

Introduction

فيه حاجات كتير عن ال data structure ه تكون عارفها لأنك طبقت عليها قبل كده ولو مش واحد الكورسات اللي قبل دي بالترتيب هتنتب هنا والكورس متقسم لجزئين

The image shows a promotional banner for a Data Structures course. On the left, there's a thumbnail of the course video titled "Data Structures Level 1" featuring a man in a suit. The main background image shows a person's hands typing on a laptop keyboard with a "90% OFF" discount tag hanging above it. A notepad and pen are in the foreground. Text overlays include "ProgrammingAdvices.com" at the top left, "Mummed Abu-Hadhou" at the bottom left, and "مميزات الكورس" (Course Features) at the top right.

ProgrammingAdvices.com

Mummed Abu-Hadhou

90% OFF

Data Structures Level 1

مميزات الكورس

نظرة عامة على مفاهيم بنية البيانات
بالإضافة إلى مناقشة التطبيقات
المختلفة لأنواع بنية البيانات وأنماط
البرمجة باستخدامها. الكورس مبني
على التدرج في المفاهيم لكي ترسخ
في ذهانكم كما عودناكم

كل التوفيق للجميع
محمد أبوهادهود

A grid of 12 course cards from ProgrammingAdvices.com. Each card has a number and a title. The cards are:

- 1. سلسلة أساسيات مهمة لكل مبرمج المستوى الأول
- 2. سلسلة الخوارزميات وحل المشاكل المستوى الأول
- 3. مقدمة للبرمجة بلغة C++ المستوى الأول
- 4. سلسلة حلول متقدمه للخوارزميات المستوى الاول
- 5. سلسلة الخوارزميات وحل المشاكل المستوى الثاني
- 6. مقدمة للبرمجة بلغة C++ المستوى الثاني
- 7. سلسلة الخوارزميات وحل المشاكل المستوى الثالث
- 8. سلسلة حلول متقدمه للخوارزميات وحل المشاكل المستوى الرابع
- 9. سلسلة أساسيات مهمة لكل مبرمج المستوى الثاني
- 10. البرمجة الكائنية Object Oriented OOP مفاهيم واساليب
- 11. البرمجة الكائنية Object Oriented OOP تطبيقات مشروع صغير
- 12. هيكل البيانات Data Structure DS. المستوى الاول

يجب ان تكون انهيت جميع الكورسات بالاخضر

ProgrammingAdvices.com

Mohammed Abu-Hadhou

Telegram Group for This Course

https://t.me/+k_APGHxsXiphYmNk

Before You Start...

Before you start with this course you should review the following from the previous courses:

- Recursion
- Stack vs Heap
- Pointers
- Arrays , 2d Arrays.
- Dynamic Arrays

What is Data Structure? and Why?

هنا عاوزين نعرف يعني ايه data structure ولية بنسخدمه؟

قالك قبل مانجاوب عالسؤال تعالى الأول نعرف يعني ايه برنامج وايه البرنامج اللي انت بتكتبه ده والاکواد دي كلها:

قالك ان البرنامج هو عباره عن حاجتين :-

Algorithms -1 : - ودي الخطوات اللي بتبعها عشان احل مشكله

طيب انت دلوقتي عندك مشكله وعندك طريقة الحل بتاعتها

محتاج أدوات عشان تطبق طريقة الحل ودي بت分成 لنويعين :-

1- أدوات ليها علاقه باللغه :- زي ال if statement وال for statement وهكذا

2- أماكن تمثل فيها الداتا وتخزنها في ال memory بحيث انك تقدر توصلها اثناء تشغيل البرنامج بكفاءة وسرعة عالية وهيا دي ال data structure

Data Structure -2 :- هيا الطريقة اللي بتخزن بيها الداتا عشان توصلها بسرعه وكفاءه

زي مثلا انك عايزة تعمل برنامج يقرأ اسمك ويطبعه عالشاشة

فاللي محتاج تعمله هو انك تعرف متغير وتقرأ الاسم من الشاشه وتخزن فيه وبعدين تطبعه

ده كده اسمه algorithm

طيب ايه ال data structure اللي استخدمنته؟

هوا انك عرفت متغير من النوع string

طيب لو قولتلك انك عايزةك تقرأ 10 درجات وتحسب ال average بتاعهم

ايه ال algorithm اللي هتبقي عشان تحل المشكله دي او تتحقق البرنامج؟
قالك ان ال algorithm الأول انك تعرف 10 متغيرات مفرطين وتقرأ القيم بتاعت كل واحد فيهم
ال قالك عرفت 10 متغيرات
ال الثاني قالك تعرف array وتعمل for loop وتسأل اليوزر عن كل درجه وتخزنهم
وتحسب ال average وتطبعهم على الشاشه
ال array هنا انك عرفت data structure
فيه أنواع عديده من ال data structure هننشر لهم بالتفصيل ومهم اني اعرفهم عشان اتعامل مع الداتا
بشكل افضل

What is Program?

Program = Algorithms + Data Structure.

- You know Algorithms is a step by step procedure to solve a particular problem.
- To Develop a program of an algorithm, we should select an appropriate data structure for it.

Data Structure : is a way of organizing all data items and their relationships to each others inside the program in order to deal with them.

Data Structure : affects the design of both structural and functional aspects of the program.

Data structure : is how u organize, manage and store data for efficiency reasons.

A data structure: is not only used for organizing the data. It is also used for processing, retrieving, and storing data.

There are different basic and advanced types of data structures that are used in almost every program or software system that has been developed. So we must have good knowledge of data structures.

- Data structures are an integral part of computers used for the arrangement of data in memory. They are essential and responsible for organizing, processing, accessing, and storing data efficiently. But this is not all.
- Various types of data structures have their own characteristics, features, applications, advantages, and disadvantages.
- So how do you identify a data structure that is suitable for a particular task? What is meant by the term ‘Data Structure’? How many types of data structures are there and what are they used for?

الواجب

Program = Algorithms

True

False

Program = Algorithms + Data Structure.

True

False

Data Structure : is a way of organizing all data items and their relationships to each others inside the program in order to deal with them.

True

False

Data Structure : does not affects the design of both structural and functional aspects of the program.

True

False

Data Structure : affects the design of both structural and functional aspects of the program.

True

False

Data structure : is how u organize, manage and store data for efficiency reasons.

True

False

A data structure: is not only used for organizing the data. It is also used for processing, retrieving, and storing data.

True

False

Various types of data structures have their own characteristics, features, applications, advantages, and disadvantages.

True

False

Differences between Data Structures and Database

ال data structure هيا وسيلة لتمثيل البيانات داخل البرنامج
ال data base هيا وسيلة لتخزين البيانات خارج البرنامج بشكل دائم زي ال file اللي كنا بنستخدمه
في المشاريع وعباره عن جداول وبينها علاقات
في المشاريع اللي فاتت لما كنا بنخزن ال records في الملف كنا بنفصل بينا ب separator دي كده
انما عشان نعالج البيانات دي عالبرنامح ونستدعيها ونخزنها في متغيرات في ال ram
ونرجع نخزنها تاني ده كده اسمه data structure

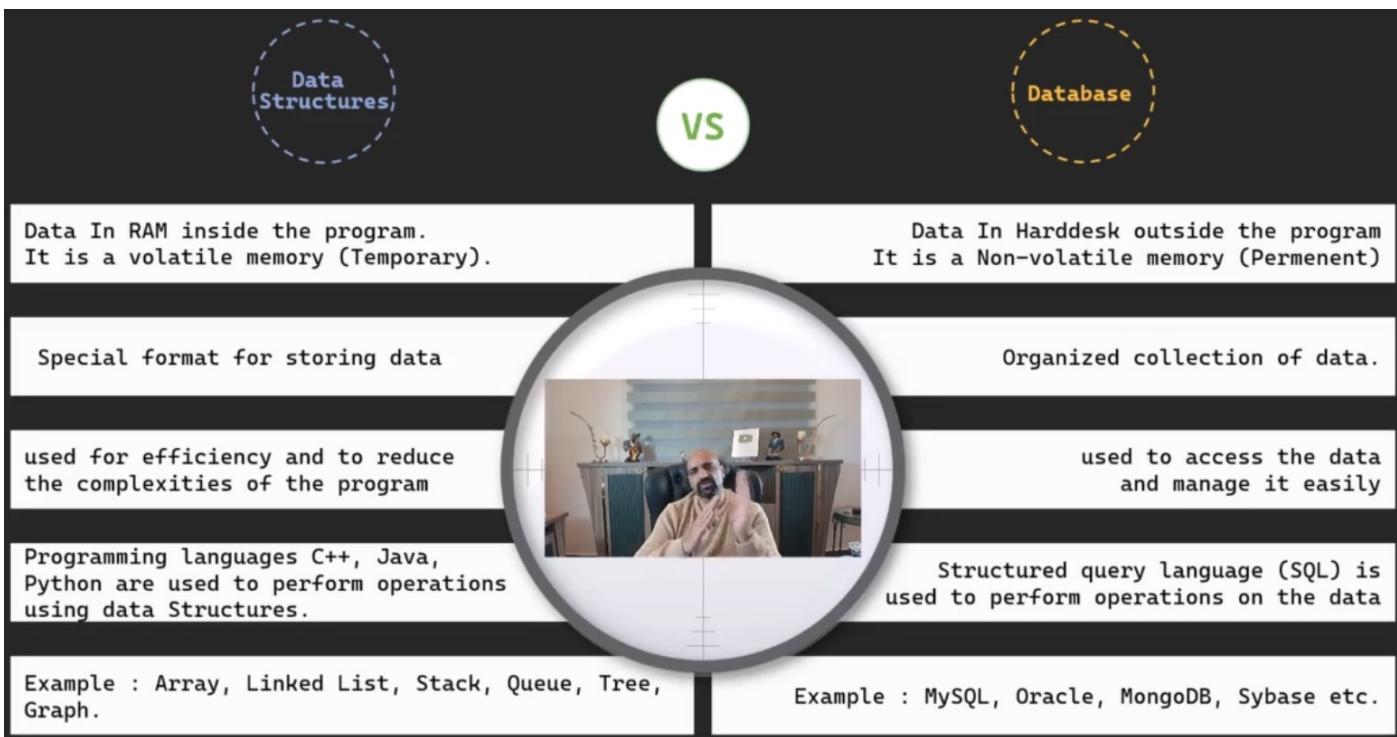
Data Structure vs Database?

Data Structure : In computer Programming, Data structure is a way of organizing and storing data so as to ease the accessing and modification of data.

Some commonly used data structures are Array, Linked List, Stack, Queue, Heap, Binary Tree and Graph.

Database: is a collection of data that is stored in an organized fashion in a table containing rows and columns using a software package known as Database Management System (DBMS).

DBMS is used to modify, define, manipulate and manage data.
Some examples of DBMS are : MySQL, Oracle Database and Microsoft Access.



الواجب

Data Structure is the Same as Database

True

False

Data Structure : In computer Programming, Data structure is a way of organizing and storing data so as to ease the accessing and modification of data.

True

False

Database: is a collection of data that is stored in an organized fashion in a table containing rows and columns using a software package known as Database Management System (DBMS).

True

False

DBMS is used to modify, define, manipulate and manage data. Some examples of DBMS are : MySQL, Oracle Database and Microsoft Access.

True

False

Some commonly used data structures are Array, Linked List, Stack, Queue, Heap, Binary Tree and Graph.

True

False

Data In Database is stored In RAM inside the program.

True

False

Data in data structure is stored In RAM inside the program.

True

False

Structured query language (SQL) is used to perform operations on the data in Database

True

False

Programming languages C++, Java, Python are used to perform operations using data Structures.

True

False

Example of Data Structures : MySQL, Oracle, MongoDB, Sybase etc.

True

False

Example Of DBMS (Database) : MySQL, Oracle, MongoDB, Sybase , SQL Server, Access, etc.

True

False

What are the Classification/Types of Data Structures?

ال data structure ليها تصنيفات :-

Primitive (basic) data structure -1
مع لغة البرمجه زي ال int وال float وال char ومش بتكتب كود معقد عشان
تعامل معاهم

Non-primitive (advanced) data structure -2
ال array زي ال primitive وبتنقسم لنوعين :-

-1 - زى ال linear list يعني الداتا queue وال stack وال linked list كلها جايه على خط واحد زى ال array العناصر مرسوسة جنب بعضها تقدر توصلهم كلهم ب واحد for loop

-2 - زى ال non-linear list tree وال graph زى شجرة العيلة مثلا مش كلهم متوصلين ورا بعضهم متقدرش توصلهم ب واحد for loop

ال non-primitive data structure بيقسموها كمان لنوعين :-

-1 - يعني الداتا نفسها من نفس النوع لو بتعمل array كده الداتا اللي جوه ال int او العناصر اللي جواه كلها نوع واحد وبتعرفه من النوع

-2 - هنا بيقولك ان العناصر مش بتكون من نفس النوع مخطوطتين في data structure واحده اه بس مش كلهم نفس النوع

فيه تقسيمه تالتة لـ :- data structure

-1 - ليها حجم محدد زى ال array العاديه وبيكون الوصول للعناصر فيها اسهل

-2 - ملهاش حجم محدد والوصول للعناصر فيها اصعب زى ال pointer وال vector

طيب ايه هي العمليات اللي يستخدم ال data structure عشان اعملها ؟

قالك العمليات هيا (create - update – search – select – sorting – merging – (destroy or delete

Classification of Data Structure?

Primitive (Basic) Data Structure.

- Integer
- Float
- Char
- Pointer

Non-Primitive (Advance) Data Structure.

- Linear List
- Non-Linear List

Differences:

A primitive data structure:

- Is generally a basic structure that is usually built into the language, such as an integer, a float.

A non-primitive data structure:

- Is built out of primitive data structures linked together in meaningful ways, such as arrays, linked-list, binary search tree, Tree, graph etc.

Primitive Data Structure:

Primitive Data Structure.

- Integer
- Float
- Char
- Pointer

- They are basic structures and directly operated upon the machine instructions.
- Integer, Float, Char, pointers..etc., fall in this category.

Non-Primitive Data Structure:

Linear List

- Array
- Linked List
- Stack
- Queue

Non-Linear List

- Tree
- Graph

- Complex/Sophisticated Data Structure derived from primitive data structure.
- Emphasize on structuring of group of homogeneous (same type) or heterogeneous (different type) data items.
- The design of an efficient data structure must take operations to be performed on data structure.

Non-Primitive Data Structure:



- Complex/Sophisticated Data Structure derived from primitive data structure.
- Emphasize on structuring of group of homogeneous (same type) or heterogeneous (different type) data items.
- The design of an efficient data structure must take operations to be performed on data structure.

Linear vs Non-Linear Data Structures

Linear Data Structure

- Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.
- Examples of linear data structures are array, stack, queue, linked list, etc.

Non-Linear Data Structure

- Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can't traverse all the elements in a single run only.
- Examples of this data structure are Tree, Graph, etc.

Static vs Non-Static Data Structures

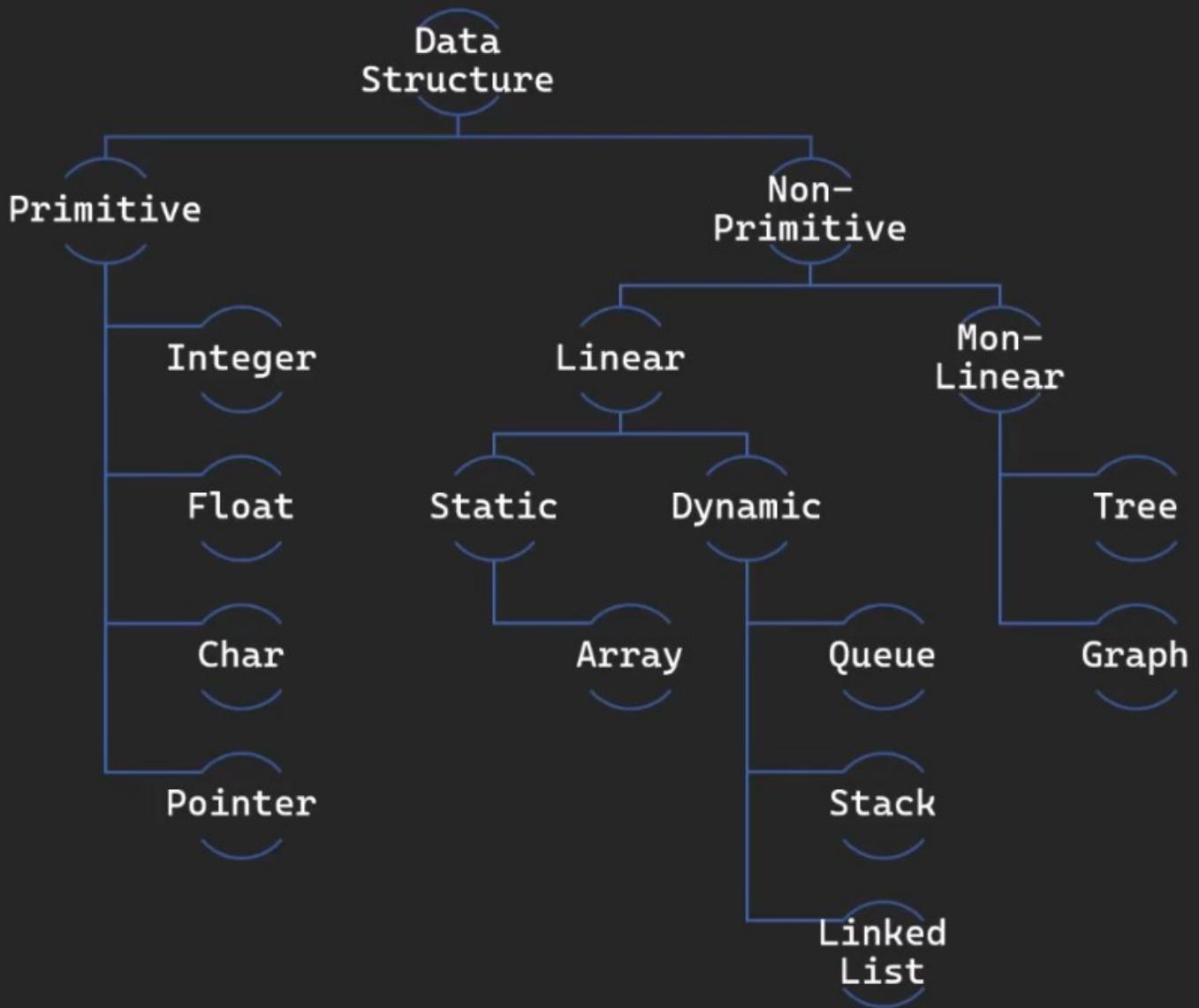
Static data structure

- Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.
- An example of this data structure is an array.

Dynamic data structure

- In dynamic data structure, the size is not fixed. It can be randomly updated during the runtime which may be considered efficient concerning the memory (space) complexity of the code.
- Examples of this data structure are Stack, Queue, etc.

Classifications of Data Structure:



Common Operations on Data Structure:

Operations On Data

- Create
- Update
- Search
- Select
- Sorting
- Merging
- Destroy or Delete

الواجب

Integer is non-Primitive data structure

True

False

Integer is Primitive Data Structure

True

False

Array is Non-Primitive Data Structure

True

False

Non-Primitive is derived from Primitive Data Structure

True

False

Heterogenous means Different Types

True

False

Homogenous means Same Types

True

False

Linear Data Structure means that data is in non-sequence order

True

False

Linear Data Structure means that data is in sequence order

True

False

Tree and Graph are:

Non-Linear, Non-Primitive Data Structure

Linear, Non-Primitive Data Structure

Non-Linear, Primitive Data Structure

Linear, Primitive Data Structure

Array is:

Linear, Primitive Data Structure

Non-Linear, Primitive Data Structure

Non-Linear, Non-Primitive Data Structure

Linear, Non-Primitive Data Structure

Array is:

Homogenous Data Structure

Heterogenous Data Structure

Stack, Queue, Linked Lists are:

Dynamic Data Structure

Static Data Structure

Things Affect Program Speed & Efficiency

فيه عاملين بيأثر و عالسرعه وكفاءة البرنامج بتابعك و هما :-

-: algorithm -1

- ال hardware اللي البرنامج هيشتغل عليه نفسه :-

بعض النظر عن ال algorithm اللي انت مستخدمها فاكانيات الجهاز نفسه اكيد هتفرق لما تشغلي البرنامج فكل ما إمكانيات الجهاز كانت عاليه من رامات وبروسيسور وغيره كل ما سرعة التشغيل ه تكون اعلي

انما لو هنتكلم علي أداء ال algorithm نفسها فيه حاجتين بيأثرروا علي أداء ال algorithm دي و هما :-

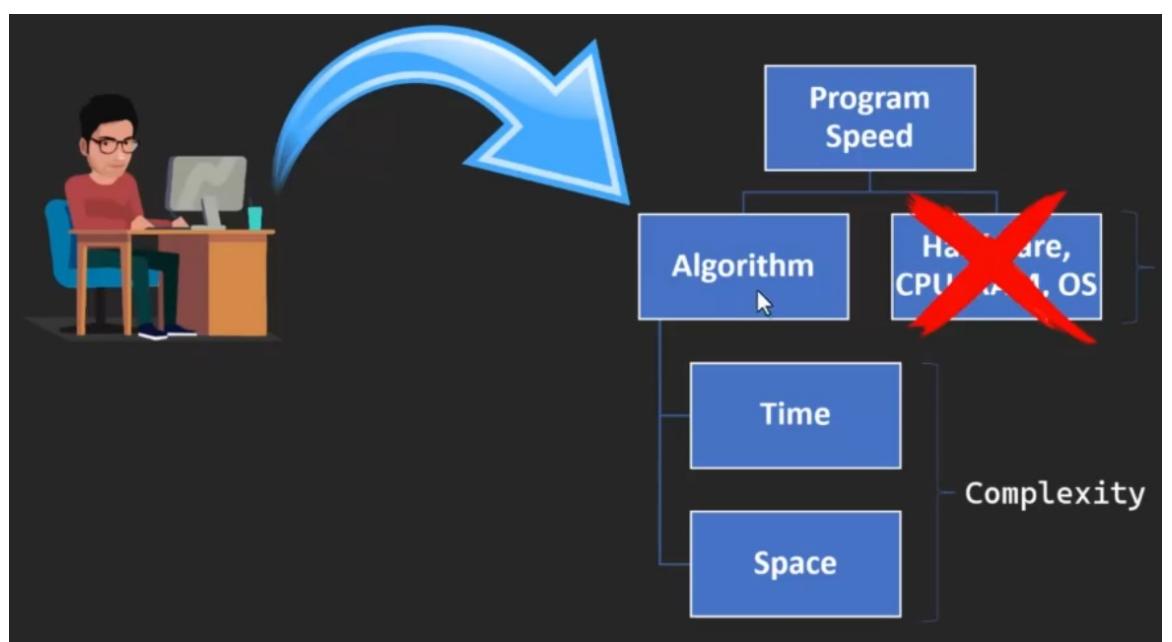
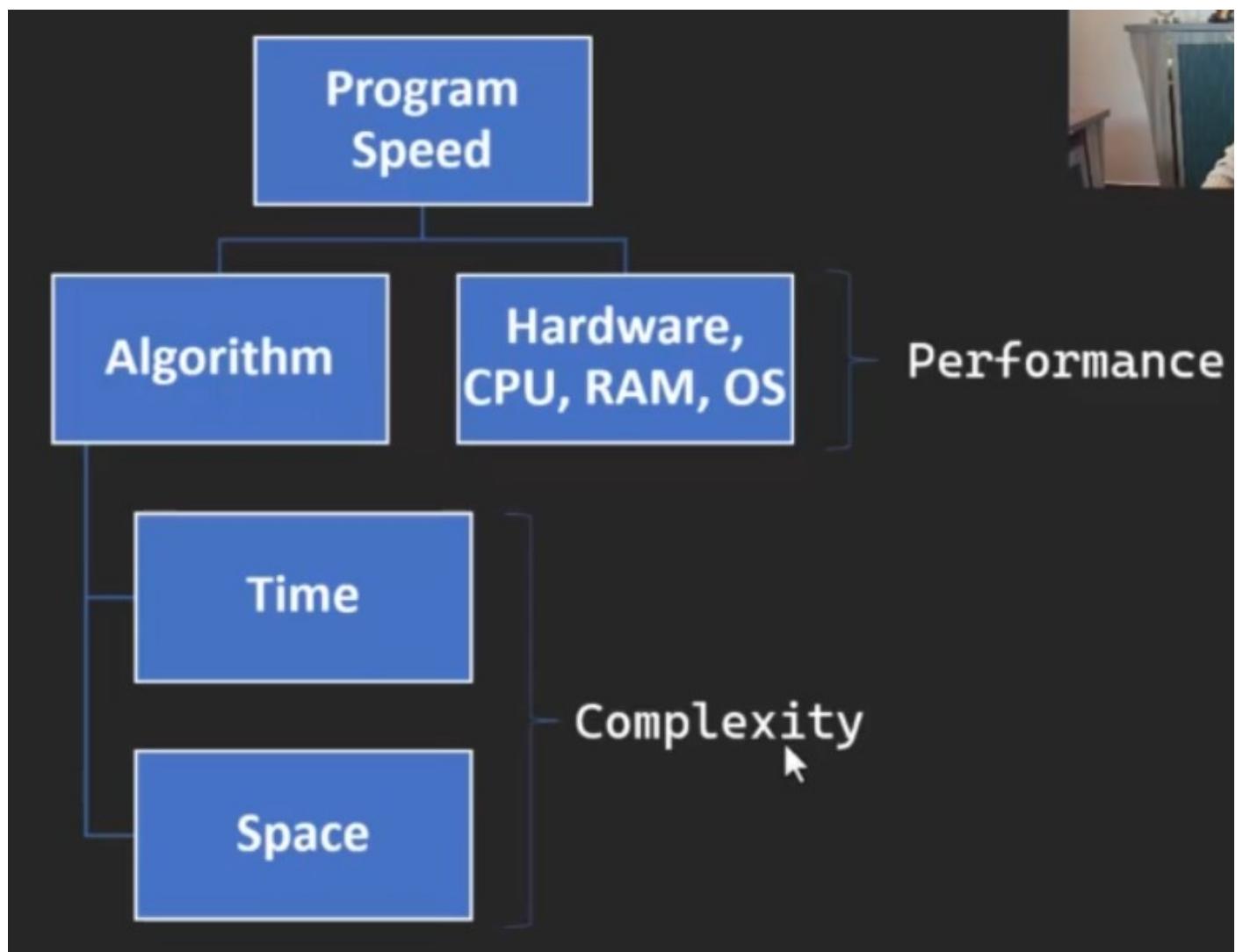
- وده الوقت اللي ال algorithm بيأخذه عشان يشتغل Time -1

- وده المساحه اللي بيأخذها ال algorithm Space -2

فلو هنتكلم عن ال hardware احنا هنا بنتكلم عن performance

انما لو هنتكلم عن ال algorithm فالحنا هنا بنتكلم عن complexity بالنسبياك كمبرمج مالكش دعوه خالص بال hardware كل اللي ليك علاقه بيه هو ال algorithm وده اللي هنركز عليه

Things Affect Program Speed



Things that affects Speed:

Program Speed → Algorithm & (hardware, OS, and CPU).

