FACULTY OF ENGINEERING AND TECHNOLOGY
UNIVERSITY OF LUCKNOW
LUCKNOW

Computer System and Programming in ‘C’

CS-101/201

Er. Zeeshan Ali Siddiqui
Assistant Professor
Deptt. of C.S.E.

FUNCTIONS

Function-Overview

- In C language, a function is a block of code (a group of statements) that performs *a specific task*. It has a *name* and it is *reusable* i.e. it can be executed from as many different parts in a C Program as required. It also optionally *returns a value* to the calling program

Properties:

- A unique name.
- Independent
- Reusable
- Performs a specific task
- Returns a value (optional)

Function type

- Two types:

1. Library function or system defined function

2. User defined function

Library function/System defined function

- System defined functions can’t be modified, it can only read and can be used.
- These function are supplied with every C compiler.
- Some examples: *printf()*, *scanf()*, *getch()*, *clrscr()*, *etc.*

User defined function

- The user defined functions defined by the user according to its requirement.

- Function skeleton:

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

User defined function: Ingredients

- **Return Type:** *A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In that case, the keyword **void** is the return type.*

- **Function Name:** *This is the actual name of the function. The function name and the parameter list together constitute the **function signature**.*

- **Parameters:** *A parameter is like a **placeholder**. When a function is invoked, we pass a value to the parameter. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.*

- **Function Body:** *The function body contains a collection of statements that **define** what the function does.*

User defined function: Terminology

- **Function declaration:-** Function declaration is also known as function *prototype*. It inform the compiler about three thing, those are *name* of the function, *number* and *type* of argument received by the function and the type of value *returned* by the function. While declaring the name of the argument is optional and the function prototype always terminated by the *semicolon*.
- A function declaration has the following parts:

return_type function_name(parameter list);

Function definition:- Function definition consists of the *whole code* of the function. It tells about what function is *doing* what are its *inputs* and what are its *output* It consists of two parts function *header* and function *body*.

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

User defined function: Example1

```
//* function that returns the greatest of two numbers
int max(int num1, int num2)
{
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Function Call

- When the function get called by the calling function then that is called, function call.
- **Example**:- function(arg1,arg2,arg3);

- **Actual arguments:** The argument that are used inside the function call. These are the original values and copy of these are actually sent to the called function.
- **Example:-** Sumresult = sum(a, b); //actual arguments

- **Formal arguments/dummy arguments:** The arguments which are mentioned in function definition. dummy arguments are used to hold the copy of the values that are sent by the calling function through the function call.
- **Example:-**

```
int sum (int x, int y) //formal/dummy arguments
    {
    return x+y;
    }
```

Note: *Data type and Order number of actual arguments in the function call should be match with the Data type and order number of the formal arguments.*

Keyword return

- It is used to return value to the calling function. It can be used in two way as:

- *return //used to terminate the function without returning any value*

Or

- *return(expression);*

- **Example**:-
- return a;
- return (a);
- return (a*b);
- return (a*a+b);

User defined function: Example2

```c
#include<stdio.h>
int sum(int x, int y); //function declaration
int main()
{
    int result, a=5, b=6;
    result=sum(a,b);//function calling, here a and b are actual arguments
    printf("Sum=%d",result);
    return 0;
}
// function definition start
int sum(int x, int y) //here x and y are formal/dummy arguments
{
    return x+y;
}
// function definition end
```

Category of Function: Based on argument and return type

Category of Function: Based on argument and return type

1. Function with no argument and no return value
2. Function with no argument but return value
3. Function with argument but no return value
4. Function with argument and return value

Function with no argument and no return value

```c
//Function with no argument and no return value
#include<stdio.h>
void printmessage();
int main()
{
    printf("There are months that have 30 days and some have 31 days.\n");
    printmessage();
    return 0;
}
void printmessage()
{
    printf("How many months have 28 days?");
}
```

Function with no argument but return value

```c
//Function with no argument but return value
#include<stdio.h>
int showdata();
int main()
{
    int data;
    data=showdata();
    printf("Data=%d",data);
    return 0;
}
int showdata()
{
    int datavar;
    printf("Please enter an integer value\n");
    scanf("%d",&datavar);
    return datavar;
}
```

Function with argument but no return value

```c
//Function with argument but no return value
#include<stdio.h>
void swap(int a, int b);
int main()
{
    int a=5, b=6;
    printf("Before swapping\n a=%d\tb=%d\n",a,b);
    swap(a,b);
    return 0;
}
void swap(int a, int b)
{
    a=a+b;
    b=a-b;
    a=a-b;
    printf("After swapping\n a=%d\tb=%d",a,b);
}
```

Function with argument and return value

