Objected Oriented Programming Concepts Using C++ & Data Structures

Intro

- Programming : giving instructions
- Programming Paradigm : Way of Thinking
- C++ : Programming Language that supports procedural and object-oriented.
- Data Structures : Way of organizing data

Topics covered

1. Difference between C and C++
2. Introduction to C++: Identifier, Keywords, DataTypes, Constants,Variables,Functions
3. Operators: Arithmetic, relational, logical, conditional and assignment. Sizeof operator, Operator precedence and associativity.
4. Type conversion, Variable declaration, expressions, statements, manipulators
5. Input and output statements, stream I/O, Conditional and Iterative statements,breaking control statements.
6. OOP Concepts
7. Class and Objects
8. Executing sample C++ programs

Language Timeline

Machine Language	Assembly Language	Procedural Programming
Lowest-level	Intermediate level	High-level
Understood by computers.	Converted into machine language by assembler	High-level languages are translated into assembly language or machine language by a compiler.
Almost impossible for humans to use because they consist entirely of numbers/Binary	Generally lack high-level conveniences such as variables and functions	Instructions and variables have names instead of being just numbers.
Eg : 101010101011	Eg : MOV A B ADD 2 3	Eg : C

Difference between C and C++

C	C++
C is a subset of C++.	C++ is a superset of C
C supports procedural programming paradigm for code development.	C++ supports both procedural and object oriented programming paradigms; therefore C++ is also called a hybrid language.
Function driven language.	Object driven language.
Eg : 101010101011	Eg : MOV A B ADD 2 3

Introduction to C++

- C++ is a statically-typed, compiled, multi-paradigm, general-purpose programming language
- C++ is completely free and readily available on all platforms.
- C++ compiler : Converts C++ code to machine understandable code.
- Many major applications like Game Development, Android Native Development, Desktop Application Development,Web Application Development, etc.
- Its base for all commonly widely used languages.

Keywords in C++

- Every word in C++ language is either a keyword or an identifier.
- They are specifically used by the compiler for its own purpose and they serve as building blocks of a C++ program.
- These words convey speical meanin to the compliler
- C++ language has some reserve words which are called keywords of C++ language. These are the part of the C++ Tokens.
- There are 63 keywords currently defined for Standard C++.

Identifiers in C++

- Every word in C++ language is either a keyword or an identifier.
- The name of a variable, function, class, or other entity in C++ is called an identifier. C++ gives you a lot of flexibility to name identifiers as you wish. However, there are a few rules that must be followed when naming identifiers:
  - The identifier can not be a keyword. Keywords are reserved.
  - The identifier can only be composed of letters, numbers, and the underscore character. That means the name can not contains no symbols (except the underscore) or whitespace.
  - The identifier must begin with a letter or an underscore. It can not start with a number.
  - C++ distinguishes between lower and upper case letters nvalue is different than nValue is different than NVALUE.

Constants

- Constants refer to fixed values that the program may not alter and they are called literals
- Integer literal can be a decimal, octal, or hexadecimal constant.
  - Eg : **85** // decimal **0213** // octal **0x4b** // hexadecimal
- Floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part.
  - Eg : 3.14159
- Boolean literals and they are part of standard C++ keywords
  - value of **true** representing true.
  - value of **false** representing false.
- Character literals are enclosed in single quotes.
  - **'a' '1' '2'**
- String literals are enclosed in double quotes.
  - "hello, dear"

Operators in C++

Operators Name	Operators	Example
Arithmetic Operators	+, -, *, /, %	2 + 3 9%2
Assignment Operators	=, +=, -=, *=, /=, %=	a = 2 a+=4 a is now 6
Auto-increment and Auto-decrement Operators	++ and — Increment / Decrement by one	a++ , a is now 5
Logical Operators	&&, ||, ! perform logical operation	true && true !false
Comparison (relational) operators	==, !=, >, <, >=, <= perform the compasion and yields true or false	4>5 false 4=='4' false 2<=2 true
Conditional Operator	evaluates a boolean expression and assign the value based on the result.	num1 = (expression) ? value if true : value if false a = (3>2) ? 5 : 6

Operator Precedence in C++

- Determines which operator needs to be evaluated first if an expression has more than one operator. Operator with higher precedence at the top and lower precedence at the bottom.

