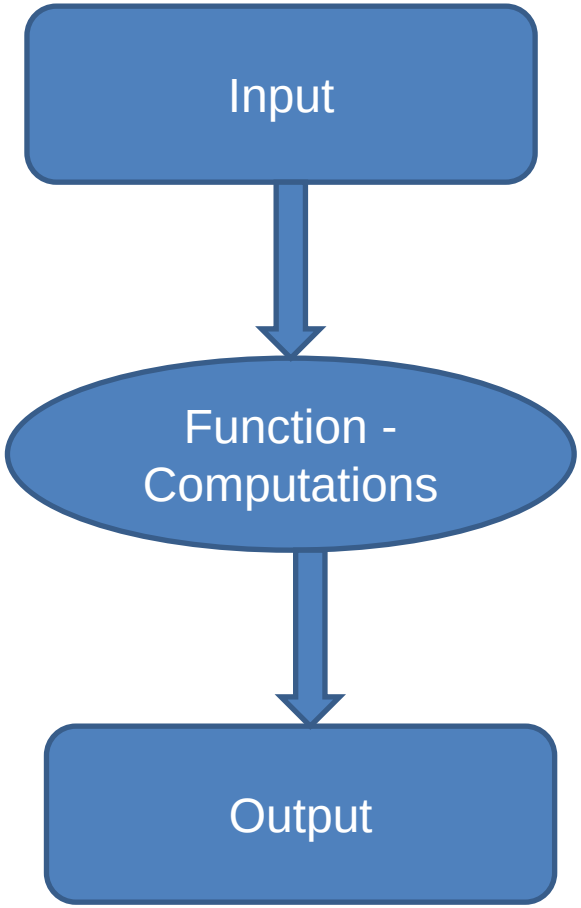
“C Programming”

Functions

Functions

- **Definition:** A function is a self-contained program segment that carries out some specific, well-defined task.
- Every C program consists of one or more functions.
- Function should be declared or prototyped before it is being used anywhere
- One of these functions must be called “main” * (conditions apply)
- Function will carry out its intended action whenever it is called.



Defining a Function

- **Function Definition:** A function definition has two principal components:
  - the first line - Return type, Function Name, Parameters or Arguments
  - the body of the function
- Every C program consists of one or more functions.

Syntax:

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

Function Prototypes

- **Function Declaration:** A function **declaration** tells the compiler about a function name and how to call the function.
- Parameter names are not compulsory, but data types must be mentioned.

Syntax:

```
return_type   function_name( parameter list );
```

- **Function Call:** To use a function, you will have to call that function to perform the defined task.
- When a program calls a function, program control is transferred to the called function.
- A called function performs defined task and when its return statement is executed

Function prototypes…

Function prototype at the beginning of C code:

- describe what the function returns and what it takes as input
- Prototype should be followed by semicolon
- Parameter list is optional
- Default return type is integer

Examples:

float sqrt(float);
double max(double , double);

Example:

```
double  max( double param1, double param2)

    {
      if (param1 > param2)
        {
          return param1;       }
      else
        {
          return param2;
        }
    }
```

Return type of a C Function

- Every C function must specify the type of data being generated by the function.
- If the function does not generate any data, its return type can be “void”.
- The type of expression returned must match the type of the function, or be capable of being converted, otherwise we can see undefined behaviour.

Example:

```
void print_happy_birthday( int age )
   {
      printf("Congratulations on your %d th Birthday\n", age);
      return;
    }
```

Functions - Example

Example:

```
#include <stdio.h>
long int facrorial( int ); → Function prototype
int main()
{
   int n;  long int fact;
   fact = factorial( n ); → Function Call
   printf("Factorial of %d is %ld", n, fact);
   return 0;
}
long int factorial( int n )   /* calculate the factorial of n */
{
   int i;
   long int prod = 1;
   if (n > 1)
         for (i = 2; i <= n; ++i)
                  prod *= i;
   return(prod);
}
```

Actual Argument

Formal Argument

Function Parameters

- Parameters are the symbolic name for "data" that are passed to a function
  - Formal Arguments - Called function
  - Actual Arguments - Calling function
- Two ways that arguments can be passed to a function:
  - Call by value
    - Copy of data is sent to the function being called
    - Doesn't affect the original data
  - Call by Reference
    - Reference to the data is sent
    - Reference parameter "refers" to the original data in the calling function.
    - Changes affect the original data

```
void swap(int x, int y)

// call by value
swap(x,y);
```

```
void swap( int *x, int *y)

//call by reference
swap( &x, &y);
```

Function Parameters…

Point	Call by Value	Call by Reference
Copy	Duplicate Copy of Original Parameter is Passed	Actual Copy of Original Parameter is Passed
Modification	No effect on Original Parameter after modifying parameter in function	Original Parameter gets affected if value of parameter changed inside function

Function Parameters - Rules

- The number of arguments in a function call must be the same as the number of parameters in the function definition. This number can be zero.
- The max no. of arguments is 253 for a single function. (Machine dependent)
- If a function is called with arguments of the wrong type, **the presence of a prototype** means that the actual argument is converted to the type of the formal argument ‘as if by assignment’. If prototype is not present then it may give garbage values or undefined behaviour.
- In the old style, parameters that are not explicitly declared are assigned a default type of int.
- The scope of function parameters is the function itself. Therefore, parameters of the same name in different functions are unrelated.

Functions - Rules

- **Function Declaration:** A function **declaration** tells the compiler about a function name and how to call the function.
- C program can have one or more functions.
- Any function can be called from any function
- A function can be called any number of times.
- A function can call itself, called RECURSION.
- A function cannot be defined in another function.

Functions with no arguments No return value

```
#include<stdio.h>
 void area(); // Prototype Declaration
int main()
 {
        area();
}
void area()
 {
        float area_circle;
        float rad;
        printf("\nEnter the radius of circle : ");
        scanf("%f",&rad);
        area_circle = 3.14 * rad * rad ;
        printf("Area of Circle = %f",area_circle);
 }
```

Functions with arguments No return value

```
#include<stdio.h>
 void area(float rad);

void main()
 {
    float rad;
   printf("nEnter the radius : ");
   scanf("%f",&rad);
   area(rad);
}
 void area(float rad)
 {
   float ar;
   ar = 3.14 * rad * rad ;
   printf("Area of Circle = %f",ar);
}
```

Functions with arguments with return value

```
#include<stdio.h>
float calculate_area(int);
int main()
 {
        int radius;
        float area;
        printf("\nEnter the radius of the circle : ");
         scanf("%d",&radius);
        area = calculate_area(radius);
        printf("\nArea of Circle : %f ",area);
         return(0);
 }
 float calculate_area(int radius) {
        float areaOfCircle;
        areaOfCircle = 3.14 * radius * radius;
        return(areaOfCircle);
 }
```

Functions - Advantages

Advantages:

- Modularity
- Readability
- Reusability
- Easy to develop, debug and test
- Allows test-driven development
- Allows unit testing
- Allows top-down modular approach

THANK YOU