

Advance c language

Unit-1

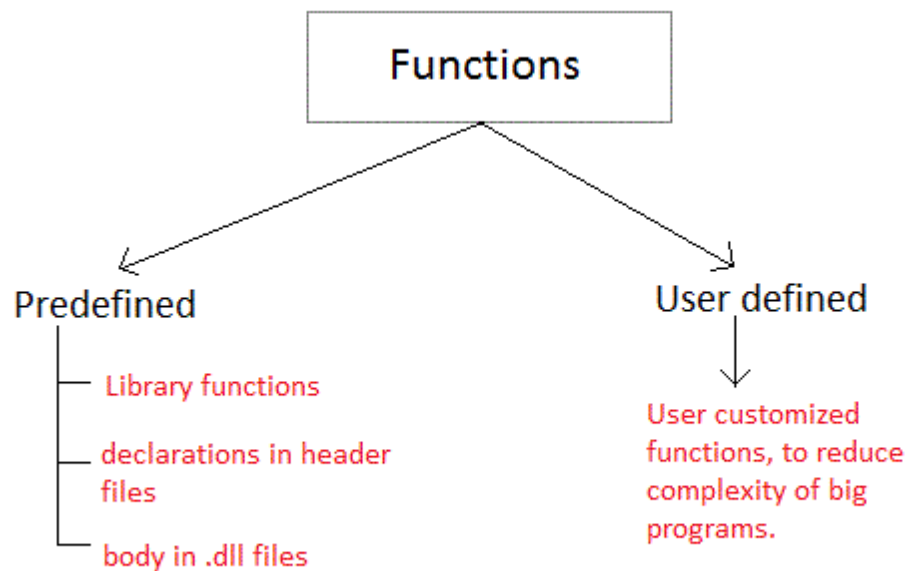
Function

A **function** is a block of code that performs a particular task.

These functions defined by the user are also known as **User-defined Functions**

C functions can be classified into two categories,

1. **Library functions**
2. **User-defined functions**



Library functions are those functions which are already defined in C library, example `printf()`, `scanf()`, `strcat()` etc.

.

Predefined functions are built-in functions that perform standard operations and that generally do not depend on any classes.

A predefined void function is simply a predefined function that has no return value.

These are present in any of the various headers that can be included in a program. For example, the function

`double sqrt(double)`

present in `<math.h>` that computes the square root of the argument passed to it.

suppose, you need to create a circle and color it depending upon the radius and color. You can create two functions to solve this problem:

- `createCircle()` function
- `color()` function

Example: User-defined function

Here is an example to add two integers.

To perform this task, we have created an user-defined `addNumbers()`.

```
1. #include <stdio.h>
2. int addNumbers(int a, int b);
3.
4. int main()
5. {
6.     int n1,n2,sum;
7.
8.     printf("Enters two numbers: ");
9.     scanf("%d %d",&n1,&n2);
10.
11.     sum = addNumbers(n1, n2);
12.     printf("sum = %d",sum);
13.
14.     return 0;
15. }
16.
17. int addNumbers(int a, int b)    // function definition
18. {
19.     int result;
20.     result = a+b;
21.     return result;    // return statement
22. }
```

Function prototype

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type.

It doesn't contain function body.

Syntax of function prototype

```
returnType functionName(type1 argument1, type2 argument2, ...);
```

In the above example, `int addNumbers(int a, int b);` is the function prototype which provides the following information to the compiler:

1. name of the function is `addNumbers()`
2. return type of the function is `int`
3. two arguments of type `int` are passed to the function

The function prototype is not needed if the user-defined function is defined before the `main()` function.

Calling a function

Control of the program is transferred to the user-defined function by calling it.

Syntax of function call

```
functionName(argument1, argument2, ...);
```

In the above example, the function call is made using `addNumbers(n1, n2);` statement inside the `main()` function.

Function definition

Function definition contains the block of code to perform a specific task.

In our example, adding two numbers and returning it.

Syntax of function definition

```
returnType functionName(type1 argument1, type2 argument2, ...)
```

```
{  
  
    //body of the function  
  
}
```

When a function is called, the control of the program is transferred to the function definition.

And, the compiler starts executing the codes inside the body of a function.

Passing arguments to a function

In programming, argument refers to the variable passed to the function.

In the above example, two variables `n1` and `n2` are passed during the function call.

The parameters `a` and `b` accepts the passed arguments in the function definition.
These arguments are called formal parameters of the function.

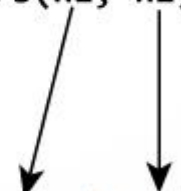
How to pass arguments to a function?

