строке s и сколько раз он в нее входит. Если таких символов несколько, то взять первый из них по алфавиту. С помощью этой функции провести анализ строки str.

5.34. Программа. Для целочисленного массива a, содержащего n элементов, описать функцию f(a, n, last, k, nlast), определяющую last - значение последнего из элементов массива a, значение которого принадлежит диапазону [-k, k], и nlast - индекс этого элемента. С помощью этой функции вычислить соответствующие значения last и nlast для целочисленных массивов x[20] и y[30].

5.35. Программа. Для вещественного массива a, содержащего n элементов, описать функцию G, определяющую значения максимального и минимального элементов этого массива. С помощью этой функции для вещественных массивов x[25] и y[40] вычислить соответствующие значения.

5.36. Описать функцию, которая изменяет заданную строку следующим образом: сначала записывает все элементы с четными индексами, а затем все элементы с нечетными индексами (с сохранением их относительного порядка в каждой группе).

Например, abcdefgh => acegbdfh, vwxyz => vxzwy.

5.37. Описать функцию, которая в заданной строке меняет местами ее первую и вторую половины.

Например, abcdefgh => efghabcd, vwxyz => yzxvw.

5.38. Описать функцию, осуществляющую циклический сдвиг на n позиций вправо элементов целочисленного массива, содержащего m элементов (n<m).

5.39. Описать функцию, осуществляющую циклический сдвиг на n позиций влево элементов целочисленного массива, содержащего m элементов (n<m).

5.40. Написать программу, обнуляющую каждую четную двоичную единицу в коде, размещенном в переменной типа int. Вывести исходные данные и полученный результат в виде, удобном для анализа проведенных преобразований.

5.41. Написать программу, обнуляющую каждую нечетную двоичную единицу в коде, размещенном в переменной типа int. Вывести исходные данные и полученный результат в виде, удобном для анализа проведенных преобразований.

5.42. Описать функцию, которая в каждом элементе беззнакового целочисленного массива заменяет старший байт нулевым кодом, если в этом байте размещен код латинской буквы.

СТРУКТУРЫ, ОБЪЕДИНЕНИЯ

6.1 Основные сведения

6.1. Верны ли следующие утверждения:

a) описание структуры начинается с ключевого слова struct и содержит список объявлений членов структуры, заключенный в фигурные скобки;

b) за словом struct должен следовать идентификатор, называемый тегом структуры;

c) тег структуры используется в качестве имени типа при описании переменных;

d) имена членов структуры могут совпадать с именами переменных в той же области видимости;

e) имя тега структуры может совпадать с именами переменных в той же области видимости;

f) имя тега структуры может совпадать с именами членов этой структуры;

g) имена членов разных структур могут совпадать;

h) за описанием структуры (после правой закрывающей фигурной скобки) обязательно должен следовать список переменных;

i) переменные x, y, z разных типов

1)
```
struct s { int a; float f; } x, y;
struct s z;
```

2)
```
typedef struct { int a; float f;} s;
s x, y;
struct { int a; float f; } z;
```

3)
```
struct s { int a; float f; };
typedef struct s new_s;
struct s x; new_s y, z;
```

4)
```
struct s { int a; float f; };
typedef struct s s1;
typedef struct s s2;
s1 x, y; s2 z;
```

j) переменные x, y, z одного типа

1)
```
struct { int a; float f; } x, y;
struct { int a; float f; } z;
```

2)
```
struct { int a; float f; } x, y;
struct { float f; int a; } z;
```

k) для доступа к членам структуры используется операция **.** (точка);

l) структуры не могут быть вложенными;

m) структурную переменную при ее описании можно инициализировать списком константных выражений, заключенным в фигурные скобки;

6.2. Каким образом в Си определяется эквивалентность типов? Какая эквивалентность типов рассматривается: структурная или именная? Чем они отличаются?

6.3. Описать в виде структуры следующие понятия:

a) дата (число, месяц, год);

b) адрес (страна, город, улица, дом, квартира);

c) треугольник (две стороны и угол между ними);

d) окружность (радиус и центр);

e) расписание занятий студента 209 группы факультета ВМК (день недели, предметы (с указанием – лекции или семинары), часы занятий, аудитория, фамилия преподавателя)

f) результаты проверки контрольной работы (номер группы, номер контрольной работы, тема, 25 строчек с полями: фамилия студента, вариант, информация о каждой из пяти задач (ее номер, оценка за ее решение, характеристика ошибок), итоговая оценка студента за эту контрольную работу.

6.4. Используя определенный в задаче 6.3 тип, описать переменную этого типа и присвоить ей значение:

a) дата – 16 ноября 1999 года;

b) адрес – Россия, Москва, Ильинка, дом 3, кв. 34;

c) треугольник – 5, 6.7, 35°;

d) окружность – радиус 4.567, центр (1.4, 5.6);

e) расписание занятий студента 209 группы факультета ВМК – понедельник, математический анализ (лекция) –1 пара, П-12, Ломов И.С., математический анализ (семинар) – 2 пара, 706, Григорьев Е.А., программирование (семинар) – 3 пара, 713, Пильщиков В.Н.