- Regardless of used algorithm, printing an array[100] items is faster if you run it on a faster computers ☺.
- So as a developer it's not my problem to take care of the hardware.
- But it is My Problem to take care of algorithm ☺
- Therefore we need to focus on Algorithms only on writing code.

الواجب

What Affects Program Speed?

Hardware (CPU, RAM)

Operating System

Algorithm

None of the above

What affects Algorithm?

Time

Space

Hardware

OS

Performance has to do with?

Hardware , OS

Algorithms

Complexity has to do with?

Hardware Complexity

Time Complexity

Space Complexity

Developer should focus on ?

Hardware

Algorithms

Time & Space Complexity - Big O Notation

هنا بيقولوك اننا هنأخذ رؤيه عame عن الموضوع وبشكل المبسط وان ال big o notation ليها علاقه بالرياضيات والمعادلات وكده

دلوقتي لو اوانا لو عملت برنامج بيطبع array من 10 عناصر هيكون اسرع من برنامج بيطبع 1000 عنصر

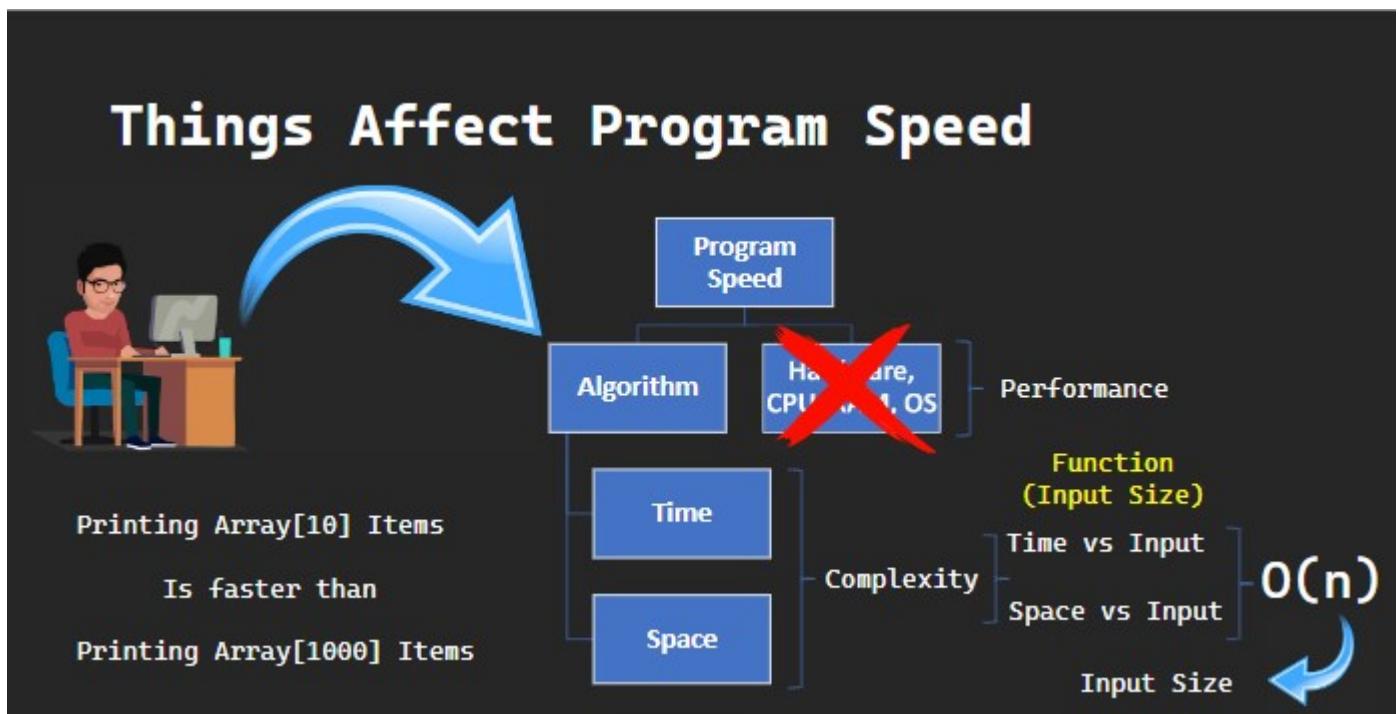
طيب لو البرنامج مثلاً بيطبع 1000 في دققه وانا عايذه يطبعهم في ثانيه
طب ايه مواصفات الجهاز اللي بتشغل عليه البرنامج؟ ماهوا اكيد لو إمكانيات الجهاز عاليه البرنامج
هيستغل اسرع من لو كانت إمكانيات الجهاز ضعيفه من غير متغير ال algorithm ودي حاجه احنا
مش هنركز عليها

فاصبحنا محتاجين آلية حكم بيها على سرعة ال algorithm ونستثنى بيها ال hardware من
الموضوع

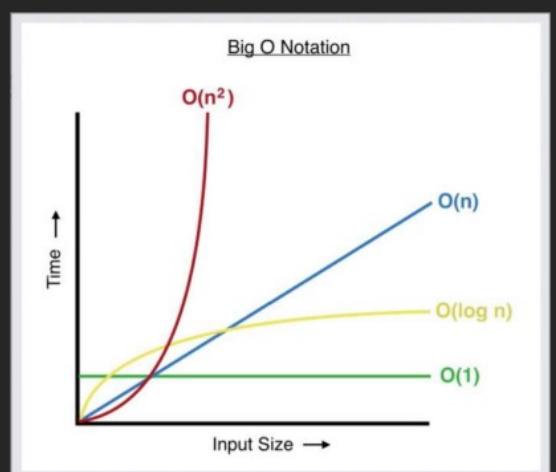
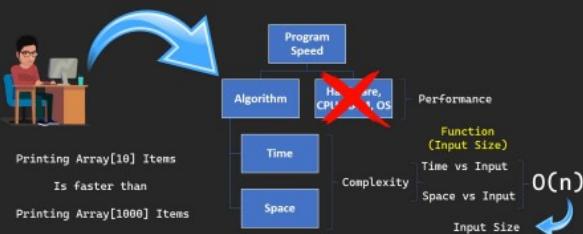
فالحنا محتاجين حاجه تاخد حجم ال input اللي عندى وتقارنه بال algorithm يعني عاوزين ال
algorithm بيتأثر ازاي لو حجم الداتا اللي عندى زادت

لأنه عندك algorithm لو بتطبع student واحد بتكون سريعة ولو زاد العدد بـ student كمان هيزيد شوية وده مقبوله لأنه حجم الداتا زاد

انما بتلاقي algorithm لو العدد زاد بـ student واحد بيكون ابطأ من الـ algorithm الأول فالحنا محتاجين حاجه تربط الـ algorithm as function of input size وعشان كده بيستخدموا الرياضيات فيها عشان هيا اللي هتقدر تبين تأثير بين التغير في أشياء علي الأشياء الأخرى والـ O(n) هو تعبير جبri بيقصد العلاقة بين الـ time والـ input size بيكتب كده () والـ O معناها order of والـ n عباره عن الـ input size



What is Big O?



الصورة اللي فانت بتبييناك انواع الـ O فعندك () وفيه أنواع
ـ O(n^2) - O(n) - O(log n) - O(1)

وال chart بتبيّن لك تأثير الزيادة في ال time على ال input size كل ما في ذلك time يزيد بيزيد بيحصل الاتي بالنسبة لكل نوع من ال O :-

- O(1) :- ملوش تأثير على ال time وهو اسرعهم وماتقدرش تعمل optimization للبرنامج اكتر من كده

- كل ما يزيد ال student الوقت بيزيدي حاجه بسيطه اقل من ال O(log n) -2

- كل ما يزيد ال student بيزيدي ال time شويه وده عادي ممكن ماتقدرش تسرع البرنامج اكتر من كده كل ماتزود student بيزيدي ال time شويه مش مهم حبه قد ايه

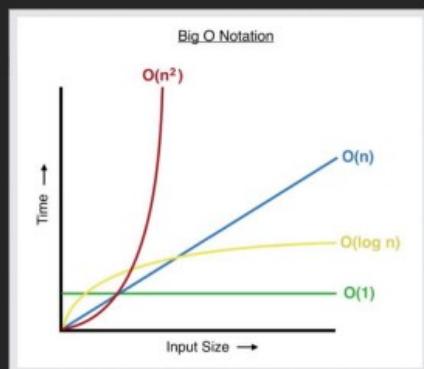
- وده كل ال time بيزيدي ال input بيتضاعف وعاده ال algorithm ده بيكون سئ ويـه حاجـات اكـثـر او اـقـل مـنـه كـمان

ال O big هو تعبر جري مش بيديك time بيديك بس علاقـه وبـيـديـهـاـكـ بال worst case زي مثلاً لو اديتك 1000 طالب في array وقولـتـكـ دورـلـيـ عـالـطـالـبـ اللي اسمـهـ محمدـ هناـ أـسوـاـ اـحـتمـالـ انهـ الطـالـبـ يكونـ اـخـرـ واحدـ فيـ الـ arrayـ واحـسنـ اـحـتمـالـ انهـ يـطـلـعـ اـولـ عـنـصـرـ فـهـوـ هـنـاـ بـيـديـكـ اـسوـاـ اـحـتمـالـ

ال O big مش هـتـقـولـكـ البرـنـامـجـ هـيـاخـدـ وقتـ قدـ اـيهـ وـمـالـوـشـ عـلـاقـهـ بالـ performanceـ بـتـاعـ الـ hardwareـ

What Big O is NOT?

- Big O is not going to give you an exact answer on how long a piece of code will take to run.
- Does not consider the performance of hardware.



Time & Space Complexity

An algorithm's time complexity: specifies how long it will take to execute an algorithm as a function of its input size.

Similarly, an algorithm's space complexity: specifies the total amount of space or memory required to execute an algorithm as a function of the size of the input.

What is Big O Notation?

- The big-O originally stands for "Order Of".
- Efficiency of Algorithm
- Time Factor of Algorithm
- Space Complexity of Algorithm
- Represented by $O(n)$ where n is the number of Inputs.
- Big O, also known as Big O notation, represents an algorithm's worst-case complexity.
- It uses algebraic terms to describe the complexity of an algorithm.
- Big O defines the runtime required to execute an algorithm, by identifying how the performance of your algorithm will change as the input size grows.
- Relationship between Input Size and Performance.
- But it does not tell you the exact time your algorithm's runtime is.
- Big O notation measures the efficiency and performance of your algorithm using time and space complexity.
- Big O allows us to discuss our code algebraically to get a sense of how quickly it might operate under the strain of large data sets.
- with Big O Notation, we can look at our algorithm and see that it will take $O(n)$ time to run. Big O Notation, written as $O(\text{blank})$, show us how many operations our code will run, and how its runtime grows in comparison to other possible solutions.

What is Big O?



الواجب

An algorithm's time complexity: specifies how long it will take to execute an algorithm as a function of its input size.

True

False

An algorithm's space complexity: specifies the total amount of space or memory required to execute an algorithm as a function of the size of the input.

True

False

The big-O originally stands for “Order Of“.

True

False

Big O tells us about the Efficiency of Algorithm.

True

False

Big O, also known as Big O notation, represents an algorithm's worst-case complexity.

True

False

Big O represents an algorithm's best-case complexity.

True

False

Big O uses algebraic terms to describe the complexity of an algorithm.

True

False

Big O describes the relationship between Input Size and Time/Space

True

False

Big O tells you the exact time your algorithm's runtime is.

True

False

Big O does not tell you the exact time your algorithm's runtime is.

True

False

with Big O Notation, we can look at our algorithm and see that it will take $O(n)$ time to run. Big O Notation, written as $O(\text{blank})$, show us how many operations our code will run, and how its runtime grows in comparison to other possible solutions.

True

False

Big O(1) : Constant Time Function

هنا هنركز عال (big o ($O(1)$))
لو طلبت منك function ترجع اخر حرف في ال string
فيه اتنين algorithms بيطلعوا نفس النتيجه

Big O for the following Algorithms:

Algorithm 1

```
char GetLastCharacter(string S1)
{
    return S1[ S1.length() - 1 ];
}
```

Algorithm 2

```
char GetLastCharacter2(string S1)
{
    int n = S1.length() - 1 ;
    for ( int i = 0 ; i <= n ; i++ )
    {
        if ( i == n )
        {
            return S1 [ n ] ;
        }
    }
}
```

لو بصيت عالحلين اللي فوق هتلaci اللي عالشمال اسرع لانه مش بيمشي ب for loop يدوب بيجيب ال length وطرح منه واحد وهو مش مرتبط بعدد الحروف اللي جوه ال string فال o big بتاعها هو constant time

انما الثاني لو ال string فيه 10 حروف بيمشي اكتر من لو كان فيه حرفين لانه مربوط بعدد ال input موضوع ال o big زي ماكنوا بيقولوك برهن ان المعادله دي بتساوي المعادله دي فكنت بتكتب صفحتين كاملين عشان تثبتها بس هنا هيكون بشكل ابسط

ال algorithm اللي عالشمال هيا $O(1)$ والثانية هيا $O(n)$
طيب ازاي نحسب ال $O(1)$ ؟

قالك عشان تحسب ال o big كل اللي بتعمله انك بتحسب عدد الخطوات اللي بيمسيها ال algorithm بافتراض ان كل خطوه بتاخذ نفس ال time يعني ال + زيها زي ال - زي ال return زي الوصول لعنصر من عناصر المصفوفه

طيب في ال algorithm اللي فوق عالشمال بتاخذ كام خطوه؟

- | | |
|--|----|
| انها بتجيب ال string بتاع ال length | -1 |
| بيطرح من ال length الرقم 1 | -2 |
| بيستدعي العنصر اللي ال index بتاعه بيساوي الرقم اللي طلع | -3 |
| ال return | -4 |

اذا هذا ال algorithm بي عمل 4 خطوات بعض النظر عن زيادة حجم ال string اذا فهو constant time function يعني ثابت ويصبح اسمه

عشان نحسب ال o big بتاعتته هنقول

ان ال o big بتاعتته بتاخذ 4 خطوات وكل خطوه بتكون $O(1)$ يعني خطوه واحدة او وقت واحد او كلهم بيأخذوا نفس ال time يعني بتكتب كده

$$\text{Big } O = 4 * O(1) = 4 \ O(1)$$

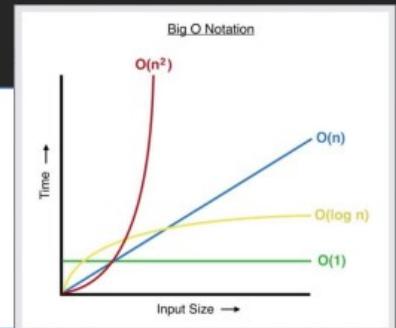
بعدين قالك ان ال o big بتاخذ ال worst case وفي المعادله اللي فوق بنعتبر ان ال $O(1)$ اكبر من ال 4 (بتشيل ال factor) فبتطلع عندي النتيجه هيا $O(1)$

Calculating Algorithm Complexity:

- Count the number of steps during execution.
- Simplification:
 - Each step costs the same time.
 - $+, -, \text{return}$, access array element, ... etc.

```
char GetLastCharacter(string s1)
{
    return s1[ s1.length() - 1 ];
}
```

Big O = $4 * O(1) = 4 O(1)$
Remove Factor
 $O(1)$



- Number of Steps: 4 Steps, and it will be always 4 steps, it has nothing to do with array size ☺
- Constant Time , independent of array size

الواجب

$O(1)$ Means Constant Time Function

True

False

$O(1)$ is affected by Input Size?

True

False

$O(1)$ is not affected by Input Size, so it always takes the same time.

True

False

$O(1)$ is the fastest notation, which means that your algorithm is excellent.

True

False

Big O(n) : Linear Time Function

الدرس اللي فات حسبنا ال O بتاعت ال big algorithm دي وطلعت (1)

Algorithm 1

```
char GetLastCharacter(string S1)
{
    return S1[ S1.length() - 1 ];
}
```

$O(1)$

وال algorithm ده مرتبط بعدد ال input يزيد وقت بيزيدي

Algorithm 2

```
char GetLastCharacter2(string S1)
{
    int n = S1.length() - 1 ;

    for ( int i = 0 ; i <= n ; i++ )
    {
        if ( i == n )
        {
            return S1 [ n ] ;
        }
    }
}
```

ولو حسبنا ال O بتاعتھ هطلع (n) وهو ابطأ من ال (1)

طيب ازاي حسبناه ؟ $O(n)$ ؟

قالك زي ماعملنا في ال algorithm اللي فات هنعمل هنا وهنعد الخطوات بتاعتته هتطلع كده :-

- | | |
|----------------------------|----|
| بتشوف ال length | -1 |
| بتطرح 1 | -2 |
| بتخزن اللي طلع في ال n | -3 |
| بتعرف ال i و بتحط فيها صفر | -4 |

كده معاك اربع خطوات خارج ال loop واي حاجه هندها بعد كده هيا داخل ال loop

- | | |
|--|----|
| بيشوف ال i هل هيا اصغر من او تساوي ال n ولا لا | -1 |
| بيزود ال i بواحد | -2 |
| بيشوف ال i بتساوي ال n ولا لا | -3 |
| بيجيب ال n | -4 |
| بيستدعي العنصر | -5 |
| بيعمل return | -6 |

كده اصبح عندك 4 خطوات خارج ال loop و 6 جوه ال loop

فاصبحت ال $O(6)$ بتساوي ال 6 خطوات بتوع ال loop مضروبين في n لأن لو زادت ال n الخطوات هتزيد

وبعدين تجمع عليهم ال 4 خطوات اللي خارج ال loop

Big O= 6 n + 4

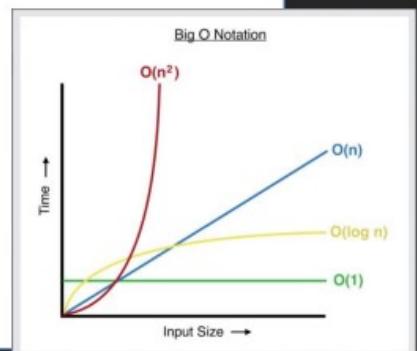
وزي ماقولنا هنشيل بعد كده ال factors هيتبقى ال $O(n)$ وب يكون اسمها linear time function لأن كل مايزيد ال input بيزيد ال time بقيمه ثابتة علاقه خطية

Calculating Linear Complexity $O(n)$:

```
char GetLastCharacter2(string S1)
{
    int n = S1.length() - 1;
    for (int i = 0; i <= n; i++)
    {
        if (i == n)
        {
            return S1[n];
        }
    }
}
```

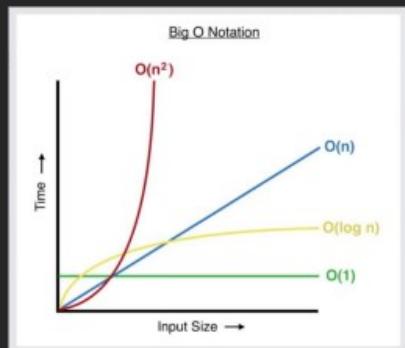
Number of Steps outside loop = 4
Number of Steps inside loop = 6

Big O= 6 n + 4
Remove Factors
 $O(n)$



- Depends on array size, and relation is Linear Time Function.

Comparing the two Algorithms



Faster

```
char GetLastCharacter(string S1)
{
    return S1[S1.length() - 1];
}
```

Big O= $4 * O(1) = 4 O(1)$
Remove Factor
 $O(1)$

Slower

```
char GetLastCharacter2(string S1)
{
    int n = S1.length() - 1;
    for (int i = 0; i <= n; i++)
    {
        if (i == n)
        {
            return S1[n];
        }
    }
}
```

Number of Steps outside loop = 4
Number of Steps inside loop = 6

Big O= $6 n + 4$
Remove Factors
 $O(n)$

الواجب

$O(n)$ is a constant time function.

True

False

$O(n)$ is a linear time function.

True

False

$O(n)$ is faster than $O(1)$

True

False

$O(1)$ is faster than $O(n)$

True

False

$O(n)$ is affected by Input Size Linearly.

