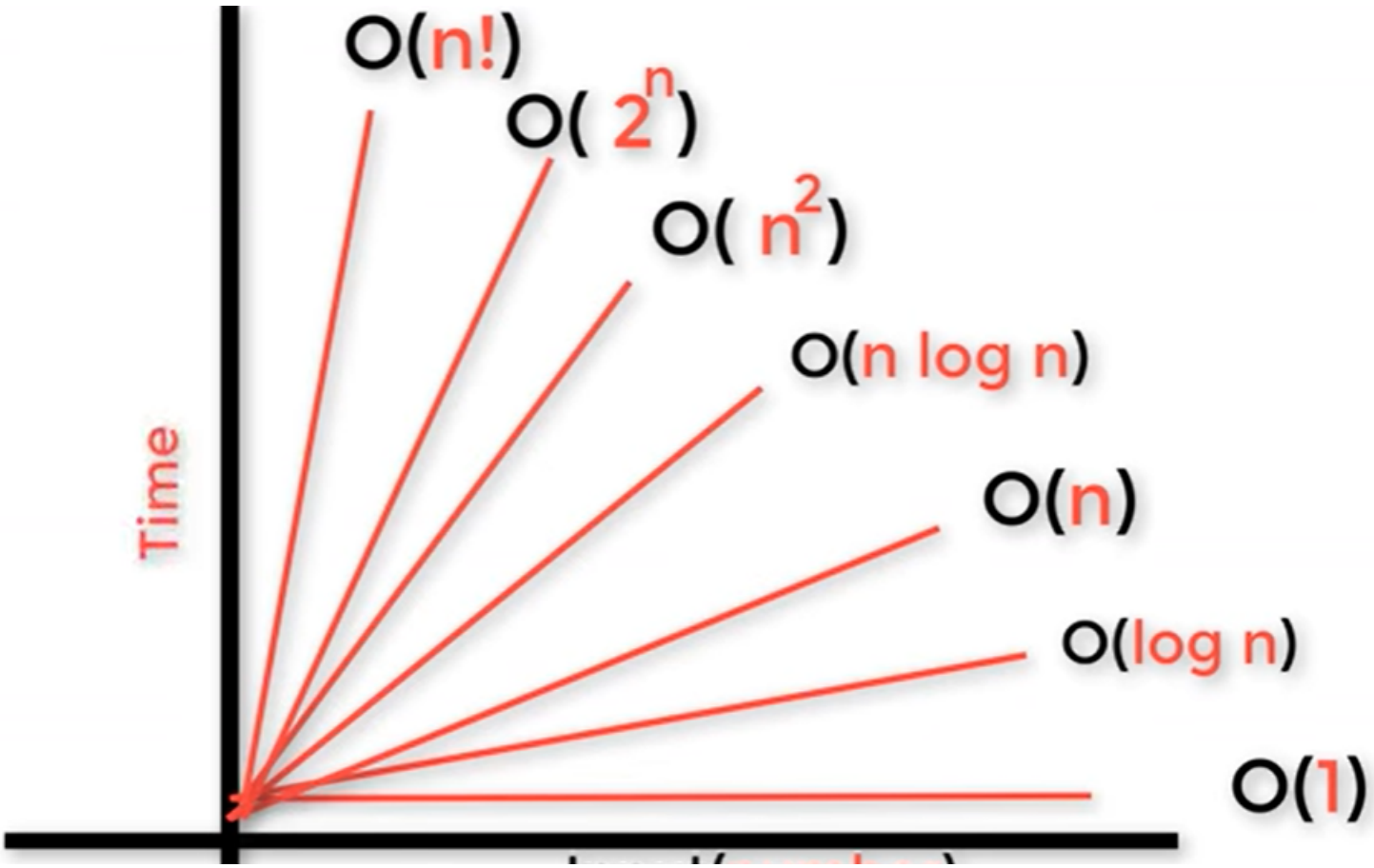
Complexity

best case	Omega Notation, Ω
average case	theta notation, Θ
worst case	Big O notation, O