6.5. Что напечатает программа?

```
#include <stdio.h>
main()
  { struct data1 { char c[4]; char *s; } d1 = { "abc", "def" };
    struct data2 { char * cp; struct data1 inf; } d2 = { "ghi", { "jkl", "mno"} };
    printf("d1.c[0]=%c  *d1.s=%c\n", d1.c[0], *d1.s);
    printf("d1.c=%s  d1.s=%s\n", d1.c, d1.s);
    printf("d2.cp=%s  d2.inf.s=%s\n", d2.cp, d2.inf.s);
    printf("++d2.cp=%s  ++d2.inf.s=%s\n", ++d2.cp, ++d2.inf.s);
  }
```

6.6. Верны ли следующие утверждения:

a) описание объединения начинается с ключевого слова union и содержит список объявлений членов объединения, заключенный в фигурные скобки;

b) каждый член объединения располагается в памяти с одного и того же адреса; объем памяти для каждого члена выделяется в соответствии с его размером;

c) для каждого из членов объединения выделяется одна и та же область памяти;

d) все проблемы, связанные с выравниванием, решает компилятор;

e) в каждый момент времени объединение может содержать значение только одного из его членов;

f) все операции, применимые к структурам, применимы и к объединениям;

g) «рассогласованность» при работе с активным вариантом объединения контролируется компилятором.

6.7. Можно ли в Си создать аналог вариантных записей Паскаля?

6.8. Описать тип, с помощью которого можно организовать хранение данных о различных видах транспорта: грузовиках, автобусах, легковых автомобилях и мотоциклах. Для каждого вида транспорта имеются как общие характеристики (владелец, год производства и модель), так и индивидуальные (для грузовиков - число осей, грузоподъемность, для автобусов - число мест для пассажиров, для легковых автомобилей - число дверей (2 или 4), для мотоциклов - тип двигателя (двух- или четырехтактный)).

6.2 Структуры и функции. Указатели на структуры.

6.9. Верны ли следующие утверждения:

a) к структурам одного типа применима операция присваивания;

b) к структурам одного типа, не содержащим вложенных структур, применима операция сравнения (выполняется почленное сравнение);

c) параметром функции может быть указатель на структуру, но не сама структура;

d) параметры функции – структуры передаются по значению;

e) результатом работы функции может быть структура;

f) результатом работы функции может быть указатель на структуру;

g) функция sizeof(struct any) выдает результат, равный сумме длин всех полей этой структуры;

h) к структурам применима операция взятия адреса;

6.10. Перечислить все операции, применимые к структурам.

6.11. Пусть точка на плоскости описана следующим образом:

```
struct point { int x; int y;}
```

Верно ли решена задача: «описать функцию, которая присваивает значение структуре типа struct point »

a)
```
void assign_to_point ( struct point p, int a, int b)
    { p.x = a;  p.y = b; }
```

b)
```
void assign_to_point ( struct point *p, int a, int b)
    { (*p).x = a;  (*p).y = b; }
```

c)
```
void assign_to_point ( struct point *p, int a, int b)
    { *p.x = a;  *p.y = b; }
```

d)
```
void assign_to_point ( struct point *p, int a, int b)
    { p -> x = a;  p -> y = b; }
```

6.12. Пусть точка на плоскости описана следующим образом:

```
struct point { int x; int y;}
```

Верно ли решена задача: «описать функцию, которая создает точку из двух целых чисел»

a)
```
struct point create_point ( int a, int b)
    { struct point p;
      p.x = a;  p.y = b;  return p;
    }
```

b)
```
struct point *create_point ( int a, int b)
```

```
{ struct point p;
  p.x = a;  p.y = b;  return &p;
}
```

c)
```
struct point *create_point ( int a, int b)
{ struct point *pp;
  pp -> x = a;  pp -> y = b;  return pp;
}
```

d)
```
struct point *create_point ( int a, int b)
{ struct point *pp;
  pp = (struct point *) malloc(sizeof(struct point));
  pp -> x = a;  pp -> y = b;  return pp;
}
```

6.13. Пусть точка на плоскости описана следующим образом:

```
struct point { int x; int y;}
```

Описать функцию, которая по трем точкам, являющимися вершинами некоторого прямоугольника, определяет его четвертую вершину;

6.14. Описать в виде структуры

a) точку на плоскости;
b) цветную точку на плоскости;
c) комплексное число;
d) рациональное число.

Разработать совокупность операций для данных этого типа; реализовать каждую из них в виде функции.

