МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Інститут комп’ютерних наук та інформаційних технологій
Кафедра комп’ютерних наук (проектування і програмування інтелектуальних систем та пристроїв)

ЗВІТ

Про виконання лабораторної роботи №2

«ДИНАМІЧНІ ОБ'ЄКТИ ТА СПОСОБИ ЇХ ВИКОРИСТАННЯ. СТЕКИ»

Виконав:
Студент групи ПП-12
Венгрин Владислав

Прийняв:
доцент кафедри САП
Кулешник Я. Ф.

Львів – 2025

Мета роботи: ознайомитись із особливостями застосування динамічних об'єктів складної структури, а саме стеками та чергами; з операціями, які виконуються над елементами цих об'єктів. Набути практичних навичок програмування з використанням динамічних об'єктів на прикладі стеків.

Варіант 3

Хід виконання лабораторної роботи:

Завдання 1. Для роботи зі стеком використати динамічне виділення пам’яті для його елементів, без використання бібліотеки STL.
Визначити, чи є два стеки однаковими (однаковий розмір і елементи).

Код програми:

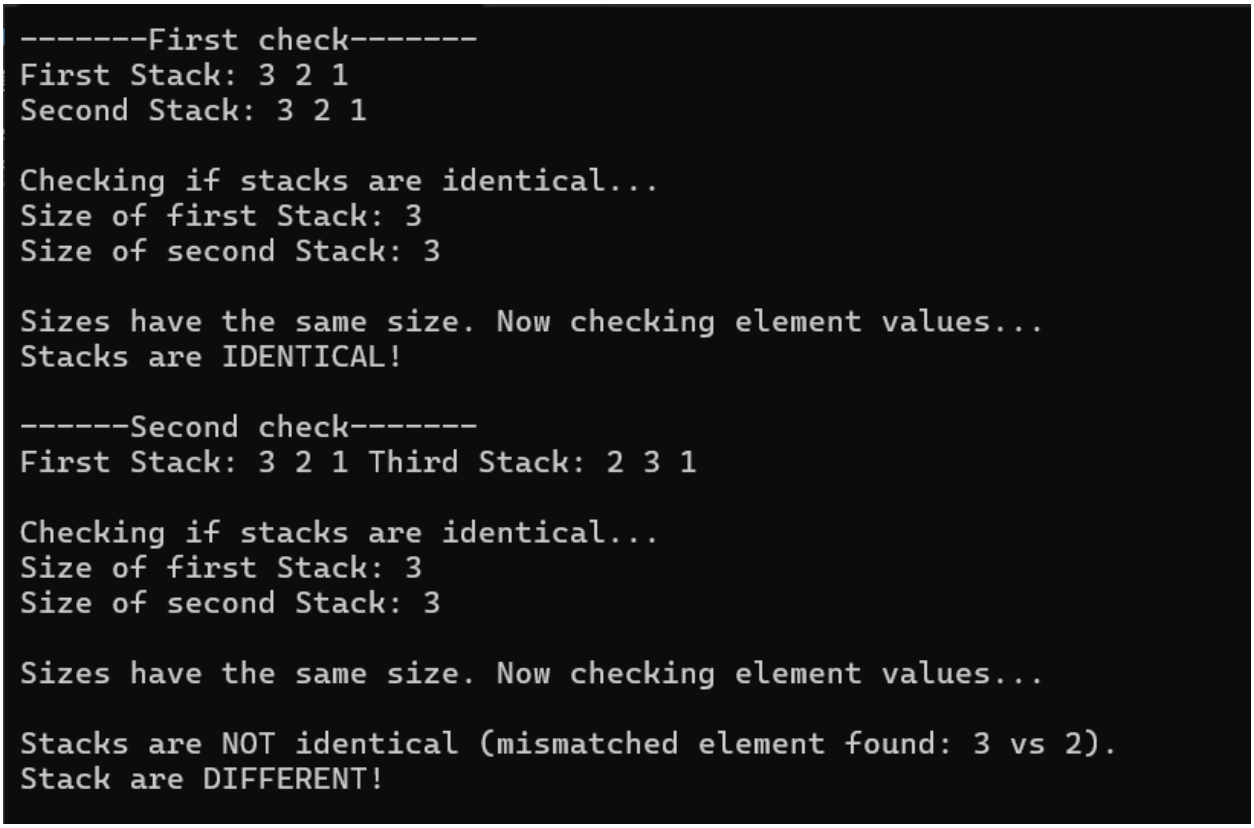
```
#include <iostream>

class Node {
public:
	int data;
	Node* next;

	Node(int new_data) {
		this->data = new_data;
		this->next = nullptr;
	}
};

class Stack {
private:
	Node* head;
public:
	Stack() {
		this->head = nullptr;
	}
	Node* getHead() {
		if (!head) {
			std::cout << "No head found." << std::endl;
			return nullptr;
		}
		return head;
	}

	bool isEmpty() {
		return head == nullptr;
	}

	void push(int element) {
		Node* newNode = new Node{element};
```

```
        if (!newNode) {
                std::cout << "\nStack Overflow";
        }

        newNode->next = head;
        head = newNode;
}

void pop() {
        if (this->isEmpty()) {
                std::cout << "Stack is empty" << std::endl;
        }
        else {
                Node* temp = head;
                head = head->next;
                delete temp;
        }
}

int peek() {
        if (!isEmpty()) {
                return head->data;
        }
        else {
                std::cout << "\nStack is empty";
                return INT_MIN;
        }

}

void display() {
        if (isEmpty()) {
                std::cout << "There's nothing to display. Stack is empty" << std::endl;
        }
        else {
                Node* temp = head;
                while (temp != nullptr) {
                        std::cout << temp->data << " ";
