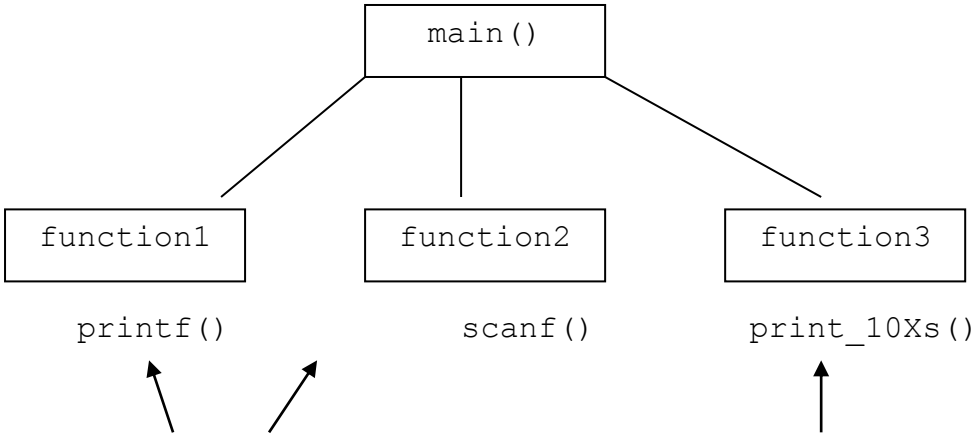
FUNCTIONS

A function is a module that performs its task by accepting a set of input values, and after doing the necessary calculations **returns zero** or **one** output value.

All C programs must contain the function main(). Execution starts from the main function

Modular Design



standard library functions requires `#include<stdio.h>`

user defined function.

How Functions Work

- The code of a function is not part of `main`. i.e. `main()` does not contain the code of any standard-library or user-defined function. For example, the code of `printf()` is in `<stdio.h>`. the code of a user-defined function can be inside the same file together with `main()` or can be in a different file.

```
main()
{
:
}
function()
{
:
}
```

- Functions may require one or more or no argument to perform their tasks. Arguments are inside parentheses separated by a comma.

```
function_name(argument1, argument2, . . .)
```

e.g. `printf("\n Sum=%d",sum)`

arg1 arg2

Input values

- When a C program is executed, `main()` transfers control of the computer to a standard-library or user-defined function; when the function completes its task it returns control back to `main()`.

- The input values that a program passes to a function are called arguments, and the output of the function is called return value.

- Just as variables must be declared before they are used in a program, functions must also be declared. Functions are usually declared before `main()`.

Declaration of user defined functions

Function Prototype:

```
return_type function_name(argument_type, argument_type, . . .);
```

must use a semicolon here

Note that return_type is the data type of the value that is returned by the function

- use the keyword `void` when there is no return value or no argument

Examples:

```
int Rect_Area(int, int);        /*accepts sides, returns area*/
```

area_type; Function_name; Side1_type; Side2_type

```
double Sales_Tax(double, double);
```

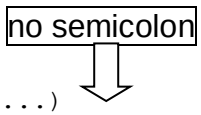
total_price_type; unit_price_type; tax_rate_type

```
void Print_char(char, int);  /*prints a character integer times*/
```

means function does not return any value; 1st arg. is a character; 2nd arg. is an integer

Function Definition:

After the declaration, function must be defined. A function definition has the following format.




```
return_type function_name(arg1_type arg1_name, arg2_type arg2_name, ...)
{
function body;
}
```

Examples:

```
int Rect_Area(int side1, side2)
{
int area;           /* declaration */
area=side1*side2;
return area;
}



void Print_Char(char ch, int num)
{
int i;     /*declare variables except arguments*/
for(i=1; i<num; ++i)
     printf("%c", ch);
return;          /* just goes back to main */
}



void Print_10Xs(void)
{
int j;
for(j=1; j<=10; ++j)
     printf("X");
return;
}
```

Exercises

1) Write a program that displays the following pattern on the screen using `Print_10Xs()` function:

```
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
```

Analysis: `Print_10Xs()` function should be used 5 times

```
#include<stdio.h>
void Print_10Xs(void);      /* function declaration */
main()
{
int i;
for(i=1; i<=5; ++i)
      {
      Print_10Xs();
            printf("\n");
      }
} /* main() ends here */

void Print_10Xs(void)
{
int j;
for(j=1; j<=10; ++j)
     printf("X");
return;
}
```

2) Using `Print_Char()` function write a program that accepts a character and displays a triangular pattern as shown in the Sample Run:

Sample Run

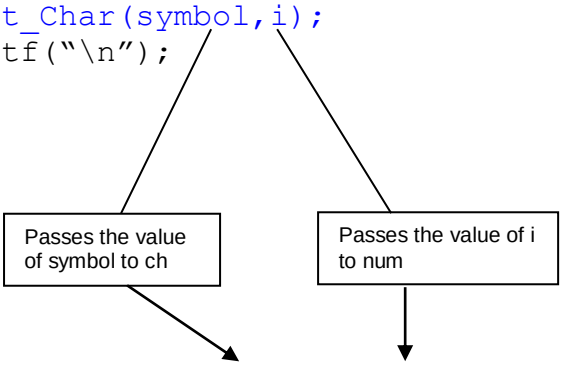
```
Enter a character: A
A
AA
AAA
AAAA
AAAAA
```

