C Programming

Trainer : Nisha Dingare

Email : nisha.dingare@sunbeaminfo.com

Functions – Introduction :

- Functions are the basic building blocks of C program.
- They are set of instructions written with some name to complete a specific task.
- A function is a enclosed in curly brackets {}, that may take inputs, do the processing and provide the resultant output.
- Types of functions :
  - Built-In functions (Library functions).
  - User defined functions.
- Advantages :
  - It enables reusability and reduces redundancy.
  - It breaks an extensive program into smaller and simpler units.
  - It makes the program easy to understand and manage.

Functions :

- Syntax :

Return_type Function_name(arg1,arg2…. arg n)

{

Function body / statements to be processed

}

1. **Return type** : Here we declare the data type of the value returned by the function. However not all functions return a value. In such cases void keyword is used to indicate the compiler that function does not return a value.
2. **Function name** : This helps the compiler identify the function whenever it is called.
3. **Arguments** : It is the input values given to the function for processing. This defines the data type, sequence, number of parameters to be passed in the function. However the function may or may not take the arguments. This list is optional.
4. **Function body** : It contains all the statements to be processed whenever the function is called.

Fundamental Aspects :

- Functions have 3 important aspects :

- **Function Declaration** : It is the basic prototype of a function. It lets the compiler know the name, number of arguments, data types of parameters and return type of the function. Function declaration is written at the top of source file.

- **Function definition** : It is defining the actual statements that the compiler will execute upon calling the function. You can call it as the function body.

- **Function call** : As the name gives out, a function call is calling a function to be executed by the compiler. You can call the function at any point of time in the entire program. The only thing to take care is that you need to pass the same number and type of parameters as mentioned in the function declaration.

Example : Function with return type

- Function declaration :
  - int addition (int num1, int num2); OR
  - int addition (int,int);
- Function definition :
  - int addition(int num1,int num2) // formal arguments

```
{
        int num3;
        num3 = num1 + num2;
        return num3;
}
```

- Function call :
  - int main()

```
{
  int a = 10. b= 20;
  int result = addition(a,b);  // actual arguments
  printf("result =  %d",result);
}
```

Example : Function without return type

- Function declaration :
  - void addition (int num1, int num2); OR
  - void addition (int,int);
- Function definition :
  - void addition(int num1,int num2) // formal arguments

```
{
    int num3;
    num3 = num1 + num2;
    printf("Result = %d",num3);
}
```

- Function call :
  - int main()

```
{
  int a = 10. b= 20;
  addition(a,b); // actual arguments
}
```

Function Execution :

- When a function is called, function activation record/stack frame is created on stack of current process.
- When function is completed, function activation record is destroyed.
- Function activation record contains:
  - Local variables
  - Formal arguments
  - Return address
- Upon completion, next instruction after function call continue to execute.

Function Types :

- User defined functions
  - Declared by programmer
  - Defined by programmer
  - Called by programmer
- Library (pre-defined) functions
  - Declared in standard header files e.g. stdio.h, string.h, math.h, …
  - Defined in standard libraries e.g. libc.so, libm.so, …
  - Called by programmer
- main()
  - Entry point function –code perspective
  - User defined
  - System declared
  - int main(void) {…}
  - int main(int argc, char *argv[]) {…}

Storage Classes :

- auto
- register
- static
- extern
- Local variables declared inside the function.
  - Created when function is called and destroyed when function is completed.
- Global variables declared outside the function.
  - Available through out the execution of program.
  - Declared using extern keyword, if not declared within scope.
- Static variables are same as global with limited scope.
  - If declared within block, limited to block scope.
  - If declared outside function, limited to file scope.
- Register is similar to local storage class, but stored in CPU register for faster access.
  - register keyword is request to the system, which will be accepted if CPU register is available.

Storage classes :

	Storage	Initial value	Life	Scope
• **auto / local**	Stack	Garbage	Block	Block
• **register**	CPU register	Garbage	Block	Block
• **static**	Data section	Zero	Program	Limited
• **extern / global**	Data section	Zero	Program	Program

- Each running process have following sections:
  - Text
  - Data
  - Heap
  - Stack
- Storage class decides
  - Storage (section)
  - Life (existence)
  - Scope (visibility)
- Accessing variable outside the scope raise compiler error.

static :

- Can be declared within function
- It is necessary to initialize static variables at the time of declaration else it violates the rule of static.
- Static variable is to be initialized with constant value only.
- Static variables helps to retain state of particular variable through multiple calls of same function.
- static variables are initialized only once on first invocation of particular function in which is declared.

register :

- when register storage class is used we try to request register to identify with some name.
- There is no guarantee that our register request will be entertained.
- As number of registers availability is very less our request may be rejected and it will be automatically converted in auto type. Needs more time and slows down performance
- If request is entertained then programmer can enjoy speed/performance of application.
- we can not apply address of operator on registers.
- We can not request registers other than local scope.
- Use of register in global section is not allowed
- Syntax to declare a register variable:
  - register intregvar;

extern :

- Extern keyword extends the visibility of the C variables and C functions.
- We write **extern** keyword before variable to tell the compiler that this variable is declared somewhere else. Basically, by writing extern keyword before any variable tells us that this variable is a global variable declared in some other program file.
- 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 throughout the program.
- While declaring a global variable, some space in memory gets allocated to it like all other variables. We can assign a value to it in the same program file in which it is declared.
- **Extern** is used if we want to use it or assign it a value in any other program file.

Recursion :

- Function calling itself is called a recursion .
- Recursive function is called within its own function definition.
- For successful execution of recursive function, terminating condition is mandatory, else it causes runtime error – stack overflow.
- It follows LIFO execution(Last In First Out) where the function which is called last completes the job first.

Recursion vs Loop :

Recursion

- uses more memory
- Follows LIFO
- more time consumption
- if terminating condition is not mentioned, it causes to runtime error-stack overflow state.

Loop

- less memory utilization
- FIFO
- less time consumption
- If terminating condition is not mentioned, it causes to infinite loop.

Recursion Execution Example :

```
int fact(int n) {
int r;
 if(n==0)
return 1;
r = n * fact(n-1);
return r;
 }
```

```
int fact(int n) {
int r;
 if(n==0)
return 1;
r = n * fact(n-1);
return r;
 }
```

```
int fact(int n) {
int r;
 if(n==0)
return 1;
r = n * fact(n-1);
return r;
 }
```

```
int fact(int n) {
int r;
 if(n==0)
return 1;
r = n * fact(n-1);
return r;
 }
```

```
int fact(int n) {
int r;
 if(n==0)
return 1;
r = n * fact(n-1);
return r;
 }
```

```
int fact(int n) {
int r;
 if(n==0)
return 1;
r = n * fact(n-1);
return r;
 }
```

```
int main() { int
res; res =
fact(5);
printf("%d",
res); return 0; }
```

5! = 5 * 4!
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1 * 0!
0! = 1

Thank You