Programming Abstractions

Other readings and relevant websites:

- https://www.w3schools.com/CPP/default.asp
- https://www.programiz.com/cpp-programming
- https://cplusplus.com/doc/tutorial/
- https://www.tutorialspoint.com/cplusplus/index.htm
- https://www.javatpoint.com/cpp-tutorial

Topics:

- C++ Fundamentals: Data types, Conditional Statements, Loops
- Arrays, Pointers, Strings & 2-D Arrays in C++
- Functions in C++ : Inline functions, Default arguments, Function prototyping, Function Overloading
- Pointers & Dynamic Memory Management
- Asymptotic Notation (Big O)
- Recursion
- Bitwise Operators
- Classes and Objects
- Constructors and Destructors
- Operator Overloading and Type Conversion
- Inheritance
- Virtual base class, Overriding member functions
- Virtual Functions and Polymorphism: Concept of Binding - Early binding and late binding, Virtual functions, Pure virtual functions, Abstract classes
- Exception Handling
- Templates and Generic Programming
- Introduction to Data Structures.
- Arrays
- Strings
- Dynamic Memory Management and Memory Model
- Linked List - Single, Double and Circular
- Stacks - using Arrays, LL

- Queues - Implement using Arrays and LL
- Circular Queues
- Project Work

- ## Introduction to C++:

Languages that support OOP features include C++, Smalltalk, Object Pascal and Java. C++ is most successful, practical, general-purpose language and widely used in industry. It is a general-purpose programming language designed by Bjarne Structure as an extension to the C language with object-oriented data abstraction mechanisms and strong static type safety.

- **Salient Features of C++:**
- **C++ is C**: C++ supports (almost) all the features of C. Like C, C++ allows programmers to manage the memory directly, so as to develop efficient programs.
- **C++ is OO**: C++ enhances the procedural-oriented C language with the object-oriented extension. The OO extension facilitates design, reuse and maintenance for complex software.
- **Template C++**: C++ introduces generic programming, via the so-called template. You can apply the same algorithm to different data types.
- **STL**: C++ provides a huge set of reusable standard libraries, in particular, the Standard Template Library (STL).

- ## Data types:

All variables use data-type during declaration to restrict the type of data to be stored.

C++ supports the following data types:

- Primary or Built in or Fundamental data type
- Derived data types
- User defined data types

- **Primitive Data Types:**

These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char, float, bool, etc. Primitive data types available in C++ are:

- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Valueless or Void
- Wide Character

- **Derived Data Types:**

The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:

- Function
- Array
- Pointer
- Reference

- **Abstract or User-Defined Data Types:**

These data types are defined by the user itself. Like, as defining a class in C++ or a structure. C++ provides the following user-defined datatypes:

- Class
- Structure
- Union
- Enumeration

- Typedef defined Datatype

GeeksforGeeks Link:

https://www.geeksforgeeks.org/c-data-types/

Programiz Link:

https://www.programiz.com/cpp-programming/data-types

- ## Conditional Statements:

C++ supports the usual logical conditions from mathematics:

- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- Equal to a == b
- Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

C++ has the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

W3Schools Link:

https://www.w3schools.com/cpp/cpp_conditions.asp

GeeksforGeeks Link:

https://www.geeksforgeeks.org/decision-making-c-c-else-nested-else/

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus/cpp_conditional_operator.htm

- ## Loops:

There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

- **while loop:**

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

- **for loop:**

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

- **do...while loop:**

Like a 'while' statement, except that it tests the condition at the end of the loop body.

- **nested loops:**

You can use one or more loop inside any another 'while', 'for' or 'do..while' loop.

- **break statement:**

Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.

- **continue statement:**

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

- **goto statement:**

Transfers control to the labelled statement. Though it is not advised to use go to statement in your program.

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus/cpp_loop_types.htm

GeeksforGeeks Link:

https://www.geeksforgeeks.org/loops-in-c-and-cpp/

- **Array:**
- It is a group of variables of similar data types referred by single element.
- It elements are stored in a contiguous memory location.
- The size of array should be mentioned while declaring it.
- Array elements are always counted from zero (0) onward.

- **Advantages:**
- Code Optimization: we can retrieve or sort the data efficiently.
- Random access: We can get any data located at an index position.

- **Disadvantages:**
- **Size Limit**: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.

```
#include <iostream>
using namespace std;
int main()
{
    int arr1[10];
    int n = 10;
    int arr2[n];
    return 0;
}
```

GeeksforGeeks Link:

https://www.geeksforgeeks.org/arrays-in-c-cpp/

W3School Link:

https://www.w3schools.com/cpp/cpp_arrays.asp

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus/cpp_arrays.htm

- **C++ Pointers:**

The pointer in C++ language is a variable, it is also known as locator or indicator that points to an address of a value.

- **Usage of pointer:**

There are many usage of pointers in C++ language.

1. **Dynamic memory allocation:**
   In c language, we can dynamically allocate memory using malloc() and calloc() functions where pointer is used.

2. **Arrays, Functions and Structures:**
   Pointers in c language are widely used in arrays, functions and structures. It reduces the code and improves the performance.

Javatpoint Link:

https://www.javatpoint.com/cpp-pointers#:~:text=The%20pointer%20in%20C%2B%2B%20language,with%20arrays%2C%20structures%20and%20functions.

