

LAB 08

Miss Atika Aslam

Muhammad Abdullah

53457

### Lab Task 1:

Write a program to implement a stack using a linked list. Include the following operations:

* Push (Insert an element at the top of the stack)
* Pop (Remove the top element of the stack)
* Peek (Return the top element without removing it)
* Check if the stack is empty

**Code:**

#*include* <iostream>

using namespace std;

class Stack {

private:

    struct Node {

        int data;

        Node\* next;

*Node*(int value) : *data*(value), *next*(nullptr) {}

    };

    Node\* top;

public:

*Stack*() : *top*(nullptr) {}

    void *push*(int value) {

        Node\* newNode = new *Node*(value);

        newNode->next = top;

        top = newNode;

    }

    void *pop*() {

*if* (*isEmpty*()) {

            cout *<<* "Stack is empty. Cannot pop.\n";

*return*;

        }

        Node\* temp = top;

        top = top->next;

        delete temp;

    }

    int *peek*() {

*if* (*isEmpty*()) {

            cout *<<* "Stack is empty.\n";

*return* -1;

        }

*return* top->data;

    }

    bool *isEmpty*() {

*return* top == nullptr;

    }

    void *display*() {

        Node\* temp = top;

*while* (temp) {

            cout *<<* temp->data *<<* " ";

            temp = temp->next;

        }

        cout *<<* *endl*;

    }

};

int *main*() {

    Stack s;

    s.*push*(10);

    s.*push*(20);

    s.*push*(30);

    cout *<<* "Stack elements: ";

    s.*display*();

    cout *<<* "Top element: " *<<* s.*peek*() *<<* *endl*;

    s.*pop*();

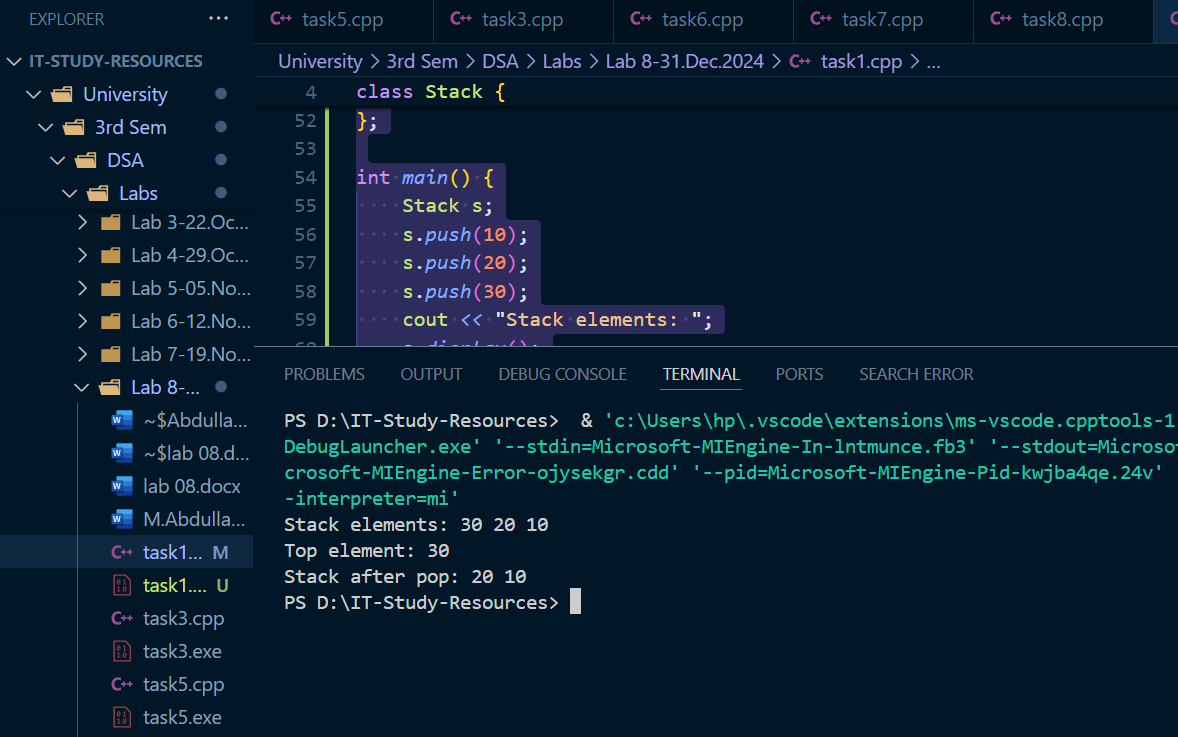
    cout *<<* "Stack after pop: ";

    s.*display*();