O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n)
O(1)
Time

* , / , % , ^ , + , -

++ , --

+= , -= , /= , *=

if, else , ifelse

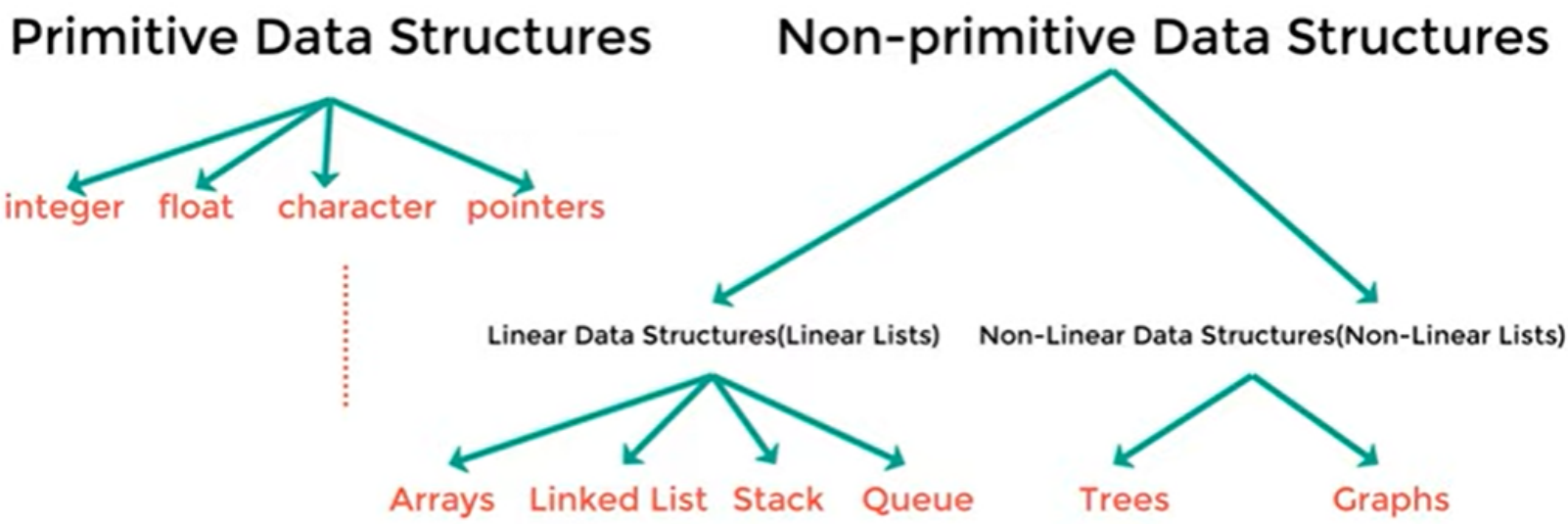
constant time

Complexity

```
sum;
for(i = 1;i<=n;i++)//n
sum=sum+i; //constant time 1
```

Time Complexity : 1 + n = O(n)

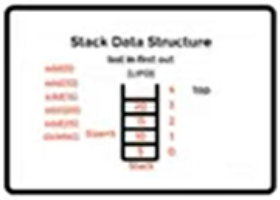
Data Structure



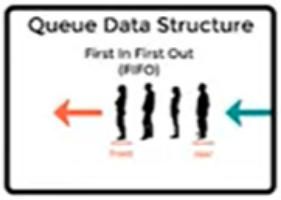
Data Structure

Linear Data Structures(Linear Lists)

Non-Linear Data Structures(Non-Linear Lists)



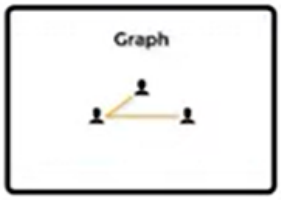
Stack



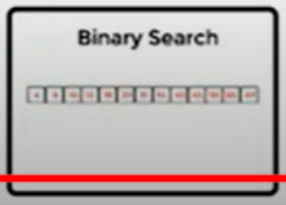
Queue

Linked List

Tree



Graph



Searching

Sorting

Hashing

Stack Data Structure

last in first out

(LIFO)

Array-Based Implementation of the ADT Stack

push(val) pop() getTop() isEmpty()

