What is the function?

A function is a block of code that performs a specific task and the major goal of function is reuse the code as well as modularity in programming
Suppose example we want to create our own custom library for reuse in future then we can use the functions example we are working on school management project and we have the admission logic means admission logic is necessary to the every school and when we have the multiple school project then we not need to write the same admission logic in every school project we can create our own function and use it in different school project by calling simple function this is called as reusability of code and this is the major goal of function in c language.

What is the purpose of function or why use the function in c language

The major goal of function is

1) **Reuse the code:** means we can define the function only once and can use it more than one time just we need to call the function.
2) **Modularity:** we divide the large program in to sub modules or in functions and integrate them as per our need by calling it and passing parameter to the function and returning value from function or result from function.
3) **Simplicity in application for modification:** we divide the large program in sub module or sub section and if we want to change in future then we not need to trace the complete code of program just we can perform change in specific module which module changes required called as simplicity in application for modification.

Q. Explain the types of function or how many types of function in c language?

Basically there are two types of function in c language

1) **Library function:** those function already provided by c language to us called as library function.
   C provide the library in the form header file just we have to include the header file and use the inbuilt function provided by c language to us
   E.g. scanf, printf, etc
2) **User Defined Function:** user defined function means those function defined by user for creating its own library for future use. Means c provide the feature to developer he can create its own function and can create its

own library and can reuse it in future as per his need just he need to call its created function

Q. Explain the steps to create user defined functions in c language?

If we want to work with a user defined functions in c language we have the three major important steps.

1) **Declaration:** declaration means we specify the function name, its return type and its parameter type means we provide the idea to compiler which kind of function we are planning to define in future his name and its parameter detail.
   Syntax: return type functionname (data type);
   E.g void add (int, int): this statement give the idea to compiler we will define the function add in future in program and if we compiler find it then it will consider it not generate the compile time error
   note: when we define the function after main function then we must be provide the declaration or if we call the one function from another function then we must be provide the declaration and if we define the function before main then we not provide the declaration of function.
2) **Definition:** definition means we write the actual logical body of function means when we want to write the logic in function we need to define the function but when we define the function then definition must be match with a declaration of function.
   In function definition we need to pass data type and variable name as parameter
   Syntax:
   Return type functionname (data type variable name)
   {
   }
   e.g void add (int x, int y)
   {int z=x+y;
   printf ("Addition is %d\n",z);
   }
3) **Calling:** when we want to use the function then we need to call the function when we call the function then function jump on its definition and through the calling we can pass the parameter or required values to the function definition but from calling just we can pass variable or values not need to pass the data type from function calling point as well

as at the time of function calling we can get the result return by function definition at its left hand side.

Syntax of function calling: funtionname (variables);

Example: add (100,200); this statement jump on function definition and pass 100 to x and 200 to y as per our example.

Following Example shows the working of function in c language

```
#include<stdio.h>
#include<conio.h>
void main()
{
   void add(int,int);//declaration
   int a,b;
  printf("Enter the two values\n");
  scanf("%d %d",&a,&b);// 10  20

  add( a , b) ; //declaration
       10 20
  getch();
}
void add(int x,int y)   //defination
{
  int z=x+y;        //here we can write you function logic as per your need

  printf("Addition is %d\n",z);

}
```

jump from calling to defination

pass value of variable a to x and variable b to y

Q. what is the local variable and what is the global variable in c language?

Local Variable:

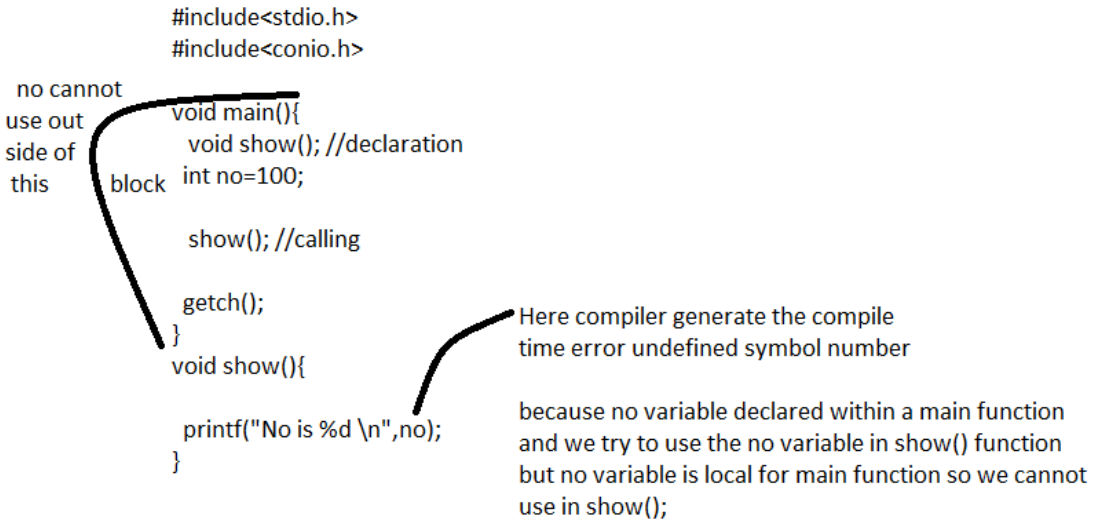
1) Local variable means a variable declared within a function
2) Local variable having default value as garbage in c language
3) Local variable cannot access outside of his block
4) Local variable life present in function when ever function get executed in memory
5) Storage area of local variable is stack

