2070 Chaitra

Q1. a)What do you mean by software and explain about the generation of programming languages?

Ans: Computer software is a computer program which is a sequence of instructions designed to direct a computer to perform certain task. A software is an interface between user and computer. It is responsible for controlling, integrating and managing the hardware components of a computer system and for accomplishing specific task.

The language used to give instructions to a computer is known as programming language. There are five generations of programming languages .They are as follows:

1st generation of programming languages(Machine language):

Machine language is made up of entirely 0s and 1s and is the only programming language that the computer can understand directly without translation. There is not one universal machine language because the language must be written in accordance with the special characteristics of a given processor. Each type or family of processor requires its own machine language. For this reason, machine language is said to be machine-dependent (also called hardware-dependent). Machine language programs have the advantage of very fast execution speeds and efficient use of primary memory. Use of machine language is very tedious, difficult and time consuming method of programming. Machine language is low-level language. Since the programmer must specify every detail of an operation, a low level language requires that the programmer have detailed knowledge of how the computer works. Programmers had to know a great deal about the computer's design and how it functioned. As a result, programmers were few in numbers and lacked complexity.

2nd generation of programming languages(Assembly Language):

They are also classified as low-level languages because detailed knowledge of hardware is still required. They were developed in 1950s. They are also machine dependent. Assembly languages use mnemonic operation codes and symbolic addresses in place of 1s and 0s to represent the operation codes. A mnemonic is an alphabetical abbreviation used as memory aid. This means a programmer can use abbreviation instead of having to remember lengthy binary instruction codes. Before they can be used by the computer, assembly languages must be translated into machine language. A language translator program called an assembler does this conversion. The advantages of programming with assembly languages are that they produce programs that are efficient, use less storage, and execute much faster than programs designed using high-level languages.

3rd generation of programming languages(Procedure Oriented Language):

Third generation languages, also known as high-level languages, are very much like everyday text and mathematical formulas in appearance. They are designed to run on a number of different computers with few or no changes. Most high level languages are considered to be procedure-oriented, or Procedural languages, because the program instructions comprise lists of steps, procedures, that tell the computer not only what to do but how to do it. The programmer spends less time developing software with a high level language than with assembly or machine language because fewer instructions have to be created. A language translator is required to convert a high-level language program into machine
language. Two types of language translators are used with high level languages: compilers and interpreters.

4th generation of programming languages (Problem Oriented Language):

Fourth generation languages are also known as very high level languages. They are nonprocedural languages, so named because they allow programmers and users to specify what the computer is supposed to do without having to specify how the computer is supposed to do it. Consequently, fourth generation languages need approximately one tenth the number of statements that a high level languages needs to achieve the same results. Because they are so much easier to use than third generation languages, fourth generation languages allow users, or non-computer professionals, to develop software. Depending on the language, the sophistication of fourth generation languages varies widely. These languages are usually used in conjunction with a database and its data dictionary. Five basic types of language tools fall into the fourth generation language category.

