Examination Paper, 2016

[NCT]

Time Allowed : 3 *Hours*] [*Maximum Marks* : 100

General Instructions :

1. Programming Language: Section 'A' is common for C++ & Python.
2. Programming Language: Section 'B' only for Python candidates.
4. Programming Language: Section 'C' only for C++ candidates.

Section A

1. (*a*) Give one example of mainframe computer and also give a situation of its usage in real life. **1**

(*b*) Write differences between customised application software and general application software. **1**

(*c*) What is virus ? Name any type of it. **1**

(*d*) What is Preemptive Scheduling ? Explain any two scheduling techniques. **2**

(*e*) Convert $(B2B)_{16}$ into octal. **2**

(*f*) Explain EPIC and any two major properties. **1**

(*g*) Suppose a number system has been designed with radix 16 with symbols (ordered from small to large) A, B, C, D, E, F, G H, I, J, K, L, M, N, O, P. Convert the following i.e. $(MAGNIFIED)_{16}$ hybrid number to equivalent **binary number.** **2**

Ans. (*a*) **Examples of mainframe computers:** IBM-3000 series, IBM 4300, HP 9000 - 8705/400. The mainframe systems are used in: Airline booking systems, broadcasting, advertising and sales systems.

(*b*) **Customized application software:** The software that is developed for a particular customer or organization is called customized application software. For example, School and College Management Software, Hospital Management Software, etc.

General application software: It is a program or group of programs designed for end users for a specific purpose. Some examples of application software are: MS-Office, Payroll Systems, Financial Accounting, Word Processing, etc.

(*c*) A virus is a computer program or scripts that can negatively affect the functioning of your computer, *i.e.*, it can create files, move files, erase files, consume your computer's memory and cause your computer not to function correctly. Some examples of computer viruses are: *Trojan Horse, Worms, Boot Sector Virus, Macro Virus and Memory Resident Virus, etc.*

(*d*) Preemptive scheduling is a scheduling mechanism where some running jobs can be interrupted by other running jobs because tasks are usually assigned with priorities. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution. The most common example of preemptive scheduling is *round robin scheduling.*

Two scheduling techniques are:

- **Shortest-Job-First Scheduling (SJFS):** In this scheduling, all the processes are arranged according to their size means how much time a process requires CPU for execution. Here, CPU arranges all their processes according to the requirement time.
- **Round Robin Scheduling:** In this scheduling, the CPU time is divided equally for the processes, *i.e.*, each process which is requested for execution will consume the equal amount of time of the CPU and after that quantum time of first process, the CPU will automatically go to the next process.

(*e*) The binary equivalent of $(B2B)_{16}$ is:

Hexa-decimal	B	2	B
Binary	1011	0010	1011

Hence, $(B2B)_{16} = (101100101011)_2$

The octal equivalent of $(101100101011)_2$ is

Binary	101	100	101	011
Octal	5	4	5	3

Hence, $(101100101011)_2 = (5453)_8$

Hence, $(B2B)_{16} = (101100101011)_2 = (5453)_8$

(*f*) EPIC (Explicitly Parallel Instruction Computing) is a 64-bit microprocessor instruction set invented by Intel and Hewlett-Packard, and designed to be used on Intel's Itanium processor. The architecture enables higher levels of instruction-level parallelism without unacceptable hardware complexity. The two properties are:

- EPIC technology provides up to 128 general and floating point unit registers.
- Also, it has massive Hardware Resources to support parallel execution.