```c
//Function with argument and return value
#include<stdio.h>
int increment(int x);
int main()
{
    int a=2019,rv;
    rv=increment(a);
    printf("First call, a=%d\n",rv);
    rv=increment(rv);
    printf("Second call, a=%d\n",rv);
    return 0;
}
int increment(int x)
{
    ++x;
    return x;
}
```

Any idea?

- Through function, can we send back more than one value?

Methods of passing the arguments to the function

Call by value

- *Copy* of the actual argument is passed to the formal argument and the operation is done on formal/dummy argument.

- It does not affect *content* of the actual argument.

- Changes made to formal argument are local to block of called function so when the control is back to calling function the changes made is *disappear*.

Call by value: Example

```c
//Swapping values of two variables using call by value
#include<stdio.h>
void swap(int a, int b);
int main()
{
    int a=19, b=20;
    printf("Before swapping\n a=%d\tb=%d\n",a,b);
    swap(a,b);
    return 0;
}
void swap(int a, int b)
{
    a=a+b;
    b=a-b;
    a=a-b;
    printf("After swapping\n a=%d\tb=%d",a,b);
}
```

Call by reference

- In call-by-reference, *address* of the variable is passed to the calling function by the called function.

- If data is passed by reference, a pointer to the data is copied instead of the actual variable as is done in a call by value. Because a pointer is copied, if the value at that pointers address is *changed* in the function, the value is also changed in main().

- Called function works on the *original* variables. So, the changes are automatically reflected in the calling function.

Call by reference: Example

```c
//Swapping values of two variables using call by reference
#include<stdio.h>
void swap(int *a, int *b);
int main()
{
    int a=19, b=20;
    printf("Before swapping\n a=%d\tb=%d\n",a,b);
    swap(&a,&b); //address passing
    printf("After swapping\n a=%d\tb=%d",a,b);
    return 0;
}
void swap(int*a, int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

Recursive function

Recursive function

- A function is recursive, if a statement in the body of the function calls *itself*.
- Recursion is the process of defining something in terms of *itself*.
- The speed of a recursive program is slower because of stack *overheads*.
- A recursive function must have *recursive conditions*, *terminating conditions*, and *recursive expressions*.

Recursive function: Example1

```c
//Calculate factorial of a given number using recursive function
int factfun(int x);
int main()
{
    int num, f;
    printf("Please enter a positive integer value of num\n");
    scanf("%d",&num);
    f = factfun (num);
    printf ("\n Factorial of %d =%d\n", num, f);
    return 0;
}
int factfun(int x)
{
    return x==0?1:x*factfun(x-1);
}
```

Recursive function: Example2

```c
//A recursive function to compute the Fibonacci series upto nth place
#include<stdio.h>
int fibfun(int x);
int main()
{
    int n,c,next;
    printf("Please enter number of terms\n");
    scanf("%d",&n);
    for(c=0;c<n;c++)
    {
        next = fibfun(c);
        printf("%d\t",next);
    }
    return 0;
}
int fibfun(int x)
{
    if(x==0||x==1)
    return x;
    else
    return (fibfun(x-1)+fibfun(x-2));
}
```

Function-Advantages

- Top down modular programming.
- Reduced code.
- Easy error detection.
- Reusability.
- ? (*Homework*)

Exercise

- What is the difference between call by value and call by reference? Explain with the help of a program for swapping the two numbers.

- What is the difference between recursion and iteration?

- Write a C program to keep calculate the sum of the digits of a number until the number is a single digit. For example: Input=2018, Process: 2018=>2+0+1+8=11, now 11=> 1+1=2. So Output=2.

- Write recursive functions to -
  - Find the factorial of a given number.
  - Find GCD.
  - Generate the Fibonacci series up to n terms.
  - Find the sum of first n integers.