Expression Evaluation

infix: 5 / 5 + 2 - 1 * 9

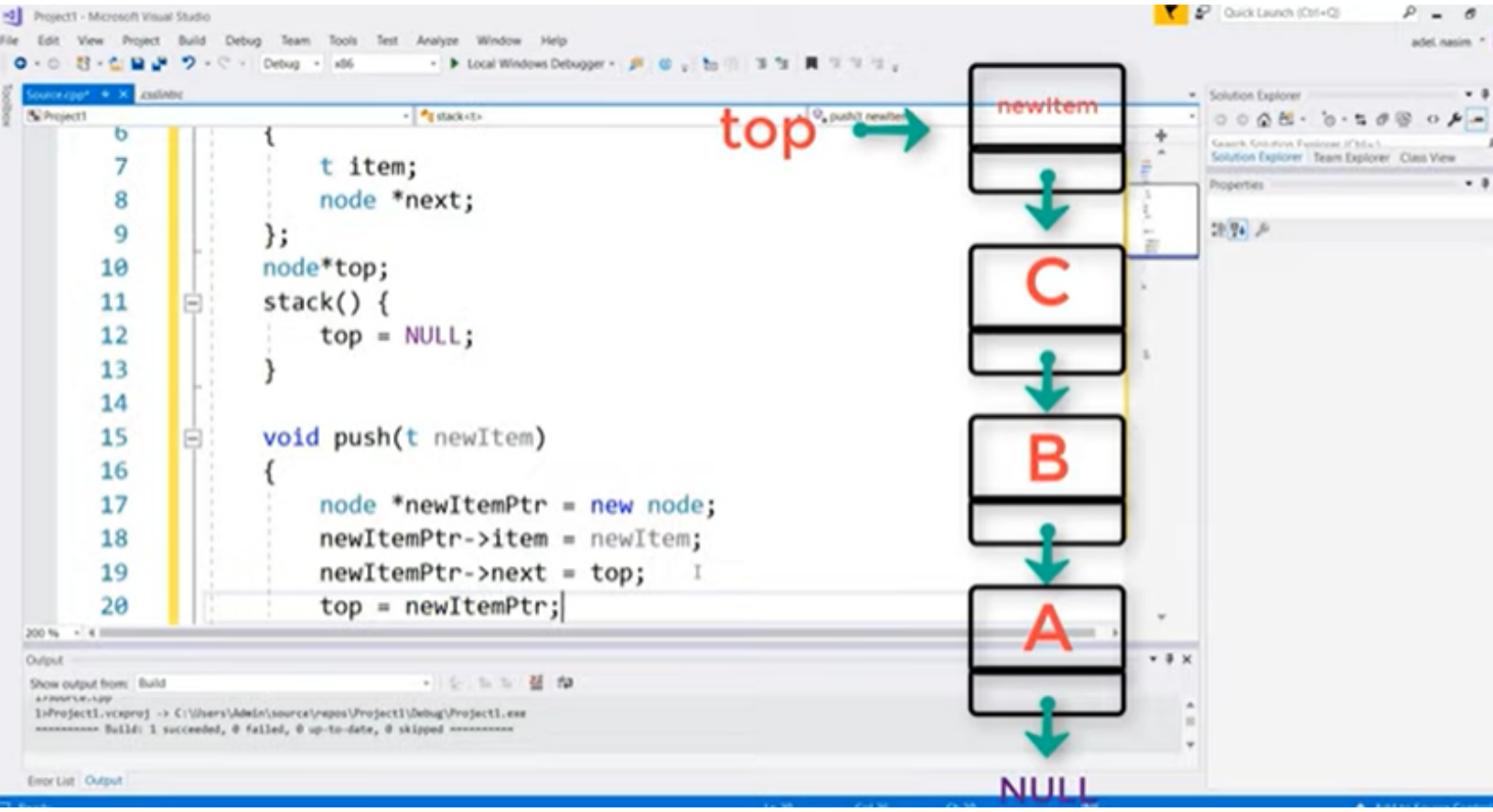
postfix: 5 5 / 2 + 1 9 * -

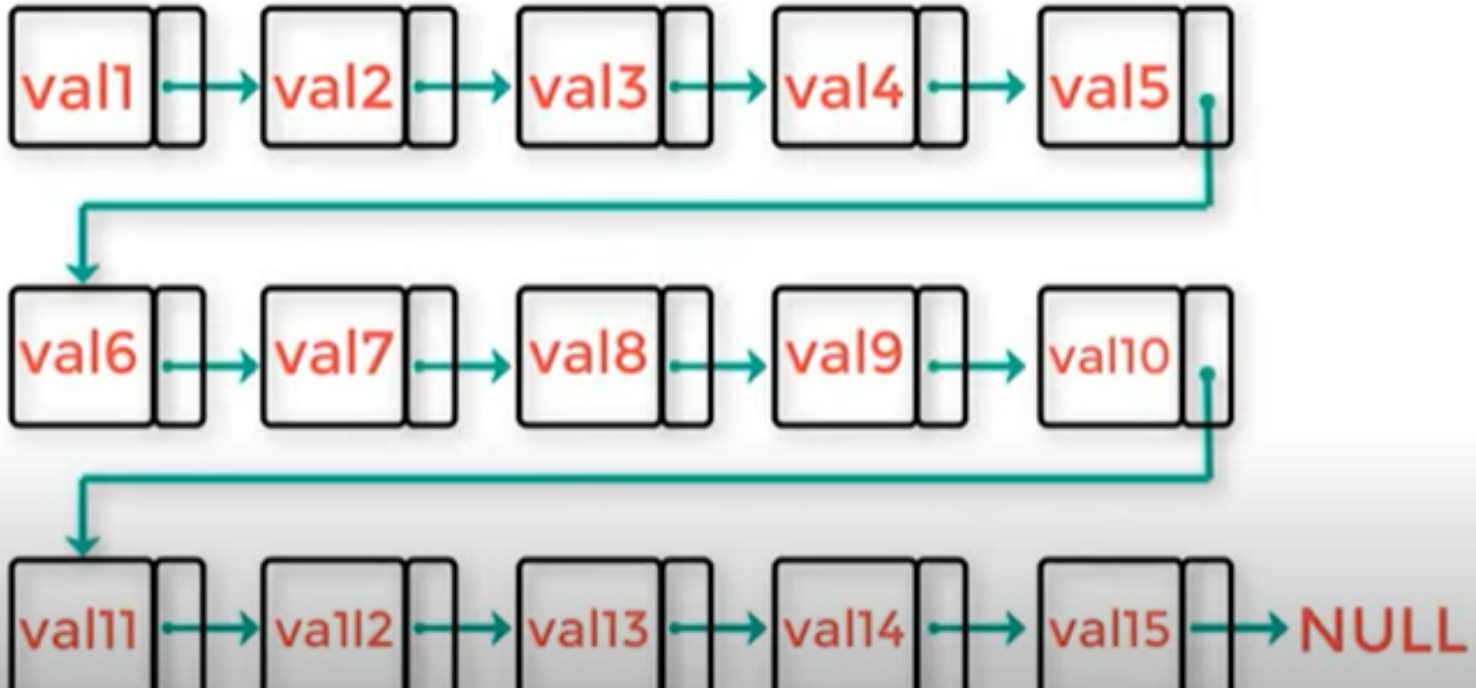
suffix or reverse Polish notation

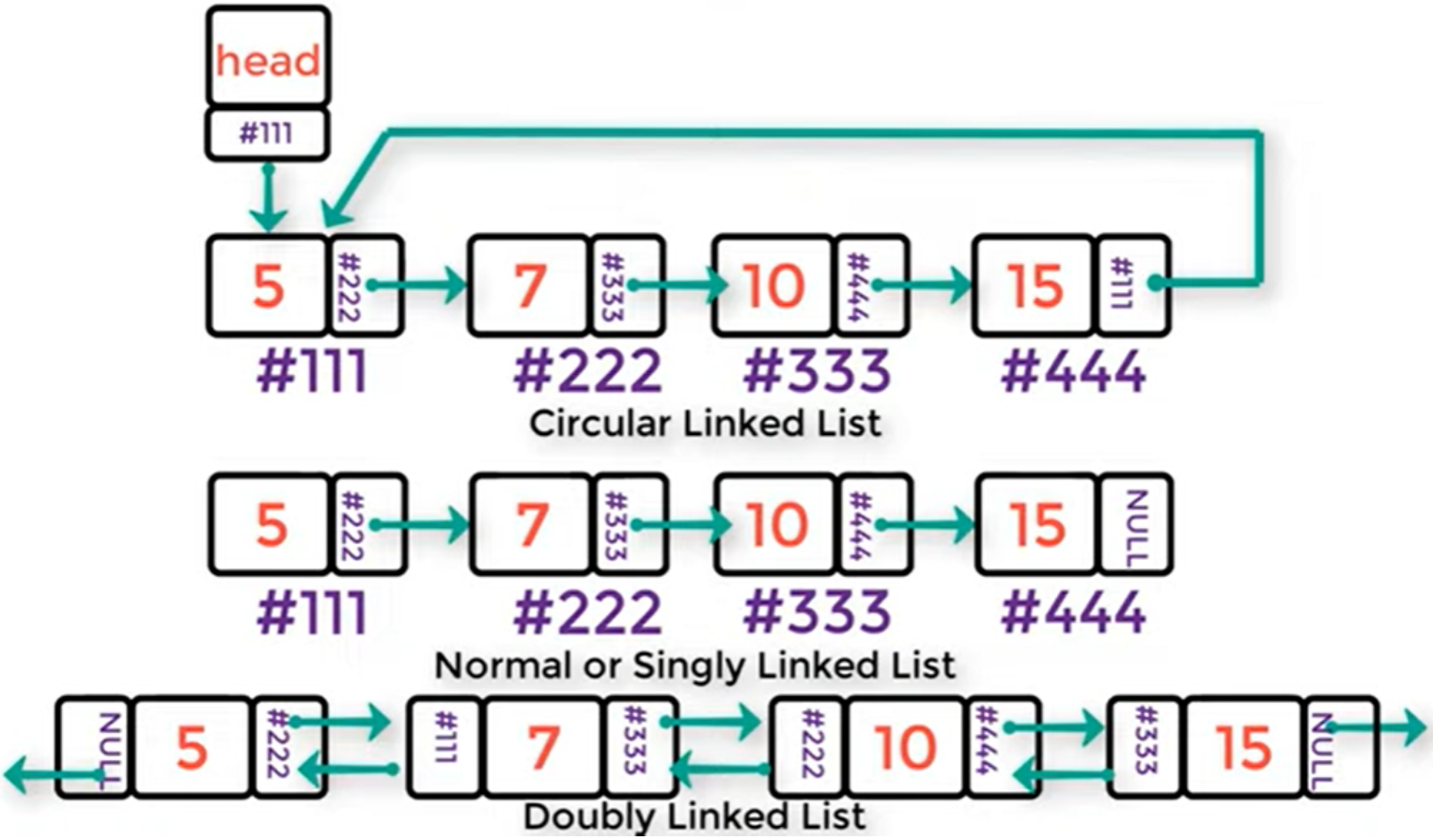
prefix: - + / 5 5 2 * 1 9

Polish notation

Stack


Project1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug x86 Local Windows Debugger
Source.cpp* stack.cs(Mix)
Project1 stack<t> push(t newItem)
{
t item;
node *next;
};
node*top;
stack() {
top = NULL;
}
void push(t newItem)