1. Query languages

2. Report generators.

3. Applications generators.

4. Decision support systems and financial planning languages.

5. Some microcomputer application software.

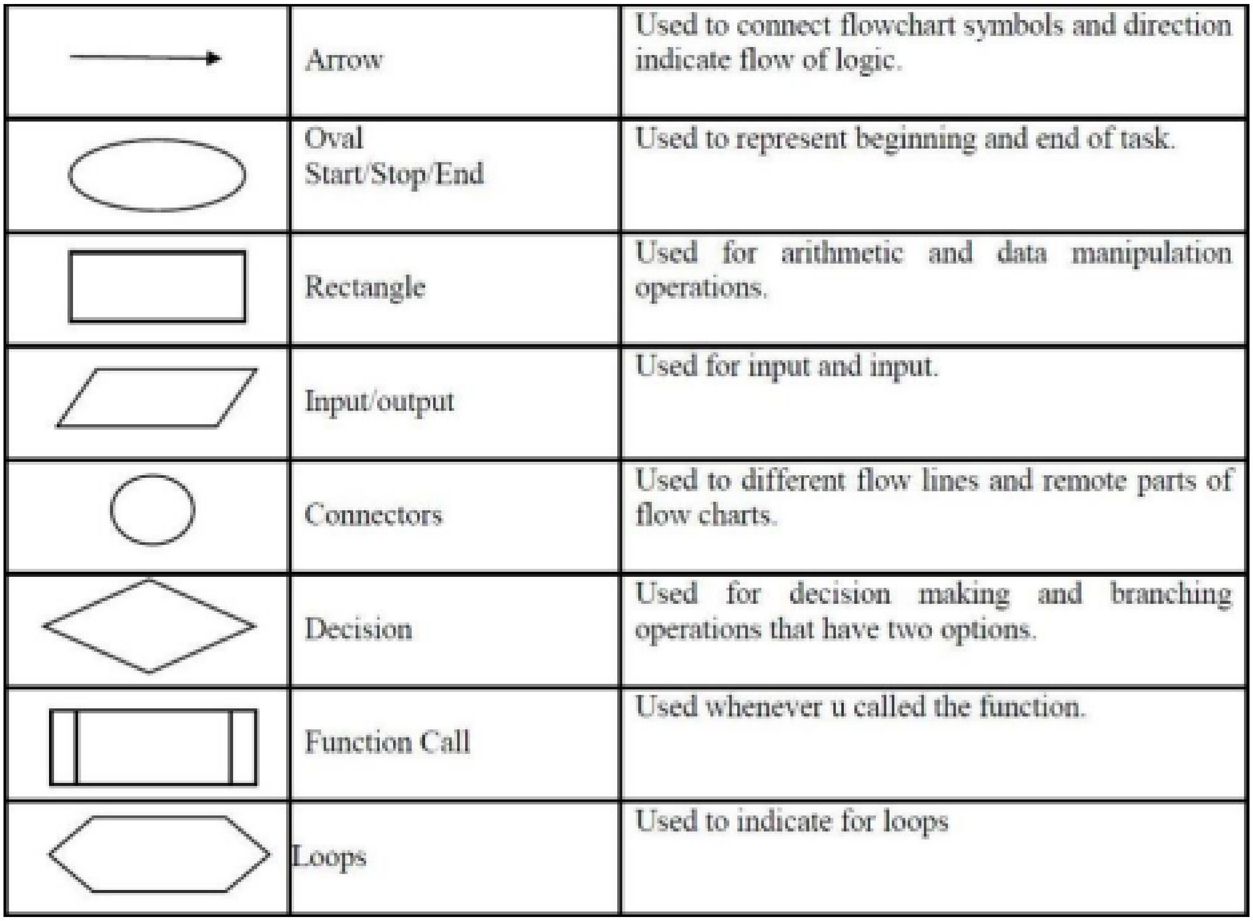
5th Generation of programming languages (Natural Language):

Natural Languages represent the next step in the development of programming languages, i-e fifth generation languages. The text of a natural language statement very closely resembles human speech. In fact, one could word a statement in several ways perhaps even misspelling some words or changing the order of the words and get the same result. These languages are also designed to make the computer “smarter”. Natural languages already available for microcomputers include Clout, ,Q&A, and Savvy Retriever (for use with databases) and HAL (Human Access Language. The use of natural language touches on expert systems, computerized collection of the knowledge of many human experts in a given field, and artificial intelligence, independently smart computer systems.

1.b) Define the term flowchart. Discuss about different symbols used in flowchart.

Ans: A flowchart is a pictorial representation of an algorithm that uses boxes of different shapes to denote different types of instructions. The actual instructions are written within these boxes using clear and concise statements. These boxes are connected by solid lines having arrow marks to indicate the flow of operation, that is, the exact sequence in which the instructions are to be executed.

