| Activity No. 4.1 | |
|---|---|
| **Stacks** | |
| **Course Code:** CPE010 | **Program:** Computer Engineering |
| **Course Title:** Data Structures and Algorithms | **Date Performed: Aug. 26, 2025** |
| **Section: CPE21S4** | **Date Submitted: Aug. 26, 2025** |
| **Name(s): Crishen Luper S. Pulgado** | **Instructor: Engr. Jimlord Quejado** |

**6. Output**

ILO A:

```
#include<stack>

int main(){
    std::stack<int> newStack;
```

This header contains the stack to access the stack container from STL. In the main function, newStack is declared as a stack that has integer value.

```
newStack.push(3);
newStack.push(8);
newStack.push(15);
```

Since we declared the stack, next is to put a stack which is 3, 8, and 15 in order following the LIFO (15 at the top and 3 is at the bottom).

```
std::cout << "Stack Empty? " << newStack.empty() << std::endl;
std::cout << "Stack size: " << newStack.size() <<   std::endl;
std::cout << "Top element of the stack: " << newStack.top() << std::endl;
newStack.pop();
std::cout << "Top element of the stack: " << newStack.top() << std::endl;
std::cout << "Stack size: " << newStack.size() << std::endl;
return 0;
```

The next code block is it checks if the container is empty using the function empty. We can also determine the size of the container using the size function. Then, we print what is the top element in the stack, which is 15, then we remove the top element using pop function. We print again what is the current top element and the size of the container, which is now 8 is the top and the size is two.

```
PS Z:\Crishen\cpp> & 'c:\Use
e-Out-14e2c1ym.2b2' '--stderr
Stack Empty? 0
Stack size: 3
Top element of the stack: 15
Top element of the stack: 8
Stack size: 2
```

ILO B:

```
const size_t maxCap = 100;
int stack[maxCap];
int top = -1, i, newData;

void push();
void pop();
void Top();
bool isEmpty();
void display();
```

First, declaration of maxCap which is the maximum capacity that can store is 100. Then we put it on a array which is the stack. We indicate if the top is -1 it means empty and I is the chosen size of stack by the user and newData is an input before pushing onto stack. After that, we declare a global function and added a display function.

```
int main(){
    int choice;
    std::cout<<"Enter number of max elements for new stack: ";
    std::cin >> i;

    while (true){
        std::cout << "\nStack operations: " << std::endl;
        std::cout << "1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT" << std::endl;
        std::cout << "choice: ";
        std::cin >> choice;

        switch (choice){
            case 1: push(); break;
            case 2: pop(); break;
            case 3: Top(); break;
            case 4: std::cout << (isEmpty() ? "Stack is empty" : "Stack is not empty") << std::endl; break;
            case 5: display(); break;
            case 6: return 0;
            default: std::cout << "Invalid choice." << std::endl;
        }
    }
    return 0;
}
```

Here in the main function, the user chooses the number of elements in the stack array that is store in i. Next is the user asks to which operation or function the user would choose. I added the display and return 0 to end the code. The isEmpty is added to print if the stack is empty or stack is not empty so that it would not print the function pointer.

```
bool isEmpty(){
    return (top == -1);
}

void push(){
    if(top == i-1){
        std::cout << "Stack Overflow." << std::endl;
        return;
    }
    std::cout << "Enter new value: ";
    std::cin >> newData;
    stack[++top] = newData;
}

void pop(){
    if (isEmpty()){
        std::cout << "Stack Underflow." << std::endl;
        return;
    }
    std::cout << "Popping: " << stack[top] << std::endl;
    top--;
}

void Top(){
    if (isEmpty()){
        std::cout << "Stack is empty." << std::endl;
        return;
    }
    std::cout << "The element on the top of the stack is: " << stack[top] << std::endl;
}
```

The isEmpty function returns True if there is no element in the stack. Otherwise, it will return as False. The push function has the condition if the top is equal to i-1, then it indicates that it is full or overflows then it returns. If it is not full, then it will ask the user to input the newData and increments the top and stores a new value in the stack. The pop function checks if the stack is empty and will print “Stack Underflow” then it returns. If not, it will decrement the top which it will remove the element which is at the top of the stack. The top function checks first if the stack is empty and will return. If not, it will print the top element in the stack.

```
void display(){
    if (isEmpty()){
        std::cout << "Stack is empty." << std::endl;
        return;
    }
    std::cout << "Stack elements (top to bottom): ";
    for (int j = top; j >= 0; j--){
        std::cout << stack[j] << " ";
    }
    std::cout << std::endl;
}
```

The display function, which is added, checks also first if the stack is empty and returns it. If not, it will print the elements from top to bottom by creating a for loop. It will start at the top element and loops it until it is greater or equal to 0 and it will decrement so that it will show the elements below.

```
e-Out-ao0g3b34.fhr' '--stderr=Microsoft-MIEngine-Error-3awvrwt5.ktr
Enter number of max elements for new stack: 5

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 4
Stack is empty

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 1
Enter new value: 1

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 1
Enter new value: 2

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 1
Enter new value: 3

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 1
Enter new value: 4

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 1
Enter new value: 5

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 4
Stack is not empty

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 5
Stack elements (top to bottom): 5 4 3 2 1

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 2
Popping: 5

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 5
Stack elements (top to bottom): 4 3 2 1

Stack operations:
1. PUSH, 2. POP, 3. TOP, 4. ISEMPTY, 5. DISPLAY, 6. EXIT
choice: 6
PS Z:\Crishen\cpp>
```

```cpp
#include <iostream>
class Node {
   public:
   int data;
   Node *next;
};
```

