11, April, 2022

DATE

Stacks & Queues

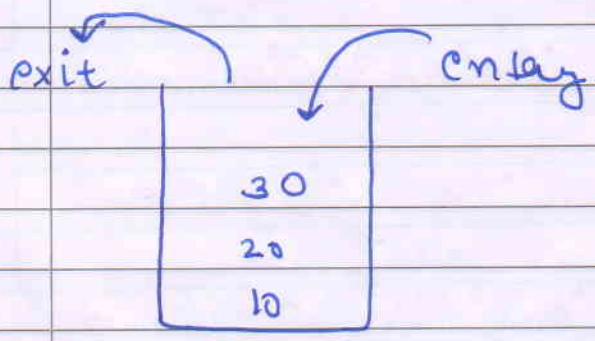
STACK ① → Recursion is based upon it.

② Abstract datatype

Revision

1) Introduction
2) Implementation → ARRAY
   → LinkedList
3) INBUILD STACK
4) Dynamic stack
5) Templates

Revision → 30/05/22



1) Insert → push() → push(10)

2) Delete → pop() → pop()

3) Access top Most Element
   ↓
   top() → 10

4) size() → 1

5) is empty() → Bool

PAGE

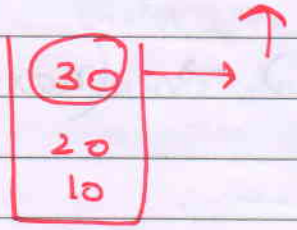
DATE

Stack using array →

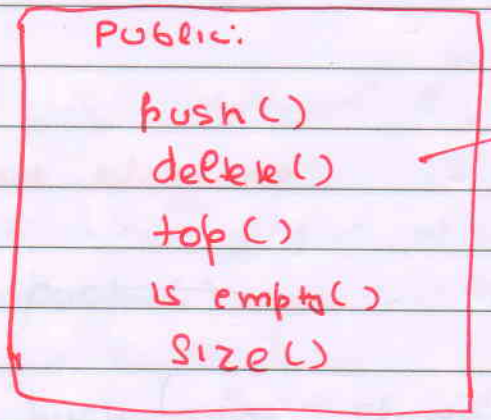
Stack → L I F O

↓ ↓ ↓

Last In First Out



Stack Can be implimented using
- ↳ ARRAYS
- ↳ linked list

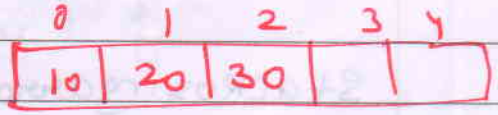


→ STACK CLASS

push (10);

push (20);

push (30)



next index

~~0~~

~~1~~

~~2~~

3

DATE

- top(); → 30 → [next index - 1]
- pop(); → [next index - -]
- size() → (next index)

is empty() → (next index == 0) else
↓ true ↓ false

Stacksusingarray.cpp

```
Class StackUSing Array {
int * data ;
int nextindex ;
int Capacity ;
Public :
// initliasation with Size
Stackusingarray (int totalsize) {
data = new int [totalsize] ;
next index =0 ;
}
```

classmate

PAGE

DATE

- // to see the size of Stack

```
int size() {
    return nextIndex;
}
```

- // Check weather Stack is empty or not

```
bool isempty() {
if ( nextIndex == 0) return true;
else return false;
```

or

```
return nextIndex == 0;
}
```

- // Insert element

~~Void push (t)~~

```
Void push ( int element ) {
  if ( nextIndex == Capacity) {
     Cout << " Stack full " << endl;
     return;
  }
  data [next Index] = Element;
  next Index ++;
}
```

PAGE

DATE

```
// Delete Element
int pop () {
if ( isempty ()) {
  Cout << "Stack is empty" << endl;
  return Int_MIN;
}

nextIndex --;
return data [nextIndex];

}
// Display the value at top of stack.
int top () {
if ( isempty ()) {
  Cout << "Stack is empty" << endl;
  return INT_MIN;
}
return data [nextIndex - 1];

}
```

