НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Дисциплина «Информатика» (основной уровень), 1-й курс, 1-й семестр.

Методические указания

Тематическое занятие 8

Функции.

Содержание

Создание функций

Предназначение функций

При создании программы для решения сложной задачи выполняется разделение (*декомпозиция*) задачи на подзадачи, а подзадач – на еще меньшие фрагменты и так далее, до элементарных подзадач, которые легко программировать.

Все языки высокого уровня содержат средства для разделения программ на самостоятельные части – *подпрограммы*. В языке С подпрограммы называются ***функциями*****.** Функция позволяет скомпоновать группу операторов для выполнения единого действия, чтобы затем обращаться к ней столько раз, сколько потребуется.

Обычно программы на языке С состоят из множества мелких функций. Даже действие, которое выполняется только один раз, следует выделять в отдельную функцию, если это проясняет структуру и смысл кода программы.

Функция должна иметь уникальное имя (идентификатор), которое употребляется для ее ***вызова***. Такой вызов функции можно выполнять многократно. Функции можно передавать данные и она может возвращать результаты вычислений.

Функции делятся на *стандартные* и определенные программистом. Стандартные функции являются частью стандартной библиотеки С, они объявлены в соответствующих стандартных *заголовочных файлах*. Например, функции `printf` и `scanf`, объявлены в заголовочном файле `<stdio.h>`.

Функции позволяют программистам пользоваться разработками предшественников, а не начинать все заново. Если функция составлена правильно, то нет нужды знать ***как*** она работает, достаточно знать ***что*** именно она делает.

Определение функций

Определение функции имеет следующую форму:

```
ТипВозвращЗнач ИмяФункции(ОбъявленияПараметров)
{
    /* объявления и операторы (тело функции) */
}
```

Например, составим свою функцию для возведения числа ***a*** в целую положительную степень ***n***. Наша функция `power`, будет возвращать значение a^n, а числа `a` и `n` будем передавать в нее в качестве параметров `power(a,n)`:

```
int power(int a, int n) /* заголовок функции power */
{
    int i, p=1; /* локальные переменные */
    for (i=1; i<=n; ++i)
        p*=a;
    return p; /* возвращаемое значение */
}
```

Чтобы использовать эту функцию, ее нужно поместить в программу и вызвать из главной функции `main()`.

Функции и структура программы

В языке С нельзя определить функцию внутри другой функции. Программа на С состоит из функций, которые являются внешними по отношению друг к другу. При этом каждая функция может вызывать каждую другую.

Определения функций могут следовать в произвольном порядке, а также находиться в нескольких файлах исходного кода программы. Пока будем полагать, что все функции находятся в одном файле.

Главная функция программы `main()` подчиняется тем же правилам. Но при запуске программы выполняется только функция `main()`, а остальные функции могут быть вызваны из нее напрямую или посредством других функций.

Составим программу, которая вызывает нашу функцию `power(a,n)` для вычисления значения x^y :

```
#include <stdio.h>

int power(int a, int n); /* прототип функции power */
```

```
int main(void) /* главная функция программы */
{
     int x, y, res; /* локальные переменные */
     printf("Input x: "); scanf("%d", &x);
     printf("Input y: "); scanf("%d", &y);
     res = power(x,y); /* вызов функции power */
     printf("Result x^y=%d\n", res);
     return 0;
}
int power(int a, int n) /* заголовок функции power */
{
     int i, p=1; /* локальные переменные */
     for (i=1; i<=n; ++i)
          p*=a;
     return p; /* возвращаемое значение */
}
```

Прототипы функций

В данном случае определение функции `power` следует после главной функцией `main()`, поэтому до функции `main()` необходимо описать ***прототип*** функции `power` – следующую строку:

```
int power(int a, int n);
```

Прототип сообщает, что `power` является функцией с двумя параметрами типа `int`, возвращающей значение тоже типа `int`.

