

Annexure-I

C++ : Foundation to Advance with DSA

FIFTH FORCE

A Training Report

Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

Computer Science and Engineering

Submitted to

Lovely Professional University

Phagwara, Punjab



**L OVELY
P ROFESSIONAL
U NIVERSITY**

Phagwara, Punjab

From MM/DD/YY to MM/DD/YY

Submitted by

Name of the student: Arpit Tyagi

Registration Number: 12012536

Signature of the Student:

Annexure-II: Student Declaration

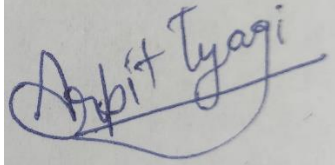
To whom so ever it may concern

I, Arpit Tyagi, 12012536 hereby declare that the work done by me on “C++ foundation to advance with DSA” from 26th May,2022 to 4th July, 2022 is a record of original work for the partial fulfilment of the requirements for the award of the degree, Bachelors of Technology in Computer Science and Engineering.

Name of the Student (Registration Number)

Arpit Tyagi, 12012536

Signature of the student

A handwritten signature in blue ink that reads "Arpit Tyagi". The signature is written in a cursive style with a large, sweeping underline.



Fifth Force

www.fifthforce.in

Ph: +917003720431

Reg. No: KL04482N2022000001

CERTIFICATE OF APPRECIATION

Presented to

Arpit Tyagi

For successfully completing 6 Weeks Summer Training' 2022 On
"C++ with Data Structure and Algorithm"

Sudipta Paitandi

Sudipta Paitandi

Program In-charge

Victor Banerjee

Victor Banerjee

Chief of Operations

ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to Fifth Force Courses (Prof. Chandan Mukherjee) for his exemplary guidance, monitoring and constant encouragement throughout the course. The blessing, help and guidance given by him, time to time shall carry me a long way in the journey of life on which I am about to embark.

I would also like to extend my gratitude to our Lovely Professional University for allowing me such a course which will improve my programming skill.

Finally, I would like to thank my parents and friends who have helped me with their valuable suggestions and guidance for choosing this course.

Date: 15/06/2022

Arpit Tyagi

Index

| S. No | Title | Page |
|-------|--|------|
| 1 | Declaration by Student | 1 |
| 2 | Training Certification from organization | 2 |
| 3 | Acknowledgement | 3 |
| 4 | Index | 4 |
| 5 | List of Figures/ Charts/tables | 5 |
| 6 | Chapter-1 | 7 |
| 7 | Chapter-2 | 18 |
| 8 | Chapter-3 | 28 |
| 9 | Chapter-4 | 47 |
| 10 | References | 48 |

List of Figures/ Charts/ Tables

1.1 = The table below shows the fundamental data types, their meaning, and their sizes (in bytes)

1.2 = The table shows the C++ modified data types list

1.3 = The table shows how loops work with an example

3.1 = The table shows difference between linear data structure and non linear data structure

3.2 = The table shows time complexity in Linked lists.

Introduction of the Course Undertaken

C++ with Data Structure & Algorithm: Foundation To Advanced!

About Course

Which programming language is frequently regarded as a source of pride among software developers? C++

What programming language can you learn that, when added to your resume, will almost always land you a job interview? C++

Which programming language is consistently ranked in the top five programming languages by popularity, and has been in the top ten for nearly two decade.

Why should you learn C++?

Much, if not the majority, of today's software, is still written in C++ and has been for many, many years. C++ is not only popular, but it is also a very relevant language. If you visit GitHub, you will notice that there are a large number of active C++ repositories, and C++ is also very active on stack overflow. Many popular software titles are written entirely or partially in C++. These include the operating systems Windows, Linux, and Mac OSX! Many Adobe products, including Photoshop and Illustrator, as well as the MySQL and MongoDB database engines, are written in C++.

Many of the world's leading technology companies use C++ for their products and internal research and development. Amazon, Apple, Microsoft, PayPal, Google, Facebook, Oracle, and many others are among them.

One of the foundations of the software industry is a data structure. That is the distinction between a regular software engineer and a professional software engineer. However, according to a survey, 90% of software engineers do not understand data structures and algorithms.

This is why we developed the C++ with Data Structure: Foundation to Advanced Course.

Not only do we teach you about data structures, but we also teach you how to think correctly! That is extremely significant!

Chapter 1

INTRODUCTION TO C++

C++ Data Types

In C++, data types are declarations for variables. This determines the type and size of data associated with variables. For example,

➤ `int age = 13;`

Here, age is a variable of type int. Meaning, the variable can only store integers of either 2 or 4 bytes.

C++ Fundamental Data Types

| Data Type | Meaning | Size (in Bytes) |
|-----------|-----------------------|-----------------|
| int | Integer | 2 or 4 |
| float | Floating-point | 4 |
| double | Double Floating-point | 8 |
| char | Character | 1 |
| wchar_t | Wide Character | 2 |
| bool | Boolean | 1 |
| void | Empty | 0 |

Table - 1.1

1. C++ int

The int keyword is used to indicate integers.

Its size is usually 4 bytes. Meaning, it can store values from -2147483648 to 2147483647.

For example,

➤ `int salary = 85000;`

2. C++ float and double

float and double are used to store floating-point numbers (decimals and exponentials).

The size of float is 4 bytes and the size of double is 8 bytes. Hence, double has two times the precision of float. To learn more, visit [C++ float and double](#).

For example,

- `float area = 64.74;`
- `double volume = 134.64534;`

As mentioned above, these two data types are also used for exponentials. For example,

- `double distance = 45E12` // 45E12 is equal to 45×10^{12}
-

3. C++ char

Keyword `char` is used for characters.

Its size is 1 byte.

Characters in C++ are enclosed inside single quotes `' '`.

For example,

- `char test = 'h';`
-

4. C++ wchar_t

Wide character `wchar_t` is similar to the `character(char)` data type, except its size is 2 bytes instead of 1. It is used to represent characters that require more memory to represent them than a single `char`.

For example,

- `wchar_t test = L'ד'` // storing Hebrew character;

Notice the letter `L` before the quotation marks.

5. C++ bool

The `bool` data type has one of two possible values: `true` or `false`. Booleans are used in conditional statements and loops (which we will learn in later chapters).

For example,

- `bool cond = false;`
-

6. C++ void

The `void` keyword indicates an absence of data. It means "nothing" or "no value".

We will use `void` when we learn about functions and pointers.

C++ Type Modifiers

We can further modify some of the fundamental data types by using type modifiers. There are 4 type modifiers in C++. They are:

- signed
- unsigned
- short
- long

We can modify the following data types with the above modifiers:

- int
- double
- char

C++ Modified Data Types List

| Data Type | Size (in Bytes) | Meaning |
|--------------------|-----------------|---|
| signed int | 4 | used for integers (equivalent to int) |
| unsigned int | 4 | can only store positive integers |
| short | 2 | used for small integers (range -32768 to 32767) |
| unsigned short | 2 | used for small positive integers (range 0 to 65,535) |
| long | at least 4 | used for large integers (equivalent to long int) |
| unsigned long | 4 | used for large positive integers or 0 (equivalent to unsigned long int) |
| long long | 8 | used for very large integers (equivalent to long long int). |
| unsigned long long | 8 | used for very large positive integers or 0 (equivalent to unsigned long long int) |
| long double | 12 | used for large floating-point numbers |
| signed char | 1 | used for characters (guaranteed range -127 to 127) |
| unsigned char | 1 | used for characters (range 0 to 255) |

Table 1.2

Let's see a few examples.

