Q. P. Code : 23993

Q1. a) Define union. Compare Structure and Union.
Ans.

Union :- A **union** is a special data type available in **C** that allows storing different data types in the same memory location.

	Structure	Union
Keyword	The keyword **'struct'** is used to define a structure.	The keyword **'union'** is used to define a union.
Size	When a variable associated with a structure, the compiler allocates the memory for each member. The size of structure is greater than or equal to the sum of sizes of its members.	When a variable associated with a union, the compiler allocates the memory by considering the size of the largest memory. Hence, The size of union is equal to the size of largest members.
Memory	Each member within a structure is assigned and unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initializing Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Q1. b) What is an error ? Explain different types of errors occurred in program.
Ans.

Error :-

While writing c programs, errors also known as bugs in the world of programming may occur unwillingly which may prevent the program to compile and run correctly as per the expectation of the programmer.

Types of errors :-

Basically there are three types of errors in c programming:

1. Runtime Errors :

C runtime errors are those errors that occur during the execution of a c program and generally occur due to some illegal operation performed

in the program. For example,

- Dividing a number by zero
- Trying to open a file which is not created
- Lack of free memory space

2. Compile Errors :-

Compile errors are those errors that occur at the time of compilation of the program. C compile errors may be further classified as:

2.1 Syntax Errors :

When the rules of the c programming language are not followed, the compiler will show syntax errors.

2.2 Semantic Errors :

Semantic errors are reported by the compiler when the statements written in the c program are not meaningful to the compiler.

3. Logical Errors :-

Logical errors are the errors in the output of the program. The presence of logical errors leads to undesired or incorrect output and are caused due to error in the logic applied in the program to produce the desired output. Also, logical errors could not be detected by the compiler, and thus, programmers have to check the entire coding of a c program line by line.

Q1. c) Explain switch case and if-else ladder with example.

Ans.

Switch Case :-

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

Example :

Code :

```
// Following is a simple program to demonstrate syntax of switch.
#include <stdio.h>
#include <conio.h>
int main()
{
  int x = 2;
```

```
switch (x)
 {
   case 1: printf("Choice is 1");
         break;
   case 2: printf("Choice is 2");
          break;
   case 3: printf("Choice is 3");
         break;
   default: printf("Choice other than 1, 2 and 3");
          break;
 }
 return 0;
}
```

Output :

```
Choice is 2
```

If-else ladder :-

The if else ladder statement in C programming language is used to test set of conditions in sequence. An if condition is tested only when all previous if conditions in if-else ladder is false. If any of the conditional expression evaluates to true, then it will execute the corresponding code block and exits whole if-else ladder.

Example :

Code :

```
#include<stdio.h>
#include<conio.h>
 void main()
{
     int a;
     printf(“Enter a number : “);
     scanf(“%d”,&a);
     if(a > 0)
     {
          printf(“\nThe number is positive “);
     }
     else if(a<0)
```

```
        {
                printf("\n The number is negative");
        }
    else
        {
                printf("Number is zero");
        }
     getch();
    }
```

Output :

```
Enter a number : 25
The number is positive.
```

Q1. d) Explain any four standard library functions from sting.h ?
Ans.

Standard Library Functions from string.h :-

In the C Programming Language, the Standard Library Functions are divided into several header files.

The following is a list of functions found within the <string.h> header file:

I. Comparison functions
   1. memcmp - Compare Memory Blocks
   2. strcmp - String Compare
   3. strcoll - String Compare Using Locale-Specific Collating sequence.
   4. strncmp - Bounded String Compare
   5. strxfrm - Transform Locale-Specific String

II. Concatenation functions
   1. strcat - String Concatenation
   2. strncat - Bounded String Concatenation

III. Copying functions
   1. memcpy - Copy Memory Block
   2. memmove - Copy Memory Block
   3. strcpy - String Copy
   4. strncpy - Bounded String Copy

IV. Search functions
   1. memchr - Search Memory Block for Character
   2. strchr - Search String for Character
   3. strcspn - Search String for Intial Span of Characters Not in Set

4. strpbrk - Search String for One of a Set of Characters
5. strrchr - Search String in Reverse for Character
6. strspn - Search String for Initial Span of Characters in Set
7. strstr - Search String for Substring
8. strtok - Search String for Token

V. Miscellaneous functions

1. memset - Initialize Memory Block
2. strerror - Convert Error Number to String
3. strlen - String Length

Q1. e) Explain break and continue statement with example.

