



B.C.A. Semester – 2

UNIT - 1

BCA -201

Advance Programming Language ‘C’

User defined function

* Function :-

Independent coded programs are called subprograms that are much easier to understand debug and testing.

In C such a subprograms are referred to as function.

Functions can be classified into two categories.

1) library function

2) user defined function

printf and scanf functions belong to the library functions.

main() is example of user defined function.

The main difference between library function and user defined function is that library functions are not required to be written by us.

Whereas user defined function has to be developed by the user at the time of writing a program.

* Need for user defined function or Advantages of user defined function

- - - - -
→ The program may become too large and complex and as a result the task for debugging testing and maintaining becomes difficult if program is divided into functional parts then each part may be independently coded and later combine into a single unit.

→ It facilitates top-down modular programming in this programming style the high level of over all problem is solved first while the details of each lower level function are addressed later.

length →
etc →
multiple time
is function →
use in →
elements of user defined function
The length of a source program can be reduced by using the function at appropriate places.

It is easy to locate and isolate a function for investigation.

=) elements of user defined function

Three elements of user defined function

- 1) Function definition
- 2) Function call
- 3) Function declaration

1) Function definition :-

function definition is an independent program module that is specially written to implement the requirement of the function.

→ A function definition is also known as function implementation that includes following elements.

- | | | |
|-------------------------------|---|-----------------|
| 1) Function name | } | function header |
| 2) Function Type | | |
| 3) List of Parameters | } | function body |
| 4) Local variable declaration | | |
| 5) Function Statement | | |
| 6) A return statement | | |

Syntax Function definition

function-type Function Name (Parameter list)

{

 local variable declaration ;
 executable statement 1 ;
 executable statement 2 ;

 Return statement ;

}

-> Function type function name (argument list) is known as the function header and the statements within the opening and closing braces {} is known as function body which is a compound statement.

-> Function header :-

The function header consists of three parts.

- > The function type (It is also known as return type)
- > Function name and the
- > Formal parameter list

NOTE :-

Semicolon is not used at the end of function header.

=> function name & type :-

Function type specifies

The types of value like integer, float, double.
If the return type is not specified C will be assume the integer.

If the function is not returning anything then we need to specify return type as void.

The function name is any valid C identifier the name should be appropriate to the task performed by the function.

* Formal parameter list :-

The parameter list declares the variables that will receive the data sent by calling program.

Ex.

int sum(int a, int b)

float multiplication(float x, float y)

* function body :-

Function body contains 3 parts

- 1) local variable declaration that specify the variables needed by the function.
- 2) function statement that perform the task to the function.
- 3) return statement that returns the value evaluated by the function.

Ex.

① int sum(int a, int b)

{

 int c;

 c = a+b;

 return (c);

}

② void sum(int a, int b)

{

 printf("sum = %d", a+b);

 return;

3

2) void main()
{

Struct book b1;

printf("Enter value of books title=");

scanf("%s ", b1.title);

printf("Enter name of book author=");

scanf("%s ", b1.author);

printf("Enter the Pages of book=");

scanf("%d", &b1.pages);

printf("Enter the Price of book=");

scanf("%d", &b1.price);

printf("Book title = %s", b1.title);

printf("Author name = %s", b1.author);

printf("Book Pages=%d", b1.page);

printf("Book Price: %f", b1.price);

getch();

3

Return value and their types

A function may or may not send back any value to the calling function.

If it does it is done to the return statement while it is possible to pass to the called function any number of values.

Syntax of return

return (expression);

Ex → return (c);

2) Function call :-

A function can be called by simply using the function name followed by a list of parameters or arguments.

Ex.

main()

2

```
int c;  
c = sum(15, 10);  
printf("%d", c);
```

3

```
int sum(int a, int b)
```

4

```
int c;  
c = a+b;  
return (c);
```

- > Here when the compiler encounter function call the control is transfer to the function sum.
- > The function is executed line by line as described and value is return when return statements is executed.
- 3) function declaration :-
function declaration is also known as function prototype.
It consists 4 parts.
 - 1) function type (Return type)
 - 2) function name
 - 3) Parameter list
 - 4) Terminating semicolon

Syntax

function-type function-name (Parameter list)

Ex

int sum (int a, int b);

* Definition of calling function :-

function defined main program it is known as called calling function. it is userdefined function which is defined outside the main program it is called called function.

=) Actual Argument :-

A variable which are declared in calling function is known as actual argument.

=) Formal Argument :-

A variable which are declared in called function or user define function is known as formal argument.

* Category of User defined function

There are 5 categories of udf.

- 1) Function with no arguments and no return values.
- 2) Function with Arguments and no return value.
- 3) Function with Arguments and one return values.
any companion

- 4) function with no arguments but return value.
- 5) function that return multiple values.

- 1) function with no arguments no return value :-

when a function has no argument it does not receive any data from the calling function.

Ex

```
#include<stdio.h>
#include<conio.h>
void sum(void); // function declaration
void main()
{
    sum(); // calling function
    getch();
}
void sum(void) // called function
{
    int a=10, b=5, c=0;
    c=a+b;
    printf("%d", c);
}
```

2) Function with Arguments and no return value:-

The calling function can send the data to the called function but it can not return any value to the calling function as it has no return statement.

