Function

Structured Programming Language (CSE-1271)

Course Instructor : Mohammed Mamun Hossain
Assistant Professor, Dept. of CSE, BAUST

Outline

1. Introduction
2. Library Function
3. User Defined Function
4. Call by Value & Call by Reference
5. Recursive Function
6. Return Values

Function

- A function is a self-contained program segment that caries out some specific, well-defined task.
- Every C program consists of one(at least) or more functions
- One of these function must be called **main()**
- Execution of a program always begin by carrying out the instructions in **main()**.

- Basically, there are two types of functions in C on basis of whether it is defined by user or not.
  - Library function
  - User defined function

Advantages of user defined functions

- User defined functions helps to decompose the large program into small segments
  - ✓ makes programmer easy to understand, maintain.
- If repeated code occurs in a program.
  - ✓ Function can be used to include those codes and execute when needed by calling that function.
- Programmer working on large project can divide the workload by making different functions.

Working of Function

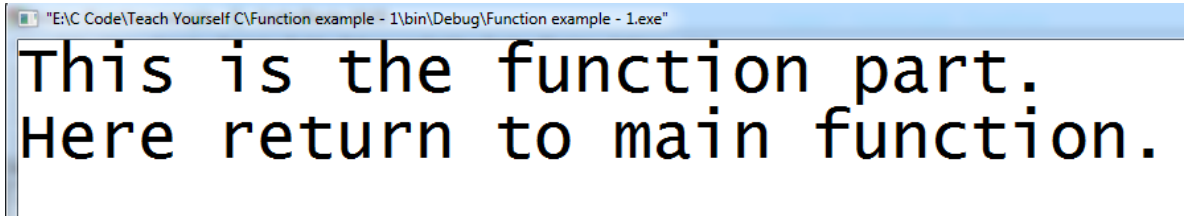
fn

```
#include <stdio.h>
int add(int a, int b);
int main(){
     .............
  sum=add(num1,num2);
     ...........
 }
        int add(int a, int b){
           ..................
           ..................
  }
 Here,
        a=num1
        b=num2
```

Simple example

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

void print(void);

int main()
{
    print();
    printf("\nHere return to main function.\n\n");

    return 0;
}

void print(void)
{
    printf("This is the function part.");
}
```

Function Prototype



General Form of a Function

```
return-type fun-name(type parameter1, type parameter2)
{
    /*……function body…*/
}
```

```
int sum(int n1, int n2)
{
    int s;
    s = n1 + n2;

    return s;
}
```

```
void print(void)
{
    printf("This is the function part.");
}
```

Function Prototype

- The function prototype declares the input and output parameters of the function.
- The function prototype has the following syntax:

type function-name(parameter list);

```
return-type fun-name(type parameter1, type parameter2)
{
    /*……function body…*/
}
```

```
return-type fun-name(type parameter1, type parameter2);
```

Function Prototype

```c
void print(void)
{
    printf("This is the function part.");
}
```

```c
void print(void);
```

Function Prototype

```c
int sum(int n1, int n2)
{
    int s;
    s = n1 + n2;

    return s;
}
```

Function Definition

```c
int sum(int n1, int n2);
```

Function Prototype

- A function prototype declares a function before it is used and prior to its definition.
- Compiler needs to know this information in order for it to properly execute a call to function.
- main() does not need prototype.

Argument and Parameter

- A function's argument is a value that is passed to the function when the function is called.
- A function can have zero to several argument.
- For function to be able to take arguments, special variables to receive argument values must be declared, called parameter of the function.
- The parameters are declared between the parentheses that follow the function's name during function definition.
- Functions that take arguments are called parameterized function.

Argument and Parameter