(*g*) According to the number system designed with radix 16, the new numbers are:

Hexa-decimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
New Numbers	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P

The binary equivalent of $(MAGNIFIED)_{16}$ is:

Hybrid Number	M	A	G	N	I	F	I	E	D
Hexa-decimal	C	0	6	D	8	5	8	4	3
Binary	1100	0000	0110	1101	1000	0101	1000	0100	0011

Hence, $(MAGNIFIED)_{16} = (110000000110110110000101100001000011)_2$

Section C

2. (*a*) Write difference between Register and bus. **2**

(*b*) List any two devices that are connected inside the computer case. **1**

(*c*) What are Fundamental data types? **1**

(*d*) Differentiate between Run Time Error and Syntax Error. Also give suitable example. **3**

(*e*) Consider the following C++ statements. Are they equivalent? Give reason, if any.

```
char grade = 65;
char grade = 'A';
```

(*f*) Write a C++ program that accepts a character between j to q and prints its previous 4 characters. **2**

(*g*) Write the corresponding C++ expressions for the following mathematical expressions: **1**

(*i*) $2 - ye^{-2y} + 4y$

(*ii*) $|e^x - x|$

(*h*) (*i*) Determine the output: **2**

```
int a =10,b;
b= 11 + ++a;
cout<<b<<a++<<a<<++a;
```

What will be the output when the following code is executed ?

(*ii*) Write declaration for **1**

(*a*) a pointer to a character and an array of 10 integers.

Ans. (*a*) The differences are:

- A register is a temporary unit of memory in the CPU whereas a bus is an electronic path (set of wires) to connect various components and motherboard together.
- A register holds data/information temporarily for fast processing whereas a bus transfers data/information between multiple hardware components in the form of electronic signals.
- A register is made up of semiconductor devices whereas a bus is made up of wires.

(*b*) The two devices are: motherboard, power supply unit, central processing unit (CPU), etc.

(*c*) The basic data types which are implemented directly by a computer language are called fundamental data types. For example, we think about such a language called C++, then the fundamental data types are: bool, char, int, float and double.

(*d*) The differences between run-time error and syntax error are as follow:

- Runtime error can be found only when you are trying to execute your program, whereas syntax error can be found during compilation.

- Runtime errors are caused by incorrect usage of programming logic. For example, a runtime divide method will throw a run time error if the divisor is '0' because numerically you cannot divide a number by 0 whereas syntax errors are those caused by incorrect usage of the programming language syntax. For example, incorrect use of parentheses, comma, curly braces, etc.

(*e*) The two statements are equal and they have the same effect.

The first statement,

```
char grade = 65;
```

is a character type value and 65 is stored in memory.

The second statement

```
char grade = 'A';
```

is declared as an ASCII character constant 'A' whose value is 65.

So, in both the cases, the memory is stored as 65.

(*f*) The program is:

```
#include <iostream.h>
#include<ctype.h>
void main()
{
    char ch;
    cout << "Enter any character: ";
    cin >> ch;
    ch = tolower(ch);
    if ((ch>=106) && (ch <=113)) {
        cout << "Previous characters are: ";
        ch = ch - 1;
        for (int i=1; i<=4; i++) {
            cout << ch << " ";
            ch = ch - 1;
        }
    } else
        cout << "Invalid input!";
}
```

(*g*) (*i*) 2 – y*e(pow(2, –2*y) + 4*y or 2 – y* exp (–2*y) + 4*y

(*ii*) abs(pow(e, x) - x) or abs (exp(x) – x)

(*h*) (*i*) The output is: 22111213

(*ii*) The declaration of a pointer to a character is: char *ch;

The declaration of an array of 10 integers is: int N[10];

3. (*a*) Define Prettyprinting. **1**

(*b*) Explain any three types of Program Maintenance. **3**

(*c*) What type of information is being provided by a user documentation or manual ? Explain any three. **3**

(*d*) Give the output of the following program: **3**

```
#include<iostream.h>
struct Package
{   int Length, Breadth, Height;
}
void Occupies(PackageM)
{   cout<<M.Length<<"x"<<M.Breadth<<"x";
    cout<<M.Height<<endl;
}
int main( )
{   Package P1={100,150,50}, P2, P3;
    ++P1. Length;
Occupies(P1);
P3=P1;
++P3.Breadth;
P3.Breadth++;
Occupies(P3);
P2=P3;
P2.Breadth+=50;
P2. Height--;
Occupies(P2);
return 0;
}
```

(*e*) Write an algorithm for finding the simple interest. **2**

[S.I = (Principal Amount * Rate of Interest * No. ofyears)/100]

Ans. (*a*) Pretty Printing or pretty printers are the language tools or applications which are used to beautify the programming source code. For example, these are like syntax highlighting (bold, *italics*, and different fonts or colors), tabs, brackets, etc.