6.15. Пусть «целочисленная» окружность на плоскости описана следующим образом:

```
struct point { int x; int y;};
struct circle { int radius; struct point center;};
```

Пусть есть массив struct circle plane [50], содержащий информацию об окружностях на плоскости. Описать функцию, определяющую

a) есть ли среди этих окружностей хотя бы две концентрические окружности;

b) есть ли среди этих окружностей хотя бы две вложенные (не обязательно концентрические) окружности;

c) есть ли среди этих окружностей три попарно пересекающихся окружности;

d) есть ли среди этих окружностей хотя бы одна «уединенная», т.е. не имеющая общих точек ни с какой другой окружностью массива plane.

6.16. Пусть результаты анализа некоторого текста, состоящего из английских слов, содержатся в следующем частотном словаре dictionary:

```
#define MAXSIZE 1000
#define LENGHT 20 /* максимальная длина слова */
struct elem { char * word; struct info *data;};
struct info { int count; /* количество повторений слова в данном тексте */
              char *alias; /* синоним данного слова */
            };
struct { struct elem *tabl [ MAXSIZE];
```

```
    int number; /* количество слов в словаре */
  }
dictionary;
```

Каждое слово (word) встречается в словаре только один раз; синонимы (alias) могут быть одинаковыми у разных слов.

Описать функцию, определяющую

a) встречается ли данное слово в этом словаре: результат – указатель на соответствующий элемент либо NULL:

1) слова неупорядочены по алфавиту;
2) слова упорядочены по алфавиту;

b) какое слово чаще других встречается в анализируемом тексте; если таких слов несколько, то взять первое по алфавиту; результат работы функции – указатель на соответствующий элемент:

1) слова неупорядочены по алфавиту;
2) слова упорядочены по алфавиту;

c) на каждую ли букву латинского алфавита в этом словаре найдется хотя бы одно слово:

1) слова неупорядочены по алфавиту;
2) слова упорядочены по алфавиту;

d) на какие буквы (букву) латинского алфавита в этом словаре больше всего слов; результат работы функции – указатель на строку, составленную из этих букв, перечисленных в алфавитном порядке:

1) слова неупорядочены по алфавиту;
2) слова упорядочены по алфавиту;

e) есть ли в словаре различные слова, имеющие одинаковые синонимы.

6.17. Пусть результаты анализа некоторого текста содержатся в частотном словаре (см. предыдущую задачу).

Описать функцию

a) struct elem *add_word (char * word, char *alias), вставляющую новое слово и информацию о нем в словарь dictionary (предполагается, что такого слова в словаре еще нет, оно первый раз встретилось в тексте);

1) слова неупорядочены по алфавиту;
2) слова упорядочены по алфавиту;

Результат работы функции – указатель на вставленный элемент.

b) struct elem *update_word (char * word, char *alias), изменяющую значение синонима для данного слова (предполагается, что такое слово в словаре обязательно есть);

1) слова неупорядочены по алфавиту;
2) слова упорядочены по алфавиту;

Результат работы функции – указатель на элемент словаря, где изменено значение синонима.

c) struct elem *delete_word (char *word), удаляющую данное слово из словаря; результат работы функции – указатель на структуру, содержащую информацию об удаленном слове либо NULL, если такого слова нет в словаре;

1) слова неупорядочены по алфавиту;
2) слова упорядочены по алфавиту;

6.18. Программа. Определить число вхождений каждого служебного слова в данную программу на Си. Тщательно продумать способ представления информации о служебных словах.

6.19. Допустимо ли в Си? Если "да" - опишите семантику этих действий, объясните, что будет напечатано; если "нет" - объясните почему.

```
#include <stdio.h>
struct data { char *s; int i; struct data *dp; };
main()
  { static struct data a[ ] = {  { "abcd", 1, a+1 },
                                 { "efgh", 2, a+2 },
                                 { "ijkl",3, a }
                               };
    struct data  *p = a;  int i;
    printf("a[0].s=%s  p -> s=%s  a[2].dp -> s=%s\n",
            a[0].s, p -> s, a[2].dp -> s);
    for ( i = 0; i < 2; i++ ) printf("--a[i].i=%d  ++a[i].s[3]=%c\n",
                                  --a[i].i, ++a[i].s[3]);
    printf("++(p->s)=%s\n", ++(p->s) );
    printf("a[(++p) -> i].s=%s\n", a[(++p) -> i].s);
    printf("a[--(p -> dp -> i)].s=%s\n", a[--(p -> dp -> i)].s);
  }
```

6.20. Пусть

```
struct s { int k; float *f; char *p[2];};
struct s *ps;
```

Верно ли присвоено значение переменной ps и всем объектам, с ней связанным:

```
char str[5] = “abcd”;
ps = (struct s *) malloc(sizeof(struct s));
(*ps).k = 5;
ps -> f = (float *) malloc(sizeof(float)); *(ps -> f) = 3.1415;
(*ps).p[0] = (char*) malloc(5*sizeof(char));
(*ps).p[1] = (char*) malloc(10*sizeof(char));
(*ps).p[0] = str;
(*ps).p[1] = “abcdefghi”;
```