- `long b = 4523232;`
 - `long int c = 2345342;`
 - `long double d = 233434.56343;`
 - `short d = 3434233; // Error! out of range`
 - `unsigned int a = -5; // Error! can only store positive numbers or 0`
-

Derived Data Types

Data types that are derived from fundamental data types are derived types. For example: arrays, pointers, function types, structures, etc.

C++ Basic Input/Output

C++ Output

In C++, `cout` sends formatted output to standard output devices, such as the screen. We use the `cout` object along with the `<<` operator for displaying output.

Example 1: String Output

- `#include <iostream>`
- `using namespace std;`
- `int main() {`
- `// prints the string enclosed in double quotes`
- `cout << "This is C++ Programming";`
- `return 0;`
- `}`

Run Code

Output

❖ This is C++ Programming

How does this program work?

We first include the `iostream` header file that allows us to display output.

The `cout` object is defined inside the `std` namespace. To use the `std` namespace, we used the `using namespace std;` statement.

Every C++ program starts with the `main()` function. The code execution begins from the start of the `main()` function. `cout` is an object that prints the string inside quotation marks `" "`. It is followed by the `<<` operator.

`return 0;` is the "exit status" of the `main()` function. The program ends with this statement, however, this statement is not mandatory.

Note: If we don't include the `using namespace std;` statement, we need to use `std::cout` instead of `cout`.

This is the preferred method as using the `std` namespace can create potential problems.

However, we have used the `std` namespace in our tutorials in order to make the codes more readable.

- `#include <iostream>`
- `int main() {`
- `// prints the string enclosed in double quotes`
- `std::cout << "This is C++ Programming";`
- `return 0;`
- `}`

Run Code

The `endl` manipulator is used to insert a new line. That's why each output is displayed in a new line.

The `<<` operator can be used more than once if we want to print different variables, strings and so on in a single statement. For example:

- `cout << "character: " << ch << endl;`
-

C++ Input

In C++, `cin` takes formatted input from standard input devices such as the keyboard. We use the `cin` object along with the `>>` operator for taking input.

Example 3: Integer Input/Output

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

```
➤ int main() {
➤ int num;
➤ cout << "Enter an integer: ";
➤ cin >> num; // Taking input
➤ cout << "The number is: " << num;
➤ return 0;
➤ }
```

Run Code

Output

```
❖ Enter an integer: 70
❖ The number is: 70
```

In the program, we used

```
➤ cin >> num;
```

to take input from the user. The input is stored in the variable num. We use the >> operator with cin to take input.

C++ if, if...else and Nested if...else

In computer programming, we use the if...else statement to run one block of code under certain conditions and another block of code under different conditions.

For example, assigning grades (A, B, C) based on marks obtained by a student.

if the percentage is above 90, assign grade A

if the percentage is above 75, assign grade B

if the percentage is above 65, assign grade C

How if Statement Works

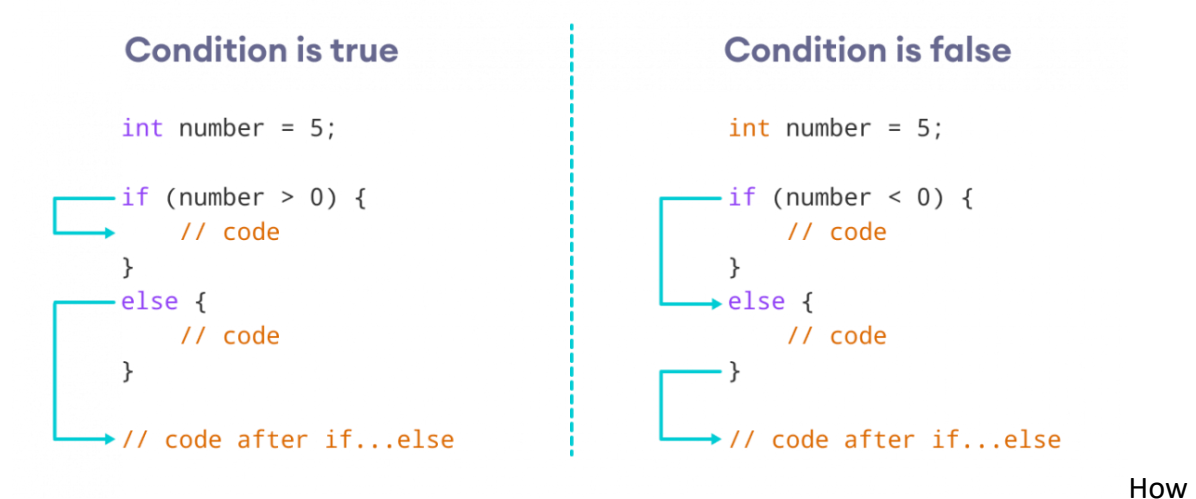
C++ if...else

The if statement can have an optional else clause. Its syntax is:

```
➤ if (condition) {
```

- // block of code if condition is true
- }
- else {
- // block of code if condition is false
- }

The if..else statement evaluates the condition inside the parenthesis.



if...else Statement Works

If the condition evaluates true,

the code inside the body of if is executed

the code inside the body of else is skipped from execution

If the condition evaluates false,

the code inside the body of else is executed

the code inside the body of if is skipped from execution

Example : C++ if...else Statement

- #include <iostream>
- using namespace std;
- int main() {
- int number;
- cout << "Enter an integer: ";
- cin >> number;
- if (number >= 0) {

```

➤ cout << "You entered a positive integer: " << number << endl;
➤ }
➤ else {
➤ cout << "You entered a negative integer: " << number << endl;
➤ }
➤ cout << "This line is always printed.";
➤ return 0;
➤ }

```

Run Code

```

❖ Output 1
❖ Enter an integer: 4
❖ You entered a positive integer: 4.

```

This line is always printed.

C++ for Loop

In computer programming, loops are used to repeat a block of code. For example, let's say we want to show a message 100 times. Then instead of writing the print statement 100 times, we can use a loop. There are 3 types of loops in C++.

1. for loop
2. while loop
3. do...while loop

We will see an example for “for loop”.

