----------Game Development with C++ ----------

What is C++?

C++ is an object-oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980's.

C++ is a superset of C.

Simple C++ Program ? Printing A String

```
#include <iostream>                    // include header file
 using namespace std;
 int main()
 {
        cout << "C++ is better than C.\n";     // printing C++ statement
        return 0;
 }
```

Compile the program: CC prog1.cpp

The compiler would produce an object file prog1.o and then automatically link with the library functions to produce an executable file. The default executable filename is a.out.

Comments *: single line comment // , multiline comments /*...*/*

Output Operator : *The operator << is called the insertion or put to operator. It inserts (or sends) the contents of the variable on its right to the object on its left. bit-wise left-shift*

operator and it can still be used for this purpose. This is an example of how one operator can be used for different purposes, depending on the context. This concept is known as operator overloading, an important aspect of polymorphism.

Namespace : *Use to avoid Naming Collisions. This defines a scope for the identifiers that are used in a program. std is the namespace where ANSI C++ standard class libraries are defined. using and namespace are the new keywords of C++.*

Example : Average of Two Numbers

```
#include <iostream>
using namespace std;
int main()
{
        float numberl, number2, sum, average;
        cout << "Enter two numbers: "; // prompt
        cin >> numberl; // Reads numbers from keyboard
        cin >> number2; //

        sum = numberl + number2;
        average = sum/2;

        cout << "Sum = " << sum << "\n";
        cout << "Average = " << average << "\n";

         return 0;
}
```

The output of Program would be:

Enter two numbers: 6.5 7.5

Sum = 14

Average = 7

Input Operator : *The operator >> is known as extraction or get from operator. It extracts (or takes) the value from the keyboard and assigns it to the variable on its right.*

Structure of C++ Program :

Include file -------> Class declaration ---------> Member functions Definition --------> Main function program

The client-server model : *This approach is based on the concept of client-server model. The class definition including the member functions constitute the server that provides services to the main program known as client. The client uses the server through the public interface of the class.*

Member functions ---------> Class definition (server) -------> main function program (Client)

Q1. Find errors, if any, in the following C++ statements.

(a) cout << "x=" x;

(b) m = 5; // n = 10; // s = m + n;

(c) cin >>x; >>y;

(d) cout << \n "Name:" << name;

(e) cout <<"Enter value:"; cin >> x;

(f) /*Addition*/ z = x + y;

Q2. Write a C++ program that will ask for a temperature in Fahrenheit and display it in Celsius.

Tokens, Expressions and Control Structures

C++ has the following tokens:

- *Keywords : bool, namespace, mutable*
- *Identifiers*
- *Constants*
- *Strings*
- *Operators*

C++ keywords

Hierarchy of C++ data types

Storage Classes :

The storage class of a variable specifies the lifetime and visibility of a variable within the program. There are four types of storage classes:

1. Automatic:

It is the default storage class of any type of variable. Its visibility is restricted to the function in which it is declared. Further, its lifetime is also limited till the time its container function is executing.

Example : The variable a is local to the function displayAutomatic and is destroyed when the function exits.

```
#include <iostream>

using namespace std;

void displayAutomatic() {
   int a = 10; // Automatic variable
   cout << "Automatic variable a = " << a << endl;
}

int main() {
   displayAutomatic();
   return 0;
}
```

2. External:

As the name suggests, an external variable is declared outside of a function but is accessible inside the function block. Also called global variable, its visibility is spread all across the program that means, it is accessible by all the functions present in the program.

Example : The variable globalVar is declared outside any function, making it accessible to both main() and displayExternal().

```
#include <iostream>

using namespace std;

int gVar = 100; // External variable

void displayExternal() {

   cout << "External variable globalVar = " << gVar << endl;

}

int main() {

   cout << "Accessing global variable in main: " << gVar << endl;

   displayExternal();

   return 0;

}
```

3. Static:

A static variable has the visibility of a local variable but the lifetime of an external variable. That means, once declared inside a function block, it does not get destroyed after the function is executed, but retains its value so that it can be used by future function calls.

