Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

Факультет вычислительной математики и кибернетики

Отчёт по лабораторной работе

Стек на списке

Выполнил:
студент ф-та ИИТММ гр. 381908-01
Козел С. А.

Проверил:
ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Нижний Новгород
2020 г.

Содержание

Введение

Стек — структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека. Притом первым из стека удаляется элемент, который был помещен туда последним, то есть в стеке реализуется стратегия «последним вошел — первым вышел» (last-in, first-out — LIFO). Примером стека в реальной жизни может являться стопка тарелок: когда мы хотим вытащить тарелку, мы должны снять все тарелки выше.

Постановка задачи

Цель данной работы - разработка структуры данных для хранения стека с использованием списка и реализация стандартных функций стека.

Выполнение работы предполагает решение следующих задач:

1. Реализация класса стека Stack.
2. Реализация метода для нахождения максимального элемента в стеке.
3. Реализация метода для нахождения минимального элемента в стеке
4. Реализация методов для ввода/вывода структуры данных в файл
5. Публикация исходных кодов в личном репозиторий на GitHub.

Руководство пользователя

Пользователю нужно запустить файл StackList.exe.

Откроется консольное приложение для тестирования стека.

Программа покажет функциональность каждой функции стека по средствам вывода данных в консоль.

Руководство программиста

Описание структуры программы

Программа состоит из следующих модулей:

- Приложение StacList
- Статическая библиотека Stack:
  - Stack.h – описание класса стека
- Приложение main:
  - main.cpp – тестирование работы программы

Описание структур данных

Класс Stack:

```
1. void push(T val) – вставка элемента;
2. void pop() – удаление элемента с вершины;
3. bool empty() – проверка на пустоту;
4. int size() – получить количество элементов в стеке;
5. T Top() – просмотр элемента с вершины;
6. T FindMin() – поиск минимума;
7. T FindMax() – поиск максимума;
8. void OutputFile(std::string path) – вывод стека в файл;
9. void FromFile(std::string path) – чтение стека с файла;
10. void print() – вывод стека в консоль;
```

Описание алгоритмов

В программе используется вставка элементов в начало стека и удаление элемента с конца – способ организации (LIFO). То есть самый верхний элемент стека, который добавлен последним, извлекается самым первым.

Эксперименты

```
9.6 7.8 -5.5 -4.4 -2.1 -7.4 9.9 -0.5
9.6 7.8 -5.5 -4.4 -2.1 -7.4 9.9 -0.5
Min: -7.4
Max: 9.9
File opened and rewritten!

8 5 -2 9 8 -3 23 12
8 5 -2 9 8 -3 23 12
Min: -3
Max: 23
File opened and rewritten!
После прочтения из файла: 8 5 -2 9 8 -3 23 12

S F N Y D W S W
S F N Y D W S W
Min: D
Max: Y
File opened and rewritten!

Пустой стек: 1
Pop из пустого стека: Stack is empty
Top из пустого стека: Stack is empty
FindMax из пустого стека: Stack is empty
FindMin из пустого стека: Stack is empty
OutputFile из пустого стека: Stack is empty
Print из пустого стека: Stack is empty
```

Заключение

При выполнении данной работы мною была полностью изучена и успешно реализована структура данных стек с использованием списков.

Литература

1. https://www.guru99.com/stack-in-cpp-stl.html
2. https://prog-cpp.ru/data-stack/
3. https://ru.wikipedia.org/wiki/Стек

Приложения

Приложение 1

Stack.h

```
template<class T>
class Stack
{
private:
    std::list<T> list;
public:
    void push(T val);
    void pop();
    bool empty();
    int size();
    T Top();

    T FindMin();
    T FindMax();

    void OutputFile(std::string path);
    void FromFile(std::string path);

    void print();
};
```

```cpp
template<class T>
void Stack<T>::push(T val)
{
    list.push_front(val);
}

template<class T>
void Stack<T>::pop()
{
    if (empty()) { throw std::out_of_range("Stack is empty"); }
    list.pop_back();
}

template<class T>
bool Stack<T>::empty()
{
    if (list.size() == 0) { return true; }
    else { return false; }

}

template<class T>
int Stack<T>::size()
{
    return list.size();
}

template<class T>
T Stack<T>::Top()
{
    if (empty()) { throw std::out_of_range("Stack is empty"); }
    return list.back();
}

template<class T>
T Stack<T>::FindMin()
{
    if (empty()) { throw std::out_of_range("Stack is empty"); }
    int countElements = size();
    Stack<T> tmp;
    T min = Top();

    for (int i = 0; i < countElements; i++)
    {
        if (Top() < min) { min = Top(); }
        tmp.push(Top());
        pop();
    }
    for (int i = 0; i < countElements; i++)
    {
        T tmpVal = tmp.Top();
        push(tmpVal);
        tmp.pop();
    }
    return min;
}
```

```cpp
template<class T>
T Stack<T>::FindMax()
{
    if(empty()) { throw std::out_of_range("Stack is empty"); }
    int countElements = size();
    Stack<T> tmp;
    T max = Top();

    for (int i = 0; i < countElements; i++)
    {
        if (Top() > max) { max = Top(); }
        tmp.push(Top());
        pop();
    }
    for (int i = 0; i < countElements; i++)
    {
        T tmpVal = tmp.Top();
        push(tmpVal);
        tmp.pop();
    }
    return max;
}

template<class T>
void Stack<T>::OutputFile(std::string path)
{
    if (empty()) { throw std::out_of_range("Stack is empty"); }
    std::fstream fs;
    fs.open(path, std::ios::out | std::ios::trunc);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
    }
    else
    {
        int countElements = size();
        Stack<T> tmp;
        for (int i = 0; i < countElements; i++)
        {
            tmp.push(Top());
            pop();
        }
        for (int i = 0; i < countElements; i++)
        {
            T tmpVal = tmp.Top();
            push(tmpVal);
            fs << tmpVal << " ";
            tmp.pop();
        }
        std::cout << "File opened and rewritten!" << std::endl;
    }
    fs.close();
}
```

```cpp
template<class T>
void Stack<T>::FromFile(std::string path)   // only int
{
    std::fstream fs;
    fs.open(path, std::ios::in | std::ios::app);
    if (!fs.is_open())
    {
        std::cout << "Error open file!" << std::endl;
    }
    else
    {
        std::string str;
        std::string num;
        getline(fs, str);

        for (int i = 0; i < str.size(); i++)
        {
            if (((str[i] == 45) || ((int(str[i]) > 47) && (int(str[i] < 58)))))
            {
                num += str[i];
            }
            else
            {
                push(atoi(num.c_str()));
                num.clear();
            }
        }
    }
    fs.close();
}


template<class T>
void Stack<T>::print()
{
    if (empty()) { throw std::out_of_range("Stack is empty"); }
    int countElements = size();
    Stack<T> tmp;
    for (int i = 0; i < countElements; i++)
    {
        tmp.push(Top());
        pop();
    }
    for (int i = 0; i < countElements; i++)
    {
        T tmpVal = tmp.Top();
        push(tmpVal);
        std::cout << tmpVal << " ";
        tmp.pop();
    }
}
```