                        temp = temp->next;
                }
        }
}

bool operator==(Stack* other_stack) {
        std::cout << "\n\nChecking if stacks are identical...\n";
```

```
		int count1 = 0;
		Node* temp1 = head;
		while (temp1 != nullptr) {
			count1++;
			temp1 = temp1->next;
		}

		int count2 = 0;
		Node* temp2 = other_stack->getHead();
		while (temp2 != nullptr) {
			count2++;
			temp2 = temp2->next;
		}

		std::cout << "Size of first Stack: " << count1 << std::endl;
		std::cout << "Size of second Stack: " << count2 << std::endl;

		if (count1 != count2) {
			std::cout << "\nStacks are NOT identical (different sizes).\n";
			return false;
		}

		std::cout << "\nSizes have the same size. Now checking element values...\n";

		temp1 = head;
		temp2 = other_stack->getHead();

		while (temp1 != nullptr && temp2 != nullptr) {
			if (temp1->data != temp2->data) {
				std::cout << "\nStacks are NOT identical (mismatched element
found: "
					<< temp1->data << " vs " << temp2->data << ").\n";
				return false;
			}
			temp1 = temp1->next;
			temp2 = temp2->next;
		}

		return true;
	}

};

int main() {
	Stack st1;
```

```
	st1.push(1);
	st1.push(2);
	st1.push(3);

	Stack st2;
	st2.push(1);
	st2.push(2);
	st2.push(3);

	Stack st3;
	st3.push(1);
	st3.push(3);
	st3.push(2);

	std::cout << "-------First check-------\n";

	std::cout << "First Stack: ";
	st1.display();
	std::cout << "\nSecond Stack: ";
	st2.display();

	if (st1 == &st2) {
		std::cout << "Stacks are IDENTICAL!\n";
	}
	else {
		std::cout << "Stacks are DIFFERENT!\n";
	}

	std::cout << "\n------Second check-------\n";

	std::cout << "First Stack: ";
	st1.display();
	std::cout << "Third Stack: ";
	st3.display();

	if (st1 == &st3) {
		std::cout << "Stack are IDENTICAL!\n";
	}
	else {
		std::cout << "Stack are DIFFERENT!\n";
	}


	return 0;
}
```

Результати програми:



Завдання 2. Створити стек згідно з варіантом табл. 1. У програмі повинні бути передбачені наступні функції: «Додавання елемента»; «Видалення елемента»; «Перегляд»; «Звільнення стеку». Повинні бути передбачені аварійні ситуації (наприклад: не можна видалити елемент, якщо стек порожній). Для роботи зі стеком використати динамічне виділення пам’яті для його елементів, без використання бібліотеки STL.

