
Parul University

# Fundamentals of Programming Using C - 15101104

# Unit 4 : User Defined Function

**Bhavika Vaghela**

**Assistant Professor**

**Parul Institute of Computer Application**

**Faculty of IT & Computer Science**

**Parul University**

# Function in C

What is function?

- Function is group of executable statement to perform some task.
- A functions takes some data as input, perform some operation on that data and then may return a value.
- Any C program must contain at least one function, which is main().
- There is no limit on the number of functions that might be present in a C program.
- Also known as subprograms which are used to compute a value or perform a specific task.
- They can't run independently and are always called by the main() program or by some other function.

# Function in C

- The function contains the set of programming statements enclosed by {}.
- A function can be called multiple times to provide reusability and modularity to the C program.
- In other words, we can say that the collection of functions creates a program. The function is also known as procedure or subroutine in other programming languages.

# Advantages of Function (why to use it?)

- Code reusability : we can use the function again and again once it is declared.
- It called n times any where in c program.
- One can divide a big program into sub program using function.
- Debugging of the code would be easier if you use functions, as errors are easy to be traced.
- Reduces the size of the code, duplicate set of statements are replaced by function calls.

# Types of function

**C support two types of functions**

1) Inbuilt function
2) User Defined function

**Library functions**: These are also known as Pre defined functions

Examples are scanf(), printf(), getch(), strlen(), strcmp(), strcat(), sqrt(), pow()

**User-Defined functions**: User defined functions are self-contained blocks of statements which are written by the user to compute or perform a task

- They can be called by the main program repeatedly as per the requirement.

# Element of User Defined Function

| C function aspects | Syntax |
|---|---|
| Function declaration | return_type function_name (argument list); |
| Function call | function_name (argument_list) |
| Function definition | return_type function_name (argument list) {function body;} |

# Function Prototype / Declaration

- All Identifiers in C must be declared before they are used. This is true for functions as well as variables.
- For functions, the declarations needs to be done before the first call of the function.
- A function declaration specifies the name, return type, and arguments of a function.
- Having the prototype available before the first use of the function allows the compiler to check that the correct number and types of arguments are used in the function call.
- A return type indicating the variable that the function will be return. It may be **int, float, double, char, short, void** etc.

**Syntax : <Return type> <function name> (parameter/argument list);**

# Function Implementation or Defination

- It is the actual function that contains the code that will be executed.
- Should be identical (same) to the function prototype.
- **Syntax : return-type function_name ( arg-type name-1,...,arg-type name-n)**

```
return-type function_name ( arg-type name-1,...,arg-type name-n)
{
declarations;
statements;
return(expression);
}
```

# Cont…

- General form of any function definition is:

```
return-type function-name(argument declarations)
{
    declarations and statements
}
```

- Return-type refers to the data type of the value being returned from the function. If the return type is omitted, int is assumed.
- The values provided to a function for processing are the arguments.
- The set of statements between the braces is called as the function body.

# Simple Example to Understand UDF

```
#include<stdio.h>
#include<conio.h>
void myfun();      //prototype of function
void hello_fun();  //prototype of function
void main()
{
        clrscr();
        printf("\n You are in main function");
        myfun();       //calling of function
        hello_fun();    //calling of function
        getch();
}
//Defination of User defined function
void myfun()
{
        printf("\n You are user defined function");          //Function Body
        hello_fun(); //can also call function from here
}
void hello_fun()
{
        printf("\n You are in hello function");              //Function Body
}
```

# Different aspects of function calling

- Without Argument with return value
- Without Argument without return value (void function)
- With Argument with return value
- With Argument without return value (void function)

**What is argument or parameter in function?**

Value which is pass in function as parameter or argument to perform some task. It of any data type like int, float, double, char etc.

