UNIT 8
USER DEFINED FUNCTION

LH – 5HRS

PRESENTED BY:

ER. SHARAT MAHARJAN

C PROGRAMMING

CONTENTS (LH – 5HRS)

8.1 Introduction, Components,

8.2 Function Parameters,

8.3 Library Function vs User Defined Function,

8.4 Different forms of functions,

8.5 Recursion,

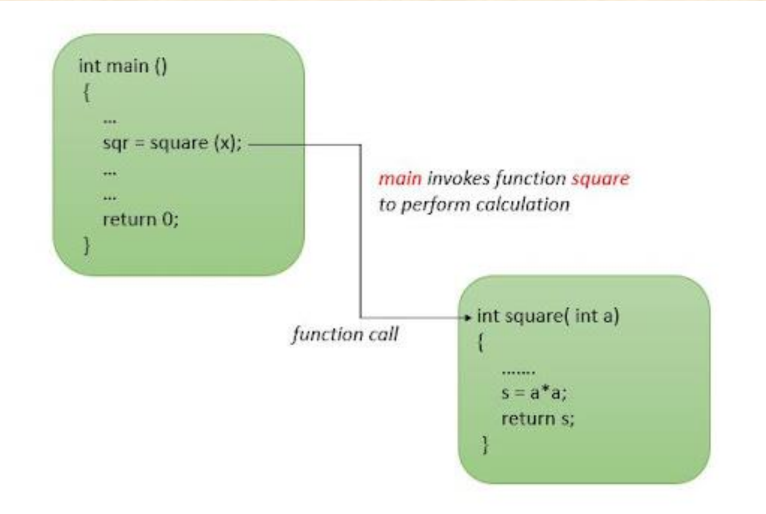
8.6 Passing Array to Function, Passing String to Function,

8.7 Accessing a function (Call by Value and Call by Reference),

8.8 Macros, Storage Class.

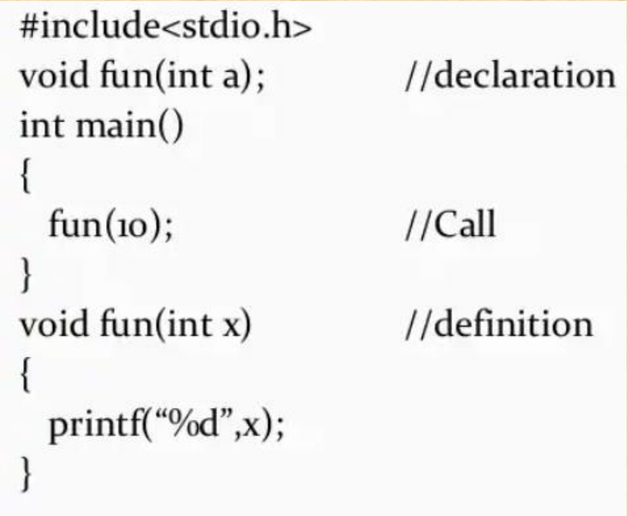
8.1 Introduction, Components

- A function is a block of code that performs a specific task. C allows us to define functions according to our need. These functions are known as **user-defined functions**.
- It can be accessed from any location within a C-program.
- The function main() is always present in every C program which is executed first and other functions are optional.



```
int main ()
{
   ...
   sqr = square (x);
   ...
   ...
   return 0;
}
```

```
int square( int a)
{
   .......
   s = a*a;
   return s;
}
```



```
#include<stdio.h>
void fun(int a);          //declaration
int main()
{
  fun(10);                //Call
}
void fun(int x)           //definition
{
  printf("%d",x);
}
```

Advantages of Function:

- Avoid repetition of codes.
- Increases program readability.
- Divide a complex problem into simpler ones.
- Reduces chances of error.
- Modifying a program becomes easier by using function.

Components of Function

A function has four components. They are:

1. Function Prototype/Declaration
2. Function Definition
3. Function Call
4. The return and void statement

1. Function Prototype or Declaration

- Function declaration is a statement that informs the compiler about
  a. Name of the function
  b. Type of arguments
  c. Number of arguments
  d. Type of Return value
- **Syntax: return_type function_name(type1, type2,……, typen);**

```
e.g.; int add(int a, int b);      //the name of arguments is not mandatory but type is
                                  //mandatory
```

2. Function Definition

- Function definition consists of the body of function. The body consists of block of statements that specify what task is to be performed.
- When a function is called, the control is transferred to the function definition.
- **Syntax: return_type function_name(data_type variable1, data_type variable2,….., data_type variablen)**

```
{
………
statements;
}
```