Код програми:

```
#include <iostream>

class Node {
public:
	int data;
	Node* next;

	Node(int element) {
		this->data = element;
		this->next = nullptr;
	}
};

class Stack {
private:
	Node* head;
public:
```

```
Stack() {
	head = nullptr;
}

~Stack() {
	while (head != nullptr) {
		Node* temp = head;
		head = head->next;
		delete temp;
	}
}

bool isEmpty() {
	return head == nullptr;
}

void push(int element) {
	Node* newNode = new Node{ element };
	if (!newNode) {
		std::cout << "\nStack Overflow";
	}

	newNode->next = head;
	head = newNode;
}

void pop() {
	if (head == nullptr) {
		std::cout << "The Stack is empty" << std::endl;
		return;
	}
	else {
		Node* temp = head;
		head = head->next;
		delete temp;
	}
}

void peek() {
	if (!isEmpty()) {
		std::cout << "The top of the stack is " << head->data << std::endl;
		return;
	}
	else {
		std::cout << "\nStack is empty";
	}
```

```
    }

    void displayElements() {
        if (isEmpty()) {
            std::cout << "There's nothing to display. Stack is empty" << std::endl;
            return;
        }
        else {
            Node* temp = head;
            std::cout << "Stack: \n";
            while (temp != nullptr) {
                std::cout << temp->data << " ";
                temp = temp->next;
            }
        }
    }

    void displayTopPointer() {
        if (isEmpty()) {
            std::cout << "\nStack is empty to display its top pointer." << std::endl;
            return;
        }
        std::cout << "\n\nPointer of the top of the stack: " << head << std::endl;
    }

};

int main() {
    Stack st1;

    std::cout << "Displaying first filled with values stack: \n";
    st1.push(1);
    st1.push(10);
    st1.push(2);
    st1.push(3);
    st1.push(1);

    st1.displayElements();
    st1.displayTopPointer();

    Stack st2;

    std::cout << "\nDisplaying second unfilled stack: \n";
    st2.displayElements();
    st2.displayTopPointer();
```

```
    return 0;
}
```

Результати програми:

```
Displaying first filled with values stack:
Stack:
1 3 2 10 1

Pointer of the top of the stack: 00000258B7CB8640

Displaying second unfilled stack:
There's nothing to display. Stack is empty

Stack is empty to display its top pointer.
```

Відповіді на питання для самоконтролю:

1. **У чому переваги і недоліки організації структур у вигляді стеку?**
   Переваги: проста реалізація, швидкі операції push і pop (O(1)), використовується в рекурсії, управлінні пам’яттю, алгоритмах пошуку. Недоліки: доступ тільки за принципом LIFO, неможливість довільного доступу до середніх або нижніх елементів, обмеження за розміром у масивній реалізації.
2. **Для моделювання яких реальних завдань зручно використовувати стек?**
   Стек застосовується для управління викликами функцій у програмах, механізму "назад/вперед" у браузерах, відкату операцій у текстових редакторах, обчислення математичних виразів у компіляторах і калькуляторах, а також у пошукових алгоритмах (DFS).
3. **Яке значення зберігає вказівник на стек?**
   Вказівник на стек (head у зв’язаному списку або top у масиві) зберігає адресу або індекс верхнього елемента, що дозволяє швидко виконувати операції додавання та видалення.
4. **Які існують обмеження на тип інформаційного поля стеку?**
   У класичному стеку можна зберігати будь-який тип даних, але в масивній реалізації тип зазвичай фіксований (наприклад, int або char). У шаблонних класах C++ (template) стек може працювати з будь-якими типами даних.
5. **З якою метою в програмах виконується перевірка на відсутність елементів стеку? Чи можлива ситуація запису нових елементів стеку на вже зайняті власними елементами ділянки пам'яті?**
   Перевірка на порожній стек необхідна, щоб уникнути помилок при спробі видалення або зчитування елемента, якого немає. У масивній реалізації стеку (якщо не передбачена зміна розміру) можливий переповнення (stack overflow), коли нові елементи не мають місця для зберігання. У зв’язаному списку така проблема відсутня, але можливе вичерпання пам’яті, якщо стек зростає без обмежень.
6. **З якою метою в програмах виконується видалення стеку після закінчення роботи з ним? Як зміниться робота програми, якщо операцію видалення не**

виконувати?

Видалення стеку (звільнення пам’яті) після його використання запобігає витокам пам’яті (memory leaks). Якщо цього не робити, програма може використовувати більше пам’яті, ніж потрібно, що особливо критично для довготривалих процесів або обмежених ресурсів. У мовах з автоматичним керуванням пам’яттю (наприклад, Python) ця проблема менш актуальна, а в C++ необхідно явно видаляти виділені динамічно елементи.

Висновок:

Я ознайомився із особливостями застосування динамічних об'єктів складної структури, а саме стеками та чергами; з операціями, які виконуються над елементами цих об'єктів. Набув практичних навичок програмування з використанням динамічних об'єктів на прикладі стеків.