{
node *newItemPtr = new node;
newItemPtr->item = newItem;
newItemPtr->next = top;
top = newItemPtr;
200 %
Output
Show output from: Build
1>Project1.vcxproj -> C:\Users\Admin\source\repos\Project1\Debug\Project1.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
Error List Output
Solution Explorer
Solution Explorer Team Explorer Class View
Properties
top
newItem
C
B
A
NULL

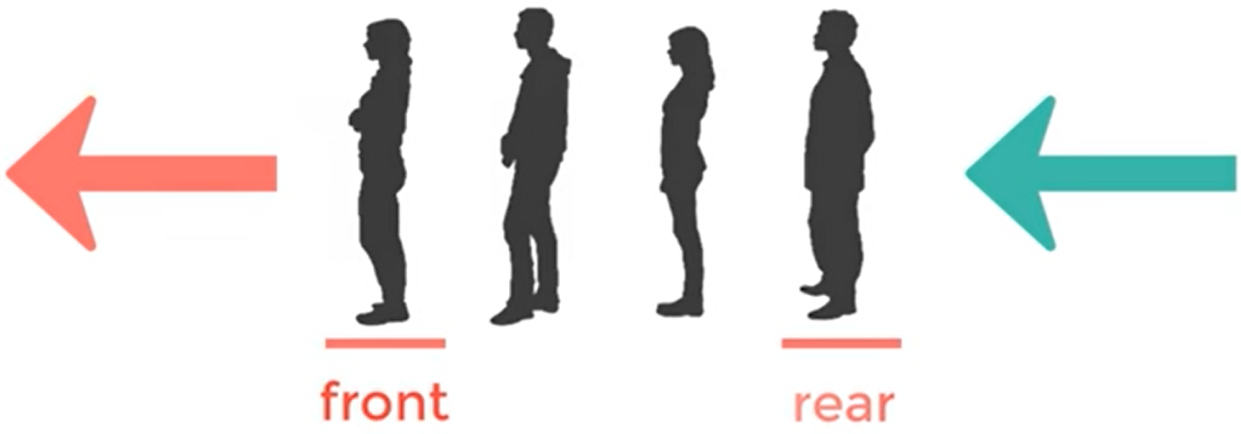

val1
val2
val3
val4
val5
val6
val7
val8
val9
val10
val11
val12
val13
val14
val15
NULL