Example : The static variable count retains its value between function calls.

```
#include <iostream>

using namespace std;

void staticExample() {

   static int count = 0; // Static variable

   count++;

   cout << "Static variable count = " << count << endl;

}

int main() {

   staticExample(); // First call
```

```
    staticExample(); // Second call
    staticExample(); // Third call
    return 0;
}
```

4. Register:

Similar in behaviour to an automatic variable, a register variable differs in the manner in which it is stored in the memory (register). Unlike, automatic variables that are stored in the primary memory (RAM), the register variables are stored in CPU registers. The objective of storing a variable in registers is to increase its access speed, which eventually makes the program run faster.

Example : The variable num is declared as a register variable. Note that modern compilers manage register allocation automatically, so using register is often ignored.

```
#include <iostream>
using namespace std;
void registerExample() {
    register int num = 5; // Register variable
    cout << "Register variable num = " << num << endl;
}
int main() {
    registerExample();
    return 0;
}
```

Derived Data Types

Character Array Initialization in C++ :

In C, it is permissible to initialize a character array with a size equal to the number of characters in the string constant, excluding the null terminator (\0). For example:

char string[3] = "xyz"; // Valid in C, but \0 is omitted.

In C++, the size of the character array must account for the null terminator.

char string[4] = "xyz"; // Valid in C++

Pointers in C++ :

Memory management (dynamic allocation).

Enables polymorphism in object-oriented programming.

Basic Pointer Declaration:

*int *ip; // Pointer to an integer*

ip = &x; // Assign address of x to ip

**ip = 10; // Modify x indirectly through ip*

Constant Pointer:

*char *const ptr1 = "GOOD"; // Constant pointer*

// Address in ptr1 cannot be changed.

Pointer to a Constant:

*int const *ptr2 = &m; // Pointer to a constant*

// Contents pointed by ptr2 cannot be modified.

Constant Pointer to a Constant:

*const char *const cp = "xyz";*

// Neither address nor the contents can be changed.

Reference Variables :

C++ introduces a new kind of variable known as the reference variable. A reference variable provides an alias (alternative name) for a previously defined variable.

For example, if we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to represent that variable. A reference variable is created as follows:

data-type & reference-name = variable-name

Example 1:

float total = 100;

float & sum = total;

cout << total; and cout << sum; // Both print the value 100.

Example 2:

C++ assigns additional meaning to the symbol &. Here, & is not an address operator. The notation float & means reference to float.

int n[10];

int & x = n[10]; // x is alias for n[10]

char & a = '\n'; // initialize reference to a literal

The variable x is an alternative to the array element n[10]. The variable a is initialized to the newline constant. This creates a reference to the otherwise unknown location where the newline constant \n is stored.

A major application of reference variables is in passing arguments to functions : When the function call f(m) is executed, the following initialization occurs: int & x = m;

Thus, x becomes an alias of m after executing the statement. Such function calls are known as call by reference.

Since the variables x and m are aliases, when the function increments x, m is also incremented. The value of m becomes 20 after the function is executed. In traditional C, we accomplish this operation using pointers and dereferencing techniques.

Operators in C++

All C operators are valid in C++ also.

In addition, C++ introduces some new operators. We have already seen two such operators, namely, the insertion operator <<, and the extraction operator >>

:: Scope resolution operator

:: Pointer-to-member declarator (To declare a pointer to a member of a class)*

-> Pointer-to-member operator (To access a member using a pointer to the object and a pointer to that member)*

. Pointer-to-member operator*

delete : Memory release operator

endl : Line feed operator

new : Memory allocation operator

setw : Field width operator

Scope Resolution Operator

Like C, C++ is also a block-structured language. Blocks and scopes can be used in constructing programs. We know that the same variable name can be used to have different meanings in different blocks. Declaration in an inner block hides a declaration of the same variable in an outer block.

In C, the global version of a variable cannot be accessed from within the inner block. C++ resolves this problem by introducing a new operator called the scope resolution operator.