W3school Link:

https://www.w3schools.com/cpp/cpp_pointers.asp

- **What is a String in C++?**

A string in C++ is a type of object that represents a collection (or sequence) of different characters. Strings in C++ are a part of the standard string class (std::string). The *string class* stores the characters of a string as a **collection of bytes** in contiguous memory locations.

```
string str_name = "This is a C++ string";
```

- **C-Style Character String:**

In the C programming language, a string is defined as an array of characters that ends with a null or termination character (\0). The termination is important because it tells the compiler where the string ends.

```
char str_array[8] = {'D', 'S', 'A', 'C', 'A’, 'M', 'P', '\0'};
char str_array[] = "DSACAMP";
```

- **Operations on Strings:**

- **getline():**

This function is used to store a stream of characters as entered by the user in the object memory.

- **push_back():**

This function is used to store a stream of characters as entered by the user in the object memory.

- **pop_back():**

Introduced from C++11(for strings), this function is used to delete the last character from the string.

- **length():**

This function finds the length of the string.

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

int main()
{
    string str = "dsacamp is for champs";

    cout << "The initial string is : ";
    cout << str << endl;

    str.resize(13);

    cout << "The string after resize operation is : ";
    cout << str << endl;

    cout << "The capacity of string is : ";
    cout << str.capacity() << endl;

    cout << "The length of the string is :" << str.length()
         << endl;
}
```

- **C++ String Compare Example:**

Let's see the simple example of string comparison using strcmp() function.

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
  char key[] = "mango";
  char buffer[50];
  do {
     cout<<"What is my favourite fruit? ";
     cin>>buffer;
  } while (strcmp (key,buffer) != 0);
 cout<<"Answer is correct!!"<<endl;
  return 0;
}

Output:

What is my favourite fruit? apple
What is my favourite fruit? banana
What is my favourite fruit? mango
Answer is correct!!
```

GeeksforGeeks Link:

https://www.geeksforgeeks.org/stdstring-class-in-c/

W3School Link:

https://www.w3schools.com/cpp/cpp_strings.asp

- **Multi-dimensional Arrays:**

C++ allows multidimensional arrays. Here is the general form of a multidimensional array declaration

```
type name[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional 10 . 20 . 40 integer array −

```
int li[10][20][40];
```

we can create an array of an array, known as a multidimensional array. For example:

```
int arr[3][4];
```

Here, arr is a two-dimensional array. It can hold a maximum of 12 elements.

Three-dimensional arrays also work in a similar way. For example:

```
float arr[2][4][3];
```

This array arr can hold a maximum of 24 elements.

- **Two Dimensional Array:**

```
// two dimensional array

#include <iostream>
using namespace std;

int main() {
    int arr[3][2] = {{2, -5},
                      {4, 0},
                      {9, 1}};
    for (int i = 0; i < 3; ++i)
      {
        for (int j = 0; j < 2; ++j)
             {
            cout << "arr[" << i << "][" << j << "] = " <<
arr[i][j] << endl;
        }
    }

    return 0;
}

Output:

arr[0][0] = 2
arr[0][1] = -5
arr[1][0] = 4
arr[1][1] = 0
arr[2][0] = 9
arr[2][1] = 1
```

GeeksforGeeks Link:

https://www.geeksforgeeks.org/multidimensional-arrays-c-cpp/

Programiz Link:

https://www.programiz.com/cpp-programming/multidimensional-arrays

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus/cpp_multi_dimensional_arrays.htm

- **Inline function:**

If make a function as inline, then the compiler replaces the function calling location with the definition of the inline function at compile time. Any changes made to an inline function will require the inline function to be recompiled again because the compiler would need to replace all the code with a new code; otherwise, it will execute the old functionality.

```
inline return_type function_name(parameters)
{
   // function code?
}
```

We cannot provide the inlining to the functions in the following circumstances:

- If a function is recursive.
- If a function contains a loop like for, while, do-while loop.
- If a function contains static variables.
- If a function contains a switch or go to statement

- **Advantages of inline function:**
- In the inline function, we do not need to call a function, so it does not cause any overhead.

- It also saves the overhead of the return statement from a function.
- It does not require any stack on which we can push or pop the variables as it does not perform any function calling.

- **Disadvantages of inline function:**
- If we use many inline functions, then the binary executable file also becomes large.
- The use of so many inline functions can reduce the instruction cache hit rate, reducing the speed of instruction fetch from the cache memory to that of the primary memory.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/inline-functions-cpp/

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus/cpp_inline_functions.htm

Programiz Link:

https://www.programiz.com/cpp-programming/inline-function

- **Default Arguments in C++:**

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the calling function doesn't provide a value for the argument. In case any value is passed, the default value is overridden.

- **Template Function:**

A template is a simple yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types.

- **Can there be more than one argument to templates?**

Yes, like normal parameters, we can pass more than one data type as arguments to templates.

- **Template Specialization in C++:**

Template in C++is a feature. We write code once and use it for any data type including user defined data types. For example, sort() can be written and used to sort any data type items. A class stack can be created that can be used as a stack of any data type.

- **Example with template and default argument:**

```
template<class t>
t add(t a,t b,t c=0,t d=10,t e=11)
{
	return a+b+c+d+e;
}

int main()
{
	cout<<add<int>(10,20)<<endl;
	cout<<add<int>(10,20,30)<<endl;
	cout<<add<int>(10,20,30,40)<<endl;
	cout<<add<float>(10.10,20.20,30.30,40.40,50.50)<<endl;
	return 0;

}
```

GeeksforGeeks Link:

https://www.geeksforgeeks.org/default-arguments-c/

Programiz Link:

https://www.programiz.com/cpp-programming/default-argument

- **function prototype in C/C++:**

The function prototypes are used to tell the compiler about the number of arguments and about the required datatypes of a function parameter, it also tells about the return type of the function.