Unary Operators	++ – – ! ~
Multiplicative	* / %
Additive	=+–
Shift	<< >> >>>
Relational	> >= < <=
Equality	== !=
Bitwise AND	&
Bitwise XOR	^
Bitwise	OR
Logical AND	&&
Logical OR	||
Ternary	?:
Assignment	= += -= *= /= %= > >= < <= &= ^= |=

C++ Statements

- The program elements that control how and in what order objects are manipulated.

- Statements are executed sequentially, except when an expression statement, a selection statement, an iteration statement, or a jump statement specifically modifies that sequence.

- Categories of Statements

  - **Expression statements**. These statements evaluate an expression for its side effects or for its return value.

  - **Null statements**. These statements can be provided where a statement is required by the C++ syntax but where no action is to be taken.

  - **Compound statements**. These statements are groups of statements enclosed in curly braces ({ }). They can be used wherever a single statement may be used.

Cont.

- **Selection statements**. These statements perform a test; they then execute one section of code if the test evaluates to true (nonzero). They may execute another section of code if the test evaluates to false.

- **Iteration statements**. These statements provide for repeated execution of a block of code until a specified termination criterion is met.

- **Jump statements.** These statements either transfer control immediately to another location in the function or return control from the function.

- **Declaration statements.** Declarations introduce a name into a program.

Expressions (C++)

- Expressions are sequences of operators and operands that are used for one or more of these purposes:
  - Computing a value from the operands.
  - Designating objects or functions.
  - Generating "side effects." (Side effects are any actions other than the evaluation of the expression — for example, modifying the value of an object.)
- Example : a = b +c;

Data Types

- Data types define the type of data a variable can hold, for example an integer variable can hold integer data, a character type variable can hold character data etc.

Primary(Built-in)	User Defined Data Types	Derived Data Types
Basic storage units of a computer and the most common ways of using them to hold data which understoodable by machine	User can define additional types using primitive and derived datatypes	Construct other types using declarator operator
Example : char,int,float,boolean ,double ,void	Example : Structure,Union Class, Enumeration	Example : Array, Function, Pointer, Reference

Variables in C++

- Store any type of values within a program
- Declared in various ways each having different memory requirements and storing capability
- Name of memory locations that are allocated by compilers, and the allocation is done based on the data type used for declaring the variable.
- **Syntax :** data_type variable_name;
  - Eg : int a; char b;
- Built-in Data types : predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc.

Variable Declaration in C++

- Provides assurance to the compiler that there is one variable existing with the given type and name .
- Declaration has meaning at the time of compilation only, compiler needs actual variable definition at the time of linking of the program.
- **Named stored location**

Eg :

```
int var1;
char var2 = 'a'
// Memory address 1020 and its content can be accessed
// using variable name var2
```

Scope of Variables

- All the variables have their area of functioning, and out of that boundary they don't hold their value, this boundary is called scope of the variable
- Scope are defined using blocks {} in C++
- Global variables are those, which ar once declared and can be used throughout the lifetime of the program by any class or any function
- Local variables are the variables which exist only between the curly braces, in which its declared. Outside that they are unavailable and leads to compile time error

Type Conversion

- Conversion from one type to another.
- Two types of type conversion: Implict & Explict
- Implicit Type Conversion
  - Also known as ‘automatic type conversion’.
  - Done by the compiler on its own, without any external trigger from the user.
  - Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
  - All the data types of the variables are upgraded to the data type of the variable with largest data type.
  - It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Cont.