Here, it defines the node in linked list. Each node has a stored value which is the data and next which points to the next node.

```cpp
Node *head=NULL, *tail=NULL;
```

In this part, the head points to the top of the stack and tail points to the bottom.

```
void push(int newData){
    Node *newNode = new Node;
    newNode->data = newData;
    newNode->next = NULL;

    if (head==NULL) {
        head = tail = newNode;
    }
    else {
        newNode->next = head;
        head = newNode;
    }
}
int pop(){
    int tempVal;
    Node *temp;
    if(head == NULL){
        head = tail = NULL;
        std::cout << "Stack Underflow." << std::endl;
        return -1;
    }
    else {
        temp = head;
        tempVal = temp->data;
        head = head->next;
        delete (temp);
        return tempVal;
    }
```

Here in the push function, it creates a new node with newData the next points to a null. If the stack is empty, both head and the tail points to it. If not, it inserts at the front of the list and updates the head. Then, the new element becomes the top of the stack. In the pop function, it checks if the stack is empty and returns -1. Otherwise, the tempVal will hold the value of the node being removed and the temporary pointer used to delete the node. The temp = head stores the current top node in a temporary pointer and will get the value from the top node. Then, moves the head pointer to the next node and deallocates memory for the old top node, it will return to tempVal.

```
void Top(){
    if(head == NULL){
        std::cout << "Stack is empty." << std::endl;
        return;
    }
    else {
        std::cout << "Top os Stack: " << head->data << std::endl;
    }
}
```

The top function checks first if the stack is empty. If not, it prints the top stack which is the node head->data.

```
void display(){
    if(head == NULL){
        std::cout << "Stack is empty." << std::endl;
        return;
    }
    Node* temp = head;
    std::cout << "Stack elements (top to bottom): ";
    while(temp != NULL){
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    std::cout << std::endl;
}
```

Here, we added a display function. First, it checks if the head is null and will print the stack is empty and exits the function. Then, we traverse by creating a temporary pointer temp starting at the head or the top of the stack. After that, I added a while loop and it loops as long as the temp is not null.

```
    std::cout << std::endl;
}
int main(){
    push(1);
    std::cout << "After the first PUSH top of the satck is: ";
    Top();
    display();

    push(5);
    std::cout << "After the second PUSH top of the stack is: ";
    Top();
    display();

    pop();
    std::cout << "After the first POP operation, top of the stack is: ";
    Top();
    display();

    pop();
    std::cout << "After the second POP operation, top of the stack is: ";
    Top();
    display();
    pop();
    return 0;
}
```

For the main function, it pushes 1 and prints the top of the stack and calls the display function. Next is it pushes 5 and it prints the top of the stack and calls the display function. Then, it pops twice and also calls the display.

```
PS Z:\Crishen\cpp>  & 'c:\Users\user\.vscode\extensions\ms-vscode.cpp
e-Out-z1u22xqe.hd2' '--stderr=Microsoft-MIEngine-Error-wueb2qrc.gbd'
After the first PUSH top of the satck is: Top os Stack: 1
Stack elements (top to bottom): 1
After the second PUSH top of the stack is: Top os Stack: 5
Stack elements (top to bottom): 5 1
After the first POP operation, top of the stack is: Top os Stack: 1
Stack elements (top to bottom): 1
After the second POP operation, top of the stack is: Stack is empty.
Stack is empty.
Stack Underflow.
PS Z:\Crishen\cpp>
```

## 7. Supplementary Activity

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

bool isBalanced(string expr) {
    stack<char> s;

    for (char ch : expr) {
        if (ch == '(' || ch == '{' || ch == '[') {
            s.push(ch);
        }
        else if (ch == ')' || ch == '}' || ch == ']') {
            if (s.empty()) {
                return false;
            }

            char top = s.top();
            s.pop();

            if ((ch == ')' && top != '(') ||
                (ch == '}' && top != '{') ||
                (ch == ']' && top != '[')) {
                return false;
            }
        }
    }

    return s.empty();
}

int main() {
    string expressions[] = {
        "(A+B)+(C-D)",
        "((A+B)+(C-D)",
        "((A+B)+[C-D])",
        "((A+B]+[C-D})"
    };
```

```
    for (string expr : expressions) {
        cout << "Expression: " << expr << endl;
        if (isBalanced(expr)) {
            cout << "Valid? YES" << endl;
        } else {
            cout << "Valid? NO" << endl;
        }
    }

    return 0;
}
```

```
e-Out-muyr0ec0.sam' '--stder
Expression: (A+B)+(C-D)  nqz
Valid? YES
Expression: ((A+B)+(C-D)
Valid? NO
Expression: ((A+B)+[C-D])
Valid? YES
Expression: ((A+B]+[C-D})
Valid? NO
PS Z:\Crishen\cpp>
```

The internal representation of the stack does not affect the correctness; it is because that all representational schemes maintain the LIFO property. Yet, it does affect the efficiency and memory consumption. An array or vector-based implementation is fast and cache-friendly but it is restricted to fixed-size or requires resizing. To avoid size limits a linked list dynamically allocates its nodes at the cost of using extra memory and a lower cache fragmentation. A deque the default of std::stack balances both of these by being able to grow efficiently but with a good performance.

**8. Conclusion**

To conclude, stack is a linear structure which follows the property of LIFO or FILO. There are different operation to performed in the stack and these are the push, pop, top, and isEmpty. A stack can be implemented with an array or linked list. They're differences is the memory consumption in which a array has a fixed sized and needs to resize and the nodes dynamically allocates its nodes which saves memory efficiently. I think I did only good at this activity and I think I should improve the linked list and a stack.

**9. Assessment Rubric**