6.21. Присвоить значение переменной q и всем объектам, с ней связанным:

```
struct data { double **p; char *s; int *a[2]; };
        struct data *q;
```

6.22. Присвоить значение переменной a и всем объектам, с ней связанным:

```
struct b { double *q; int * (*p)[2]; };
        struct b **a[1];
```

6.23. Присвоить значение переменной x и всем объектам, с ней связанным:

```
struct r { double *a[3]; char **s;
                union { int i; float f; } u;
              }
        struct r x[2];
```

6.24. Присвоить значение переменной x и всем объектам, с ней связанным:

```
struct a { char ***s;
           char (*p)[2];
         };
typedef struct a * data;
data x[2];
```

6.25. Присвоить значение переменной pt и всем объектам, с ней связанным:

```
struct t { int **pi;
           double (*k)(double,int*);
           char *p[2];
         };
struct t *pt;
```

6.26. Присвоить значение переменной a и всем объектам, с ней связанным:

```
struct data { int *i; int (*f)(int); char **s; };
struct data a[2];
```

ВВОД-ВЫВОД

7.1 Стандартный ввод-вывод

7.1. Программа. Даны натуральные числа i, n ($i \le n$), вещественные числа a_1, a_2, …,a_n. Найти среднее арифметическое всех чисел, кроме a_i.

7.2. Программа. Даны вещественные числа a_1, a_2, …,a_{50}. Распечатать “сглаженные” значения a_1, a_2, …,a_{50} , заменив в исходной последовательности все члены, кроме первого и последнего, по формуле

$$a_i = (a_{i-1} + a_i + a_{i+1})/3 \quad i = 2, 3, \dots ,49;$$

считая, что

a) после того, как получено новое значение некоторого члена последовательности, оно используется для вычисления нового значения следующего за ним члена последовательности;

b) при “сглаживании” используются лишь старые члены последовательности.

7.3. Программа. Даны вещественные числа a_1, a_2, … Известно, что $a_1 > 0$ и что среди a_2, a_3, … есть хотя бы одно отрицательное число. Пусть a_1, a_2, …,a_n - члены данной последовательности, предшествующие первому отрицательному члену (n заранее неизвестно). Распечатать

a) $a_1 + a_2 + \dots + a_n$

b) $a_1 * a_2 * \dots * a_n$

c) среднее арифметическое a_1, a_2, …,a_n

d) a_1 , $a_1 * a_2$, $a_1 * a_2 * a_3$,…, $a_1 * a_2 * \dots * a_n$

e) $a_1 + 2*a_2 + 3*a_3 + \dots + (n-1)*a_{n-1} + \dots n*a_n$

f) $| a_1 - a_2 |, | a_2 - a_3 | , \dots , | a_{n-1} - a_n |, | a_n - a_1 |$

7.4. Программа. Даны целые положительные числа n, a_1, a_2, …, a_n ($n \geq 4$). Считать, что a_1, a_2, …, a_n - это измеренные в сотых долях секунды результаты n спортсменов в беге на 100 метров. По этим результатам составить команду из четырех лучших бегунов для участия в эстафете 4×100, т.е. распечатать номера спортсменов, имеющих четыре лучших результата.

7.5. Верно ли решена следующая задача: «читать символы из стандартного входного потока, пока код каждого следующего символа больше кода предыдущего; определить, сколько символов было прочитано»

a) ... i = 0;
   while (getchar() < getchar()) i = i + 2;
b) … i = 0; c = getchar();
   while (c < (c = getchar())) i++;
c) … i = 0; c = getchar();
   while (c < (d = getchar())) { i++; c = d;}
d) … i = 0; c = getchar();
   while (d = getchar(), c<d)) { i++; c = d; }
e) … i = 0; c = getchar();
   while (c != EOF && (d = getchar()) != EOF && c<d) { i++; c=d; }

7.6. Сравнить следующие фрагменты программы:

a) while (c = getchar() = EOF)
b) while (c = getchar() == EOF)
c) while ((c = getchar()) == EOF)
d) while ((c = getchar()) = -1)

7.7. Допустимо ли в Си? Если "да" - опишите семантику этих действий; если "нет" - объясните почему.

```
int i,k,sum;
for ( i=1; scanf("%d",&k) == 1; i++)
   printf("i = %d, k = %d, sum = %d\n", i, k, sum+=k );
```

7.8. Программа. Дана непустая последовательность слов, разделенных одним или несколькими пробелами. Признак конца текста – точка. Распечатать этот текст, удалив из него лишние пробелы (каждую группу из нескольких пробелов заменить одним пробелом).

7.9. Программа. Дана непустая последовательность слов из прописных (больших) латинских букв. Слова разделены пробелом; признак конца текста – точка.

a) подсчитать количество слов в этом тексте;
b) подсчитать количество слов, у которых совпадают первая и последняя буквы;
c) подсчитать количество слов, являющихся некоторым фрагментом латинского алфавита;
d) подсчитать количество слов, содержащих все буквы, которые входят в состав слова UNIX.