(*b*) Three types of program maintenance are:

- **Fix:** It is not always possible to find all errors during the testing stage. When the system is put into use some errors may arise. These errors are to be fixed.
- **Enhance:** The system is to be expanded to take care of new requirements.
- **Adapt:** The system is to be exposed to different environments.

(*c*) The information provided by the user documentation or manual are:

- To describe the design of the program like diagrams, illustrations, text, etc.
- To tells other programmers about set rules about what the program does and how it works.
- Maintenance can be easily done by someone other than the original programmer.

(*d*) The output is:

101x150x50

101x152x50

101x202x49

(*e*) The algorithm is as follows :

Step 1. Start

Step 2. Read Principal Amount, Rate of Interest and No. of years

Step 3. SI = (principal amount * rate of interest * number of years) / 100

Step 4. Write SI

Step 5. Stop

4. (*a*) (*i*) Give the output of the following below code: **2**

```
#include<iostream.h>
int main()
{   int i= 1, ua = 1, ub = 1, uc = 0, fail = 0;
while(i< = 5)
{   switch(i++)
        {
    case 1;
    case 2;++ua;
    case 3:
    case 4: ++ub;
    case 5: ++uc;
            default: ++fail;
            }
        }
    cout<< "ua="<<ua<<"\t"<< "ub="<<ub<<endl;
    cout<< "uc="<<uc<<"\t"<< "fail="<<fail<<endl;
    return 0;
}
```

(*ii*) Write a function using for loop to generate the following series: **2**

1,2,4,8,16,32,64,128

(Note : The output should come exactly which has been given above i.e. including "," (comma) also)

(*iii*) Give an example in C++ of goto statement. **2**

(*iv*) Illustrate the difference between break and continue statement with the help of C++ program. **2**

Ans. (*a*) (*i*) The output is:

ua=3 ub=5

uc=5 fail=5

(*ii*) The function is:

```
void Print_Series() {
    for (int num = 1; num <= 128; num *= 2)
        if (num != 128)
            cout << num << ",";
        else
            cout << num;
}
```

(*iii*) The example demonstrates that when the value of num will be 5, then it will repeat the loop without printing num value.

```
int main ()
{
    int num = 1;
    STEP:
    do {
        if( num == 5) // value 5 will not print
        {
                num = num + 1;
                goto STEP;
        }
        cout << "value of num : " << num << endl;
        num = num + 1;
    }while( num < 10 );
    return 0;
}
```

(*iv*) The differences are:

- The **break** statement provides immediate termination of the entire loop body (i.e. for, while, do-while) whereas **continue** statement forces the computer to perform another iteration of the loop skipping any code following the **continue** statement in the loop body.

Example of break statement:

```
Int Num = 0;
// The following do-while loop will be terminated when Num will be more than 5.
```

```
do {
    cout << " Num : " << Num << endl;
    Num ++;
    if(Num > 5) {
        // Terminate the loop
        break;
    }
} while(Num < 20 );
```

Example of continue statement:

// In the following do-while loop, when Num = 6 then both conditions becomes true and thus continue statement gets executed without printing Num value 6.

```
int Num = 0;
do {
    Num ++;
    if(Num > 5 && Num < 7)
        continue;
    cout << " Num: " << Num << endl;
}while( count < 10 );
```

5. (*a*) Write difference between the formal and actual parameters. Also, give a suitable C++ code to illustrate both. **3**

