**Функции в языке С**

Рассмотрим пример программы, которая использует функцию для вычисления куба числа:

**#include <stdio.h>**

**long cube (long x); /прототип функции/**

**long input, answer;**

**int main ()**

**{**

**printf (“введите число:”);**

**scanf(%d“, & input);**

**answer=cube(input); /вызов функции cube/**

**printf(“ \n куб числа %d равен %d ”, input, answer);**

**return 0;**

**}**

**long cube (long x) /определение функции/**

**{**

**long x\_cubed; /объявление локальной переменной/**

**x\_cubed=x\*x\*x;**

**return x\_cubed; /оператор предает в вызывающую программу значение переменной x\_cubed/**

**}**

Использование функций в программе позволяет практиковать подход структурного программирования. При этом отдельные задачи программы выполняются независимыми фрагментами кода. Используя структурный подход, программирование происходит фактически сверху вниз (дерево). Как правило, вся фактическая работа выполняется функциями на концах «ветвей» дерева. Функции, находящиеся ближе к «стволу», в основном распределяют выполнение программы между этими функциями.

Написание функции

Прототип функции всегда должен быть объявлен до начала функции main. Имеет следующий синтаксис описания:

**Тип Имя (Тип1 Параметр1, … Типj Параметрj);**

Объявление функции:

**Тип Имя (Тип1 Параметр1, … Типj Параметрj)**

**{**

**Операторы;**

**return (значение);**

**}**

*Тип –* тип функции, тип значения, которое функция возвращает. Если функция не возвращает значение, то ее тип – **void.** В теле функции инструкцию return в этом случае не пишут.

int f1(…)

void f2(…)

*Имя* - имя функции.

*Типj, Параметрj*- тип и параметр функции. Если параметр используется для возврата результат, то параметр должен быть ссылкой, т.е. перед именем параметра должен быть символ \*.

В функцию можно передавать аргумент любого типа. Информация о типах приводится в заголовке функции в виде списка параметров:

**void f2(int x, float y, char z)**

Если функция вообще не принимает аргументов тогда список параметров должен состоять из ключевого слова void (или пустые круглые скобки).

**int f3(void)**

**float f4()**

Если указатель типа в заголовке отсутствует, то предполагается что функция возвращает целое значение. Список формальных параметров может отсутствовать, но () сохраняются. После заголовка ; не ставится. Формальные параметры в списке перечисляются через ,. Для возврата значения вычислений функции используется специальный оператор RETURN. Он может использоваться в форме return 0 или return выражение. В первом случае оператор завершает выполнение функции и возвращает управление вызывающей его программе. Во втором случае оператор передает в качестве результата значение заданного выражения.

**int fact(n)**

**{**

**int i, j;**

**y=1;**

**for(i=1; i<=n; i++) y=y\*I;**

**return y;**

**}**

Пример описания функции сравнения двух строк в соответствии с лексикографическим порядком:

strcmp(s,t)

char s[ ],t[ ];

{

int I;

i=0;

while (s[i]==t[i])

if (s[i++]==’\0’)

return (0);

return (s[i]-t[i]);

}

Эта функция будет выдавать значение 0 если s и t совпадают т.е. длина этих строк одинаковая. Функция возвращает значение >0 если s>t и <0 если t<s.

Описание функции является блоком. Все переменные в ней считаются локальными объектами. действующими только внутри описанной функции.

Для возврата значения из функции используется оператор return, за которым следует выражение. Когда выполнение программы подходит к оператору return выражение вычисляется и передается в вызывающую программу одновременно с передачей в нё управления. В функции может быть несколько операторов return. Фактически выполняется только один, до которого быстрее всего дошло выполнение.

**#include <stdio.h>**

**int x, y, z;**

**int larger\_of(int a, inr b);**

**int main()**

**{**

**puts(“Enter two values:”);**

**scanf(“%d%d”, &x, &y);**

**z= larger\_of(x, y);**

**printf(“\n The larger value is %d.”, z);**

**return 0;**

**}**

**int larger\_of(int a, inr b)**

**{**

**if (a>b)**

**return a;**

**else**

**return b;**

**}**

Передача аргументов в функцию

Количество аргументов и их типы должны совпадать со списком параметров в прототипе и заголовке функции. Например, если объявлено, что функция принимает 2 аргумента типа int, то в неё следует передавать именно 2 аргумента типа int. Каждый аргумент может быть любым выражением на языке С: константой, переменной, арифметическим или логическим выражением или даже вызовом другой функции.