7.10. Программа. Дана непустая последовательность слов, разделенных пробелом; признак конца текста – точка. Длина каждого слова – не более 20 литер.

a) распечатать все слова, у которых не совпадают первая и последняя буквы;
b) распечатать все слова, являющиеся «перевертышами», т.е. словами, одинаково читающимися слева направо и справа налево;
c) распечатать текст, оставив из рядом стоящих одинаковых слов только одно;
d) распечатать текст, удалив все слова, где есть символы, отличные от латинских букв.

7.11. Программа. Дана непустая последовательность слов, разделенных пробелом; признак конца текста – точка. Длина каждого слова – не более 20 литер. Распечатать данный текст следующим образом: все строки должны быть одинаковой длины (длина строки задается в командной строке); каждое слово должно быть распечатано в одной строке без переносов; если в строке несколько слов, то пробелы между ними должны быть равномерно распределены; если в строке помещается только одно слово и его длина меньше длины строки, то оно должно быть выровнено по левому краю; если длина слова больше длины строки, то такие слова из текста удаляются, при этом после распечатки текста о каждом таком слове выдается предупреждение.

7.12. Программа. Дана непустая последовательность слов из строчных (малых) латинских букв. Слова разделены пробелом; признак конца текста – точка. Напечатать все буквы, которые

a) чаще других встречаются в данном тексте;
b) входят в каждое слово данного текста;
c) входят в наибольшее количество слов данного текста;

ПРИЛОЖЕНИЯ

8.1 Библиотека стандартных функций языка С

Здесь приводится краткое описание некоторых библиотечных функций, утвержденных в качестве ANSI-стандарта языка Си. Более подробное и полное описание этой библиотеки можно найти в [1] и [2].

8.1.1 Функции работы со строками

Эти функции определены в головном файле **<string.h>**. Если копирование имеет дело с объектами, перекрывающимися по памяти, то поведение функций не определено. Функции сравнения рассматривают аргументы как массивы элементов типа unsigned char.

char *strcpy(char *s, const char *ct)
копирует строку ct в строку s, включая ‘\0’; возвращает s.

char *strcat (char *s, const char *ct)
приписывает ct к s; возвращает s.

char strcmp(const char *cs, const char *ct)

сравнивает cs с ct; возвращает значение меньшее 0, если cs<ct; равное 0, если cs==ct, и большее 0,если cs>ct.

char *strchr(const char *cs, char c)
возвращает указатель на первое вхождение с в cs или, если такового не оказалось, NULL.

char *strrchr(const char *cs, char c)
возвращает указатель на последнее вхождение с в cs или, если такового не оказалось, NULL.

char *strstr(const char *cs, const char *ct)
возвращает указатель на первое вхождение ct в cs или, если такового не оказалось, NULL.

size_t strlen(const char *cs)
возвращает длину cs (без учета ‘\0’).

char *strtok(char *s, const char *ct)
ищет в s лексему, ограниченную литерами из ct.
Последовательные вызовы функции strtok разбивают строку s на лексемы. Ограничителем лексемы может быть любая литера, входящая в строку ct. В первом вызове функции указатель s не равен NULL. Функция находит в строке s первую лексему, состоящую из литер, не входящих в ct; работа этого вызова завершается тем, что поверх следующей литеры пишется ‘\0’ и возвращается указатель на выделенную лексему. Каждый последующий вызов функции strtok, в котором указатель s равен NULL, выдает указатель на следующую лексему, которую функция будет искать сразу за концом предыдущей. Функция возвращает NULL, если далее никакой лексемы не обнаружено. Параметр ct от вызова к вызову может варьироваться.

void *memcpy(void *s, const void *ct, size_t n)
копирует n литер из ct в s и возвращает s.

void *memset(void *s, char c, size_t n)
размещает литеру с в первых n позициях строки s и возвращает s.

8.1.2 Функции проверки класса литер

Головной файл **<ctype.h>** предоставляет функции, которые позволяют определить, принадлежит ли литера определенному классу. Параметр каждой из этих функций имеет тип int и должен быть либо значением unsigned char, приведенным к int, либо значением EOF; возвращаемое значение тоже имеет тип int. Функции возвращают ненулевое значение (“истину”), если аргумент принадлежит указанному классу литер, и нуль (“ложь”) – в противном случае.

int isupper (int c) буква верхнего регистра

int islower (int c)	буква нижнего регистра
int isalpha (int c)	isupper(c) или islower(c) истины
int isdigit (int c)	десятичная цифра
int isalnum (int c)	isalpha (c) или isdigit (c) истины
int isxdigit (int c)	шестнадцатиричная цифра
int isspace (int c)	пробел, новая-строка, возврат-каретки, табуляция
int isgraph (int c)	печатаемая литера, кроме пробела
int isprint (int c)	печатаемая литера, включая пробел

Кроме этих функций в файле есть две функции, выполняющие преобразование букв из одного регистра в другой.

int tolower (int c)

если c – буква верхнего регистра, то tolower(c) выдаст эту букву на нижнем регистре; в противном случае она вернет c.

int toupper (int c)

если c – буква нижнего регистра, то toupper(c) выдаст эту букву на верхнем регистре; в противном случае она вернет c.