```c
int sum(int n1, int n2);

int main()
{
    int num1, num2, result;

    printf("Enter numbers: ");
    scanf("%d %d", &num1, &num2);

    result = sum(num1, num2);

    printf("Sum = %d\n\n", result);
}

int sum(int n1, int n2)
{
    int s;
    s = n1 + n2;

    return s;
}
```

Function Prototype

Function Call

Arguments

Parameter

Function Call

- We can accessed a function by **specifying its name**, followed by a **list of arguments enclosed in parentheses** and **separated by commas**.
- If the function call does not require any arguments, an empty pair of parentheses must follow the name of the function.
- A function will carry out its intended action whenever it is accessed (called) from some other portion of the program.
- Once the function has carried out its intended action (end of that function is reached), control will be returned to the point from which the function was accessed.
- Traditionally, main function is not called by any other function, but there is no technical restriction.

all

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

void add(int a, int b);

int main()
{
    int a, b;

    printf("Enter a and b: ");
    scanf("%d %d",&a,&b);

    add(a,b);

    return 0;
}

void add(int a, int b)
{
    printf("Result = %d",a+b);
}
```

1
2
3
4
5
10
7
8

Function Call

Finish!

Start the called function execution

End of Function

Function Type

There are four types of functions and they are:

Function Type	Parameter	Return Value
Type 1	Accepting Parameter	Returning Value
Type 2	Accepting Parameter	Not Returning Value
Type 3	Not Accepting Parameter	Returning Value
Type 4	Not Accepting Parameter	Not Returning Value

Functions with no arguments and no return values

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

void area();

int main()
{
    area();

    return 0;
}

void area()
{
    int a, b;

    printf("Enter the length and width: ");
    scanf("%d %d",&a, &b);

    printf("Area = %d",a*b);
}
```

Functions with arguments and no return values

```c
void area(int a, int b);

int main()
{
    int a, b;

    printf("Enter the length and width: ");
    scanf("%d %d",&a, &b);

    area(a,b);

    return 0;
}

void area(int a, int b)
{
    int result;

    result = a*b;

    printf("Area = %d",result);
}
```

Functions with no arguments and return values

fn

```c
int area();

int main()
{
    int result;

    result = area();
    printf("Area = %d",result);

    return 0;
}

int area()
{
    int a, b, r;

    printf("Enter the length and width: ");
    scanf("%d %d",&a, &b);

    r = a*b;

    return r;
}
```

Functions with arguments and return values

fn

```c
int area(int a, int b);

int main()
{
    int a, b, result;

    printf("Enter the length and width: ");
    scanf("%d %d",&a, &b);

    result = area(a,b);
    printf("Area = %d",result);

    return 0;
}

int area(int a, int b)
{
    int r;

    r = a*b;

    return r;
}
```

Functions arguments

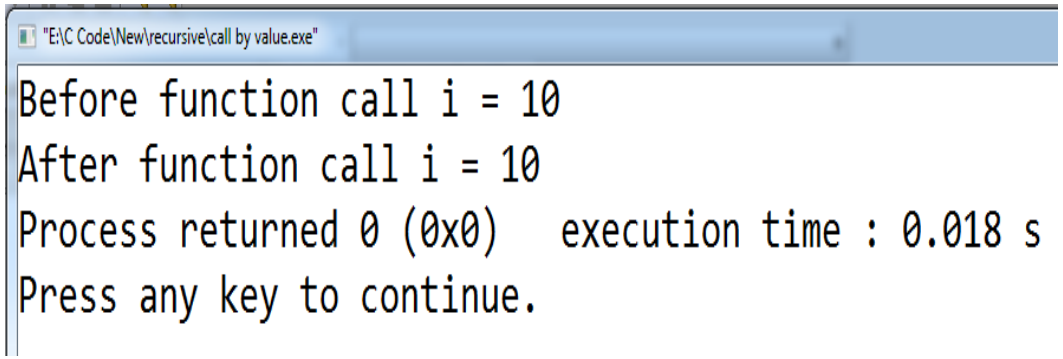
- We can pass arguments to functions in two way
  - Call by value
  - Call by reference

Call by value

- This method copies the value of an argument into the formal parameter of the subroutine (function).
- The change made to a parameter of the subroutine have no effect on the argument used to call it.

Call by value