DATE

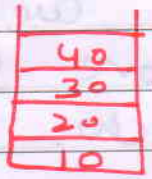
```
#include <iostream>
using namespace std;
#include "StackUsingArray.cpp"

int main() {

    StackUsingArray S(4);
}
```

Constructor called that we made

```
    S.push(10);
    S.push(20);
    S.push(30);
    S.push(40);
    S.push(50);
```

→ 

— X "Stack FULL"

```
    cout << S.top() << endl;
```
→ 40

```
    cout << S.pop() << endl;
    cout << S.pop() << endl;
    cout << S.pop() << endl;
```
40
30
20

```
    cout << S.size() << endl;
```
→ 1

```
    cout << S.isEmpty() << endl;
```
→ false

Output →

```
Stack FULL
40
40
30
20
1
0
```


PAGE

12, April, 2022.

DATE

DYNAMIC STACK

Note → Previously, we saw that we were having constraints on length/size of ARRAY once the size is initialized we can't increase it

Now we will see how we can remove size constraints using Dynamic Approach.

- Stack usig array ( ) {

```
data = new int [ 4 ] ; // Decided by us
next index = 0 ;
Capacity = 4 ; // Decided by us.
}
```

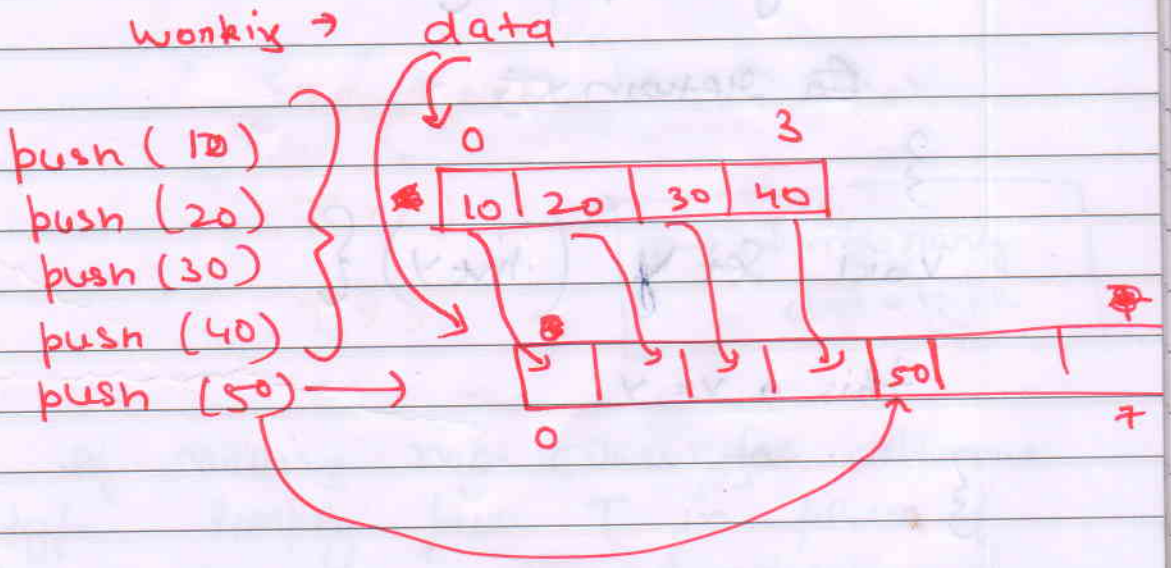
- void push (int element) {

```
if (next index == Capacity)
{
    // New data made of Doubled size
    int * newdata = new int [2 * Capacity]
```

DATE

// Copying values of original data to new data

```
for ( int i = 0; i < capacity ; i++ ) {
    newdata [i] = data [i] ;
}
capacity * = 2 ; // twiced the capacity
delete [] data ;
data = newdata ;
}
data [ nextIndex ] = element ;
nextIndex ++ ;
}
```



DATE

TEMPLATES

Suppose we made a pair class →

```
Class pair {

int x;
int y;

Public :

void setx ( int x ) {

   this → x = x;

}

int getx ( ) {

   return x;

}

void sety ( int y ) {

  this → y = y

}

void gety ( ) {

   return y;

} classmate

};
```