STORAGE CLASSES

Storage Classes

- Storage class tells us:

1. Storage place (*where variable would be stored*).

2. Default Initial value (*default value of the variable*).

3. Scope (*specifies the part of the program which a variable is accessed*).

4. Life time (*It is the time between the creation and distribution of a variable or how long would variable exists*).

Storage Classes: Types

- There are four types of storage classes:

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

Automatic storage class

- Keyword: **auto**
- Features:

1. Storage place (*Main memory*).

2. Default Initial value (G*arbage value*).

3. Scope (*Local to the block*).

4. Life time (*With in the block in which the variable is defined*).

Note: *The variable without any storage class specifier is called automatic variable.*

Automatic storage class: Examples

```c
#include<stdio.h>
int main()
{
    auto int var=2020;
    printf("var=%d",var);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int var=2020;
    printf("var=%d",var);
    return 0;
}
```

Register storage class

- Keyword: **register**
- Features:

1. Storage place (*CPU registers*).

2. Default Initial value (G*arbage value*).

3. Scope (*Local to the block*).

4. Life time (*With in the block in which the variable is defined*).

Register storage class: Example

```c
#include<stdio.h>
int main()
{
    register int var;
    for(var=0;var<10;var++)
    {
        printf("%d\t",var);
    }
    return 0;
}
```

Static storage class

- Keyword: **static**
- Features:

1. Storage place (*Main memory*).

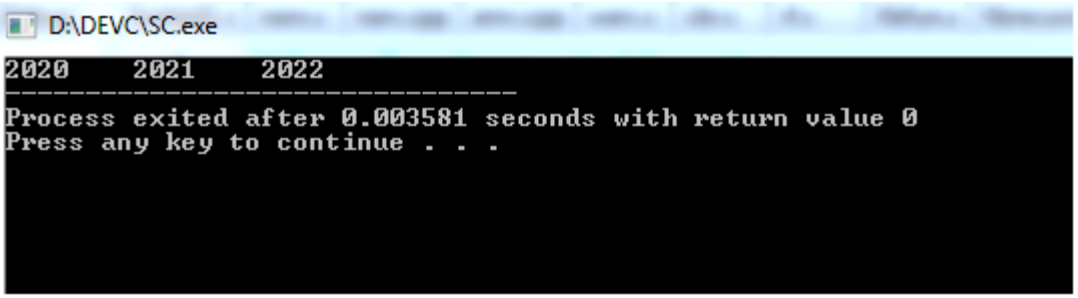
2. Default Initial value (zero).

3. Scope (*Local to the block*).

4. Life time (*value of the variable persists between different function calls*).

Static storage class: Example

```c
#include<stdio.h>
void increment();
int main()
{
    increment();
    increment();
    increment();
    return 0;
}
void increment()
{
    static int x=2020;
    printf("%d\t",x);
    x++;
}
```



Output

External storage class

- Keyword: **extern**
- Features:

1. Storage place (*Main memory*).

2. Default Initial value (zero).

3. Scope (*Global*).

4. Life time (*as long as the program execution doesn't come to an end*).

External storage class: Examples

```c
#include<stdio.h>
void display();
extern int i=2020;//extern keyword is optional
main()
{
    int i=2021;
    printf("%d\n",i);
    display();
}
void display()
{
    printf("%d",i);
}
```

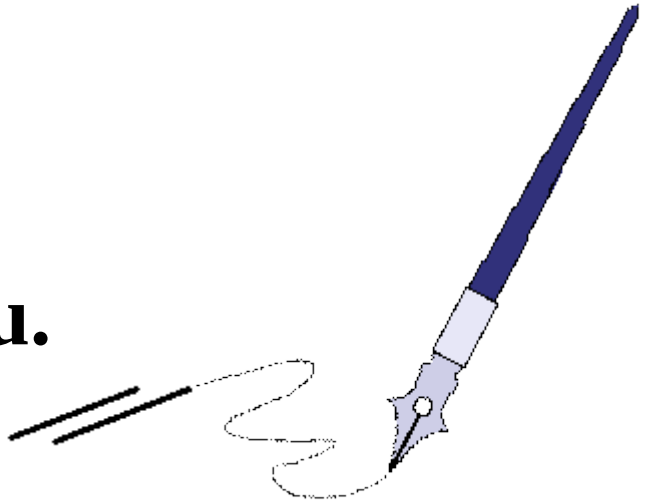
```c
#include<stdio.h>
void display();
int i=2020;
main()
{
    int i=2021;
    printf("%d\n",i);
    display();
}
void display()
{
    printf("%d",i);
}
```

Exercise

- Can we apply address operator on register variable?
- Can we apply storage classes only for integers, characters, pointer type? Explain.
- Give limitations of register storage class.
- Variable stored in register storage class always access faster. How will you reap this benefit?
- Write the output of below program-

```
int main()
{
    static int num = 2020;
    printf("%d\t", num);
    num=num - 505;
    if(num)
    main();
    return 0;
}
```

Thank You.



BTQ

BTQ: Brain Teaser Question

Which word does not belong in the following list:
Stop pop cop mop chop prop shop or crop?