Following Example shows the Local Variable

```
void show(){
 int x;  //local variable for show function
}
void main(){
 int no; //local variable for main function
 getch();
}
```

Note: when we try to access the local variable outside of function definition compiler will generate the error at compile time.

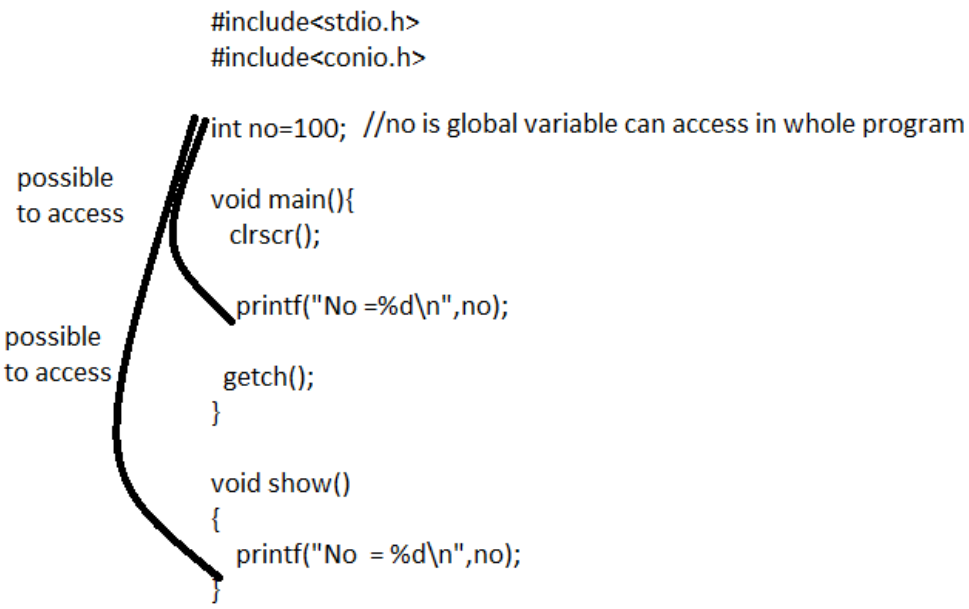
Following example shows the example



Global Variable:

1. Global variable declare outside of function e.g before main function
2. Global variable can access in through the program if we declare it before main
3. The default value of global variable is 0
4. The storage area of global variable is data segment
5. The life of global variable is through the program

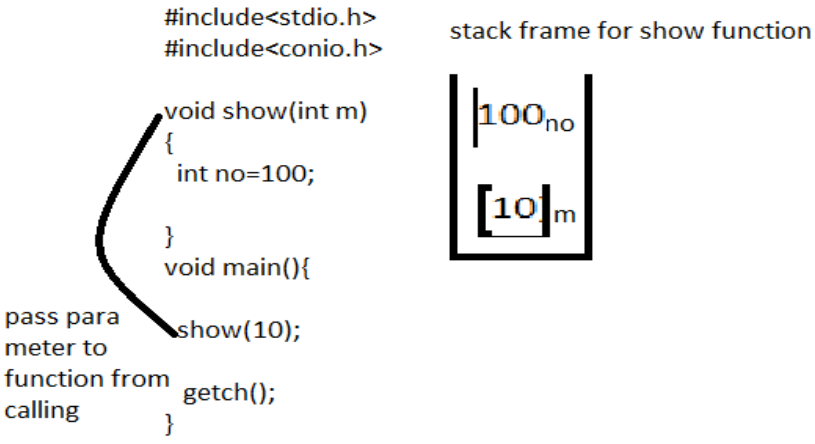
Following Example show the use of Global Variable



Q. which data structure use by function in c language?

Internally function use the stack as data structure in c language means when we call the function then internally in c language for function create the stack frame and allocate all its local variable in that stack and when function close or end its execution then automatically stack frame get deleted and its variable also get deleted and in function contain the local variable so we can say the life of local variable is present when ever function present in memory.