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    ... ..
}
```

A diagram with two arrows. One arrow starts from the variable 'n1' in the function call 'addNumbers(n1, n2)' inside the 'main' function and points down to the parameter 'a' in the function definition 'int addNumbers(int a, int b)'. The other arrow starts from the variable 'n2' in the same function call and points down to the parameter 'b' in the function definition.

The type of arguments passed to a function and the formal parameters must match, otherwise, the compiler will throw an error.

If `n1` is of char type, `a` also should be of char type. If `n2` is of float type, variable `b` also should be of float type.

A function can also be called without passing an argument.

Return Statement

The return statement terminates the execution of a function and returns a value to the calling function.

The program control is transferred to the calling function after the return statement.

In the above example, the value of the `result` variable is returned to the main function. The `sum` variable in the `main()` function is assigned this value.

Return statement of a Function

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    return result;
}
```

The diagram illustrates the flow of data. An arrow originates from the `return result;` statement in the `addNumbers` function and points to the `addNumbers(n1, n2);` call in the `main` function. Another arrow points from the `addNumbers(n1, n2);` call to the `sum =` part of the assignment statement `sum = addNumbers(n1, n2);`. A callout box labeled `sum = result` indicates the value being passed back to the caller.

Syntax of return statement

```
return (expression);
```

For example,

```
return a;  
  
return (a+b);
```

The type of value returned from the function and the return type specified in the function prototype and function definition must match.

A **User-defined functions** are those functions which are defined by the user at the time of writing program.

These functions are made for code reusability and for saving time and space.

Benefits of Using Functions

1. It provides modularity to your program's structure.
 2. It makes your code reusable. You just have to call the function by its name to use it, wherever required.
 3. In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.
 4. It makes the program more readable and easy to understand.
-

Function Declaration

General syntax for function declaration is,

```
Return type function name(argument list);  
Int sum(int a,int b);
```

Like any variable or an array, a function must also be declared before its used.

Advance c Language

Function declaration informs the compiler about the function name, parameters is accept, and its return type.

The actual body of the function can be defined separately.

It's also called as **Function Prototyping**. Function declaration consists of 4 parts.

- returntype
- function name
- parameter list
- terminating semicolon

returntype

When a function is declared to perform some sort of calculation or any operation and is expected to provide with some result at the end, in such cases, a return statement is added at the end of function body.

Return type specifies the type of value(int, float, char, double) that function is expected to return to the program which called the function.

functionName

Function name is an identifier and it specifies the name of the function.

The function name is any valid C identifier and therefore must follow the same naming rules like other variables in C language.

parameter list

The parameter list declares the type and number of arguments that the function expects when it is called.

Also, the parameters in the parameter list receives the argument values when the function is called.

They are often referred as **formal parameters**.

Category of Functions

Function with no arguments and no Return Value

Function with arguments and no Return Value

Function with no arguments and Return Value

Function with arguments and Return Value

Function with no arguments and no Return Value In C

In the chapter Types of Function Calling we have seen different ways of calling function. Here is more elaborated example of Function taking no argument and not returning any value.

Live Example :

```
#include<stdio.h>

void area(); // Prototype Declaration
void main()
{
    area();
}

void area()
{
    float area_circle;
    float rad;
```

Advance c Language

```
printf("\nEnter the radius : ");
scanf("%f",&rad);

area_circle = 3.14 * rad * rad ;

printf("Area of Circle = %f",area_circle);
}
```

Output :

```
Enter the radius : 3
Area of Circle = 28.260000
```

1.).In this category, the function has no arguments.

It does not receive any data from the calling function. Similarly, it doesn't return any value.

The calling function doesn't receive any data from the called function.

Function with arguments and no Return Value :

Note :

- Function accepts argument but it does not return a value back to the calling Program .
- It is Single (One-way) Type Communication
- Generally Output is printed in the Called function
- Here **area** is called function and **main** is calling function.

Program :

```
#include<stdio.h>
#include<conio.h>

//-----
void area(float rad); // Prototype Declaration
//-----

void main()
{
```

```
float rad;
printf("\nEnter the radius : ");
scanf("%f",&rad);
area(rad);
getch();
}
//-----
void area(float rad)
{
float ar;
ar = 3.14 * rad * rad ;
printf("Area of Circle = %f",ar);
}
```

Output :

```
Enter the radius : 3
Area of Circle = 28.260000
```

2.)In this category, function has some arguments .

it receives data from the calling function, but it doesn't return a value to the calling function.

The calling function doesn't receive any data from the called function.

So, it is one way data communication between called and calling functions.

3) Function with arguments and Return Value

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
Int sum(int x,int y);
```

```
Void main()
```

Advance c Language

```
{  
  
    Int a,b,c;  
  
    Clrscr();  
  
    Printf("\nenter a=");  
  
    Scanf("%d",&a);  
  
    Printf("\nenter b=");  
  
    Scanf("%d",&b);  
  
    C=sum(a,b);  
  
    Printf("ans is=%d",c);  
  
    Getch();  
  
}
```

```
Int sum(int x,int y)
```

```
{  
  
    Int z;  
  
    Z=x+y;  
  
    Return(z);  
  
}
```

Recursive Functions

Those functions which are called by themselves are called Recursive functions.

It means that the same function is called again within itself.

The function itself becomes the calling function of it.

Control is cycled within the function until a break point is reached in the program.

1. A function declared globally i.e, outside the main() function can be used by any other function in the program.
2. Any C program contains atleast one function, which is nothing but the main().
3. There is no limit on the number of functions that might be present in a C program.
4. Each function in a program is called in the sequence specified by the function calls in main().
5. A function can be called from another function, but a function cannot be defined in another function.
6. Variables declared in the main() function and called functions are different. So, same Same variable names can be used in main() and called functions.

Write a "C" Program to find Factorial of given no using Recursion.