```
#include<stdio.h>

void function(int);

int main() {
   function(50);
}

void function(int x) {
   printf("The value of x is: %d", x);
}
```

Tutorialspoint Link:

https://www.tutorialspoint.com/what-is-the-purpose-of-a-function-prototype-in-c-cplusplus

Codescracker Link:

https://codescracker.com/cpp/cpp-function-definition.htm

- **Function Overloading:**

With function overloading, multiple functions can have the same name with different parameters:

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

float absolute(float var)
{
    if (var < 0.0)
        var = -var;
    return var;
}

int absolute(int var)
{
     if (var < 0)
```

```
        var = -var;
    return var;
}

int main()
{
    cout << "Absolute value of -5 = " << absolute(-5) << endl;
    cout << "Absolute value of 5.5 = " << absolute(5.5f) <<
endl;
    return 0;
}
```

W3school Link:

https://www.w3schools.com/cpp/cpp_function_overloading.asp

Programiz Link:

https://www.programiz.com/cpp-programming/function-overloading

GeeksforGeeks Link:

https://www.geeksforgeeks.org/function-overloading-c/

- ## Dynamic memory allocation in C++:

There are times where the data to be entered is allocated at the time of execution. For example, a list of employees increases as the new employees are hired in the organization and similarly reduces when a person leaves the organization. This is called managing the memory.

- **Static allocation or compile-time allocation** - Static memory allocation means providing space for the variable. The size and data type of the variable is known, and it remains constant throughout the program.
- **Dynamic allocation or run-time allocation** - The allocation in which memory is allocated dynamically. In this type of allocation, the exact size of the variable is not known in advance. Pointers play a major role in dynamic memory allocation.

In C++, memory is divided into two parts -

- **Stack** - All the variables that are declared inside any function take memory from the stack.
- **Heap** - It is unused memory in the program that is generally used for dynamic memory allocation.

Syntax

Pointer_variable = new data_type;

The pointer_varible is of pointer data_type. The data type can be int, float, string, char, etc.

Example

int *m = NULL // Initially we have a NULL pointer

m = new int // memory is requested to the variable

It can be directly declared by putting the following statement in a line -

int *m = new int

Initialize memory

Javatpoint Link:

https://www.javatpoint.com/dynamic-memory-allocation-in-cpp

W3schools.in Link:

https://www.w3schools.in/cplusplus/dynamic-memory-allocation

Programiz Link:

https://www.programiz.com/cpp-programming/memory-management#:~:text=C%2B%2B%20allows%20us%20to%20allocate,the%20memories%20allocated%20to%20variables.

Asymptotic Notation (Big O):

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Usually, the time required by an algorithm falls under three types −

- **Best Case** − Minimum time required for program execution. (Ω Notation)
- **Average Case** − Average time required for program execution. (θ Notation)
- **Worst Case** − Maximum time required for program execution. (O Notation)

Big Oh Notation, O:

The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

Omega Notation, Ω:

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

Theta Notation, θ:

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time.

Common Asymptotic Notations:

constant	−	O(1)
logarithmic	−	O(log n)
linear	−	O(n)
n log n	−	O(n log n)

quadratic	−	O(n2)
cubic	−	O(n3)
polynomial	−	nO(1)
exponential	−	2O(n)

Tutorialspoint Link:

https://www.tutorialspoint.com/data_structures_algorithms/asymptotic_analysis.htm

GeeksforGeeks Link:

https://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/

Programiz Link:

https://www.programiz.com/dsa/asymptotic-notations

- **Recursion:**

A function that calls itself is known as a recursive function. And, this technique is known as recursion. The recursion continues until some condition is met.

To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and the other doesn't.

- **Advantages of C++ Recursion:**
- It makes our code shorter and cleaner.
- Recursion is required in problems concerning data structures and advanced algorithms, such as Graph and Tree Traversal.
- **Disadvantages of C++ Recursion:**
- It takes a lot of stack space compared to an iterative program.

- It uses more processor time.
- It can be more difficult to debug compared to an equivalent iterative program.

```
void order(int x)
{
	if(x==-1)
	{
		return;
	}
	cout<<x<<endl;
	order(x-1);
}

int main()
{
	int a=5;
	order(a);
	return 0;
}
```

Programiz Link:

https://www.programiz.com/cpp-programming/recursion

GeeksforGeeks Link:

https://www.geeksforgeeks.org/recursion/

Javatpoint Link:

https://www.javatpoint.com/cpp-recursion

- **Bitwise Operators:**

1. **The & (bitwise AND)** in C or C++ takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
2. **The | (bitwise OR)** in C or C++ takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.
3. **The ^ (bitwise XOR)** in C or C++ takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
4. **The << (left shift)** in C or C++ takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.
5. **The >> (right shift)** in C or C++ takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.
6. **The ~ (bitwise NOT)** in C or C++ takes one number and inverts all bits of it.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp/

Programiz Link:

https://www.programiz.com/cpp-programming/bitwise-operators

Cprogramming Link:

https://www.cprogramming.com/tutorial/bitwise_operators.html

- ## Classes and Objects:

A class in C++ is the building block that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

- A Class is a user defined data-type which has data members and member functions.

- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behaviour of the objects in a Class.

An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

- **Declaring Objects:**

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

```
class MyClass {         // The class
  public:               // Access specifier
    int myNum;           // Attribute (int variable)
    string myString;   // Attribute (string variable)
};

int main() {
  MyClass myObj;   // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

Syntax:

```
ClassName ObjectName;
```

GeeksforGeeks Link:

https://www.geeksforgeeks.org/c-classes-and-objects/

W3school Link:

https://www.w3schools.com/cpp/cpp_classes.asp

Programiz Link:

https://www.programiz.com/cpp-programming/object-class

- **Constructor and Destructor:**
- **Constructor:**

A constructor is a member function of a class that has the same name as the class name. It helps to initialize the object of a class. It can either accept the arguments or not. It is used to allocate the memory to an object of the class. It is called whenever an instance of the class is created. It can be defined manually with arguments or without arguments. There can be many constructors in a class. It can be overloaded but it can not be inherited or virtual.

Syntax:

```
ClassName()
 {
   //Constructor's Body
 }
```

- **Destructor:**

Like a constructor, Destructor is also a member function of a class that has the same name as the class name preceded by a tilde(~) operator. It helps to deallocate the memory of an object. It is called while the object of the class is freed or deleted. In a class, there is always a single destructor without any parameters so it can't be overloaded. It is always called in the reverse order of the constructor.

Syntax:

```
~ClassName()
  {
     //Destructor's Body
  }
```

Note: If we do not specify any access modifiers for the members inside the class then by default the access modifier for the members will be Private.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/difference-between-constructor-and-destructor-in-c/

scaler Link:

https://www.scaler.com/topics/constructor-and-destructor-in-cpp/#:~:text=Constructor%20in%20C%2B%2B%20is%20a,whenever%20an%20object%20is%20destroyed.

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus/cpp_constructor_destructor.htm

- **Operator Overloading:**

we can change the way operators work for user-defined types like objects and structures. This is known as operator overloading.

Suppose we have created three objects c1, c2 and result from a class named Complex that represents complex numbers.

Since operator overloading allows us to change how operators work, we can redefine how the + operator works and use it to add the complex numbers of c1 and c2 by writing the following code:

```
result = c1 + c2;
```

instead of something like

```
result = c1.addNumbers(c2);
```

- **Syntax for C++ Operator Overloading:**

To overload an operator, we use a special operator function. We define the function inside the class or structure whose objects/variables we want the overloaded operator to work with.

```
class className {
    ... .. ...
    public
       returnType operator symbol (arguments) {
           ... .. ...
       }
    ... .. ...
};
```

- **returnType** is the return type of the function.
- **operator** is a keyword.
- **symbol** is the operator we want to overload. Like: +, <, -, ++, etc.
- **arguments** is the arguments passed to the function.

Programiz Link:

https://www.programiz.com/cpp-programming/operator-overloading

GeeksforGeeks Link:

https://www.geeksforgeeks.org/operator-overloading-c/

Javatpoint Link:

https://www.javatpoint.com/cpp-overloading

Type Conversion in C++:

Type conversion is the process that converts the predefined data type of one variable into an appropriate data type. The main idea behind type conversion is to convert two different data type variables into a single data type to solve mathematical and logical expressions easily without any data loss.

For example, we are adding two numbers, where one variable is of int type and another of float type; we need to convert or typecast the int variable into a float to make them both float data types to add them.

Type conversion can be done in two ways in C++, one is **implicit type conversion**, and the second is **explicit type conversion**.

Implicit Type Conversion:

The implicit type conversion is the type of conversion done automatically by the compiler without any human effort. It means an implicit conversion automatically converts one data type into another type based on some predefined rules of the C++ compiler. Hence, it is also known as the **automatic type conversion**.

Explicit type conversion:

Conversions that require **user intervention** to change the data type of one variable to another, is called the **explicit type conversion**. In other words, an explicit conversion allows the programmer to manually changes or typecasts the data type from one variable to another type. Hence, it is also known as typecasting. Generally, we force the explicit type conversion to convert data from one type to another because it does not follow the implicit conversion rule.

The explicit type conversion is divided into two ways:

- Explicit conversion using the cast operator
- Explicit conversion using the assignment operator

Program to convert float value into int type using the cast operator:

Cast operator: In C++ language, a cast operator is a unary operator who forcefully converts one type into another type.

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

int main ()
{
float f2 = 6.7;
// use cast operator to convert data from one type to another
int x = static_cast <int> (f2);
cout << " The value of x is: " << x;
return 0;
}

Output:

The value of x is: 6
```

• Inheritance in C++:

The capability of a class to derive properties and characteristics from another class is called **Inheritance**. Inheritance is one of the most important features of Object-Oriented Programming.

Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”. The derived class now is said to be inherited from the base class.

- **Sub Class:** The class that inherits properties from another class is called Subclass or Derived Class.
- **Super Class:** The class whose properties are inherited by a subclass is called Base Class or Superclass.