Following Example shows how stack frame created for function

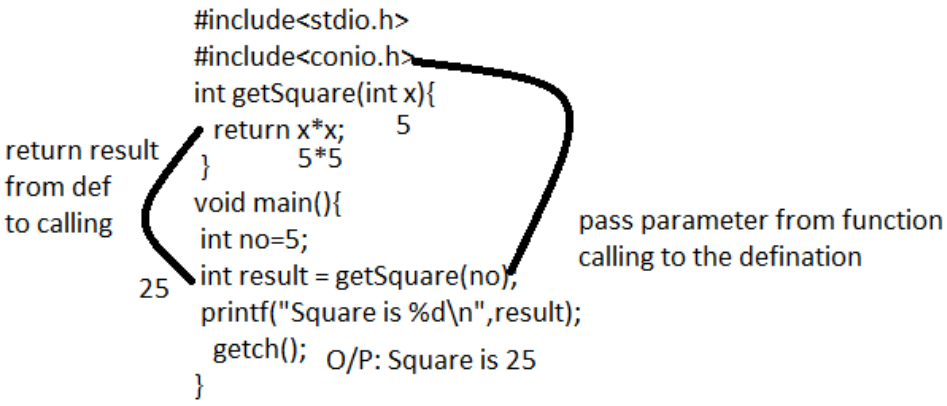


Q. How to return the value from function definition to function calling?

If we want to return value from a function definition to function calling we have the following important points

1. Function return type should not be void means it should be int ,float etc
2. Use the return statement in function definition for return the value from function definition to function calling.

Following Example shows the how to return value from a function?



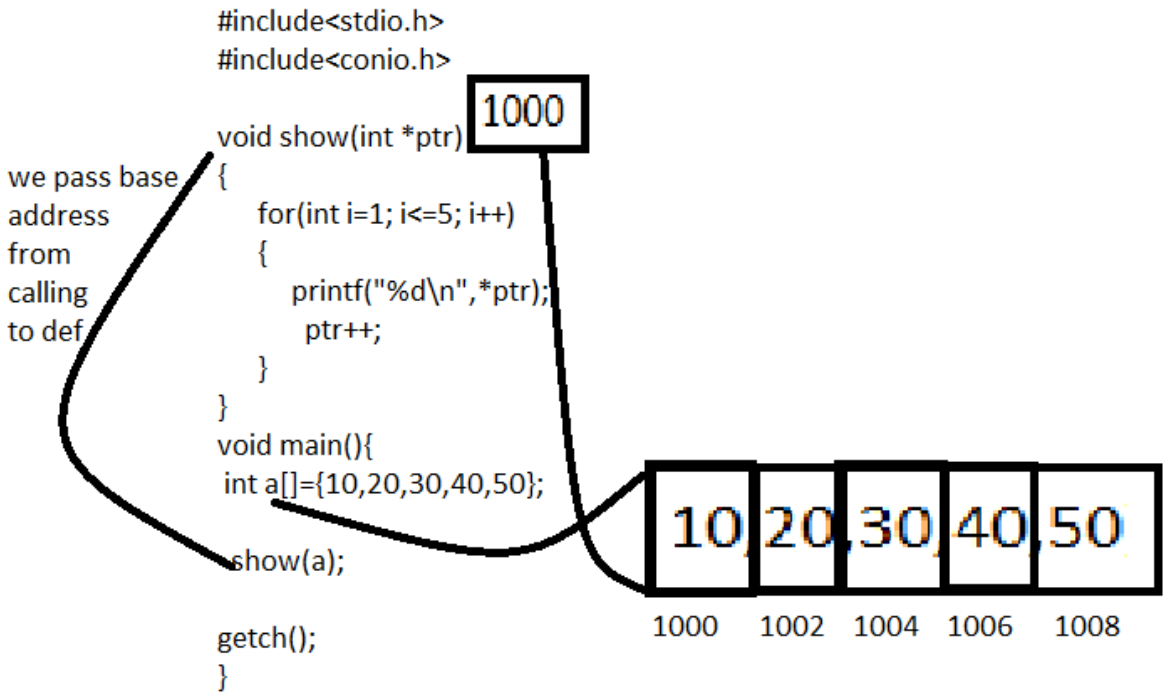
Q. can we return more than one values from function at a time?