****note** if function return any value that mean return type of function is any datatype like int, float, char, double, long int etc…

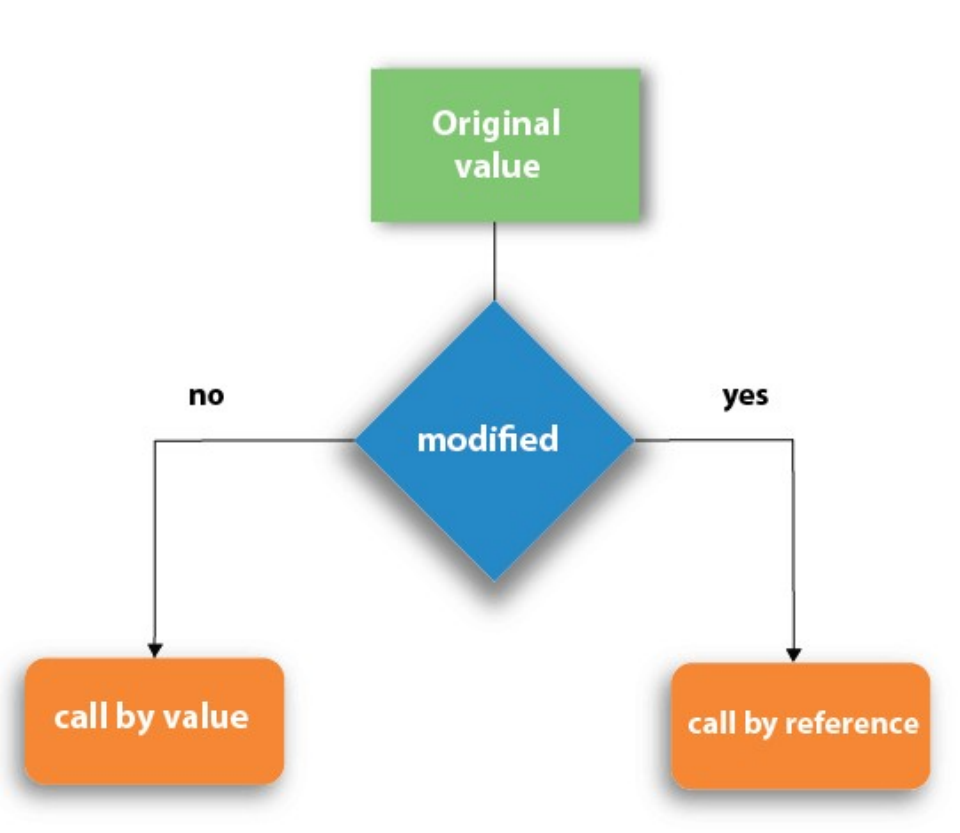
****note** : if function not return any thing mean return type of that function is void.

****Argument** is not mandatory to pass but where it is required try to pass it.

# Function calling

There are two methods to pass the data into the function in C language, i.e., call by value and call by reference.

Will be discuss program later on in unit 5.



# Recursion Function

- Recursion Function means function call itself within the same function. It called recursive function, and such function calls are called recursive calls.
- This concept is not applicable for every problem.
- Recursion code is shorter than iterative code however it is difficult to understand.
- Some argument values cause the recursive method to return without calling itself. This is the base case.
- Either omitting the base case or writing the recursion step incorrectly will cause infinite recursion (stack overflow error)

Write a program to find addition of natural numbers using concept of recursion function.

# Recursion function to find factorial of number

```
//proram to understand concept of recurssion call
#include<stdio.h>
#include<conio.h>
int fact_num(int n);
void main()
{
   int n;
   clrscr();
   printf("Enter a positive integer: ");
   scanf("%d",&n);
   printf("Factorial is %d ",fact_num(n));
   getch();
}

int fact_num(int n)
{
   if (n>=1)
       return n*fact_num(n-1);
   else
      return 1;
}
```

# Passing array as an argument to function

- Like a normal variable one can also pass 1D, 2D or Nd array in user defined function.
- One can pass array using call by value or call by reference. (will discuss call by reference in unit 5)
- One can also pass single single elements of an array into user define function.

**Why to pass array in UDF?**

When some one wants to pass n numbers of arguments of same data type to perform operation array is best option.