*return* 0;

}

## Output



**Lab Task 2:**

Use a stack (implemented using a linked list) to reverse a given string.

* Input: "hello"
* Output: "olleh"

**Code:**

#*include* <iostream>

#*include* <string>

using namespace std;

class Stack {

private:

    struct Node {

        char data;

        Node\* next;

*Node*(char value) : *data*(value), *next*(nullptr) {}

    };

    Node\* top;

public:

*Stack*() : *top*(nullptr) {}

    void *push*(char value) {

        Node\* newNode = new *Node*(value);

        newNode->next = top;

        top = newNode;

    }

    void *pop*() {

*if* (*isEmpty*()) *return*;

        Node\* temp = top;

        top = top->next;

        delete temp;

    }

    char *peek*() {

*return* *isEmpty*() ? '\0' : top->data;

    }

    bool *isEmpty*() {

*return* top == nullptr;

    }

};

string *reverseString*(*const* string*&* str) {

    Stack s;

*for* (char ch : str) s.*push*(ch);

    string reversed = "";

*while* (!s.*isEmpty*()) {

        reversed *+=* s.*peek*();

        s.*pop*();

    }

*return* reversed;

}

int *main*() {

    string input = "Abdullah";

    cout *<<* "Original: " *<<* input *<<* *endl*;

    cout *<<* "Reversed: " *<<* *reverseString*(input) *<<* *endl*;

*return* 0;

}

**OUTPUT**



**Lab Task 3:**

Write a program to check if a string containing parentheses ({}, [], ()) is balanced. Use a stack implemented with a linked list.

* Input: "({[()]})"
* Output: Balanced
* Input: "({[([)])}"
* Output: Not Balanced

**Code:**

#*include* <iostream>

#*include* <string>

using namespace std;

class Stack {

private:

    struct Node {

        char data;

        Node\* next;

*Node*(char value) : *data*(value), *next*(nullptr) {}

    };

    Node\* top;

public:

*Stack*() : *top*(nullptr) {}

    void *push*(char value) {

        Node\* newNode = new *Node*(value);

        newNode->next = top;

        top = newNode;

    }

    void *pop*() {

*if* (*isEmpty*()) *return*;

        Node\* temp = top;

        top = top->next;

        delete temp;

    }

    char *peek*() {

*return* *isEmpty*() ? '\0' : top->data;

    }

    bool *isEmpty*() {

*return* top == nullptr;

    }

};

bool *isBalanced*(*const* string*&* str) {

    Stack s;

*for* (char ch : str) {

*if* (ch == '(' || ch == '{' || ch == '[') {

            s.*push*(ch);

        } *else* *if* (ch == ')' || ch == '}' || ch == ']') {

*if* (s.*isEmpty*()) *return* false;

            char top = s.*peek*();

*if* ((ch == ')' && top == '(') || (ch == '}' && top == '{') || (ch == ']' && top == '[')) {

                s.*pop*();

            } *else* {

*return* false;

            }

        }

    }

*return* s.*isEmpty*();

}

int *main*() {

    string input1 = "({[()]})";

    string input2 = "({[([)])})";