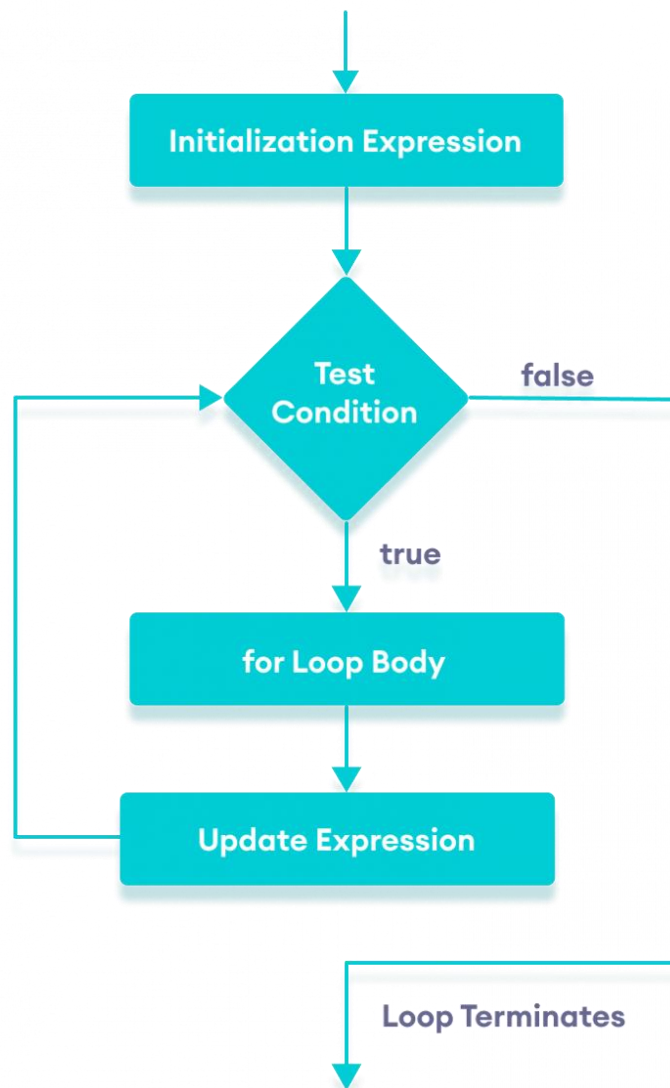
C++ for loop

The syntax of for-loop is:

```

for (initialization; condition; update) {
    // body of-loop
}

```



Flowchart of for loop in C++

Example : Printing Numbers From 1 to 5

- `#include <iostream>`
- `using namespace std;`
- `int main() {`
- `for (int i = 1; i <= 5; ++i) {`
- `cout << i << " ";`
- `}`
- `return 0;`
- `}`

Run Code

❖ Output

❖ 1 2 3 4 5

Here is how this program works

| Iteration | Variable | i <= 5 | Action |
|-----------|----------|--------|------------------------------------|
| 1st | i = 1 | true | 1 is printed. i is increased to 2. |
| 2nd | i = 2 | true | 2 is printed. i is increased to 3. |
| 3rd | i = 3 | true | 3 is printed. i is increased to 4. |
| 4th | i = 4 | true | 4 is printed. i is increased to 5. |
| 5th | i = 5 | true | 5 is printed. i is increased to 6. |
| 6th | i = 6 | false | The loop is terminated |

Table 1.3

C++ Functions

A function is a block of code that performs a specific task. Suppose we need to create a program to create a circle and color it. We can create two functions to solve this problem:

1. a function to draw the circle
2. a function to color the circle

Dividing a complex problem into smaller chunks makes our program easy to understand and reusable. There are two types of function:

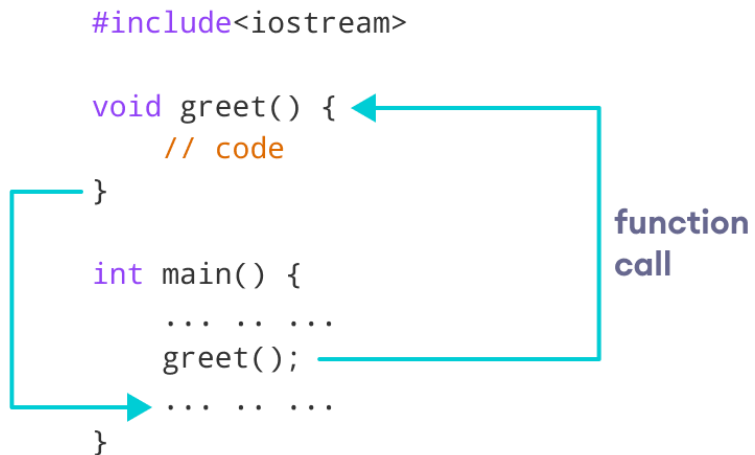
1. Standard Library Functions: Predefined in C++
2. User-defined Function: Created by users

we will focus on user-defined functions.

C++ User-defined Function

C++ allows the programmer to define their own function. A user-defined function groups code to perform a specific task and that group of code is given a name (identifier). When the

function is invoked from any part of the program, it all executes the codes defined in the body of the function.



Example : Display a Text

- #include <iostream>
- using namespace std;
- // declaring a function
- void greet() {
- cout << "Hello there!";
- }
- int main() {
- // calling the function
- greet();
- return 0;
- }

Run Code

- ❖ Output
- ❖ Hello there!

Chapter 2

ADVANCED CONCEPTS OF C++

C++ Function Overloading

In C++, two functions can have the same name if the number and/or type of arguments passed is different. These functions having the same name but different arguments are known as overloaded functions. For example:

- `// same name different arguments`
- `int test() { }`
- `int test(int a) { }`
- `float test(double a) { }`
- `int test(int a, double b) { }`

Here, all 4 functions are overloaded functions. Notice that the return types of all these 4 functions are not the same. Overloaded functions may or may not have different return types but they must have different arguments. For example,

- `// Error code`
- `int test(int a) { }`
- `double test(int b){ }`

Here, both functions have the same name, the same type, and the same number of arguments. Hence, the compiler will throw an error.

Example 1: Overloading Using Different Types of Parameter

- `// Program to compute absolute value`
- `// Works for both int and float`
- `#include <iostream>`
- `using namespace std;`
- `// function with float type parameter`
- `float absolute(float var){`
- `if (var < 0.0)`
- `var = -var;`
- `return var;`

```

➤ }
➤ // function with int type parameter
➤ int absolute(int var) {
➤   if (var < 0)
➤     var = -var;
➤   return var;
➤ }
➤ int main() { // call function with int type parameter
➤   cout << "Absolute value of -5 = " << absolute(-5) << endl;
➤   // call function with float type parameter
➤   cout << "Absolute value of 5.5 = " << absolute(5.5f) << endl;
➤   return 0;
➤ }

```

Run Code

Output

- ❖ Absolute value of -5 = 5
- ❖ Absolute value of 5.5 = 5.5

```

float absolute(float var) { ←
    // code
}

int absolute(int var) { ←
    // code
}

int main() {
    absolute(-5);
    absolute(5.5f);
    ...
}

