Введение в язык программирования С.

Лекция 4

Поток управления. Функции.

Сергей Леонидович Бабичев

Оператор for

- Цикл в теле другого цикла называют *вложенным*.
- Уровней вложенности может быть много.

```
#include <stdio.h>

int main() {
    int count = 0, i1, i2, i3, i4, i5, i6;
    for (i1 = 0; i1 < 10; i1++)
        for (i2 = 0; i2 < 10; i2++)
            for (i3 = 0; i3 < 10; i3++)
                for (i4 = 0; i4 < 10; i4++)
                    for (i5 = 0; i5 < 10; i5++)
                        for (i6 = 0; i6 < 10; i6++)
                            count += i1+i2+i3 == i4+i5+i6;
    printf("Total %d lucky tickets\n", count);
}
```

Оператор break

Форма записи проста:

```
break;
```

- В боксе это слово говорит рефери говорит это слово, чтобы разнять боксёров.
- В Си его действие похоже на действие рефери: прекращается какое-то действие, например, прекращается очередная итерация любого цикла и управление передаётся на первый оператор, следующий за телом цикла.

```
for( ; ; ) {
   // ...
   if (x > 0) break;
   // Here x <= 0.
}
```

Оператор break

Ещё раз перепишем программу по вычислению экспоненты:

```
double sum = 1;
int n = 1;
double el = 1;
for (;;) {
    el *= x / n;
    if (el < 1e-7) break;
    sum += el;
    n++;
}
```

Оператор `break`: важное правило

Если имеется несколько циклов, вложенных друг в друга, то оператор `break` принудительно завершает только самый **внутренний** цикл.

Оператор continue

- Если `break` принудительно *завершает* цикл, то `continue` принудительно *продолжает* цикл.

Наивная задача. Вывести все числа от 1 до 100, которые делятся на 3, но не делятся на 5.

```
#include <stdio.h>

int main() {
    int x;
    for (x = 3; x <= 100; x += 3) {
        if (x % 5 == 0) continue;
        printf("%d ", x);
    }
}
```

Оператор switch

Задача. На вход алгоритма поступает число `10 <= n <= 30`. На выходе требуется получить истину, если число простое и ложь в противном случае.

- Первая идея — проверять число на делимость на простые множители.
- Вторая — воспользоваться ограниченностью множества решений.

Оператор switch: решение задачи

```
// input: n
// output: ans
int ans;
switch (n) {
case 11:
case 13:
case 17:
case 19:
case 23:
case 29:
    ans = 1;
    break;
default:
    ans = 0;
    break;
}
```

Оператор switch

Состоит из:

- *заголовка* (`switch (expr)`) в круглых скобках и *тела*.
- Выражение `expr` должно иметь перечислимое значение.
- Тело состоит из *случаев* `case`, с обязательным двоеточием. Ещё их называют *метки case*.
- Случаев может быть любое количество, в них должны фигурировать обязательно **константы**.
- Мы просматриваем случаи сверху вниз, и как только определилось, что значение `expr` совпадает с одной из констант, перечисленных в случаях, начинается исполняться последовательность операторов, следующая за данной меткой.
- Исполнение прекращается или когда мы доходим до самого конца оператора `switch`, или до оператора `break`, или исполняется оператор `return`.

Оператор switch: примеры

Что здесь происходит?

```
double p = 1.;
double q = 2.72;
switch (n) {
case 8: p *= q;
case 7: p *= q;
case 6: p *= q;
case 5: p *= q;
case 4: p *= q;
case 3: p *= q;
case 2: p *= q;
case 1: p *= q;
default:
   break;
}
```

Оператор switch: примеры

Вот ещё один пример:

```
switch (n) {
case 0:
   printf("Zero");
   break;
 case 1:
   printf("One");
   break;
 case 2:
   printf("Two");
   break;
 default:
   printf("What??? Unknown number!");
   break;
}
```

Функции

- Функция в Си — основной строительный блок.
- Функции расширяют возможности языка, добавлением новых *абстракций*.
- Всё взаимодействие любой программы с окружением происходит через функции.
- Используемая функция должна быть где-то *определена*.
- *Определение* функции задаёт её *имя*, *параметры* и *тип возвращаемого значения*.
- Обычно функция как-то обрабатывает свои *параметры* и *возвращает значение*.
- Одним из способов определения функции является помещение её *тела* в *исходном файле* программы где-нибудь *перед* её *использованием*.

Функции: пример определения

- В Си нет встроенной операции возведения числа в квадрат, которая есть в Pascal.
- Её очень легко дописать.

```
double sqr(double x) {
    double result = x * x;
    return result;
}
```

- x — параметр функции с объявленным типом double.
- В теле функции мы заводим любое количество *локальных* переменных, которые существуют только внутри функции.