Ans.

It is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

break Statement :-

The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. The break statement is used with decision making statement such as if...else. In C programming, break statement is also used with switch...case statement.

Syntax of break statement :

```
break;
```

Example :-

Code :

```
// Program to calculate the sum of maximum of 10 numbers
// Calculates sum until user enters positive number
# include <stdio.h>
# include <conio.h>
int main()
{
int i;
double number, sum = 0.0;
for (i=1; i <= 10; ++i)
        {
printf("Enter a n%d: ",i);
```

```
scanf("%lf",&number);
// If user enters negative number, loop is terminated
if(number < 0.0)
        {
                break;
        }
        sum += number; // sum = sum + number;
 }
 printf("Sum = %.2lf",sum);
 return 0;
}
```

Output :

```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

continue Statement :-

The continue statement skips some statements inside the loop. The continue statement is used with decision making statement such as if...else.

Syntax of continue Statement :

```
continue;
```

Example :-

Code :

```
// Program to calculate sum of maximum of 10 numbers
// Negative numbers are skipped from calculation
# include <stdio.h>
# include <conio.h>
int main()
{
   int i;
   double number, sum = 0.0;
```

```
    for(i=1; i <= 10; ++i)
    {
        printf("Enter a n%d: ",i);
        scanf("%lf",&number);
        // If user enters negative number, loop is terminated
        if(number < 0.0)
        {
            continue;
        }
        sum += number; // sum = sum + number;
    }
    printf("Sum = %.2lf",sum);
    return 0;
}
```

Output :

```
Enter a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

Q2. a) Define Algorithm. Write Algorithm to check whether given number is Armstrong number or not also mention input and output specifications to algorithm.

Ans.

Algorithm :-

An Algorithm is a sequence of steps to solve a problem. Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

Characteristics of Algorithm :-

1) ***Finiteness*** - An algorithm must always terminate after a finite number of steps.
2) ***Definiteness*** - Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.
3) ***Input*** - An algorithm has zero or more inputs, i.e, quantities which are given to it initially before the algorithm begins.
4) ***Output*** - An algorithm has one or more outputs i.e, quantities which have a specified relation to the inputs.
5) ***Effectiveness*** - An algorithm is also generally expected to be effective. This means that all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time.

Algorithm to check whether given number is Armstrong or not :-

Step I : Start.
Step II : Input n, sum, rem, temp.
Step III : sum = 0, rem=0.
Step IV : Print "Enter an integer number : "
Step V : Read n.
Step VI : temp = n.
Step VII : If temp is less than equal to zero Then,
Go to *Step IX*.
Else
rem = temp mod 10
sum = sum + (rem X rem X rem)
temp = temp / 10
Step VIII : Go to *Step VII*.
Step IX : If sum is equal to n Then,
Print "Number n is an Armstrong number."
Else
Print "Number n is not an Armstrong number."
Step X : Stop

Q2. b) Explain various storage classes with example.
Ans.

Storage Classes :-

Storage Classes are used to describe about the features of a variable/function. 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. To specify the storage class for a variable, the following syntax is to be followed:

Syntax:

storage_class var_data_type var_name;

C language uses 4 storage classes, namely:

i. 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). Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared. However, they can be accessed outside their scope as well using the concept of pointers given here by pointing to the very exact memory location where the variables resides. They are assigned a garbage value by default whenever they are declared.

ii. 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. It can be accessed within any function/block. Also, a normal global variable can be made extern as well by placing the 'extern' keyword before its declaration/definition in any function/block. This basically signifies that we are not initializing a new variable but instead we are using/accessing the global variable only. The main purpose of using

extern variables is that they can be accessed between two different files which are part of a large program. For more information on how extern variables work, have a look at this link.

iii. 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. So we can say that they are initialized only once and exist till the termination of the program. Thus, no new memory is allocated because they are not re-declared. Their scope is local to the function to which they were defined. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

iv. 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. If a free register is not available, these are then stored in the memory only. Usually few variables which are to be accessed very frequently in a program are declared with the register keyword which improves the running time of the program. An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.

