1. Простые программы и алгоритмы. Сюрпризы, советы.

1.1.

Составьте программу приветствия с использованием функции *printf*. По традиции принято печатать фразу "**Hello, world** !" ("**Здравствуй, мир** !").

1.2.

Найдите ошибку в программе

```
#include <stdio.h>
main(){
    printf("Hello, world\n");
}
```

Ответ: раз не объявлено иначе, функция *main* считается возвращающей целое значение (int). Но функция *main* не возвращает ничего - в ней просто нет оператора *return*.

Корректно было бы так:

```
#include <stdio.h>
main(){
    printf("Hello, world\n");
    return 0;
}
```

или

```
#include <stdio.h>
void main(){
    printf("Hello, world\n");
    exit(0);
}
```

а уж совсем корректно - так:

```
#include <stdio.h>
int main(int argc, char *argv[]){
    printf("Hello, world\n");
    return 0;
}
```

1.3.

Найдите ошибки в программе

```
#include studio.h
main
{
    int i
    i := 43
    print ('В году i недель')
}
```

1.4.

Что будет напечатано в приведенном примере, который является частью полной программы:

```
int n;
n = 2;
printf ("%d + %d = %d\n", n, n, n + n);
```

1.5.

В чем состоят ошибки?

```
if( x > 2 )
then    x = 2;
if  x < 1
        x = 1;
```

Ответ: в Си нет ключевого слова *then*, условия в операторах *if*, *while* должны браться в ()-скобки.

1.6.

Напишите программу, печатающую ваше имя, место работы и адрес. В первом варианте программы используйте библиотечную функцию *printf*, а во втором - *puts*.

1.7.

Составьте программу с использованием следующих постфиксных и префиксных операций:

```
a = b = 5
a + b
a++ + b
++a + b
--a + b
a-- + b
```

Распечатайте полученные значения и проанализируйте результат.

1.8.

Цикл *for*

```
for(INIT; CONDITION; INCR)
        BODY
        INIT;
repeat:
        if(CONDITION){
                BODY;
        cont:
                INCR;
                goto repeat;
        }
out:    ;
```

Цикл *while*

```
while(COND)
        BODY
cont:
repeat:
        if(CONDITION){
                BODY;
                goto repeat;
        }
out:    ;
```

Цикл *do*

```
do
        BODY
while(CONDITION)
cont:
repeat:
        BODY;
        if(CONDITION) goto repeat;
out:    ;
```

В операторах цикла **внутри** тела цикла **BODY** могут присутствовать операторы *break* и *continue*; которые означают на наших схемах следующее:

```
#define break     goto out
#define continue  goto cont
```

1.9.

Составьте программу печати прямоугольного треугольника из звездочек

```
*
**
***
****
*****
```

используя цикл *for*. Введите переменную, значением которой является размер катета треугольника.

1.10.

Напишите операторы Си, которые выдают строку длины **WIDTH**, в которой сначала содержится **x0** символов '-', затем **w** символов '*', и до конца строки - вновь символы '-'. Ответ:

```
int x;
for(x=0; x < x0;     ++x) putchar('-');
for(   ; x < x0 + w; x++) putchar('*');
for(   ; x < WIDTH ; ++x) putchar('-');
putchar('\n');
```

либо

```
for(x=0; x < WIDTH; x++)
        putchar( x < x0     ? '-' :
                 x < x0 + w ? '*' :
                              '-' );
putchar('\n');
```

1.11.

Напишите программу с циклами, которая рисует треугольник:

```
                *
               ***
              *****
             *******
            *********
```

Ответ:

```
/* Треугольник из звездочек */
#include <stdio.h>
/* Печать n символов c */
printn(c, n){
        while( --n >= 0 )
                putchar(c);
}
int lines = 10;         /* число строк треугольника */
void main(argc, argv) char *argv[];
{
        register int nline;  /* номер строки */
        register int naster; /* количество звездочек в строке */
        register int i;
        if( argc > 1 )
                lines = atoi( argv[1] );
        for( nline=0; nline < lines ; nline++ ){
                naster = 1 + 2 * nline;
                /* лидирующие пробелы */
                printn(' ', lines-1  - nline);
                /* звездочки */
                printn('*', naster);
                /* перевод строки */
                putchar( '\n' );
        }
        exit(0);        /* завершение программы */
}
```

1.12.