Вызов функции можно осуществить двумя способами:

1. назвав её по имени со списком аргументов

**wait(12)**

1. указав функцию в выражении

**printf(“Half of%d is %d”, x, half\_of(x))**

В языке допускается использование рекурсивных функций, т.е. описание функций вызывающих самих себя. Для обращения к подобным функциям используются два способа:

1. имя\_функции(a1, a2, …, an) т.е. указывается имя функции, а в () перечисляются её фактические параметры.
2. (\*указатель\_ на\_функцию) (a1, a2, …, an)

Указатель\_ на\_функцию это переменная, содержащая адрес функции. Адрес функции в переменную может быть присвоен с помощью конструкции:

указатель\_ на\_функцию=имя\_функции

При вызове функции значение её параметров передаются следующим способом. В начале для каждого формального параметра как бы заводится одноименная ему локальная переменная. Затем в локальную переменную засылается значение соответствующих фактических параметров, а затем выполняются операторы, описанные в функции. Этот способ передачи называется способом передачи по значению. В соответствии с этим следует помнить, что если любой формальный параметр должен возвращать результат выполнения функции, то соответствующий ему фактический параметр должен быть указателем, а не обычной переменной.

Пример: функция, которая меняет значение двух заданных переменных целого типа a и b.

void swap(a,b)

int \*a, \*b;

{

int z;

z=\*a;

\*a=\*b;

\*b=z;

}

Если в программе потребуется поменять значениями x и y, то обращаться к функции надо следующим образом swap(&x, &y) и результат возвратиться в основную программу.

**Классы переменных**

Все переменные а программах языка С подразделяются на глобальные и локальные. Глобальные переменные определяются на том же уровне, что и функции. Т.е. их описание не содержится внутри описания других функций. Если программа состоит из нескольких файлов, то глобальная переменная определяется в одном из них, а в остальных файлах должна использоваться конструкция

**extern <имя><имя переменной>**

Переменные, описанные внутри функций, называют локальными. Локальные переменные действуют только внутри блока, в котором описаны. Все локальные переменные существуют только во время выполнения блока, где они описаны. Поэтому при повторном входе в блок эти переменные создаются заново, а при выходе как бы перестают существовать. К локальным переменным относят формальные параметры процедур. Локальные переменные делятся на:

* автоматические,
* регистровые,
* статические.

Локальные переменные, определяемые обычным образом являются автоматическими. Они являются временными. Автоматические переменные имеют приоритет перед глобальными переменными, т.е. если Х-глобальная и х-локалная переменные определены в некотором блоке, то внутри этого блока х-будет использоваться как локальная переменная.

Переменные называются регистровыми если описаны в блоке так:

**registers int<список переменных>**

они также временные, но для хранения их значений компилятор пытается отвести внутренние регистры. Использование регистров позволяет значительно ускорить доступ к значению переменных т.к. эти значения будут размещаться в регистрах, а не в ячейках памяти. Однако т.к. число регистров не велико и обычно ограничивается 10-20 в зависимости от типа ЭВМ. Т.о. число регистровых переменных может быть небольшим. Взять адрес у регистровых переменных нельзя.

Статические переменные определяются следующим образом:

**static<тип><список переменных>**

Статические переменные являются постоянными, т.е. их значения не теряются при выходе из функции. Это означает, что повторном входе в блок, где определена такая переменная она будет иметь то значение, которое имела при последнем выходе из блока. Статической может быть и глобальная переменная, но тогда ее можно использовать только в файле, где она описана.

Описание внешних объектов

Программа на языке С может состоять и одного или нескольких файлов. Каждый из указанных файлов может транслироваться и отлаживаться не зависимо от других. Затем программа собирается в единое целое с помощью редактора внешних связей(компановщик). Объекты, которые определены в одном из файлов, но используются и в других должны быть определены в других как внешние. Для этого используется ключевое слово extern, которое должно стоять перед описанием имени внешнего объекта по отношению к заданному файлу.

**extern int g1, g2;**

**extern double funct()** – это определение внешней функции

т.е. функция возвращает значение из другого файла, причем этот файл будет присоединен к программе на стадии ее компоновки. Можно использовать внешние массивы. Для одномерного внешнего массива в конструкции extern их размер можно опускать.