**Функции**

Функция – это независимая совокупность объявлений и операторов, предназначенных для выполнения определенной задачи. В языке Си нет процедур, подпрограмм, есть только функции. Каждая функция должна иметь имя, которое используется для вызова функции. Имя одной из функций – main(), зарезервировано. Эта функция должна присутствовать в каждой программе, с нее начинается выполнение программы, хотя она необязательно должна следовать первой в программе.

При вызове функции ей могут быть переданы данные посредством аргументов функции. Функция может возвращать и не возвращать значение. Возвращаемое значение и есть основной результат выполнения функции, который при выполнении программы подставляется на место вызова функции.

С использованием функций в языке Си связаны три понятия:

1. Объявление функции.
2. Определение функции.
3. Вызов функции.

**Объявление функции (прототип)** задает имя функции, типы и число формальных параметров, тип значения, возвращаемого функцией, и класс памяти.

Объявление функции, в отличие от определения функции, является оператором, то есть за заголовком функции следует точка с запятой. Заголовок функции может заканчиваться списком имен. Это имена других функций, возвращаемых значение того же типа, который задан спецификатором типа для первого описателя функции. Компилятор использует прототип функции для сравнения типов фактических параметров в вызове функции с типами формальных параметров. Прототипы библиотечных функций находятся в файлах включения, поставляемых в составе системы программирования и включаемых в программу с помощью директивы препроцессора #include.

**Определение функции** задает имя функции, типы и число формальных параметров, тип значения, возвращаемого функцией, объявления и операторы, которые определяют действие функции (тело функции).

Формат определения функции:

[спецификатор класса памяти][спецификатор типа] имя([список формальных параметров])

тело функции

Необязательный спецификатор класса памяти задает класс памяти, который может быть или static или extern. Если спецификатор не задан, то по умолчанию extern.

Спецификатор типа может задавать любой основной тип. Если тип не задан, то по умолчанию принимается int.

Имя функции может быть задано со звездочкой, что означает, что функция возвращает указатель. Функция не может возвращать массив или функцию, но может возвращать указатели на любой тип, включая массивы и функции. Функция возвращает значение, если ее выполнение заканчивается выполнением оператора return в теле функции следующего формата:

return выражение;

Выражение вычисляется, преобразуется, если необходимо, к типу возвращаемого значения в определении (объявлении) функции, и управление передается в точку вызова.

Если оператор return не задан или не содержит выражения, то функция не возвращает никакого значения. В этом случае спецификатор типа возвращаемого значения должен быть задан ключевым словом void, означающим отсутствие возвращаемого значения.

Формальные параметры – это переменные, которые принимают значения, передаваемые функции во время вызова.

Список формальных параметров – это последовательность объявлений формальных параметров, разделенных запятыми. Список может заканчиваться запятой и многоточием. Это означает, что число аргументов функции переменно. Если формальный параметр представлен в списке, но не объявлен, то предполагается, что параметр имеет тип int.

Все переменные, объявленные в теле функции, имеют класс памяти auto, если они не объявлены иначе, и являются локальными.

Параметры функции передаются по значению и могут рассматриваться как локальные переменные. Они инициализируются значениями переданных аргументов и теряются при выходе из функции. Поэтому в теле функции нельзя изменить значение параметра, так как функция работает с копиями аргументов. Однако, если в качестве параметра передать указатель переменной, можно использовать операцию разадресации этой переменной для осуществления доступа и изменения ее значения.

Вызов функции имеет следующий формат:

выражение([список выражений])

Выражение вычисляется как адрес функции. Список выражений представляет собой список фактических аргументов, передаваемых в функцию. Фактический аргумент может быть величиной основного типа, структурой, перечислением, смесью или указателем. Массивы и функции не могут быть переданы как параметры, но указатели на эти объекты могут быть переданы.

Выполнение вызова функции происходит следующим образом:

1. Вычисляются значения в списке выражений. Если в прототипе функции вместо списка формальных параметров задано ключевое слово void, это значит, что в функцию не передается никаких аргументов и в определении функции не должно быть формальных параметров.
2. Происходит замена формальных параметров на фактические (последовательно).
3. Управление передается на первый оператор функции.
4. Выполнение оператора return в теле функции возвращает управление и, возможно, значение в вызывающую функцию. Если оператор return не задан, то управление возвращается после выполнения последнего оператора тела функции. При этом возвращаемое значение не определено.

Выражение перед скобками вычисляется как адрес функции. Это означает, что функцию можно вызвать через указатель на функцию.

**Пример:**

#include <stdio.h>

#include <stdlib.h>

int(\*fun)(int, int), sum(int, int), razn(int, int);

int main()

{

int a = 10, b = 20, c;

c = sum(a, b);

printf("c=%d\tsum=%d\trazn=%d\n", c, sum(a, b), razn(a, b));

fun = sum;

printf("\*fun=%d\n", (\*fun)(a, b));

fun = razn;

printf("\*fun=%d\n", (\*fun)(a, b));

system("pause");

return 0;

}

int sum(int x, int y)

{

return (x + y);

}

int razn(int x, int y)

{

return (y - x);

}

**Пример:/\* передача аргументов по значению и по адресу\*/**

#include <stdio.h>

#include <stdlib.h>

void swap(int x, int y);

void swap1(int \*x, int \*y);

int main()

{

int a = 10, b = 20;

swap(a, b);

printf("a=%d\tb=%d\n", a, b);

swap1(&a, &b);

printf("a=%d\tb=%d\n", a, b);

system("pause");

return 0;

}

void swap(int x, int y)

{

int temp = x;

x = y;

y = temp;

printf("x=%d\ty=%d\n", x, y);

}

void swap1(int \*x, int \*y)

{

int temp = \*x;

\*x = \*y;

\*y = temp;

printf("x=%d\ty=%d\n", \*x, \*y);

}

Для передачи в функцию в качестве фактического параметра массива достаточно указать адрес его начала.

**Пример:**

#include <stdio.h>

#include <stdlib.h>

#define n 5

float Sum(float[]);

int main()

{

float aVar[] = { 1.5,4.5,5.0,10.0,15.5 };

float rez;

rez = Sum(aVar) / (float)n;

printf("Sum=%f\n", rez);

system("pause");

return 0;

}

float Sum(float aFVar[])

{

float f = 0;

for (int i = 0; i < n; i++)

f += aFVar[i];

return f;

}

**Замечание**: Компилятору не требуется размер массива, подлежащего обработки.

**Замечание**: В функции можно передавать многомерные массивы.

**Допустимые варианты:**

float Det(float b[][n]);

det\_a=Det(d); // вычисление определителя матрицы системы

float Det(float c[][n])

float Det(float b[n][n]);

det\_a=Det(d); // вычисление определителя матрицы системы

float Det(float c[n][n])

/\* Передача в функцию массива структур\*/

#include <stdio.h>

#include <stdlib.h>

#define n 2

struct st {

int a;

int b;

int c;

};

float middle(struct st[]);

int main()

{

struct st s1[n];

s1[0].a = 5;

s1[0].b = 4;

s1[0].c = 3;

s1[1].a = 5;

s1[1].b = 4;

s1[1].c = 5;

float bn = middle(s1);

printf("sr=%f\n", bn);

system("pause");

return 0;

}

float middle(struct st d[])

{

float sr = (d[0].a + d[0].b + d[0].c + d[1].a + d[1].b + d[1].c) / 6.0;

return sr;

}

***Практическое занятие***

1. Написать программу решения СЛАУ методом Крамера.
2. Написать программу нахождения обратной матрицы (с проверкой).