(*b*) What is Function Prototype? **1**

(c) Write a user defined function in C++ to display the sum of column elements of a two dimensional array R[7][7] containing integers. **3**

(*d*) In the following program, find the correct possible output(s) from the options: **2**

```
#include<iostream.h>
#include<stdio.h>
void main()
{   randomize();
char city[][10] = {"75", "85", "95", "105", "115"};
int Fly;
for(int i=0;i<3;i++)
{   Fly=random(2)+1;
cout<<city[Fly]<< ":";
    }
}
```

Outputs:

(*i*) 75:85:95:

(*ii*) 85:95:85:

(*iii*) 95:85:95:

(*iv*) 95:105:115:

(*e*) What will be the output of the following program? **1**

```
#include<iostream.h>
void main()
{   int a, b = 0;
    int c[10] = {1,2,3,4,5,6,7,8,9,0};
    for(a=0; a<10; ++a)
        b+=c[a];
        cout<<b;
}
```

Ans. (*a*) The differences are:

- **Formal parameter:** Formal parameters are written in the function prototype and function header of the definition. Formal parameters are local variables which are assigned values of actual variables when the function is called.
- **Actual parameter:** When a function is *called*, the values (expressions) that are passed in the function call are called the *arguments* or *actual parameters* (both terms mean the same thing). At the time of the function call, each actual parameter is assigned to the corresponding formal parameter in the function definition.

  For example,

```
void CalcVal(int Num)// Num is a formal parameter
{
    cout << 10*Num;
}
void main()
{
    int N = 20;
    CalcVal(N);  // N is actual parameter
}
```

Here, the value of N is passed to the variable Num.

(*b*) A function prototype declares the name, return-type of the function and declares the number, the types and order of the parameters, the function expects to receive. The function prototypes enable the compiler to verify that functions are called correctly.

(*c*) The function is:

```
void COLSUM(int R[7][7], int M, int N) {
    int SUMC;
    for(int j = 0; j<M; j++) {
        SUMC=0;
```

```
        for (int i=0; i<N; i++)
            SUMC = SUMC + R[i][j];
        cout << "Sum of Column " << j << " = " << SUMC << endl;
    }
}
```

(*d*) The possible outputs are: (*ii*) and (*iv*)

(*e*) The output is: 45

6. (*a*) This matrix is named as Scores: **2**

86	75	95
67	88	93
95	79	83
86	98	82
74	68	90

What is stored in each of the following cells?

Scores[0][0] = ...

Scores[2][1] = ...

Scores[3][2] = ...

Scores[5][2] = ...

(*b*) What is a structure ? Give one example. **2**

(*c*) Write a program to find total number of vowels, consonants, digits and other characters in string. **4**

(*d*) Write a program in C++ to create a 3 × 3 matrix and store the first nine natural numbers in it row wise e.g.: **4**

1 2 3

4 5 6

7 8 9

Now write functions to print the following output:

(*i*) Sum of the numbers in each row.

(*ii*) Sum of the numbers in each column.

(*e*) Write a function to find the sum of series: **4**

1 + 1/3 + 1/5 + 1/7 + 1/9 + ... upto 1/N terms.

Ans. (*a*) Scores[0][0] = 86

Scores[2][1] = 79

Scores[3][2] = 82

Scores[5][2] = 90

(*b*) A structure is a group of variables of different data types grouped under one name. The variables also called the data members of the structure. A structure is declared with the keyword of struct. For example,

```
struct person
{
    char name[20];
    int age;
    char design[15];
    float salary;
}P1, P2, P3;
```

Here, name, age, design and salary are the data members of structure person. P1, P2, P3 are the structure variables of type person.

