**Lab #2**

**Implementation of Stack**

**Stack ADT(Abstract Data type) Operations**

Initialize -- Sets stack to an empty state.

IsEmpty -- Determines whether the stack is currently empty. IsFull -- Determines whether the stack is currently full.

Push (ItemType newItem) -- Adds newItem to the top of the stack.

Pop (ItemType& item) -- Removes the item at the top of the stack and returns it in item.

### Implementation of Stack Using Static Array

### **Stack implementation using a static array** refers to implementing a stack with a fixed-size array where the maximum number of elements that can be stored is predetermined. This is a common approach where the size of the array is specified during its declaration, and no dynamic memory allocation is used.

//

// SPECIFICATION FILE (stack.h)

//

#define MAX\_ITEMS 100 **//** This defines a constant MAX\_ITEMS with a value of 100, which will be used to specify the maximum number of items in the stack.

**typedef** int ItemType; // This creates an alias ItemType for the int type. The stack will store items of type ItemType, which in this case are integers.

**class Stack** { public:

Stack ( ); // Default constructor.

int IsEmpty( ) const; int IsFull( ) const;

void Push( ItemType newItem );

void Pop( ItemType& item ); // item is a copy of removed element.

private:

int top;

ItemType items[MAX\_ITEMS]; // An array to store the stack elements, where the maximum size is 100.

};

//

// IMPLEMENTATION FILE (stack.cpp)

//

// Private data members of class:

// int top;

// ItemType items[MAX\_ITEMS];

//

#include “stack.h”

Stack::Stack ( ) // This is the default constructor for the Stack class. It initializes the top variable to -1, indicating that the stack is initially empty.

{

top = -1;

}

// int Stack::IsEmpty( ) const

{

return ( top == -1 ); // This function returns 1 (or true) if the top is -1, meaning the stack is empty. Otherwise, it returns 0 (or false).

}

// int Stack::IsFull( ) const

{

return ( top == MAX\_ITEMS-1 ); // This function checks if the stack is full by comparing top with MAX\_ITEMS - 1. If top equals 99, the stack is full because there are 100 elements (from index 0 to 99).

}

// void Stack::Push ( ItemType newItem )

{

if (IsFull())

{

cout << “Stack Overflow” << endl;

exit(1);

}

top++;

items[top] = newItem;

}

// This function pushes a new item onto the stack:

1. It first checks if the stack is full using the IsFull() function. If it is, it prints an error message ("Stack Overflow") and exits the program.
2. Otherwise, it increments top and assigns the newItem to the top of the stack.

void Stack::Pop ( ItemType& item )

{

if (IsEmpty())

{

cout << “Stack Underflow” << endl; exit(1);

}

item = items[top]; top--;

}

//This function pops an item from the stack:

1. It first checks if the stack is empty using the IsEmpty() function. If it is, it prints an error message ("Stack Underflow") and exits the program.
2. If the stack is not empty, the top item is assigned to the reference variable item (passed by reference), and top is decremented to effectively remove the item.

// DRIVER FILE (driver.cpp)

//

#include <iostream> #include <stdlib.h> #include “stack.cpp”

using namespace std; int main()

{

Stack s; // Create a Stack object

int item;

//Push 20 items onto the stack

for (int i = 0; i < 20; i++) s.Push(i);

// Pop all 20 items from the stack

for (i = 0; i < 20; i++)

{ s.Pop(item);

cout << item << endl;

}

return 0;

}

//**Purpose of the Driver Program:**

* A stack object s is created.
* A loop runs 20 times, pushing integers (0 to 19) onto the stack.
* Another loop runs 20 times, popping the items from the stack and printing them.