No we cannot return more than one statement at a time from function definition to
Function calling and if we want to return multiple values so we can return the array with
Return keyword.

Q. how to pass array as parameter in function?

If we want to pass array as parameter in function we have to pass the base address of array from function calling to the function definition.
Normally we pass array as parameter to the function when we have the large number of parameter in function of same type then we can pass array as parameter to function.

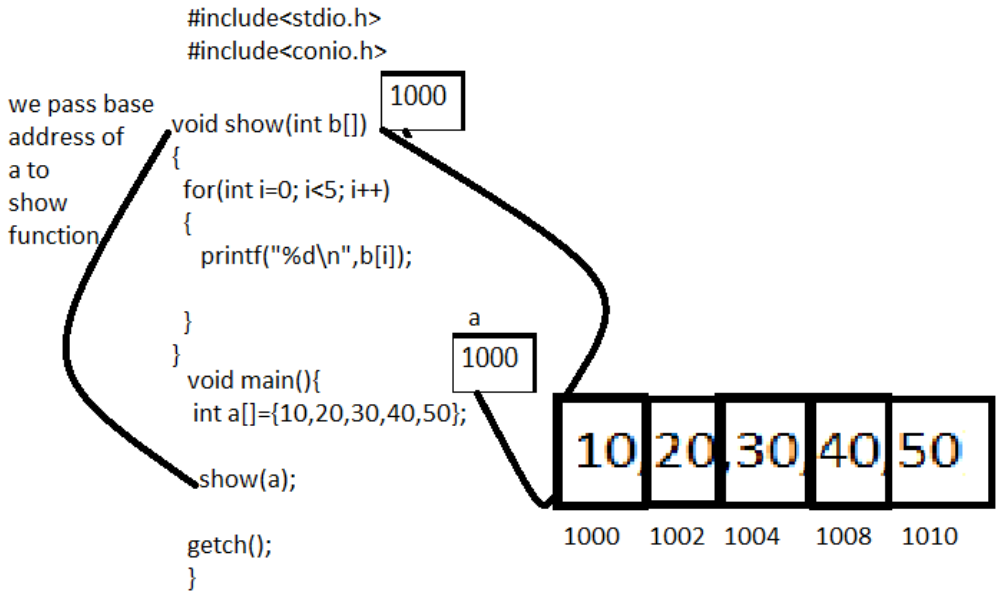
Following Example shows how to pass array as parameter to function definition



If we think about the above code we declare the array in main function and we call the show() function and we pass the base address of array i.e 1000 address to the pointer in show function and we can access the array values in show function using the array address

If we think about the for loop mention in show function ptr pointer points to 1000 address and when i=1 and i<=5 this condition is true when we use the *ptr in printf it will display the 10 values and when we perform ptr++ then pointer move to the next address location i.e 1002 and then i value get incremented by 1 then i<=5 is true and we use the *ptr so we get the 200 and so on.

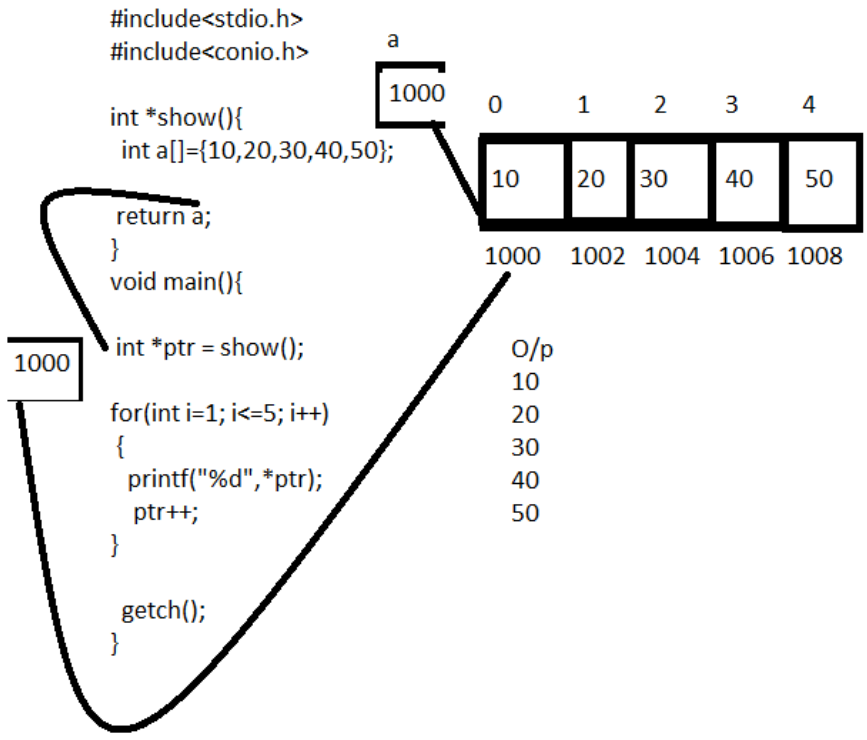
We have the one more way to pass array as parameter to the function given below.