head
#111
5
#222
7
#333
10
#444
15
#111
#111
#222
#333
#444
Circular Linked List
5
#222
7
#333
10
#444
15
NULL
#111
#222
#333
#444
Normal or Singly Linked List
NULL
5
#222
#111
7
#333
#222
10
#444
#333
15
NULL
Doubly Linked List

```cpp
// Balanced parentheses
bool arepaired (char open , char close ) { // check if exp is paired
    if (open == '(' && close == ')')
        return true ;
    else if (open=='{' && close == '}')
        return true ;
    else if (open == '[' && close == ']')
        return true ;
    return false ;

}
bool arebalanced (string exp ) { // create getting exp function
stack <char> s ;
for (size_t  i =0 ;i<exp.length() ; i++) {
    if(exp[i] == '(' || exp[i] == '{' || exp[i] =='[')
        s.push(exp[i]) ;
    else if (exp[i] == ')' || exp[i] == '}' || exp[i] ==']' ){

        if(s.isempty() ||  !arepaired(s.gettop() , exp[i]) )
            {
            return false   ;
        }else
        s.pop() ;
    }
}return s.isempty()?true : false  ;

}

int main(){
string b ;
cout<<"Enter your exp : " << endl;
cin>>b ;
if (arebalanced(b)){
    cout<<"Balanced"<<endl;
}
else
    cout<<"NotBalanced"<<endl ;
return 0;
}
```

Queue Data Structure

First In First Out
(FIFO)



Queue Data Structure

An Array-Based Implementation of the ADT Queue

isEmpty()

isFull()

enqueue()

dequeue()

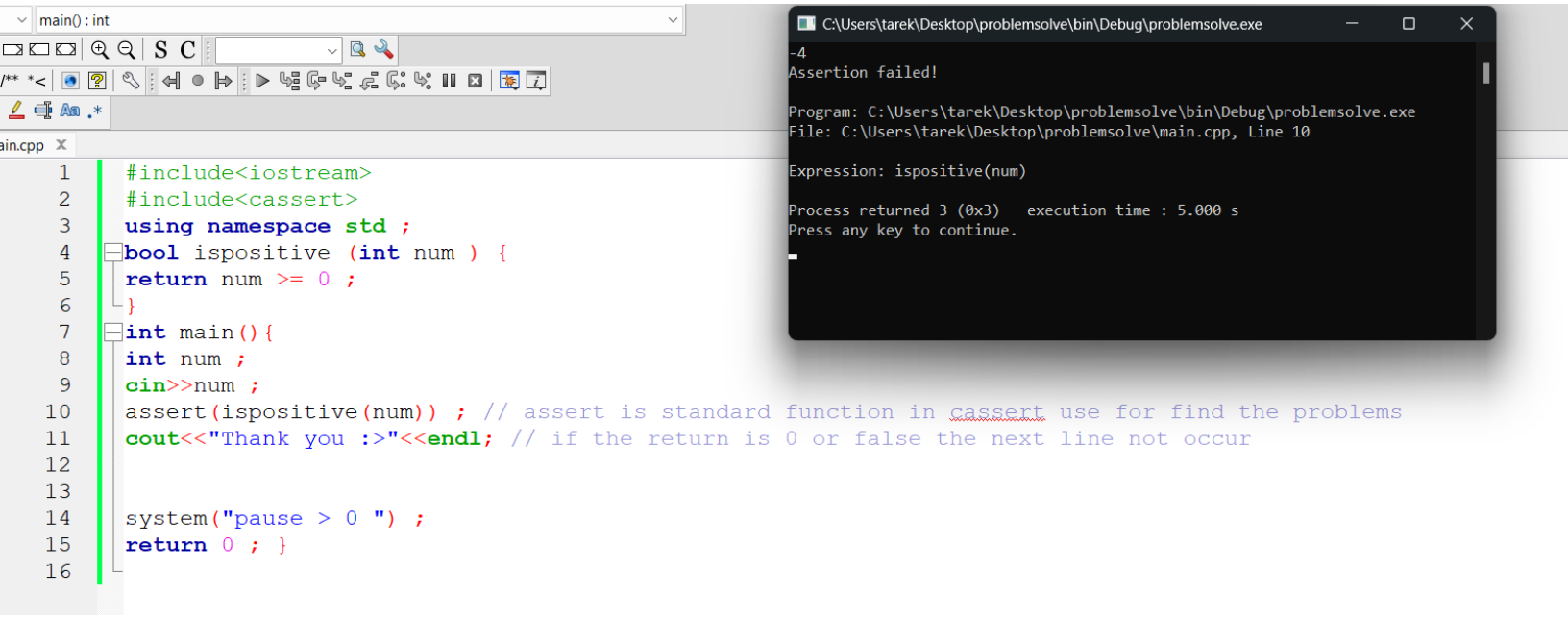
getFront()