```

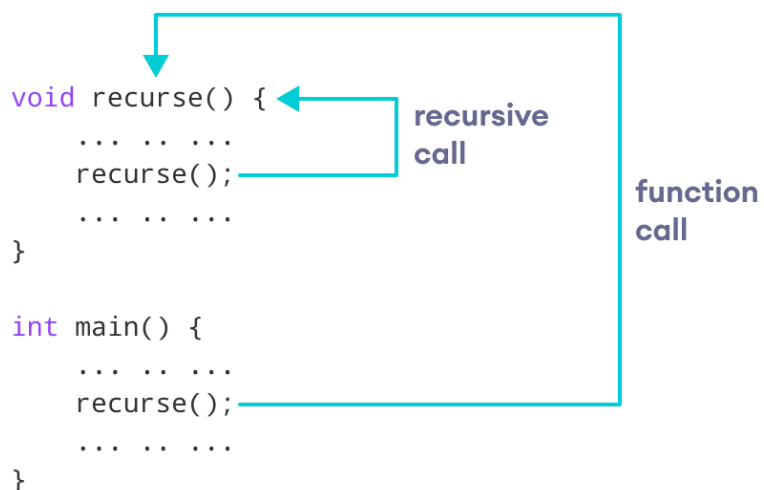
C++ Recursion

A function that calls itself is known as a recursive function. And, this technique is known as recursion.

Working of Recursion in C++

- void recurse()
- {
-
- recurse();
-
- }

The figure below shows how recursion works by calling itself over and over again.



Example : Factorial of a Number Using Recursion

- // Factorial of n = 1*2*3*...*n
- #include <iostream>
- using namespace std;
- int factorial(int);
- int main() {
- int n, result;
- cout << "Enter a non-negative number: ";

```

➤ cin >> n;
➤ result = factorial(n);
➤ cout << "Factorial of " << n << " = " << result;
➤ return 0;
➤ }
➤ int factorial(int n) {
➤ if (n > 1) {
➤ return n * factorial(n - 1);
➤ } else {
➤ return 1;
➤ }
➤ }

```

Run Code

Output

- ❖ Enter a non-negative number: 4
- ❖ Factorial of 4 = 24

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.

C++ Arrays

In C++, an array is a variable that can store multiple values of the same type. For example, Suppose a class has 27 students, and we need to store the grades of all of them. Instead of creating 27 separate variables, we can simply create an array:

➤ `double grade[27];`

Here, `grade` is an array that can hold a maximum of 27 elements of `double` type. In C++, the size and type of arrays cannot be changed after its declaration.

C++ Array Declaration

`dataType arrayName[arraySize];`

For example,

`int x[6];`

Here, `int` - type of element to be stored

1. `x` - name of the array
 2. `6` - size of the array
-



C++ Array

Another method to initialize array during declaration:

`// declare and initialize an array`

`int x[] = {19, 10, 8, 17, 9, 15};`

`int x[6] = {19, 10, 8};`

Example : Displaying Array Elements

- `#include <iostream>`
- `using namespace std;`
- `int main() {`
- `int numbers[5] = {7, 5, 6, 12, 35};`
- `cout << "The numbers are: ";`
- `// Printing array elements`
- `// using range based for loop`
- `for (const int &n : numbers) {`
- `cout << n << " ";`

```

➤ }
➤ cout << "\nThe numbers are: ";
➤ // Printing array elements
➤ // using traditional for loop
➤ for (int i = 0; i < 5; ++i) {
➤     cout << numbers[i] << " ";
➤ }
➤ return 0;
➤ }

```

Run Code

Output

- ❖ The numbers are: 7 5 6 12 35
- ❖ The numbers are: 7 5 6 12 35&n, however, uses the memory address of the array elements to access their data without copying them to a new variable. This is memory-efficient.

C++ Pointers

In C++, pointers are variables that store the memory addresses of other variables. If we have a variable `var` in our program, `&var` will give us its address in the memory. For example, Here is how we can declare pointers.

```
➤ int *pointVar;
```

Here, we have declared a pointer `pointVar` of the `int` type.

We can also declare pointers in the following way.

```
➤ int* pointVar; // preferred syntax
```

Let's take another example of declaring pointers.

```
➤ int* pointVar, p;
```

Here, we have declared a pointer `pointVar` and a normal variable `p`.

Example : Working of C++ Pointers

```

➤ #include <iostream>
➤ using namespace std;

```



```

➤ int main() {
➤ int var = 5;
➤ // declare pointer variable
➤ int* pointVar;
➤ // store address of var
➤ pointVar = &var;
➤ // print value of var
➤ cout << "var = " << var << endl;
➤ // print address of var
➤ cout << "Address of var (&var) = " << &var << endl
➤ << endl;
➤ // print pointer pointVar
➤ cout << "pointVar = " << pointVar << endl;
➤ // print the content of the address pointVar points to
➤ cout << "Content of the address pointed to by pointVar (*pointVar) = " << *pointVar
  << endl;
➤ return 0;
➤ }

```

Run Code

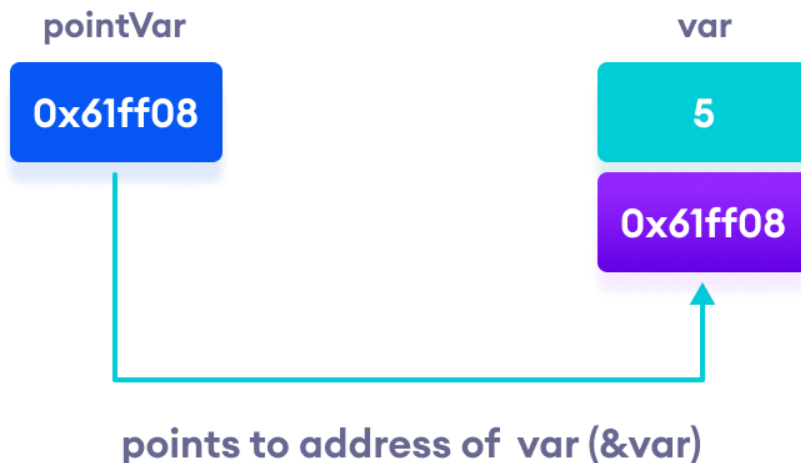
Output

```

❖ var = 5
❖ Address of var (&var) = 0x61ff08
❖ pointVar = 0x61ff08

```

Content of the address pointed to by pointVar (*pointVar) = 5



Polymorphism

Polymorphism allows us to create consistent code. For example, Suppose we need to calculate the area of a circle and a square. To do so, we can create a Shape class and derive two classes Circle and Square from it. In this case, it makes sense to create a function having the same name `calculateArea()` in both the derived classes rather than creating functions with different names, thus making our code more consistent. 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).