True

False

Big O(n^2) : Quadratic Time Function

هناخد ال $O(n^2)$ ودي اسمها quadratic time function وهو ابطأ من ال linear time function

- بصل المثال ده :-

Calculating Quadratic Complexity $O(n^2)$:

```
int MultiplicationSum(short n)
{
    int Sum = 0 ;
    for (int i = 1 ; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            Sum = Sum + (i * j) ;
        }
    }
    return Sum ;
}
```



ده كنا مستخدمنه في طباعة جدول الضرب بس هنا بنجمع ال results بتاعت جدول الضرب

تعالي نعد الخطوات اللي بره ال loop :-

- 1- تعریف ال sum
- 2- تعریف ال i
- 3- استدعاء ال sum
- 4- ال return

تعالی نعد الخطوات بتاعت ال loop الداخليه :-

- 1- اختبار الشرط
- 2- زيادة ال j
- 3- عملية الضرب
- 4- عملية الجمع
- 5- تعیین النتیجه للمتغير

ونحسب ال o بتاعت ال loop الداخليه نلاقيها

$$\begin{aligned} \text{Big } O &= \\ 5 * n &\rightarrow n \end{aligned}$$

تعالی نحسب الخطوات بتاعت ال loop الخارجيه :-

- 1- اختبار الشرط
- 2- زيادة ال i
- 3- تعریف ال j

وتزود عليها ال loop الداخليه كلها اللي هيا $O(n)$

$$\begin{aligned} \text{Big } O &= \\ n * (3 + n) &\rightarrow 3n + n^2 \\ &\rightarrow n^2 \end{aligned}$$

وبعدين تحسب ال big o بتاعت ال algorithm كلها

$$\begin{aligned} \text{Big } O &= \\ 4 + n^2 &\rightarrow n^2 \end{aligned}$$

Calculating Quadratic Complexity $O(n^2)$:

```
int MultiplicationSum(short n)
{
    int Sum = 0;

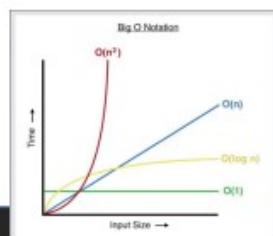
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            Sum = Sum + (i * j);
        }
    }

    return Sum;
}
```

Number of Steps outside loop1 = 4
Number of Steps inside loop2 = 5

$$\text{Big } O = 5 * n \rightarrow n$$

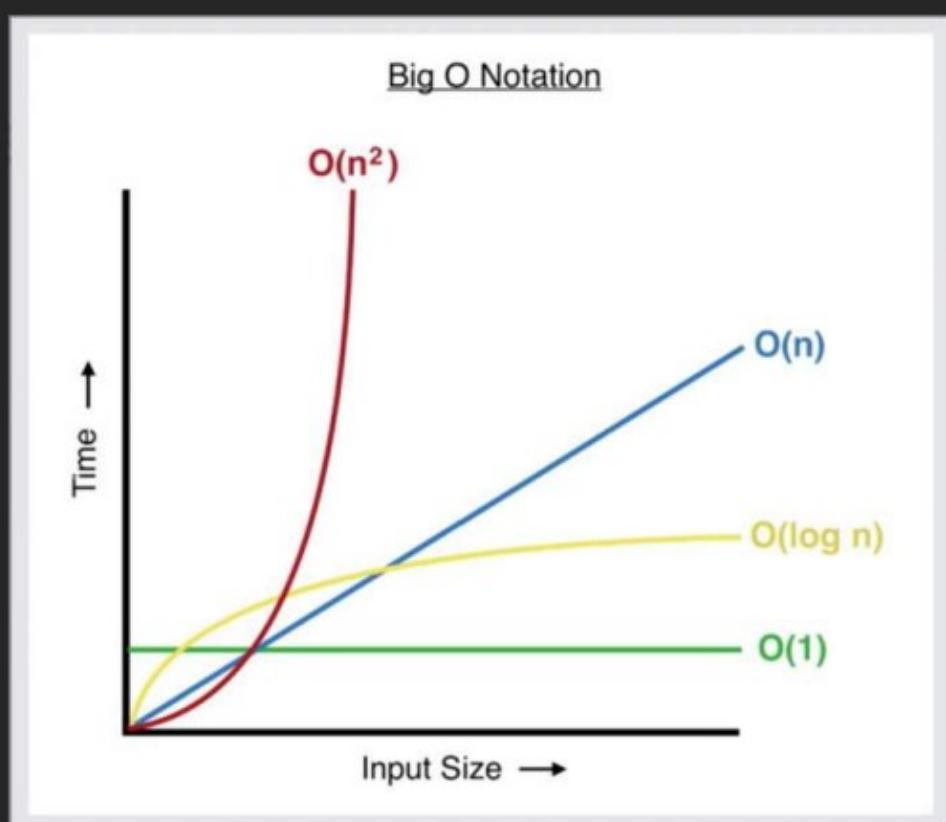
$$\begin{aligned} \text{Big } O &= n * (3 + n) \\ &\rightarrow 3n + n^2 \\ &\rightarrow n^2 \end{aligned}$$



- Depends on n size, and relation is Quadratic Time Function.

وهي ال big o الانواع اللي اخذناها وبتبين من اسرع

Which is faster?



هناقي الأسرع بالترتيب هو $O(1) > O(\log n) > O(n) > O(n^2)$

فلو عندك 3 algorithm ولو افترضنا انه فيه 100 خطوه كل خطوه بتاخذ ثانية مثلا فهناقي ان البرنامج اللي شغال بال $O(1)$ هيأخذ ثانية واحده واللي شغال بال $O(n)$ هيأخذ 100 ثانية واللي شغال بال $O(n^2)$ هيأخذ 10000 ثانية خد مثال :-

فاكر ازاي حلينا ال permissions ؟

لما مثلنا الصلاحيات مثناها برقم واحد واستخدمنا ال bit and binary عشان نشوف الصلاحيه منوحة ولا لا

فكان حلها بسطر واحد افترض انه بيأخذ ثانية

لو ماحليتهاش كده منت هتضطر تشو夫 ال permissions كلها وتعمل اتنين loop واحده تمشي عال users والتانيه تمشي عالصلاحيات هل اليوزر ده عنده الصلاحيه دي ولا لا

كده أصبحت ال $O(n^2)$ بقاعدتك n تربع يعني لو استخدمت الطريقة اللي حلينا فيها هتاخذ ثانية واحده ولو عملت for loops هتاخذ 10 الاف ثانية لو كان عندك 100 يوزر و100 صلاحيه أصبح البرنامج بقاعة الأستاذ اكبر من برنامجك ب 10 الاف مره

ومش دايما بتعرف توصل لـ $O(n^2)$ بس الفكرة انك بحاول توصل للحل الأمثل وفيه حلول أسوأ من ال $O(n^3)$ زي ال

الواجب

Big $O(n^2)$ is Quadratic Time Function

True

False

$O(n)$ is much faster than $O(n^2)$

True

False

$O(1)$ is slower than $O(n^2)$

True

False

$O(1) < O(n) < O(n^2) < O(n^3)$

True

False

if you have $f = 3 + 5n + 6n^2$, then big O is $O(n^2)$

True

False

Big O(Log n) : Logarithmic Time Function

-: $\log n$ عاوزين نعرف يعني ايه

لو جيت قولتلك يعني ايه x^2 هتقولي معناها ان ال x بتضاعف نفسها يعني بتضرب ال x في ال x

ال \log ليها اكتر من base بس لما بتشوفها في الكمبيوتر \log_2 معناها انه بتقسم نفسها

What do we mean by log?

$X^2 \rightarrow$ double itself

$\log_2 \rightarrow$ half itself ☺

بص المثال ده :-

Calculating Logarithmic Complexity $O(\log n)$:

```

void fun1(short n)
{
    short x = n ;
    while ( x > 0 )
    {
        x = x / 2 ;
        cout << x << endl ;
    }
}

```

هنا عندنا خطوه واحده بره ال loop وهيا ال $x=n$

وجوه ال loop عندنا :-

- 1 الشرط
- 2 القسمه
- = اال
- 4 ال x
- 5 ال <>
- 6 ال endl
- 7 ال <>

بس السبع خطوات دول هيكرروا كام مره؟

هيكرروا بس مش بعدد المرات بتاعت ال n لأنك كل مره بتمشي فيها بتقص نصف النتائج

لان لو ال n بـ 25 هيمشي اول خطوه وبعدها ال x ه تكون بـ 12.5 وبعدين بـ 6.25 وبعدين 3.125 وبعدين بـ 1.5 وبعدين هتبقي بصفر لأن المتغير من النوع short

$$\begin{aligned}
 \text{Big O} &= \\
 &7 * \log n \\
 \Rightarrow &\log n
 \end{aligned}$$

مثال عالموضوع ده هو ال binary search و هنا خده بعدين وهو بيقولك انك لو عندك array فيها 1000 رقم مثلاً وعايز تدور علي رقم معين فيه فبتقسم ال array نصين و بترجع تشواف النص اللي عندك تقسمه نصين وهكذا

Calculating Logarithmic Complexity $O(\log n)$:

```
void fun1(short n)
{
```

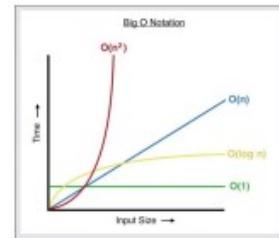
```
    short x = n ;  
  
    while ( x > 0 )  
    {  
        x = x / 2 ;  
        cout << x << endl ;  
    }  
}
```

Number of Steps outside loop = 1

Number of Steps inside loop = 7

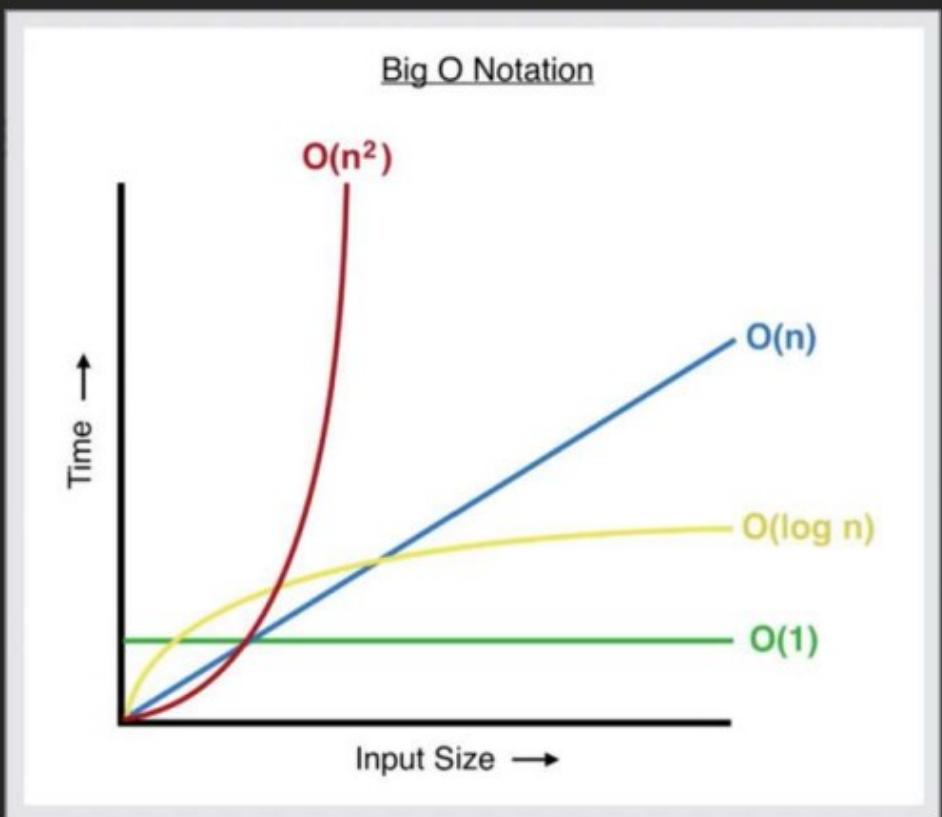
$$\text{Big O} = 7 * \log n \rightarrow \log n$$

$$\text{Big O} = 1 + \log n \rightarrow \log n$$



- Depends on n size, and relation is Logarithmic Time Function.

Which is faster?



Log2 always half itself.

True

False

Log means that the number of iterations in the loop is always less than n

True

False

$O(\log n)$ is much faster than $O(n)$

True

Flase

Important Question :- , Work Smart.

لو انا دلوقتي عندي algorithm معين ال big o بقى عامل هيا $O(n)$ وقامت بقى عامل تانى $O(n)$ بقى عامل نفس الشغل وال big o بقى عامل هوا $O(n)$ برضه

هل الاتنين هىكون ليهم نفس السرعه ؟

بص عالبرنامح ده بيدور على رقم في ال array باتنين algorithms مختلفين بس ليهم نفس ال big o

```
//ProgrammingAdvices.com  
//Mohammed Abu-Hadhoud
```

```
#include <iostream>  
using namespace std;  
  
short FindNumberAlgorhim1(short arr1[10], short Number)  
{  
    int n = 10;  
    short pos = -1;  
  
    for (int i = 0; i <= n; i++)  
    {  
        if (arr1[i] == Number)  
        {  
            pos = i;  
        }  
    }  
}
```

```

    return pos;
}

short FindNumberAlgorhim2(short arr1[10], short Number)
{
    int n = 10;

    for (int i = 0; i <= n; i++)
    {
        if (arr1[i] == Number)
        {
            return i;
        }
    }

    return -1;
}

int main()
{
    short arr1[10] = { 10,20,30,40,50,60,70,80,90,100 };

    cout << FindNumberAlgorhim1(arr1, 100) << "\n";
    cout << FindNumberAlgorhim2(arr1, 100) << "\n";

    system("pause>0");
    return 0;
}

```

هنا هتلاقي ان ال algorithm الثاني اسرع من ال الاول مع انهم نفس ال big o وعشان كده بيقولك انه تساوي ال big لا يعني بالضرورة تساوي ال speed لانه طريقة كتابتك لل algorithm بتفرق في السرعه

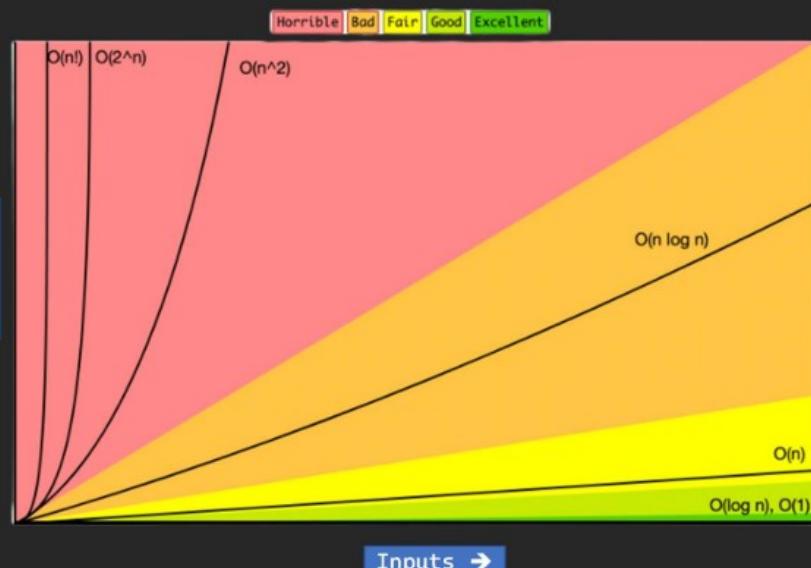
في المثال اللي فوق عندك ال algorithm الاولي بفضل مكمله في ال loop حتى بعد ماتلاقي العنصر انما التاييه اول مابتلacieh بتطلع من ال loop

هنا الاثنين بيتساوى في السرعه ان كنت بتدور على اخر عنصر في ال array لان ال big o بتديك indication مش بتديك الوقت الفعلي

Big O Comparison and Conclusion

دول ال common big o

Big O Cheat Chart



Time ↑

- O($n!$) : Exhaustive Search
- O(2^n) : Exponential
- O(n^3) : Cubic
- O(n^2) : Quadratic
- O($n \log n$) : Log Linear
- O(n) : Linear
- O($\log n$): Logarithmic
- O(1) : Constant

Inputs →

وده جدول بيوريک الزياده في عدد ال input وعلاقته بال time

Growth Rate Comparison:

	n=1	n=2	n=4	n=8	n=16	n=32
1	1	1	1	1	1	1
log n	0	1	2	3	4	5
n	1	2	4	8	16	32
n log n	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
$n!$	1	2	24	40320	20.9T	Don't ask!

Time ↑

- O($n!$) : Exhaustive Search
- O(2^n) : Exponential
- O(n^3) : Cubic
- O(n^2) : Quadratic
- O($n \log n$) : Log Linear
- O(n) : Linear
- O($\log n$): Logarithmic
- O(1) : Constant

Always take the worst Big O

$$F(t) = 10 + O(n) + O(\log n) + O(n^2) + O(n^3)$$

$$F(t) = O(n^3)$$

Binary Data Structure: Real Examples.

الbinary data structure هو نوع من أنواع ال data structure لو جيت سألك عن الرقم 79 فيه ناس كتير هتقولك مجرد رقم لكن بالنسبة لنا هو رقم متخزن فيه داتا اذا هو نوع من أنواع ال data structure واستخدام ال binary في التخزين بيكون افضل من ناحية ال space complexity زي ماشوفنا في موضوع الصلاحيات لو جينا حسبنا ال big o بقاعدته هناقيها (1) لكن لو كنا عملنا array فيها الصلاحيات هناقيها $O(n)$

مثال اخر :-

هو بيقولك انه بيلعب اورج وفيه من الاورج أنواع وهو عنده منهم نوع اسمه kronos النوع ده بـ 6000 دولار وكويس جدا وصوته حلو بس سئ من ناحيه ال user experience فيه حاجه معينه لما بيجي يعملها لازم يدوس على زرار معين 120 مره فقلالك انا هعمل برنامج احل المشكله دي فيه وهو مش بيفهم في ال musical programming وفي خلال 6 ساعات قدر يعمل برنامج يقدر منه خلال يعمل ضغطه واحده تغطيه عن ال 120 ضغطه وعمل البرنامج بحيث انه يدعم كل الاورجات اللي في الدنيا

المهم انه عمل البرنامج وخلاله يدعم أنواع مختلفه من الاورجات بس عشان حد يشتري نوع معين كان بيديله البرنامج كامل بس بيفتحه صلاحيه عالنوع اللي العميل دافعه بس من خلال ال binary فكان بيعمله activation key موجود فيه الصلاحيات بتاعت البرنامج

مش كل الحاجات بتقدر تعملها بال binary بس فيه حاجات مش مستدعية انك تعملها بطرق معقده

You can store data in a number using binary.

What does this number mean?

79

Number 79 Means:

Permission	Has Permission?	Binary Value
Show Client List	✓ Yes	1
Add New Client	✓ Yes	2
Delete Client	✓ Yes	4
Update Client	✓ Yes	8
Find Client	✗ No	16
Transactions	✗ No	32
Manage Users	✓ Yes	64
Show Login Register	✗ No	128

= 79

What is Time & Space Function of it? O(?)

Permission	Has Permission?	Binary Value
Show Client List	✓ Yes	1
Add New Client	✓ Yes	2
Delete Client	✓ Yes	4
Update Client	✓ Yes	8
Find Client	✗ No	16
Transactions	✗ No	32
Manage Users	✓ Yes	64
Show Login Register	✗ No	128

If
(PermissonToCheck & UserPermissions =
PermissionToCheck)

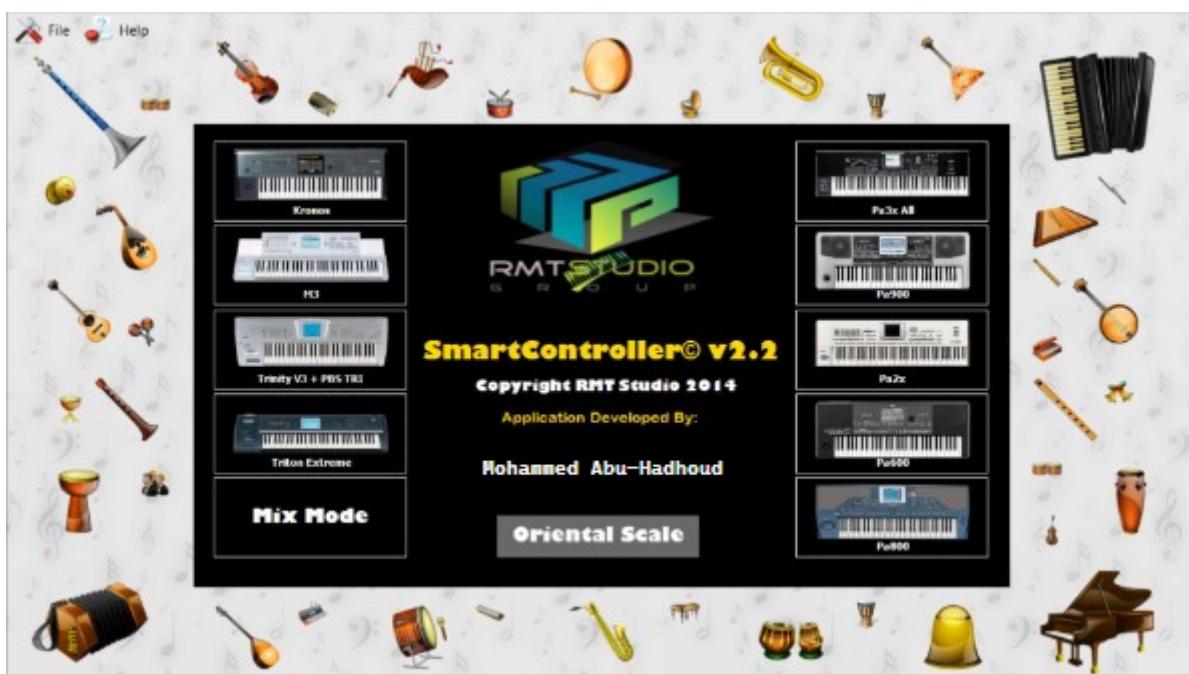
yes

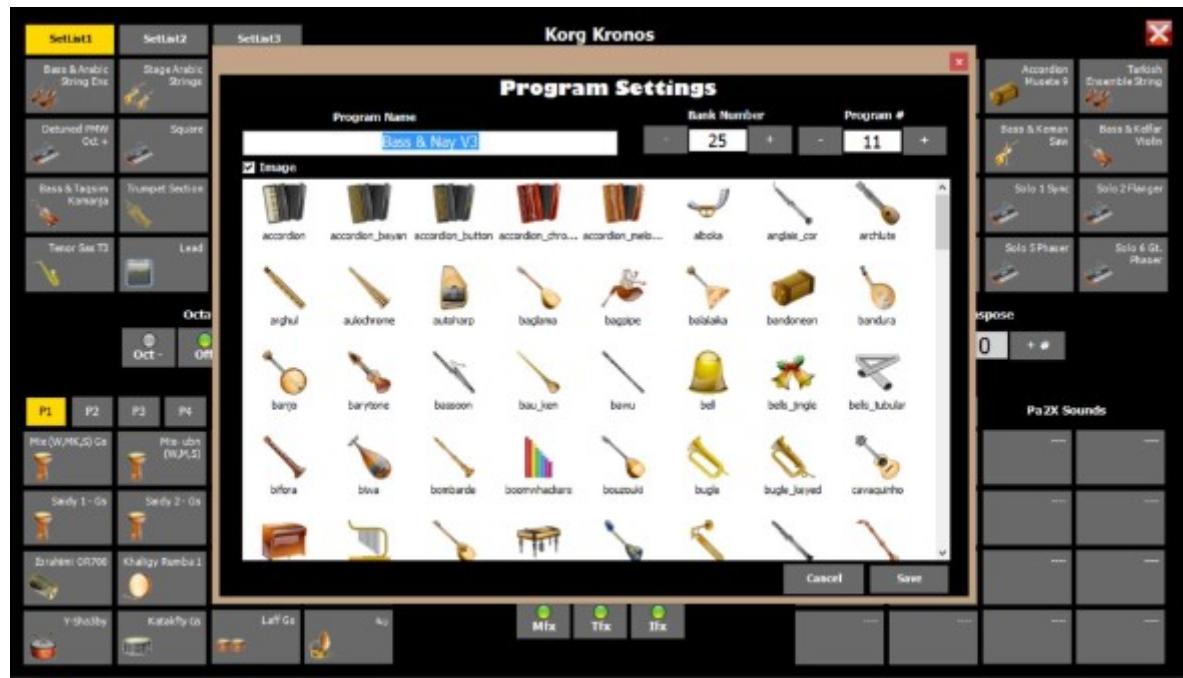
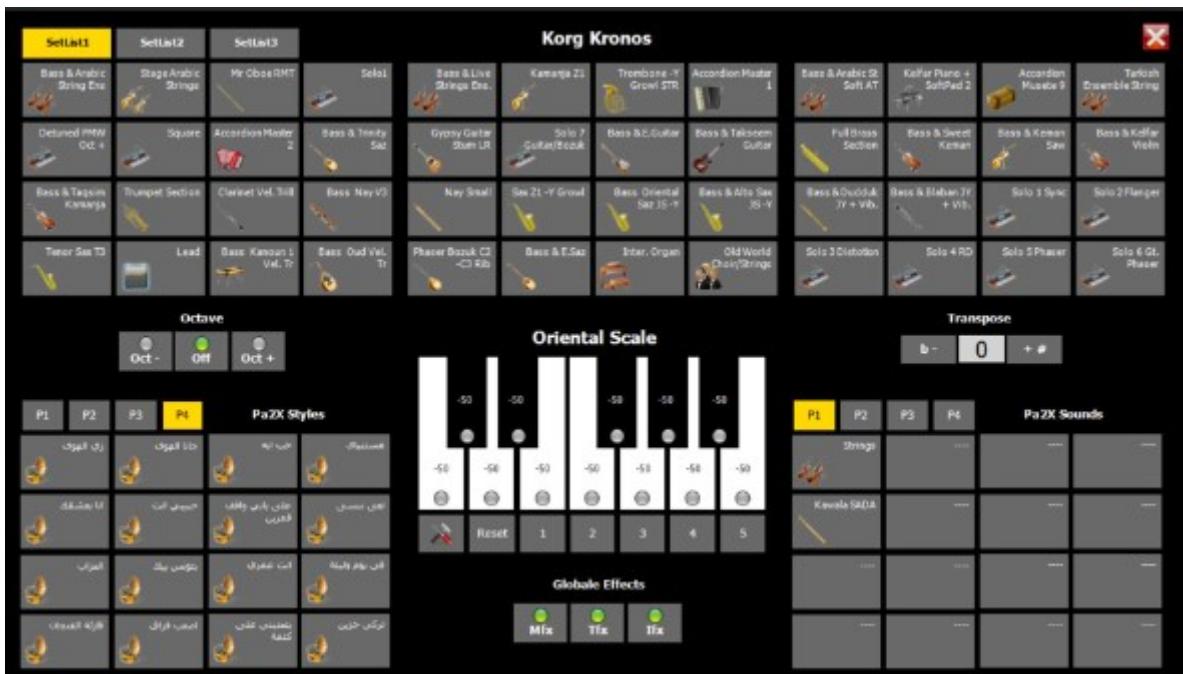
Else

No

O(1)

Another Practical Example ☺





DKHSF-PYHQ8-58A8P-GRHCB-BV2ZS

Array is One of the Most Important Data Structures

الـ **array** هيا عباره عن سلسله من التغيرات اللي ليها نفس النوع وها نوع من أنواع الـ **data homogeneous data structure** وهيا **structures**

الموضوع اتشرح كتير واطبق عليه كتير فمفيش داعي شرحه تاني هنا

هل الـ **array** لها **fixed size** ؟

علي حسب لغة البرمج اللي بتسخدمها في الـ **C** ليها **fixed size** وفي الـ **C++** فيه عندك **arrays**

طب ايه العمليات اللي بتدعها الـ **array** ؟

باقولك انه من مميزاتها انها بتديك **random access** للعناصر بتاعتتها وتقدر تمشي علي ب loop وتعمل **search** و **insert** و حاجات تانية كتير

لو هتعبي عنصر واحد فال time complexity هيا $O(1)$ لكن لو هتعبي كل العناصر يبقى $O(N)$

Array Data Structure:

Array: is a variable that can store multiple values of the same type.