8.1.3 Ввод-вывод

Поток - это источник или получатель данных; его можно связать с диском или каким-либо другим внешним устройством. Библиотека поддерживает два вида потоков: текстовый и бинарный.

Текстовый поток - это последовательность строк. Каждая строка имеет нуль или более литер и заканчивается литерой ‘\n’.

Бинарный поток - это последовательность не преобразуемых байтов, представляющих собой некоторые промежуточные данные, которые обладают тем свойством, что, если их записать, а затем прочитать той же системой ввода-вывода, то мы получим информацию, совпадающую с исходной.

В UNIXе эти виды потоков не различаются.

Поток соединяется с файлом или устройством посредством его открытия; эта связь разрывается при закрытии потока. Открытие файла возвращает указатель на объект типа FILE, который содержит всю информацию, необходимую для управления этим потоком. Если не возникает двусмысленности, далее будем использовать термины «файловый указатель» и «поток» как равнозначные.

Когда программа начинает работу, уже открыты три потока: stdin, stdout и stderr.

Все приведенные ниже функции, связанные с вводом-выводом, определены в головном файле **<stdio.h>**.

8.1.3.1 Функции ввода-вывода литер

int fgetc (FILE *stream)

fgetc возвращает очередную литеру из потока stream в виде unsigned char, переведенной в int, или EOF, если исчерпан файл или обнаружена ошибка.

char *fgets (char *s, int n, FILE *stream)

fgets читает не более n-1 литер в массив s, прекращая чтение, если встретилась литера новая-строка, которая включается в массив; кроме того, записывает в массив литеру ‘\0’. Функция fgets возвращает s или NULL, если исчерпан файл или обнаружена ошибка.

int fputc (int c, FILE * stream)

fputc пишет литеру c, переведенную в unsigned char, в stream. Возвращает записанную литеру или EOF в случае ошибки.

int fputs (const char *s, FILE * stream)

fputs пишет строку s, которая может не иметь ‘\n’, в stream. Возвращает неотрицательное целое или EOF.

int getc (FILE *stream)

getc делает то же, что и fgetc, но в отличие от нее может быть макросом; в этом случае stream может быть вычислен более одного раза.

int getchar (void)

getchar() делает то же, что и getc(stdin).

char *gets (char *s)

gets читает следующую строку ввода в массив s, заменяя литеру новая-строка на ‘\0’. Возвращает s или NULL, если исчерпан файл или обнаружена ошибка.

int putc (int c, FILE *stream)

putc делает то же, что и fputc, но в отличие от нее может быть макросом; в этом случае stream может быть вычислен более одного раза.

int putchar (int c)

putchar(c) делает то же, что и putc(c,stdout).

int puts (const char *s)

puts пишет строку s и литеру новая-строка в stdout. В случае ошибки возвращает EOF; если запись прошла нормально - неотрицательное значение.

int ungetc (int c, FILE *stream)

ungetc отправляет литеру c (переведенную в unsigned char) обратно в stream; при следующем чтении из stream она будет получена снова. Для каждого потока можно вернуть не более одной литеры. Нельзя возвращать EOF. В качестве результата ungetc выдает отправленную назад литеру или, в случае ошибки, EOF.

8.1.4 Математические функции

В головном файле **<math.h>** описываются следующие математические функции:

double sin(double x)	синус x
double cos(double x)	косинус x
double tan(double x)	тангенс x
double asin(double x)	арксинус x; $x \in [-1,+1]$
double acos(double x)	арккосинус x; $x \in [-1,+1]$
double atan(double x)	арктангенс x
double sinh(double x)	гиперболический синус x
double cosh(double x)	гиперболический косинус x
double tanh(double x)	гиперболический тангенс x
double exp(double x)	экспоненциальная функция e^x
double log(double x)	натуральный логарифм $\ln(x)$, $x>0$
double log10(double x)	десятичный логарифм $\log_{10}(x)$, $x>0$
double sqrt(double x)	квадратный корень $\sqrt{x}$, $x \geq 0$
double fabs(double x)	абсолютное значение $\|x\|$
double pow(double x, double y)	x^y
double ldexp(double x, int n)	$x*2^n$

8.1.5 Функции общего назначения

Функции этого раздела предназначены для преобразования чисел и запроса памяти; они описаны в головном файле **<stdlib.h>**.

double atof(const char *s)

atof переводит строку s в значение double. В случае переполнения выдает HUGE_VAL - некоторое положительное double-значение, определенное в головном файле <math.h>; при исчезновении порядка - нуль.

int atoi(const char *s)

atoi переводит строку s в значение int. В случае переполнения выдает (int)HUGE_VAL - некоторое положительное double-значение, определенное в головном файле <math.h>, преобразованное в int; при исчезновении порядка - нуль.

void *calloc(size_t nobj, size_t size)

calloc возвращает указатель на место в памяти, отведенное для массива nobj объектов, каждый из которых имеет размер size. Выделенная область памяти обнуляется. Если память отвести не удалось, то результат работы функции - NULL.

void *malloc(size_t size)

malloc возвращает указатель на место в памяти для объекта размера size. Выделенная память не инициализируется. Если память отвести не удалось, то результат работы функции - NULL.

void free(void *p)

free освобождает область памяти, на которую указывает p; если p равно NULL, то функция ничего не делает. Значение p должно указывать на область памяти, ранее выделенную с помощью функций calloc или malloc.