Q. how to return the array from a function?

If we want to return the array from a function we have to return the base address of array from a function definition to function calling and give the * with a return type in function definition

Following Example demonstrate the above concept



If we think about the code we call the main function and return the base address of array a and store the base address at calling point i.e show() in *ptr and when we travel the loop five time and fetch data of array using ptr at function calling point.

Q. what is the function recursion?

Function recursion means a function call itself again and again called as function recursion means if we call the same function from its own definition called as function recursion in c language.
Normally function recursion is used for to perform some repetitive task in programming it is alternative way to loop in programming.

Syntax:

```
returntype functionname(arguments)
{
    if(condition)
    { stop recursion
    }
    else{
       write here your logics
       functionname(arguments);//recursive call
    }
}
```

Following Example show the recursion demo

```
#include<stdio.h>
#include<conio.h>

void show(int x)
{   if(x==0)
     {  printf("END");
     }
    else{
        printf("Good Morning");
       show(x-1); //recursive call
     }
}
void main(){
 clrscr();
  int no=5;

  show(no); //calling

  getch();
}
```

If we think about the above code we define the show function with a recursion technique .
We call the main function first time from main function and 5 value as parameter in show() function the value pass to x in show then in show function we have the condition if(x==0) is false first time because first time x is 5 so our condition like as if(5==0) this is false and it will execute the else block and display the good morning message using printf("good morning") statement and after that we have the recursive call show(x-1) means our function jump on function definition and update the value of x in function definition so the update value of x is 4 and our condition check if(x==0) i.e if(4==0) this is false and execute the else block and again print good morning using printf("good morning") statement and perform recursive call and this process get executed when x is 0
When the value of x is zero so if(x==0) means if(0==0) is true and terminate the recursive call.

Q. what is the diff between loop and recursion?

Time Complexity: Finding the Time complexity of Recursion is more difficult than that of Iteration.

- **Recursion**: Time complexity of recursion can be found by finding the value of the nth recursive call in terms of the previous calls. Thus, finding the

destination case in terms of the base case, and solving in terms of the base case gives us an idea of the time complexity of recursive equations

- **Iteration**: Time complexity of iteration can be found by finding the number of cycles being repeated inside the loop.

Usage: Usage of either of these techniques is a trade-off between time complexity and size of code. If time complexity is the point of focus, and number of recursive calls would be large, it is better to use iteration. However, if time complexity is not an issue and shortness of code is, recursion would be the way to go.

- **Recursion:** Recursion involves calling the same function again, and hence, has a very small length of code. However, as we saw in the analysis, the time complexity of recursion can get to be exponential when there are a considerable number of recursive calls. Hence, usage of recursion is advantageous in shorter code, but higher time complexity.

- **Iteration:** Iteration is repetition of a block of code. This involves a larger size of code, but the time complexity is generally lesser than it is for recursion.

Overhead: Recursion has a large amount of Overhead as compared to Iteration**.**

- **Recursion**: Recursion has the overhead of repeated function calls, that is due to repetitive calling of the same function, the time complexity of the code increases manyfold.
- **Iteration:** Iteration does not involve any such overhead.

Infinite Repetition: Infinite Repetition in recursion can lead to CPU crash but in iteration, it will stop when memory is exhausted.

- **Recursion:** In Recursion, Infinite recursive calls may occur due to some mistake in specifying the base condition, which on never becoming false, keeps calling the function, which may lead to system CPU crash.