В чем состоит ошибка?

```
main(){  /* печать фразы 10 раз */
   int i;
   while(i < 10){
       printf("%d-ый раз\n", i+1);
       i++;
   }
}
```

Ответ: автоматическая переменная **i** не была проинициализирована и содержит не 0, а какое-то произвольное значение. Цикл может выполниться не 10, а **любое** число раз (в том числе и 0 по случайности). Не забывайте инициализировать переменные, возьмите описание с инициализацией за **правило**!

```
int i = 0;
```

Если бы переменная **i** была статической, она бы имела начальное значение 0.

В данном примере было бы еще лучше использовать цикл *for*, в котором все операции над индексом цикла собраны в одном месте - в заголовке цикла:

```
for(i=0; i < 10; i++) printf(...);
```

1.13.

Вспомогательные переменные, не несущие смысловой нагрузки (вроде счетчика повторений цикла, не используемого в самом теле цикла) принято по традиции обозначать однобуквенными именами, вроде **i**, **j**. Более того, возможны даже такие курьезы:

```
main(){
  int  _ ;
  for( _ = 0; _ < 10; _++) printf("%d\n", _ );
}
```

основанные на том, что подчерк в идентификаторах - полноправная буква.

1.14.

Найдите 2 ошибки в программе:

```
main(){
        int x = 12;
        printf( "x=%d\n" );
        int y;
        y = 2 * x;
        printf( "y=%d\n", y );
}
```

Комментарий: в теле функции все описания должны идти перед всеми выполняемыми операторами (кроме операторов, входящих в состав описаний с инициализацией). Очень часто после внесения правок в программу некоторые описания оказываются после выполняемых операторов. Именно поэтому рекомендуется отделять строки описания переменных от выполняемых операторов **пустыми строками** (в этой книге это часто не делается для экономии места).

1.15.

Найдите ошибку:

```
int n;
n = 12;
main(){
        int y;
        y = n+2;
        printf( "%d\n", y );
}
```

Ответ: выполняемый оператор n=12 находится вне тела какой-либо функции. Следует внести его в main() после описания переменной y, либо переписать объявление перед main() в виде

```
int n = 12;
```

В последнем случае присваивание переменной n значения 12 выполнит компилятор еще во время компиляции программы, а не сама программа при своем запуске. Точно так же происходит со всеми статическими данными (описанными как *static*, либо расположенными вне всех функций); причем если их начальное значение не указано явно - то подразумевается 0 ('\0', NULL, ""). Однако нулевые значения не хранятся в скомпилированном выполняемом файле, а требуемая "чистая" память расписывается при старте программы.

1.16.

По поводу описания переменной с инициализацией:

```
TYPE x = выражение;
```

является (почти) эквивалентом для

```
TYPE x;         /* описание */
x = выражение;  /* вычисление начального значения */
```

Рассмотрим пример:

```
#include <stdio.h>
extern double sqrt();   /* квадратный корень */
double x   = 1.17;
double s12 = sqrt(12.0);            /* #1 */
double y   = x * 2.0;               /* #2 */
FILE  *fp  = fopen("out.out", "w"); /* #3 */
main(){
  double ss = sqrt(25.0) + x;       /* #4 */
  ...
}
```

Строки с метками #1, #2 и #3 ошибочны. Почему?

Ответ: при инициализации статических данных (а **s12**, **y** и **fp** таковыми и являются, так как описаны вне какой-либо функции) выражение должно содержать только константы, поскольку оно вычисляется КОМПИЛЯТОРОМ. Поэтому ни использование значений переменных, ни вызовы функций здесь недопустимы (но можно брать адреса от переменных).

В строке #4 мы инициализируем автоматическую переменную **ss**, т.е. она отводится уже во время **выполнения** программы. Поэтому выражение для инициализации вычисляется уже не компилятором, а самой программой, что дает нам право использовать переменные, вызовы функций и.т.п., то есть выражения языка Си без ограничений.

1.17.