gerRear()

getSize()

clear()

```
// size_t is a type definition for unsigned long long
// This means that writing size_t is the excact same as
// writing unsigned long long
```


main() : int
ain.cpp
#include<iostream>
#include<cassert>
using namespace std ;
bool ispositive (int num) {
return num >= 0 ;
}
int main(){
int num ;
cin>>num ;
assert(ispositive(num)) ; // assert is standard function in cassert use for find the problems
cout<<"Thank you :>"<<endl; // if the return is 0 or false the next line not occur
system("pause > 0 ") ;
return 0 ; }
C:\Users\tarek\Desktop\problemsolve\bin\Debug\problemsolve.exe
-4
Assertion failed!
Program: C:\Users\tarek\Desktop\problemsolve\bin\Debug\problemsolve.exe
File: C:\Users\tarek\Desktop\problemsolve\main.cpp, Line 10
Expression: ispositive(num)
Process returned 3 (0x3) execution time : 5.000 s
Press any key to continue.

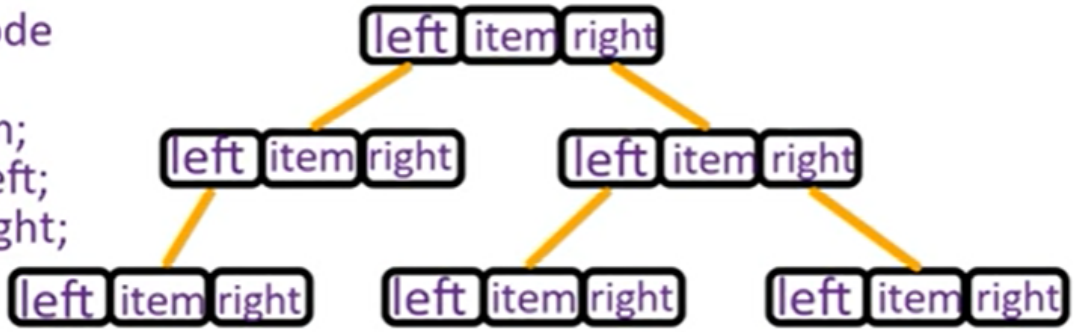
Tree Data Structure

- **Binary Tree**
  "Binary Search Tree" "AVL tree or height balanced binary tree"
- **Binary Space Partition**
- **GGM Tree**
- **Syntax Tree**
- **Huffman Coding Tree**

Binary Tree

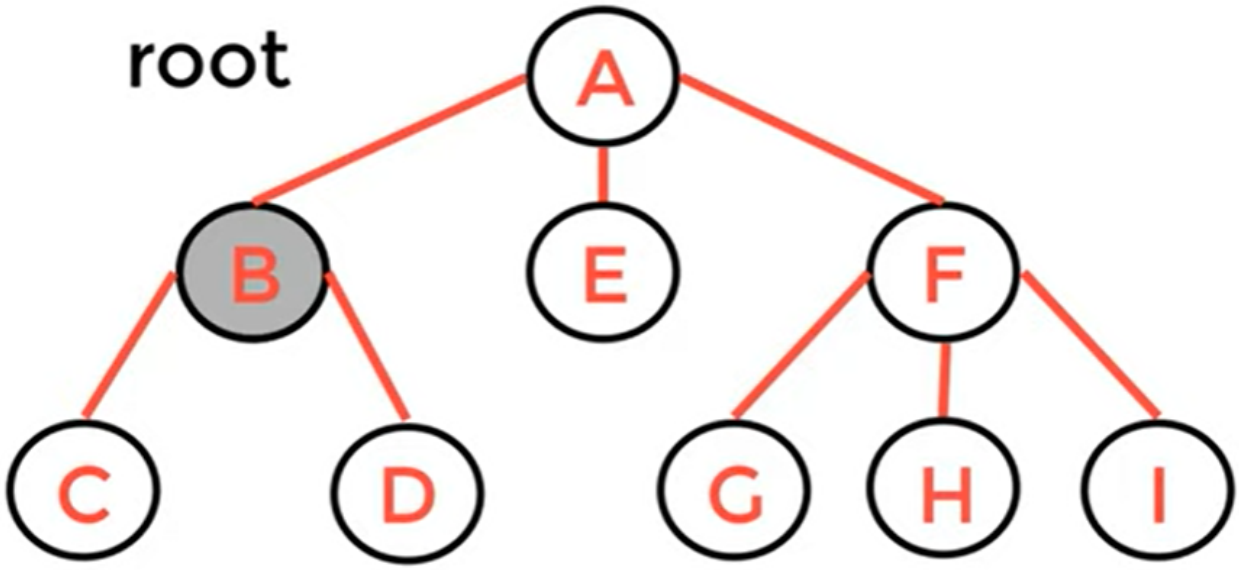
```
struct Node
{
Type item;
Node* Left;
Node* right;
}
```

في لغة سي بلس بلس




```
class Node
{
Type item;
Node Left;
Node right;
}
```

في باقي اللغات



Parents : A,B,F

Children: B,E,F,C,D,G,H,I

Leaves: C,D,E,G,H,I

Internal nodes: B,F

Siblings:{{B,E,F},{C,D}{G,H,I}}

Ancestor:C{A,B} E{A} I{A,F}

Depth(C): 2

Height :