- **Iteration**: Infinite iteration due to mistake in iterator assignment or increment, or in the terminating condition, will lead to infinite loops, which

may or may not lead to system errors, but will surely stop program execution any further.

Q. Explain Call by Value and Call by Reference?

Functions can be invoked in two ways: Call by Value or Call by Reference. These two ways are generally differentiated by the type of values passed to them as parameters.

The parameters passed to function are called *actual parameters* whereas the parameters received by function are called *formal parameters*.

Call by Value: In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.

Call by Reference: Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.

In the case of call by reference we pass the address of actual argument from function calling to the function definition means if we perform any change in function definition it will reflect on function calling

Call By Value	Call By Reference
While calling a function, we pass values of variables to it. Such functions are known as "Call By Values".	While calling a function, instead of passing the values of variables, we pass address of variables(location of variables) to the function known as "Call By References.
In this method, the value of each variable in calling function is copied into corresponding dummy variables of the called function.	In this method, the address of actual variables in the calling function are copied into the dummy variables of the called function.

Call By Value

With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function.

```
// C program to illustrate
// call by value

#include

// Function Prototype
void swapx(int x, int y);

// Main function
int main()
{
int a = 10, b = 20;

// Pass by Values
swapx(a, b);

printf("a=%d b=%d\n", a, b);
```

Call By Reference

With this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them.

```
// C program to illustrate
// Call by Reference

#include

// Function Prototype
void swapx(int*, int*);

// Main function
int main()
{
int a = 10, b = 20;

// Pass reference
swapx(&a, &b);

printf("a=%d b=%d\n", a, b);
```

Call By Value

```
return 0;
}

// Swap functions that swaps
// two values
void swapx(int x, int y)
{
int t;

t = x;
x = y;
y = t;

printf("x=%d y=%d\n", x, y);
}
```

Output:
x=20 y=10
a=10 b=20

Thus actual values of a and b remain unchanged even after exchanging the values of x and y.

Call By Reference

```
return 0;
}

// Function to swap two variables
// by references
void swapx(int* x, int* y)
{
int t;

t = *x;
*x = *y;
*y = t;

printf("x=%d y=%d\n", *x, *y);
}
```

Output:
x=20 y=10
a=20 b=10

Thus actual values of a and b get changed after exchanging values of x and y.

Call By Value	Call By Reference
In call by values we cannot alter the values of actual variables through function calls.	In call by reference we can alter the values of variables through function calls.
Values of variables are passes by Simple technique.	Pointer variables are necessary to define to store the address values of variables.

Q. Should a function contain a return statement if it does not return a value?

In C, void functions (those that do not return a value to the calling function) are not required to include a return statement. Therefore, it is not necessary to include a return statement in your functions declared as being void. In some cases, your function might trigger some critical error, and an immediate exit from the function might be necessary. In this case, it is perfectly acceptable to use a return statement to bypass the rest of the function's code. However, keep in mind that it is not considered good programming practice to litter your functions with return statements-generally; you should keep your function's exit point as focused and clean as possible.

Q. Is it possible to execute code even after the program exits the main() function?

The standard C library provides a function named atexit() that can be used to perform "cleanup" operations when your program terminates. You can set up a set of functions you want to perform automatically when your program exits by passing function pointers to the atexit() function. Here's an example of a program that uses the atexit() function:

```
#include <stdio.h>
#include <stdlib.h>
void close_files(void);
void print_registration_message(void);
```

```
int main(int, char**);
int main(int argc, char** argv)
{  atexit(print_registration_message);
   atexit(close_files);
   while (rec_count < max_records)
   {
      process_one_record();
   }
   exit(0);
}
```

This example program uses the atexit() function to signify that the close_files() function and the print_registration_message() function need to be called automatically when the program exits. When the main() function ends, these two functions will be called to close the files and print the registration message. There are two things that should be noted regarding the atexit() function. First, the functions you specify to execute at program termination must be declared as void functions that take no parameters. Second, the functions you designate with the atexit() function are stacked in the order in which they are called with atexit(), and therefore they are executed in a last-in, first-out (LIFO) method. Keep this information in mind when using the atexit() function. In the preceding example, the atexit() function is stacked as shown here:

```
atexit(print_registration_message);
atexit(close_files);
```

Because the LIFO method is used, the close_files() function will be called first, and then the print_registration_message() function will be called. The atexit() function can come in handy when you want to ensure that certain functions (such as closing your program's data files) are performed before your program terminates.