8.2 Фрагменты стандарта языка Си

8.2.1 Классификация типов

типы ::= типы_данных | функциональные_типы | неполные_типы

типы_данных ::= скалярные_типы | нескалярные_типы
скалярные_типы ::= арифметические типы | указатели
арифметические_типы ::= целочисленные_типы | плавающие_типы
целочисленные_типы ::= ***char*** | ***signed char*** | ***unsigned char*** | ***short*** | ***unsigned short*** | ***int*** | ***unsigned int*** | ***long*** | ***unsigned long*** | перечислимые типы | поля_битов
плавающие_типы ::= ***float*** | ***double*** | ***long double***
указатели ::= указатели_на_данные | указатели_на_функции | указатели_на_неполные_типы
нескалярные_типы ::= структуры | массивы | объединения
неполные_типы ::= неполные_структуры | неполные_массивы | неполные_объединения | ***void***

8.2.2 Приоритеты и порядок выполнения операций

операции	**выполняются**
( ) [ ] → . постфиксные ++ и - -	слева направо (→)
! ~ префиксные ++ и - - унарные + - *** & (**тип**) sizeof**	справа налево (←)
*** / %**	слева направо (→)
+ –	слева направо (→)
<< >>	слева направо (→)
< <= > >=	слева направо (→)
== !=	слева направо (→)
&	слева направо (→)
^	слева направо (→)

|	слева направо (→)
&&	слева направо (→)
||	слева направо (→)
? :	справа налево (←)
= += –= *= /= %= &= ^= |= <<= >>=	справа налево (←)
,	слева направо (→)

Несмотря на строго определенный приоритет операций, при вычислении выражения существует некоторая свобода в выборе порядка вычисления его подвыражений.

Например, y = *p++; может быть вычислено как
temp = p; p += 1; y = *temp; либо как
y = *p; p += 1;

Порядок вычислений важен для понимания того, когда проявляется побочный эффект. Побочный эффект при вычислении выражения - это занесение в память значений объектов, изменение состояния файла либо доступ к volatile - объектам.

Точка последовательных вычислений (sequence point) - это точка в программе, где можно точно определить, какие из побочных эффектов уже проявились, а какие - еще нет.

Если выражение является частью оператора, то точкой, где заведомо выполнились все побочные эффекты его вычисления - это конец этого оператора. Например, в y = 37; x += y; можно быть уверенным, что 37 будет занесено в y раньше, чем значение y будет извлечено из памяти при вычислении суммы x + y.

Кроме того, точки последовательных вычислений могут быть расположены внутри самого выражения:

- при выполнении операции **x , y** такая точка находится между вычислением x и y;
- при выполнении операции **z ? x : y** такая точка находится между вычислением z и вычислением x либо y;
- при вызове функции все побочные эффекты вычисления значений ее аргументов проявятся перед выполнением ее тела;
- при выполнении операций **x && y** и **x || y** такая точка находится между вычислением x и вычислением y.

Например, в if ((c = getchar()) != EOF && isprint(c)) вызов функции isprint(c) произойдет только после того, как переменная c получит новое значение.

Между двумя точками последовательных вычислений изменение значения переменной возможно не более одного раза.

Например, верно val = 10 * val + (c - ‘0’); но неверно i = ++i + 2;

Выражение может содержать точки последовательных вычислений, и тем не менее, порядок вычислений не будет однозначным. Например, f(x) + g(x) содержит такие точки, однако операция + допускает произвольный порядок вычисления ее операндов.

8.2.3 Арифметические преобразования при выполнении арифметических операций вида X op Y

1. если есть операнд типа short или signed char, то он преобразуется к int; если есть операнд типа char, unsigned char или unsigned short, и все значения этого типа могут быть представлены как int, то он преобразуется к int; иначе - к unsigned int. Это преобразование называется «целочисленное расширение» (promoting).

2. если после выполнения п.1 операнды имеют различные типы, то осуществляется их приведение к общему типу. Общим для двух типов (кроме случая «unsigned int - long») является тот, который расположен позже в последовательности int, unsigned int, long, unsigned long, float, double, long double.

Если операнды имеют типы unsigned int и long, и все значения типа unsigned int могут быть представлены как long, то общим типом является long; иначе - unsigned long. Это преобразование называют «согласование типов» (balansing).