Ex:-

```
#include<stdio.h>
#include<conio.h>
void sum(int a, int b);
void main()
{
```

```
    int a=10, b=20;
    sum(a,b);
    getch();
```

3

```
void sum(int a, int b)
{
```

```
    int c=0;
    c=a+b;
    printf("%d",c);
```

3

3) Function with Argument and one return value.

```
#include<stdio.h>
#include<conio.h>
int sum(int a, int b);
void main()
{
```

2
sum(a,b);
printf("y.d",c);
getch();

3
int sum(int a, int b)

{
 a=10, b=5, c=;
 c=a+b;
 Return(c);

4) Function with no argument with return value

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

int sum(void)

void main();

{
 sum();
 printf("y.d",c);
 getch();

3
int sum(void)

{
 int a=10, b=5, c=;
 c=a+b;
 Return(c);

3

② functions with no arguments and no result:-

return value

when the function doesn't have any arguments and doesn't have any return value then it will call a function with no argument and no return value.

Ex: void add(); for linking with rest

```
2922100; Itemmain() { int i=0; i<10; i++ {  
    std::cout << "Hello world " << i; } }  
  
void calculate() {  
    add();  
    calculate();  
}  
  
int main() {  
    calculate();  
    return 0;  
}
```

```

int a, b;           // int a=10, b=5, c;
scanf("x%d\n%d", &a, &b); // printf("x%d", c);
printf("add=%d", a+b); // c=a+b;
}

```

In this kind of user define function the main() does not take value and does not passes the value to the UDF as argument and the function does not return any value to main() that's called function with no argument and no return value.

2) Function with arguments and no return value:-

The calling function can send the data to the called function but it can not return any value to the calling function as it has no return statement.

Ex:-

```
#include<stdio.h>
#include<conio.h>
void sum(int a, int b);
Void main()
{
    int a=10, b=20;
    sum(a,b);
    getch();
}
```

3
Void sum (int a, int b)

```
{ 
    int c=0;
    c=a+b;
    printf("%d",c);
}
```

3

③ function iC with argument and return value

Ex.

```
int add (int, int);
```

main()

int main() { global var to bind with at

2nd line below int a, b; 2nd line C int

2d add(a, b); // if scanf("x,y,z", &a, &b) ;

printf("x+y=%d", add(a, b)); // if printf("x+y=%d", a+b);

2nd line C int printf("x+y=%d", a+b); // if printf("x+y=%d", a+b);

2nd line C int add (int a, int b); // if printf("x+y=%d", a+b);

a=10, b=5, c=0;

int n;

c=a+b; // if n=h+j; // after comment

return(c); // if return(n);

} // function block end

Condition

when the function does have any argument
and does have any return value it's
called function "with argument and return
value."

The main() take value and passes
the value to the UDF as argument
and the called function does return
any value to the main().

* Nesting of Functions

nesting of function means
 main function can call function 1 which
 calls function 2 which calls function
 3 and so on...

- There is in principle no limit as to
 how deeply functions can be nested.

Ex

```
float ratio(int x, int y, int z);
int difference(int x, int y);
void main()
```

{

```
int a, b, c;
scanf("%d %d %d", &a, &b, &c);
printf("y.f", ratio(a, b, c));
```

y

```
float ratio(int x, int y, int z)
```

{

```
if (difference(y, z))
    return (x / (y - z));
```

y

else

{

```
return (0.0);
```

}

y

```
int difference(int p, int q)
```

{

```
int (p, j = 9)
```

return(1);
else
 return(0);

y

* Passing the array to the function over

How can we pass array as argument
function.

- like the value of simple variable it is also possible to pass the values of an array to a function.

Previously we pass the variable in function similarly we can also pass the whole array as a actual parameter through the function. But the formal parameter should be declared as array of same datatype.

```
=) #include<stdio.h>
#include<conio.h>
float largest(float a[], int n);
void main()
{
    float value[4] = {2.5, 4.5, 7.5, 8.5};
    printf("%.f", largest(value, 4));
}

float largest(float a[], int n)
{
    int i;
    float max;
    for(i=1; i < n; i++)
    {
        if(max < a[i])
            max = a[i];
    }
    return (max);
}
```

```
printf("enter value of *b = ");
scanf("%d", &b);
swap(*a, *b);
getch();
```

3

```
void swap(int *a, int *b)
```

2

```
int temp;
temp = *a;
*a = *b;
*b = temp;
```

```
printf("value of *a & *b = %d %d", *a, *b);
```

3

*
Q

The scope visibility and lifetime of variable.

In a c not only do we have datatype but they also have a storage classes.

There are 4 types of storage classes.

- 1) Automic variable
- 2) External variable
- 3) static variable
- 4) Register variable

my companion

1) Automic variable :-

Automic variable is also known as local or internal variable that is called declared inside the function. They are created when the function is created and the function exists.

2) External variable:-

External variable is also known as global variable that is declared inside and outside of the function. This variable are both alive and active throughout out the entire program.

3) static variable :-

The value of static variable persist until the end of program. A variable can be declared static using the keyword static.

Ex static int x;

A static variable may be either internal or an external type depending on the place of declaration.

4) Register variable :-

We can tell the compiler that a variable should be kept in on

of the machine register instead
of keeping in memory.

since Register class is
much faster than memory access
keeping the frequently access variable
register will lead to faster
execution.