Introduction to Programming

Sample Question Paper

Note: Attempt five Questions in all, Q.No. 1 is compulsory.

Q. 1. a). What is a flowchart? 2.5 X10

b) Explain 4 characteristics of c language.

c). Write a small code for do while loop.

d). What is the use of gets and puts.

e) Explain switch statement.

f) Differentiate between array and variable.

g) Explain structure.

h) Write a small code for call by value.

i) Whatis the use of typedef in c.

j) What is a static function?

Q.2. Explain in detail High level, Machine language and Assembly language. 12.5

Q 3. Write a program showing the use of if0else and switch statements in C . 12.5

Q4.Write a program to add 2 matrices. 12.5

Q. 5. Explain File handling in c in detail. 12.5

Answers

Ans. 1. a) A **flowchart** is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution to a givenproblem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields. Flowcharts are used in designing and documenting complex processes or programs. Like other types of diagrams, they help visualize what is going on and thereby help the people to understand a process, and perhaps also find flaws, bottlenecks, and other less-obvious features within it. There are many different types of flowcharts, and each type has its own repertoire of boxes and notational conventions. The two most common types of boxes in a flowchart are:

- a processing step, usually called *activity*, and denoted as a rectangular box
- a decision, usually denoted as a diamond.

b) ***Clarity of source code*** *– the extent to which inherent language features support source code that is readable and understandable and that clearly reflects the underlying logical structure of the program.*

Complexity management (architecture support) *– the extent to which inherent language features support the management of system complexity, in terms of addressing issues of data, algorithm, interface, and architectural complexity.*

Concurrency support *– the extent to which inherent language features support the construction of code with multiple threads of control (also known as parallel processing).*

Maintainability *– the extent to which inherent language features support the construction of code that can be readily modified to satisfy new requirements or to correct deficiencies.*

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

int main ()
{
  /* local variable definition */
  int a = 10;

  /* do loop execution */
  do
  {
     printf("value of a: %d\n", a);
     a = a + 1;
  }while( a < 20 );

  return 0;
```

```
}
```

d) gets: from standard input to memory
puts: from memory to standard input

Example :

```
#include<stdio.h>
void main( )
{
char name[10];
printf("What is your first and last name?");
gets(name);
puts(name);
}
```

gets() and puts() both are unformatted function.

gets() is used to read stdin into the character array pointed to by a string variable str until a newline character is found or end-of-file occurs. A null character is written immediately after the last character read into the array. It is defined in stdio.h header file.

Prototype : char *gets(char *str);

Example: char *sname; gets(sname);

puts() is an unformatted string function that writes character string str and a newline to stdout.

Prototype : int puts(char *str);

Example: puts("Welcome to the world of C language");

.

e) A *switch statement* is a selection statement that lets you transfer control to different statements within the switch body depending on the value of the switch expression. The switch expression must evaluate to an integral or enumeration value. The body of the switch statement contains *case clauses* that consist of

- A case label
- An optional default label
- A case expression
- A list of statements.

If the value of the switch expression equals the value of one of the case expressions, the statements following that case expression are processed. If not, the default label statements, if any, are processed.

f) Ordinary variable store one value at a time.

Arrays are used for storing more than one value at a time in a single variable name
Array is the set of an multiple values where as variable can store single value at a time.

• The difference between the definition of array and ordinary variable is the, array is always declared, initialized, and accessed using subscript whereas ordinary variable do not have any subscript.

• The syntax for ordinary variable definition is data_type v1, v2, ….;

• And the syntax for array variable is data_type v1[N1],v2[N2],…;where v1,v2 are name of variable and N1, N2 are the integer constants indicating the maximum size of array.

g) Structure is the collection of variables of different types under a single name for better handling.

struct keyword is used to create structures in C programming
syntax:

```
struct <name>
{
datatype item;
...
};
```

A structure is a collection of variables under a single name. These variables can be of different types, and each has a name which is used to select it from the structure. A structure is a convenient way of grouping several pieces of related information together.

A structure can be defined as a new named type, thus extending the number of available types. It can use other structures, arrays or pointers as some of its members, though this can get complicated unless you are careful.

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

/* function declaration */
void swap(int x, int y);

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;
   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values */
   swap(a, b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
```

i) typedef is a keyword in the C and C++ programming languages. The purpose of typedef is to assign alternative names to existing types, most often those whose standard declaration is cumbersome, potentially confusing, or likely to vary from one implementation to another.

j) In C, functions are global by default. The "*static*" keyword before a function name makes it static. For example, below function *fun()* is static.

```
static int fun(void)
{
  printf("I am a static function ");
}
```

Unlike global functions in C, access to static functions is restricted to the file where they are declared. Therefore, when we want to restrict access to functions, we make them static. Another reason for making functions static can be reuse of the same function name in other files.

For example, if we store following program in one file *file1.c*

```
/* Inside file1.c */
static void fun1(void)
{
  puts("fun1 called");
}
```

And store following program in another file *file2.c*

```
/* Iinside file2.c  */
int main(void)
{
  fun1();
  getchar();
  return 0;
}
```

Ans.2.