Функции: пример использования

- Использовать её можно, например, присвоив какой-либо переменной результат её *вызова*:

```
double t2 = sqr(t);
```

- Чтобы вызов был успешным, компилятору надо знать как тип возвращаемого значения функции, так и типы всех её аргументов.
- Если функция *определена* выше *точки вызова*, то у компилятора достаточно информации для того, чтобы проверить правильность её вызова.
- Функция *определяется* где-то точки вызова, то компилятор может сделать собственные предположения о том, что это за функция.
- Эти предположения могут не совпасть с тем, что он увидит, как только доберётся до её определения.
- Можно помочь компилятору, поместив *объявление* или *прототип* функции перед точкой её вызова. Для нашей функции `sqr` объявление будет выглядеть так:

```
double sqr(double t);
```

Функция: полный пример использования

```
#include <stdio.h>

double sqr(double); // prototype or declaration

int main() {
    double a = 1234.567;
    double a2 = sqr(a); // usage
    printf("a^2=%lf\n", a2);
}

double sqr(double x) { // definition
    double result = x * x;
    return result;
}
```

Функции

- Функция может иметь любое число параметров.
- Они перечисляются через запятую, каждый со своим типом.
- Перед именем функции — тип её возвращаемого значения.
  Правильно:

```
int max3(int a, int b, int c);
```

  Неправильно:

```
int max3(int a,b,c);
```

Следующая функция принимает координаты двух точек на плоскости и возвращает расстояние между ними. Она использует *библиотечную* функцию `sqrt`.

```
double distance(double x1, double y1, double x2, double y2) {
    return sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
}
```

Функции

- При объявлении функций допустимо опустить имена переменных, оставив только их типы:

  ```
  double sqr(double);
  ```

- Имена переменных лучше всё же писать, выбирая что-то осмысленное.
- Какое объявление функции `distance` более понятно?

  ```
  double distance(double x1, double y1, double x2, double y2);
  double distance(double, double, double, double);
  ```

Передача аргументов в функцию

- То, что приходит в параметры функций, считаются переменными, которым присвоены начальные значения. Они приходят из *точки вызова*.
- В месте вызова функции *аргументы* вызова должны соответствовать *параметрам*.

```
int max3(int a, int b, int c) {
  // ...
}


  ...
  int a = 33;
  int b = 77;
  int m = max3(10, a+5, b);
```

- Здесь значением параметра a в функции max3 будет 10, параметра b будет 38 (сумма значения переменной a в точке вызова и 5), а значением параметра c — 77.

Параметры функции как переменные

- Функции могут распоряжаться аргументами, полученными из точки вызова, как им заблагорассудится.
- Они могут их изменять, при этом то, что передано в функцию в точке вызова, не изменится.

```
void foo(int x) {
  x = 10;
}


...
int a = 33;
foo(a);
// Here a still equal 33.
```

Если мы хотим, чтобы функция изменила значение переменной, которую мы в неё передали, нужно воспользоваться *указателями*.

Функции и ключевое слово *static*

- Большие проекты состоят из сотен и тысяч файлов.
- Если определить функцию в одном файле, во всех местах, где она используется, требуется её описание.
- Определять функции с одним именем и разными именами в Си нельзя.
- Некоторые функции в проекте могут использоваться исключительно в одном исходном файле.
- Если заранее известно, что именно этой функцией в других файлах пользоваться не будут, добавляют слово *static*.
- Тогда имя этой функции не будет конфликтовать с другими такими же именами в проекте.

Несколько слов о проектах

- Большие проекты пишутся несколькими людьми или группами.
- Каждая группа работает над своим набором файлов.
- Мы разделяем функции на те, которыми мы будем пользоваться только в нашей части проекта, в нашем файле (*внутренние*), и на те, которые мы специально пишем для того, чтобы другие смогли ими воспользоваться (*внешние* или *интерфейсные*).
- Большое количество внешних функций в программе может сделать невозможным стыковку двух частей большого проекта, созданных в разных файлах разными людьми из-за *коллизий* имён.
- Ключевое слово `static`, перед именем функции уменьшает эту проблему.

```
static int func(int n) {
    ...
}
```

Теперь имя `func` можно будет использовать в других файлах проекта.

Рекомендации к использованию слова `static`

Всегда используйте аттрибут `static` для тех функций, которые не планируется делать общими в проекте.

Это поможет вам, так как компилятор сможет оптимизировать использование этой функции, зная, что нигде, кроме данного файла она не доступна.

Это поможет другим, так как вы не *загрязняете пространство глобальных имён*.

Функции: несколько советов

- Используйте хорошо читаемые и осмысленные имена функций.
- Используйте хорошо читаемые и осмысленные имена параметров.
- Не пишите очень больших функций: постарайтесь, чтобы она поместилась на экране.
- Постарайтесь не использовать более 5-6 параметров в функции.
- Используйте `static` для тех функций, которые не должны быть видимы в других файлах.

