Chapter 7 FUNCTIONS

- **Functions** are blocks of code that perform a number of pre-defined commands to accomplish something productive. You can either use the built-in library **functions** or you can create your own **functions**. **Functions** that a **programmer** writes will generally require a prototype.
- The complex problem may be decomposed into small or easily manageable parts and each module is called a function.
- The functions are very useful to read, write, debug, modify and they are easy to use in the main program.
- In C programming, the main () itself is a function, this means that the main function is invoking the other functions to perform the task.
- The main body of a C program, identified by the keyword main, and enclosed in by the left and right parentheses is a function.

Function,...

- Programs combine user-defined functions with library functions
- There are basically 2 types of function:

  -Library functions: **e.g:** printf(),scanf(),…

  -User defined functions: **e.g:** message(),…
- **Library functions:** are functions already known by the machine that is built in the compiler.

UNIVERSITY Of KIGALI

Library functions

- Libraries are very important in C because the C language supports only the most basic features that it needs.
- The codes are often placed in **libraries** to make them easily reusable.
- We have seen the standard I/O, or **stdio**, library already: Standard libraries exist for standard I/O, math functions, string handling, and so on.
- You can use libraries in your own programs to split up your programs into modules.
- This makes them easier to understand, test, and debug, and also makes it possible to reuse code from other programs that you write.

UNIVERSITY Of KIGALI

Some Built in Functions for String Handling

- **string.h**
- The following macros are built into the file string.h

• strcmp	compares two strings
• strcpy	copies one string to another
• strlen	finds length of a string
• strlwr	converts a string to lowercase
• strncat	appends n characters of string
• strrev	reverses string
• strupr	converts string to uppercase

UNIVERSITY Of KIGALI

Example1: Using function to convert a string to uppercase

```
#include <stdio.h>
#include <string.h>
main()
{       char name[80];
/* declare an array of characters 0-79 */
printf("Enter in a name in lowercase\n");
scanf( "%s", name );
strupr( name );
printf("The name is uppercase is %s", name );
}
```

Sample Program Output

```
Enter in a name in lowercase
samuel
The name in uppercase is SAMUEL
```

Example2: C program to display a name in lowercase, reverse and show its length using function.

```
#include<stdio.h>
#include<string.h>
main()
{
char name[25];
 printf("Enter name in upper case:\n");
scanf("%s",&name);
strlwr(name);
printf("The name in Lowercase is %s\n",name);
strrev(name);
printf("The name in reverse is %s\n",name);
printf("The length of name is:%d\n",strlen(name));
}
```

- **Sample Program Output**
- Enter in a name in uppercase   SAMUEL
- The name in Lowercase is samuel
- The name in reverse is leumas
- The length of name is:6

Math Functions :math.h

sqrt( ) calculate the square root of a number

cos( ) calculate the cosine

sin( ) calculate the sine

tan( ) calculate the tangent

abs( ) calculate the absolute value

fabs( ) calculate the absolute value of a floating point

ceil( ) is used to round up to an integer

floor( ) is used to round down to an integer

exp( ) is used to compute a floating point approximation to the exponential function

log( ) is used to compute a floating point approximation to the exponential function

log10( ) is used to compute a floating point approximation to the natural logarithms function

pow( ) is used to compute the power of a number

Example3: C program to calculate the square root of a number using math function

```
#include<stdio.h>
#include<math.h>
main()
{
float a,b;
printf("Enter a number:\n");
scanf("%f",&a);
printf("The square root of %.2f is %.3f\n",a,sqrt(a));
printf("The power of %.2f is %.3f\n",a,pow(a,2));
}
```

UNIVERSITY Of KIGALI

Functions for Character Handling

The following character handling functions are defined in **ctype.h**
tolower converts character to lowercase toupper converts character to uppercase

Example 4:
To convert a string array to uppercase a character at a time using toupper( )

```
#include <stdio.h>
#include <ctype.h>
    main()
{
char name[80];
    int loop;
    printf("Enter in a name in lowercase\n");
     scanf( “%s", name );
    for( loop = 0; name[loop] != 0; loop++ )
     name[loop] = toupper( name[loop] );
    printf("The name is uppercase is %s", name );
    }
```