3. Function Call

- A function can be called by specifying the function’s name, followed by a list of arguments enclosed in parenthesis and separated by commas.
- For example, the function add() can be called with two arguments from main() function as: add(a,b);. These arguments appearing in the function call are called **actual arguments or actual parameters**. In this case, **main() is calling function** and **add() is called function**.
- The general form of the function call statements are:
  - If function has parameters but it does not return value

  function_name(variable1, variable2,……);

  - If function has no arguments and it does not return value

  function_name();

  - If function has parameters and it returns value

  variable_name=function_name(variable1, variable2,…..);

  - If function has no parameters but returns value

  variable_name=function_name();

4. The return and void statements

The return statement serves two purposes:

- It immediately transfers the control back to the calling function (i.e. no statements within the function body after the return statement are executed).
- It returns the value to the calling function.

Syntax: return (expression);

where expression is optional and if present, it must evaluate to a value of the data type specified in the function header for the return_type.

- A function may or may not return any value. If a function does not return a value, the return type in the function definition and declaration is specified as void. For void return_type, return statement is optional.

8.2 Function Parameters

- They are the means for communication between the calling and the called functions.

Two types: actual parameters and formal parameters.

- **Actual parameters** are given in the function call while **formal parameters** are given in the function definition.
- The **name** of formal and actual parameters **need not be same** but **data types and the number of parameters must match.**
- The variables defined within calling function can not be used in the called function and vice versa. The calling function and called functions are two different worlds.
- If we have to supply values from calling function to called function, the values are supplied as actual parameters which are copied to formal arguments of called function.
- Thus, actual parameters act as sender and formal parameters act as receiver for the data values. Actually, the parameters act as communication medium between calling and called function.

8.3 Library Function vs User Defined Function

C Programming Language has **two types of functions**:

1. **Built-in Functions/library Functions**
2. **User-defined Functions**

1. <u>Library Functions</u>

- There functions are already defined in the C compilers. They are used for String handling, I/O operations, etc. These functions are defined in the header file. To use these functions we need to import the specific header files.

E.g.:

- The library function <stdio.h> includes these common functions(there are many other functions too):
  - printf() shows the output in the user’s format.
  - scanf() used to take the user’s input which can be a character, numeric value, string, etc.
- The library function <math.h> includes these common functions(there are many other functions too):
  - pow() finds the power of the given number.
  - sin() finds the sine of the given number.
  - sqrt() finds the square root of the number..
- Apart from <stdio.h> and <math.h> there are many other header files that contain library functions such as <conio.h> that contains clrscr() and getch().

2. User Defined Functions

- User-defined functions are the ones created by the user. The user can program it to perform any desired function.
- It is like customizing the functions that we need in a program. A program can have more than one user-defined functions.
- All the user-defined functions need to be called inside the main() function in order to be executed.
- Any function has 4 building blocks to be declared –
  - Function name
  - Function Parameters
  - Return type
  - Statements to be executed

8.4 Different forms of functions

According to the arguments and return values present in functions, we can categorize the function in four categories:

1. **Function with no arguments and no return value**
2. **Function with no arguments but return value**
3. **Function with arguments but no return type**
4. **Function with arguments and return type**

1. Function with no arguments and no return value

- When a function has no arguments, it does not receive any data from the calling function.
- When a function does not return a value, the calling function does not receive any data from the called function.
- Thus, there is no data transfer between the calling function and called function.

Syntax:

```
void function_name()
{
//body of function
}
```

2. Function with no arguments but return value

- The data cannot be passed from calling function to called function.
- Therefore the required data should be defined within user defined functions as per required.
- Then after manipulating these data, the result is returned to the calling function.

Syntax:

```
return_type function_name()
{
/*body of function
return return_type_data;*/
}
```

3. Function with arguments but no return type

- This type of function has arguments and receives the data from the calling function.
- But after the function completes its task, it does not return any values to the calling function.

Syntax:

```
void function_name(argument_list){
//body of function
}
```

4. Function with arguments and return type

- This type of function has arguments and receives the data from the calling function.
- After the task of the function is complete, it returns the result to the calling function via return statement.
- So, there is data transfer between called function and calling function using return values and arguments.