Example :-

Code :

```
// A C program to demonstrate different storage classes
#include <stdio.h>
 #include <conio.h>
extern int x = 9;
int z = 10;
int main()
{
```

```
    auto int a = 32;
    register char b = 'G';
    extern int z;
    printf("Hello World!\n");
    printf("\nThis is the value of the auto " " integer 'a': %d\n",a);
    printf("\nThese are the values of the" " extern integers 'x' and 'z'" "
                respectively: %d and %d\n", x, z);
    printf("\nThis is the value of the " "register character 'b': %c\n",b);
    x = 2;
    z = 5;
    printf("\nThese are the modified values " "of the extern integers 'x'
                and " "'z' respectively: %d and %d\n",x,z);
    printf("\n'y' is a static variable and its " "value is NOT initialized to
                5 after" " the first iteration! See for" " yourself :)\n");
    while (x > 0)
    {
       static int y = 5;
       y++;
       printf("The value of y is %d\n",y);
       x--;
    }
    printf("\nBye! See you soon.\n");
    return 0;
}
```

Output :

```
Hello World!
This is the value of the auto  integer 'a': 32
These are the values of the extern integers 'x' and 'z'
respectively: 9 and 10
This is the value of the register character 'b': G
These are the modified values of the extern integers 'x'
and 'z' respectively: 2 and 5
'y' is a static variable and its value is NOT initialized
to 5 after the first iteration! See for yourself :)
The value of y is 6
The value of y is 7
Bye! See you soon.
```

Q3. a) Explain Nested Structure. Write a program using nested structure to create an Array of structure to store the details of N students.
The details are,

1. **Student name**
2. **Student roll no**
3. **Marks of Physics, Chemistry, Maths.**

Calculate total of P-C-M. Display the data in the format
Name Roll no Total marks
Ans.

Nested structure :-

Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure. The structure variables can be a normal structure variable or a pointer variable to access the data. Nested structure in c language can have another structure as a member. There are two ways to define nested structure in c language:

1) Separate structure :

We can create 2 structures, but dependent structure should be used inside the main structure as a member. Let's see the code of nested structure.

2) Embedded structure :

We can define structure within the structure also. It requires less code than previous way. But it can't be used in many structures.

The syntax of nested structure is given as :

```
struct structure_name
{
data_type variable_name;
_ _ _ _ _ _ _
struct
{
data_type variable_name;
_ _ _ _ _ _ _
internal_structure_name;
_ _ _ _ _ _ _ _
}
```

Solution Program :-

Sourse Code :

```
#include <stdio.h>
#include <conio.h>
struct students
{
        char name[30] ;
        int roll_no, total;
        struct
        {
                int physics, chemistry, maths;
        }
        marks;
};
void main( )
{
        struct students n[100];
        int n, i, j;
        clrscr( );
        printf("Enter number of students : ");
        scanf("%d", &n);
        for (i=0 ; i<=n-1 ; i++)
        {
                printf("Enter following details of student :- ");
                printf("Name : ");
                scanf("%s", & n[i].name);
                printf("Roll number : ");
                scanf("%d", & n[i].roll_no);
                printf("Marks in Physics : ");
                scanf("%d", & n[i].marks.physics);
                printf("Marks in Chemistry : ");
                scanf("%d", & n[i].marks.chemistry);
                printf("Marks in Mathematics : ");
                scanf("%d", & n[i].marks.maths);
                n[i].total = n[i].marks.physics + n[i].marks.chemistry +
                                n[i].marks.maths;
        }
        printf("\n Name \t Roll Number \t Total \n");
```

```
        printf(" _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _\n");
        for (i=0 ; i<=n-1 ; i++)
        {
            printf("%s \t %d \t %d \n", n[i].name, n[i].roll_no,
                n[i].total);
        }
getch();
}
```

Output :-

```
Enter number of students : 3
Enter following details of student :-
Name : Saurav Palande
Roll number : 45
Marks in Physics : 50
Marks in Chemistry : 60
Marks in Mathematics : 30
Enter following details of student :-
Name : Sayali Rajapurkar
Roll number : 40
Marks in Physics : 50
Marks in Chemistry : 60
Marks in Mathematics : 40
Enter following details of student :-
Name : Amrit Shah
Roll number : 65
Marks in Physics : 60
Marks in Chemistry : 60
Marks in Mathematics : 60

Name                Roll Number          Total
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Saurav Palande      45                   140
Sayali Rajapurkar   40                   150
Amrit Shah          65                   180
```

Q3. b) Define pointer and its use. Explain array of pointer with example. Write program to swap the values by using call by reference concept.
Ans.