Low Level Languages

Low level computer languages are machine codes or close to it. Computer cannot understand instructions given in high level languages or in English. It can only understand and execute instructions given in the form of machine language i.e. language of 0 and 1. There are two types of low level languages:

- **Machine Language.**
- **Assembly Language**

<u>Machine Language:</u> It is the lowest and most elementary level of Programming language and was the first type of programming language to be Developed. Machine Language is basically the only language which computer Can understand. In fact, a manufacturer designs a computer to obey just one Language, its machine code, which is represented inside the computer by a String of binary digits (bits) 0 and 1. The symbol 0 stands for the absence of Electric pulse and 1 for the presence of an electric pulse . Since a computer is Capable of recognizing electric signals, therefore, it understand machine Language.

Advantages of Machine Language

i) It makes fast and efficient use of the computer.

ii) It requires no translator to translate the code i.e.Directly understood by the computer

Disadvantages of Machine Language:

i) All operation codes have to be remembered

ii) All memory addresses have to be remembered.

iii) It is hard to amend or find errors in a program written

In the machine language

iv) These languages are machine dependent i.e. a particular

Machine language can be used on only one type of computer

Assembly Language

It was developed to overcome some of the many

inconveniences of machine language. This is another low level but a very important language in which operation codes and operands are given in the form of alphanumeric symbols instead of 0's and l's. These alphanumeric symbols will be known as mnemonic codes and can have maximum up to 5 letter combination e.g. ADD for addition, SUB for subtraction, START,LABEL etc. Because of this feature it is also known as 'Symbolic Programming Language'. This language is also very difficult and needs a lot of practice to master it because very small

English support is given to this language. The language mainly helps in compiler orientations. The instructions of the Assembly language will also be converted to machine codes by language translator to be executed by the computer.

Advantages of Assembly Language

i) It is easier to understand and use as compared to machine language.

ii) It is easy to locate and correct errors.

iii) It is modified easily

Disadvantages of Assembly Language

i) Like machine language it is also machine dependent.

ii) Since it is machine dependent therefore programmer Should have the knowledge of the hardware also.

High Level Languages

High level computer languages give formats close to English language and the purpose of developing high level languages is to enable people to write programs easily and in their own native language environment (English). High-level languages are basically symbolic languages that use English words and/or mathematical symbols rather than mnemonic codes. Each instruction in the high level language is translated into many machine language instructions thus showing one-to-many translation

Types of High Level Languages

Many languages have been developed for achieving different variety of tasks, some are fairly specialized others are quite general purpose.

These are categorized according to their use as

a) Algebraic Formula-Type Processing. These languages are oriented towards the computational procedures for solving mathematical and statistical problem

Examples are

- **BASIC (Beginners All Purpose Symbolic Instruction Code).**
- **FORTRAN (Formula Translation).**
- **PL/I (Programming Language, Version 1).**
- **ALGOL (Algorithmic Language).**
- **APL (A Programming Language).**

b) Business Data Processing:

- These languages emphasize their capabilities for maintaining data processing procedures and files handling problems. Examples are:

- **COBOL (Common Business Oriented Language).**
- **RPG (Report Program Generator**

b) String and List Processing: These are used for string manipulation including search for patterns, inserting and deleting characters. Examples are:

- **LISP (List Processing).**
- **Prolog (Program in Logic).**

Object Oriented Programming Language

In OOP, the computer program is divided into objects. Examples are:

- **C++**
- **Java**

e) **Visual programming language**: these are designed for building Windows-based applications Examples are:

- **Visual Basic**
- **Visual Java**
- **Visual C**

Advantages of High Level Language

Following are the advantages of a high level language:

- User-friendly
- Similar to English with vocabulary of words and symbols
- Therefore it is easier to learn.
- They require less time to write.
- They are easier to maintain.
- Problem oriented rather than 'machine' based.
- Program written in a high-level language can be translated into many machine language and therefore can run on any computer for which there exists an appropriate translator.
- It is independent of the machine on which it is used i.e.Programs developed in high level language can be run on any Computer

Disadvantages of High Level Language

- A high-level language has to be translated into the machine language by a translator and thus a price in computer time is paid.
- The object code generated by a translator might be inefficient Compared to an equivalent assembly language program
- Types of computer languages
  As we human beings communicate with each others in different language such as Urdu, French, Punjabi and Arabic etc. Similarly to communicate with the computers we have to use specific languages and for this...