The symbols used in a flowchart are as follows:t



→	Arrow	Used to connect flowchart symbols and direction indicate flow of logic.
⬭	Oval Start/Stop/End	Used to represent beginning and end of task.
▭	Rectangle	Used for arithmetic and data manipulation operations.
▱	Input/output	Used for input and input.
○	Connectors	Used to different flow lines and remote parts of flow charts.
◇	Decision	Used for decision making and branching operations that have two options.
◫	Function Call	Used whenever u called the function.
⬡	Loops	Used to indicate for loops

2.a) Find out the value of a ,b, and c where following expressions are executed. int a = 2, b =3, c;

a =(b++) + (++b) + a;

c = a>b? a:b;

b=(a++)+(b—)+a;

c = c++ *b--;

Ans: a= 11, b = 25, c= 260.

a b c

2 3 0

10 26 10

11 25 260

2.b) What are the difference between formatted and unformatted I/O statements? Describe with proper examples..

Ans:

Formatted Input/Output Statements	Unformatted Input/Output Statements
1. The user can decide the format of the output.	1. The computer decides the format of output. The user can not chage it.
2. It is more complex to write.	2. It is less complex to write.
3. Commonly used formatted I/O statements are: scanf(), printf().	3. Commonly used unformatted I/O statements are: gets(),putch(),etc.
4. They contain format specifier in the syntax.	4. They do not contain format specifier in the syntax.

3.a)

Explain importance of break and default statements in switch statement.

Ans: The syntax of switch statement is as follows:

```
{

case constant1:

 block of case constant1;

 break;
case constant2:

 block of case constant2;

 break;

case constant3:

 block of case constant3;

 break;

default:

 default block;

}
```

The break statement at the end of each block indicates the end of a particular case and causes as exit from the switch statement, transferring the control to statement following the switch. If the break statement is omitted, execution continues on into the next case statement until either a break or the end of the switch break or the end of the switch is reached. It is important because it prevents unnecessary execution of subsequent case statements.

The default is an optional case. When present, it will be executed if the value of expression does not match with any of the case constants. If not present, no action takes place if all the matches fails and control goes to the next statement. Thus the break statement and default statement are important in switch statement in order to avaoid unexpexted flow of the program.

3. b) Write a C program to display following patter using unformatted output statements.

```
P
Pu
PuL
PULCH
PULCHO
puLcHoW
PULCHOWK
```

```c
#include <stdio.h>

int main() {
    char text[] = "PULCHOWK";

    for (int i = 1; i <= 8; i++) {
        for (int j = 0; j < i; j++) {
            if (j % 2 == 0) {
                printf("%c", text[j]);
            } else {
                printf("%c", text[j] + (32));
            }
        }
        printf("\n");
    }
```

```
      return 0;
}
```

4. a) Define “function definition” and write the program to find sum of two numbers using user defined functions.

Ans:

Function definition is a block of code that specifies the behavior or operation of a function. Function definition in C consists of:

a. function declaration

```
    return_type function_name(type1,, type2,,……type n);
```

b. function call

```
    function_name(arg1, arg2, …argn)
```

c. function definition

```
    return_type function_name(type1 arg1 type 2 arg2,,……..type n,argn)
    {
     Function body;
    }
```

```
#include<stdio.h>
int sum(int a, int b)
{
   return a+b;
}
int main()
{
      int n1, n2, s;
      printf("Enter two numbers");
      scanf("%d%d",&n1,&n2);
      s = sum(n1,n2);
      printf("\nThe sum is:%d", s);
      return 0;
}
```

4.b) What do you mean by “call by value and call by reference” along with suitable example? Ans: Call by reference and call by value are the two methods of passing argument to the function. In call by value, the value of the actual parameter is passed. The original value of passed variable is not modified. The value of argument in the calling function is not changed even if they are changed in called function.