(*c*) The program is:

```
// Program to find total number of vowels, consonants, digits and other characters in a string
#include <iostream.h>
#include <conio.h>
#include <ctype.h>
#include <stdio.h>
void main()
{
    char str[50];
    int i = 0, vc= 0, dc = 0, ot = 0, con = 0;
    clrscr();
    cout << "\n\tEnter any string -> ";
    gets(str);
    while (str[i] != '\0')
    {
        str[i] = tolower(str[i]);
            if (isalpha(str[i]))
            {
                if ((str[i] == 'a') || (str[i] == 'e') || (str[i] == 'i') || (str[i] == 'o') ||
                (str[i] == 'u'))
                    vc++;
                else
                    con++;
            }
                else
                if ((str[i] >= '0') && (str[i] <= '9'))
                    dc++;
                else
```

```
                    ot++;
                    i++;
        }
        cout << "\n\tNumber of vowels are : " << vc;
        cout << "\n\tNumber of consonants are : " << con;
        cout << "\n\tNumber of digits are : " << dc;
        cout << "\n\tNumber of other characters are : " << ot;
    }
```

(*d*) The program is as follows:

```
    \\ The program to create 3 × 3 matrix
    #include<iostream.h>
    void main() {
        int R[3][3];
        int i, j, Num = 1, RowSum = 0, ColSum = 0;
        for (i = 0; i < 3; i++) {
            for (j = 0; j < 3; j++) {
                R[i][j] = Num;
                Num++;
            }
        }
        cout << "The matrix is:\n";
        for (i = 0; i < 3; i++) {
            for (j = 0; j < 3; j++) {
                cout << R[i][j] << "    ";
        }
        cout << endl;
    }
    \\ program to calculate sum of the numbers in each row
(i) for(i = 0; i < 3; i++) {
        RowSum = 0;
        for(j = 0; j < 3; j++) {
            RowSum = RowSum + R[i][j];
        }
        cout << "\nRow 1 sum is " << RowSum;
    }
    \\ program to calculate sum of the numbers in each column
(ii) for (i = 0; i < 3; i++) {
        ColSum = 0;
        for (j = 0; j < 3; j++)
                ColSum = ColSum + R[j][i];
            cout << "\nColumn 1 sum is " << ColSum;
```

```
    }
}
```

(*e*) The function is as follows:

```
// 1 + 1/3 + 1/5 + 1/7 +...+ 1/n
float sumseries_odd(int N) {
    int i = 1;
    float term, sum=0.0;
    while (i <= N) {
        term = 1 / float(i);
        sum = sum+term;
        i = i + 2;
    }
    return(sum);
}
```

Examination Paper, 2015
[NCT]

Maximum Marks: 70] [*Duration: 3 Hours*

General Instructions: (*i*) *All questions are compulsory.* (*ii*) *Programming Language: C++.*

1. (*a*) Explain briefly : **1 + 1**

(*i*) RAM (*ii*) ROM

(*b*) When a folder is copied to another place, do the subfolders in the folder also get copied ? **1**

(*c*) Explain briefly the concept of time-sharing. **2**

(*d*) What is the base of octal, decimal and binary numbers ? **1**

(*e*) Convert $(3674)_8$ via binary to hexa-decimal. **1 + 1**

(*f*) What are the two categories of printer ? Name them. **1 + 1**

Ans. (*a*) (*i*) RAM (Random Access Memory) is the main memory of a computer and is used to retain users' instructions and data for processing purposes. This is temporary in nature. RAM chip is made with metal-oxide semiconductor (MOS).

(*ii*) ROM (Read Only Memory) is an essential component of memory unit. This memory is permanent and is not erased when the system is switched off. As its name suggests, it is read only memory, *i.e.*, it can be read only and not be re-written or modified by the user.

(*b*) Yes.

(*c*) Time-sharing is a form of resource sharing through multiprogramming and multitasking processes. In this, operating system operates in an interactive mode with a quick response time. The user types a request to the computer through a keyboard. The computer processes it and a response is displayed on the user's terminal. A time-sharing system allows many users to share the computer resources simultaneously. Some examples of time-sharing operating systems are : *Linux* and *Unix*.