Syntax:

```
return_type function_name(argument_list){
/*body of function
return return_type_data;
}
```

8.5 Recursion

- A function that calls itself is known as recursive function and this technique is known as recursion in C programming.

```
void recursion() {
   recursion(); /* function calls itself */
}
int main() {
   recursion();
}
```

- The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.
- Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

LAB 1: WAP to compute the factorial of a number using recursion.

```
#include<stdio.h>
int fact(int);//declaring a function
int main(){
    int a;
    printf("Enter a number:");
    scanf("%d",&a);
    printf("The factorial of given number = %d",fact(a));//function call from inside main.
    return 0;
}
int fact(int n){//defining  a function
    if(n==0){
        return 1;//recursion breaker
    }
    else{
        return n*fact(n-1);//recursion 3*fact(2) 3*2*fact(1) 3*2*1*fact(0) 3*2*1*1=6
    }
}
```

8.6 Passing Array to Function, Passing String to Function

1. Passing Array to Function

- Like any other variables, we can also pass entire array to a function.
- An array name can be named as an argument for the prototype declaration and in function header.
- When we call the function no need to subscript or square brackets.
- When we pass array that pass as a call by reference(address) because the array name is address for that array. The array elements themselves are not copied.

2. Passing String to Function

- void Strfun(char *ptr) //(*x=&a)

  Here,

  - void is the return type of the function i.e. it will return nothing.
  - Strfun is the name of the function.
  - char *ptr is the character pointer that will store the base address of the character array (string) which is going to be passed through main() function.

- Function calling statement,

  char buff[20]="Hello Function";

  Strfun(buff);

  Here,

  - buff is the character array (string).

8.7 Accessing a function (Call by Value and Call by Reference),

The arguments in function can be passed in two ways:

a. Pass arguments by value

b. Pas arguments by address or reference or pointers.

a. Function Call by Value (or Pass arguments by value)

- When values of actual arguments are passed to the function as arguments, it is known as function call by value.
- In this call, the value of each actual arguments is copied into corresponding formal argument of the function definition.
- The content of the arguments in the calling function are not altered, even if they are changed in the called function.

LAB 2: WAP to illustrate function call by value. (Note: Example below should be written in exam for call by value question.)

```
#include<stdio.h>
void swap(int, int);
int main(){
        int a=10,b=20;
        printf("Before swapping: \na=%d \t b=%d",a,b);
        swap(a,b);//only values of a and b are copied to x and y. so whatever
           //changes we make in function below with x and y,it doesn’t
//affect the a b variables.
        printf("\nAfter swapping: \na=%d \t b=%d",a,b);
        return 0;
}
void swap(int x, int y){//x=10    y=20only values of x and y are swapped but not a and b.
        int temp;
        temp=x;
        x=y;
        y=temp;
}
```

b. Function Call by Reference (Pass argument by address)

- In call by reference method, the address of actual arguments in calling function are copied into formal arguments of called function.
- Using these addresses we can access the actual arguments and use for further processing.
- Since the references of arguments are used, the values of actual arguments change with change in values of corresponding formal arguments.

LAB 3: WAP to swap two numbers using call by address/reference method

```
#include<stdio.h>
void swap(int*, int*);
int main(){
        int a=10,b=20;
        printf("Before swapping: \na=%d \t b=%d",a,b);
        swap(&a,&b);//passing memory addresses of a and b to swap function.
        printf("\nAfter swapping: \na=%d \t b=%d",a,b);
        return 0;
}
void swap(int *x, int *y){//*x=&a        * is dereference operator which gives value stored
in that memory address.
        int temp;
        temp=*x;        //x=memory address *x=value pointed by that MA. We are
swapping values not addresses.
        *x=*y;
        *y=temp;
}
```

8.8 Macros, Storage Class

- A macro is a fragment of code that is given a name. We can define a macro in C using the **#define preprocessor directive**.
- Here's an example.

```
#define c 299792458  // speed of light
```

- Here, when we use **c** in our program, it is **replaced with 299792458**.

- Storage Classes are used to describe the **features of a variable**. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.
- C language uses **4 storage classes**, namely:

1. Automatic storage class
2. External storage class
3. Static storage class
4. Register storage class

1. **Automatic Storage Class:**

This is the **default storage class** for all the variables declared inside a function or a block. Hence, the **keyword auto** is rarely used while writing programs in C language. Auto variables can be only **accessed within the block/function** they have been declared and not outside them (which defines their scope).