Array: An array is a collection of data items stored at contiguous memory locations.

The idea is to store multiple items of the (homogeneous - same type) together.

Array Data Structure:

```
int x[5] = { 22, 18, 2, 55, 520 };
```



Is Array Always of Fixed Size?

Depends on the language for example: In C language, the array has a fixed size meaning once the size is given to it, it cannot be changed i.e. you can't shrink it nor can you expand it.

In C++ you can have dynamic arrays.

Operations on The Array?

- Arrays allow random access to elements. This makes accessing elements by position faster.
- Hence operation like searching, insertion, and access becomes really efficient.
- Array elements can be accessed using the loops.

Time Complexity on The Array?

- **Insertion:** We try to insert a value to a particular array index position, as the array provides random access it can be done easily using the assignment operator.

```
arr[2] = 10;
```

- Time Complexity:
 - $O(1)$ to insert a single element
 - $O(N)$ to insert all the array elements [where N is the size of the array]

Time Complexity on The Array?

- **Searching in Array:** We try to find a particular value in the array, in order to do that we need to access all the array elements and look for the particular value.

```
// searching for value 55 in the array;
```

```
Loop from i = 0 to 10:  
    check if arr[i] = 55:  
        return true;
```

- Time Complexity:
 - $O(N)$ [where N is the size of the array]

Applications of The Array?

- Array stores data elements of the same data type.
- Arrays are used when the size of the data set is known.
- Used in solving matrix problems.
- Applied as a lookup table in computer.
- Databases records are also implemented by the array.
- Helps in implementing sorting algorithm.
- The different variables of the same type can be saved under one name.
- Arrays can be used for CPU scheduling.
- Used to Implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.

Matrix Data Structure

ال matrix هي عباره عن مجموعه من العناصر محاطه في صفوف واعمده

Matrix Data Structure:

Matrix: A matrix represents a collection of numbers arranged in an order of rows and columns.

Matrix Data Structure:

	Col 1	Col 2	Col 3	Col 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

	Col 1	Col 2	Col 3	Col 4
Row 1	1	2	3	4
Row 2	5	6	7	8
Row 3	9	10	11	12

لو عايز توصل لعنصر واحد فيها فال time complexity بتاعها هو $O(1)$ انما لو عايز تمثي على كل العناصر هيكون $O(n^2)$

Time Complexity on The Matrix?

- **Insertion:** We try to insert a value to a particular array index position, as the array provides random access it can be done easily using the assignment operator.

```
arr[2][1] = 10;
```

- **Time Complexity:**

$O(1)$ to insert a single element

$O(N^2)$ to insert all the matrix elements [where N is the size of the array]

Time Complexity on The Array?

- **Searching in Array:** We try to find a particular value in the matrix, in order to do that we need to access all the matrix elements and look for the particular value.

```
// searching for value 55 in the matrix;
```

```
Loop from i = 0 to 10:  
Loop from j = 0 to 10:  
    check if arr[i][j] = 55:  
        return true;
```

- **Time Complexity:**

$O(N^2)$ [where N is the size of the array]

Important: What is Stack Data Structure?

لو انت عايز تبقي مجرم بتروح عالسوق السوداء عشان تشتري مسدس بالطلقات بتاعته والمسدس بيكون ليه خزنه عشان تحط فيها الطلقات

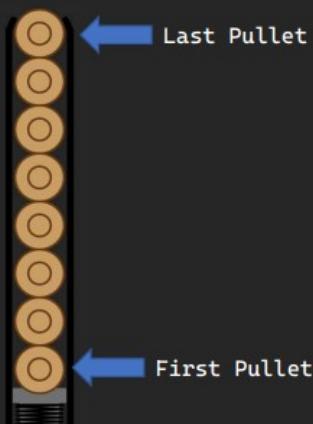
المخزن ده هو ال stack بتاعنا

طيب انت دلوقتي عبئته بالرصاص وعايز نقتل حد فقومت ضربت عليه رصاصه انهي رصاصه اللي هتطلع الأول؟

آخر رصاصه دخلت هيا اول واحده بتطلع والمبدأ ده اسمه **last in first out** معناه **lifo** لو انت محاسب هتعرفه

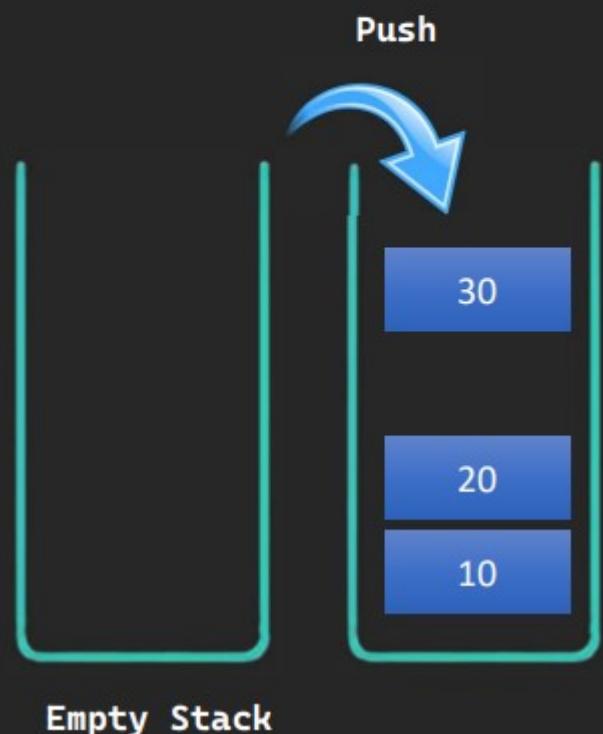
وهو نفسه ال stack في ال data structure

Gun Bullets Stack

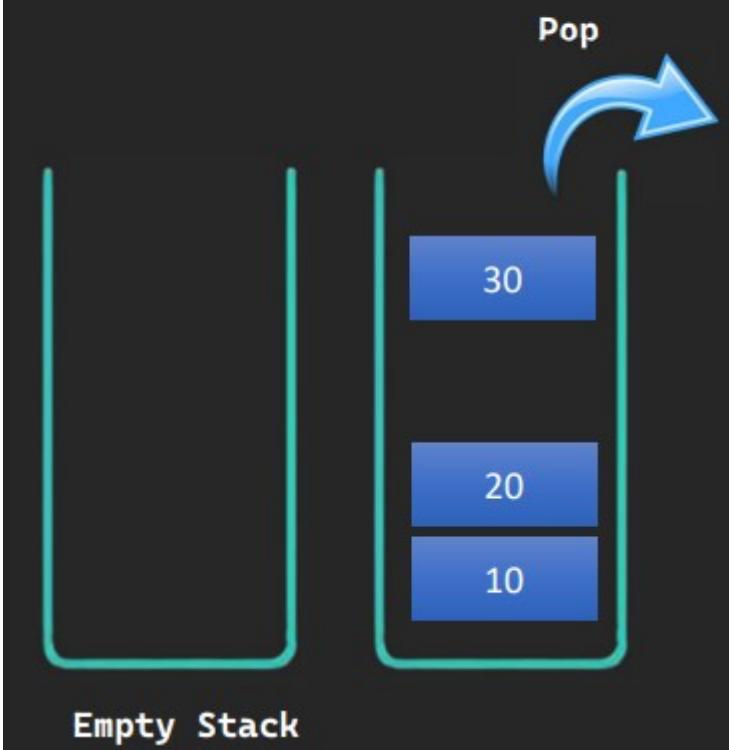


LIFO:
Last In First Out

Stack Push



Stack Pop



ال vector هو data structure مبنيه على stack وال stack data structure مبني على ال array

من استخدام ال stack هو ال call stack بتابع ال stack

STL: Stack

هناخد ال stack من ال stl وهيا اختصار ل standard template library بخليل اسرع في البرمجه وخذنا منها قبل كده ال vector c++

STL : Standard Template Library

Stack

المكتبه دي فيها حاجه اسمها stack ودي حاجه خاصه لينا عشان تستدعيها بتكتب كده

#include <stack>

وبتعرفه زي ال vector بالضبط بس بدل ما تكتب

stack <int> stkNumbers;

```
// ProgrammingAdvices.com
// Mohammed Abu-Hadhoud

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

int main()

{
    // create a stack of ints
    stack <int> stkNumbers;

    // push into stack
    stkNumbers.push(10);
    stkNumbers.push(20);
    stkNumbers.push(30);
    stkNumbers.push(40);
    stkNumbers.push(50);

    // we can access the element by getting
    // the top and popping
    // until the stack is empty
    cout << "count=" << stkNumbers.size() <<
endl;

    cout << "Numbers are:\n";
    while (!stkNumbers.empty()) {
        // print top element
        cout << stkNumbers.top() << "\n";

        // pop top element from stack
        stkNumbers.pop();
    }

    system("pause>0");
    return 0;
}
```

هنا دي المكتبه

وهنا بعرف ال stack

وهذا بضيف عناصر زي ال vector بالضبط

وهذا يستدعي ال size عشان اعرف فيها كام عنصر

وهذا بقوله انه طول ما ال stack مش فاضي اطبعي العنصر اللي في ال top في ال

وبعدين شيله

{

ال o big بتاع ال stack هو (n) لو همشي عليهم كلهم ولو هجيب عنصر معين منهم هيكون (1)

الواجب

To add an Item to a stack you use:

Pop

Push

To remove the top item from stack you use:

Pop

Push

To access the last item in stack you use:

top

bottom

To get the count of the items in stack you use:

count

size()

To check if the stack has items or not you use:

empty()

full()

LIFO means Last In First Out

True

False

Stack Swap

فيه `function` ال اسمها `swap` ودي وظيفتها انك لو عندك `stack1` و `stack2` بتقدر تخل العناصر اللي في `stack1` تنقلها `stack2` والعناصر اللي في `stack2` تنقلها في `stack1`

```
// ProgrammingAdvices.com
// Mohammed Abu-Hadhoud

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

int main()
{
    // stack container declaration
    stack<int> MyStack1;
    stack<int> MyStack2;

    // pushing elements into first stack
    MyStack1.push(10);
    MyStack1.push(20);
    MyStack1.push(30);
    MyStack1.push(40);

    // pushing elements into 2nd stack
    MyStack2.push(50);
    MyStack2.push(60);
    MyStack2.push(70);
    MyStack2.push(80);

    // using swap() function to swap elements of stacks
    MyStack1.swap(MyStack2);

    // printing the first stack
    cout << "MyStack1 = ";
    while (!MyStack1.empty()) {
        cout << MyStack1.top() << " ";
        MyStack1.pop();
    }

    // printing the second stack
    cout << endl << "MyStack2 = ";
    while (!MyStack2.empty()) {
        cout << MyStack2.top() << " ";
        MyStack2.pop();
    }

    system("pause>0");

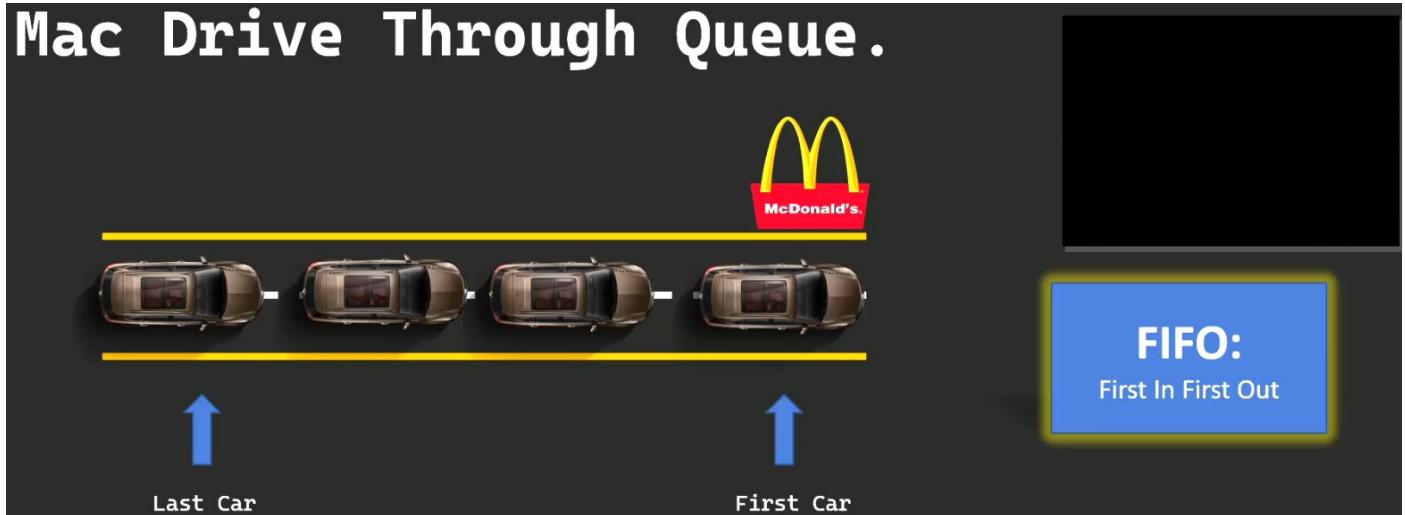
    return 0;
}
```

ال **vector** نوع من أنواع ال **stack** وهو مبني على ال **data structure** تعامله معاملة ال **array** يعني بتقدر تستدعي عنصر معين من ال **vector** لكن ماتقدرش تعمل كده مع ال **stack** وهو من ضمن مكتبه ال **stl**

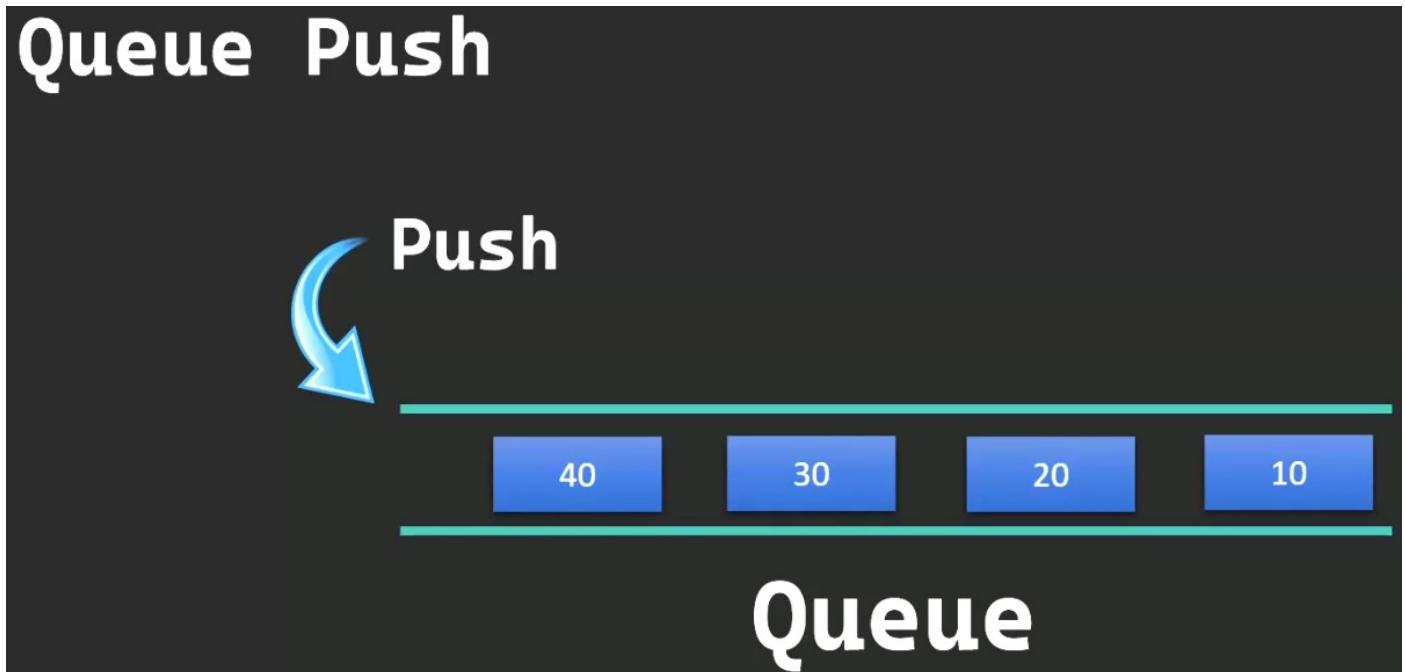
Important: What is Queue Data Structure?

تخيل انك رايح نشتري عيش من فرن بلدي هتلaci فيه طابور من الناس واقفين عشان يشتروا عيش زيak واللي بييجي الأول بيقف الأول ويشتري الأول ويمشي الأول هنا ال **queue** ده شغال بمبدأ ال **FIFO** وهو اختصار ل **first in first out** لو كنت محاسب هتعرفه والطابور ده ممثل في المكتبه **stl** وبرضه عشان تضيف عنصر بتعمله **push** وعشان تشيل عنصر بتعمله **pop**

Mac Drive Through Queue .



Queue Push



Queue Pop



40 30 20 10

Queue

STL: Queue

عشان تستخدم ال `queue` بتستدعي المكتبه

`#include <queue>`

واستخدامه زي ال `vector`

```
// ProgrammingAdvices.com
// Mohammed Abu-Hadhoud

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

int main()
{
    // Queue container declaration
    queue<int> MyQueue;

    // pushing elements into first stack
    MyQueue.push(10);
    MyQueue.push(20);
    MyQueue.push(30);
    MyQueue.push(40);

    cout << "\nCount: " << MyQueue.size();
    cout << "\nFront: " << MyQueue.front();
    cout << "\nBack: " << MyQueue.back() <<
endl;

    cout << "\nMyQueue = ";
    while (!MyQueue.empty())
    {
        cout << MyQueue.front() << " ";
        MyQueue.pop();
    }

    system("pause>0");

    return 0;
}
```

هنا يعرف `queue` بضيف عناصر

طبع الحجم وأول وآخر عنصر

بلف عالعناصر وبحذفها

الواجب

FIFO: means First In First Out.

True

False

If you want to print the first item in Queue, you use:

front()

back()

If you want to print the last item in Queue, you use:

front()

back()

If you want to print the count of items in Queue, you use:

size()

length()

If you want add item in Queue, you use:

push

pop

If you want to remove item from Queue, you use:

push

pop

Queue is using LIFO

True

False

Queue is using FIFO

True

False

Swap Queue

هنا بيقولك انه زي مكان فيه stack فيه برضه swap في ال queue

≡ Swap Queue

Swap method in Queue swaps two queues , same as swap method in stack.

see the code below

Download

Code.txt

```
// ProgrammingAdvices.com
```

```
// Mohammed Abu-Hadhoud
```

```
#include <queue>
#include <iostream>
```

```

using namespace std;

int main()
{
    // queue container declaration
    queue<int> MyQueue1;
    queue<int> MyQueue2;

    // pushing elements into first queue
    MyQueue1.push(10);
    MyQueue1.push(20);
    MyQueue1.push(30);
    MyQueue1.push(40);

    // pushing elements into 2nd queue
    MyQueue2.push(50);
    MyQueue2.push(60);
    MyQueue2.push(70);
    MyQueue2.push(80);

    // using swap() function to swap elements of queues
    MyQueue1.swap(MyQueue2);

    // printing the first queue
    cout << "MyQueue1 = ";
    while (!MyQueue1.empty()) {
        cout << MyQueue1.front() << " ";
        MyQueue1.pop();
    }

    // printing the second queue
    cout << endl << "MyQueue2 = ";
    while (!MyQueue2.empty()) {
        cout << MyQueue2.front() << " ";
        MyQueue2.pop();
    }

    system("pause>0");

    return 0;
}

```

What is Linked List?