(*d*) The base of octal number is 8.

The base of decimal number is 10.

The base of binary number is 2.

(*e*) The binary equivalent of $(3674)_8$ is :

Octal	3	6	7	4
Binary	011	110	111	100

Hence, $(3674)_8 = (011110111100)_2$

The hexa-decimal equivalent of $(011110111100)_2$ is :

Binary	0111	1011	1100
Hexa-decimal	7	B	C

Hence, $(011110111100)_2 = (7BC)_{16}$

(*f*) The two categories of printers are : impact and non-impact.

2. (*a*) Explain the following : **1 + 1 + 1**

(*i*) Source code (*ii*) Object code (*iii*) Syntax

(*b*) How many types of errors can occur in a program at the time of compilation or execution ? Name and explain them. **5**

(*c*) Why are logical errors harder to locate ? **2**

(*d*) Why we use comments and indentation in a program ? **2**

Ans. (*a*) (*i*) **Source code.** The program code which is written by the programmer using a high-level language is called source program/code. The source program is human readable code and computer does not directly understand it.

(*ii*) **Object code.** The program code which is produced by using any program like compiler, interpreter or assembler is called object program/code. The object program is machine-oriented and the computer can understand it.

(*iii*) **Syntax.** The basic rule that must be followed to write a program code is called syntax.

(*b*) There are three types of errors can occur at compilation or execution time. These are : syntax errors, run-time errors and linker errors.

Syntax error. It is an error that you can make when you are writing lines of code for your program and you make spelling errors. This is an error of language resulting from code that does not conform to the syntax of the programming language. Syntax errors can be recognized at the compilation time. For example, missing of comma, parentheses, invalid character, invalid keywords, etc.

Run-time error. A run-time error is an error that causes abnormal program termination during execution. For example, a divide method will throw a run-time error, if the divisor is '0' because numerically you cannot divide a number by 0.

Linker error. Even if a program code compiles fine, but sometimes the program does not found some functions or libraries. This type of error is called linker error.

For example :

```
#include <iostream.h>
void TryMe();
void main() {
    TryMe(); //Linker Error: Undefined symbol TryMe() in module TEST.CPP
}
```

```
void tryMe() {
    int Tmarks = 450                    // Declaration Syntax error
    int Ctr = 0;
    float Average = Tmarks / Ctr;       // Runtime error: Divide error
    cout << Average;
}
```

(*c*) Logical errors are harder to locate because incorrect algorithms result in unexpected output. Also, these types of errors are not located easily due to lack of internal flow of objects in the program code.

(*d*) For improving the understanding of the program, it is important to give self-explanatory comments at the proper places, which explain the programming logic and also help to remove the errors that may occur.

Indentation makes a program or a function clear and readable. In program like Python, indentation is used to delimit blocks as there are no braces to indicate blocks of code for class and function definitions or flow controls. Similarly, in C++ programming language we use curly brackets ({ }) to indicate blocks of code.

3. (*a*) What is the difference between a keyword and an identifier ? **3**

(*b*) (*i*) What do you mean by operator precedence ? **2**

(*ii*) Can nongraphic characters be used and processed ? Give example to support your answer. **2**

(*c*) Explain the difference between value and variable. (Give examples) **3**

(*d*) What is the purpose of using Input and Output statements in a program ? **2**

(*e*) Explain different types of data types which can be used while programming. **2**

(*f*) (*i*) How can a programmer invoke a built-in function into a program ? **2**

(*ii*) Write any four built-in function's names ? **2**

Ans. (*a*) **Keyword.** The words which have predefined meaning to its interpreter or compiler are called keywords. Keywords are also called reserved words.

For example, *while*, *continue*, *break*, etc.

Identifier. An identifier is an arbitrarily long sequence of letters and digits. They are used to give names to program elements such as variables, arrays and functions etc.

For example, emp_name, Number10, RouteID, etc., are identifiers which are used to identify a memory location.