Example 5: A program to calculate the volume of a sphere

```
/* volume of a sphere Formular:V=4/3(r*r*r)*pi */

#include<stdio.h>
#include<math.h>
#define pi 3.1421
main()
{
float r,v;
printf("Enter the radius:\n");
scanf("%f",&r);
v=4*pow(r,3)*pi/3;
printf("The volume is:%.2f\n",v);
return 0;
}
```

UNIVERSITY Of KIGALI

User Defined Functions:

Are those functions developed by the user.
In function we have: A program that calls or activates the function and The function itself.

```
#include<stdio.h>
 main()
{
message();          /* Calling of a function */
printf("This is a C function\n");
}
message()           /*  Function declaration*/
{
printf("\n Message function");
}
```

Output

```
Message function
This is a C function
```

- Here, through main() we are calling the function message().
- What do we mean when we say that main() calls the function message()?
- We mean that the control passes to the function message(). The activity of main() is temporarily suspended;
- it falls asleep while the message() function makes up and goes to work.
- When the message runs out of statements to execute, the control returns to main(), which comes to life again and begins executing its code at the exact point it left off.
- Thus, main() becomes calling function, whereas message() becomes the called function.

UNIVERSITY Of KIGALI

The general form of a function:

-
- function_name(arg1,arg2,….argn)
- type arg1,arg2,…argn;
- {
- statements;
- statements;
- ……..
- }
- **N.B:** A function definition has 2 parts:
- -argument declaration
- -body of the function

Summary

```
#include<stdio.h>
main()
{
printf("Country\n");
rwanda();
kenya();
}
rwanda()
{
printf("Rwanda\n");
}
kenya()
{
printf("Kenya\n");
}
```

Output

```
Country
Rwanda
```

1. C program is a collection of one or more functions.
2. A function gets called when is followed by a semicolon
3. A function is defined when function name is followed by a pair of braces in which one or more statements may be present.
4. Any function can be called from any other function.
5. A function can be called any number of times
6. The order in which the functions are defined in a C program and the order in which they get called need not necessary to be same but for logical output it should be respected.
7. A function can call itself. Such process is called "**recursion**".

From this program a number of conclusion can be taken:

- Any C program contains at least one function
- If a program contains one function, it must be main()
- In a C program if there are more than one functions present, then one (and only one) of these functions must be main(), because program execution always begins with main().
- There is no limit on number of functions that might be present in a C Program
- Each function in a program is called in the sequence specified by the function calls in main().
- After each function has done its thing, control returns to main, when main() runs out of function calls, the program ends.
- The program execution always begins with main()

UNIVERSITY of KIGALI

Example1 : factorial upto thousands

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n;
long int fact=1;
clrscr();
printf("Enter a number:\n");
scanf("%d",&n);
getch();
for(i=1;i<=n;i++)
{
fact=fact*i;
}
printf("The factorial of %d is equal to
%d",n, fact);
getch();
}
```

Example 2 : factorial number upto billions

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n;
Long long int fact=1;
clrscr();
printf("Enter a number:\n");
scanf("%d",&n);
getch();
for(i=1;i<=n;i++)
{
fact=fact*i;
}
printf("The factorial of %lld is equal to
%d",n, fact);
getch();
}
```

UNIVERSITY Of KIGALI

- **Recursive function**

- In C, it is possible for the functions to call themselves. A function is called "**recursive**" if a statement within the body of a function calls the same function (sometimes called circular definition).
- Recursion is thus the process of defining something in terms of itself.