```c
#include<stdio.h>

void fun(int v);

int main()
{
    int i;

    i = 10;
    printf("Before function call i = %d", i);

    fun(i); //function call by value
    printf("After function call i = %d", i);

    return 0;
}

void fun(int v)
{
    v = 1000;
}
```



Call by Reference

- In this method, the address of an argument is copied into the parameter.
- Should use pointer
- Inside the subroutine, the address is used to access the actual argument.
- This means the **change made to the parameter will affect the argument**.

Call by Reference

```c
#include<stdio.h>

void fun(int *v);

int main()
{
    int i;

    i = 10;
    printf("Before function call i = %d", i);

    fun(&i); //function call by reference
    printf("\nAfter function call i = %d", i);

    return 0;
}

void fun(int *v)
{
    *v = 1000;
}
```

```
"E:\C Code\New\recursive\call by value.exe"
Before function call i = 10
After function call i = 1000
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Call by Reference (Array Argument)

```c
#include<stdio.h>
void fun(int value[]);
int main()
{
    int j, data[5]={10,20,30,40,50};

    printf("Before : ");
    for(j=0; j<=4; j++)
    {
        printf("%d ",data[j]);
    }

    fun(data);

    printf("\nAfter : ");
    for(j=0; j<=4; j++)
    {
        printf("%d ",data[j]);
    }
    printf("\n\n");
    return 0;
}

void fun(int value[])
{
    value[0]=999;
    value[3]=333;
}
```

```c
#include<stdio.h>
void fun(int data[]);
int main()
{
    int j, data[5]={10,20,30,40,50};

    printf("Before : ");
    for(j=0; j<=4; j++)
    {
        printf("%d ",data[j]);
    }

    fun(data);

    printf("\nAfter : ");
    for(j=0; j<=4; j++)
    {
        printf("%d ",data[j]);
    }
    printf("\n\n");
    return 0;
}

void fun(int data[])
{
    data[0]=999;
    data[3]=333;
}
```

What will happen?

```c
#include<stdio.h>

void fun(int i);

int main()
{
    int i;

    i=10;
    printf("Before function call i = %d\n",i);

    fun(i);

    printf("Before function call i = %d\n",i);

    return 0;
}

void fun(int i)
{
    i=45;
}
```

```c
#include<stdio.h>

void fun(int *i);

int main()
{
    int i;

    i=10;
    printf("Before function call i = %d\n",i);

    fun(&i);

    printf("Before function call i = %d\n",i);

    return 0;
}

void fun(int *i)
{
    *i=45;
}
```

Local vs Global Variables

- Variables that are declared inside a function or block are local variables.
- Global variables are defined outside of all the functions, usually on top of the program.

Local Variables Scope

- Local variables can be used only by statements that are inside that function, for which it is local.
- Local variables are created when a function is called and they are destroyed when the function is exited.
- Local variables are not known to functions outside their own.
- Local variable of one function have no relation to the local variable in another function.
- Several functions can have local variables with same name(s) but have no relation with one another.
- Formal Parameters are also local variable to that function.

Global Variables Scope

- Global variables are exist the entire time our program is executing.
- Global variables may be accessed by any function in our program.
- Hold their value during the entire execution of the program.

Local vs Global : Initialization

- When a local variable is defined, it is not initialized by the system, you must initialize it yourself.

- Global variables are initialized automatically by the system when you define them as follows:

Data Type	Initializer
int	0
char	'\0'
float	0
double	0
pointer	NULL

Recursive Function

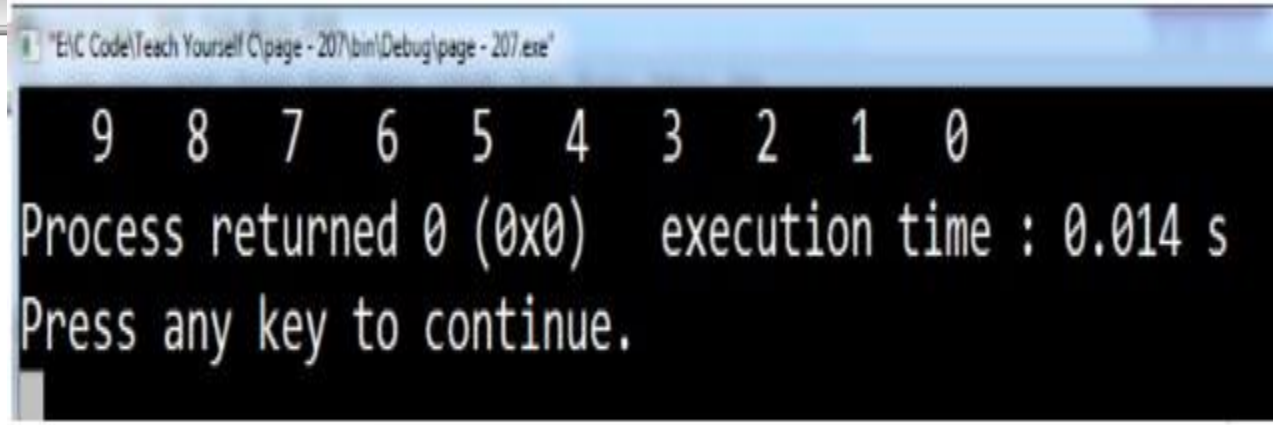
- Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied.
- The process is used for repetitive computations in which each action is stated in terms of a previous result.
- In recursion, no multiple copies of the recursive function will create. Only one copy exist.
- When a function is called, storage for its parameters and local data are allocated on the stack.
- Thus, when a function is called recursively, the function begins executing with a new set of parameter and local variables, but the function code remain same.

Recursive Function

- In order to solve a problem recursively, two conditions must be satisfied.
  - the problem must be written in a recursive form
  - the problem include a stopping condition

main.c

```
3
4   void recurse(int i);
5
6   int main()
7   {
8       recurse(0);
9
10      return 0;
11  }
12
13  void recurse(int i)
14  {
15      if(i<10)
16      {
17          recurse(i+1);
18          printf("%4d", i);
19      }
20  }
21
```



Functions that return multiple values

Method 1 : Using Array

- If more than two variables of same type is to be returned then we can use array .
- Store each and every value to be returned in an array and return base address of that array.

Method 2 : Using Pointer and One Return Statement

- Pointer Variable can updated directly using Value at ['*'] Operator.
- Usually Function can return single value.
- If we need to return more than two variables then update 1 variable directly using pointer and return second variable using '**return Statement**'. [Call by Value + Call by Reference]

Method 3 : Using Structure

- Construct a Structure containing values to be returned and then return the base address of the structure to the calling function.

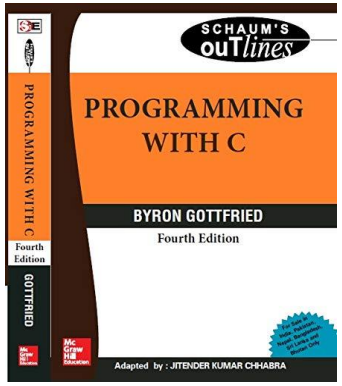
Thank You.

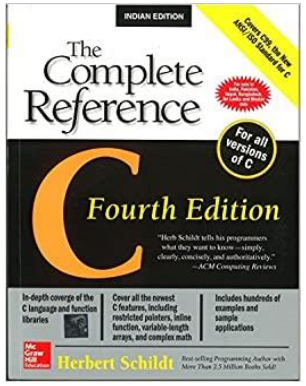
Questions and Answer

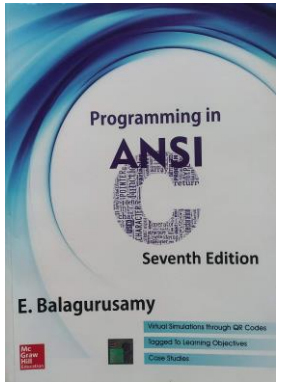
References

Books:

1. Programming With C. *By Byron Gottfried*
2. The Complete Reference C. *By Herbert Shield*
3. Programming in ANSI C *By E. Balagurusamy*
4. Teach yourself C. *By Herbert Shield*







Web:

1. www.wikbooks.org

and other slide, books and web search.