ال linked list هو عباره عن data structure تانيه structures بيتبني عليها

ال linked list هيا عباره عن linear data structure زي ال array وهو سلسلة من الداتا بتؤشر على بعضها

يعني بيكون فيها اول عنصر بيؤشر على عنصر اللي بعده

وفي بدايته بيكون حاجه اسمها head وده بيؤشر علي اول block ال node ده بيسموه وال node ده مكون من جزئين اول جزء هو الداتا نفسها و الثاني جزء بيكون مؤشر عال node اللي بعده مكان ال nodes مايفرقش معاك هنا

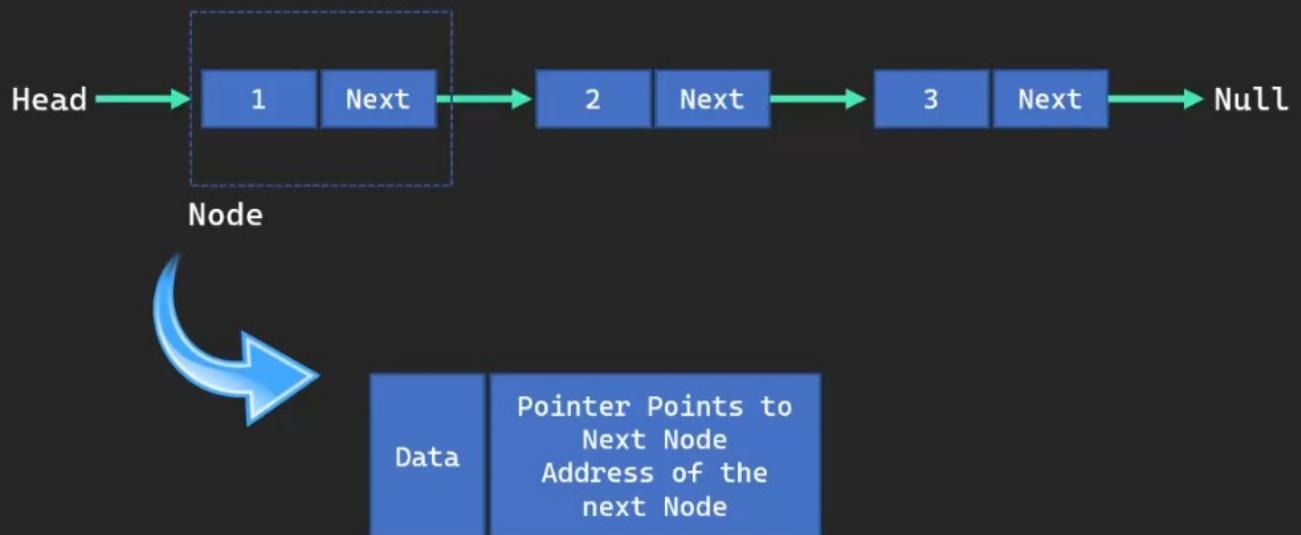
اخر عنصر في ال linked list مش بيؤشر علي حاجه فبيكون المؤشر بتاعه null

ال singly linked list دي اسمها linked list

بيقولك انه زمان في لغة ال c ال dynamic array مكانتش كانوا يستخدموا ال linked list

Linked List (Singly Linked List)

A linked list is a linear data structure that includes a series of connected nodes.



الواجب

A linked list is a linear data structure that includes a series of connected nodes.

 True False

In singly linked list each node consists of two parts: Data and Pointer to the address of next node.

 True False

Linked List is a data structure used to build other data structures like Stack, Queue , and others.

True

False

Linked list allows you to add data dynamically on run time.

True

False

Singly Linked List Implementation

عشان تعمل node بتعمل كلاس وتسميه node (وتقدر تعمله struct) وبتعرف فيه متغير من نوع الداتا اللي انت عايز تخزنها وبتعمل pointer من نفس نوع الكلاس زي كده

```
class Node
{
public:
    int value;
    Node * next;
};
```

وبعدين بتجي عال main بتعرف head pointer بتسميه head بيكون من النوع node اللي هو الكلاس اللي عملته

بعدها بتعرف كمن pointers وبتقول =null

وبعدين بتقول node1= new Node(); من اللي عملتهم

```

Node* head;

Node* Node1 = NULL;
Node* Node2 = NULL;
Node* Node3 = NULL;

// allocate 3 nodes in the heap
Node1 = new Node();
Node2 = new Node();
Node3 = new Node();

```

كده عرفنا ال node محتاجين نعمل حاجتين اول حاجة هيا انك تعبيها بالاداتا بتاعتك ودي كنا عاملينلها المتغير اللي اسمه value

```

// Assign value values
Node1->value = 1;
Node2->value = 2;
Node3->value = 3;

```

وتاني حاجة اننا نربط كل node بالثانية ودي وظيفة ال pointer اللي عرفناه جوه الكلاس اللي اسمه null لحد ما نيجي على اخر node ونخلي ال next بتاعه ب next

```

// Connect nodes
Node1->next = Node2;
Node2->next = Node3;
Node3->next = NULL;

```

طيب دلوقتي عاوزين نطبع القيم اللي في ال linked list اللي عملناها فنقوم مستخدمين ال while loop بس قبلها لازم نربط ال head بأول node

```

// print the linked list value
head = Node1;
while (head != NULL) {
    cout << head->value;
    head = head->next;
}

```

في الكود اللي فات هنا خليت ال head يتحرك ويلف بين ال nodes فاول مره خليته يحدد اول node وبعدين قولته لو ال head مش فاضي هطبعه وبعدين تخلی ال head نفسه ينقل علي ال head اللي بعده ودي وظيفة ال head في الموضوع انه بيلف عالعناصر مش اكتر

```

#include <iostream>
using namespace std;

// Creating a node
class Node
{
public:
    int value;
    Node* next;
};

int main()
{
    Node* head;

    Node* Node1 = NULL;
    Node* Node2 = NULL;
    Node* Node3 = NULL;

    // allocate 3 nodes in the heap
    Node1 = new Node();
    Node2 = new Node();
    Node3 = new Node();

    // Assign value values
    Node1->value = 1;
    Node2->value = 2;
    Node3->value = 3;

    // Connect nodes
    Node1->next = Node2;
    Node2->next = Node3;
    Node3->next = NULL;

    // print the linked list value
    head = Node1;

    while (head != NULL) {
        cout << head->value << endl;
        head = head->next;
    }

    system("pause>0");
    return 0;
}

```

Operations - Insert At Beginning

هنا عاوزين نضيف node في بداية ال linked list قبل باقي العناصر
بص كده عالكود ده

```

//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

#include <iostream>
using namespace std;

// Create a node
class Node
{
public:
    int value;

```

عملنا كلاس ال node اللي بيكون من
متغير بخزن فيه الداتا ومؤشر من نفس نوع
الكلاس

```

    Node* next;
};

void InsertAtBeginning(Node*& head, int value)
{
    // Allocate memory to a node
    Node* new_node = new Node();

    // insert the data
    new_node->value = value;
    new_node->next = head;

    // Move head to new node
    head = new_node;
}

// Print the linked list
void PrintList(Node* head)
{
    while (head != NULL) {
        cout << head->value << " ";
        head = head->next;
    }
}

int main()
{
    Node* head = NULL;

    InsertAtBeginning(head, 1);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 5);

    PrintList(head);

    system("pause>0");
}

```

وهنا ال function اللي بضيف من خلالها العناصر في بداية ال linked list

وهنا بطبع العناصر عن طريق تحريك ال head

وهنا تعريف ال head

وهنا بضيف داتا في البدايه

طيب ال insert اشتغلت ازاي ؟

اول حاجه انا اخذت ال head هنا byref عشان مياخدش نسخه منه ويشتغل في حته بره عن ال main

void InsertAtBeginning(Node*& head, int value)

تاني حاجه عاززين نفهمها انه مافيش حاجه اسمها linked list زي ال array كده او ال vector تيجي تعرفها هوب عملنا list لا ده مش موجود وان الموضوع كله اتنا عندنا كلاس مافيهوش غير داتا و pointer

الداتا وظيفتها تخزين المعلومات وال pointer وظيفته انه يؤشر على object من نفس نوع الكلاس يعني الحوار كله بيتكلم عن اتنا عندنا مجموعه من ال objects متنطوريين بعيد عن بعض عاززين نربطهم ببعض وبترتيب معين مش اكتر

خلينا كمان عارفين انه وظيفته ال head هنا هوا انه بيشتغل زي الماوس بيتنقل بين ال nodes

طيب اول ما استدعينا ال function ايه اللي حصل ؟
 اول حاجه عملها هيا انه عمل pointer من النوع node اللي هو الكلاس بتاعنا و عمل object

```
Node* new_node = new Node();
```

قام جاي عامل تعبيين للمتغيرات اللي جوه ال value قاله ان ال object بتتساوي الرقم اللي جايلك اللي هو الرقم 1

وقاله ان ال next بتتساوي ال address اللي موجود في ال head اللي هو دلوقتي ب null
 وبعدين قال لـ head يخزن ال object اللي عملناه

كده في اول مره اصبح عندنا head واقف عند اول object وواخد عنوانه (واللي مفيش غيره دلوقتي)
 وال next ده ال object بتاعه ب null

نجي لثاني مره هي عمل فيها node ويخرن فيه الرقم 2
 هيدخل عال function ويعرف object جديد ويديله ال value اللي بتتساوي الرقم 2
 وييجي عال next ويقوله يأخذ العنوان اللي موجود في ال head (ال head دلوقتي واحد عنوان
 اول object) وبعدها يخلي ال head يقف عند ال object الجديد اللي فيه الرقم 2

كده في ثانى مره عندنا ال 2 متخزن فيه عنوان ال object1 وال head واقف عند ال object2

في تالت مره هي عمل object جديد ويديله ال value ب 3 وييجي عال next ويقوله يأخذ العنوان اللي
 في ال head (ال head دلوقتي واقف عند ال object2) وبعدها يخلي ال head يقف عند ال
 object الجديد اللي فيه الرقم 3

كده في تالت مره عندنا 3 object متخزن فيه عنوان ال 2 object وال 2 object متخزن فيه
 عنوان ال 1 object وال head واقف عند ال 3 object اللي هو ترتيبه الأول

Operations - Find

عاوزين نبحث عن node معين ونرجعه

```
Node* Find(Node* head, int Value)
{
    while (head != NULL) {
        if (head->value == Value)
            return head;
    }
}
```

هنا ال function بتاخذ pointer من النوع node و بتاخذ ال value اللي انت عايز تبحث عنها

طول مال head لا يساوي null
 شوف هل ال value اللي جوه ال head هل

بتساوي ال value اللي انا بدور عليها ولاً ولو
بتساويها رجلي ال head

head = head->next;	لو مش بتساويها حرك ال head عال object اللي ال next بি�شاور عليه
return NULL;	بره ال loop بيرجع null عشان لو مش موجود

وده الكود كله

```
//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

#include <iostream>
using namespace std;

// Create a node
class Node
{
public:
    int value;
    Node* next;
};

void InsertAtBeginning(Node*& head, int value)
{
    // Allocate memory to a node
    Node* new_node = new Node();

    // insert the data
    new_node->value = value;
    new_node->next = head;

    // Move head to new node
    head = new_node;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {
        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

// Print the linked list
void PrintList(Node* head)
{
    while (head != NULL) {
        cout << head->value << " ";
        head = head->next;
    }
}
```

```

int main()
{
    Node* head = NULL;
    InsertAtBeginning(head, 1);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 5);

    PrintList(head);

    Node* N1 = Find(head, 2);

    if (N1 != NULL)
        cout << "\nNode Found :-)\n";
    else
        cout << "\nNode Is not found :-(\n";

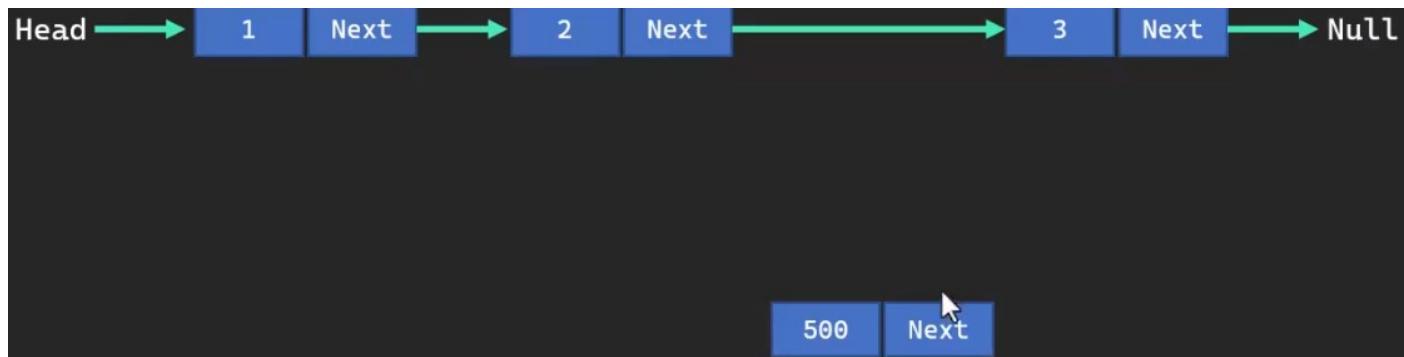
    system("pause>0");
}

```

Operations - Insert After

عاوزين نعمل insert after يعني نضيف عناصر بعد عنصر معين

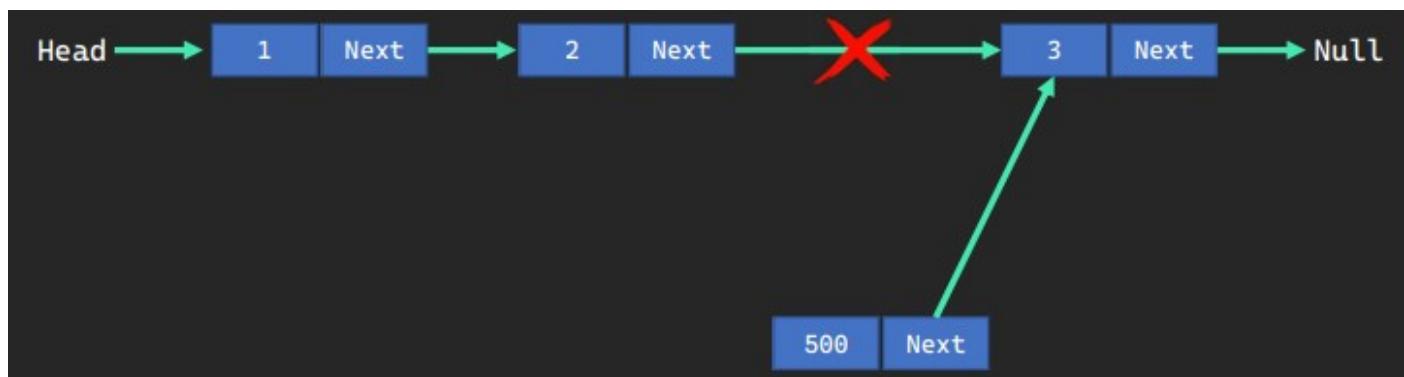
دلوقي انا عندي ال linked list فيها عناصر 1 و 2 و 3 و عايز بعد الرقم 2 احط node جديد

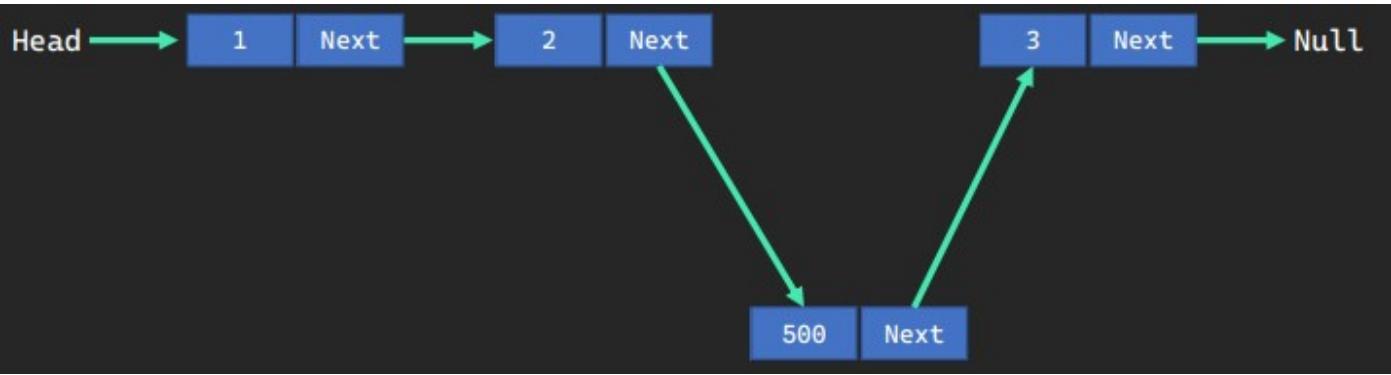


اول حاجه بعملها هيا اني ادور عال node رقم 2 واخلي ال head يقف عليها

وبما ان ال next ال node 2 بتاعها بيشاور علي ال node 3 فالانا هخلي ال next بتاع ال node الجديد هيا كمان تشاور علي ال node رقم 3 هنا اصبح عندي اتنين nodes بيشاوروا على نفس العنوان

هقوم مخلي ال next بتاع node2 يشاور عال node الجديد بدل مايشاور علي رقم 3 لا انا هخلية
يشاور علي ال node الجديد





```

//ProgrammingAdvices.com
//Mohammed Abu-Hadoud

#include <iostream>
using namespace std;

// Create a node
class Node
{
public:
    int value;
    Node* next;
};

void InsertAtBeginning(Node*& head, int value)
{
    // Allocate memory to a node
    Node* new_node = new Node();

    // insert the data
    new_node->value = value;
    new_node->next = head;

    // Move head to new node
    head = new_node;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {
        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

// Insert a node after a node
void InsertAfter(Node* prev_node, int Value) {

    if (prev_node == NULL) {
        cout << "the given previous node cannot be NULL";
        return;
    }

    Node* new_node = new Node();
    new_node->value = Value;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
}
  
```

```

}

// Print the linked list
void PrintList(Node* head)
{
    cout << "\n";
    while (head != NULL) {
        cout << head->value << " ";
        head = head->next;
    }
}

int main()
{
    Node* head = NULL;
    InsertAtBeginning(head, 1);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 5);

    PrintList(head);
    Node* N1 = NULL;

    N1 = Find(head, 2);

    InsertAfter(N1, 500);

    PrintList(head);

/* N1 = Find(head, 500);
   InsertAfter(N1, 600);
   PrintList(head); */

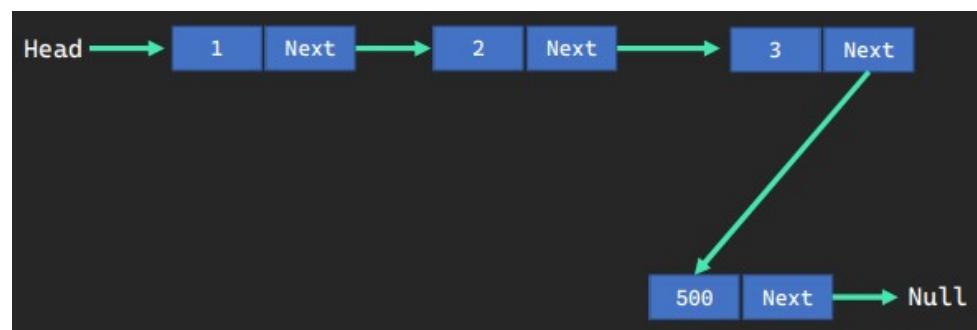
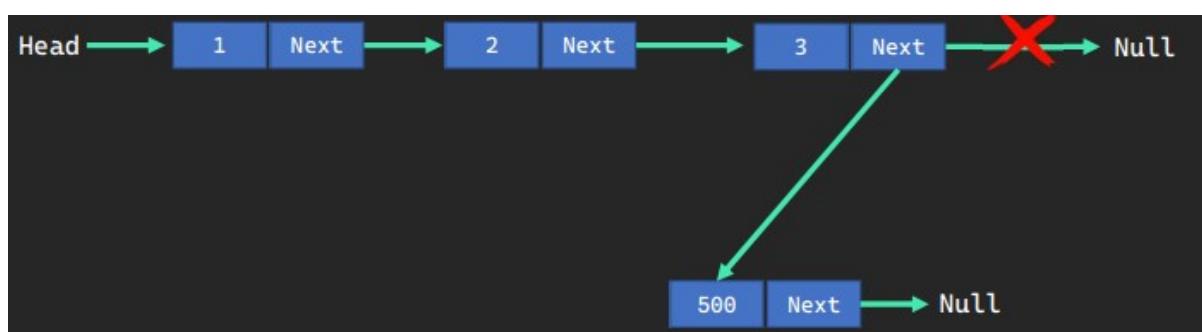
    system("pause>0");
}

```

Operations - Insert At End

عاوزين نضيف node في اخر السلسله

كل الموضوع اني هدور علي اخر node next اللي بتاعها ب null واخليه يؤشر عال الجديده



```

//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

#include <iostream>
using namespace std;

// Create a node
class Node
{
public:
    int value;
    Node* next;
};

void InsertAtBeginning(Node*& head, int value)
{
    // Allocate memory to a node
    Node* new_node = new Node();

    // insert the data
    new_node->value = value;
    new_node->next = head;

    // Move head to new node
    head = new_node;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {

        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

// Insert a node after a node
void InsertAfter(Node* prev_node, int Value) {

    if (prev_node == NULL) {
        cout << "the given previous node cannot be NULL";
        return;
    }

    Node* new_node = new Node();
    new_node->value = Value;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
}

// Insert at the end
void InsertAtEnd(Node*& head, int Value) {

    Node* new_node = new Node();

    new_node->value = Value;
    new_node->next = NULL;

    if (head == NULL) {
        head = new_node;
    }
}

```

```

        return;
    }

Node* LastNode = head;
while (LastNode->next != NULL)
{
    LastNode = LastNode->next;
}

LastNode->next = new_node;
return;
}

// Print the linked list
void PrintList(Node* head)

{
    cout << "\n";
    while (head != NULL) {
        cout << head->value << " ";
        head = head->next;
    }
}

int main()
{
    Node* head = NULL;

InsertAtEnd(head, 1);
InsertAtEnd(head, 2);
InsertAtEnd(head, 3);
InsertAtBeginning(head, 0);

PrintList(head);

system("pause>0");
}

```

Operations - Delete Node

عاوزين نحذف node

عشان نحذف ال node هنحتاج اتنين pointer واحد اسمه current وده اللي هيشارو عليه ال head وانا بعمل البحث وواحد تاني اسمه previous وده هخزن فيه ال head قبل ماينتقل لل next وبعدين هخلی ال next بتاع ال current previous يساوي ال next وبعدين احذف ال current



```

//Mohammed Abu-Hadhoud

#include <iostream>
using namespace std;

// Create a node
class Node
{
public:
    int value;
    Node* next;
};

void InsertAtBeginning(Node*& head, int value)
{
    // Allocate memory to a node
    Node* new_node = new Node();

    // insert the data
    new_node->value = value;
    new_node->next = head;

    // Move head to new node
    head = new_node;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {
        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

// Insert a node after a node
void InsertAfter(Node* Prev_node, int Value) {
    if (Prev_node == NULL) {
        cout << "the given Previous node cannot be NULL";
        return;
    }

    Node* new_node = new Node();
    new_node->value = Value;
    new_node->next = Prev_node->next;
    Prev_node->next = new_node;
}

// Insert at the end
void InsertAtEnd(Node*& head, int Value) {
    Node* new_node = new Node();

    new_node->value = Value;
    new_node->next = NULL;

    if (head == NULL) {
        head = new_node;
        return;
    }
}

```

```

}

Node* LastNode = head;
while (LastNode->next != NULL)
{
    LastNode = LastNode->next;
}

LastNode->next = new_node;
return;
}

// Delete a node
void DeleteNode(Node*& head, int Value) {

Node* Current = head, * Prev = head;

if (head == NULL)
{
    return;
}

if (Current->value == Value) {
    head = Current->next;
    delete Current; //free from memory
    return;
}

// Find the key to be deleted
while (Current != NULL && Current->value != Value) {
    Prev = Current;
    Current = Current->next;
}

// If the value is not present
if (Current == NULL) return;

// Remove the node
Prev->next = Current->next;
delete Current; //free from memory
}

// Print the linked list
void PrintList(Node* head)

{
    cout << "\n";
    while (head != NULL) {
        cout << head->value << " ";
        head = head->next;
    }
}

int main()
{
    Node* head = NULL;

    InsertAtEnd(head, 1);
    InsertAtEnd(head, 2);
    InsertAtEnd(head, 3);
    InsertAtEnd(head, 4);
    InsertAtEnd(head, 5);
    InsertAtEnd(head, 6);
    PrintList(head);
}