# Program to do addition of element by passing array

Program 1 : By passing 1d array element index by index.

Program 2 : By passing whole(entire) 1D array.

Program 3 : By Passing 2D array to perform addition of elements.

Program 4 : By passing 2D array perform program for transpose matrix using UDF.

Program 5 : Perform matrix multiplication by passing array in function.

Program 6 : Perform matrix addition by passing array in function.

**likewise you can do any program by passing array as an argument to function.

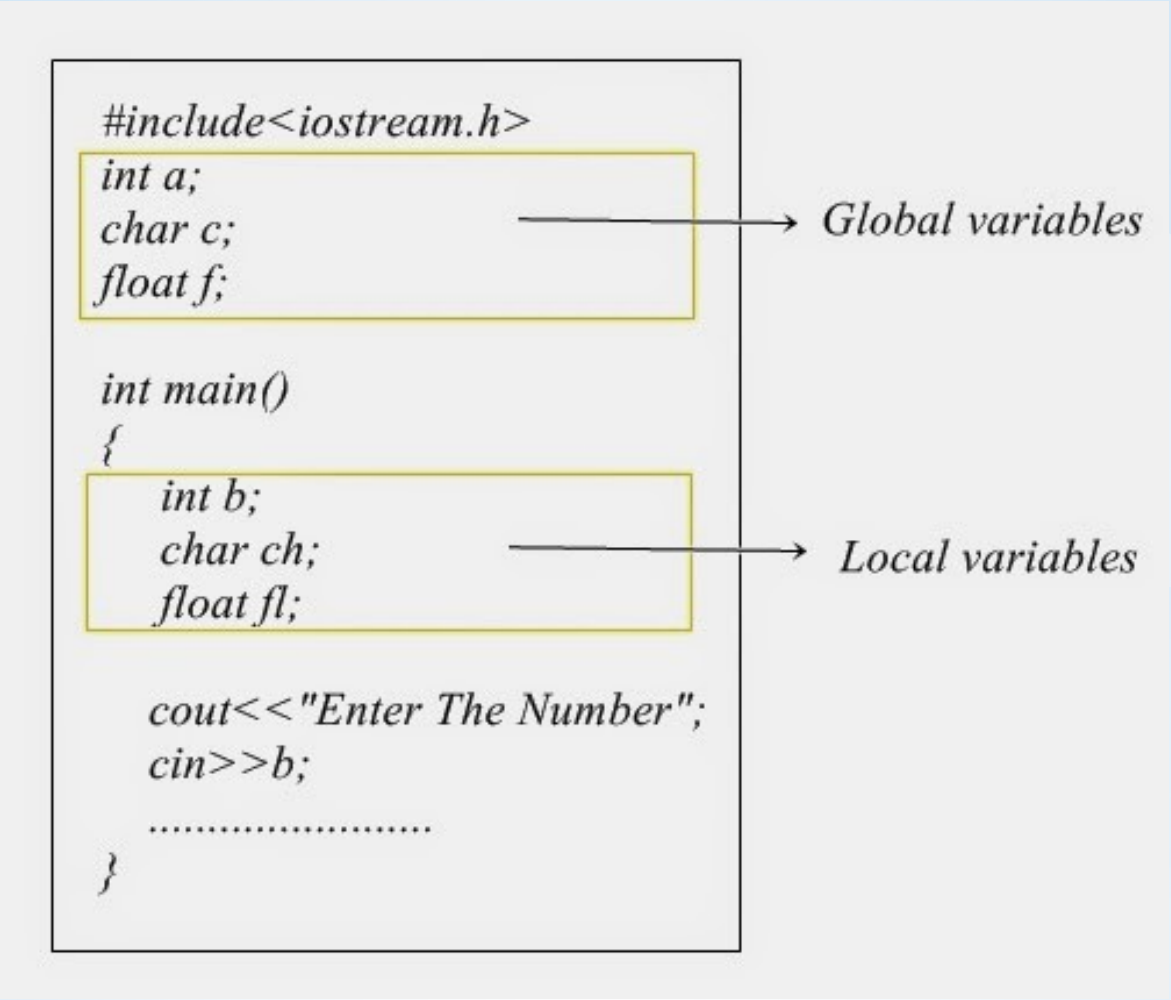
# Scope, Visibility and lifetime of variable

- **A scope** in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed.
- **Visibility** : The Program's ability to access a variable from the memory.
- **LifeTime** : The lifetime of a variable is the duration of time in which a variable exists in the memory during execution.