Example:

/*Recursive function to calculate the factorial value */

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a;
 long int fact;
clrscr();
printf("Enter any number\n");
scanf("%d",&a);
fact=rec(a);
printf("Factorial number of  %d is: %ld",a,fact);
getch();
}
```

```
int rec(int x)
//or rec(int x)
{
long int f;
if(x==1)
return(1);
else
f=x*rec(x-1);
return(f);
}
```

- Normally for all computer programs (either C, C++, Java, etc) are having datatypes, variables and functions as main building blocks to develop an application or to write a program.
- The function belong to any of the classifications:

- No arguments (inputs) and No return values (Outputs)

- With arguments and No return values

- With arguments and with return values

- No arguments and with return values
- **Examples1 :**
- **No arguments and No return values**
- Once there are not existing: we need to put void main()

UNIVERSITY Of KIGALI

- **Examples2:**

With arguments and No return values

Once there are not existing: we need to put int main()

Or void function (int X) for passing character.

Examples3:

With arguments and with return values

Int function (char x)

{ Arguments } return value;

Examples4:

No arguments and with return values

Float function (void)

{

Arguments

Return 34.56

}

UNIVERSITY Of KIGALI

- 8.A function can be called from other function. Thus the following program code would be wrong, since argentina() is being defined inside another function, main()
- **e.g :**

```
main()
{
print("\n I am in main");
argentina()
{
printf("\n I am in argentina\n");
}
}
```

UNIVERSITY
Of KIGALI

Passing values between functions

```
#include<stdio.h>
#include<conio.h>
int sum(int a, int b);
 main()
{
int a,b,total;
printf("Enter 2 numbers:\n");
scanf("%d %d",&a,&b);
total= sum(a,b);
printf("The Sum of variables a and b =%d\n",total);
getch();
}
int sum(int x , int y)                    // or calsum(int x, int y)
{
int d;
d=x+y;
printf("The following is:\n");
return (d);
}
```

- In the calsum() function these values get called in two variables x and y
- calsum(x,y)
- int x,y;
- The variables a and b are called “**actual arguments**”, whereas the variables x y are called “**formal arguments**”. Any number of arguments can be passed to a function being called. The order, type and number of the actual and formal arguments must always be same.

UNIVERSITY Of KIGALI

- The 2 methods of declaring the formal arguments are the same:
- calsum()
- int x,y;
- and
- calsum(int x,int y) commonly used
- In the above program if we want to return the sum of x, y, it is necessary to use the **return** statement.
- The return statement serves 2 purposes:
- On executing the return statement it immediately transfers the control back to the calling program.
- It returns the value present in the parentheses after return to the calling program. In the above program the value of sum of 2 numbers is being returned.

UNIVERSITY of KIGALI

- If we want that a called function should not return any value, in that case, we must mention so by using the keyword **void**.

```
void message()
{
printf("C programming is easy\n");
}
```

- A function can return only one value at time. The following statement is invalid: return(a,b);

Function declaration and Prototypes

- A **function prototype** declares the function name, its parameters, and its return type to the rest of the program.
- This is done before a function is declared. (In most cases at the beginning of a program)
- Any C function by default returns an int value.

UNIVERSITY Of KIGALI

What is the difference between return 0 and getch in C programming?

In C, any function that is declared with void, for example **void** main() can not return a value. However, functions that are declared with **int**, **float** or **char** **MUST** return a value. (**return** is a statement used to return a value at the end of a function).

Example:

```
int add(int a, int b)
{
int c;
c = a + b;
return c;
}
 void main()
{
int x, y, z;
x = 5;
y = 10;
z = add(x,y);
print("%d", c);
}
```

In the aside program there are two functions. add() and main(). When the program is executed, the main() function runs first. Inside is a call to the add() function. So when the program reaches **z = add(x,y);**, it sees there is a function and the computer decides to execute that function. And the add() function **RETURNS a value** or like sends the value of c as a message to the main function, which is stored in z and then printed.

This program has been written with **void** main(). However, most new compilers of C do not accept main() functions declared with void. **int** main() is the widely accepted standard. So main() is declared as int. Now we saw earlier that functions defined with int() MUST return a value. The function add() returned the value of c. Similarly main() functions return the value 0. That's it. **return** 0; means the program has correctly executed.

UNIVERSITY Of KIGALI

Rewriting above program with int main()

Example:

```
 int add(int a, int b)
{
int c;
c = a + b;
return c;
}

int main()
{
int x, y, z;
x = 5;
y = 10;
z = add(x,y);
print("%d", c);
return 0;
}
```

There is absolutely no difference in what is printed by the above two programs.
Now that we've explained what return is, let us look at getch(). This is a function that can record a single character or a single press of any keyboard button. It stands for "get character". The reason why getch() is used is for user friendliness. In this case, the user is you. When you execute the above code sample in Turbo C, it opens a new Command Prompt window, runs the code, prints the value and closes the window so fast, you wouldn't even know what happened. How will you know the program executed correctly if you couldn't even see what printed?

Therefore getch() is used. Rewriting the above program with getch():

Example:

```
 int add(int a, int b)
{
int c;
c = a + b;
return c;
}
 void main()
{
int x, y, z;
x = 5;
y = 10;
z = add(x,y);
print("%d", c);
getch();
}
```

So when the program has printed the value of c, it reaches getch( ).
Now the Command Prompt window will wait for you to press a single key therefore actually letting you see what was printed.

UNIVERSITY Of KIGALI

Storage classes

- In C, all the variables have data types and storage classes.
- Every identifier also has a storage class that provides its visibility, lifetime and location. There are 4 different storage classes in C.
- They are

- Automatic storage class

- Register storage class

- Static storage class

- Extern storage class

Automatic storage class

- Variables defined inside a function are local (internal) to the function they are declared, and called automatic variables.
- They are more often referred to as automatic, after the fact that their memory space is automatically allocated as the function is entered and released.
- In other words, automatic variables are given only temporary memory space.
- They are only accessed by the function in which it is defined.
- They have no meaning outside the function in which they are declared.
- The C compiler treats any variable declared inside a function as an automatic variable,
- it is not necessary to specify the keyword **auto** along with the variable declaration

UNIVERSITY Of KIGALI

```
#include<stdio.h>
main()
{
auto int x=5;
clrscr();
  {
    auto int x=4;
{
  auto int x=3;
   printf("%d\t",x);
}
           printf("%d\t",x);
}
printf("%d\t",x);
}
```

- The output will be   3   4   5

Register storage class

- The register storage class is similar to the previous one since the variables defined inside a function are local.
- It can be accessed by the function in which it is defined
- Its values cannot be accessed by any other function.
- When defining a register variable inside a function it is more precise in C to use the keyword **register** before the definition of the variable.

UNIVERSITY of KIGALI

```
#include<stdio.h>
main()
{
register int  x;
for(x=1;x<=10;x++)
printf("\n%d",x);
}
```

- We cannot use the register storage class for all types of variables

i.e

register double x;

register float y;

Static storage class

- The static variables are stored in the memory. When the value of a static variable is not initialized it takes a value of zero.
- The word static in general refers to anything that is inert to change. When defining static variable inside a function is more precise in C use the keyword **static** before the definition of the variable.

UNIVERSITY Of KIGALI

C Program to add two numbers using static storage class

```
#include<stdio.h>
main()
{
int i;
clrscr();
for(i=1;i<=10;i++)
printf("%d ,%d\n",i,f(i));
}
f(x)
int x;
{
static int s=100;
return s+=x;   //s=s+x
}
```

Extern storage class

- Variables that are both alive and active throughout the entire program are called as external variables.
- They are also called global variables;
- they can be accessed by any function in a program.
- Global variables do not belong to any particular function.
- When defining an extern variable or global variable in a program, it is more precise in C to use the keyword **extern** before the definition of the variable.

UNIVERSITY Of KIGALI

example

- #include<stdio.h>
- int x;
- main()
- {
- x=1;
- f();
-
- }
- f()
- {
- printf("Value = %d",x);
- }
- **output**
- Value=1

Exercises

1) Write a function that adds two integers and prints the sum.

3) Write a C program to compare two strings using library function

4) C Program to Compare Two Strings Without Using Library Function

UNIVERSITY Of KIGALI