For example:

```
#include<stdio.h>

void swap(int, int);
int main()
{
   int x, y;
   printf("Enter 2 numbers");
   scanf("%d%d",&x,&y);
   printf("Before swapping:x=%d and y=%d",x,y);
   swap(x,y);
   printf("\nAfter swapping:x=%d and y=%d",x,y);
   return 0;
}
void swap(int a, int b)
{
   int temp;
   temp = b;
   b=a;
   a=temp;
}
```

Output:

```
Enter 2 numbers

1

2

Before swapping:x=1 and y=2

After swapping:x=1 and y=2
```

Here, the values of a and b are swapped in the function. However, in the main function, the value of x and y are not swapped. Thus, in call by value method, any change in a or b is not reflected in x and y.

In call by reference, rather than the actual value, the address of variable is passed as argument. Pointers are used to point to the memory location of the variables. Any change in the variable in called function is reflected in the calling function.

For example:

```
#include<stdio.h>
void swap(int *,int *);
int main()
{
   int x, y;
   printf("Enter 2 numbers\n");
   scanf("%d%d",&x,&y);
   printf("Before swapping:x=%d and y=%d",x,y);
   swap(&x,&y);
   printf("\nAfter swapping:x=%d and y=%d",x,y);
   return 0;
}
void swap(int*a, int*b)
{
   int temp;
   temp = *b;
   *b=*a;
   *a=temp;
}
```

OUTPUT

```
Enter 2 numbers

1

2

Before swapping:x=1 and y=2

After swapping:x=2 and y=1
```

Here, address of the variable is passed to the called function. Here addresses of x and y are passed to the function swap. These addresses are copied to a and b respectively. Any operation done inside the swap is actually done on the memory location pointed by the address. Thus, swapping done on *p and *q is reflected on x and y.

Q.5)Can we pass whole array element to function? Wrire a program to display only those students information which are passed. Use separate function to check the result of student. The information of students like Name, Roll No, Address and Marks are passed from main function and pass to functions using array type arguments.

Ans: To pass an array to a function as an argument, we need to pass the name of the array. It means starting address of the memory where members of the array are stored. If we know the starting address of contagious memory inside function and number of members of the array, we can easily access each member by using loop. Passing arrays to function is passing by reference.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>


struct Student {
    int rollNo;
    char name[50];
    char address[100];
    float marks;
};


bool isPassed(float marks) {
    return marks >= 40.0;
}


void displayPassedStudents(struct Student students[], int numStudents) {
    printf("Passed Students:\n");

    for (int i = 0; i < numStudents; i++) {
        if (isPassed(students[i].marks)) {
            printf("roll No: %d\n", students[i].rollNo);
            printf("name: %s\n", students[i].name);
printf("address: %s\n", students[i].address);
            printf("marks: %f\n", students[i].marks);

        }
    }
}

int main() {
    int numStudents;
    printf("Enter the number of students: ");
    scanf("%d", &numStudents);
    struct Student students[numStudents];


    for (int i = 0; i < numStudents; i++) {
        printf("\nEnter details for student %d:\n", i + 1);
        printf("Roll No: ");
        scanf("%d", &students[i].rollNo);
        printf("Name: ");
        scanf(" %[^\n]", students[i].name);
        printf("Address: ");
        scanf(" %[^\n]", students[i].address);
        printf("Marks: ");
        scanf("%f", &students[i].marks);
    }


    displayPassedStudents(students, numStudents);

    return 0;
}
```

Q.6.a)Explain the use of typedef of keyboard in structures.

Ans: Typedef is a keyword in C whose main purpose is to form complex data types from more basic data types. A typedef can be used to simplify the declaration for structure.

For example:

```
typedef struct
{
```

```
type
member1;
type
member2;
type
member3;
} type_name;
```

Here type_name represents the stucture definition associated with it. Now this type_name can be used to declare a variable of this stucture type.

Q.6.b)Explain the need of nested structure. Write a C program to convert data in BS to data in AD using structure. Use the difference of current data.

Ans: A structure within another student is known as nested structure. It is needed for organizing and handling complex hierarchy between data elements. For example: a structure could have elements: name, date of birth and address. The date of birth could further be a structure with elements year, month and day.