Q. **Is using exit() the same as using return?**

No. The exit() function is used to exit your program and return control to the operating system. The return statement is used to return from a function and return control to the calling function. If you issue a return from the main() function, you are essentially returning control to the calling function, which is the operating system. In this case, the return

statement and exit() function are similar. Here is an example of a program that uses the exit() function and return statement:

```
#include <stdio.h>
#include <stdlib.h>
int main(int, char**);
int do_processing(void);
int do_something_daring();
int main(int argc, char** argv)
{
    int ret_code;
    if (argc < 3)
    {
        printf("Wrong number of arguments used!\n");
        /* return 1 to the operating system */
        exit(1);
    }
    ret_code = do_processing();
    ...
    /* return 0 to the operating system */
    exit(0);
}
int do_processing(void)
{
    int rc;
    rc = do_something_daring();
    if (rc == ERROR)
    {
        printf("Something fishy is going on around here..."\n);
        /* return rc to the operating system */
        exit(rc);
    }
    /* return 0 to the calling function */
    return 0;
}
```

In the main() function, the program is exited if the argument count (argc) is less than 3. The statement exit(1); tells the program to exit and return the

number 1 to the operating system. The operating system can then decide what to do base on the return value of the program. For instance, many DOS batch files check the environment variable named ERRORLEVEL for the return value of executable programs.

MCQ QUESTION ON FUNCTION

Q. what will be the output of given code?

```
#include <stdio.h>
int main()
{
int i = 5;
printf("%d %d %d", i++, i++, i++); return 0;
}
```

Q. what will be the output of given code?

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

Q. what will be the output of given code?

```
#include <stdio.h>
int main()
{  int (*ptr)(int ) = fun; (*ptr)(3);
    return 0;
}
int fun(int n)
{  for(;n > 0; n--)
   printf("GIRI'S TECH HUB PVT.LTD ");
   return 0;
}
```

Q.what will be the output of given code?

```
#include<stdio.h>
void dynamic(int s, ...)
{  printf("%d ", s);
}
int main()
{  dynamic(2, 4, 6, 8);
   dynamic(3, 6, 9);
   return 0;
}
```

Q. what will be the output of given code ?

```
#include <stdio.h>
int main()
{  void demo();
   void (*fun)();
   fun = demo;
    (*fun)();
    fun();
  return 0;
}
void demo()
{  printf("GIRI'S TECH HUB PVT.LTD");
```

Q. what will be the output of given code?

```
void foo(int n, int sum)
{    int k = 0, j = 0;
     if (n == 0)
        return; k = n % 10;
     j = n / 10;
     sum = sum + k; foo (j, sum); printf ("%d,", k);
}
int main ()
{  int a = 2048, sum = 0; foo (a, sum);
   printf ("%dn", sum);
getchar();
}
```

Q. what will be the output of given code ?

```
 double foo (double); /* Line 1 */ int main()
{   double da, db;
// input da
  db = foo(da);
}
double foo(double a)
{
     return a;
}
```

Q. what will be the output of given code ?

```
void swap (int a, int b)
{
    int temp; temp = a; a = b;
     b = temp;
}
```

Q. what will be the output of given code?

```
int incr(int i)
{
     static int count = 0;
     count = count + i;
      return (count);
}
main()
{
      int i,j;
     for (i = 0; i <=4; i++) {
        j = incr(i);
      printf("%d",j);
      }
   }
```

Q. what will be the output of given code ?

```
 void f1 (int a, int b)
{
   int c;
    c=a;
    a=b;
```

```
    b=c;
}
void f2 (int *a, int *b)
{
    int c;
    c=*a;
  *a=*b;
   *b=c;
}
int main()
{
   int a=4, b=5, c=6; f1(a, b);
  f2(&b, &c);
   printf ("%d", c-a-b); return 0;
}
```

Q. what will be the output of given code?

```
#include "stdio.h"
 int main()
{
   int a = 10; int b = 15;
    printf("=%d",(a+1),(b=a+2));
      printf(" %d=",b);
    return 0;
}
```

Q. what will be the output of given code?

```
#include "stdio.h"
 int main()
{
    int a = 10;
    printf("=%d %d=",(a+1));
    return 0;
}
```

Q. what will be the output of given code?

```
#include "stdio.h"
int main()
```

```
{
   printf("abcde"); main();
   return 0;
}
```

Q. what will be the output of given code?

```
  #include "stdio.h"
int foo(int a)
{
   printf("%d",a); return 0;
}

int main() {
   foo;
   return 0;
}
```