Example: C++ Abstract Class and Pure Virtual Function

- `// 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;
➤ }

```

Run Code

Output

❖ Enter the length of the square: 4

❖ Area of square: 16

❖ Enter radius of the circle: 5

❖ Area of circle: 78.5

In this program, virtual float calculateArea() = 0; inside the Shape class is a pure virtual function. That's why we must provide the implementation of calculateArea() in both of our derived classes, or else we will get an error.

Chapter 3

Data Structure and Types

What are Data Structures?

Data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently. Depending on your requirement and project, it is important to choose the right data structure for your project. For example, if you want to store data sequentially in the memory, then you can go for the Array data structure.

| memory locations | | | | | | |
|------------------|------|------|------|------|------|------|
| 1004 | 1005 | 1006 | 1007 | 1008 | 1009 | 1010 |
| ... | 2 | 1 | 5 | 3 | 4 | ... |
| | 0 | 1 | 2 | 3 | 4 | |
| index | | | | | | |

Array data Structure Representation

Note: Data structure and data types are slightly different. Data structure is the collection of data types arranged in a specific order.

Types of Data Structure

Basically, data structures are divided into two categories:

1. Linear data structure
2. Non-linear data structure
3. Let's learn about each type in detail.

Linear data structures

In linear data structures, the elements are arranged in sequence one after the other. Since elements are arranged in particular order, they are easy to implement.

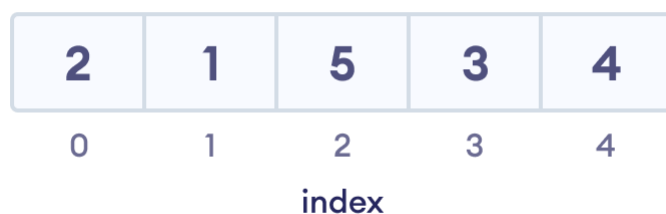
However, when the complexity of the program increases, the linear data structures might not be the best choice because of operational complexities.

Popular linear data structures are:

1. Array Data Structure

In an array, elements in memory are arranged in continuous memory. All the elements of an array are of the same type. And, the type of elements that can be stored in the form of arrays is determined by the programming language.

To learn more, visit [Java Array](#).



An array with each element

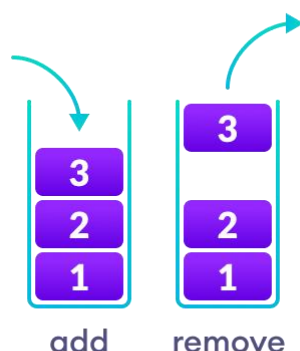
represented by an index

2. Stack Data Structure

In stack data structure, elements are stored in the LIFO principle. That is, the last element stored in a stack will be removed first.

It works just like a pile of plates where the last plate kept on the pile will be removed first.

To learn more, visit [Stack Data Structure](#).



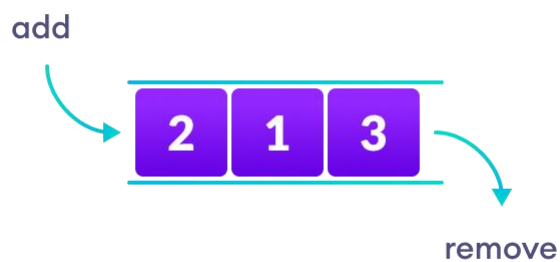
In a stack, operations can be perform only from one end

(top here).

3. Queue Data Structure

Unlike stack, the queue data structure works in the FIFO principle where first element stored in the queue will be removed first.

It works just like a queue of people in the ticket counter where first person on the queue will get the ticket first. To learn more, visit [Queue Data Structure](#).

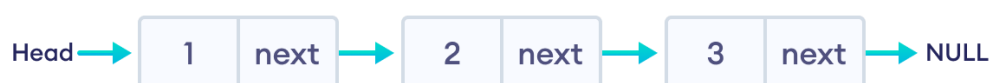


In a queue, addition and removal are performed from separate ends.

4. Linked List Data Structure

In linked list data structure, data elements are connected through a series of nodes. And, each node contains the data items and address to the next node.

To learn more, visit [Linked List Data Structure](#).



Non linear data structures

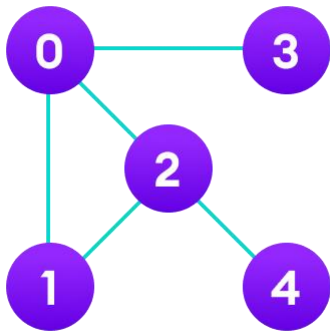
Unlike linear data structures, elements in non-linear data structures are not in any sequence. Instead they are arranged in a hierarchical manner where one element will be connected to one or more elements.

Non-linear data structures are further divided into graph and tree based data structures.

1. Graph Data Structure

In graph data structure, each node is called vertex and each vertex is connected to other vertices through edges.

To learn more, visit [Graph Data Structure](#).



Graph data structure example

Popular Graph Based Data Structures:

Spanning Tree and Minimum Spanning Tree

Strongly Connected Components

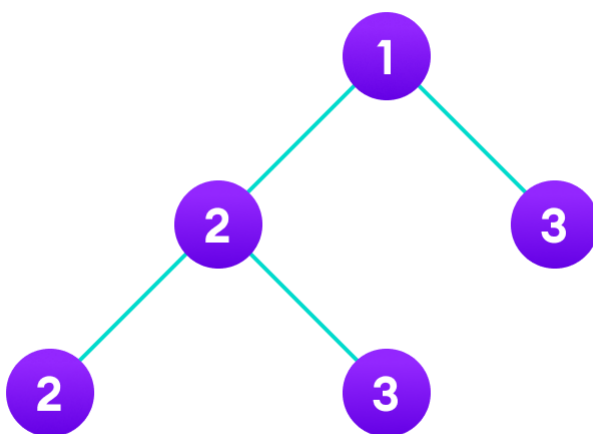
Adjacency Matrix

Adjacency List

2. Trees Data Structure

Similar to a graph, a tree is also a collection of vertices and edges. However, in tree data structure, there can only be one edge between two vertices.

To learn more, visit [Tree Data Structure](#).



Popular Tree based Data Structure

1. Binary Tree
2. Binary Search Tree

3. AVL Tree
4. B-Tree
5. B+ Tree
6. Red-Black Tree

Linear Vs Non-linear Data Structures

Now that we know about linear and non-linear data structures, let's see the major differences between them.

| Linear Data Structures | Non Linear Data Structures |
|---|--|
| The data items are arranged in sequential order, one after the other. | The data items are arranged in non-sequential order (hierarchical manner). |
| All the items are present on the single layer. | The data items are present at different layers. |
| It can be traversed on a single run. That is, if we start from the first element, we can traverse all the elements sequentially in a single pass. | It requires multiple runs. That is, if we start from the first element it might not be possible to traverse all the elements in a single pass. |
| The memory utilization is not efficient. | Different structures utilize memory in different efficient ways depending on the need. |
| The time complexity increase with the data size. | Time complexity remains the same. |
| Example: Arrays, Stack, Queue | Example: Tree, Graph, Map |

Table 3.1

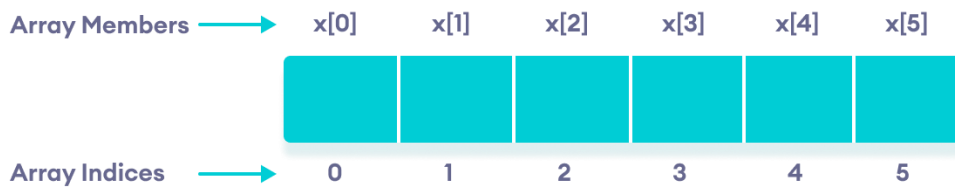
C++ Arrays

In C++, an array is a variable that can store multiple values of the same type. Each element in an array is associated with a number. The number is known as an array index. We can access elements of an array by using those indices.

// syntax to access array elements