Приложение 2

Main.cpp

```cpp
#define PATH_FOR_FILE_DBL "C:\\outDBL.txt"
#define PATH_FOR_FILE_INT "C:\\outINT.txt"
#define PATH_FOR_FILE_CHAR "C:\\outCHAR.txt"


int main()
{
    setlocale(LC_ALL, "RUS");

    // DOUBLE
    Stack<double> s1;
    int countElements = 8;
    srand(time(NULL));
    for (int i = 0; i < countElements; i++)
    {
        double num = (double)(rand() % (100 - (-100) + 1) + (-100)) / 10;   // -10...10
        s1.push(num);
        std::cout << num << " ";
    }
    std::cout << std::endl;

    s1.print();

    std::cout << "\nMin: " << s1.FindMin();
    std::cout << "\nMax: " << s1.FindMax() << std::endl;

    s1.OutputFile(PATH_FOR_FILE_DBL);
    std::cout << std::endl;

    // INT
    Stack<int> s2;
    countElements = 8;
    srand(time(NULL));
    for (int i = 0; i < countElements; i++)
    {
        int num = -5 + rand() % 30;
        s2.push(num);
        std::cout << num << " ";
    }
    std::cout << std::endl;

    s2.print();

    std::cout << "\nMin: " << s2.FindMin();
    std::cout << "\nMax: " << s2.FindMax() << std::endl;

    s2.OutputFile(PATH_FOR_FILE_INT);

    Stack<int> s22;
    s22.FromFile(PATH_FOR_FILE_INT);
    std::cout << "После прочтения из файла: ";
    s22.print();
    std::cout << std::endl << std::endl;
```

```
//CHAR
Stack<char> s3;
countElements = 8;
srand(time(NULL));
for (int i = 0; i < countElements; i++)
{
    char letter = 'A' + rand() % ('Z' - 'A');
    s3.push(letter);
    std::cout << letter << " ";
}
std::cout << std::endl;

s3.print();

std::cout << "\nMin: " << s3.FindMin();
std::cout << "\nMax: " << s3.FindMax() << std::endl;

s3.OutputFile(PATH_FOR_FILE_CHAR);
std::cout << std::endl;

//Tests
Stack<int> s_t1;
std::cout << "Пустой стек: " << s_t1.empty() << std::endl;  // 1 - pustoi, 0 - ectb element

s_t1.push(5);
s_t1.push(22);
s_t1.push(-3);

s_t1.pop();
s_t1.pop();
s_t1.pop();

std::cout << "Pop из пустого стека: ";
try
{
    s_t1.pop();
}
catch (std::logic_error err)
{
    std::cout << err.what();
}
std::cout << std::endl;

std::cout << "Top из пустого стека: ";
try
{
    s_t1.Top();
}
catch (std::logic_error err)
{
    std::cout << err.what();
}
std::cout << std::endl;
```

```
    std::cout << "Top из пустого стека: ";
    try
    {
        s_t1.Top();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    std::cout << "FindMax из пустого стека: ";
    try
    {
        s_t1.FindMax();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    std::cout << "FindMin из пустого стека: ";
    try
    {
        s_t1.FindMin();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    std::cout << "OutputFile из пустого стека: ";
    try
    {
        s_t1.OutputFile(PATH_FOR_FILE_INT);
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    std::cout << "Print из пустого стека: ";
    try
    {
        s_t1.print();
    }
    catch (std::logic_error err)
    {
        std::cout << err.what();
    }
    std::cout << std::endl;

    system("pause");
    return 0;
}
```