• Modes of Inheritance:

There are 3 modes of inheritance.

a. **Public Mode**: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.

b. **Protected Mode**: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.
c. **Private Mode**: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

- **Types Of Inheritance: -**
  a. Single inheritance
  b. Multilevel inheritance
  c. Multiple inheritance
  d. Hierarchical inheritance
  e. Hybrid inheritance

GeeksforGeeks Link:

https://www.geeksforgeeks.org/inheritance-in-c/

Javatpoint Link:

https://www.javatpoint.com/cpp-inheritance

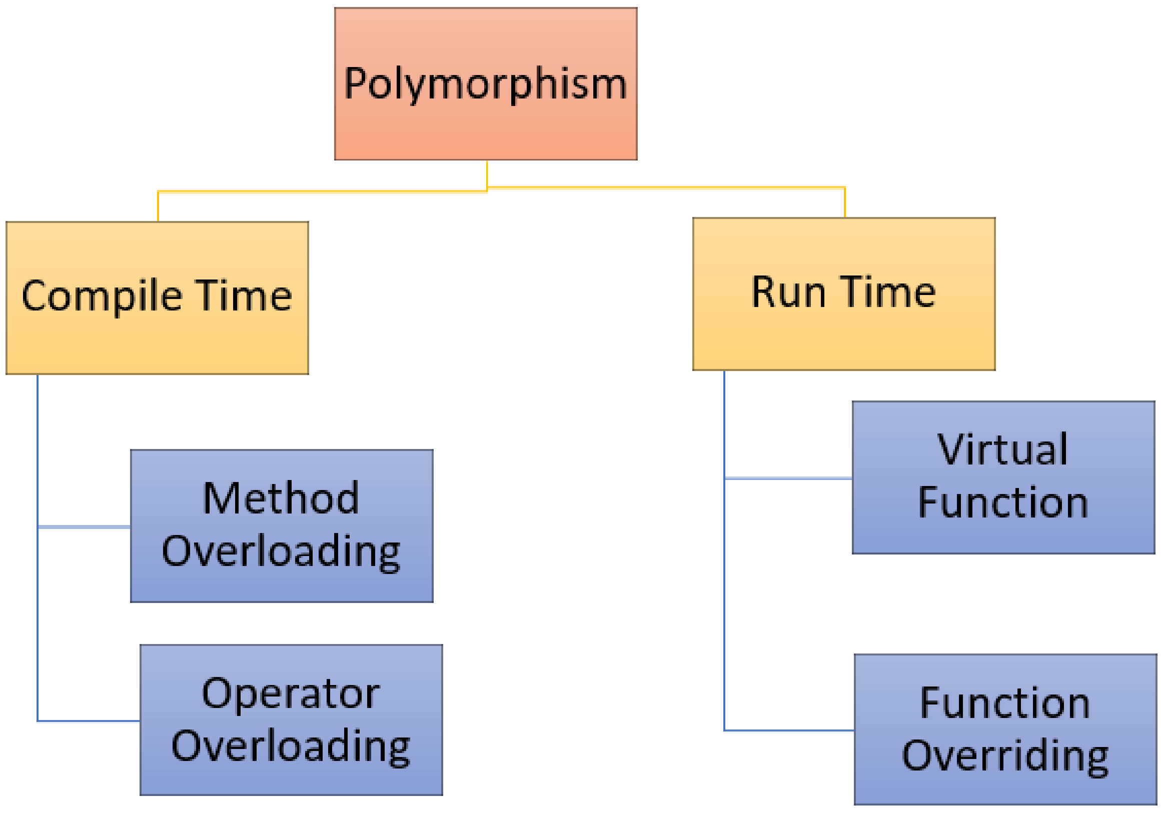
W3school Link:

https://www.w3schools.com/cpp/cpp_inheritance.asp

- **Polymorphism in C++:**

The word "polymorphism" means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A real-life example of polymorphism is a person who at the same time can have different characteristics. Like a man at the same time is a father, a husband and an employee.

polymorphism is mainly divided into two types:
- **Compile-time Polymorphism**
- **Runtime Polymorphism**



1. **Compile-time polymorphism:**

This type of polymorphism is achieved by function overloading or operator overloading.

- **Function Overloading**:
  When there are multiple functions with the same name but different parameters, then the functions are said to be **overloaded**. Functions can be overloaded by **changing the number of arguments** or/and **changing the type of arguments**.

- **Operator Overloading**:
  C++ also provides the option to overload operators. For example, we can make use of the addition operator (+) for string class to concatenate two strings. We know that the task of this operator is to add two operands. So a single operator '+', when placed between integer operands, adds them and when placed between string operands, concatenates them.

2. **Runtime polymorphism:**

This type of polymorphism is achieved by Function Overriding.

- **Function overriding:**
  occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/polymorphism-in-c/

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus/cpp_polymorphism.htm

W3school Link:

https://www.w3schools.com/cpp/cpp_polymorphism.asp

- ## Virtual base class in C++:

Virtual base classes are used in virtual inheritance in a way of preventing multiple “instances” of a given class appearing in an inheritance hierarchy when using multiple inheritances.

- **What is Virtual Class?**

Virtual Class is defined by writing a keyword “virtual” in the derived classes which allows only one copy of data to be copied to Class B and Class C (referring to the above example). It is a way of preventing multiple instances of a class appearing as a parent class in inheritance hierarchy when multiple inheritances are used.

- **Need for Virtual Base Class in C++:**

In order to prevent the error and let the compiler work efficiently, we’ve to use a virtual base class when multiple inheritances occur. It saves space and avoids ambiguity.