```
#include<stdio.h>


void main()
{
    struct date{

    int day;
    int month;
    int year;



    }d;
int y=56;
int m=8;
int dae=16;
    printf("enter year\n");
    scanf("%d",&d.year);
```

```
                 printf("enter month\n");
                mon:
                scanf("%d",&d.month);
if(d.month>12){printf("invalid month enter again \n") ; goto mon;}


                 printf("enter day\n");
                 da:
                scanf("%d",&d.day);
if(d.day>32){printf("invalid day enter again \n") ; goto da;}

                    if(d.day<dae)
                    {
                        d.month --;
                       d.day= d.day+30;
                       d.day=d.day-dae;

                    }
                    else {
                       d.day= d.day-dae;
                    }
                    if(d.month<m)
                    {
                        d.year--;
                        d.month=d.month+12;
                        d.month=d.month-m;

                    }
                    else{
                        d.month=d.month-m;
                    }
                    d.year=d.year-y;

                printf("%d-%d-%d",d.year,d.month, d.day);



            }
```

Q7.a) A pointer variable is used to store address of some other variables,

however, we need to specify datatype while declaring pointer variable. Why?

Ans:

A pointer is used to store address of some other variable. However, We need to specify the datatype while declaring pointer because that datatype is referring to the data type of the variable to which the pointer is pointing. For example:

int *p;

Here the pointer variable points to a variable of int type.

Different data types have different sizes in memory, and pointer arithmetic also relies on data type. While dereferencing a pointer, the compiler needs to know the data type in order to perform operations. Due to the ability of a pointer to directly refer to the value that it points to, it becomes necessary to specify which data type is pointing to, in the declaration.

Q.7.b) Briefly explain array of pointers. How are array and pointer related? Give example. Ans:

An array of pointers is an arrangement where each element of the array is a pointer to some other data, rather than being a direct value. This concept combines the features of arrays and pointers in C, allowing the user to store and manipulate multiple pointers in a sequential manner.

When an array is declared, compiler allocates sufficient amount of memory to contain all the elements of the array. Base address i.e address of the first element of the array is also allocated by the compiler. we can use a pointer to point to first element of array, and then we can use that pointer to access the array elements.

For example:

```
#include <stdio.h>
int main()
{
int i;
int a[5] = {1, 2, 3, 4, 5};
int *p = a; // same as int*p = &a[0]
for (i = 0; i < 5; i++)
{
printf("%d",*p); p++;
}
return 0;
}
```

In the above program, the pointer *p will print all the values stored in the array one by one. We can also use the Base address (a in above case) to act as a pointer and print all the values. The generalized form for using pointer with an array, *(a+i) is same as: a[i]

Q.8.a) Define opening and closing a file along with suitable examples.

Ans:

Opening a file is performed using the library function in the "stdio.h"

header file: fopen(). The syntax for opening a file in standard I/O is:

fptr = fopen("file","mode").

The file can be opened in various modes and their meanings are as follows:

1. r: It opens the file in reading mode. If the file does not exist, fopen() returns NULL. 2. w: It opens the file for writing. If the file exists, the contents are overwritten and if the file does not exist, it is created.

3. r+: It opens the file for both reading and writing. If the file does not exist, fopen() returns NULL. 4. w+: It opens the file for both reading and writing. If the file exists, the contents are overwritten and if the file does not exist, it is created.

5. a: It opens the file in append mode i.e. data is added to the end of the file. If the file does not exist, it is created.

6. a+: It opens the file for both reading and appending. If the file does not exist, it is created.

fopen() finds the file from specified path and loads the file from the disk to the buffer and sets up a file pointer that points to the first character of the pointer.

Closing a file is performed using library function fclose() in the <stdio.h> header file. When the file is closed, the following operations are performed:

1. The characters in buffer would be written to the file on the disk.
2. A character with ASCII value 26 would get written to the end of the file.
3. The buffer associated with the file is removed from the memory.

For example:

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