Q. what will be the output of given code ?

```
#include <stdio.h>
int funcf (int x);
int funcg (int y);
main()
{
   int x = 5, y = 10, count;
 for (count = 1; count <= 2; ++count)
 {
    y += funcf(x) + funcg(x); printf ("%d ", y);
  }
}

funcf(int x)
{   int y;
    y = funcg(x); return (y);
}

funcg(int x)
{  static int y = 10; y += 1;
   return (y+x);
}
```

Q. what will be the output of given code?

f (x1, x2, ..., xn) = x1'f(x1, x2, ..., xn) + x1f(x1, x2, ..., xn) B
f (x1, x2, ..., xn) = x2f(x1, x2, …, xn) + x2'f(x1, x2, …,xn) C
f (x1, x2, ..., xn) = xn'f(x1, x2, …, 0) + xnf(x1, x2, …,1) D
f (x1, x2, ..., xn) = f(0, x2, …, xn) + f(1, x2, .., xn)

Q. what will be the output of given code ?

```
#include <stdio.h>
 int funcf (int x);
int funcg (int y); main()
{  int x = 5, y = 10, count;
  for (count = 1; count <= 2; ++count)
   {
      y += funcf(x) + funcg(x); printf ("%d ", y);
   }
}
funcf(int x)
{   int y;
    y = funcg(x); return (y);
}
funcg(int x)
{
   static int y = 10; y += 1;
  return (y+x);
}
```

Q. what will be the output of given code ?

```
int f(int *p, int n)
{
     if (n <= 1)
       return 0;
     else
      return max(f(p+1,n-1),p[0]-p[1]);
}
int main()
{
```

```
    int a[] = {3,5,2,6,4};
    printf("%d", f(a,5));
}
```

Q. what will be the output of given code?

```
#include<stdio.h>
int f(int n, int k)
{
    if (n == 0)
      return 0;
    else if (n % 2)
      return f(n/2, 2*k) + k; else return f(n/2, 2*k) - k;
}
int main ()
{
    printf("%d", f(20, 1));
    return 0;
}
```

Q. what will be the output of given code ?

```
int fun()
{
     static int num = 16;
     return num--;
}
int main()
{
     for(fun(); fun(); fun())
     printf("%d ", fun()); return 0;
}
```

Q. what will be the output of given code?

```
 Count (x, y)
 {
    if (y !=1 )
    {
       if (x !=1)
        {
            print("*");
              Count (x/2, y);
        }
     else
      {
          y=y-1;
         Count (1024, y);
      }
   }
}
```

Q. what will be the output of given code?

```
 int fun(x: integer)
{

      If x > 100 then
        fun = x – 10; else
        fun = fun(fun(x + 11));
}
```

Q. what will be the output of given code?

```
int func(int num) {
  int count = 0;
   while(num) {
     count++;
     num >>= 1;
 }
  return(count) ;
}
```

Q. what will be the output of given code?

```
#include void main()
{   double pi = 3.1415926535; int a = 1;
     int i;
for(i=0; i < 3; i++)
  if(a = cos(pi * i/2) )
     printf("%d ",1);
      else
        printf("%d ", 0);
}
```

Q. what will be the output of given code?

```
int f(int n)
{
    static int i = 1;
     if(n >= 5)
      return n;
       n = n+i;
      i++;
     return f(n);
}
```

Q. what will be the output of given code?

```
main ()
{
    int x = 2, y = 5;
     if(x < y)
      return (x = x + y);
        else
          printf ("z1");
```

```
        printf("z2");
    }
```

Q. what will be the output of given code?

```
int main()
{
   int x = 1;
    printf ("%d", (*char(char *)&x)) ;
}
```

Q. what will be the output of given code?

```
#include main()
{
   float sum = 0.0, j =1.0, i = 2.0;
  while(i/j > 0.001) {
   j = j + 1;
   sum = sum + i/j;
  printf ( "%fn", sum );
  }
}
```

Q. what will be the output of given code?

```
void swap (int a, int b)
{  int temp; temp = a; a = b;
    b = temp;
}
int main( )
{  int p = 0, q = 1; swap (p, q);
}
```

Q. what will be the output of given code?

```
 void swap (int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
int main( )
{
   int p = 0, q = 1;
    swap (p, q);
```

```
}
```

Q. what will be the output of given code?

```
void swap (int * a, int * b)
{   int * temp;
   temp = a;
   a = b;
   b = temp;
}
int main( )
{ int p = 0, q = 1;
     swap (&p, &q);
}
```