```
array[index];
```

Consider the array x we have seen above.



Few Things to Remember:

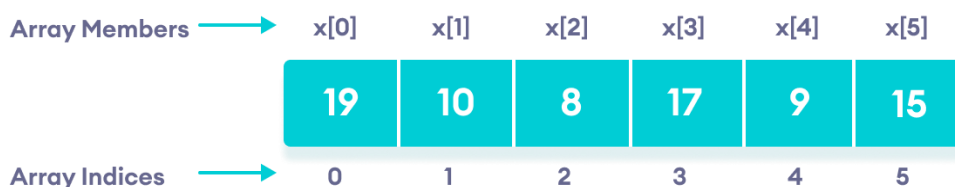
1. The array indices start with 0. Meaning `x[0]` is the first element stored at index 0.
2. If the size of an array is `n`, the last element is stored at index `(n-1)`. In this example, `x[5]` is the last element.
3. Elements of an array have consecutive addresses. For example, suppose the starting address of `x[0]` is 2120. Then, the address of the next element `x[1]` will be 2124, the address of `x[2]` will be 2128, and so on. Here, the size of each element is increased by 4. This is because the size of `int` is 4 bytes.

C++ Array Initialization

In C++, it's possible to initialize an array during declaration. For example,

// declare and initialize an array

```
int x[6] = {19, 10, 8, 17, 9, 15};
```



C++ Array elements and their data

Another method to initialize array during declaration:

// declare and initialize an array

```
int x[] = {19, 10, 8, 17, 9, 15};
```

Here, we have not mentioned the size of the array. In such cases, the compiler automatically computes the size.

Example : Display Sum and Average of Array Elements Using for Loop

```
➤ #include <iostream>
➤ using namespace std;
➤ int main() {
➤ // initialize an array without specifying size
➤ double numbers[] = {7, 5, 6, 12, 35, 27};
➤ double sum = 0;
➤ double count = 0;
➤ double average;
➤ cout << "The numbers are: ";
➤ // print array elements
➤ // use of range-based for loop
➤ for (const double &n : numbers) {
➤ cout << n << " ";
➤ // calculate the sum
➤ sum += n;
➤ // count the no. of array elements
➤ ++count;
➤ }
➤ // print the sum
➤ cout << "\nTheir Sum = " << sum << endl;
➤ // find the average
➤ average = sum / count;
➤ cout << "Their Average = " << average << endl;
➤ return 0;
➤ }
```

Run Code

Output

- ❖ The numbers are: 7 5 6 12 35 27
- ❖ Their Sum = 92
- ❖ Their Average = 15.3333

In this program:

We have initialized a double array named numbers but without specifying its size. We also declared three double variables sum, count, and average.

C-strings

In C programming, the collection of characters is stored in the form of arrays. This is also supported in C++ programming. Hence it's called C-strings.

C-strings are arrays of type char terminated with null character, that is, \0 (ASCII value of null character is 0).

How to define a C-string?

- `char str[] = "C++";`

In the above code, str is a string and it holds 4 characters.

Although, "C++" has 3 character, the null character \0 is added to the end of the string automatically.

Alternative ways of defining a string

- `char str[4] = "C++";`
- `char str[] = {'C','+', '+', '\0'};`
- `char str[4] = {'C','+', '+', '\0'};`

Like arrays, it is not necessary to use all the space allocated for the string. For example:

- `char str[100] = "C++";`

Example : C++ String to read a line of text

C++ program to read and display an entire line entered by user.

- `#include <iostream>`
- `using namespace std;`
- `int main()`
- `{`
- `char str[100];`
- `cout << "Enter a string: ";`
- `cin.get(str, 100);`
- `cout << "You entered: " << str << endl;`