# Dynamic Implementation of Stack

### ****A** dynamic implementation of a stackrefers to creating a stack data structure where the size of the stack is not fixed. Unlike a static implementation (which uses a fixed-size array), the dynamic implementation allows the stack to grow and shrink as needed during runtime. This is typically done using dynamic memory allocation, such as with pointers and data structures like linked lists or dynamically allocated arrays.**

### Stack Using Class Template and Dynamic Array

* The construct that allows us to create a class of undetermined type is called a **template**.
* A class template allows the compiler to generate multiple versions of a class type by using type parameters.
* The formal parameter appears in the class template definition, and the actual parameter appears in the client code. Both are enclosed in pointed brackets, < >.

template<class ItemType>

##### class Stack {

public:

Stack ( );

Stack ( int max ); // PARAMETERIZED CONSTRUCTOR

~Stack ( ) ; // DESTRUCTOR . . .

int IsEmpty( ) const; int IsFull( ) const;

void Push( ItemType newItem ); void Pop( ItemType& item );

private:

int top;

int maxStack;

ItemType\* items; // DYNAMIC ARRAY IMPLEMENTATION

};

//

// CLASS TEMPLATE IMPLEMENTATION FILE (stack.cpp)

//

#include “stack.h”

template<class ItemType>

Stack<ItemType>::Stack( ) //DEFAULT CONSTRUCTOR

{

maxStack = 500;//Stack can holds value of 500

top = -1;

items = new ItemType[500]; // dynamically allocates array

}

template<class ItemType>

Stack<ItemType>::Stack( int max ) // PARAMETERIZED

{

maxStack = max; // maxStack is a private data member of the Stack class that stores the maximum capacity of the stack.

top = -1;// The top variable is initialized to -1.The top keeps track of the current position of the top element in the stack.Setting top = -1 indicates that the stack is empty initially, since there are no elements in the stack yet. The first element added to the stack will move top to 0.

items = new ItemType[max]; // dynamically allocates array

}

template<class ItemType> Stack<ItemType>::~Stack( )

{

delete [ ] items; // deallocates array

}

template<class ItemType>

int Stack<ItemType>::IsEmpty( )

{

return (top == - 1); // (top == maxStack - 1): This is the condition that checks whether the stack is full.top == maxStack - 1 means the top index is equal to the last valid index in the stack. For example, if maxStack = 10, the valid indices for the stack array would be 0 to 9, so maxStack - 1 is 9.

}

template<class ItemType>

int Stack<ItemType>::IsFull( )

{

return (top == maxStack - 1); // The function IsFull() checks whether the stack has reached its maximum capacity by comparing the top index with maxStack - 1.It returns 1 if the stack is full and 0 if it’s not.

}

template <class ItemType>

void Stack<ItemType>::Push (ItemType newItem )

{

if (IsFull())

{ cout << “Stack Overflow” << endl; exit(1);

}

top++;

items[top] = newItem;

} //The Push function checks if the stack is full. If it's not full, it adds a new item to the stack at the next available position. If the stack is full, it triggers a "Stack Overflow" error and exits the program.

template<class ItemType>

void Stack<ItemType>::Pop (ItemType& item )

{

if (IsEmpty())

{ cout << “Stack Underflow” << endl; exit(1);

}

item = items[top]; top--;

}

//The Pop function removes the top item from the stack. It checks if the stack is empty before proceeding. If it is, a "Stack Underflow" error is triggered, and the program exits. If not, it retrieves the top item and decreases the top index.

// Driver Program Using Class Template

#include <iostream>

#include "stack.cpp"

using namespace std;

int main() {

Stack<int> IntStack; // Create a stack for integers

Stack<float> FloatStack; // Create a stack for floats

int data;

float val;

IntStack.Push(35); // Push an integer (35) onto the integer stack

FloatStack.Push(3.1415927); // Push a float (3.1415927) onto the float stack

IntStack.Pop(data); // Pop the top element from the integer stack (35) into 'data'

cout << data << endl; // Print the popped value (35)

FloatStack.Pop(val); // Pop the top element from the float stack (3.1415927) into 'val'

cout << val << endl; // Print the popped value (3.1415927)

return 0;

}

### Output:

The output of the program will be:

35

3.14159

**Exercise 2**

Use the Stack class to solve the following problems:

##### Dynamic Implementation of Stack

Write a program to implement stack dynamically.

##### Finding maximum and minimum using Templates.