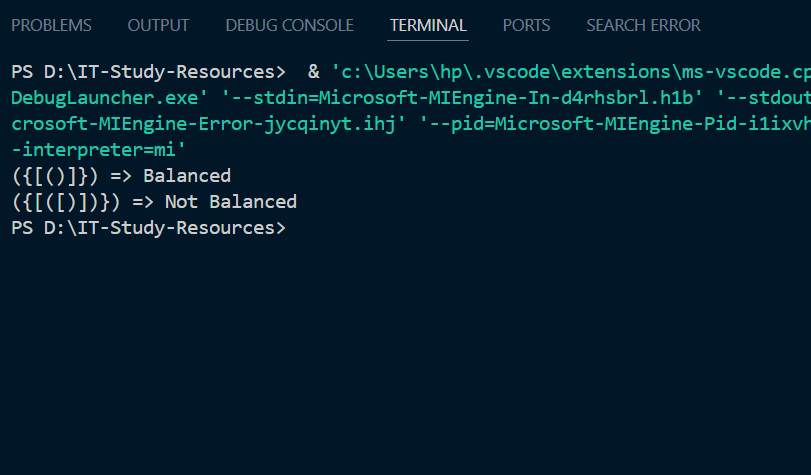
    cout *<<* input1 *<<* " => " *<<* (*isBalanced*(input1) ? "Balanced" : "Not Balanced") *<<* *endl*;

    cout *<<* input2 *<<* " => " *<<* (*isBalanced*(input2) ? "Balanced" : "Not Balanced") *<<* *endl*;

*return* 0;

}

**OUTPUT**



**Lab Task 4:**

Sort a stack using recursion and only one additional stack for temporary storage. Implement the stack using a linked list.

* Input Stack: [3, 1, 4, 2]
* Output Stack: [1, 2, 3, 4]

**Code:**

#*include* <iostream>

using namespace std;

class Stack {

private:

    struct Node {

        int data;

        Node\* next;

*Node*(int value) : *data*(value), *next*(nullptr) {}

    };

    Node\* top;

public:

*Stack*() : *top*(nullptr) {}

    void *push*(int value) {

        Node\* newNode = new *Node*(value);

        newNode->next = top;

        top = newNode;

    }

    void *pop*() {

*if* (*isEmpty*()) *return*;

        Node\* temp = top;

        top = top->next;

        delete temp;

    }

    int *peek*() {

*return* *isEmpty*() ? -1 : top->data;

    }

    bool *isEmpty*() {

*return* top == nullptr;

    }

    void *display*() {

        Node\* temp = top;

*while* (temp) {

            cout *<<* temp->data *<<* " ";

            temp = temp->next;

        }

        cout *<<* *endl*;

    }

};

void *sortedInsert*(Stack*&* s, int value) {

*if* (s.*isEmpty*() || value > s.*peek*()) {

        s.*push*(value);

*return*;

    }

    int temp = s.*peek*();

    s.*pop*();

*sortedInsert*(s, value);

    s.*push*(temp);

}

void *sortStack*(Stack*&* s) {

*if* (!s.*isEmpty*()) {

        int temp = s.*peek*();

        s.*pop*();

*sortStack*(s);

*sortedInsert*(s, temp);

    }

}

int *main*() {

    Stack s;

    s.*push*(3);

    s.*push*(1);

    s.*push*(4);

    s.*push*(2);

    cout *<<* "Original stack: ";

    s.*display*();

*sortStack*(s);

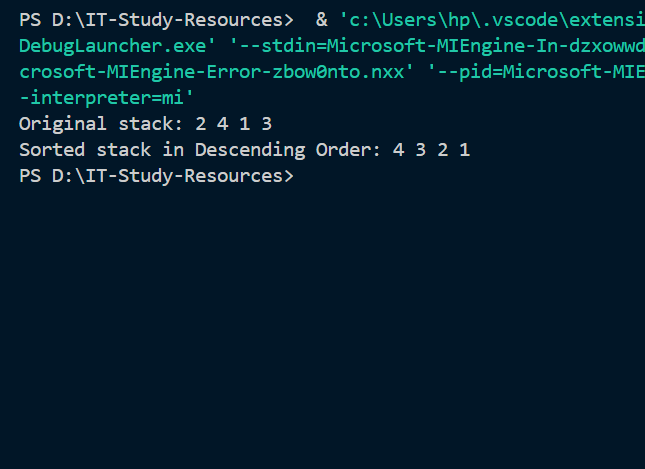
    cout *<<* "Sorted stack in Descending Order: ";

    s.*display*();

*return* 0;

}

**OUTPUT**



**Lab Task 5:**

Use a stack (implemented using a linked list) to evaluate a postfix expression.