Data Type	Size in Memory (Byte)	Data Type	Size in Memory (Byte)	Example
char	1	Int	2	char var1 = 'a'; int var2 = var1 , val2 value is 69
int	2	float	4	int var1 = 2; float var2 = var1 , val2 value is 2.0
float	4	int	2	float var1 = 2.2; int var2 = var1 , val2 value is 2

Cont.

- Explicit Type Conversion:
  - This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type
  - Converting by assignment: This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

```
Syntax:
Eg :  double x = 1.2;  // Explicit conversion from
                       //double to int
    int sum = (int)x + 1;
```

Functions in C++

- Defines an logic or action/steps in the program
- Provide modularity to a program
- Implementation Units of function in C++ :
  - Return-type: suggests what the function will return. It can be int, char, ..etc There can be functions which does not return anything, they are mentioned with void.
  - Function Name: is the name of the function, using the function name it is called.
  - Parameters: are variables to hold values of arguments passed while function is called. A function may or may not contain parameter list.

- Eg **int add(int a,int b); void print(char a); void getSomething();**

Declaration, Definition and Calling a Function

- Function declaration, is done to tell the compiler about the existence of the function. Function's return type, its name & parameter list is mentioned.
- Function Definition , tells what the function do. Here the logic or action/steps to be done is written.
- Calling a Function : Functions are called by their names, its where we tell the machine to execute the instructions listed in defintion and we can pass the data to the functions

Example

```
int g = 100;

//declaring the function
int sum (int x, int y);  //presents ';' to indicate
                         //declaration or end of stmt

void main()
{
    int a = 10;
    int b = 20;
    int c = sum (a, b);    // calling the function
}
//defining the function
int sum (int x, int y) //presents {} to define the
                       //function
{
    int z = x + y;
    return z;
}
```

Input / Output in C++

- Performing input and output. In C++ input and output is performed in the form of sequence of bytes or more commonly known as streams.

- Input Stream: If the direction of flow of bytes is from device(for example: Keyboard) to the main memory then this process is called input.

- Output Stream: If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.

- In C++ articles, these two keywords **cout and cin** are used very often for taking inputs and printing outputs. These two are the most basic methods of taking input and output in C++. For using cin and cout we must include the header file **iostream** in our program.

if-else Statement (C++)

- Controls conditional branching.
- Statements in the if-block are executed only if the if-expression evaluates to a non-zero value (or TRUE).
- If the value of expression is nonzero, statement1 and any other statements in the block are executed and the else-block, if present, is skipped.
- If the value of expression is zero, then the if-block is skipped and the else-block, if present, is executed.

Iteration Statements (C++)

- Iteration statements cause statements (or compound statements) to be executed zero or more times, subject to some loop-termination criteria. When these statements are compound statements, they are executed in order
- C++ provides four iteration statements — while, do, for
- Without Iteration

```
Eg :   display("Hello")
display("Hello")
display("Hello")
display("Hello")
.....
display("Hello")
```

Using Iteration

```
Eg : Iterate 100 Times
        display("Hello")
```

Iteration Statements (C++)

```
int counter =1       // intialization stmt
while(i<=10)         // Condition checking
{
    DISPLAY  i       // body of loop
    i = i +1;        // Increment stmt
}
```

```
\\intialization;conditionchecking;increment stmt
for(int counter=1;i<=10;i++)      \\     Entry checking Loop
       DISPLAY i                  \\     Body of Loop
```

```
int counter =1 // intialization stmt
do
{                  // Exit Checking Loop
       DISPLAY i
       i = i +1;    // Increment stmt
                    //     Body of Loop
}while(i<=10);  // Condition checking
```

Jump Statements

- jump statement performs an immediate local transfer of control.
- **Break Statement** ends execution of the nearest enclosing loop or conditional statement in which it appears. Control passes to the statement that follows the end of the statement, if any.
- **continue Statement** : Forces transfer of control to the controlling expression of the smallest enclosing do, for, or while loop Any remaining statements in the current iteration are not executed
- **return Statement** Terminates the execution of a function and returns control to the calling function (or to the operating system if you transfer control from the main function). Execution resumes in the calling function at the point immediately following the call.