**There are three places where variables can be declared in C programming language –**

1. Inside a function or a block which is called local variables.
2. Outside of all functions which is called global variables.
3. In the definition of function parameters which are called formal parameters.

# Cont…


#include<iostream.h>
int a;
char c;
float f;
Global variables
int main()
{
int b;
char ch;
float fl;
Local variables
cout<<"Enter The Number";
cin>>b;
.....
}

# lifetime of variable

- The lifetime of a variable is the period of time in which the variable is allocated a space (i.e., the period of time for which it "lives").

**There are four lifetimes in C:  Or Storage Classes :**

- It is required when we declare any variable

**Four Types of Classes are there**

- Automatic
- Register
- Static
- External

****please refer unit 1 for storage class or variable type**

# Scope Rules

- Global variable is the entire program file.
- Scope of local variable is only up to the block in which it is declare.
- Scope of formal function argument is its own function.
- Scope of auto variable is up to entire program execution time. Its scope is only in the main function.
- Scope of static variable is limited up to the function call. Its lifetime extends till the program is executing.

# Structure in C

# Structure in C

- Structure is user defined data type.
- It is like an array it allow to combine numbers of data of same data type that is why it called **homogenous.**
- While structure is use to combine numbers of data of different data type that is why it called **heterogeneous.**
- **All the elements of a structure are stored at contiguous memory locations.**
- **A structure is a user defined data type** that groups logically related data items of different data types into a single unit.
- A variable of structure type can **store multiple data items of different data types under the one name**
- **Example : employee have name, age, salary, designation, company name, blood group, phone number, address etc,**

# Cont…

To declare structure **struct** keyword is use followed by structure name.

**Syntax :**

```
struct <name of structure>
{
        datatype <variable name>;
        datatype <variable name>;
        datatype <variable name>;
        ……..
        ……..
        ……..
};
```

**Example :**

```
struct employee
{
        int emp_id;
        char name[20];
        float salary;
        char address[50];
        int dept_no;
        int age;
};
```

# Memory Representation of Structure

| Memory address (starting address) | Member of Sturucture | Size occupy |
|---|---|---|
| 8000 | int emp_id; | 2 byte |
| 8002 | char name[20]; | 1 byte * 20 = 20 byte |
| 8022 | float salary; | 4 byte |
| 8026 | char address[50]; | 1 byte * 50 = 50 byte |
| 8076 | int dept_no; | 2 byte |
| 8078 | int age; | 2 byte |

# Declaring variable of structure

There are two way to declare structure variable

**1) Outside of structure (mean in function)**

```
main()
{
        struct <struct name> s1,s2;
}
```

**Example :**

```
void main()
{
        struct student s1,s2,s3;
}
```

Program 1

**2) At the end of structure**

```
struct <structure name>
{
        datatype variable;
        datatype variable;
        ……..
} var1,var2…;
```

**Example :**

```
struct student
{
        int rollnum;
        char name[20];
        long int phone;
        ……
} s1,s2,s3;
```

Program 2

# Initializing structure variable (data member of structure)

- We can not initialize the member or element of structure at the declaration time or inside the structure
- The members of individual structure variable is initialize one by one or in a single statement after the structure declaration. The example to initialize a structure variable is

**struct employee e1 = {1, “Hemant”,12000.00, “3 vikas colony new delhi”,10, 35);**

**Or**

**e1.emp_id=1;**

**e1.name=“Arpankumar”; //not possible without using strcpy() function**

**e1.salary=12000;  e1.dept_no=1**

**e1.address=“ Parul University”;**

# Accessing Members of structure

- For accessing member of structure have to use (.) operator followed by variable of structure and member of structure.
- The structure members cannot be directly accessed in the expression.
- They are accessed by using the name of structure variable followed by a dot and then the name of member variable.
- The method used to access the structure variables are

**e1.emp_id, e1.name, e1.salary, e1.address, e1.dept_no, e1.age.**

- The data with in the structure is stored and printed by this method using scanf and printf statement in c program.