Pointer :-

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

type *var-name;

Uses of pointer :-

1. Pointers reduce the length and complexity of a program.
2. They increase execution speed.
3. A pointer enables us to access a variable that is defined outside the function.
4. Pointers are more efficient in handling the data tables.
5. The use of a pointer array of character strings results in saving of data storage space in memory.

Array of Pointers :-

Just like we can declare an array of int, float or char etc, we can also declare an array of pointers, here is the syntax to do the same.

Syntax :

datatype *array_name[size];

Example :

int *arrop[5];

Here arrop is an array of 5 integer pointers. It means that this array can hold the address of 5 integer variables, or in other words, you can assign 5 pointer variables of type pointer to int to the elements of this array. arrop[i] gives the address of i th element of the array. So arrop[0] returns address of variable at position 0, arrop[1] returns address of variable at position 1 and so on. To get the value at address use indirection operator (*). So *arrop[0] gives value at address[0], Similarly *arrop[1] gives the value at address arrop[1] and so on.

Solution Program :-

Source Code :

```
//Program to swap the values by using call by reference concept.
#include <stdio.h>
#include <conio.h>
void swap( int *x, int *y )
{
        int t ;
        t = *x ;
        *x = *y ;
        *y = t ;
        printf("In function :");
        printf( "\nx = %d \t y = %d\n", *x,*y);
}
void main( )
{
        int a, b;
        printf("Enter the value of a : ");
        scanf("%d", &a);
        printf("Enter the value of b : ");
        scanf("%d", &b);
        printf("Before swapping : \n");
        printf ( "\na = %d \t b = %d\n", a, b ) ;
        swap ( &a, &b ) ;
        printf("After swapping : \n");
        printf ( "\na = %d \t b = %d\n", a, b ) ;
        getch();
}
```

Output :

```
Enter the value of a : 10
Enter the value of b : 20
Before swapping :
a=10        b=20
In function :
x=20        y=10
After swapping :
a=20        b=10
```

Q4. a) Explain recursive function. Write a program to find the GCD of a number by using recursive function.

Ans.

Recursive function :-

In C, a function can call itself. This process is known as recursion. And a function that calls itself is called as the recursive function. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function. 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.

Program :-

Source Code :

```
//Program to find GCD of a number by using recursive function.
#include <stdio.h>
#include <conio.h>
int gcd(int n1, int n2);
int main( )
{
        int n1, n2;
        printf("Enter two positive integers : \n");
        scanf("%d %d", &n1, &n2);
        g = gcd(n1,n2);
        printf("G.C.D of %d and %d is %d.", n1, n2, g);
        return 0;
}
int gcd(int n1, int n2)
{
         while (n1 != n2)
        {
                if (n1 > n2)
                        return gcd(n1 – n2, n2);
                else
                        return gcd(n1, n2 – n1);
        }
 return a;
 }
```

Output :

```
Enter two positive integers:
366
60
G.C.D of 366 and 60 is 6.
```

Q4. b) Write a program to perform matrix multiplication by passing input matrix to the function and printing resultant matrix.

Ans.