Прототип функции обязан согласовываться по форме как с заголовком в определении функции, так и с каждым ее вызовом. В противном случае возникает ошибка.

При этом имена параметров согласовывать не обязательно, можно даже не указывать их. Поэтому прототип функции `power` можно записать так:

```
int power(int, int);
```

Теперь переработаем функцию `main()`, чтобы продемонстрировать многократный вызов функции `power`:

```
#include <stdio.h>
int power(int, int); /* прототип функции power */
int main(void) {
     int x, y;
     printf("Calculate x^y (input x=0,y=0 for finish).\n");
     do {
          printf("Input x: "); scanf("%d", &x);
          printf("Input y: "); scanf("%d", &y);
          if (y>=0 && (x||y)) {
               printf("Result x^y=%d\n", power(x,y));
          } else
               printf("Error: Incorrect value.\n");
     } while (x||y);
     return 0;
}
```

```
int power(int a, int n) { /* заголовок функции power */
     int i, p=1; /* локальные переменные */
     for (i=1; i<=n; ++i)
          p*=a;
     return p; /* возвращаемое значение */
}
```

Здесь функция `power` вызывается непосредственно как один из параметров стандартной функции `printf`.

Передача параметров

Параметры функций и локальные переменные

Параметры, объявленные в заголовке функции при ее определении, называют ***формальными параметрами***. При объявлении параметры указывают через запятую, каждый со своим типом.

Переменные, массивы и другие данные, описанные внутри функции, называются ***локальными***.

В рассмотренном примере `a` и `n` – формальные параметры функции `power`, а `i` и `p` – локальные переменные. И те, и другие доступны только в пределах этой функции и не доступны из вызывающей функции.

Вызов функций

Вызов функций осуществляется указанием ее имени и списка ***фактических параметров*** через запятую.

Фактические параметры – это параметры, которые передаются функции при ее вызове. Фактические параметры так же называют ***аргументами*** функции.

Количество, тип и порядок следования формальных и фактических параметров должны совпадать.

При вызове функции `power` работа главной функции `main()` приостанавливается, и управление вычислительным процессом передается на участок кода, занимаемый функцией `power`. После окончания выполнения функции `power`, управление возвращается в точку вызова, и работа функции `main()` продолжается.

Оператор `return`

Оператор `return` – это механизм возвращения значения из вызываемой функции в вызывающую. После `return` может идти любое выражение, которое преобразуется к типу, возвращаемому функцией согласно ее определению.

```
return Выражение;
```

Если после `return` выражение не указано, то в вызывающую функцию ничего не передается.

После выполнения оператора `return` выполнение функции прекращается, и управление передается в точку ее вызова. То же самое происходит. если достигнут конец тела функции.

Отсутствие параметров

Если в процедуру или функцию не передаются данные, то в ее заголовке необходимо указать отсутствие значения с помощью типа `void`. Например:

```
int myfunc(void) {
     ...
     return ...
}
```

При вызове такой функции скобки опускать нельзя, но их нужно оставлять пустыми

```
f = myfunc();
```

Тип `void` также следует указывать, если функция не возвращает никаких значений. Пример определения минимальной функции-заглушки:

```
void myfunc(void) {}
```

и вызова этой функции

```
myfunc();
```

Способы передачи параметров

Передача по значению

В языке С все параметры функций передаются ***«по значению»***. Это означает, что вызываемая функция получает значения своих параметров в виде временных копий передаваемых переменных. Поэтому вызываемая функция не может изменить значения переменных в вызывающей функции, она вправе изменять только локальные временные копии этих переменных.

Таким образом, при работе с формальным параметром используется **копия фактического параметра.** Значения формальных параметров может изменяться в функции, но на значениях фактических параметров это **не отражается.**

Например, нужно составить функцию, которая меняет местами значения двух переданных в нее переменных:

```
void swap(int x, int y) { /* НЕПРАВИЛЬНО - передача по значению */
     int temp;
     temp = x;
     x = y;
     y = temp;
}
```

Если теперь вызвать данную функцию `swap` с фактическими параметрами `a` и `b`

```
int a=2, b=3;
swap(a, b);
```

то она **не измени**т значений переменных `a` и `b`.