```

```

DeleteNode(head, 4);
PrintList(head);

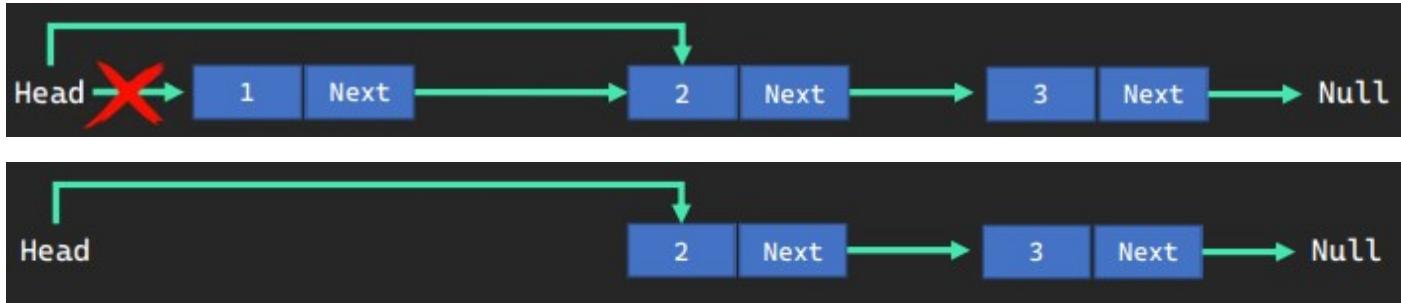
system("pause>0");
}

```

Operations - Delete First Node

عاوزين نحذف اول node

هنخلي ال head يشاور على ال node الثانيه مش الاولى ونحذف الاولى نفس الطريقة السابقه بدون loop



```
//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud
```

```

#include <iostream>
using namespace std;

// Create a node
class Node
{
public:
    int value;
    Node* next;
};

void InsertAtBeginning(Node*& head, int value)
{
    // Allocate memory to a node
    Node* new_node = new Node();

    // insert the data
    new_node->value = value;
    new_node->next = head;

    // Move head to new node
    head = new_node;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {
        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

```

```

}

// Insert a node after a node
void InsertAfter(Node* Prev_node, int Value) {

    if (Prev_node == NULL) {
        cout << "the given Previous node cannot be NULL";
        return;
    }

    Node* new_node = new Node();
    new_node->value = Value;
    new_node->next = Prev_node->next;
    Prev_node->next = new_node;
}

// Insert at the end
void InsertAtEnd(Node*& head, int Value) {

    Node* new_node = new Node();

    new_node->value = Value;
    new_node->next = NULL;

    if (head == NULL) {
        head = new_node;
        return;
    }

    Node* LastNode = head;
    while (LastNode->next != NULL)
    {
        LastNode = LastNode->next;
    }

    LastNode->next = new_node;
    return;
}

// Delete a node
void DeleteNode(Node*& head, int Value) {

    Node* Current = head, * Prev = head;

    if (head == NULL)
    {
        return;
    }

    if (Current->value == Value) {
        head = Current->next;
        delete Current; //free from memory
        return;
    }

    // Find the key to be deleted
    while (Current != NULL && Current->value != Value) {
        Prev = Current;
        Current = Current->next;
    }

    // If the value is not present
    if (Current == NULL) return;

    // Remove the node
    Prev->next = Current->next;
    delete Current; //free from memory
}

```

```

}

// Delete First Node node
void DeleteFirstNode(Node*& head) {

    Node* Current = head;

    if (head == NULL)
    {
        return;
    }

    head = Current->next;
    delete Current;//free from memory
    return;
}

// Print the linked list
void PrintList(Node* head)

{
    cout << "\n";
    while (head != NULL) {
        cout << head->value << " ";
        head = head->next;
    }
}

int main()
{
    Node* head = NULL;

    InsertAtEnd(head, 1);
    InsertAtEnd(head, 2);
    InsertAtEnd(head, 3);
    InsertAtEnd(head, 4);
    InsertAtEnd(head, 5);
    InsertAtEnd(head, 6);
    PrintList(head);

    DeleteFirstNode(head);

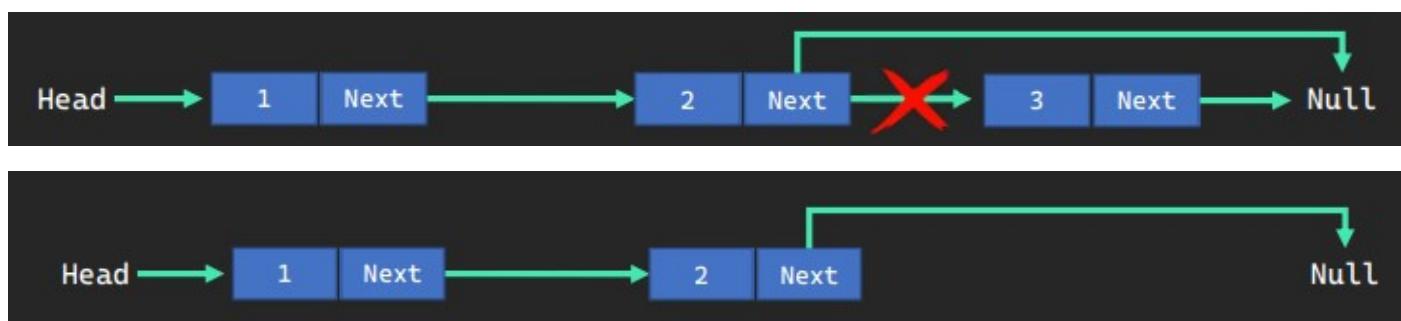
    PrintList(head);

    system("pause>0");
}

```

Operations - Delete Last Node

عاوزين نحذف اخر node هنقدر نلف لحد مانلاقيه ونحذفه



```

//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

#include <iostream>
using namespace std;

// Create a node
class Node
{
public:
    int value;
    Node* next;
};

void InsertAtBeginning(Node*& head, int value)
{
    // Allocate memory to a node
    Node* new_node = new Node();

    // insert the data
    new_node->value = value;
    new_node->next = head;

    // Move head to new node
    head = new_node;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {

        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

// Insert a node after a node
void InsertAfter(Node* Prev_node, int Value) {

    if (Prev_node == NULL) {
        cout << "the given Previous node cannot be NULL";
        return;
    }

    Node* new_node = new Node();
    new_node->value = Value;
    new_node->next = Prev_node->next;
    Prev_node->next = new_node;
}

// Insert at the end
void InsertAtEnd(Node*& head, int Value) {

    Node* new_node = new Node();

    new_node->value = Value;
    new_node->next = NULL;

    if (head == NULL) {
        head = new_node;
    }
}

```

```

        return;
    }

Node* LastNode = head;
while (LastNode->next != NULL)
{
    LastNode = LastNode->next;
}

LastNode->next = new_node;
return;
}

// Delete a node
void DeleteNode(Node*& head, int Value) {

Node* Current = head, * Prev = head;

if (head == NULL)
{
    return;
}

if (Current->value == Value) {
    head = Current->next;
    delete Current; //free from memory
    return;
}

// Find the key to be deleted
while (Current != NULL && Current->value != Value) {
    Prev = Current;
    Current = Current->next;
}

// If the value is not present
if (Current == NULL) return;

// Remove the node
Prev->next = Current->next;
delete Current; //free from memory
}

// Delete First Node node
void DeleteFirstNode(Node*& head) {

Node* Current = head;

if (head == NULL)
{
    return;
}

head = Current->next;
delete Current; //free from memory
return;
}

// Delete Last node
void DeleteLastNode(Node*& head)
{
Node* Current = head, * Prev = head;

if (head == NULL)
{

```

```

        return;
    }

    if (Current->next == NULL) {
        head = NULL;
        delete Current;//free from memory
        return;
    }

    // Find the key to be deleted
    while (Current != NULL && Current->next != NULL) {
        Prev = Current;
        Current = Current->next;
    }

    // Remove the node
    Prev->next = NULL;
    delete Current;//free from memory
}

// Print the linked list
void PrintList(Node* head)

{
    cout << "\n";
    while (head != NULL) {
        cout << head->value << " ";
        head = head->next;
    }
}

int main()
{
    Node* head = NULL;

    InsertAtEnd(head, 1);
    InsertAtEnd(head, 2);
    InsertAtEnd(head, 3);
    InsertAtEnd(head, 4);
    InsertAtEnd(head, 5);
    InsertAtEnd(head, 6);
    PrintList(head);

    DeleteLastNode(head);

    PrintList(head);
    system("pause>0");
}

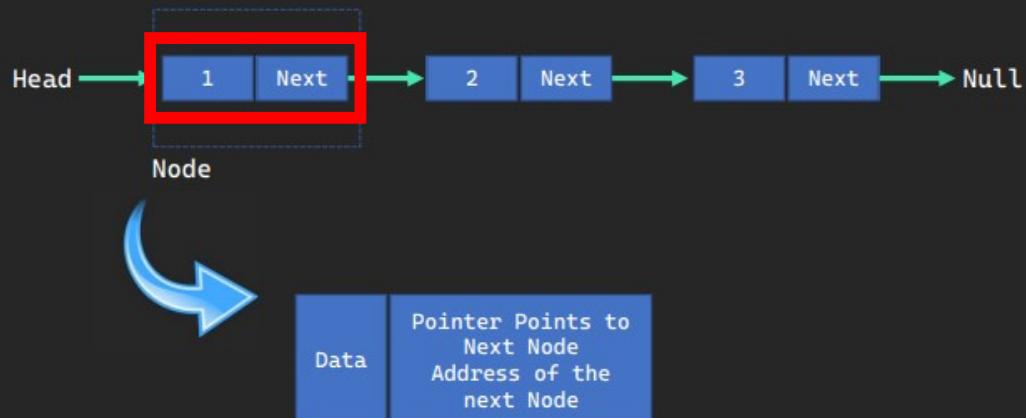
```

What is Doubly Linked List?

ال العادي ال node فيها تكون عباره عن الداتا وال pointer فتمشي في اتجاه واحد

Linked List (Singly Linked List)

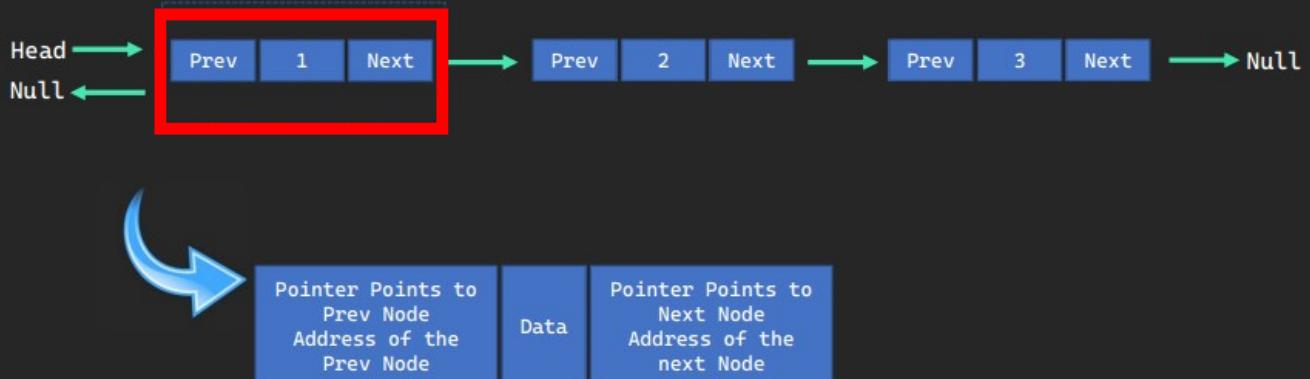
A linked list is a linear data structure that includes a series of connected nodes.



ال doubly linked list بقى بيكون فيها اتنين pointers واحد لـ next والثانى لـ previous

فبتلاقيها بتتمشى في الاتجاهين

Linked List (Doubly)



الواجب

In Doubly Linked List Each node consists of a data value, a pointer to the next node, and a pointer to the previous node.

True

False

In Doubly Linked List: Traversal can occur in both ways.

True

False

In singly linked list: Traversal can occur in both ways.

True

False

In Doubly Linked List: It requires more space because of an extra pointer.

True

False

Doubly Linked List Implementation

```
// ProgrammingAdvices.com
// Mohammed Abu-Hadoud

#include <iostream>
using namespace std;

// Creating a node
class Node

{
public:
    int value;
    Node* next;
    Node* prev;
};

int main()

{
    Node* head;

    Node* Node1 = NULL;
    Node* Node2 = NULL;
    Node* Node3 = NULL;
```

هذا بعمل ال **node** ويزود عليها ال **pointer** **prev** بس من نفس نوع الكلاس

عرف ال **head**

عمل **nodes 3**

```

// allocate 3 nodes in the heap
Node1 = new Node();
Node2 = new Node();
Node3 = new Node();

// Assign value values
Node1->value = 1;
Node2->value = 2;
Node3->value = 3;

// Connect nodes
Node1->next = Node2;
Node1->prev = NULL;

Node2->next = Node3;
Node2->prev = Node1;

Node3->next = NULL;
Node3->prev = Node2;

// print the linked list value
head = Node1;

while (head != NULL) {
    cout << head->value << endl;
    head = head->next;
}

system("pause>0");
return 0;
}

```

وهنا بعمل الحركه دي عشان اعرف اعملها
بعدين delete

هنا بعبي الدانا

وباجي علي كل node بقولها فين ال next وفين
ال prev

وهنا بلف عال nodes زي قبل كده

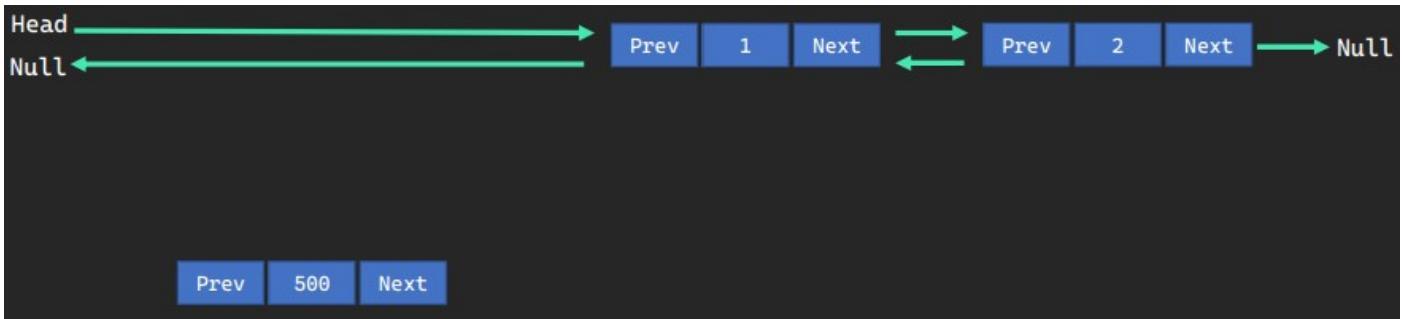
Operations -Insert At Beginning

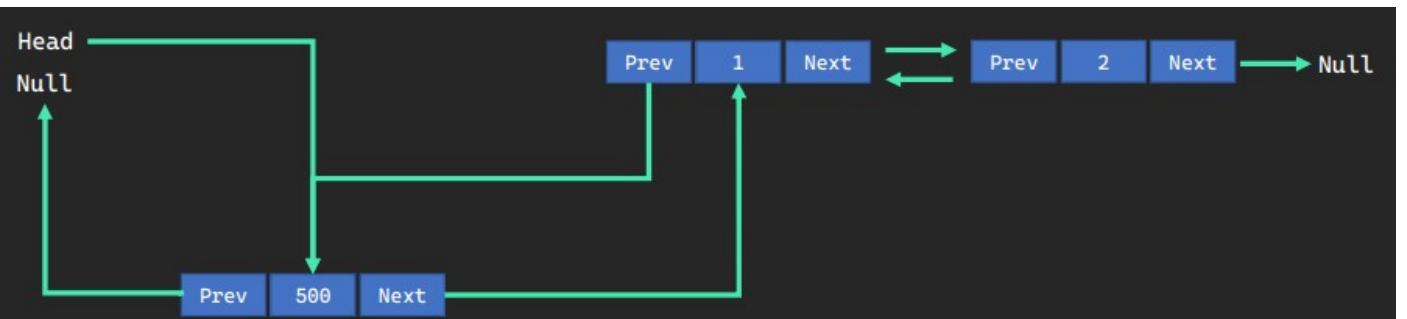
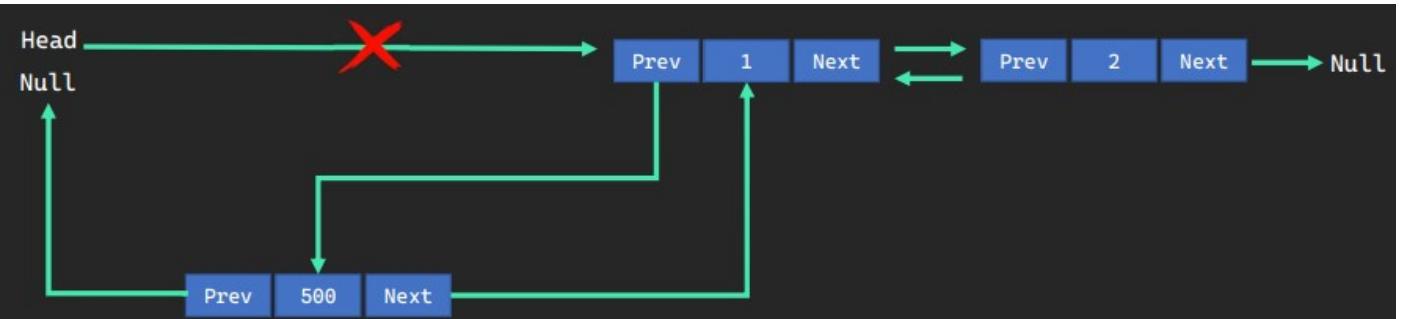
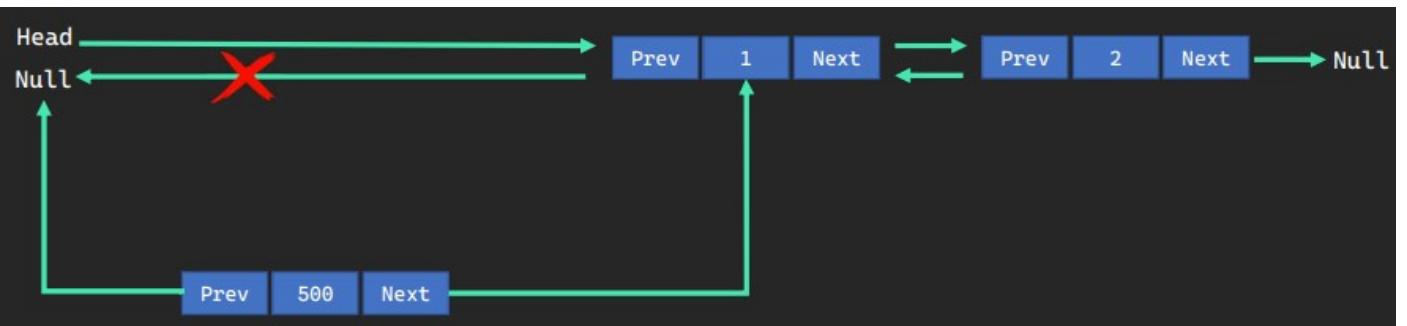
هنضيف عنصر في بداية ال list

اللي هيحصل اننا هنخلي ال prev بتاع اول node يشاور على ال الجديده بدل ما كان بيشاور
على null وال prev بتاع ال الجديده يشاور على null

وال next بتاع ال الجديده يشاور على اول node

كده يتفضل حاجه واحده اننا بدل ما ال head كان بيشاور على اول node لا احنا هنخليه يشاور على
ال الجديده





//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

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

// Create a node
class Node
{
public:
    int value;
    Node* next;
    Node* prev;
};

void InsertAtBeginning(Node*& head, int value) {

    /*
        1-Create a new node with the desired value.
        2-Set the next pointer of the new node to the current head of the list.
        3-Set the previous pointer of the current head to the new node.
        4-Set the new node as the new head of the list.
    */

    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}
```

```

}

void PrintNodeDetails(Node* head)
{
    if (head->prev != NULL)
        cout << head->prev->value;
    else
        cout << "NULL";

    cout << " <--> " << head->value << " <--> ";

    if (head->next != NULL)
        cout << head->next->value << "\n";
    else
        cout << "NULL";
}

// Print the linked list
void PrintListDetails(Node* head)
{
    cout << "\n\n";
    while (head != NULL) {
        PrintNodeDetails(head);
        head = head->next;
    }
}

// Print the linked list
void PrintList(Node* head)
{
    cout << "NULL <--> ";
    while (head != NULL) {
        cout << head->value << " <--> ";
        head = head->next;
    }
    cout << "NULL";
}

int main()
{
    Node* head = NULL;

    InsertAtBeginning(head, 5);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 1);

    cout << "\nLinked List Content:\n";
    PrintList(head);
    PrintListDetails(head);
    system("pause>0");
}

```

Operations - Find Node

نفس الشغل اللي هتعمله في ال linked list العادي هتعمله هنا

```

//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

#include <iostream>

```

```

using namespace std;

// Create a node
class Node
{
public:
    int value;
    Node* next;
    Node* prev;
};

void InsertAtBeginning(Node*& head, int value) {

/*
    1-Create a new node with the desired value.
    2-Set the next pointer of the new node to the current head of the list.
    3-Set the previous pointer of the current head to the new node.
    4-Set the new node as the new head of the list.
*/
    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {

        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

void PrintNodeDetails(Node* head)
{
    if (head->prev != NULL)
        cout << head->prev->value;
    else
        cout << "NULL";

    cout << " <-> " << head->value << " <-> ";

    if (head->next != NULL)
        cout << head->next->value << "\n";
    else
        cout << "NULL";
}

// Print the linked list
void PrintListDetails(Node* head)

```

```

{
    cout << "\n\n";
    while (head != NULL) {
        PrintNodeDetails(head);
        head = head->next;
    }
}

// Print the linked list
void PrintList(Node* head)
{
    cout << "NULL <--> ";
    while (head != NULL) {
        cout << head->value << " <--> ";
        head = head->next;
    }
    cout << "NULL";
}

int main()
{
    Node* head = NULL;

    InsertAtBeginning(head, 5);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 1);

    cout << "\nLinked List Content:\n";
    PrintList(head);