Example : ::m will always refer to the global m.

```
#include <iostream>
using namespace std;
int m = 10;                    // global m
int main()
{
    int m = 20;                // m redeclared, local to main
      {
       int k = m;
       int m = 30;             // m declared again
      cout << "we are in inner block \n";
      cout << "k = " << k << "\n";      // 20
```

```
        cout << "m = " << m << "\n";    // 30
        cout << "::m = " << ::m << "\n";   // 10
        }
        cout << "\nWe are in outer block \n";
         cout << "m = " << m << "\n";      //  20
        cout << "::m = " << ::m << "\n";   // 10
        return 0;
}
```

Memory Management Operators :

C uses malloc() and calloc() functions to allocate memory dynamically at run time.

C++ supports these functions, it also defines two unary operators new and delete that perform the task of allocating and freeing the memory in a better and easier way.

The new operator allocates sufficient memory to hold a data object of type data-type and returns the address of the object.

The new operator can be used to create objects of any type :

pointer-variable= new data-type;

Example :

```
#include <iostream>
#include<conio.h>
using namespace std;
void main()
{
        int  *arr;
        int size;
```

```
        cout<<"Enter the size of the integer array: ";
        cin>>size;
        cout<<"Creating an array of size "<<size<<"..";
        arr = new int[size];
        cout<<"\nDynamic allocation of memory for array arr is successful.";
        delete arr;
         getch();
}
```

Use of Manipulators :

```
cout << setw(5) << sum << endl;
```

The manipulator setw(5) specifies a field width 5 for printing the value of the variable sum.

Example :

```
#include <iostream>
 #include <iomanip>      // for setw
 using namespace std;
 int main()
 {
        int Basic = 950, Allowance = 95, Total = 1045;
        cout << setw(10) << "Basic" << setw(10) << Basic <<endl<< setw(10)<< "Allowance"
<< setw(10) <<Allowance << endl << setw(10) << "Total" << setw(10) << Total << endl;
        return 0;
 }
```

The output of Program would be:

Basic 950

Allowance 95

Total 1045

Type Cast Operator :

The following two versions are equivalent:

(type-name) expression // C notation

type-name (expression) // C++ notation

Examples:

average = sum/(float)i; // C notation

average = sum/float(i); // C++ notation

Operator Precedence :

Q1. Write a C++ program using those two function squareArea & circle Area to calculate area of square and area of circle using call by reference concept.

```
#include <iostream>
#define PI 3.14

int squareArea(int &a);
int circleArea(int &r);

int main() {
   int a = 10;
   cout << squareArea(a) << " ";
   cout << circleArea(a) << " ";
   cout << a << endl;
   return 0;
}

int squareArea(int &a) {
   a *= a;
   return a;
}

int circleArea(int &r) {
   int area = PI * r * r;
   return area;
}
```

Q2. An election is contested by five candidates. The candidates are numbered 1 to 5 and the voting is done by marking the candidate number on the ballot paper.

Write a C ++ program to read the ballots and count the votes cast for each candidate using an array variable count. In case, a number read is outside the range 1 to 5, the ballot

should be considered as a 'spoilt ballot', and the program should also count the number of spoilt ballots.

Sample Output :

Election Voting System

Enter the candidate numbers for voting.

Enter ballot: 2

Enter ballot: 3

Enter ballot: 3

Enter ballot: 5

Enter ballot: -1

Election Results:

Candidate 1: 0 votes

Candidate 2: 1 votes

Candidate 3: 2 votes

Candidate 4: 0 votes

Candidate 5: 1 votes

Number of spoilt ballots: 2

```
#include <iostream>
using namespace std;
int main() {
   int n= 5;
   int votes[n] = {0};
   int SB= 0;
```

```
    int ballot;


    cout << "Enter votes for candidates :" << endl;
    while (1) {
        cin >> ballot;
        if (ballot == -1) {
            break;
        }
        if (ballot >= 1 && ballot <= n) {
            votes[ballot - 1]++;        // votes[0] for candidate 1
        }
        else {
            SB ++;
        }
    }
    cout << "Election Results:\n";
    for (int i = 0; i < n; i++) {
        cout << "Candidate " << (i + 1) << ": " << votes[i] << " votes" << endl;
    }
    cout << "Spoilt ballots: " << SB << endl;
    return 0;
}
```

Q3. An electricity board charges the following rates to domestic users to discouragelarge consumption of energy:

For the first 100 units - 60P per unit

For next 200 units - 80P per unit

Beyond 300 units - 90P per unit

All users are charged a minimum of Rs. 50.00. If the total amount is more than Rs. 300.00 then an additional surcharge of 15% is added. Write a program to read the names of users and number of units consumed and print out the charges with names.