Передача по адресу

Обойти данные ограничения позволяет передача из вызывающей функции не самих переменных, а *указателей* на эти переменные. А в самой функции

соответствующие формальные параметры необходимо объявить указателями. Такой способ называют передачей ***«по адресу»***, поскольку в функцию передаются адреса переменных.

Для приведенного примера:

```
void swap(int *px, int *py) { /* ПРАВИЛЬНО – передача по адресу */
     int temp;
     temp = *px;
     *px = *py;
     *py = temp;
}
```

Теперь вызов функции `swap` с адресами переменных `a` и `b` в качестве фактических параметров

```
int a=2, b=3;
swap(&a, &b);
```

приведет к изменению значений переменных. В результате: `a=3` и `b=2`.

Примеры передачи параметров

Несколько возвращаемых значений

Способ передачи параметров по адресу используется, если необходимо, чтобы функция возвращала не одно, а несколько значений.

Допустим, требуется составить одну функцию, которая возвращает сумму и произведение двух переданных в нее чисел. Для решения этой задачи недостаточно использовать механизм `return`, который позволяет вернуть только одно значение. Организуем возврат двух значений через параметры.

Определим у функции четыре формальных параметра: два исходных числа, сумма и произведение. Первые два параметра `x` и `y` – два исходных числа – передадим по значению, поскольку внутри функции они изменяться не будут. А оставшиеся два параметра `sum` и `prod` вычисляются самой функцией, их значения изменятся, поэтому их необходимо передавать по адресу.

```
#include <stdio.h>
void calcul(int x, int y, int *sum, int *prod); /* прототип функции */
int main(void) { /* главная функция программы */
     int a, b, s, p; /* фактические параметры */
     printf("Input a: "); scanf("%d", &a);
     printf("Input b: "); scanf("%d", &b);
     calcul(a,b,&s,&p); /* вызов функции */
     printf("a+b=%d,  a*b=%d.\n", s, p);
     return 0;
}
void calcul(int x, int y, int *sum, int *prod) { /* заголовок функции */
     *sum = x + y;
     *prod = x * y;
}
```

Здесь фактические параметры `s` и `p`, переданные по ссылке, принимают новые значения после выполнения функции. При этом сама функция `calcul()` не возвращает никаких значений (`void`).

Задача об отслеживании переполнения

Рассмотрим подробно ход решения задачи отслеживания переполнения.

Пусть требуется отслеживать переполнение при арифметическом сложении двух целых положительных чисел. Составим отдельную функцию для реализации такой операции. Очевидно, что эта функция должна получать оба слагаемых, и возвращать результат сложения.

Однако операция сложения не может быть выполнена, если слагаемые настолько велики, что при их суммировании происходит переполнение типа данных, используемого для хранения результата. Поэтому кроме результата функция должна возвращать еще ***код ошибки*** (точнее, ***код завершения операции***). Например, если переполнения не произошло, то значение кода равно 0, в противном случае – оно равно 1.

Таким образом, в ходе работы такой функции *могут произойти два события:* **а)** переполнения не произошло, тогда функция возвращает код ошибки 0 и результат операции; **б)** произошло переполнение, и функция возвращает только код ошибки 1, результат операции не определен.

Теперь определим, какие параметры нужно передавать в функцию и что должна возвращать сама функция в точку вызова.

Оба слагаемых не изменяются в процессе вычисления, поэтому их можно передать в функцию в качестве двух обычных параметров, *по значению.*

Функция должна возвращать два значения – код ошибки и результат сложения. Одно из этих значений вернет сама функция, а другое придется возвращать в качестве третьего параметра, который будет передаваться в функцию *по адресу.*

Сама функция всегда возвращает некоторое значение, поэтому *в качестве значения самой функции следует взять код ошибки,* который тоже всегда принимает значение (0 или 1). Результат сложения не будет принимать никакого значения при переполнении, поэтому его удобнее возвращать через третий параметр, *по адресу.* Т.е. значение этого третьего параметра функция будет изменять только, если не произошло переполнения.

