**9.14: Define Recursion.**

**9.15: Illustrate recursion with sample programs.**

**RECURSION**: When a called function in turn calls another function a process of ‘chaining’ occurs. Recursion is a special case of this process, **where a function calls itself**. A very simple example of recursion is presented below:

main ( )

{

printf (“This is an example of recursion\n”);

main ( );

}

When executed, this program will produce an output something like this:

This is an example of recursion

This is an example of recursion

This is an example of recursion

This is an ex

**……..**

**………..**

Execution is terminated abruptly; otherwise the execution will continue indefinitely.

For example,

Factorial of 4 =4\*3\*2\*1=24

A function to evaluate factorial of n is as follows:

factorial (int n)

{

int fact;

if (n==0||n==1)

return(1);

else

fact=n\*factorial (n-1);

return (fact);

}

Recursive functions can be effectively used to **solve problems where solution is expressed in terms of successively applying the same solutions to subsets of the problem**. When we write recursive functions, we must have if statement somewhere to force the function to return without the recursive call being executed. Otherwise, the function will never return.

|  |  |
| --- | --- |
| **/\*Write a c program to find the factorial of a given number\*/**  #include<stdio.h>  int fact(int);  void main()  {  int n,result;  printf("enter a number\n");  scanf("%d",&n);  result=fact(n);  printf("%d",result);  }  int fact(int n)  {  if(n==0||n==1)  return 1;  else  return(n\*fact(n-1));  } | **/\*Write a c program to find the nth fibnacci term\*/**  #include<stdio.h>  int fib(int);  void main()  {  int n,result;  printf("enter a number\n");  scanf("%d",&n);  result=fib(n);  printf("%d",result);  }  int fib(int n)  {  if(n==1||n==2)  return 1;  else  return(fib(n-1)+fib(n-2));  } |

|  |  |
| --- | --- |
| **/\*Write a c program to find the x power y\*/**  #include<stdio.h>  int power(int,int);  void main()  {  int x,y,result;  printf("enter x,y values");  scanf("%d%d",&x,&y);  result=power(x,y);  printf("%d",result);  }  int power(int x,int y)  {  if(y==0)  return 1;  else  return(x\*power(x,y-1));  } | **/\*Write a c program to find the gcd of two numbers\*/**  #include<stdio.h>  int gcd(int,int);  void main()  {  int x,y,result;  printf("enter x,y values");  scanf("%d%d",&x,&y);  result=gcd(x,y);  printf("%d",result);  }  int gcd(int x,int y)  {  if(y==0)  return x;  else  return gcd(y,x%y);  } |

|  |
| --- |
|  |

**9.16: illustrate passing arrays to functions with sample programs**.

**Passing One dimensional arrays**: Like the values of simple variables, it is also possible to pass the values of an array to a function. **To pass a one-dimensional an array to a called function, it is sufficient to list the name of the array, without any subscripts, and the size of the array as arguments.**

for example, the call

**largest(a, n)**

Will pass the whole array **a** to the called function. The called function expecting this call must be appropriately defined. The largest function header might look like:

**float largest(float array[ ], int size)**

The function **largest** is defined to take two arguments, the array name and the size of the array to specify the number of elements in the array is made as follows:

**float array[ ];**

* The pair of brackets informs the compiler that the argument **array** is an array of numbers. It is not necessary to specify the size of the **array** here.
* In c the name of the array represents the address of its first element.
* By passing the array name, we are, in fact, passing the address of the array to the called function. The array in the called function now refers to the same array stored in the memory. Therefore, **any** **changes in the array in the called function will be reflected in the original array**.
* Passing address of parameters to the functions is referred to as pass by address (or pass by pointers).
* Note that **we cannot pass a whole array by value** as we did in the case of ordinary variables.

**Three Rules to pass an Array to a Function**

1. The function definition must be called by passing only the name of the array.
2. In the function definition, the formal parameter must be an array type; the size of the array does not need to be specified.
3. The function prototype must show that the arguments are an array and size of an array.

**Two –dimensional Arrays:** Like simple arrays, we can also pass multi-dimensional arrays to functions. The approach is similar to the one we did with one-dimensional arrays. The rules are simple.

1. The function must be called by passing only the array name.

2. In the function definition, we must indicate that the array has two-dimensions by including two sets of brackets.

3. The size of the second dimension must be specified.

4. The prototype declaration should be similar to the function header.

The function given below calculates the average of the values in a two-dimensional matrix.

double average(int x[][n], int m, int n)

{

int i, j;

double sum = 0.0;

for(i=0;i<m;i++)

for(j=1;j<n;j++)

sum += x[i][j];

return(sum/(m\*n));

}

The function can be used in a main function as illustrated below:

main()

{

int m=3, n=2;

double average(int [] [n], int, int);

double mean;

int matrix [m][n]= { {1,2},

{3,4},

{5,6}

};

mean = average(matrix, m, n);

printf(“mean of matrix %lf”, mean);

}

**9.17: Discuss the scope, visibility and life-time of variables in functions**:

**Scope**: the **region of a program in which a variable is available for use (active)**

Rules of Use:

* The scope of a global variable is the entire program file.
* The scope of a local variable begins at point of declaration and ends at the end of the block or function in which it is declared.
* The scope of a formal function argument is its own function.

**Life-time**: the life time of a variable is the **period during which a variable retains a given value during execution (alive)**

* The life time of an auto variable declared in main **is the entire program execution**, though its scope is within the main function.
* The life time of a global variable is entire program.
* The life time of a static variable is entire program.