PAGE

DATE

So previous Class will only work with integers

What if we want that it should work with float, double

① we can write whole class again for double variable

② USE TEMPLATES

Pair double	pair Character
double x	Char x ;
double x	Char y ;

TEMPLATE

T x ;

T y ;

T → temporary data type

Instead of Making new Class for different datatype simply put T in place of datatype

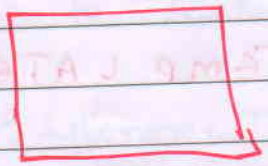
```
template < typename T >
```

DATE

```
template <typenameT>
Class pair {
Public:
Void SetX ( T x ) {
  this → X = X ;
}
Int Get x ( ) {
  return x
}
Void Set y ( T y ) {
 this→ y = y ;
}
Int get y ( ) {
  return y ;
}
};
```

DATE

pair p1 ;



We need to Specify 'T' at time of initialization as shown

```
#include<iostream>
Using namespace std;
#include "pair.cpp";

Int main() {

    Pair < int > P1 ;

    Pair < double > P2 ;

    P1.setx (10) ;
    P1.setx (20) ;

Cout << p1.getx() << " " << p1.gety() <<en

    P2.setx (100.34) ;
    P2.sety (34.21) ;

Cout << p2.getx() << " " p2.gety() <<en

Pair < Char > P3;

}
```

DATE

So, previously,

We learnt TEMPLATE

We said we can change "type" of x and y

What if I want both variable belong to diff data

y → Double

x → Integer

→

template < typename T , typename V >

class pair {

T x ;

V y ;

}

main →

pair < int , double > p1 ;

p1. setx (100) ; → Integer

p1. sety (100.32) ; → Double

classmate

PAGE

DATE

What if we want to Make triplet

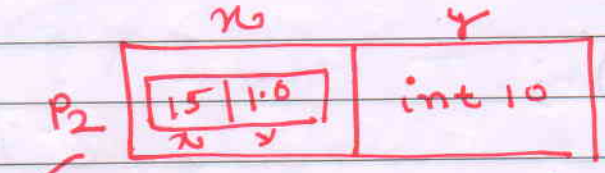
```
template <typename T, typename V, typename C>
```

```
pair <int, float, double> p1;
```

OR

```
pair <pair <int, int>, int> p2
```

T V




```
p2.setY(10)
```

$P_2(x)$ accepts a pair so we need to create a pair

```
pair <int, int> p4
```

```
p4.setX(15); p4.setY(10)
```

```
p2.setX(p4);
```

DATE

To get the values

$P_2.getY() \rightarrow 10;$

$P_2.getX().getX() \rightarrow 15;$

$P_2.getX().getY() \rightarrow 16;$

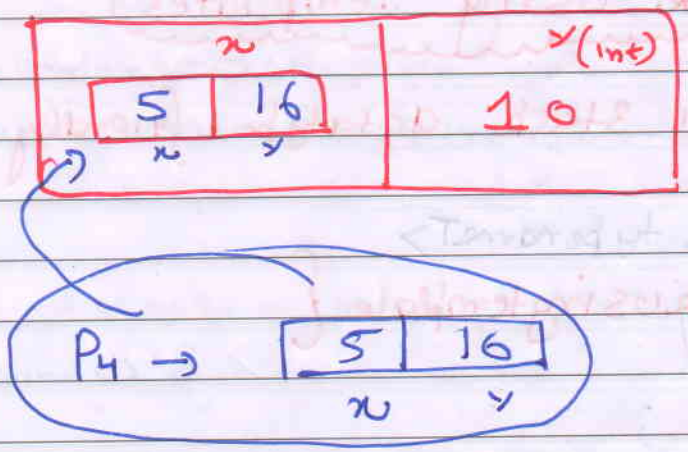
```
int Main(){
Pair<Pair<int,int>,int> P2;
P2.setY(10);
Pair<int,int> P4;
P4.setX(5);
P4.setY(16);
P2.setX(P4);
Cout<<P2.getX().getX()<<" "<<P2.getX().getY()
<<P2.getY();
```