(*b*) (*i*) The order in which operators in an expression are evaluated is called operator precedence. Operator precedence also determines the grouping of terms in an expression and tells us which operators are evaluated first. This affects how an expression is evaluated. For example, you might write :

5 + 2 * 3

The answer is 11 because like in mathematics, the multiplication operator has a higher precedence than addition operator.

Or if you write,

(5 + 2) * 3

Then the answer is 30 because the change of order of evaluation produces different result. Expressions inside brackets are always evaluated first.

(*ii*) Yes, nongraphic characters can be used and processed. Nongraphic characters which cannot be typed directly from keyboard, e.g., backspace, tab, carriage return etc. These characters can be represented by using an escape sequence. An escape sequence represents a single character. For example, \a for audible bell (beep), \b for backspace, \f for form feed, etc.

(*c*) A variable is the most fundamental aspect of any computer language. It is a location in the computer memory which can store data and is given a symbolic name for easy reference. For example, myNum, Opt, a_Number, BasicSalary, AString, etc.

A value is the data which is assigned to a variable. A value can be of any data type like integer, float, string, array, etc. A variable can also be an expression. Generally, a variable is assigned by a value which is placed on the right side of an assignment operator (=).

For example :

```
int count = 1;
int i = 0, j = 0, k = 0;
char Name[20];
float total = 0, Net = 0;
long int sal;
```

(*d*) The input statements are used to feed new data into the computer.

In C++, we use cin (console input), getch(), getche(), gets() to take input from the computer.

For example :

```
int rollno;
char name[20];
cin>>rollno;
gets(name);
```

The output statements are used to obtain output on an output device such as : VDU, printer, etc. It provides both formatted and unformatted stream of I/O statements.

In C++, we use cout (console output) to display output on screen.

For example :

```
cout<<"Your roll no. is : "<<rollno;
cout<<"Your name is : "<< name;
```

(*e*) Different primitive data types which can be used in C++ while programming are integer, floating point, character type and boolean type.

- **Integer type (int).** An integer is an integral whole number without a decimal point. Normally an integer can hold numbers from -32768 to 32767. For example : 126, 173, -3549 are valid integers.
- **Floating-point (float).** A floating-point number has a decimal point.

  For example : 126.65, 173.7, -3549.05 are valid floating-point numbers.
- **Character type (char).** It is a non-numeric data type consisting of single alphanumeric character. For example : 'A', '9', 'P', '8', '&' are valid characters.
- **Boolean type (bool).** The boolean data type, known as bool, which represent one of two states, true or false. For example : 0 and 1.

(*f*) (*i*) A program can invoke a built-in function directly to its program by using the library. For example, to invoke a function called isalpha(), we must include ctype.h header file.

(*ii*) Four built-in functions are : islower(), isupper(), sqrt(), ceil(), etc.

4. (*a*) Write a program to input a number. If the number is even, print its square, otherwise print its cube. **4**

(*b*) How many types of loops are used in a program ? Explain them briefly with examples. **4**

(*c*) What do you mean by Nested loops ? Explain with example. **2 + 2**

(*d*) Explain briefly the following :

(*i*) Entry-Control loop **2**

(*ii*) Empty Statement **2**

(*e*) What is the difference between 'Break' and 'Continue' ? Explain with examples. **4**

Ans. (*a*) The program is as follows :

```
#include <iostream.h>
void main() {
    int N;
    cout<< "Enter a number: ";
    cin>> N;
    if (N % 2 == 0)
        cout<< "The square of " << N << " is " << N*N;
    else
        cout<< "The cube of " << N << " is " << N*N*N;
}
```

(*b*) In C++, we use three types of loops. These are : *while*, *do-while* and *for*.

- ***while* loop.** This is an entry-control construct which repeats the body of the loop as long as the loop condition holds true.