2. **External Storage Class:**

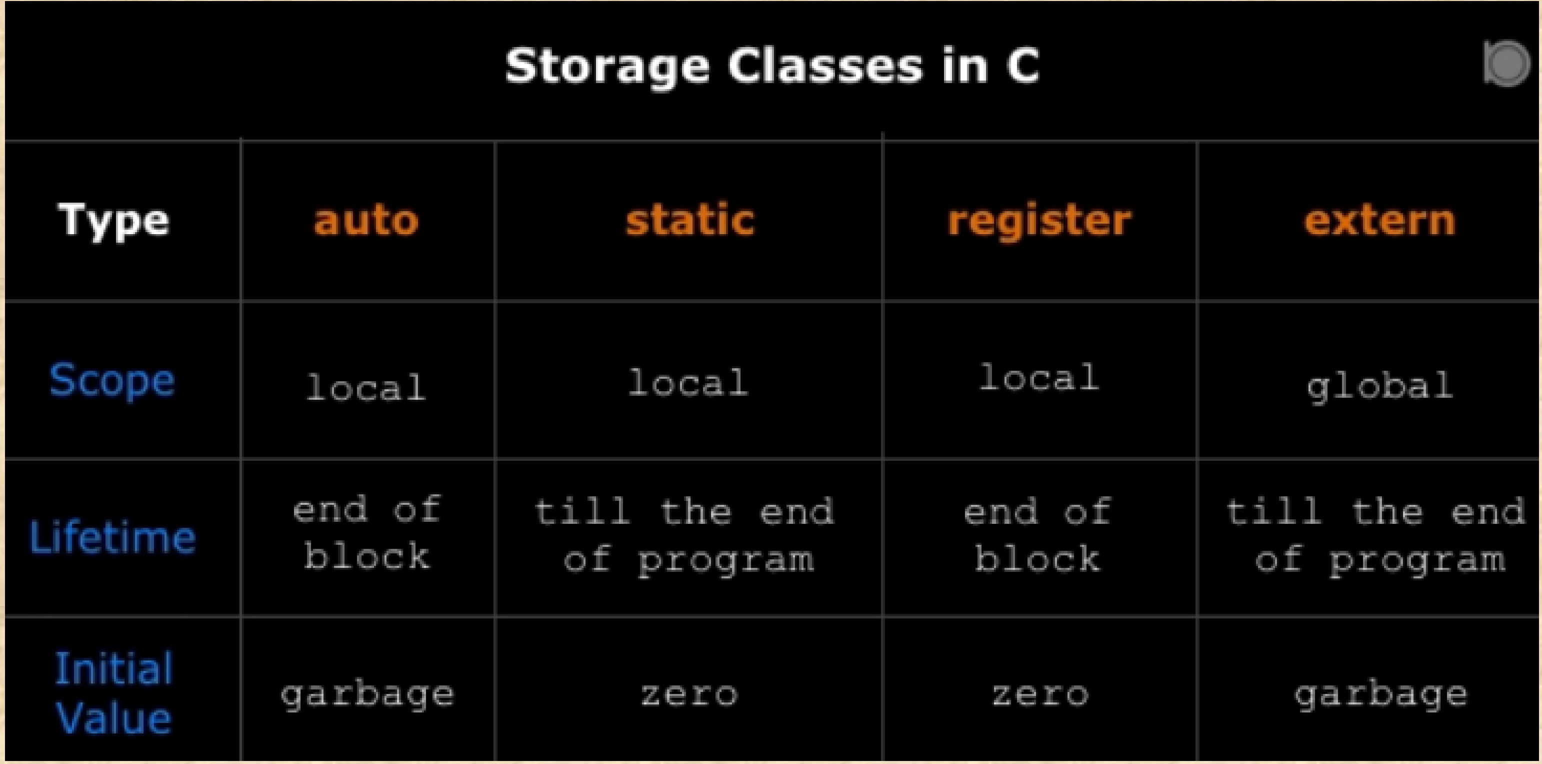
Extern storage class simply tells us that the variable is **defined elsewhere and not within the same block** where it is used. Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So an **extern variable is nothing but a global variable** initialized with a legal value where it is declared in order to be used elsewhere.

3. Static Storage Class:

This storage class is used to declare static variables which are popularly used while writing programs in C language. **Static variables have a property of preserving their value** even after they are out of their scope! Hence, static variables **preserve the value of their last use in their scope.**

4. Register Storage Class:

This storage class **declares register variables** which have the **same functionality as that of the auto variables**. The only difference is that the compiler tries to **store these variables in the register of the microprocessor** if a free register is available. This makes the **use of register variables to be much faster** than that of the variables stored in the memory during the runtime of the program.



Storage Classes in C

Type	auto	static	register	extern
Scope	local	local	local	global
Lifetime	end of block	till the end of program	end of block	till the end of program
Initial Value	garbage	zero	zero	garbage

THANK YOU FOR YOUR ATTENTION