```
➤ return 0;  
➤ }
```

Output

- ❖ Enter a string: Programming is fun.
- ❖ You entered: Programming is fun.

To read the text containing blank space, `cin.get` function can be used. This function takes two arguments. First argument is the name of the string (address of first element of string) and second argument is the maximum size of the array. In the above program, `str` is the name of the string and 100 is the maximum size of the array.

Linked list Data Structure

A linked list is a linear data structure that includes a series of connected nodes. Here, each node stores the data and the address of the next node. For example,



Linked list Data Structure

You have to start somewhere, so we give the address of the first node a special name called `HEAD`. Also, the last node in the linked list can be identified because its next portion points to `NULL`.

Linked lists can be of multiple types: singly, doubly, and circular linked list. In this article, we will focus on the singly linked list. To learn about other types, visit [Types of Linked List](#).

Note: You might have played the game Treasure Hunt, where each clue includes the information about the next clue. That is how the linked list operates.

Representation of Linked List

Let's see how each node of the linked list is represented. Each node consists:

1. A data item
2. An address of another node

We wrap both the data item and the next node reference in a struct as:

- struct node
- {
- int data;
- struct node *next;
- };

Linked list Representation

The power of a linked list comes from the ability to break the chain and rejoin it. E.g. if you wanted to put an element 4 between 1 and 2, the steps would be:

1. Create a new struct node and allocate memory to it.
2. Add its data value as 4
3. Point its next pointer to the struct node containing 2 as the data value
4. Change the next pointer of "1" to the node we just created.

Doing something similar in an array would have required shifting the positions of all the subsequent elements.

In python and Java, the linked list can be implemented using classes as shown in the codes below.

Time Complexity

| | Worst case | Average Case |
|----------|------------|--------------|
| Search | $O(n)$ | $O(n)$ |
| Insert | $O(1)$ | $O(1)$ |
| Deletion | $O(1)$ | $O(1)$ |

Table 3.2

Space Complexity: $O(n)$

Linked List Applications

1. Dynamic memory allocation
2. Implemented in stack and queue
3. In undo functionality of softwares
4. Hash tables, Graphs

Stack Data Structure

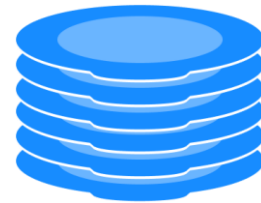
A stack is a linear data structure that follows the principle of Last In First Out (LIFO). This means the last element inserted inside the stack is removed first. You can think of the stack data structure as the pile of plates on top of another.

Stack representation similar to a pile of plate

Here, you can:

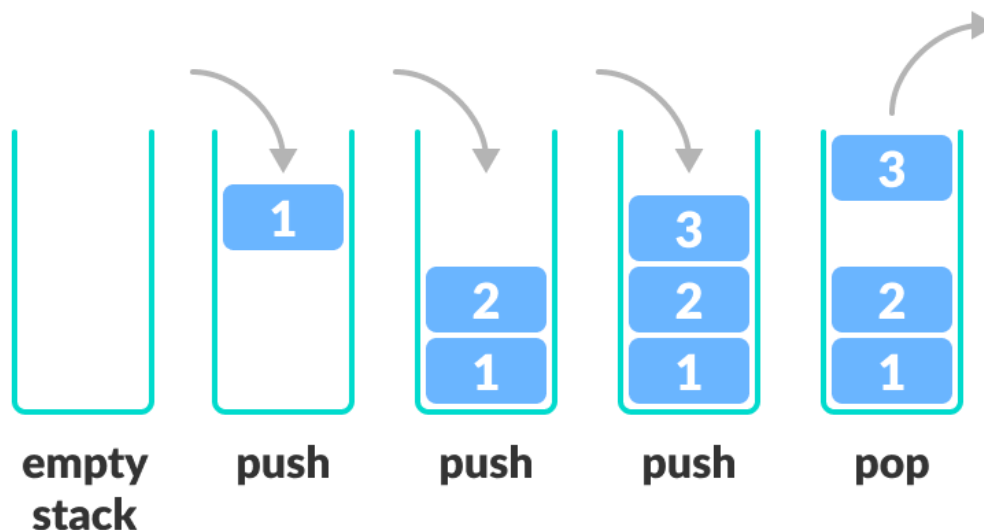
- Put a new plate on top
- Remove the top plate
- And, if you want the plate at the bottom, you must first remove all the plates on top.

This is exactly how the stack data structure works.



LIFO Principle of Stack

In programming terms, putting an item on top of the stack is called push and removing an item is called pop.



Stack Push and Pop Operations

In the above image, although item 3 was kept last, it was removed first. This is exactly how the LIFO (Last In First Out) Principle works. We can implement a stack in any programming language like C, C++, Java, Python or C#, but the specification is pretty much the same.

Basic Operations of Stack

There are some basic operations that allow us to perform different actions on a stack.

- Push: Add an element to the top of a stack
- Pop: Remove an element from the top of a stack
- IsEmpty: Check if the stack is empty
- IsFull: Check if the stack is full
- Peek: Get the value of the top element without removing it

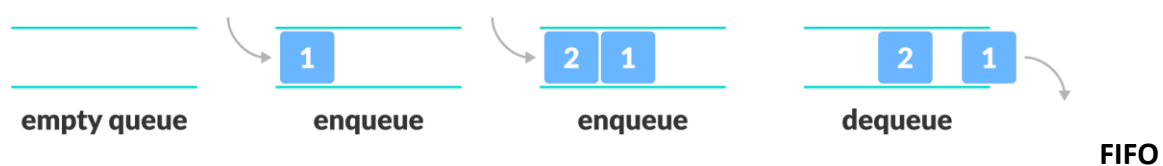
Applications of Stack Data Structure

Although stack is a simple data structure to implement, it is very powerful. The most common uses of a stack are:

1. To reverse a word - Put all the letters in a stack and pop them out. Because of the LIFO order of stack, you will get the letters in reverse order.
2. In compilers - Compilers use the stack to calculate the value of expressions like $2 + 4 / 5 * (7 - 9)$ by converting the expression to prefix or postfix form.
3. In browsers - The back button in a browser saves all the URLs you have visited previously in a stack. Each time you visit a new page, it is added on top of the stack. When you press the back button, the current URL is removed from the stack, and the previous URL is accessed.

Queue Data Structure

A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket. Queue follows the First In First Out (FIFO) rule - the item that goes in first is the item that comes out first.



Representation of Queue

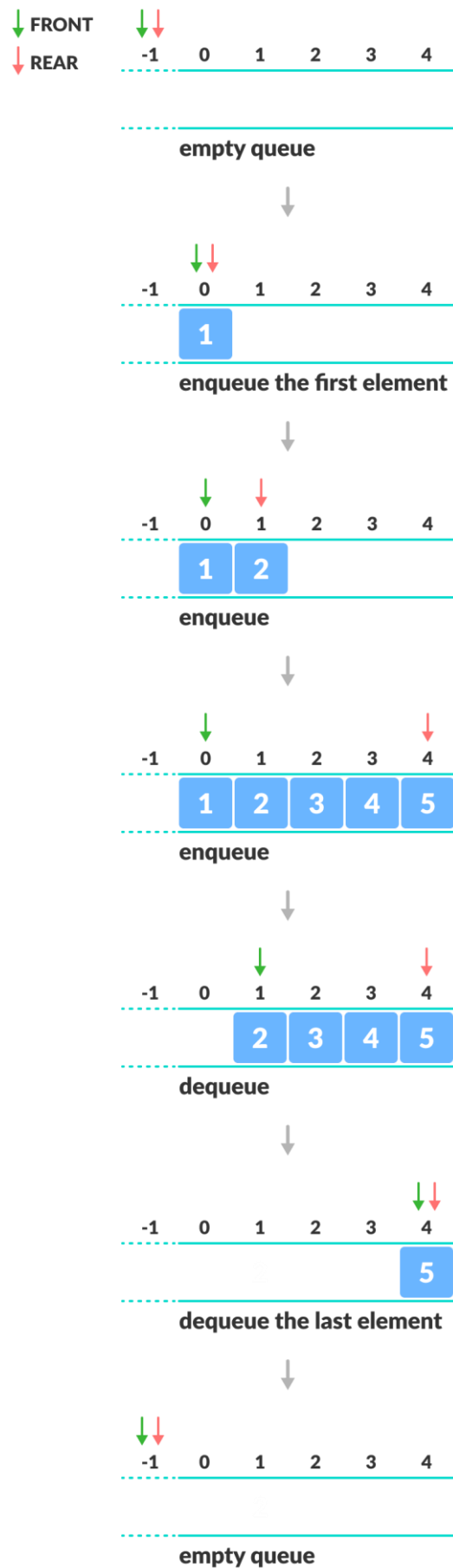
In the above image, since 1 was kept in the queue before 2, it is the first to be removed from the queue as well. It follows the FIFO rule. In programming terms, putting items in the queue is called enqueue, and removing items from the queue is called dequeue.

We can implement the queue in any programming language like C, C++, Java, Python or C#, but the specification is pretty much the same.

Basic Operations of Queue

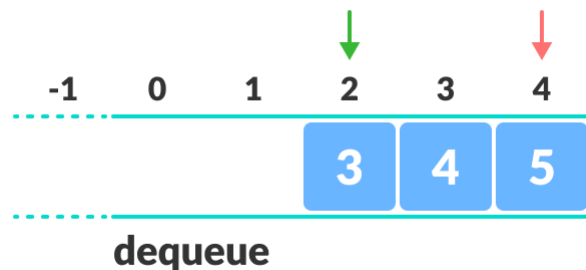
A queue is an object (an abstract data structure - ADT) that allows the following operations:

- Enqueue: Add an element to the end of the queue
- Dequeue: Remove an element from the front of the queue
- IsEmpty: Check if the queue is empty
- IsFull: Check if the queue is full
- Peek: Get the value of the front of the queue without removing it



Limitations of Queue

As you can see in the image below, after a bit of enqueueing and dequeuing, the size of the queue has been reduced.



Applications of Queue

- CPU scheduling, Disk Scheduling
- When data is transferred asynchronously between two processes. The queue is used for synchronization. For example: IO Buffers, pipes, file IO, etc
- Handling of interrupts in real-time systems.
- Call Center phone systems use Queues to hold people calling them in order.

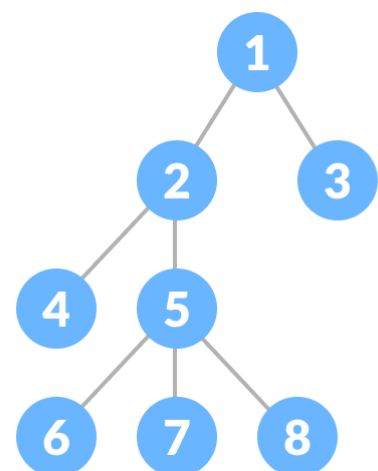
Tree Data Structure

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.

Why Tree Data Structure?

Other data structures such as arrays, linked list, stack, and queue are linear data structures that store data sequentially. In order to perform any operation in a linear data structure, the time complexity increases with the increase in the data size. But, it is not acceptable in today's computational world.

Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure.



Degree of a Node

The degree of a node is the total number of branches of that node.

Types of Tree

- Binary Tree
 - Binary Search Tree
 - AVL Tree
 - B-Tree
-

Tree Traversal

In order to perform any operation on a tree, you need to reach to the specific node. The tree traversal algorithm helps in visiting a required node in the tree.

To learn more, please visit [tree traversal](#).

Tree Applications

- Binary Search Trees(BSTs) are used to quickly check whether an element is present in a set or not.
- Heap is a kind of tree that is used for heap sort.
- A modified version of a tree called Tries is used in modern routers to store routing information.
- Most popular databases use B-Trees and T-Trees, which are variants of the tree structure we learned above to store their data
- Compilers use a syntax tree to validate the syntax of every program you write.

Dynamic Programming

Dynamic Programming is a technique in computer programming that helps to efficiently solve a class of problems that have overlapping subproblems and optimal substructure property.

If any problem can be divided into subproblems, which in turn are divided into smaller subproblems, and if there are overlapping among these subproblems, then the solutions to

these subproblems can be saved for future reference. In this way, efficiency of the CPU can be enhanced. This method of solving a solution is referred to as dynamic programming. Such problems involve repeatedly calculating the value of the same subproblems to find the optimum solution.

Dynamic Programming Example

Let's find the fibonacci sequence upto 5th term. A fibonacci series is the sequence of numbers in which each number is the sum of the two preceding ones. For example, 0,1,1, 2, 3. Here, each number is the sum of the two preceding numbers.

Algorithm

Let n be the number of terms.

1. If $n \leq 1$, return 1.
2. Else, return the sum of two preceding numbers.

We are calculating the fibonacci sequence up to the 5th term.

The first term is 0.

The second term is 1.

The third term is sum of 0 (from step 1) and 1(from step 2), which is 1.

The fourth term is the sum of the third term (from step 3) and second term (from step 2) i.e. $1 + 1 = 2$.

The fifth term is the sum of the fourth term (from step 4) and third term (from step 3) i.e. $2 + 1 = 3$.

Hence, we have the sequence 0,1,1, 2, 3. Here, we have used the results of the previous steps as shown below. This is called a dynamic programming approach.

$$F(0) = 0$$

$$F(1) = 1$$

$$F(2) = F(1) + F(0)$$

$$F(3) = F(2) + F(1)$$

$$F(4) = F(3) + F(2)$$

How Dynamic Programming Works

Dynamic programming works by storing the result of subproblems so that when their solutions are required, they are at hand and we do not need to recalculate them.

This technique of storing the value of subproblems is called memoization. By saving the values in the array, we save time for computations of sub-problems we have already come across.