Output → 5, 16, 10

classmate

PAGE

DATE



DATE

Stack using Templates

Making our stack datatype friendly

```
template <typename T>
Class StackUsingTemplate {

T * data ;

int nextIndex ;

int Capacity ;

Public :

  Void push ( T Element ) {

    if ( nextIndex == Capacity ) {

    T * newdata = new T [2 x Capacity]

    - - - -

    - - -

  }

  T Pop ( ) {

    if ( nextIndex == 0 ) {

      Cout << "Stack is empty"
```

```
    nextIndex--;
    return data[nextIndex];
}

int main() {

    StackUsingARRAY<int> S;

    StackUsingARRAY<char> C;

}
```

DATE

13, April, 2022

Stack using Linked List

DRY Approach →

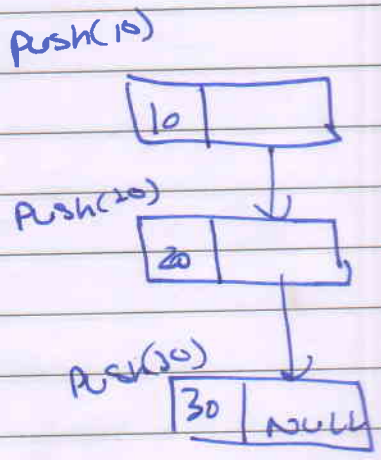
Class Stack using Linked list {

Node * head ;

	In ARRAY	In LL
push	→ O(1)	O(1)
pop	→ O(1)	O(1) or O(n)
top	→ O(1)	O(1)
Size	→ O(1)	O(1)
Isempty	→ O(1)	O(1)

Stack {

Node * head ;
int Size ;
Stack() {
head = NULL ;
tail = NULL ;
}



DATE

top() {

Return tail → data ;

pop() → 30

↳ we need to Maintain prev and tail will go to previous

So its only possible if we travel to whole LL

h ——————

10 → 20 → 30 → ~~40~~ → NULL

tail t

h ——————

10 → 20 → ~~30~~

tail

pop() → tail = temp ;

pop → tail = temp ;

---

head → null

push(10) ; [10 | NULL] → Head

So far Head → [10 | NULL]

Push (20)

Head → [20 |] → [10 | NULL]

inserting at head.

classmate

PAGE

DATE

Code →

```
class Node {

Public :

int data ;
Node * next ;

Node ( int data) {

this → data = data ;

this → next = NULL ;

}

};

class Stack {

Node * head ;
int size ;    // no of elements present in stack

Public :

Stack() {

    head = NULL ;
} size = 0 ;
```

PAGE

DATE

```
int getsize() {
    return size; // this will return size.
}

bool isempty() {
    return size == 0; // True/False depending upon Size is empty or not

void push(int Element) {
    Node * newnode = new node(Element);
    newnode -> next = head;
    head = newnode;
    size++;
}

int pop() {
    if (isempty()) return 0;
    int ans = head -> data;
    Node * temp = head;
    head = head -> next;
    delete temp;
    size--;
    return ans;
```


PAGE

DATE

```
int top() {
  if (isempty()) {
    return 0;
  }
  return head -> data;
}
```

```
#include <iostream>
using namespace std;
int main() {
  Stack S;
  S.push(10);
  S.push(20);
  S.push(30);
  S.push(40);
  Cout << S.top() << endl;
  Cout << S.pop() << endl;
  Cout << S.pop() << endl;
  Cout << S.pop() << endl;
```

DATE

```
Cout << S.getSize() << endl;
Cout << S.isempty() << endl;
```

Inbuied STack

```
# include <Stack> //<bits/G++.h>

Int Main() {
// Creation of Stack
Stack <int> S;
// Push operation
S.push(2);
S.push(3);
// pop operation
S.pop()
Cout << S.size << endl;
Cout << "Element at top" <<S.top() << endl
if (S.empty()) {
Cout << "Stack is empty" << endl;
}
```

push()
pop()
isempty()
size()
top()

↑
STACK