# Structure Assignment

- The value of one structure variable is assigned to another variable of same type using assignment statement. If the e1 and e2 are structure variables of type employee then the statement.

  **e1 = e2;**

- Assign value of structure variable e2 to e1. The value of each member of e2 is assigned to corresponding members of e1.

**Please go with attached program in slide no 25.**

# Array of Structure

- C language allows to create an array of variables of a structure.
- The array of structure is used to store the large number of similar records.
- For example to store the record of 100 employees then array of structure is used.
- The method to define and access the array element of array of structure is similar to other array.
- **The syntax to define the array of structure is,**

  **struct <struct_name> <array_var_name> [<value>];**
- **For Example:-**

  **struct employee e1[100];**

Program for array of structure

# Structure within structure or nested structure

- Nested Structure mean : structure within structure.
- C language define a variable of structure type as a member of other structure type. The syntax to define the structure within structure is

**struct** <struct_name>
{ <data_type> <variable_name>;
**struct** <struct_name>
{ <data_type> <variable_name>;
……..
}**<struct_variable>**;
<data_type> <variable_name>;
};

# Example of nested structure

```
struct employee
{
    int empid;
    char name[20];
    struct date
    {
        int day;
        int month;
        int year;
    }day;
    float salary;
    char designation
};
```

```
struct student
{
    int stdid;
    struct name
    {
        char fname[15];
        char mname[15];
        char lname[15];
    }sname;
    long int phone_no;
    struct qualification
    {
        float hse;
        float ssc;
    }per;
};
```

Program of nested Structure

# Accessing data member of nested structure

- The data member of structure within structure is accessed by using two period (.) symbol. The syntax to access the structure within structure is

**struct _var. nested_struct_var. struct_member;**

**For Example:-**

e1.doj.day;

e1.doj.month;

e1.doj.year;

**Here e1 is variable of structure employee and doj is variable of date structure.**

# Structure as an argument of UDF

- As we have two types of function inbuilt and user defined so one can also pass variable of structure as an argument of user defined function.
- For this create variable of structure either inside the main function or at the end of structure and pass that variable as an argument of function.
- If you declare variable at the end of structure than there is no need to pass it as an argument because it become global variable and global variable is accessible every where through out the program.
- So by passing structure variable as an argument you can read and print the data for structure. Even you can also perform some calculation. Please find program for the same. In next slide.

# Program to pass structure variable in udf

- Program to pass structure variable as an function argument.
- Program of employee with the same concept.
- Program to pass array of structure as an function argument.
- Program to return structure as a return type of user defined function.

# Union

- A union is a user defined data type like structure.
- The union groups logically related variables into a single unit.
- The union data type allocate the space equal to space need to hold the largest data member of union.
- The union allows different types of variable to share same space in memory.
- There is no other difference between structure and union than internal difference.
- The method to declare, use and access the union is same as structure.
- Keyword union is use to declare union. Only difference between union and structure is memory allocation.