Напишите программу, реализующую эхо-печать вводимых символов. Программа должна завершать работу при получении признака *EOF*. В *UNIX* при вводе с клавиатуры признак *EOF* обычно обозначается одновременным нажатием клавиш *CTRL* и *D* (*CTRL* чуть раньше), что в дальнейшем будет обозначаться *CTRL/D*; а в *MS DOS* - клавиш *CTRL/Z*. Используйте *getchar*() для ввода буквы и *putchar*() для вывода.

1.18.

Напишите программу, подсчитывающую число символов поступающих со стандартного ввода. Какие достоинства и недостатки у следующей реализации:

```
#include <stdio.h>
main(){ double cnt = 0.0;
        while (getchar() != EOF) ++cnt;
        printf("%.0f\n", cnt );
}
```

Ответ: и достоинство и недостаток в том, что счетчик имеет тип *double*. Достоинство можно подсчитать **очень** большое число символов; недостаток - операции с *double* обычно выполняются **гораздо медленнее**, чем с *int* и *long* (до десяти раз), программа будет работать дольше. В повседневных задачах вам вряд ли понадобится иметь счетчик, отличный от *long* **cnt**; (печатать его надо по формату "*%ld*").

1.19.

Составьте программу перекодировки вводимых символов со стандартного ввода по следующему правилу:

```
        a -> b
        b -> c
        c -> d
        ...
        z -> a
 другой символ -> *
```

Коды строчных латинских букв расположены подряд по возрастанию.

1.20.

Составьте программу перекодировки вводимых символов со стандартного ввода по следующему правилу:

```
            B -> A
            C -> B
            ...
            Z -> Y
     другой символ -> *
```

Коды прописных латинских букв также расположены по возрастанию.

1.21.

Напишите программу, печатающую номер и код введенного символа в восьмеричном и шестнадцатеричном виде. Заметьте, что если вы наберете на вводе **строку** символов и нажмете клавишу *ENTER*, то программа напечатает вам на один символ больше, чем вы набрали. Дело в том, что код клавиши *ENTER*, завершившей ввод строки – символ '\n' **тоже** попадает в вашу программу (на экране он отображается как перевод курсора в начало следующей строки!).

1.22.

Разберитесь, в чем состоит разница между символами '*0*' (цифра нуль) и '*\0*' (нулевой байт). Напечатайте

```
      printf( "%d %d %c\n", '\0', '0', '0' );
```

Поставьте опыт: что печатает программа?

```
    main(){
            int c = 060;  /* код символа '0' */
            printf( "%c %d %o\n", c, c, c);
    }
```

Почему печатается 0 48 60? Теперь напишите вместо

```
    int c = 060;
```

строчку

```
    char c = '0';
```

1.23.

Что напечатает программа?

```
    #include <stdio.h>
    void main(){
            printf("ab\0cd\nxyz");
            putchar('\n');
    }
```

Запомните, что '\0' служит признаком конца строки в памяти, а '\n' – в файле. Что в строке "abcd\n" на конце неявно уже расположен нулевой байт:

```
    'a','b','c','d','\n','\0'
```

Что строка "ab\0cd\nxyz" – это

```
    'a','b','\0','c','d','\n','x','y',z','\0'
```

Что строка "abcd\0" – избыточна, поскольку будет иметь на конце два нулевых байта (что не вредно, но зачем?). Что *printf* печатает строку до нулевого байта, а не до закрывающей кавычки. Программа эта напечатает **ab** и перевод строки.

Вопрос: чему равен *sizeof*("ab\0cd\nxyz")? Ответ: 10.

1.24.

Напишите программу, печатающую целые числа от 0 до 100.

1.25.

Напишите программу, печатающую квадраты и кубы целых чисел.

1.26.

Напишите программу, печатающую сумму квадратов первых n целых чисел.

1.27.

Напишите программу, которая переводит секунды в дни, часы, минуты и секунды.

1.28.

Напишите программу, переводящую скорость из километров в час в метры в секундах.

1.29.

Напишите программу, шифрующую текст файла путем замены значения символа (например, значение символа **C** заменяется на **C**+1 или на ~**C**).

Назад | Содержание | Вперед

[Главная] [Гостевая]