Binary Tree Traversal

Breadth-first traversal:

Level-order

A F G X V Y S M H B Q

Depth-first traversal:

-Pre-order :root left right

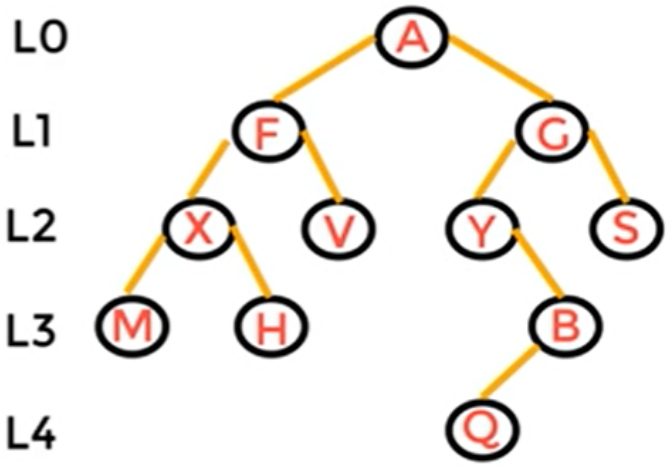
-In-order :left root right

-Post-order :left right root

:root right left

:right root left

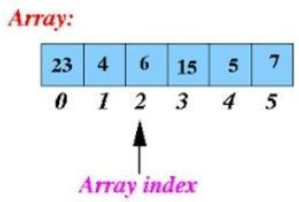
:right left root



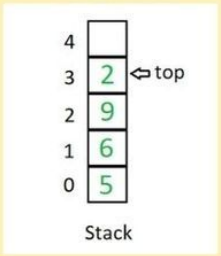
```
int main()
{
    int size;
    cout << "Size: ";
    cin >> size;
    //int myArray[size];
    int* myArray = new int[size];

    for (int i = 0; i < size; i++) {
        cout << "Array[" << i << "] ";
        cin >> myArray[i];
    }

    for (int i = 0; i < size; i++) {
        //cout << myArray[i]<<"  ";
        cout << *(myArray+i)<<"  ";
    }

    delete[]myArray;
    myArray = NULL;

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

DATA STRUCTURE
TYPES OF DATA

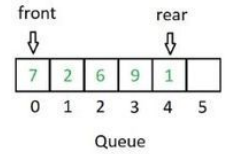
ARRAY



STACK

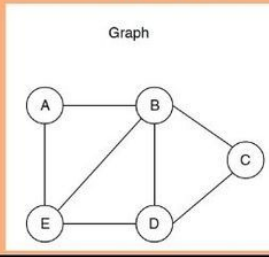


QUEUE

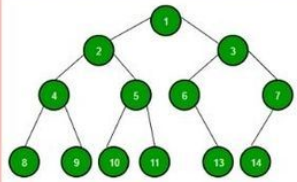


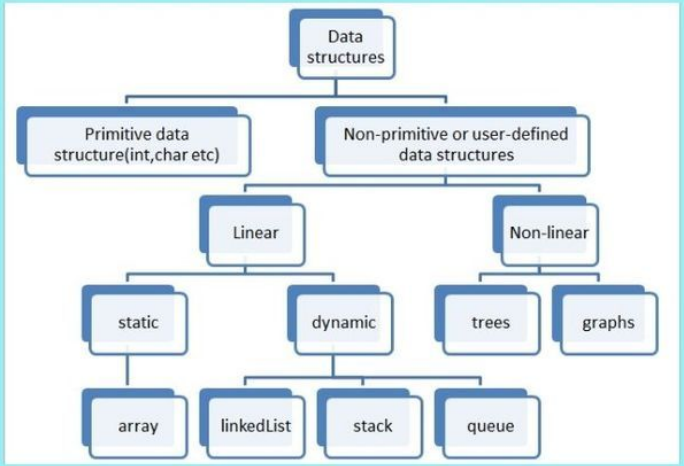
LINKED LIST

GRAPHS



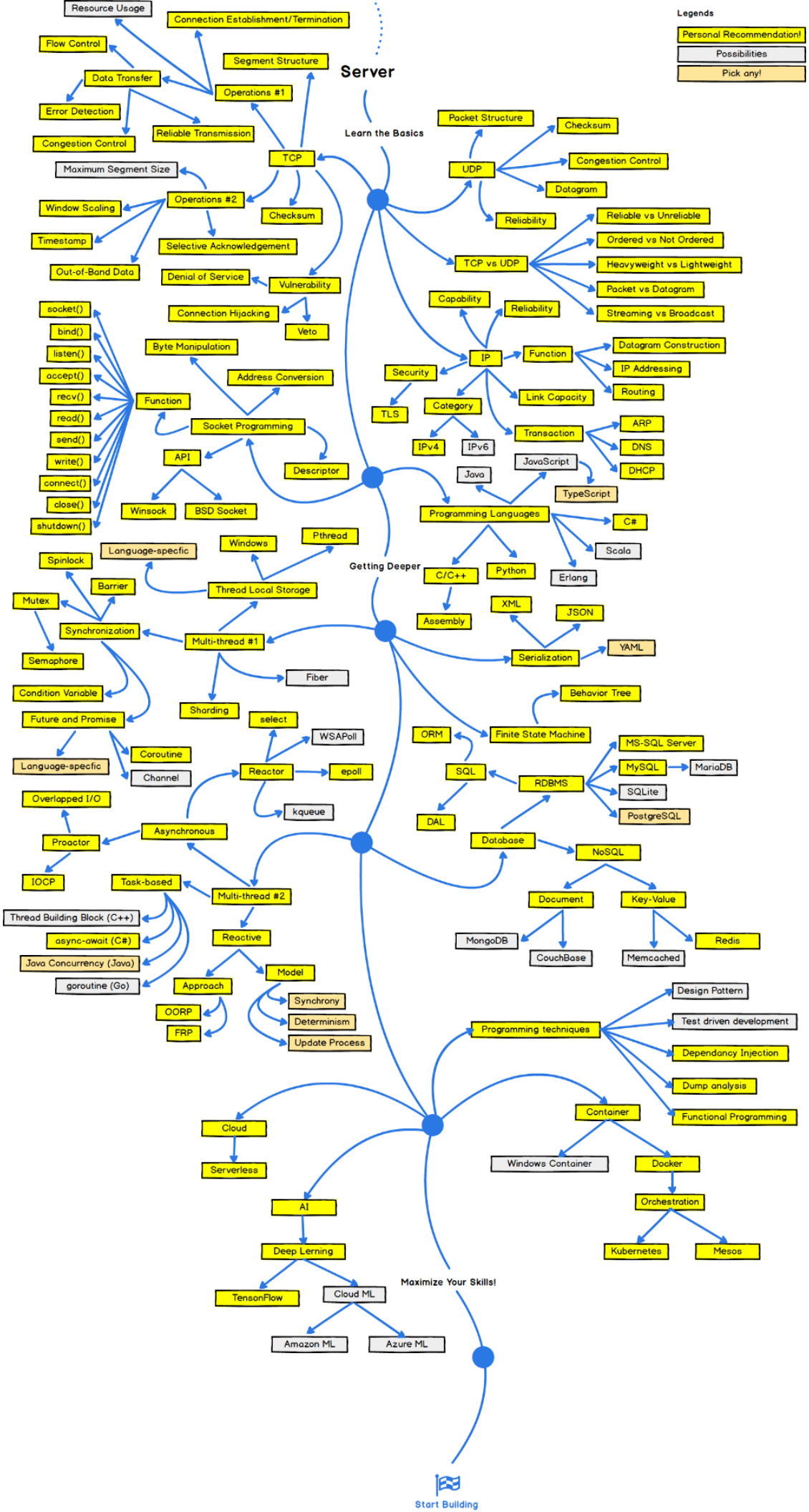
TREES





Learnbay

https://www.learnbay.co/data-science-course/


Legends
Personal Recommendation!
Possibilities
Pick any!
Server
Learn the Basics
Getting Deeper
Maximize Your Skills!
Start Building
Resource Usage
Connection Establishment/Termination
Flow Control
Segment Structure
Data Transfer
Operations #1
Error Detection
Reliable Transmission
Congestion Control
TCP
Maximum Segment Size
Operations #2
Window Scaling
Checksum
Timestamp
Selective Acknowledgement
Out-of-Band Data
Denial of Service