Program :-

Sourse code :

```
//Program for matrix multiplication using functions.
#include <stdio.h>
#include <stdlib.h>
void input(int m, int n, int a[m][n])
{
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         printf("%d, %d : ", i, j);
         scanf("%d", &a[i][j]);
      }
   }
}

void print(int m, int n, int a[m][n])
{
   int i, j;
   for (i = 0; i < m; i++) {
      for (j = 0; j < n; j++) {
         printf("%3d ", a[i][j]);
      }
      printf("\n");
   }
}
void multiply(int m, int n, int p, int a[m][n], int b[n][p], int c[m][p])
```

```
{
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < p; j++) {
         c[i][j] = 0;
         for (int k = 0; k < n; k++) {
            c[i][j] += a[i][k] * b[k][j];
         }
      }
   }
}
void main()
{
   int r1, c1, r2, c2;
   printf("Row and column for matrix #1 :\n");
   scanf("%d %d", &r1, &c1);
   printf("Row and column for matrix #2 :\n");
   scanf("%d %d", &r2, &c2);
   if (r2 != c1) {
      printf("The matrices are incompatible.\n");
      exit(EXIT_FAILURE);
   }
   int mat1[r1][c1], mat2[r2][c2], ans[r1][c2];
   printf("Enter elements of the first matrix.\n");
   input(r1, c1, mat1);
   printf("The elements of the first matrix are :\n");
   print(r1, c1, mat1);
   printf("Enter elements of the second matrix.\n");
   input(r2, c2, mat2);
   printf("The elements of the second matrix are :\n");
   print(r2, c2, mat2);
   multiply(r1, r2, c2, mat1, mat2, ans);
   printf("The product is :\n");
   print(r1, c2, ans);
   getch( );
}
```

Output :

```
Row and column for matrix #1 :
2
2

Row and column for matrix #2 :
2
2

Enter elements of the first matrix.
0, 0 : 1
0, 1 : 17
1, 0 : 25
1, 1 : 30

The elements of the first matrix are :

1  17
25  30

Enter elements of the second matrix.
0, 0 : 74
0, 1 : 89
1, 0 : 3
1, 1 : 65

The elements of the second matrix are :

74  89
3  65

The product is :

125 1194
1940 4175
```

Q5. a) Write a program to display following pattern:

1
232
34543
4567654
567898765

Ans.

Program :-

Source code :

```
#include <stdio.h>
#include <conio.h>
void main( )
{
        int i, s, r, k=0, c=0, j=0;
        printf("Enter the number of rows : ");
        scanf("%d", &r);
        for( i=1 ; i<=r ; i++ )
        {
                for( s=1 ; s<=r-i ; s++ )
                {
                        prinf(" ");
                        c++;
                }
                while( k != 2*i-1 )
                {
                        if(c<=r-1)
                        {
                                printf("%d",(i+k));
                                c++;
                        }
                        else
                        {
                                j++;
                                printf("%d",(i+k-2*j));
                        }
                        ++k;
                }
```

```
                j=c=k=0;
                printf("\n");
                }
        getch( );
}
```

Output :

```
Enter the number of rows : 5
      1
     232
    34543
   4567654
567898765
```

Q5. b) Write user defined function to implement string concatenation.
Ans.

Program :-

Source code :

```
#include<stdio.h>
#include<string.h>
void concat(char[], char[]);
int main() {
  char s1[50], s2[50];
  printf("\nEnter String 1 :");
  gets(s1);
  printf("\nEnter String 2 :");
  gets(s2);
  concat(s1, s2);
  printf("\nConcated string is :%s", s1);
  return (0);
}
void concat(char s1[], char s2[])
{
  int i, j;
  i = strlen(s1);
```

```
    for (j = 0; s2[j] != '\0'; i++, j++)
        {
            s1[i] = s2[j];
        }

    s1[i] = '\0';
}
```

Output :

```
Enter String 1 : Prathamesh
Enter String 2 : Padave
Concated string is : PrathameshPadave
```

Q5. c) Explain need of file data and various modes of files also write program to create edit copy of file.

Ans.

File Handling :-

File handling is an important part for any language. File handling helps you to store data (any type of data) in a controlled manner like for cookies, for any type of configurations etc. A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. The fopen() function is used to create a new file or to open an existing file.