Ans. 3.

```
#include<stdio.h>
int main(){
    int x,y;
    printf("Enter value for x :");
    scanf("%d",&x);
    printf("Enter value for y :");
    scanf("%d",&y);
    if ( x > y ){
        printf("X is large number - %d\n",x);
    }
    else{
        printf("Y is large number - %d\n",y);
    }
    return 0;
}
```

```
   /* C program to demonstrate the working of switch...case statement */

/* C Program to create a simple calculator for addition, subtraction,
   multiplication and division */

# include <stdio.h>
int main() {
    char o;
    float num1,num2;
    printf("Select an operator either + or - or * or / \n");
    scanf("%c",&o);
    printf("Enter two operands: ");
    scanf("%f%f",&num1,&num2);
    switch(o) {
        case '+':
            printf("%.1f + %.1f = %.1f",num1, num2, num1+num2);
            break;
        case '-':
            printf("%.1f - %.1f = %.1f",num1, num2, num1-num2);
            break;
        case '*':
            printf("%.1f * %.1f = %.1f",num1, num2, num1*num2);
            break;
        case '/':
            printf("%.1f / %.1f = %.1f",num1, num2, num1/num2);
            break;
        default:
            /* If operator is other than +, -, * or /, error message is shown */
            printf("Error! operator is not correct");
            break;
    }
    return 0;

}
```

Ans. 4. #include <stdio.h>

```
int main()
{
  int m, n, c, d, first[10][10], second[10][10], sum[10][10];

  printf("Enter the number of rows and columns of matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter the elements of first matrix\n");

  for ( c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
      scanf("%d", &first[c][d]);

  printf("Enter the elements of second matrix\n");

  for ( c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
        scanf("%d", &second[c][d]);

  for ( c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
      sum[c][d] = first[c][d] + second[c][d];

  printf("Sum of entered matrices:-\n");

  for ( c = 0 ; c < m ; c++ )
  {
    for ( d = 0 ; d < n ; d++ )
      printf("%d\t", sum[c][d]);

    printf("\n");
  }

  return 0;
}
```

Ans. 5 In any programming language it is vital to learn file handling techniques. Many applications will at some point involve accessing folders and files on the hard drive. In C, a stream is associated with a file. Special functions have been designed for handling file operations. Some of them will be discussed in this chapter. The header file stdio.h is required for using these functions.

Opening a file

Before we perform any operations on a file, we need to open it. We do this by using a file pointer. The type *FILE* defined in stdio.h allows us to define a file pointer. Then you use the function *fopen()* for opening a file. Once this is done one can read or write to the file using the *fread()* or *fwrite()* functions, respectively. The fclose() function is used to explicitly close any opened file.

Taking the preceding statements into account let us look at the following example program :

```
#include <stdio.h>
  main ()
  {
      FILE *fp;
      fp = fopen("data.txt", "r")
      if (fp == NULL) {
        printf("File does not exist,
please check!\n");
      }
      fclose(fp);
  }
```

fopen()

Let us first discuss *fopen()*. This function accepts two arguments as strings. The first argument denotes the name of the file to be opened and the second signifies the mode in which the file is to be opened. The second argument can be any of the following:

File Mode	Description
r	Open a text file for reading
w	Create a text file for writing, if it exists, it is overwritten.
a	Open a text file and append text to the end of the file.
rb	Open a binary file for reading
wb	Create a binary file for writing, if it exists, it is overwritten.
ab	Open a binary file and append data to the end of the file.

fclose()

The *fclose()* function is used for closing opened files. The only argument it accepts is the file pointer.

If a program terminates, it automatically closes all opened files. But it is a good programming habit to close any file once it is no longer needed. This helps in better utilization of system resources, and is very useful when you are working on numerous files simultaneously. Some operating systems place a limit on the number of files that can be open at any given point in time.

fscanf() and fprintf()

The functions *fprintf()* and *fscanf()* are similar to *printf()* and *scanf()* except that these functions operate on files and require one additional and first argument to be a file pointer.

```
#include <stdio.h>
main ()
{
  FILE *fp;
  float total;
  fp = fopen("data.txt", "w+")
  if (fp == NULL) {
    printf("data.txt does not exist, please check!\n");
    exit (1);
  }
  fprintf(fp, 100);
  fscanf(fp, "%f", &total);
  fclose(fp);
  printf("Value of total is %f\n", total);
}
```

getc() and putc()

The functions *getc()* and *putc()* are equivalent to *getchar()* and *putchar()* functions which you studied in Chapter 12 on Input and Output, except that these functions require an argument which is the file pointer. Function *getc()* reads a single character from the file which has previously been opened using a function like *fopen()*. Function *putc()* does the opposite, it writes a character to the file identified by its second argument. The format of both functions is as follows :

```
getc(in_file);
putc(c, out_file);
```

Note: The second argument in the *putc()* function must be a file opened in either write or append mode.

```
#include <stdio.h>
main ()
{
  char in_file[30], out_file[30];
  FILE *fpin, *fpout;
  int c;
  printf("This program copies the source file to the destination file
\n\n");
  printf("Enter name of the source file :");
  scanf("%30s", in_file);
  printf("Enter name of the destination file :");
  scanf("%30s", out_file);
  if((fpin=fopen(in_file, "r")) == NULL)
    printf("Error could not open source file for
reading\n");
  else if ((fpout=fopen(out_file, "w")) == NULL)
    printf("Error could not open destination file for
reading\n");
  else
   {
    while((c =getc(fpin)) != EOF)
      putc(c, fpout);
    printf("Destination file has been copied\n");
  }
}
```