3. после этого выполняется арифметическая операция; тип результата - это тип, к которому были приведены оба операнда.

8.2.4. Арифметические преобразования при выполнении присваивания и явного приведения

M-битового представления величины X к N-битовому представлению

преобразование	**N < M**	**N == M**	**N>M**
знаковое целое к знаковому целому	отсечение старших N-M бит	значение сохраняется	значение сохраняется
беззнаковое целое к знаковому целому	зависит от реализации	если x ≥0,знач.сохр. иначе зависит от реализации	значение сохраняется
вещественное к знаковому целому	если $\|x\|<2^{N-1}$,то trunc(x) иначе зависит от реализации	если $\|x\|<2^{N-1}$,то trunc(x) иначе зависит от реализации	если $\|x\|<2^{N-1}$,то trunc(x) иначе зависит от реализации
знаковое целое к беззнаковому целому	если x≥0, то $x\%2^N$ иначе зависит от реализации	если x≥0 знач. сохр. иначе $x+2^N$	если x≥0 знач. сохр. иначе $x+2^N$
беззнаковое целое к беззнаковому целому	$x\%2^N$	значение сохраняется	значение сохраняется

вещественное к беззнаковому целому	если $0 \le x < 2^N$ trunc(x) иначе зависит от реализации	если $0 \le x < 2^N$ trunc(x) иначе зависит от реализации	если $0 \le x < 2^N$ trunc(x) иначе зависит от реализации
знаковое целое к вещественному	сохр. знак, сохр. старшие N-1 бит	значение сохраняется	значение сохраняется
беззнаковое целое к вещественному	знак +, сохр. старшие N-1 бит	знак +, сохр. старшие N-1 бит	значение сохраняется
вещественное к вещественному	сохр. старшие N-1 бит	значение сохраняется	значение сохраняется

8.2.5 Неявное приведение типов в операторе присваивания X = Y

тип X	**тип Y**	**тип результата**
арифметический	арифметический	тип X
указатель, структура либо объединение	тип X	тип X
указатель на const T	указатель на T либо на const T	тип X
указатель на volatile T	указатель на T либо на volatile T	тип X
указатель на const volatile T	указатель на T, либо на const T, либо на volatile T, либо на const volatile T	тип X
указатель на void	указатель на T	тип X
указатель на T	указатель на void	тип X
указатель на T	целое значение 0	тип X

8.2.6 Явное приведение (тип T) X

тип X	тип T	тип результата
скалярный	целочисленный	тип T
арифметический	плавающий	тип T
целочисленный	указатель на любой тип	тип T
указатель на T1	указатель на T2	тип T
указатель на функцию	указатель на функцию	тип T
скалярный	void	void

8.2.7 Адресная арифметика

операция	тип X	тип Y	тип результата
X+Y	указатель_на_данные	целочисленный	тип X
X+Y	целочисленный	указатель_на_данные	тип X
X+=Y	указатель_на_данные	целочисленный	тип X
X–Y	указатель_на_данные	целочисленный	тип X
X–Y	указатель_на_данные	указатель_на_данные	***ptrdiff_t***
X–=Y	указатель_на_данные	целочисленный	тип X
X&&Y	указатель	указатель	***int***
! X	указатель		***int***
X || Y	указатель	указатель	***int***
X++	указатель		указатель
X––	указатель		указатель
++X	указатель		указатель

––X	указатель		указатель
sizeof X	указатель		***size_t***
X [Y]	указатель на T	целочисленный	тип T
X [Y]	целочисленный	указатель на T	тип T
X –>Y	указатель на структуру или объединение	имя поля этой структуры или объединения	тип поля Y
*X	указатель_на_данные типа T		тип T
*X	указатель_на_функцию типа T		тип T
*X	указатель_на ***void***		***void***

ЛИТЕРАТУРА

1. Б. Керниган, Д. Ритчи. Язык программирования Си. М., «Финансы и статистика», 1992
2. American National Standard for Information Systems - Programming Language C, X3.159-1989
3. Б. Керниган, Д. Ритчи, А. Фьюэр Язык программирования Си. Задачи по языку Си. М., «Финансы и статистика», 1985
4. Н. Джехани. Программирование на языке Си. М., «Радио и связь», 1988
5. Б. Керниган, Р. Пайк. Универсальная среда программирования UNIX. М., «Финансы и статистика», 1992
6. С. Баурн. Операционная система UNIX.М., «Мир», 1986
7. С.А. Абрамов, Г.Г. Гнездилова и др. Задачи по программированию. М., «Наука», 1988
8. В.Н. Пильщиков. Сборник упражнений по языку Паскаль. М., «Наука», 1989
9. Руденко Т.В. “Сборник задач и упражнений по языку Си (учебное пособие для студентов II курса)”., Московский государственный университет им. М.В. Ломоносова, Факультет вычислительной математики и кибернетики, М., 1999