```
#include<stdio.h>

#include<conio.h>

int factorial(int no)
{
    int fact;

    if(no==1)
```

```
        return(1);
    else
        fact=no*factorial(no-1)
    return(fact);
}

void main()
{
    int no,f;
    clrscr();
    printf("\n enter no");
    scanf("%d",&no);
    f=factorial(no);
    printf("\n the fact is=%d",f);
    getch();
}
```

Output:-

Enter no 5

the fact is 120

STORAGE CLASSES

Variables in C language are not only data types but are also of storage class.

They provide information about locality and visibility of variables.

Depending upon the declaration of variables they are of two types.

1. Global variables:

These are the variables which are declared before the main function and they can be used in any function with in the program.

2. Local variables: These are the variables which are declared with in the main function and they can be used with in that function only.

EX:

```
int d;  
void main()  
{  
    char c,e;  
}
```

In the above example variable d is globally declared and can be used in any function where as c and e are declared locally and can be used with in that function only.

Storage classes

In other words, not only do all variables have a data type, they also have a 'storage class'.

1. Automatic Storage class.
2. Register Storage class.
3. Static Storage class.
4. External Storage class.

There are four storage classes in C.

auto

extern

static

register

auto

this variable is also called local or internal variable

Variables which are defined within a function or a block (block is a section of code which is grouped together.

eg.statements written within curly braces constitute a block of code) by default belong to the auto storage class.

These variables are also called **local variables** because these are local to the function and are **by default assigned some garbage value**.

Since these variables are declared inside a function, therefore these can only be accessed inside that function.

There is no need to put 'auto' while declaring these variables because these are by default auto.

```
#include<stdio.h>
```

```
Void fun();
```

```
Void main()
```

```
{
```

```
    Int i=5;
```

```
    Fun();
```

```
    Printf("%d",i);
```

```
}
```

```
Void fun()
```

```
{
```

```
    Int i=10;
```

```
    Printf("\n%d",i)
```

```
}
```

extern

this variable is a global variable declared in some other program file.

Advance c Language

A global variable is a variable which is declared outside of all the functions.

it can be accessed throughout the program and we can change its value anytime within any function as follows.

```
#include<stdio.h>
```

```
Void exch();
```

```
Int x,y;
```

```
Void main()
```

```
{
```

```
Printf("enter two no=");
```

```
Scanf("%d%d",&x,&y);
```

```
Exch();
```

```
Printf("after exchange");
```

```
Printf("\n%d %d",x,y);
```

```
}
```

```
Void exch()
```

```
{
```

```
Int temp;
```

```
Temp=x;
```

```
X=y;
```

```
Y=temp;
```

```
}
```

static

A variable declared as static once initialized, exists till the end of the program.

If a static variable is declared inside a function, it remains into existence till the end of the program and not get destroyed as the function exists (as in auto).

If a static variable is declared outside all the functions in a program, it can be used only in the program in which it is declared and is not visible to other program files(as in extern).

Let's see an example of a static variable.

```
#include<stdio.h>
```

```
Void fun();
```

```
Void main()
```

```
{
```

```
Int i;
```

```
For(i=0;i<5;i++)
```

```
Fun();
```

```
}
```

```
Void fun()
```

```
{
```

```
Static int count=0;
```

```
Printf("exacution no=%d",count);
```

```
}
```

register

It tells the compiler that the **variable will get stored in a register instead of memory (RAM)**.

We can access a register variable faster than a normal variable. Not all the registers defined as **register** will get stored in a register since it depends on various restrictions of implementation and hardware.

We cannot access the address of such variables since these do not have a memory location as which becomes clear by the following example.

```
#include <stdio.h>

int main()
{
    register int n = 20;

    int *ptr;

    ptr = &n;

    printf("address of n : %u", ptr);

    return 0;
}
```

Nested function

nested function (or **nested procedure** or **subroutine**) is a function which is defined within another function, the *enclosing function*.

```
#include<stdio.h>
#include<conio.h>
```

```
Int greater(int ,int);
Void disp(void);
Main()
{
    Clrscr();
    Disp();
    Getch();

}
Int greater(int x,int y)
{
    If(x>y)
    {
        Return x;
    }
Else
{
    Return y;
}
}
Void disp(void)
{
    Int a,b,ans;
    Printf("enter two nos=");
    Scanf("%d%d",&a&b);
    ans=greater(a,b);
    Printf("greater values is  =%d",ans);
}
```