```
#include <stdlib.h>
int main()
{
int num;
FILE
*fptr;
fptr = fopen("C:\\program.txt","w");
if(fptr == NULL)
{
printf("Error!");
exit(1);s
}
printf("Enter num: ");
scanf("%d",&num);
fprintf(fptr,"%d",num);
fclose(fptr);
return 0;
}
```

Here, file is opened in write mode using fptr. After the required data is written in the file, it is closed using fclose(fptr).

Q.8.b) Write a program to display records in sorted order sorting is performed in ascending order with respect to name using data files concept.

Ans:

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

struct Record {
    char name[50];
    int age;
};

void swap(struct Record *a, struct Record *b) {
    struct Record temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    FILE *file;
    struct Record records[100];
    int numRecords, i, j;

    file = fopen("records.txt", "r");
    if (file == NULL) {
        printf("Error opening file");
    }
    fscanf(file, "%d", &numRecords);
    for (i = 0; i < numRecords; i++) {
        fscanf(file, "%s %d", records[i].name, &records[i].age);
    }


    fclose(file);


    for (i = 0; i < numRecords - 1; i++) {
        for (j = 0; j < numRecords - i - 1; j++) {
            if (strcmp(records[j].name, records[j + 1].name) > 0) {
                swap(&records[j], &records[j + 1]);
            }
        }
    }


    printf("Sorted records in ascending order by names:\n");
    for (i = 0; i < numRecords; i++) {
        printf("%s %d\n", records[i].name, records[i].age);
    }

    return 0;
}
```

Q.9.a) Compare arithmetic and logical if statements in FORTRAN with suitable examples. Ans:

Arithmetic If statement	Logical If statement
1. It uses any valid arithmetic expression depending on its value to transfer program control.	1. It uses relational and logical oerations to transfer program control.
2. The expression used in arithmetic if generates either negative, zero or positive value.	2. The expression used in logical if generates either true or false.
3. Its syntax is: if(expression)k1,k2,k3	3. Its syntax is: If (condition) then
4. For example: real x read(*,*) x if(x) 1,2,3 1. Write(*,*) ‘Negative’ goto 4 2. Write(*,*) ‘Zero’ goto 4 3. Write(*,*) ‘Positive’ goto 4 4. end	5. For example: real x read(*,*),x if(n. gt. 0) then write(*,*)’Positive’ elseif(n. lt. 0) then write(*,*)’Negative’ else write(*,*)’Zero’ endif stop end

Q.9.b) Write a FORTRAN program to read m*n matrix, transpose it and display both the matrices. Ans:

```
program MatrixTranspose


    integer, parameter :: max_rows = 10
```

```
        integer, parameter :: max_cols = 10

        integer :: m, n, i, j
real :: matrix(max_rows, max_cols), transpose(max_cols, max_rows)
        write(*,*) "Enter the number of rows (m):"
        read(*,*) m
        write(*,*) "Enter the number of columns (n):"
        read(*,*) n

        write(*,*) "Enter the matrix elements:"
        do i = 1, m
            do j = 1, n
                read(*,*) matrix(i, j)
            end do
        end do

        do i = 1, n
            do j = 1, m
                transpose(i, j) = matrix(j, i)
            end do
        end do

        write(*,*) "Original Matrix:"
        do i = 1, m
          do j = 1, n
                write(*,*) matrix(i, j)
            end do
            write(*,*)
        end do

        write(*,*) "Transposed Matrix:"
        do i = 1, n
            do j = 1, m
                write(*,*) transpose(i, j)
            end do
            write(*,*)
        end do

      end program MatrixTranspose
```

submitted by : 079bei010- AUSTINA ARYAL 079bei026- PRAZWAL CHAPAGAIN 079bei041- SOSTIKA SHRESTHA