* Input: "231\*+9-"
* Output: -4

**Code:**

#*include* <iostream>

using namespace std;

class Stack {

private:

    struct Node {

        int data;

        Node\* next;

*Node*(int value) : *data*(value), *next*(nullptr) {}

    };

    Node\* top;

public:

*Stack*() : *top*(nullptr) {}

    void *push*(int value) {

        Node\* newNode = new *Node*(value);

        newNode->next = top;

        top = newNode;

    }

    void *pop*() {

*if* (*isEmpty*()) *return*;

        Node\* temp = top;

        top = top->next;

        delete temp;

    }

    int *peek*() {

*return* *isEmpty*() ? -1 : top->data;

    }

    bool *isEmpty*() {

*return* top == nullptr;

    }

};

int *evaluatePostfix*(*const* string*&* expression) {

    Stack s;

*for* (char ch : expression) {

*if* (*isdigit*(ch)) {

            s.*push*(ch - '0');

        } *else* {

            int val2 = s.*peek*(); s.*pop*();

            int val1 = s.*peek*(); s.*pop*();

*switch* (ch) {

*case* '+': s.*push*(val1 + val2); *break*;

*case* '-': s.*push*(val1 - val2); *break*;

*case* '\*': s.*push*(val1 \* val2); *break*;

*case* '/': s.*push*(val1 / val2); *break*;

            }

        }

    }

*return* s.*peek*();

}

int *main*() {

    string postfix = "231\*+9-";

    cout *<<* "Postfix Expression: " *<<* postfix *<<* *endl*;

    cout *<<* "Evaluation Result: " *<<* *evaluatePostfix*(postfix) *<<* *endl*;

    string postfix2 = "431\*+9+2/";

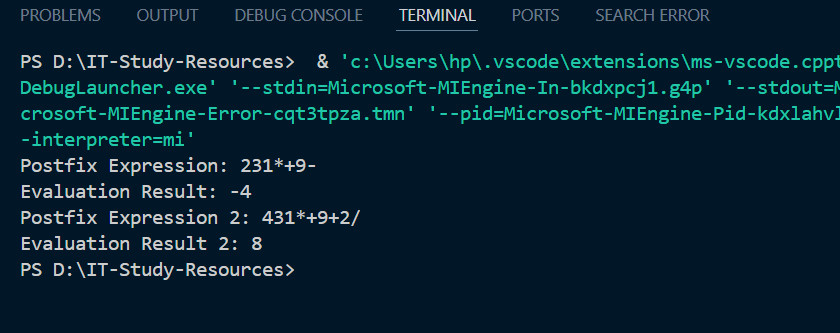
    cout *<<* "Postfix Expression 2: " *<<* postfix2 *<<* *endl*;

    cout *<<* "Evaluation Result 2: " *<<* *evaluatePostfix*(postfix2) *<<* *endl*;

*return* 0;

}

**OUTPUT**



**Lab Task 6:**

Modify the stack implementation (using a linked list) to include a getMin() function, which returns the minimum element in the stack in O(1) time.

**Code:**

#*include* <iostream>

using namespace std;

class Stack {

private:

    struct Node {

        int data;

        int minData;

        Node\* next;

*Node*(int value, int minVal) : *data*(value), *minData*(minVal), *next*(nullptr) {}

    };

    Node\* top;

public:

*Stack*() : *top*(nullptr) {}

    void *push*(int value) {

        int minVal = *isEmpty*() ? value : *min*(value, top->minData);

        Node\* newNode = new *Node*(value, minVal);

        newNode->next = top;

        top = newNode;

    }

    void *pop*() {

*if* (*isEmpty*()) *return*;

        Node\* temp = top;

        top = top->next;

        delete temp;

    }

    int *peek*() {

*return* *isEmpty*() ? -1 : top->data;

    }

    int *getMin*() {

*return* *isEmpty*() ? -1 : top->minData;

    }

    bool *isEmpty*() {

*return* top == nullptr;