Sample run :

Enter the name of the user: ABC

Enter the number of units consumed: 450

User: ABC

Units Consumed: 450

Amount: Rs. 390

Total Amount after surcharge: Rs. 448.5

```
#include <iostream>
#include <string>
using namespace std;

int main() {
   string name;
   int units;
   float amount, total;
   cout << "Enter the name of the user: ";
   getline(cin, name);
   cout << "Enter the number of units consumed: ";
   cin >> units;
```

```
 if (units <= 100) {
        amount = units * 0.60;
    } else if (units <= 300) {
        amount = 100 * 0.60 + (units - 100) * 0.80;
    } else {
        amount = 100 * 0.60 + 200 * 0.80 + (units - 300) * 0.90;
    }

    if (amount < 50.00) {
        amount = 50.00;
    }
    if (amount > 300.00) {
        totalAmount = amount + (amount * 0.15);
    } else {
        totalAmount = amount;
    }

    cout << "User: " << name << endl;
    cout << "Units Consumed: " << units << endl;
    cout << "Amount: Rs. " << amount << endl;
    cout << "Total Amount after surcharge: Rs. " << totalAmount << endl;
    return 0;
}
```

Function Overloading :

Overloading refers to the use of the same thing for different purposes. C++ also permits overloading of functions. This means that we can use the same function name to create functions that perform a variety of different tasks. This is known as function polymorphism in OOP.

For example, an overloaded add( ) function handles different types of data as shown below:

```
// Declarations
int add(int a, int b);                  // prototype 1
int add(int a, int b, int c);           // prototype 2
double add(double x, double y);         // prototype 3
double add(int p, double q);            // prototype 4
double add(double p, int q);            // prototype 5

// Function calls
cout << add (5, 10);                    // uses prototype 1
cout << add ( 15, 10.0);                // uses prototype 4
cout << add( 12.5, 7.5);                // uses prototype 3
cout << add( 5, 10, 15);                // uses prototype 2
cout << add( 0.75, 5);                  // uses prototype 5
```

Q1 .

Write a function power( ) to raise a number m to a power n. The function takes a double value for m and int value for n, and returns the result correctly. Use a default value of 2 for n to make the function to calculate squares when this argument is omitted. Write a main that gets the values of m and n from the user to test the function.

Write a function that performs the same operation as that of above program but takes an int value for m. Both the functions should have the same name. Write a main that calls both the functions. Use the concept of function overloading.

Enter a number (m) and the power (n) for m^n (default n=2): 3 4

Result using the double version: 81

Result using the double version with default power (n=2): 9

Enter a number (m) and the power (n) for m^n (int version): 3 4

Result using the int version: 81

```
#include <iostream>
#include <cmath>
using namespace std;

// Function to calculate power for double m and int n with default n = 2
double power(double m, int n = 2) {
    double result = 1;
    for (int i = 0; i < n; i++) {
        result *= m;
    }
    return result;
}

// Function to calculate power for int m and int n
int power(int m, int n) {
    int result = 1;
    for (int i = 0; i < n; i++) {
        result *= m;
```

```
    }
    return result;
}

int main() {
    double m1;
    int n1, m1, m2, n2;

   cout << "Enter a number (m) and the power (n) for m^n (default n=2): ";
    cin >> m1 >> n1;
    cout << "Result using the double version: " << power(m1, n1) << endl;

    // If n is not provided, use the default value
    cout << "Result using the double version with default power (n=2): " << power(m1) << endl;

    cout << "Enter a number (m) and the power (n) for m^n (int version): ";
    cin >> m2 >> n2;
    cout << "Result using the int version: " << power(m2, n2) << endl;

    return 0;
}
```