Реализуем описанное в программе:

```
#include <stdio.h>
#include <limits.h>   /* (содержит объявление INT_MAX) */
int add(int a, int b, int *s); /* прототип функции */
int main(void) { /* главная функция программы */
     int x, y, sum=0;
     printf("Input x: "); scanf("%d", &x);
     printf("Input y: "); scanf("%d", &y);
     if (add(x,y,&sum)) /* вызов функции, проверка кода ошибки */
          printf("Error: addition overflow.\n");
     else
          printf("Result x+y=%d\n", sum);
     return 0;
}
```

```
int add(int a, int b, int *s) { /* заголовок функции */
     if (INT_MAX-a>=b) /* INT_MAX – макс. значение типа int */
          *s = a+b;
     else
          return 1; /* возвращаемое значение кода ошибки */
     return 0; /* возвращаемое значение, если ошибки не произошло */
}
```

Обратите внимание, что в приведенном примере для использования функции *программисту нет нужды знать **как** она работает, достаточно знать **что** именно она делает.* Данным принципом следует всегда руководствоваться при реализации функций.

Поэтому, в частности, все сообщения для пользователя выводятся на экран не из вызываемой функции, а из вызывающей главной функции `main()`.

Возврат кода ошибки

Обычно значения, которые возвращает сама функция (с помощью оператора `return`), используются для отслеживания ***кодов ошибок*** выполнения этой функции.

Например, сделаем проверку переполнения типа данных `int` для обеих операций, выполняемых функцией `calcul()`.

```
#include <stdio.h>
#include <limits.h>
int calcul(int x, int y, int *sum, int *prod); /* прототип функции */
int main(void) /* главная функция программы */
{
     int a, b, s=0, p=0; /* фактические параметры */
     int error_code; /* код ошибки */
     printf("Input a: "); scanf("%d", &a);
     printf("Input b: "); scanf("%d", &b);
     error_code = calcul(a,b,&s,&p); /* вызов функции */
     if (error_code) /* проверка кода ошибки */
          printf("Overflow error: %d.\n", error_code);
     else
          printf("a+b=%d,  a*b=%d.\n", s, p);
     return 0;
}
int calcul(int x, int y, int *sum, int *prod) /* заголовок функции */
{
     if (x<=INT_MAX-y) /* проверка переполнения для сложения */
          *sum = x + y;
     else
          return 1; /* возврат кода ошибки переполнения сложения */
     if (x<INT_MAX/y+1) /* проверка переполнения для умножения */
          *prod = x * y;
     else
          return 2; /* возврат кода ошибки переполнения умножения */
     return 0; /* возврат кода успешных операций (без переполнения)*/
}
```

Теперь сама функция `calcul()` отслеживает переполнение и возвращает код ошибки той операции, при которой оно произошло (1 – при сложении, 2 – при умножении).

Упражнения

Упражнение 8.1

Составить программу, которая содержит функцию, переводящую размер в дюймах в метры. Пользователь вводит размер в дюймах, вызывается функция, которая переводит его в метры. Результат, возвращаемый функцией, выводится на экран в основной программе.

Упражнение 8.2

Составить программу, которая содержит функцию, вычисляющую периметр и площадь прямоугольника. Пользователь вводит размеры двух сторон прямоугольника ***a*** и ***b***. Вызывается функция, которая вычисляет периметр ***P*** и площадь ***S*** прямоугольника. Значения ***P*** и ***S***, возвращаемые функцией, выводится на экран в основной программе.

Упражнение 8.3

Реализовать проверку переполнения при вычислении значений ***P*** и ***S*** в функции, описанной в упражнении ***8.2***.