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

class A {
  public:
    void display() {
      cout << "Hello form Class A \n";
    }
};

class B: public A {
};

class C: public A {
};

class D: public B, public C {
};

int main() {
  D object;
  object.display();
}

Output:

error: non-static member 'display' found in multiple base-class
subobjects of type 'A':
    class D -> class B -> class A
    class D -> class C -> class A
    object.display();
           ^
note: member found by ambiguous name lookup
    void display()
         ^
1 error generated.
```

- **Overriding member functions:**

As we know, inheritance is a feature of OOP that allows us to create derived classes from a base class. The derived classes inherit features of the base class. Suppose, the same function is defined in both the derived class and the based class. Now if we call this function using the object of the derived class, the function of the derived class is executed. This is known as **function overriding** in C++

```

// C++ program to demonstrate function overriding
```

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

class Base {
   public:
    void print() {
        cout << "Base Function" << endl;
    }
};

class Derived : public Base {
   public:

    void print() {
        cout << "Derived Function" << endl;
    }
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}

Output:

Derived Function
```

Programiz Link:

https://www.programiz.com/cpp-programming/function-overriding

Upgrad Link:

https://www.upgrad.com/blog/function-overriding-in-c-plus-plus/#:~:text=Function%20overriding%20in%20C%2B%2B%20is,present%20in%20the%20parent%20class.

javatpoint Link:

https://www.javatpoint.com/cpp-function-overriding

Virtual Function in C++:

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a virtual keyword in base class.
- The resolving of function call is done at runtime.

Rules for Virtual Functions:

- Virtual functions cannot be static.
- A virtual function can be a friend function of another class.
- Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.
- The prototype of virtual functions should be the same in the base as well as derived class.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/virtual-function-cpp/

javatpoint Link:

https://www.javatpoint.com/cpp-virtual-function

Programiz Link:

https://www.programiz.com/cpp-programming/virtual-functions

C++ Pure Virtual Functions:

Pure virtual functions are used

- if a function doesn't have any use in the base class
- but the function must be implemented by all its derived classes

```
class Shape {
    public:
      // creating a pure virtual function
      virtual void calculateArea() = 0;
};
```

- **Abstract Class:**

A class that contains a pure virtual function is known as an abstract class. In the above example, the class Shape is an abstract class.
We cannot create objects of an abstract class. However, we can derive classes from them, and use their data members and member functions (except pure virtual functions).

```
// C++ program to calculate the area of a square and a circle

#include <iostream>
using namespace std;

// Abstract class
class Shape {
   protected:
    float dimension;

   public:
    void getDimension() {
        cin >> dimension;
    }

    // pure virtual Function
    virtual float calculateArea() = 0;
};

// Derived class
class Square : public Shape {
   public:
    float calculateArea() {
        return dimension * dimension;
    }
};

// Derived class
class Circle : public Shape {
   public:
    float calculateArea() {
        return 3.14 * dimension * dimension;
    }
};

int main() {
    Square square;
    Circle circle;

    cout << "Enter the length of the square: ";
    square.getDimension();
    cout << "Area of square: " << square.calculateArea() <<
endl;

    cout << "\nEnter radius of the circle: ";
    circle.getDimension();
    cout << "Area of circle: " << circle.calculateArea() <<
endl;

    return 0;
}
```

Programiz Link:

https://www.programiz.com/cpp-programming/pure-virtual-funtion

GeeksforGeeks Link:

https://www.geeksforgeeks.org/pure-virtual-functions-and-abstract-classes/

- ## Exception Handling:

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things. When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an exception (throw an error).

- ### C++ try and catch:

Exception handling in C++ consist of three keywords: try, throw and catch:
The try statement allows you to define a block of code to be tested for errors while it is being executed.

The throw keyword throws an exception when a problem is detected, which lets us create a custom error.
The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

```
try  {
    // Block of code to try
    throw  exception;  // Throw an exception when a problem arise
}
catch  ( )  {
    // Block of code to handle errors
}
```

- **Handle Any Type of Exceptions (...):**

If you do not know the throw **type** used in the try block, you can use the "three dots" syntax (...) inside the catch block, which will handle any type of exception:

```
try {
  int age = 15;
  if (age >= 18) {
    cout << "Access granted - you are old enough.";
  } else {
    throw 505;
  }
}
catch (...) {
  cout << "Access denied - You must be at least 18 years old.\n";
}
```

w3school Link:

https://www.w3schools.com/cpp/cpp_exceptions.asp

GeeksforGeeks Link:

https://www.geeksforgeeks.org/exception-handling-c/

Generic Programming:

The generic programming pattern **generalizes** the algorithm with the help of a template in C++ so that it can be used along with different types of data. In templates, we specify a placeholder instead of the actual data type, and that placeholder gets replaced with the data type which is being used during the call for execution, for this execution, an instance of function/class is created by the compiler.

Scalar Link:

https://www.scaler.com/topics/cpp/generic-programming-in-cpp/

GeeksforGeeks Link:

https://www.geeksforgeeks.org/generics-in-c/#:~:text=The%20method%20of%20Generic%20Programming,integer%2C%20string%20or%20a%20character.

Data Structure:

Data Structure can be defined as the group of data elements which provides an efficient way of storing and organising data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc. Data Structures are widely used in almost every aspect of Computer Science i.e. Operating System, Compiler Design, Artifical intelligence, Graphics and many more.

- **Arrays:** An array is a collection of similar type of data items and each data item is called an element of the array. The data type of the element may be any valid data type like char, int, float or double.

- **Linked List**: Linked list is a linear data structure which is used to maintain a list in the memory. It can be seen as the collection of nodes stored at non-contiguous memory locations. Each node of the list contains a pointer to its adjacent node.
- **Stack**: Stack is a linear list in which insertion and deletions are allowed only at one end, called top. A stack is an abstract data type (ADT), can be implemented in most of the programming languages. It is named as stack because it behaves like a real-world stack, for example: - piles of plates or deck of cards etc.