  The syntax is :

```
while (test expression)
{
    program statements;
}
```

  For example :

```
i = 1;
while (i <= 10)
{
    fact = fact * i;
    i++;
}
```

- ***do-while* loop.** This is an exit-control loop or a post-tested loop. It performs the conditional test after the body of the loop is executed.

  The syntax is :

```
do{
    program statements;
} while (test expression);
```

  For example :

```
int value;
do {
    cout<< "Enter the number to be reversed. ";
        cin>> value;
    if (value <= 0)
        cout<< "The number must be positive. \n";
} while (value <= 0);
```

- ***for* loop.** The statements in *for* loop repeat continuously for a specific number of times until a specific count is met.

  The syntax is :

```
for (initialization expression(s); text-expression; update expression)
    program statements;
```

  For example :

```
int ctr;
for (ctr = 1; ctr<= 10; ctr = ctr + 1)
{
    cout<<ctr<< ' ';
}
```

(*c*) When one loop is executed inside another loop then we call it a nested loop. The nested loops are implemented using *while, do-while* and *for* loop. For example, to print a pattern as given below using nested *for* loops :

```
1
22
333
4444
55555
```

The code using *for* loop is :

```
for(loop = 1; loop <= 5; loop = loop + 1)
{
    for(count = 1; count <= loop; count  = count + 1)
        cout<< loop;
    cout<< "\n";
}
```

(*d*) (*i*) **Entry-Control loop.** The *while* loop and *for* loop are known as entry-control loops because it first checks the condition (if *True*) then executes the related block of code. It executes a set of statements repeatedly till condition evaluates to *True*.

For example :

```
i = 1;
while (i <= 10)  // Loop begins with a condition
{
    fact = fact * i;
    i++;
}
```

The above program will execute two statements inside the *while* loop till i <= 10.

(*ii*) **Empty Statement.** An empty statement is a statement which doing nothing. This is basically useful in looping statements.

In C++, the semicolon (;) performs no action or doing nothing.

For example :

```
for (int i = 0; i< 10; ++i);
    ;
```

(*e*) The *break* statement provides immediate termination of the entire loop body whereas *continue* statement forces the program to perform another iteration of the loop by skipping any code following *continue* statement in the loop body.

For example :

The example given below shows the use of *break* and *continue* statements in *while* loop :

```
int line = 0;
char ch;
cin>>ch;
while (ch!='&')
{
        cin>>ch;
        if (ch =='0')
            break;
        if (ch!='\n')
            continue;
        line++;
}
```

5. (*a*) Write a program to find the sum of first 10 odd numbers. **4**

(*b*) Differentiate between the formal and actual parameters. Give suitable codes to illustrate both. **2**

(*c*) Write a program to print table of given number. **4**

Ans. (*a*) The program is as follows :

```
#include <iostream.h>
void main() {
    int Sum = 0, Num = 1, ctr = 1;
    while (ctr<= 10) {
        Sum += Num;
        Num = Num + 2;
        ctr += 1;
    }
        cout<< "Sum of first 10 odd numbers : " << Sum;
    }
```

(*b*) **Formal parameter.** Formal parameters are written in the function prototype and function header of the definition. Formal parameters are local variables which are assigned values of actual variables when the function is called.

Actual parameter. When a function is called, the values (expressions) that are passed in the function call are called the *arguments* or *actual parameters* (both terms having the same meaning). At the time of the call, each actual parameter is assigned to the corresponding formal parameter in the function definition.

For example :

```
#include<iostream.h>
void change(int &x)
{ x++; }
void main() {
        int n = 2;
        change(n);
        cout<<n;
}
```

In the above program, 'n' is called actual parameter and 'x' is called formal parameter.

(*c*) The program is as follows :

```
#include<iostream.h>
void main() {
    int i, j, n;
    cout<< "Enter the number to print table : ";
    cin>> n;
    for (i = 1; i <= 10; i++) {
        j = i * n;
        cout<< n << " * " << i << " = " << j << "\n";
    }
}
```