Analysis: `Print_Char()` function should be used 5 times

```
#include<stdio.h>
void Print_Char(char, int);
main()
{
int i;
char symbol;

printf("Enter a character:");
scanf("%c",&symbol);

for(i=1;i<=5; ++i)
     {
     Print_Char(symbol,i);
     printf("\n");
     }
}
```




```
void Print_Char(char ch, int num)
{
int j;    /*declare variables except arguments*/  for(j=1; j<=num; ++j)
     printf("%c",ch);
return;
}
```

Function with Arguments and with Return Value

```
#include<stdio.h>
double Sales_Tax(double, double);

main()
{
     double item_price, tax _rate, total_price;
     printf("Enter item price:");
     scanf("%lf", &item_price);

     printf("Enter tax_rate:");
     scanf("%lf", &tax_rate);

     total_price=Sales_Tax(item_price, tax_rate);
     printf("Total price is %.2f", total_price);
}
double Sales_Tax(double price, double tax)
{
     double result;
     result=price + price*tax;
     return result;
}
```

Sample Run:

```
Enter item price: 300
Enter tax rate: 0.10
Total Price is 330.00
```

Function with No Arguments and with Return Value

```
#include<stdio.h>
double Value_Pi(void);

main()
{
      double radius, a_circle;
      printf("Enter the radius:");
      scanf("%lf", &radius);
      a_circle= Value_Pi()*radius*radius;
      printf("Area of this circle is % .2f", a_circle);
}

double Value_Pi(void)
{
      double result;
      result=22.0/7.0;
      return result;
}
```

Sample Run:

```
Enter the radius:10
Area of this circle is 314.29
```

Passing Argument by Value:

When main() communicates with a function, it passes the value of an argument, not the argument itself.

Example:

```
#include<stdio.h>
int Func_Inc(int);
main()
{
      int main_num, func_num;
      printf("Enter an integer:");
      scanf("%d", &main_num);

      func_num=Func_Inc(main_num);
      printf("\n\n After Func_Inc() is executed:");
      printf("\n main_num = %d", main_num);
      printf("\n func_num ) %d", func_num);
}

int Func_Inc(int main_num)
{
main_num=main_num+1;
return main_num;
}
```

Sample Run:

```
Enter an integer :5
After Func_Inc() is executed:
main_num = 5
func_num = 6
```

Global and Local Variables:

- Any function whose definition comes after the declaration of a global variable, can access that variable.
- Global variables are the variables declared outside the `main()` or any other function.
- Local variables are variables declared inside the `main()` or any other function.

Example:

```
#include<stdio.h>
void Func(void);
int var_2 = 200; /*global variable*/

main()
{
     printf("\n\n main() s result BEFORE Func():");
     printf("\n var_2 = %d", var_2);
     printf("\n\n Func() s RESULT: ");
     Func();
     printf("\n\nmain() s result AFTER Func():");
     printf("\nvar_2 = %d", var_2);
}

void Func(void)
{
printf("\n var_2 = %d", var_2);
return;
}
```

Output:

```
main() s result BEFORE Func();
var_2 = 200

Func() s RESULT:
var_2 = 200

main() s result AFTER Func():
var_2 = 200
```

Warning: A function can change the value of a global variable. You should try avoid using global variables unless it is necessary.

Example:

```
#include<stdio.h>
void Func(void);
int var_2 = 200; /*global variable*/

main()
{
     printf("\n\n main() s result BEFORE Func():");
     printf("\n var_2 = %d", var_2);
     printf("\n\n Func() s RESULT: ");
     Func();
     printf("\n\nmain() s result AFTER Func():");
     printf("\nvar_2 = %d", var_2);
}

void Func(void)
{
     var_2 = 300; /*changes the value of global variable*/
     printf("\n var_2 = %d", var_2);
     return;
}
```

Output:

```
main() s result BEFORE Func();
var_2 = 200

Func() s RESULT:
var_2 = 300

main() s result AFTER Func():
var_2 = 300
```

Local variables are the variables declared inside the function. Because local variables are known only to the functions in which they are declared, we can declare variables with the same name in different functions of the same program.

Example:

```
#include<stdio.h>
void Func(void);

main()
{
      int var_2 = 200; /*local variable*/
      printf("\n\n main() s result BEFORE Func():");
      printf("\n var_2 = %d", var_2);
      printf("\n\n Func() s RESULT: ");
      Func();
      printf("\n\nmain() s result AFTER Func():");
      printf("\nvar_2 = %d", var_2);
}

void Func(void)
{
int var_2 = 300; /* changes the value of global variable */
printf("\n var_2 = %d", var_2);
return;
}
```

Output:

```
main() s result BEFORE Func();
var_2 = 200

Func() s RESULT:
var_2 = 300

main() s result AFTER Func():
var_2 = 200
```