    }

};

int *main*() {

    Stack s;

    s.*push*(3);

    s.*push*(5);

    s.*push*(2);

    s.*push*(1);

    cout *<<* "Minimum: " *<<* s.*getMin*() *<<* *endl*;

    s.*pop*();

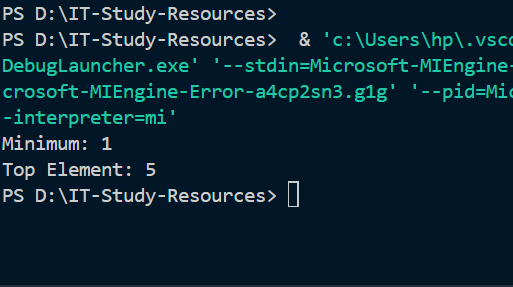
    s.*pop*();

    cout *<<* "Top Element: " *<<* s.*peek*() *<<* *endl*;

*return* 0;

}

**OUTPUT**



**Lab Task 7:**

Reverse a given linked list using a stack (implemented with another linked list).

* Input: 1 → 2 → 3 → 4
* Output: 4 → 3 → 2 → 1

**Code:**

#*include* <iostream>

using namespace std;

class LinkedList {

private:

    struct Node {

        int data;

        Node\* next;

*Node*(int value) : *data*(value), *next*(nullptr) {}

    };

    Node\* head;

public:

*LinkedList*() : *head*(nullptr) {}

    void *push*(int value) {

        Node\* newNode = new *Node*(value);

        newNode->next = head;

        head = newNode;

    }

    void *display*() {

        Node\* temp = head;

*while* (temp) {

            cout *<<* temp->data *<<* " ";

            temp = temp->next;

        }

        cout *<<* *endl*;

    }

    void *reverse*() {

        Node\* temp = head;

        LinkedList stack;

*while* (temp) {

            stack.*push*(temp->data);

            temp = temp->next;

        }

        head = stack.head;

    }

};

int *main*() {

    LinkedList list;

    list.*push*(32);

    list.*push*(1);

    list.*push*(10);

    list.*push*(2);

    cout *<<* "Original List: ";

    list.*display*();

    list.*reverse*();

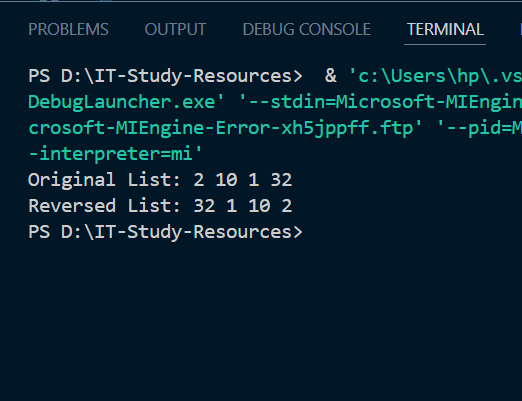
    cout *<<* "Reversed List: ";

    list.*display*();

*return* 0;

}

**OUTPUT**



**Lab Task 8:**

Given an array, find the next greater element for each element using a stack implemented with a linked list.

* Input: [4, 5, 2, 10]
* Output: [5, 10, 10, -1]

**Code:**

#*include* <iostream>

#*include* <vector>

using namespace std;

vector<int> *nextGreaterElement*(*const* vector<int>*&* arr) {

    vector<int> *result*(arr.*size*(), -1);

    vector<int> stack;

*for* (int i = arr.*size*() - 1; i >= 0; --i) {

*while* (!stack.*empty*() && stack.*back*() <= arr*[*i*]*) {

            stack.*pop\_back*();

        }

*if* (!stack.*empty*()) {

            result*[*i*]* = stack.*back*();

        }

        stack.*push\_back*(arr*[*i*]*);

    }

*return* result;

}

int *main*() {

    vector<int> arr = {4, 5, 2, 10};

    vector<int> result = *nextGreaterElement*(arr);

    cout *<<* "Array: ";

*for* (int num : arr) cout *<<* num *<<* " ";

    cout *<<* *endl*;

    cout *<<* "Next Greater Elements: ";

*for* (int num : result) cout *<<* num *<<* " ";

    cout *<<* *endl*;

*return* 0;

}

**OUTPUT**