Vulnerability
Connection Hijacking
Veto
Packet Structure
Checksum
UDP
Congestion Control
Datagram
Reliability
TCP vs UDP
Reliable vs Unreliable
Ordered vs Not Ordered
Heavyweight vs Lightweight
Packet vs Datagram
Streaming vs Broadcast
Capability
Reliability
IP
Function
Datagram Construction
IP Addressing
Routing
Security
Link Capacity
TLS
Category
IPv4
IPv6
Transaction
ARP
DNS
DHCP
socket()
bind()
listen()
accept()
recv()
read()
send()
write()
connect()
close()
shutdown()
Function
Byte Manipulation
Address Conversion
Socket Programming
API
Winsock
BSD Socket
Descriptor
Java
JavaScript
TypeScript
Programming Languages
C#
Scala
Erlang
Python
C/C++
Assembly
Windows
Pthread
Language-specfic
Spinlock
Barrier
Thread Local Storage
Mutex
Synchronization
Multi-thread #1
Semaphore
Condition Variable
Future and Promise
Language-specfic
Coroutine
Channel
Fiber
Sharding
select
WSAPoll
Reactor
epoll
kqueue
Overlapped I/O
Proactor
IOCP
Asynchronous
Task-based
Multi-thread #2
Thread Building Block (C++)
async-await (C#)
Java Concurrency (Java)
goroutine (Go)
Reactive
Approach
OORP
FRP
Model
Synchrony
Determinism
Update Process
XML
JSON
Serialization
YAML
Behavior Tree
Finite State Machine
ORM
SQL
RDBMS
MS-SQL Server
MySQL
MariaDB
SQLite
PostgreSQL
DAL
Database
NoSQL
Document
Key-Value
MongoDB
CouchBase
Memcached
Redis
Programming techniques
Design Pattern
Test driven development
Dependancy Injection
Dump analysis
Functional Programming
Container
Windows Container
Docker
Orchestration
Kubernetes
Mesos
Cloud
Serverless
AI
Deep Lerning
TensonFlow
Cloud ML
Amazon ML
Azure ML

2022 Data Analyst RoadMap