# Union

## How to declare union?

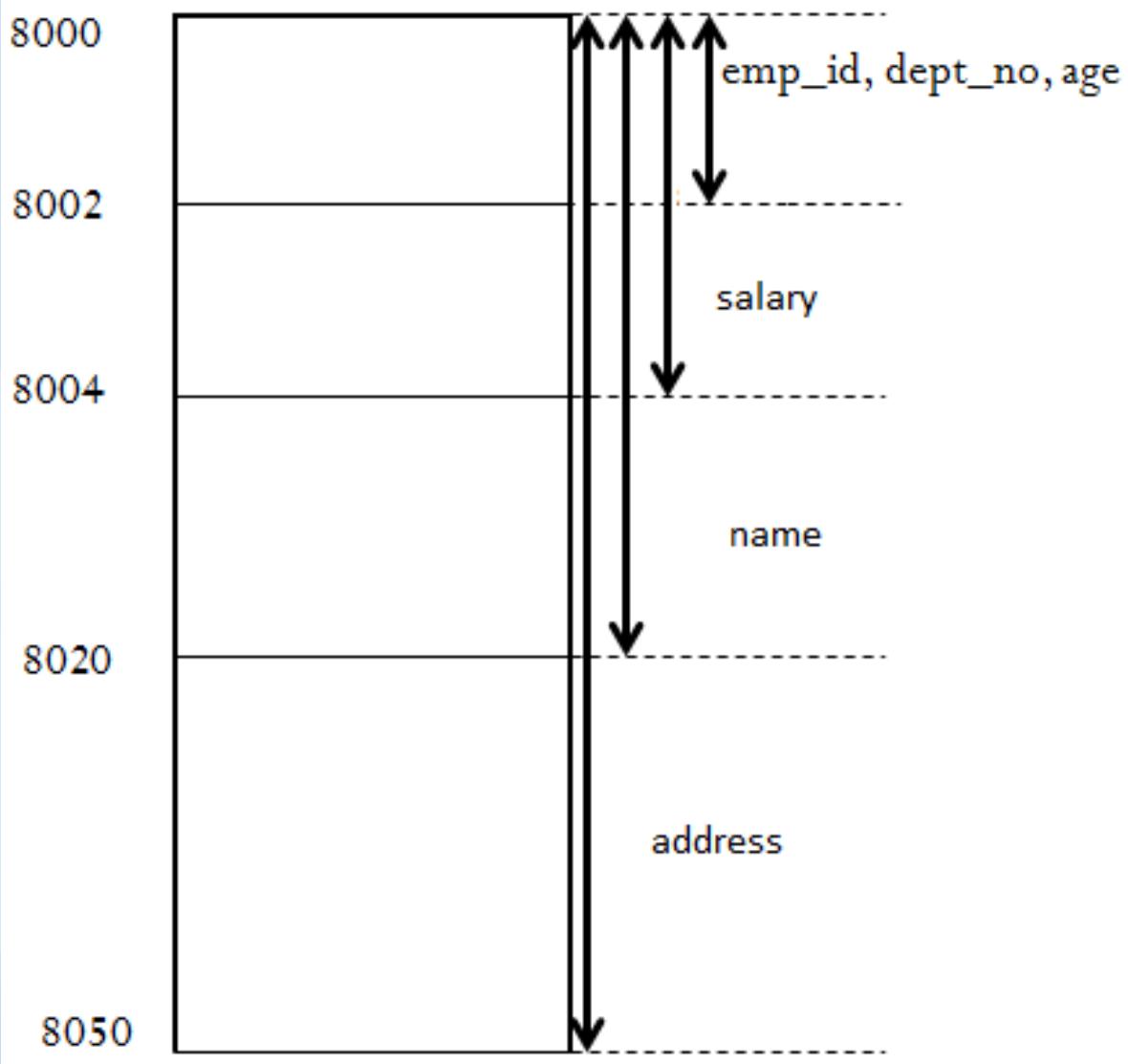
**Syntax :**

```
union <union_name>
 {
      <data_type> <variable_name>;
      <data_type> <variable_name>;
       ……..
      <data_type> <variable_name>;

}var_name;
```

**Example :**

```
union employee
{
            int  emp_id;
        char name[20];
        float salary;
        char address[50];
        int dept_no;
        int age;

};
```

# Memory allocation of union




Program to find sizeof structure and union variable so that you will get to know how memory allocation id different.

# Programs of Union

Program which shows the memory representation of union.

Program to read and print data member of union.

Program to pass union as an argument of function.

Program to create array of union.

**same as structure you can also create nested union

**Nested union : union within a union same like structure.**

Program of nested union.