Функции printf и scanf.

- Почти всё, что мы делали до этого, обрабатывало какие-то данные.
- Каждая функция принимает какие-то аргументы и выдаёт какой-то результат.
- Для того, чтобы взаимодействовать с нами нужен свой набор действий.
- В Си это — тоже функции.
- В стандарте языка имеется набор функций для простого *текстового* взаимодействия с пользователями.
- Их прототипы собраны в *заголовочный файл* `<stdio.h>`.
- Для того, чтобы начать писать более сложные программы, познакомимся с двумя из них: `printf` и `scanf`.

Внутреннее и внешнее представления

- Внутри компьютера переменные хранятся в двоичном виде.
- Нам обычно интересно *десятичное* представление.
- В десятичном представлении мы и вводим, и выводим числа в виде набора *символов*.
- Для этого приходится на выводе *преобразовывать* данные из двоичного представления в десятичные символы, на на вводе — из десятичных символов во внутреннее двоичное представление.
- Мы так и говорим: *внутреннее представление* (для компьютера) и *внешнее представление* (для нас).

Функция printf — форматирование и вывод

- Как вывести значения нескольких переменных вместе с их именами?
- Разные языки это делают по-разному.

```
Pascal: writeln('a=', a, ' b=', b, ' c=', c, ' d=', d, ' e=', e);
Python: print('a=',a,'b=',b, 'c=', c, 'd=',d, 'e=', e)
C: printf("a=%d b=%d c=%d d=%d e=%d\n", a, b, c, d, e);
```

Первый аргумент даёт общий вид вывода: вместо %d подставятся десятичные значения соответствующих переменных.

Функция printf — форматирование и вывод

- Первый аргумент — *форматная строка* в которой имеется выводимый текст ("a=")и, возможно, несколько *шаблонов* или *спецификаций формата* ("%d").
- printf следует по строке слева направо, выводя символ за символом.
- Как только встречается *метасимвол* %, он начинает собирать *шаблон*.
- Шаблон заканчивается одной из предопределённых букв, например, d означает, что вывод должен производиться в десятичной (decimal) системе счисления.
- Перед буквой, определяющей формат вывода и тип посланного в printf значения может тоже что-то находиться.

Функция printf — примеры использования

```
int  i = 123; char c = 'a'; unsigned u = 256; unsigned long ul = 4095ul;
long long ll = 65535ll; unsigned long long ull = 1024ull;
float f = 123.456; double d = 12345678.9012345;

printf("i=%d i=%4d i=%04d i=%-4d", i); // "i=123 i= 123 i=0123 i=123 "
printf("c=%c c=%d", c); // "c=a c=66"

printf("u=%u u=%o u=%x u=%X", u, u, u, u);
// "u=256 u=400 u=ff u=FF"

printf("ul=%ul ull=%ull", ul, ull); // "ul=65535 ull=1024"

printf("f=%f f=%g f=%e", f, f, f); // "f=123.456 f=1.23456e5 f=123.456"
printf("d=%lf d=%.1lf d=%.7lf d=%10.3lf", d, d, d, d);
//"d=12345678.901235 d=12345678.9 d=12345678.9012345 d=12345678.901"
printf("Result=%.2lf%%", result); // "Result=10.75%"
```

Функция scanf — форматирование и ввод

- Её формат похож на `printf`.
- Первый аргумент — форматная строка — то, что функция ожидает на вводе.
- В ней присутствуют такие же знаки процента, извещающие `scanf`, что ей потребуется ввести число в каком-то формате.
- Сами форматы в основном совпадают с теми, которые используются в `printf`.

Функция `scanf` требует, чтобы в неё передавались адреса, по которым можно присвоить введённое значение. Операция взятия адреса — символ `&` перед выражением.

Функция scanf — примеры использования

```
int  i;
char c;
unsigned u;
unsigned long ul;
long long ll;
unsigned long long ull;
float f;
double d;
int code = scanf("%d", &i);
code = scanf("%d %c %u %lu %ll %llu %f %lf",
             &i,&c,&u,&ul,&ll,&llu,&f, &d);
```

Важно: каждому элементу формата необходимо точное соответствие с типом аргумента!

Функция scanf — примеры использования

- Функция scanf возвращает число тех адресов, по которым ей удалось положить значение.
- А почему она могла не сделать какой-то работы?
- Например, потому, что мы просили число, а на входе оказалось нечто нечисловое.
- Может быть, закончились входные данные.

Пусть на входе имеется строка вида 17/12/2017. Тогда ввести её можно так:

```
int day, month, year;
code = scanf("%d/%d/%d", &day, &month, &year);
```

Если при этом code окажется равным трём, то всё ввелось успешно.