OOPs Concepts

- Way of Thinking
- Programming - giving instructions
- Machine Language -> Assembly Language -> Procedural Programming Language
  - Big logic Divided into functions
  - Not emphasis on Data
  - Need to make data global or via passing
  - gap b\w client and developer
- In real world talk about objects .
- Seek object for their functionality

1. Object
2. Class
3. Data Encapsulation
4. Data Abstractions
5. Inheritance
6. Polymorphism

Object

- Represents entity
- Attributes
  - characteristics
  - implemented via variable data members
- Functionality
  - behaviour
  - implemented via Methods/member Functions
- Instance of class

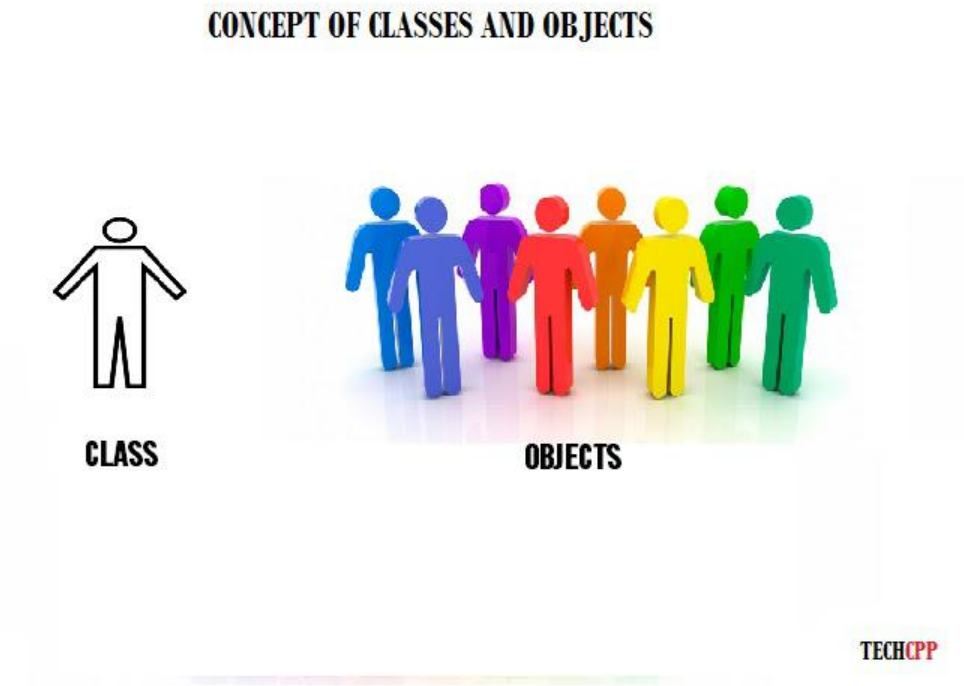
Example

Object of Student :

- Attributes
  - Roll No : 5
  - Name : Harry
  - Address : 605 –AB
  - Marks : [34,23,55]
  - Height : 4
  - Attendence : [Jan: 23,Feb: 21,..]
- Functionalities
  - calculatePercentage
  - getAttendenanceOfMonth(Month)

Class

- Specification of Object
- Blueprint that specifies the attributes and behavior of an Object
- Behaviors are actions that an object can perform.
- Classes are **user defined data type** based on which objects are created.



Example

```
class Student {
        int rollNo ;
        char [] name;
        char [] address;
        float[] marks;
        float height;
        Attendences attendences;

    float calculatePercentage()
    int getAttendenanceOfMonth(Month)
    }

Object Creation :
 Student s1 = new Student();     // s1.rollNo - 12
 Student s2 = new Student();     // s2.rollNo - 23
```