- **Queue**: Queue is a linear list in which elements can be inserted only at one end called rear and deleted only at the other end called front. It is an abstract data structure, similar to stack. Queue is opened at both end therefore it follows First-In-First-Out (FIFO) methodology for storing the data items.
- **Non Linear Data Structures**: This data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement. The data elements are not arranged in sequential structure. Types of Non Linear Data Structures are given below:
- **Trees**: Trees are multilevel data structures with a hierarchical relationship among its elements known as nodes. The bottommost nodes in the herierchy are called leaf node while the topmost node is called root node. Each node contains pointers to point adjacent nodes. Tree data structure is based on the parent-child relationship among the nodes. Each node in the tree can have more than one children except the leaf nodes whereas each node can have atmost one parent except the root node. Trees can be classfied into many categories which will be discussed later in this tutorial.
- **Graphs**: Graphs can be defined as the pictorial representation of the set of elements (represented by vertices) connected by the links known as edges. A graph is different from tree in the sense that a graph can have cycle while the tree can not have the one.

Javatpoint Link:

https://www.javatpoint.com/data-structure-introduction

GeeksforGeeks Link:

https://www.geeksforgeeks.org/introduction-to-data-structures/#:~:text=A%20data%20structure%20is%20a,it%20can%20be%20used%20effectively.

- **Arrays:**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store:

```
string cars[4];
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
int myNum[3] = {10, 20, 30};
```

- **Access the Elements of an Array:**

You access an array element by referring to the index number inside square brackets [].

This statement accesses the value of the first element in cars:

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
cout << cars[0];
```

- **Loop Through an Array:**

You can loop through the array elements with the for loop.

The following example outputs all elements in the **cars** array:

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < 4; i++)
 {
  cout << cars[i] << "\n";
}
```

- **Get the Size of an Array:**

To get the size of an array, you can use the sizeof() operator:

```
int myNumbers[5] = {10, 20, 30, 40, 50};
cout << sizeof(myNumbers);
```

W3school Link:

https://www.w3schools.com/cpp/cpp_arrays.asp

GeeksforGeeks Link:

https://www.geeksforgeeks.org/arrays-in-c-cpp/

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus/cpp_arrays.htm

- ## C++ Strings:

Strings are used for storing text. A string variable contains a collection of characters surrounded by double quotes:

```
string greeting = "dsacamp";
```

To use strings, you must include an additional header file in the source code, the <string> library:

```
// Include the string library
#include <string>
```

- ### String Concatenation:

The + operator can be used between strings to add them together to make a new string. This is called **concatenation**:

```
String a = "dsa ";
string b = "camp";
string c = a + b;
cout << fullName;

output:
John Doe
```

- **Append:**

A string in C++ is actually an object, which contain functions that can perform certain operations on strings.

- **String Length:**

To get the length of a string, use the length() function:

```
string txt = "DSAcamp.in";
cout << "The length of the txt string is: " << txt.length();
```

- **Access Strings:**

You can access the characters in a string by referring to its index number inside square brackets [].

```
string myString = "dsacamp";
cout << myString[0];

Outputs:
 d
```

- **User Input Strings:**

It is possible to use the extraction operator >> on cin to display a string entered by a user:

```
string firstName;
cout << "Type your first name: ";
cin >> firstName;
```

Scaler Link:

https://www.scaler.com/topics/cpp/strings-in-cpp/#:~:text=A%20string%20in%20C%2B%2B%20is%20a%20type%20of%20object%20that,bytes%20in%20contiguous%20memory%20locations.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/stdstring-class-in-c/

W3school Link:

https://www.w3schools.com/cpp/cpp_strings.asp

- **Singly Linked List:**

A linked list is a linked representation of the ordered list. It is a linear collection of data elements termed as nodes whose linear order is given by means of link or pointer. We know that the sequential representation of the ordered list is expensive while inserting or deleting arbitrary elements stored at fixed distance in a fixed memory. The linked representation reduces the expense because the elements are not stored at fixed distance and they are represented randomly and the operations such as insertion and deletion are required change in link rather than movement of data.

Arrays are data structures of fixed size. Insertion and deletion involves reshuffling of array elements. Thus, array manipulation is time-consuming and inefficient. But in linked list items can be added or removed easily to the end or beginning or even in the middle. Linked list does not need contiguous memory space in computer memory. Therefore space can be properly utilized in linked lists as compared to arrays.

Every node consist two parts. The first part is called INFO or DATA, contains information of the data and second part is called LINK or NEXT, contains the address of the next node in the list. A variable called START, always points to the first node of the list and the link part of the last node always contains null value. A null value in the START variable denotes that the list is empty.

```
#include<iostream>
#include<cstdio>

using namespace std;

class LinkList
{
  public:
  int iData; // data item
  LinkList* next; // next link in list
};

int main()
{
  LinkList* head = NULL; /* initialize list head to NULL */
  if (head == NULL)
  {
    cout<<"The list is empty!"<<endl;
  }
  return 0;
}
```

YouTube Link:

https://www.youtube.com/watch?v=0Z3QBdrkOAg

- **Operations on Linked List:**

We may perform the following operations on Linked List

(a) **Traversal**: Processing each element in the Linked List.