General Syntax:

*fp = FILE *fopen(const char *filename, const char *mode);

Here, *fp is the FILE pointer (FILE *fp), which will hold the reference to the opened(or created) file. Filename is the name of the file to be opened and mode specifies the purpose of opening the file. Mode can be of following types,

- ✓ r - opens a text file in reading mode
- ✓ w - opens or create a text file in writing mode.
- ✓ a - opens a text file in append mode
- ✓ r+ - opens a text file in both reading and writing mode
- ✓ w+ - opens a text file in both reading and writing mode

- ✓ a+ - opens a text file in both reading and writing mode
- ✓ rb - opens a binary file in reading mode
- ✓ wb - opens or create a binary file in writing mode
- ✓ ab - opens a binary file in append mode
- ✓ rb+ - opens a binary file in both reading and writing mode
- ✓ wb+ - opens a binary file in both reading and writing mode
- ✓ ab+ - opens a binary file in both reading and writing mode

Program :-

Source code :

```
/* Program to create edit copy of file */
# include <stdio.h>
#include <conio.h>
void main( )
{
        FILE *fp ;
        char sub[50], temp[20];
        int s, t;
        clrscr( );
        fp=fopen("Exam.txt","w+");
        printf( "Enter seat number and subject of the candidate : " ) ;
        scanf("%d %s", &s, &sub);
        printf(" Seat No \t Subject \n");
        rewind(fp);
        fscanf(fp,"%s %d", &temp, &t);
        printf("%s %d", temp, t);
        fclose(fp) ;
        return 0;
}
```

Output :-

```
Enter Seat number and Subject of the student : 2001
Chemistry
Seat number          Subject
   2001              Chemistry
```

Q6. a) Write a program to sort given array in ascending order.
Ans.

Program :-

Source code :

```
//Program to sort given array in ascending order
#include <stdio.h>
#include <conio.h>
void main()
{
        int i, j, a, n, array[50];
        printf("How many numbers in an array…? \n");
        scanf("%d", &n);
        printf("Enter the numbers. \n");
        for (i = 0; i < n; i++)
                scanf("%d", &array[i]);
        for (i = 0; i < n; i++)
        {
                for (j = i + 1; j < n; j++)
                        {
                                if (number[i] > number[j])
                                {
                                        a =  number[i];
                                        number[i] = number[j];
                                        number[j] = a;
                                }
                        }
        }
        printf("The numbers arranged in ascending order are given
                below \n");
        for (i = 0; i < n; ++i)
                printf("%d\n", number[i]);
        getch( );
}
```

Output :

```
How many numbers in an array…?
5
Enter the numbers.
3
1
456
200
150
The numbers arranged in ascending order are given below
1
3
150
200
456
```

Q6. b) Write a program for finding sum of series 1+2+3+4+…….upto n terms.
Ans.

Program :-
Source code :

```
//program to calculate sum of series up to n terms 1 + 2 + 3 +…+n.
#include <stdio.h>
#include <conio.h>
void main( )

{
    int n,i;
    int sum=0;
    printf("Enter the n i.e. max value of series: ");
    scanf("%d",&n);
    sum = (n * (n + 1)) / 2;
    printf("Sum of the series: ");
    for (i =1 ; i <= n ; i++)
    {
        if (i!=n)
```

```
                printf("%d + ",i);
            else
                printf("%d = %d ",i,sum);
        }
        getch( );
}
```

Output :-

```
Enter the n i.e. max value of series: 5
Sum of the series: 1 + 2 + 3 + 4 + 5 = 15
```

Q6. c) Draw the flow chart to find roots of a quadratic equation.
Ans.

Algorithm :-

Step I : Start.
Step II : Input a,b,c,del,r1,r2.
Step III : Print "Enter the value of coefficient of the X^2."
Step IV : Read a.
Step V : Print "Enter the value of coefficient of the X."
Step VI : Read b.
Step VII : Print "Enter the value of constant C."
Step VIII : Read c.
Step IX : del = b X b – 4 X a X c
Step X : If del is less than zero Then,
r1 = (-b / (2 X a)) + sqrt (del) / 2 X a
r2 = (-b / (2 X a)) - sqrt (del) / 2 X a
Print "Roots are complex and unequal."
Print "The roots for the entered values are r1 and r2."
Else - if del is greater than zero Then,
r1 = (-b / (2 X a)) + sqrt (del) / 2 X a
r2 = (-b / (2 X a)) - sqrt (del) / 2 X a
Print "Roots are real and unequal."
Print "The roots for the entered values are r1 and r2."
Else,
r1 = (-b / (2 X a)) + sqrt (del) / 2 X a
Print "Roots are real and equal."
Print "The root for the entered values is r1."
Step XI : Stop

Flowchart :-