МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Программирование»
Тема: Динамические структуры данных

Студент гр. 3342 _____ Галеев А.Д.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Изучение, анализ и реализации динамических структур данных с целью понимания их принципов функционирования, эффективного использования и возможностей оптимизации.

Задание

Вариант №3

Требуется написать программу, моделирующую работу стека на базе массива. Для этого необходимо:

1) Реализовать класс CustomStack, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных *int.*

Перечень методов класса стека, которые должны быть реализованы:

- void push(int val) - добавляет новый элемент в стек
- void pop() - удаляет из стека последний элемент
- int top() - возвращает верхний элемент
- size_t size() - возвращает количество элементов в стеке
- bool empty() - проверяет отсутствие элементов в стеке
- extend(int n) - расширяет исходный массив на n ячеек

2) Обеспечить в программе считывание из потока *stdin* последовательности команд (каждая команда с новой строки), в зависимости от которых программа выполняет ту или иную операцию и выводит результат ее выполнения с новой строки.

Перечень команд, которые подаются на вход программе в *stdin*:

- cmd_push n - добавляет целое число n в стек. Программа должна вывести "ok"
- cmd_pop - удаляет из стека последний элемент и выводит его значение на экран
- cmd_top - программа должна вывести верхний элемент стека на экран не удаляя его из стека
- cmd_size - программа должна вывести количество элементов в стеке
- cmd_exit - программа должна вывести "bye" и завершить работу

Если в процессе вычисления возникает ошибка (например вызов метода pop или top при пустом стеке), программа должна вывести "error" и завершиться.

Основные теоретические положения

Для решения задач в программе использовались стандартные библиотеки языка C++, они предоставляют функции для работы с Динамическими структурами данных.

Выполнение работы

CustomStack(): Конструктор класса CustomStack, который инициализирует члены данных mData, mCapacity и mSize. mData инициализируется значением nullptr, а mCapacity и mSize устанавливаются в 0.

void push(int val): Метод для добавления нового элемента в стек. Если текущий размер стека (mSize) равен его вместимости (mCapacity), вызывается функция extend, чтобы увеличить размер стека, иначе новый элемент просто добавляется в конец массива mData.

void pop(): Метод для удаления верхнего элемента из стека. Просто уменьшает значение mSize на 1.

int top(): Метод для получения значения верхнего элемента стека, не удаляя его. Возвращает значение элемента из массива mData, находящегося в позиции mSize - 1.

size_t size(): Метод для возврата текущего размера стека, т.е., количество элементов в стеке. Просто возвращает значение mSize.

bool empty(): Метод для проверки, пуст ли стек. Возвращает true, если mSize равно 0, иначе false.

void extend(int n): Метод для расширения вместимости стека. Создает новый массив newData с увеличенной вместимостью, копирует в него все элементы из старого массива mData, освобождает память старого массива, и заменяет указатель mData на новый массив. Обновляет значение mCapacity на новую вместимость.

Тестирование

Результаты тестирования представлены в табл. 1

Таблица 1 – Результаты тестирования

№ проверки	Входные данные	Выходные данные
1.	cmd_push 1 cmd_top cmd_push 2 cmd_top cmd_pop cmd_size cmd_pop cmd_size cmd_exit	ok 1 ok 2 2 1 1 0 bye

Выводы

В ходе выполнения лабораторной работы были изучены основные принципы, преимущества и недостатки динамических структур. Были изучены различные типы динамических структур данных. Также были исследованы методы оптимизации.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main_lb4

```
class CustomStack {
public:
    CustomStack() : mData(nullptr), mCapacity(0), mSize(0) {}
    ~CustomStack() {
        delete[] mData;
    }

    void push(int val) {
        if (mSize == mCapacity) {
            if (!extend(1)) {
                std::cerr << "Failed to extend memory." << std::endl;
                return;
            }
        }
        mData[mSize++] = val;
    }

    void pop() {
        --mSize;
    }

    int top() {
        return mData[mSize - 1];
    }

    size_t size() {
        return mSize;
    }

    bool empty() {
        return mSize == 0;
    }

    bool extend(int n) {
        int newCapacity = mCapacity + n;
        int* newData = new (std::nothrow) int[newCapacity];
        if (!newData) {
            return false;
        }
        for (size_t i = 0; i < mSize; ++i) {
            newData[i] = mData[i];
        }
        delete[] mData;
        mData = newData;
        mCapacity = newCapacity;
        return true;
    }

protected:
    int* mData;
```

```
private:
    size_t mCapacity;
    size_t mSize;
};

int main() {
    CustomStack stack;

    std::string cmd;
    while (std::cin >> cmd) {
        if (cmd == "cmd_push") {
            int n;
            std::cin >> n;
            stack.push(n);
            std::cout << "ok" << std::endl;
        } else if (cmd == "cmd_pop") {
            if (stack.empty()) {
                std::cout << "error" << std::endl;
                return 0;
            }
            std::cout << stack.top() << std::endl;
            stack.pop();
        } else if (cmd == "cmd_top") {
            if (stack.empty()) {
                std::cout << "error" << std::endl;
                return 0;
            }
            std::cout << stack.top() << std::endl;
        } else if (cmd == "cmd_size") {
            std::cout << stack.size() << std::endl;
        } else if (cmd == "cmd_exit") {
            std::cout << "bye" << std::endl;
            break;
        } else {
            std::cout << "error" << std::endl;
            break;
        }
    }

    return 0;
}
```