    Node* N1 = Find(head, 2);

    if (N1 != NULL)
        cout << "\n\n Node Found :-)\n";
    else
        cout << "\n\n Node Is not found :-(\n";

    system("pause>0");
}

```

Operations - Insert After

لو عايزين نضيف node5 بعد node3 وقبل node4

هندور علي node3

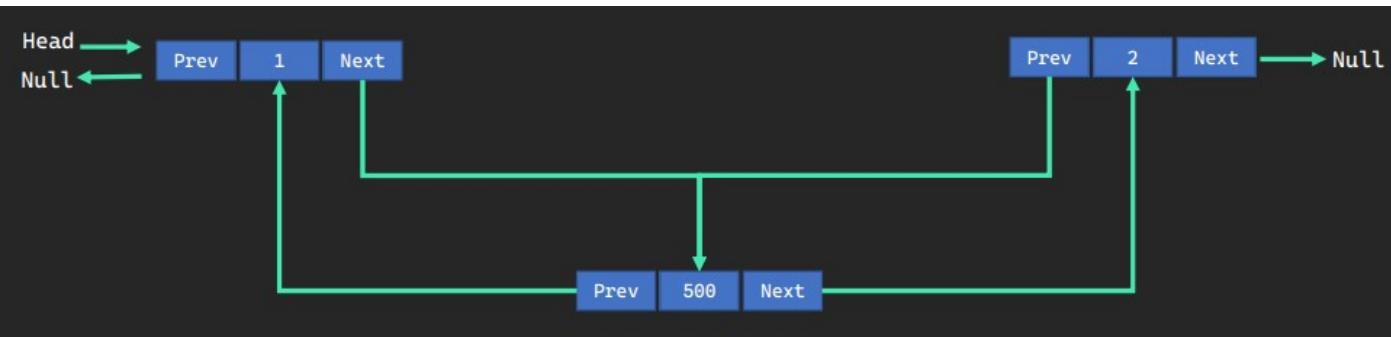
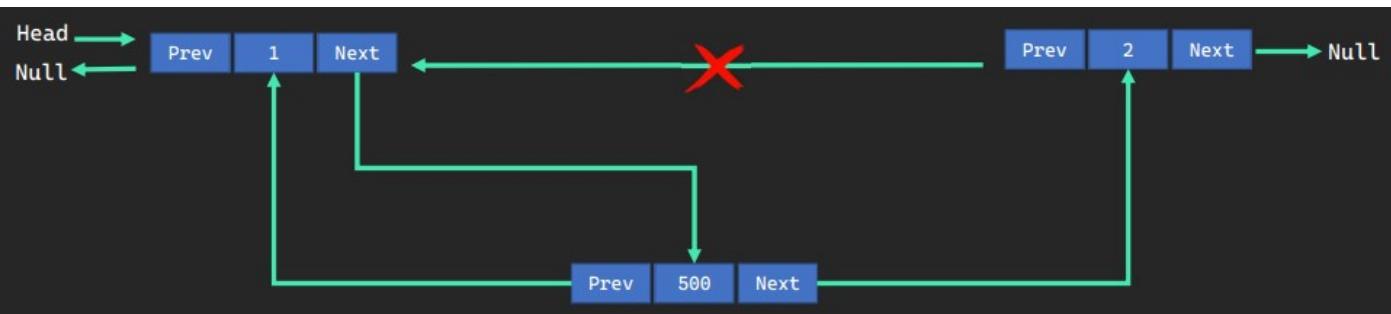
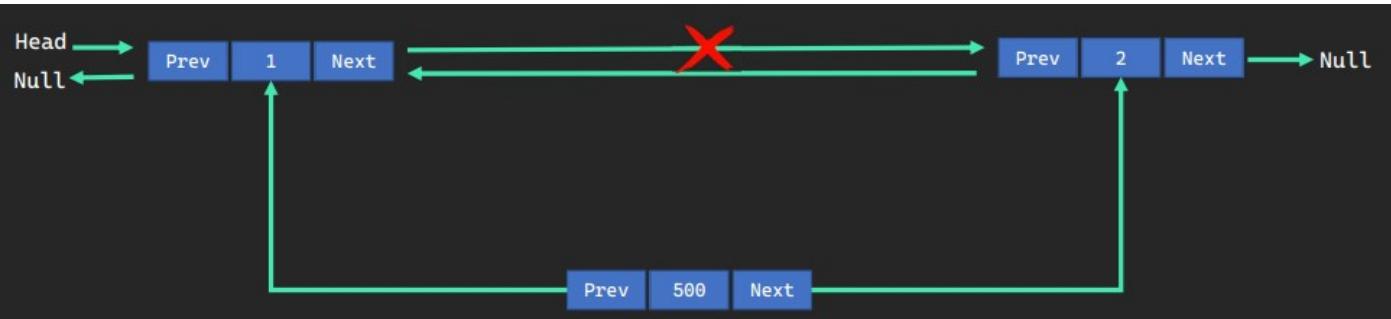
وناخد العنوان اللي موجود في ال next بتاع ال node3 ونحطه في ال next بتاع ال node4

ونخلط ال prev بتاع ال node4 بتو شر علي ال node3

كده خلصنا من ال node4

ال node3 هنخلطي ال next بتاعه بتو شر علي ال node4

وال node5 هنخلطي ال prev بتاعه بتو شر علي ال node4



//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

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

// Create a node
class Node
{
public:
    int value;
    Node* next;
    Node* prev;
};

void InsertAtBeginning(Node*& head, int value) {

    /*
        1-Create a new node with the desired value.
        2-Set the next pointer of the new node to the current head of the list.
        3-Set the previous pointer of the current head to the new node.
        4-Set the new node as the new head of the list.
    */

    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}
```

```

}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {
        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

void InsertAfter(Node* current, int value) {

    /* 1 - Create a new node with the desired value.
       2-Set the next pointer of the new node to the next node of the current node.
       3-Set the previous pointer of the new node to the current node.
       4-Set the next pointer of the current node to the new node.
       5-Set the previous pointer of the next node to the new node(if it exists).
    */

    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = current->next;
    newNode->prev = current;

    if (current->next != NULL) {
        current->next->prev = newNode;
    }
    current->next = newNode;
}

void PrintNodeDetails(Node* head)
{
    if (head->prev != NULL)
        cout << head->prev->value;
    else
        cout << "NULL";

    cout << " <--> " << head->value << " <--> ";

    if (head->next != NULL)
        cout << head->next->value << "\n";
    else
        cout << "NULL";
}

// Print the linked list
void PrintListDetails(Node* head)
{
    cout << "\n\n";
    while (head != NULL) {
        PrintNodeDetails(head);
        head = head->next;
    }
}

```

```

// Print the linked list
void PrintList(Node* head)

{
    cout << "NULL <-> ";
    while (head != NULL) {
        cout << head->value << " <-> ";
        head = head->next;
    }
    cout << "NULL";
}

int main()
{
    Node* head = NULL;

    InsertAtBeginning(head, 5);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 1);

    cout << "\nLinked List Content:\n";
    PrintList(head);
    PrintListDetails(head);

    Node* N1 = Find(head, 2);

    InsertAfter(N1, 500);

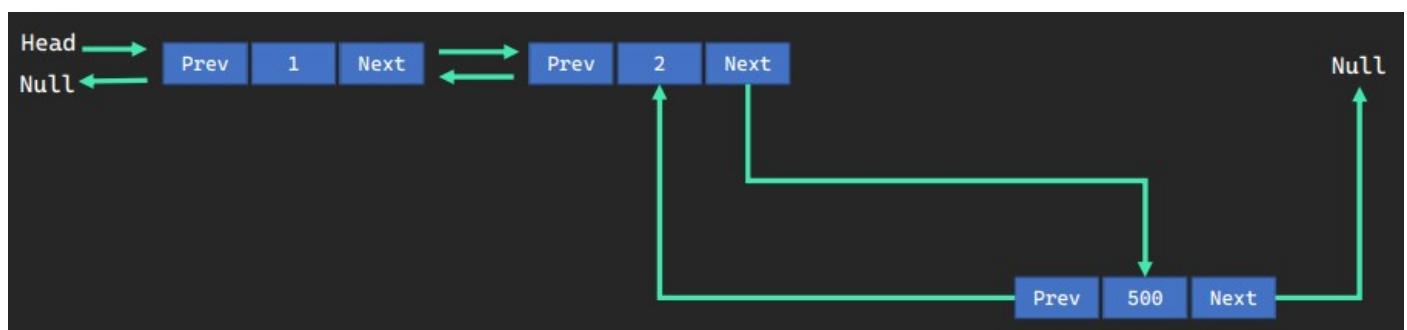
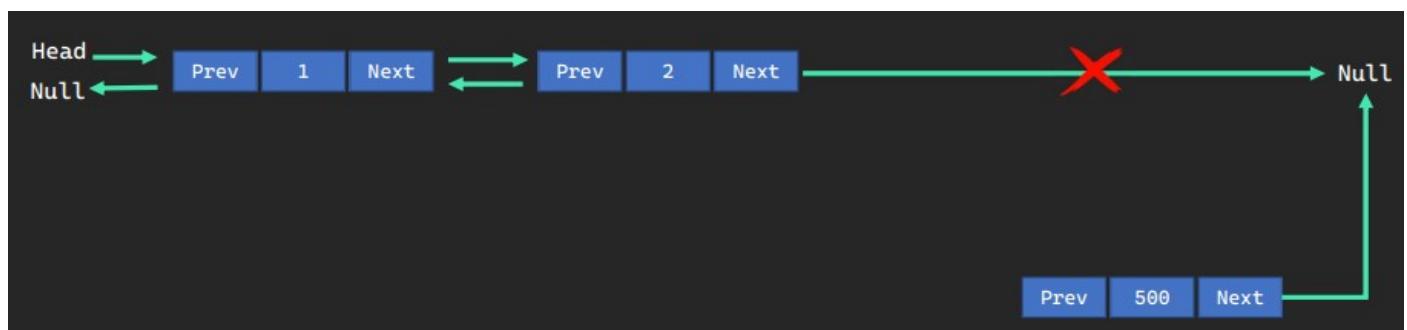
    cout << "\n\n\nLinked List Content after InsertAfter:\n";
    PrintList(head);
    PrintListDetails(head);
    system("pause>0");
}

```

Operations - Insert at End

عشان نضيف node في اخر السلسلة هنعمل node ونخلي ال next بتاعها بيؤشر على null وال node بتاعها بيؤشر على اخر

وبعدين نيجي لآخر node نخلي ال next بتاعه يؤشر على ال الجديد



```

//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

#include <iostream>
using namespace std;

// Create a node
class Node
{
public:
    int value;
    Node* next;
    Node* prev;
};

void InsertAtBeginning(Node*& head, int value) {

    /*
        1-Create a new node with the desired value.
        2-Set the next pointer of the new node to the current head of the list.
        3-Set the previous pointer of the current head to the new node.
        4-Set the new node as the new head of the list.
    */

    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {

        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

void InsertAfter(Node* current, int value) {

    /*
        1 - Create a new node with the desired value.
        2-Set the next pointer of the new node to the next node of the current node.
        3-Set the previous pointer of the new node to the current node.
        4-Set the next pointer of the current node to the new node.
        5-Set the previous pointer of the next node to the new node(if it exists).
    */

    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = current->next;
    newNode->prev = current;

    if (current->next != NULL) {

```

```

        current->next->prev = newNode;
    }
    current->next = newNode;
}

void InsertAtEnd(Node* head, int value) {

/*
    1>Create a new node with the desired value.
    2>Traverse the list to find the last node.
    3>Set the next pointer of the last node to the new node.
    4>Set the previous pointer of the new node to the last node.
*/
Node* newNode = new Node();
newNode->value = value;
newNode->next = NULL;
if (head == NULL) {
    newNode->prev = NULL;
    head = newNode;
}
else {
    Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
}
}

void PrintNodeDetails(Node* head)
{
    if (head->prev != NULL)
        cout << head->prev->value;
    else
        cout << "NULL";

    cout << " <--> " << head->value << " <--> ";

    if (head->next != NULL)
        cout << head->next->value << "\n";
    else
        cout << "NULL";
}

// Print the linked list
void PrintListDetails(Node* head)
{
    cout << "\n\n";
    while (head != NULL) {
        PrintNodeDetails(head);
        head = head->next;
    }
}

// Print the linked list
void PrintList(Node* head)
{
    cout << "NULL <--> ";
    while (head != NULL) {
        cout << head->value << " <--> ";

```

```

        head = head->next;
    }
    cout << "NULL";
}

int main()
{
    Node* head = NULL;

    InsertAtBeginning(head, 5);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 1);

    cout << "\nLinked List Content:\n";
    PrintList(head);
    PrintListDetails(head);

    InsertAtEnd(head, 500);

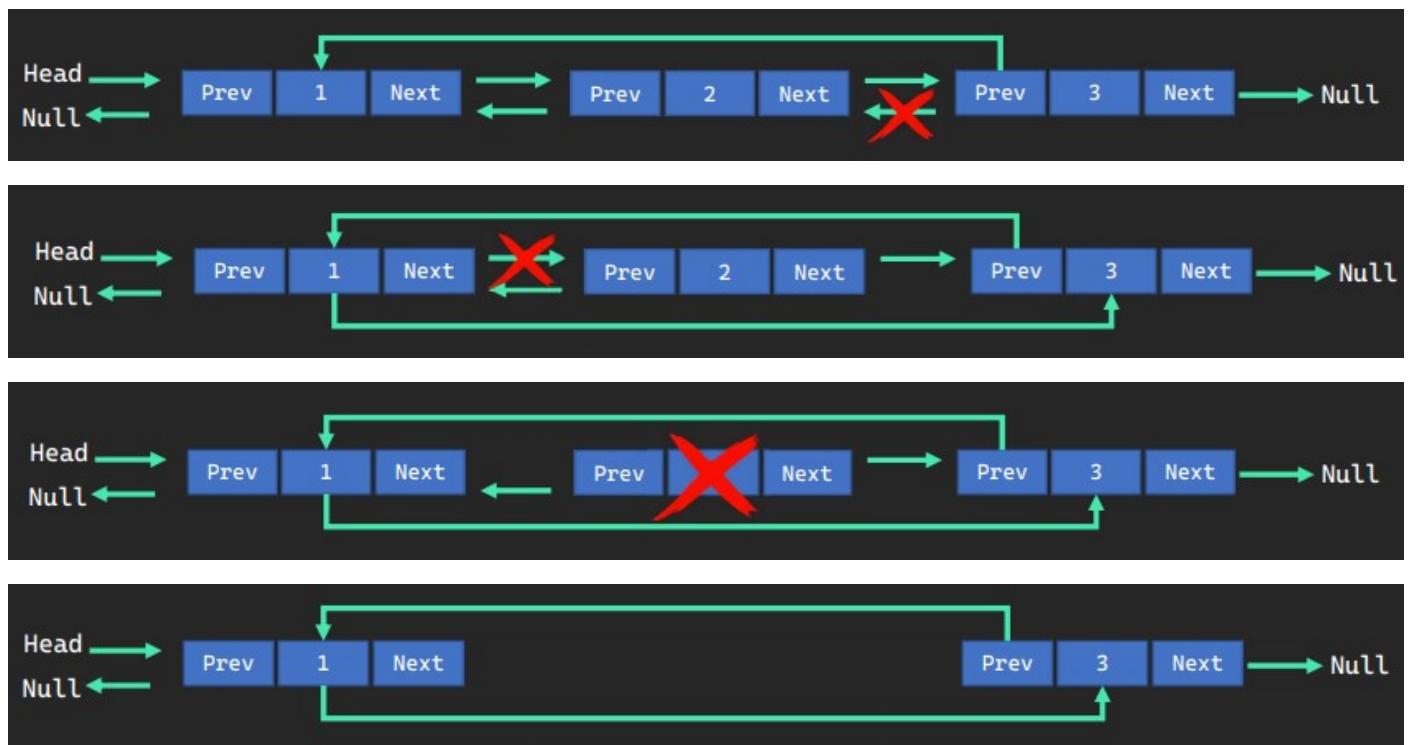
    cout << "\n\nLinked List Content after InsertAtEnd:\n";
    PrintList(head);
    PrintListDetails(head);
    system("pause>0");
}

```

Operations - Delete Node

عشان احذف node معينه بدور عليها وبعدين عايز اربط ال prev بتاعت ال node اللي بعدها باللي قبلها فبستدعها عن طريق اني اكتب `node->next->prev` كده انا وصلت لل node اللي بعدها عن طريق ال `node->next` وبعدين بستدعي ال `prev` بتاعها واديله عنوان اللي قبل اللي انا واقف عليها

ويعدين اجي عال `node` بتاع ال `node` اللي قبلها واخليه يؤشر عاللي بعدها وبعدين احذف ال `node`
يعني بتوصل الاتنين بعض وتسقط اللي في النص



```

//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

#include <iostream>
using namespace std;

// Create a node
class Node
{
public:
    int value;
    Node* next;
    Node* prev;
};

void InsertAtBeginning(Node*& head, int value) {

    /*
        1-Create a new node with the desired value.
        2-Set the next pointer of the new node to the current head of the list.
        3-Set the previous pointer of the current head to the new node.
        4-Set the new node as the new head of the list.
    */

    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {

        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

void InsertAfter(Node* current, int value) {

    /*
        1 - Create a new node with the desired value.
        2-Set the next pointer of the new node to the next node of the current node.
        3-Set the previous pointer of the new node to the current node.
        4-Set the next pointer of the current node to the new node.
        5-Set the previous pointer of the next node to the new node(if it exists).
    */

    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = current->next;
    newNode->prev = current;

    if (current->next != NULL) {

```

```

        current->next->prev = newNode;
    }
    current->next = newNode;
}

void InsertAtEnd(Node* head, int value) {

/*
    1>Create a new node with the desired value.
    2>Traverse the list to find the last node.
    3>Set the next pointer of the last node to the new node.
    4>Set the previous pointer of the new node to the last node.
*/
Node* newNode = new Node();
newNode->value = value;
newNode->next = NULL;
if (head == NULL) {
    newNode->prev = NULL;
    head = newNode;
}
else {
    Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
}
}

void DeleteNode(Node*& head, Node*& NodeToDelete) {

/*
    1>Set the next pointer of the previous node to the next pointer of the current node.
    2>Set the previous pointer of the next node to the previous pointer of the current
node.
    3>Delete the current node.
*/
if (head == NULL || NodeToDelete == NULL) {
    return;
}
if (head == NodeToDelete) {
    head = NodeToDelete->next;
}
if (NodeToDelete->next != NULL) {
    NodeToDelete->next->prev = NodeToDelete->prev;
}
if (NodeToDelete->prev != NULL) {
    NodeToDelete->prev->next = NodeToDelete->next;
}
delete NodeToDelete;
}

void PrintNodeDetails(Node* head)
{
    if (head->prev != NULL)
        cout << head->prev->value;
    else
        cout << "NULL";

    cout << " <--> " << head->value << " <--> ";

    if (head->next != NULL)
        cout << head->next->value << "\n";
}

```

```

    else
        cout << "NULL";
}

// Print the linked list
void PrintListDetails(Node* head)
{
    cout << "\n\n";
    while (head != NULL) {
        PrintNodeDetails(head);
        head = head->next;
    }
}

// Print the linked list
void PrintList(Node* head)
{
    cout << "NULL <-> ";
    while (head != NULL) {
        cout << head->value << " <-> ";
        head = head->next;
    }
    cout << "NULL";
}

int main()
{
    Node* head = NULL;

    InsertAtBeginning(head, 5);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 1);

    cout << "\nLinked List Content:\n";
    PrintList(head);
    PrintListDetails(head);

    // Traverse the list to find the node to be deleted.
    Node* N1 = Find(head, 4);

    DeleteNode(head, N1);

    cout << "\n\nLinked List Content after delete:\n";
    PrintList(head);
    PrintListDetails(head);
    system("pause>0");
}

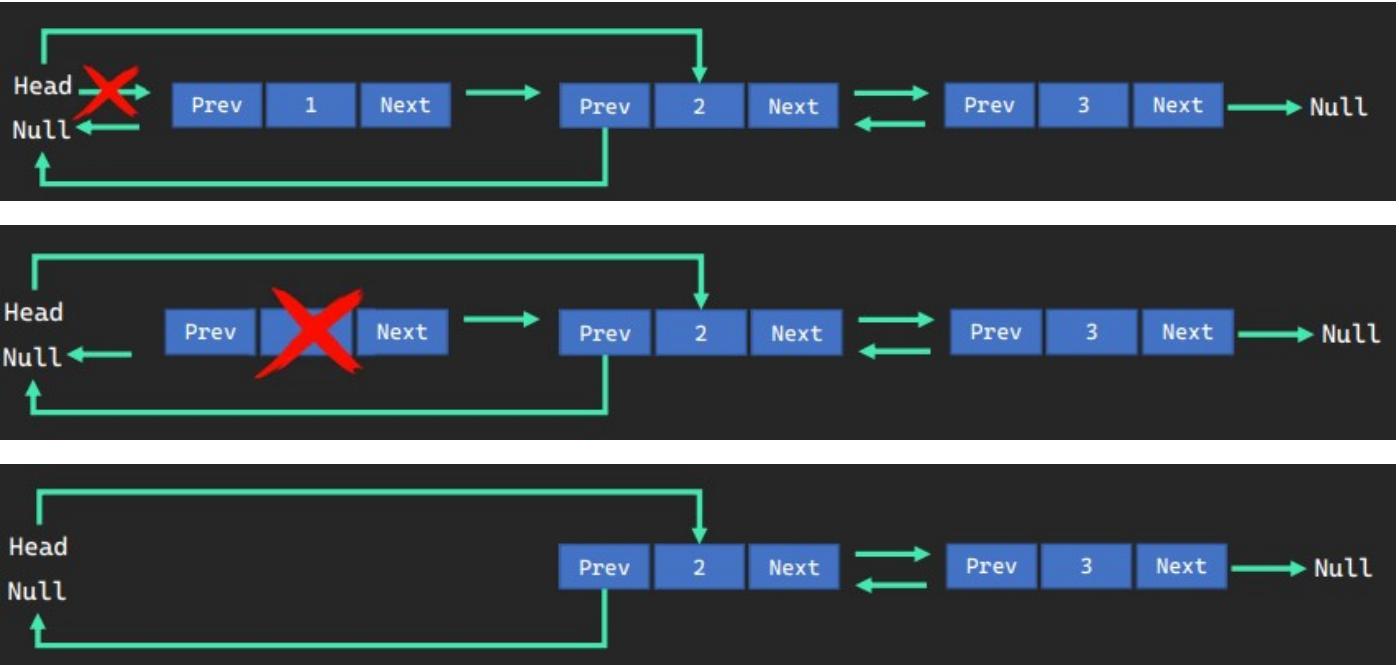
```

Operations - Delete First Node

عاوزين نحذف اول node

بنخلی ال head بیؤشر على node ونیجي لـ next null بنیجي لـ prev على تاني node ونحذف اول





//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

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

// Create a node
class Node
{
public:
    int value;
    Node* next;
    Node* prev;
};

void InsertAtBeginning(Node*& head, int value) {

    /*
        1-Create a new node with the desired value.
        2-Set the next pointer of the new node to the current head of the list.
        3-Set the previous pointer of the current head to the new node.
        4-Set the new node as the new head of the list.
    */
    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}

Node* Find(Node* head, int Value) {

    while (head != NULL) {

        if (head->value == Value)
            return head;

        head = head->next;
    }
}
```

```

    }

    return NULL;
}

}

void InsertAfter(Node* current, int value) {

/* 1 - Create a new node with the desired value.
   2-Set the next pointer of the new node to the next node of the current node.
   3-Set the previous pointer of the new node to the current node.
   4-Set the next pointer of the current node to the new node.
   5-Set the previous pointer of the next node to the new node(if it exists).
*/
Node* newNode = new Node();
newNode->value = value;
newNode->next = current->next;
newNode->prev = current;

if (current->next != NULL) {
    current->next->prev = newNode;
}
current->next = newNode;
}

void InsertAtEnd(Node* head, int value) {

/*
  1-Create a new node with the desired value.
  2-Traverse the list to find the last node.
  3-Set the next pointer of the last node to the new node.
  4-Set the previous pointer of the new node to the last node.
*/
Node* newNode = new Node();
newNode->value = value;
newNode->next = NULL;
if (head == NULL) {
    newNode->prev = NULL;
    head = newNode;
}
else {
    Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
}
}

void DeleteNode(Node*& head, Node*& NodeToDelete) {

/*
  1-Set the next pointer of the previous node to the next pointer of the current node.
  2-Set the previous pointer of the next node to the previous pointer of the current
node.
  3>Delete the current node.
*/
if (head == NULL || NodeToDelete == NULL) {
    return;
}
if (head == NodeToDelete) {
    head = NodeToDelete->next;
}
}

```

```

}

if (NodeToDelete->next != NULL) {
    NodeToDelete->next->prev = NodeToDelete->prev;
}
if (NodeToDelete->prev != NULL) {
    NodeToDelete->prev->next = NodeToDelete->next;
}
delete NodeToDelete;
}

void DeleteFirstNode(Node*& head) {
/*
    1-Store a reference to the head node in a temporary variable.
    2-Update the head pointer to point to the next node in the list.
    3-Set the previous pointer of the new head to NULL.
    4>Delete the temporary reference to the old head node.
*/
if (head == NULL) {
    return;
}
Node* temp = head;
head = head->next;
if (head != NULL) {
    head->prev = NULL;
}
delete temp;
}

void PrintNodeDetails(Node* head)
{
    if (head->prev != NULL)
        cout << head->prev->value;
    else
        cout << "NULL";

    cout << " <-> " << head->value << " <-> ";

    if (head->next != NULL)
        cout << head->next->value << "\n";
    else
        cout << "NULL";
}

// Print the linked list
void PrintListDetails(Node* head)
{
    cout << "\n\n";
    while (head != NULL) {
        PrintNodeDetails(head);
        head = head->next;
    }
}

// Print the linked list
void PrintList(Node* head)
{
    cout << "NULL <-> ";
    while (head != NULL) {
        cout << head->value << " <-> ";
        head = head->next;
    }
    cout << "NULL";
}

```

```

}

int main()
{
    Node* head = NULL;