**Visibility**: visibility of a variable is the programs **ability to access a variable from the memory**.

**9.18: Differentiate Local and External variables**:

Automatic variables (or) local (or) internal variables:

* Automatic variables are **declared inside a function** in which they are to be utilized.
* They are **created when the function is called and destroyed automatically when the function is exited**, hence the name automatic.
* Automatic variables are therefore private or local to the function in which they are declared. Because of this property, Automatic variables are also referred to as local or internal variables.
* A variable declared inside a function without storage class specification is, **by default, an automatic variable**. e.g:

|  |  |
| --- | --- |
| main()  {  int number;/\*by default automatic\*/  ……………..  …………….  } | main()  {  auto int number; /\*external declaration of automatic variable\*/  ……………..  …………….  } |

Important feature of automatic variable: their value cannot be changed by in some other function in the program.

Sample program to illustrate working of local variables:

|  |
| --- |
| void fun1();  void fun2();  main()  {  int m=1000;  fun2();  printf("%d\n",m); /\*third output\*/  }  void fun1()  {  int m=10;  printf("%d\n",m); /\*first output\*/  }  void fun2()  {  int m=100;  fun1();  printf("%d\n",m); /\*second output\*/  } |



**External variables (or) Global variables**:

* Variables that are **both alive and active throughout the entire program** are known as external variables. They are also known as global variables.
* External variables are **declared outside a function**. For example the External declaration of integer **number** and float **length** might appear as

|  |
| --- |
| int number;  float length =7.5;  main()  {  ……….  ……….  ……….  }  function1()  {  -------  -------  }  function2()  {  -------  -------  } |

Here, the variable integer **number** and float **length** are for use in all the three functions.

* In case a local variable and a global variable have the same name, then the **local variable has precedence over the global one** in the function where it is declared. consider the following eg:

|  |
| --- |
| int count;  main()  {  count=10;  ……….  ……….  }  function1()  {  int count=0;  -------  -------  count=count+1;  }  when the function references the variable **count** , it will be referencing only its local variable , not the global one. |

Note:

1. Unlike local variables, global variables can be accessed by any function in the program.

Sample program to illustrate working of local variables:

|  |
| --- |
| int fun1();  int fun2();  int fun3();  int x;  main()  {  clrscr();  x=10;  printf("x=%d\n",x);  printf("x=%d\n",fun1());  printf("x=%d\n",fun2());  printf("x=%d\n",fun3());  }  int fun1()  {  x=x+10;  return x;  }  int fun2()  {  int x=1; /\*local x\*/  return(x);  }  int fun3()  {  x=x+10; /\*global x\*/  } |



Differences between local variable and global variables:

|  |  |
| --- | --- |
| **local variable** | **global variables** |
| 1. Automatic variables are **declared inside a function** in which they are to be utilized. They are **created when the function is called and destroyed automatically when the function is exited**, hence the name automatic. 2. The scope of a local variable begins at point of declaration and ends at the end of the block or function in which it is declared. 3. The life time of an auto variable declared in a function is until end of function or block. 4. Visibility: only in that function or block. 5. Their value cannot be changed by in some other function in the program. 6. E.g.: program | 1. External variables are **declared outside a function.** 2. The scope of a global variable is the entire program file. 3. The life time of a global variable is entire program. 4. Visibility: entire file. 5. global variables can be accessed by any function in the program. 6. E.g.: program |

**Static variable:** the value of static variables persists until the end of the program. A variable can be declared static using the keyword **static** like

Static int x;

Static float y;

A static variable may be 🡪internal type

🡪external type.

Note: the internal static variable can be used to retain values between function calls. For e.g:

|  |  |
| --- | --- |
| void stat();  main()  {  int i;  for(i=1;i<=3;i++)  stat();  }  void stat()  {  static int x=0;  x=x+1;  printf("x=%d\n",x);  } |  |

A static variable is initialized only once, when the program is compiled. It is never initialized again.

🡪during first call to stat x is incremented to 1.

🡪because, x is static the value persists and therefore , the next call adds another 1 to x and gives 2.

🡪the value of x becomes 3 when the third call is made.

**Register variables:**

**🡪**since a register access is much faster than a memory access, keeping the frequently accessed variables(e.g: loop variables) in the register will lead to faster execution of programs. This is done as follows:

**register int count;**

**🡪** The scope of the register variable is only in that function or block.

🡪 The lifetime of the register variable is until end of function or block.

**9.19: define Global variables**.

**9.20: discuss passing the global variables as parameters using sample programs**.

Since all functions in a program can access global variables, they can be used for passing values between the functions.

However, using global variables as parameters for passing values poses certain problems. They are:

1. The values of global variables which are sent to the called function may changed by the called function.
2. “functions are supposed to be independent and isolated modules” this character is lost ,if they use global variables.
3. A function that uses global variables suffers from re-usability.

To make the parameters completely global one should follow the steps given below:

Step 1: declare all the variables as global variables before main () function.

Step 2: function prototypes must be with return type as a void and without any parameters.

Step 3: function definition must be with return type as a void and without any parameters.

|  |
| --- |
| void area\_circle();  void circum\_circle();  float pi=3.14; /\*global variables\*/  float area,circum;  float r;  int main()  {  clrscr();  printf("enter the radius of the circle:");  scanf("%f",&r);  area\_circle();  circum\_circle();  printf("area of circle%f\n",area);  printf("circumference of a circle is %f \n",circum);  getch();  }  void area\_circle() /\*function definition with void and without parameters\*/  {  area=pi\*r\*r;  }  void circum\_circle() /\*function definition with void and without parameters\*/  {  circum=2.0\*pi\*r;  } |