```
var m = map(0 → 0, 1 → 1)
function fib(n)
  if key n is not in map m
    m[n] = fib(n - 1) + fib(n - 2)
  return m[n]
```

Dynamic programming by memoization is a top-down approach to dynamic programming. By reversing the direction in which the algorithm works i.e. by starting from the base case and working towards the solution, we can also implement dynamic programming in a bottom-up manner.

```
function fib(n)
  if n = 0
    return 0
  else
    var prevFib = 0, currFib = 1
    repeat n - 1 times
      var newFib = prevFib + currFib
      prevFib = currFib
      currFib = newFib
    return currFib
```

Recursion vs Dynamic Programming

Dynamic programming is mostly applied to recursive algorithms. This is not a coincidence, most optimization problems require recursion and dynamic programming is used for optimization.

But not all problems that use recursion can use Dynamic Programming. Unless there is a presence of overlapping subproblems like in the fibonacci sequence problem, a recursion can only reach the solution using a divide and conquer approach.

That is the reason why a recursive algorithm like Merge Sort cannot use Dynamic Programming, because the subproblems are not overlapping in any way.

Greedy Algorithms vs Dynamic Programming

Greedy Algorithms are similar to dynamic programming in the sense that they are both tools for optimization.

However, greedy algorithms look for locally optimum solutions or in other words, a greedy choice, in the hopes of finding a global optimum. Hence greedy algorithms can make a guess that looks optimum at the time but becomes costly down the line and do not guarantee a globally optimum.

Dynamic programming, on the other hand, finds the optimal solution to subproblems and then makes an informed choice to combine the results of those subproblems to find the most optimum solution.

Different Types of Dynamic Programming Algorithms

- Longest Common Subsequence
- Floyd-Warshall Algorithm

Chapter 4

Conclusion

C is called the mother of all programming languages. Because of almost all the programming languages, we can say, born from it. C is mostly used in embedded applications and implementing operating systems. It works as a base for all programming languages. So if you want to start learning to program, you can do it with C programming language. C Programming is a straightforward language.

C++ is a highly efficient and flexible language. All of us must have run programs in this language once. This language was created in 1985. This is needed because we can use it in performance, reliability, and a variety of contexts.

Many augmented systems have been created using this language. Such as Microsoft, PayPal, Adobe, and Oracle. It is an object-oriented programming language. With C ++ you can build careers in desktop applications, especially in performance-intensive tasks. If you are comfortable with C / C++, it gives you a deep understanding of how language works.

Almost all basic-level systems such as file systems, operating systems, etc. are written in C / C++. C++ is widely used by competing programmers due to the fact that it is stable and extremely fast. Even though this language is old, but if you are entering the programming world right now, then you must learn this language.

References

https://drive.google.com/drive/folders/16-VG-B04dbXuUmypfKCkJ70f_yXc8qGw

(accessed at 21st August)

www.Fifthforce.in (accessed at 21st August)

www.geeksforgeeks.com (accessed at 15th September)

www.programiz.com (accessed at 15th September)

www.wikipedia.org (accessed at 16th September)