(b) **Search**: Finding the location of the element with a given value

(c) **Insertion**: Adding a new element to the Linked List.

(d) **Deletion**: Removing an element from the Linked List.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/program-to-implement-singly-linked-list-in-c-using-class/

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus-program-to-implement-singly-linked-list

Educative Link:

https://www.educative.io/answers/singly-linked-list-in-cpp

- ## Doubly Linked List:

Doubly linked list is a type of data structure that is made up of nodes that are created using self-referential structures. Each of these nodes contain three parts, namely the data and the reference to the next list node and the reference to the previous list node.

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  struct Node *prev;
  struct Node *next;
};
struct Node* head = NULL;
void insert(int newdata) {
  struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
  newnode->data = newdata;
  newnode->prev = NULL;
  newnode->next = head;
  if(head != NULL)
  head->prev = newnode ;
  head = newnode;
}
void display() {
  struct Node* ptr;
  ptr = head;
  while(ptr != NULL) {
    cout<< ptr->data <<" ";
    ptr = ptr->next;
  }
}
int main() {
  insert(3);
  insert(1);
  insert(7);
  insert(2);
  insert(9);
  cout<<"The doubly linked list is: ";
  display();
  return 0;
}
output
The doubly linked list is: 9 2 7 1 3
```

In the above program, the structure Node forms the doubly linked list node. It contains the data and a pointer to the next and previous linked list node. This is given as follows.

```
struct Node {
  int data;
  struct Node *prev;
  struct Node *next;
};
```

GeeksforGeeks Link:

https://www.geeksforgeeks.org/doubly-linked-list/

Programiz Link:

https://www.programiz.com/dsa/doubly-linked-list

- **Circular Linked List:**

Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.

- **Advantages of Circular Linked Lists:**
- Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.

- Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.

GeeksforGeeks Link:

https://www.geeksforgeeks.org/circular-linked-list/

Tutorialspoint Link:

https://www.tutorialspoint.com/data_structures_algorithms/circular_linked_list_algorithm.htm

Programiz Link:

https://www.programiz.com/dsa/circular-linked-list

- **Stacks:**

The STL stack provides the functionality of a stack data structure in C++.

The stack data structure follows the LIFO (Last In First Out) principle. That is, the element added last will be removed first.

- **Create a Stack:**

In order to create a stack in C++, we first need to include the stack header file.

```
#include <stack>
```

Once we import this file, we can create a stack using the following syntax:

```
stack<type> st;
```

Here, type indicates the data type we want to store in the stack. For instance,

```
// create a stack of integers
stack<int> integer_stack;
```

```
// create a stack of strings
stack<string> string_stack;
```

- **Stack Methods:**

In C++, the stack class provides various methods to perform different operations on a stack.

Operation	Description
push()	adds an element into the stack
pop()	removes an element from the stack
top()	returns the element at the top of the stack
size()	returns the number of elements in the stack
empty()	returns true if the stack is empty

Programiz Link:

https://www.programiz.com/cpp-programming/stack

GeeksforGeeks Link:

https://www.geeksforgeeks.org/stack-in-cpp-stl/

Javatpoint Link:

https://www.javatpoint.com/cpp-stack

• Queues:

This data structure works on the FIFO technique, where FIFO stands for First In First Out. The element which was first inserted will be extracted at the first and so on. There is an element called as 'front' which is the element at the front most position or say the first position, also there is an element called as 'rear' which is the element at the last position. In normal queues insertion of elements take at the rear end and the deletion is done from the front.

Queues in the application areas are implied as the container adaptors.

The containers should have a support for the following list of operations:

- empty
- size
- push_back
- pop_front
- front
- back

Member Types	Description
value_type	Element type is specified.
container_type	Underlying container type is specified.
size_type	It specifies the size range of the elements.
reference	It is a reference type of a container.
const_reference	It is a reference type of a constant container.

Javatpoint Link:

https://www.javatpoint.com/cpp-queue

Programiz Link:

https://www.programiz.com/cpp-programming/queue

GeeksforGeeks Link:

https://www.geeksforgeeks.org/queue-cpp-stl/

- ## Circular Queue:

In this tutorial, you will learn what a circular queue is. Also, you will find implementation of circular queue in C, C++, Java and Python.

A circular queue is the extended version of a regular queue where the last element is connected to the first element. Thus forming a circle-like structure.

The circular queue solves the major limitation of the normal queue. In a normal queue, after a bit of insertion and deletion, there will be non-usable empty space.

Circular Queue works by the process of circular increment i.e. when we try to increment the pointer and we reach the end of the queue, we start from the beginning of the queue.

the circular increment is performed by modulo division with the queue size.

```
if REAR + 1 == 5 (overflow!), REAR = (REAR + 1)%5 = 0 (start of queue)
```

- ### Circular Queue Operations:

The circular queue work as follows:

- two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR track the last elements of the queue

- initially, set value of FRONT and REAR to -1

- **Enqueue Operation:**
- check if the queue is full
- add the new element in the position pointed to by REAR
- **Dequeue Operation**
- check if the queue is empty
- for the last element, reset the values of FRONT and REAR to -1

Programiz Link:

https://www.programiz.com/dsa/circular-queue

Tutorialspoint Link:

https://www.tutorialspoint.com/cplusplus-program-to-implement-circular-queue

GeeksforGeeks Link:

https://www.geeksforgeeks.org/circular-queue-set-1-introduction-array-implementation/

“ for project work can contact the Dsacamp team ”