    InsertAtBeginning(head, 5);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 1);

    cout << "\nLinked List Content:\n";
    PrintList(head);
    PrintListDetails(head);

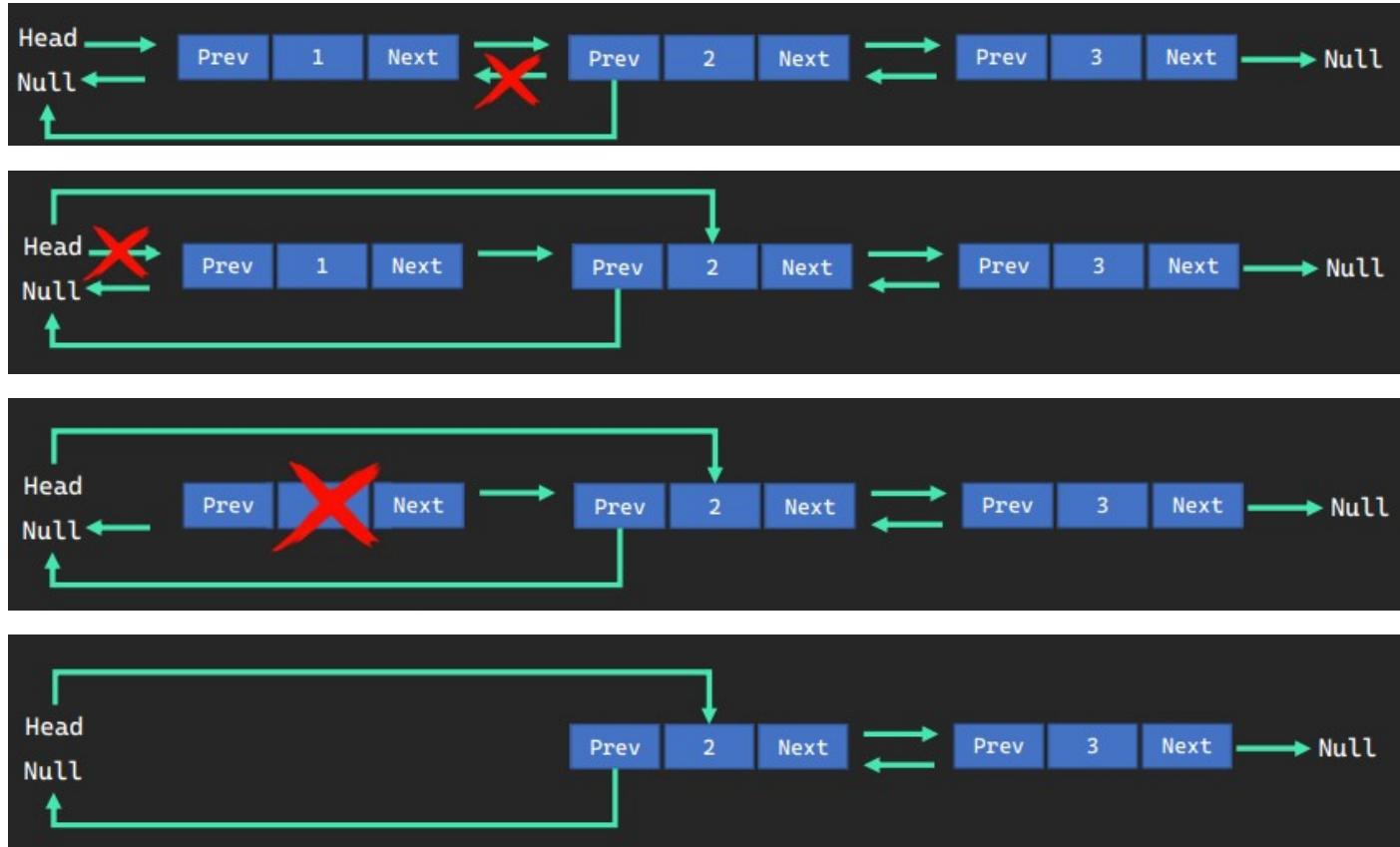
    DeleteFirstNode(head);

    cout << "\n\nLinked List Content after delete:\n";
    PrintList(head);
    PrintListDetails(head);
    system("pause>0");
}

```

Operations - Delete Last Node

عشن نحذف اخر node بنبيجي عاللي قبلها نخلي ال next بـ null ونحذف اخر احده



//ProgrammingAdvices.com
//Mohammed Abu-Hadhoud

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

// Create a node

```

```

class Node
{
public:
    int value;
    Node* next;
    Node* prev;
};

void InsertAtBeginning(Node*& head, int value) {

/*
    1-Create a new node with the desired value.
    2-Set the next pointer of the new node to the current head of the list.
    3-Set the previous pointer of the current head to the new node.
    4-Set the new node as the new head of the list.
*/
    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = head;
    newNode->prev = NULL;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}

Node* Find(Node* head, int Value)
{
    while (head != NULL) {

        if (head->value == Value)
            return head;

        head = head->next;
    }

    return NULL;
}

void InsertAfter(Node* current, int value) {

/* 1 - Create a new node with the desired value.
   2-Set the next pointer of the new node to the next node of the current node.
   3-Set the previous pointer of the new node to the current node.
   4-Set the next pointer of the current node to the new node.
   5-Set the previous pointer of the next node to the new node(if it exists).
*/
    Node* newNode = new Node();
    newNode->value = value;
    newNode->next = current->next;
    newNode->prev = current;

    if (current->next != NULL) {
        current->next->prev = newNode;
    }
    current->next = newNode;
}

void InsertAtEnd(Node* head, int value) {

```

```

/*
  1>Create a new node with the desired value.
  2-Traverse the list to find the last node.
  3-Set the next pointer of the last node to the new node.
  4-Set the previous pointer of the new node to the last node.
*/
Node* newNode = new Node();
newNode->value = value;
newNode->next = NULL;
if (head == NULL) {
    newNode->prev = NULL;
    head = newNode;
}
else {
    Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
}
}

void DeleteNode(Node*& head, Node*& NodeToDelete) {

/*
  1-Set the next pointer of the previous node to the next pointer of the current node.
  2-Set the previous pointer of the next node to the previous pointer of the current
node.
  3>Delete the current node.
*/
if (head == NULL || NodeToDelete == NULL) {
    return;
}
if (head == NodeToDelete) {
    head = NodeToDelete->next;
}
if (NodeToDelete->next != NULL) {
    NodeToDelete->next->prev = NodeToDelete->prev;
}
if (NodeToDelete->prev != NULL) {
    NodeToDelete->prev->next = NodeToDelete->next;
}
delete NodeToDelete;
}

void DeleteFirstNode(Node*& head) {

/*
  1-Store a reference to the head node in a temporary variable.
  2-Update the head pointer to point to the next node in the list.
  3-Set the previous pointer of the new head to NULL.
  4>Delete the temporary reference to the old head node.
*/
if (head == NULL) {
    return;
}
Node* temp = head;
head = head->next;
if (head != NULL) {
    head->prev = NULL;
}
delete temp;
}

```

```

void DeleteLastNode(Node*& head) {
    /*
        1-Traverse the list to find the last node.
        2-Set the next pointer of the second-to-last node to NULL.
        3>Delete the last node.
    */
    if (head == NULL) {
        return;
    }
    if (head->next == NULL) {
        delete head;
        head = NULL;
        return;
    }
    Node* current = head;

    //here we find the node before the last node
    while (current->next->next != NULL) {
        current = current->next;
    }
    Node* temp = current->next;
    current->next = NULL;
    delete temp;
}

void PrintNodeDetails(Node* head)
{
    if (head->prev != NULL)
        cout << head->prev->value;
    else
        cout << "NULL";

    cout << " <--> " << head->value << " <--> ";

    if (head->next != NULL)
        cout << head->next->value << "\n";
    else
        cout << "NULL";
}

// Print the linked list
void PrintListDetails(Node* head)
{
    cout << "\n\n";
    while (head != NULL) {
        PrintNodeDetails(head);
        head = head->next;
    }
}

// Print the linked list
void PrintList(Node* head)
{
    cout << "NULL <--> ";
    while (head != NULL) {
        cout << head->value << " <--> ";
        head = head->next;
    }
    cout << "NULL";
}

```

```

int main()
{
    Node* head = NULL;

    InsertAtBeginning(head, 5);
    InsertAtBeginning(head, 4);
    InsertAtBeginning(head, 3);
    InsertAtBeginning(head, 2);
    InsertAtBeginning(head, 1);

    cout << "\nLinked List Content:\n";
    PrintList(head);
    PrintListDetails(head);

    DeleteLastNode(head);

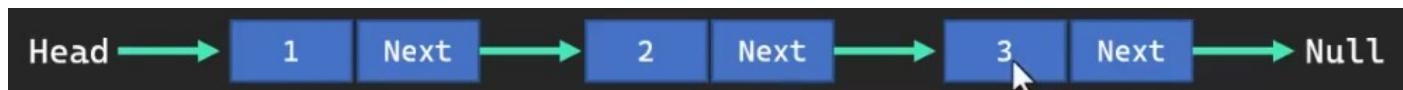
    cout << "\n\nLinked List Content after delete last node:\n";
    PrintList(head);
    PrintListDetails(head);
    system("pause>0");
}

```

What is Circular Linked List?

فيه نوع من الـ **linked list** اسمه **circular linked list** وممكن يكون يالما او

دي **single linked list**



عشان اخليها **circular** بخلي اخر **node** فيها يؤشر علي اول واحد



عشان تعملها **traversing** يعني عشان تمشي علي ال **nodes** اللي فيها هتدخل في

دي ال **doubly linked list**



عشان اخليها **circular** بخلي ال **next** بتاع اخر واحده يؤشر علي اول واحده

واخلي ال **prev** بتاع اول واحده يؤشر علي اخر واحده



من تطبيقاتها هو ال **buffer** اني لما بعبي في ال **buffer** ويبخلص بخليه يرجع لأول واحده ويعبي فيها داتا جديده

في الشاشه السوداء اللي كنا بنعمل عليها التطبيقات كنا لما نطبع داتا كتيره كان بيفضل يطبع لحد ماال buffer يتملي وبعدين يحذف القديم كله ويطبعلك الجديد زي لما طلب منك تطبع كل الحروف من aaa الي zzz لما كان بيخلص كنت بتدور على ال aaa ماكنتش بتلاقيه ده لانه لما ال buffer اتعبي حذف القديم وطبع مكانه

What is ADT?

ال ADT هو اختصار ل abstract data type

لما كنا في مكتبه ال stl كنت بتعامل مع ال vector كنت بتعرف من النوع data type وكنت بتعامل مع ال size وال push وال pop وخلافه بس مش عارف ال vector ازاي بيشتغل من جوه

هوا ده مفهوم ال abstract data type هو عباره عن كلاس فيه functions و procedures بتساعدك علي انك تتعامل مع ال data structure اللي عندك بسهوله بدون ماتعرف ازاي بيشتغل من جوه

زي ما كنت بتعمل كلاس employee مثل انت بتعمل abstract data type للناس اللي بره وبتخفي عنهم تفاصيل العمليات اللي بتحصل جوه الكلاس

زي ال abstract data type وال stack وال queue واي كلاس بيعمل هو vector type

Abstract Data Type (ADT)

- ADT stands for Abstract Data Type. It is a mathematical model of a data type, where the behavior and properties of the data type are defined, but not its implementation.
- ADT is a set of objects together with a set of operations.
- In other words, an ADT defines what a data type can do, but not how it does it.
- An ADT specifies a set of operations that can be performed on the data type, and defines the behavior of those operations.
- Examples: Vectors, Stack, Queue, ...etc

What is Map?

احنا دلوقتي عارفين ال array وعارفين اننا لما بنحتاج نبحث عن عنصر لازم نلف عالعناصر كلها لحد مانلاقيه

هنا بقى بيقولك انه فيه حاجه اسمها `map` ودي عباره عن `array` بس بتقدر توصل للعناصر اللي فيها عن طريق `key` وبيتم اعتبارها `associative container` يعني بترتبط الـ `key` مع الـ `value` والعنصر اللي في الـ `map` بتكون عباره عن `key` و `value` حاجه اشبه بال `tree` بيقولك انها مبنيه على حاجه اسمها `red black tree` وال `key` ماينفعش يتكرر في الـ `map` بتاعتك في الـ `c++` اسمها `map` وفي اللغات الثانيه زي الـ `c#` اسمها `dictionary` وبيقولك انك لما بتجي تخزن العناصر بتترتب لوحدها تصاعديا وده بيساعد في عمليات البحث والاضافه والتعديل عشان نستخدم الـ `map` بنضيف المكتبه `Map` وبعدين بناخد `object` من الكلاس `map` وفي الـ `template` بنحدد نوع للـ `key` ونوع للـ `value` زي كده

```
#include <iostream>
#include <Map>

using namespace std;

int main()
{
    map<string, int> studentsGrades;
```

طيب عايزين نضيف عناصر للـ `map` دي

هتفتح قوسين تحط فيهم `key` وتقول بيساوي القيمه اللي انت عايزها زي كده
`studentsGrades["Ali"] =77 ;`

طيب كل عنصر موجود في الـ `map` بيكون من متغيرين اتنين واحد بيكون الـ `key` والثاني بيكون القيمه

قالك ان الـ `key` بيتخزن في متغير اسمه `first` والـ `value` بيتخزن في متغير اسمه `second`
وبالتالي لو عايز الف عالعناصر كلها يستخدم `for each`

```
for (const auto& pair: studentsGrades) {
    cout << "Student: " << pair.first << ", Grade: " << pair.second << endl;
}
```

في الكود اللي فات الـ `const` ده يعني بيعرف المتغير علي انه ثابت لايمكن تغييره في الدوره الواحده للـ `for loop`

الـ `auto` دي معناها ان المتغير ممكن يكون بأي نوع `int` او `string` او أي نوع ثاني
وعلامة الـ `&` معناها انه هيخزن العنصر `by ref` وبعدين بيقوله لف علي كل عنصر واطبع الـ `key` والـ `value` بتوعه

ال `map` فيها `function` اسمها `end` ودي بترجملك العنصر اللي بعد اخر عنصر في ال `map` وفي ال `map` فيه `function` اسمها `find` بتديها ال `key` وبترجملك ال `value` لو ال `key` موجود ولو مش موجود بترجملك نفس اللي بترجمه ال `function` اللي اسمها `end` وبالتالي لو عايز دور على عنصر معين بتتأكد الأول ان اللي طالعلي من ال `find` لايساوي العنصر اللي بعد الأخير في ال `map` بقى

```
if (studentsGrades.find(StudentName) != studentsGrades.end()) {
    cout << StudentName << "'s Grade in the Map..\n";
}
else {
    cout << "Grade not found For " << StudentName << endl;
}
```

وده الكود كله

```
#include <iostream>
#include <Map>

using namespace std;

int main()
{
    map<string, int> studentsGrades;

    studentsGrades["Ali"] = 77 ;
    studentsGrades["Ahmed"] = 85;
    studentsGrades["Fadi"] = 95;

    cout << "\nPrinting all map values..\n\n";
    for (const auto&pair:studentsGrades) {
        cout << "Student: " << pair.first << ", Grade: " << pair.second << endl;
    }

    cout << "\nFinding Fadi's Grade in th map..\n";
    string StudentName = "Fadi";
    if (studentsGrades.find(StudentName) != studentsGrades.end()) {
        cout << StudentName << "'s Grade in the Map..\n";
    }
    else {
        cout << "Grade not found For " << StudentName << endl;
    }
    cout << "\nFinding Fadi's Grade in th map..\n";
    StudentName = "Omar";
    if (studentsGrades.find(StudentName) != studentsGrades.end()) {
        cout << StudentName << "'s Grade in the Map..\n";
    }
    else {
        cout << "Grade not found For " << StudentName << endl;
    }
}
```

What is Map?

- In C++, a `std::map` is a container in the Standard Template Library (STL) that represents an associative array.
- It is a sorted associative container that contains key-value pairs, where each key must be unique.
- The elements are ordered based on the keys, which are sorted in ascending order by default.
- This sorting allows for efficient search, insertion, and deletion of elements.
- In many programming languages, the concept of a map is similar to that of a dictionary.

What is map?

In C++, a `std::map` is a container in the Standard Template Library (STL) that represents an associative array. It is a sorted associative container that contains key-value pairs, where each key must be unique. The elements are ordered based on the keys, which are sorted in ascending order by default. This sorting allows for efficient search, insertion, and deletion of elements.

Here are some key characteristics and features of `std::map`:

- Associative Container:
 - `std::map` is an associative container, meaning it associates a key with a value.
- Sorted Order:
 - The elements are sorted based on the keys. By default, sorting is done using the `<` operator.
- Unique Keys:
 - Each key in a `std::map` must be unique. If you attempt to insert a key that already exists, the associated value will be updated.
- Balanced Binary Search Tree:

- Internally, `std::map` is usually implemented as a balanced binary search tree, such as a red-black tree. This ensures efficient search, insertion, and deletion operations.
- Efficient Lookups:
 - Searching for an element based on its key has a time complexity of $O(\log n)$, making it efficient for large datasets.
- Dynamic Sizing:
 - `std::map` dynamically adjusts its size as elements are inserted or removed.
- Key-Value Pairs:
 - Each element in the map is a key-value pair, where the key and value can be of any data type.
- Iterator Support:
 - It provides iterators that allow you to traverse the elements in sorted order.

In many programming languages, the concept of a `map` is similar to that of a `dictionary`. Both represent an associative container that stores key-value pairs, allowing efficient retrieval, insertion, and deletion of elements based on their keys.

Map example:

```
#include <iostream>
#include <map>

using namespace std;

int main() {
    // Declare a map with string keys and int values
    map<string, int> studentGrades;

    // Inserting grades for three students
    studentGrades["Ali"] = 77;      // Assigning a grade of 77 to the student "Ali"
    studentGrades["Ahmed"] = 85;    // Assigning a grade of 85 to the student "Ahmed"
```

```

studentGrades["Fadi"] = 95;      // Assigning a grade of 95 to the student "Fadi"

// Printing all map values
cout << "\nPrinting all map values..\n\n";

// Iterating over the map
for (const auto& pair : studentGrades) {
    cout << "Student: " << pair.first << ", Grade: " << pair.second << endl;
}

// Finding the grade for a specific student
string studentName = "Fadi";
if (studentGrades.find(studentName) != studentGrades.end()) {
    cout << studentName << "'s grade: " << studentGrades[studentName] << endl;
} else {
    cout << "Grade not found for " << studentName << endl;
}

return 0;
}

```

In this example, the `map` is used to store mappings from integers to strings, demonstrating basic operations like insertion, lookup, and iteration.

Quiz

What is a `std::map` in C++?

A container in the Standard Template Library (STL) that represents an associative array

A type of array that stores key-value pairs

A sorting algorithm in C++

What is the default sorting order of std::map?

Descending order

Random order

Ascending order

What happens if you attempt to insert a duplicate key in a std::map?

The value associated with the key will be updated

The program will throw an error

The duplicate key will be ignored

Union

دلوقي انا عندي برنامج وفيه جزء معين من البرنامج الجزء ده تحتاج اخزن فيه قيمه معينه
القيميه دي بتتغير ساعات بتكون int وساعات بتكون float وساعات بتكون double
فبيقولك انك طالما تحتاج تخزن قيمه واحده ماتقدرش تعمل متغير لـ float ومتغير تاني لـ int لأنك
كده اهدرت مساحه
اما اعمل ايه؟

قالك فيه data type union شبيهه بال auto كده اسمه
بس المميز هنا انك بتقدر تحدد ال data types اللي عايز تدخلها او عايز تبدل بينهم وماتهدرش مساحه
يعني باختصار بتقول لل compiler انا عندي متغير واحد هخزن فيه data type int يا ماما ه تكون
ياما ه تكون float ماتدورش في أي نوع تاني غيرهم
(انت في ال auto بتحدد ال data type اول مره بس ولو جيت تخزن فيه نوع تاني هيضرب منك)

مساحة ال union دي بتتحدد حسب اكبر data type تم تحديده يعني لو خزنت int و Boolean
مساحة ال int data type ه تكون من مساحة ال data type
ولو جيت تخزن داتا من النوع bool وبعدين جيت تخزن داتا من النوع int ال bool بيتحذف ويتم
استبداله بال int

وعشان كده لازم تخلي بالك من انك لازم تكون عارف ايه هيا اخر data type اتخزنـت عشان البرنامج بتاعك مايضرـ بش في وشك

تعريف ال union بيكون زي تعريف ال struct بالضبط مع اختلاف الوظيفـه يعني ال datatype اكتر من مساحـه لاكتـر من متـغير انما ال union بيـخـزن مساحـه واحدـه لاكتـر من

```
#include <iostream>

union MyUnion {
    int intValue;
    float floatValue;
    char charValue;
};

int main() {
    MyUnion myUnion;

    myUnion.intValue = 42;
    std::cout << "Integer value: " << myUnion.intValue << std::endl;

    myUnion.floatValue = 3.14f;
    std::cout << "Float value: " << myUnion.floatValue << std::endl;

    myUnion.charValue = 'A';
    std::cout << "Char value: " << myUnion.charValue << std::endl;

    return 0;
}
```

What is Union?

- In C++, a union is a user-defined data type that allows you to use the same memory location for different types of data.
- The main reason to use a union is to save memory.

Example:

```
union MyUnion
{
    int intValue;
    float floatValue;
    char charValue;
};
```

Note: the size of a union is determined by the size of its largest member.

Unlike structures, where each member has its own memory space, members of a union share the same memory space.

In C++, a union is a user-defined data type that allows you to use the same memory location for different types of data. Unlike structures, where each member has its own memory space, members of a union share the same memory space.

Here's a simple example of a union in C++:

```
#include <iostream>

union MyUnion {
    int intValue;
    float floatValue;
    char charValue;
};

int main() {
    MyUnion myUnion;

    myUnion.intValue = 42;
    std::cout << "Integer value: " << myUnion.intValue << std::endl;

    myUnion.floatValue = 3.14f;
    std::cout << "Float value: " << myUnion.floatValue << std::endl;

    myUnion.charValue = 'A';
    std::cout << "Char value: " << myUnion.charValue << std::endl;

    return 0;
}
```

In this example, `MyUnion` is a union that can store an integer, a float, or a character. When you assign a value to one of the members, it modifies the shared memory space, and accessing another member will give you the interpretation of that data based on its type.

Keep in mind that unions have some limitations and should be used with caution because they can lead to undefined behavior if not used properly. When one member of the union is accessed, you should be careful about which member was last assigned a value. Also, the size of a union is determined by the size of its largest member.

- Undefined Behavior:
 - When you use a union, it's important to be mindful of which member was last assigned a value. Accessing a member that wasn't the last one written to results in undefined behavior. This is because the union shares the same memory space for all its members. If you read from a member that was not the last one written to, the value might not make sense for that type.
- Careful Member Access:
 - If you assign a value to one member of the union and then access another member, you might get unexpected results. For example, if you store an integer and then read it as a floating-point number, the interpretation of the bits will be different.
- Size Determined by Largest Member:
 - The size of a union is determined by the size of its largest member. This means that if you have a union with an `int`, a `float`, and a `char`, the size of the union will be the size of the `float` since it's the largest member. This can lead to inefficient memory usage if you're not careful with the design of your union.

Quiz

What is a union in C++?

A user-defined data type that allows you to use the same memory location for different types of data.

A type of loop statement in C++.

A function that performs mathematical calculations in C++.

A keyword used to define variables in C++.

How are members of a union stored in memory?

Each member has its own memory space.

Members of a union share the same memory space.

Members of a union are stored in a separate memory space.

The order in which members are stored in memory is undefined.

What happens when you assign a value to one member of a union?

It modifies the shared memory space of the union.

Only the assigned member's memory space is modified.

The size of the union changes to accommodate the assigned member.

The value of other union members is preserved.

What is the size of a union determined by?

The size of its smallest member.

The average size of all its members.

The sum of the sizes of all its members.

The size of its largest member.

What can happen if you access a member of a union that wasn't the last
one written to?

The union will automatically assign a default value to the accessed member.

The union will throw an error and terminate the program.

The value might not make sense for that type, resulting in undefined behavior.

The accessed member will change its value to match the last assigned member.

THE END