Presentation on Function

Function:

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

Defining a Function:

The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list )
 {
        body of the function
  }
```

Function Return Types

- Can be any of C's data type:
  - char
  - int
  - float
  - long……

- Examples:

```
int func1(...)        /* Returns a type int.   */
float func2(...)      /* Returns a type float. */
void func3(...)       /* Returns nothing.      */
```

Function Prototype:

Function prototypes are usually written at the beginning of a program, ahead of any programmer defined function.

- **Function Prototype Example:**
  - double squared(double number);
  - void print_report(int report_number);
  - int get_menu_choice(void);

```
#include <stdio.h>
Void main (int a);  /*function prototype*/
Main()
{

}
```

Function calling:

```
#include <stdio.h>
   int show ( ) {
   printf("It's a function");
   {
Int main ( ) {
Show ();
}
```

function call

Function Arguments:

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.

The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways that arguments can be passed to a function:

Example of Parameter and Argument:

```
Main () {
  add (10, 20),
}

Add (int i, int j) {
    int j;
    j=i+j;
}
```

//Argument

// Parameter

Actual and Formal parameter:

```
#include <stdio.h>
 void main(void)
{
void test_function(int) ;
test_function(1) ;  \\ actual parameter
}
Void test_function(i_recd_value) \\formal parameter
int i_recd_value ;
{
cout << "Value passed = " << i_recd_value ;
return ;
}
```

Local Variables:

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables. Here all the variables a, b and c are local to main() function.

Global Variables:

Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

Local variable and Actual variable:

```
#include <stdio.h>
 void show (){
 int i=5; ——————→//local variable
Printf("%d\n",i);
}
 int main () {
 int i=2; ——————————→//Actual variable
  Printf("%d\n",i);
   show ();
}
```

Call by Value:

Call by Value, means that a copy of the data is made and stored by way of the name of the parameter. Any changes to the parameter have **NO** affect on data in the calling function.

Example Call by value:

```
#include<stdio.h>
 void c(int n, int m) {
   int temp;
temp = n;
n = m; m = temp;
 }
int main() {
 int n=50,m=70;
c(n,m);
 printf("\nNumber  : %d",n);
printf("\nNumber 2 : %d",m);
 return(0); }
```

Call by Reference:

A reference parameter "refers" to the original data in the calling function. Thus any changes made to the parameter are ALSO MADE TO THE ORIGINAL variable.

Example of Call by Reference:

```
#include <stdio.h>
void swap(int *x, int *y);
 int main () {
int a = 100; int b = 200;
printf("Before swap, value of a : %d\n", a );
printf("Before swap, value of b : %d\n", b );
swap(&a, &b);
 printf("After swap, value of a : %d\n", a );
printf("After swap, value of b : %d\n", b );
 return 0;
 }
```

Array of function:

```
#include <stdio.h>
float average(float a[]);
 int main(){
 float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18}; avg=average(c);
printf("Average age=%.2f",avg);
 return 0; }
 float average(float a[]){
int i;
float avg, sum=0.0;
for(i=0;i<6;++i){ sum+=a[i]; }
 avg =(sum/6);
return avg;
 }
```

Recursion:

Recursion defines a function in terms of itself.

It is a programming technique in which a function calls itself.

A recursive function calls itself repeatedly, with different argument values each time.

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).

Example:

```
#include <stdio.h>
int sum(int n);
int main(){
    int num,add;
    printf("Enter a positive integer:\n");
    scanf("%d",&num);
    add=sum(num);
    printf("sum=%d",add);
}
int sum(int n){
    if(n==0)
   return n;
    else
       return n+sum(n-1);    /*self call  to function sum() */
}
```

Thank You

Md. Abu Zaman
Daffodil International University