**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра АПУ**

отчет

**по лабораторной работе № 2**

**по дисциплине «Программирование»**

**Тема: Задание № 2**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3371 |  | Манешов Д. В. |
| Преподаватель |  | Писарев А.С. |

Санкт-Петербург

2024

## Постановка задачи.

Необходимо реализовать структуру данных очередь \*Queue\* (коллекция, обеспечивающая доступ к данным по методу FIFO (Fisrst IN Forst Out)) для обработки целочисленной последовательности. Данная структура должна обладать следующим набором функций:

* \*void queue(int)\* - добавление элемента в очередь
* \*int unqueue()\* - извлечение элемента из очереди
* \*int count()\* - возвращает количество элементов в коллекции
* \*void insertBeforeNegative()\* - вставка перед каждым отрицательным числом элемента со значением 1
* \*void removeNegative()\* - удаление из очереди всех элементов с отрицательной информационной частью
* \*int count(int)\* - подсчет количества вхождений переданного значения в коллекцию
* \*void clear()\* - удаление всех элементов коллекции

Данная структура данных будет подвергнута тестированию автоматическими unit-тестами, поэтому наименование функций (и названия структуры) и сигнатуру изменять не следует

Так же необходимо предусмотреть реализацию пользовательского интерфейса, обеспечивающего произвольный доступ к функциям очереди.

## Ход решения.

Программа, реализует структуру данных, называемую стек. Стек — это линейная коллекция элементов, в которой добавление и удаление происходит только с одного конца, называемого вершиной. Стек работает по принципу LIFO (last in, first out), то есть последний добавленный элемент будет первым извлеченным.

Она позволяет пользователю вводить последовательность целых чисел и сохранять их в стеке. Затем программа удаляет из стека все отрицательные числа и выводит на экран и в файл все положительные числа в стеке.

Программа имеет два режима работы: для человека и для программы. Режим работы определяется аргументом командной строки, который может быть t (true) или f (false). В режиме для человека программа выводит подсказки и сообщения на консоль. В режиме для программы в консоль не выводится ничего на консоль, кроме положительных чисел в стеке.

Программа также записывает введенные пользователем числа в файл data.txt и положительные числа в стеке в файл report.txt.

В данном блоке кода подключаются три библиотеки: iostream, windows и fstream. Iostream нужна для ввода-вывода данных с консоли. Библиотека windows нужна для установки кодировки консоли. Fstream нужна для работы с файлами. И используется пространство имен std, чтобы не писать std:: перед каждым именем из этого пространства. Пространство имен std содержит множество полезных классов, функций и переменных, таких как cout, cin, ofstream и т.д., что изображено на рисунке 1.1.

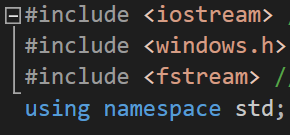


Рисунок 1.1 – Подключение библиотек и пространства имён

Далее определяется структура Node, которая представляет узел стека. Узел стека содержит два поля: data и next. Поле data хранит целое число, которое является данными узла. Поле next хранит указатель на следующий узел в стеке. Если узел является последним в стеке, то next равен nullptr, как показано на рисунке 1.2.

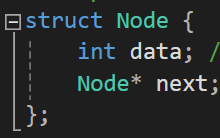


Рисунок 1.2 – Определение узла стека

После этого определяется класс Stack и затем объявляется приватное поле top, которое хранит указатель на вершину стека. Приватное поле означает, что оно доступно только внутри класса Stack и не может быть изменено извне. Указатель на вершину стека нужен для того, чтобы выполнять операции добавления, удаления и получения элементов стека. А после, объявляется публичный конструктор Stack, который инициализирует вершину стека nullptr. Публичный конструктор означает, что он может быть вызван извне класса Stack для создания объектов этого класса. Конструктор Stack не принимает никаких параметров и не возвращает никаких значений. Его единственная задача - установить вершину стека в nullptr, что означает, что стек пуст. Данные действия продемонстрированы на рисунке 1.3.

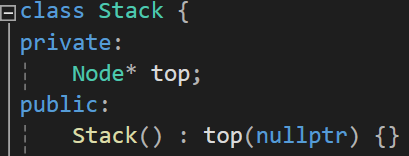


Рисунок 1.3 – Определение класса Stack, приватного поля top и публичного конструктора Stack

Далее объявляются публичные методы Stack, один из них push. Метод push предназначен для добавления элемента в стек. Принимает целое число value, которое является данными элемента. Он не возвращает никаких значений. Создает новый узел стека, присваивает ему данные value, связывает его с вершиной стека и обновляет вершину стека. Данное метод показан на рисунке 1.4.

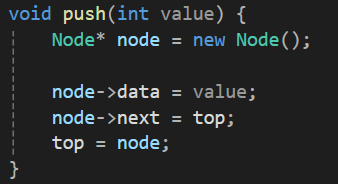


Рисунок 1.4 – Метод для добавления элемента в стек

Метод pop предназначен для извлечения элемента из стека. Он не принимает никаких параметров и не возвращает никаких значений. Проверяет, не пуст ли стек и, если стек пуст, выводит сообщение об ошибке и завершает метод. Если стек не пуст, сохраняет указатель на вершину стека во временную переменную, перемещает вершину стека на следующий узел и удаляет узел. Это представлено на рисунке 1.5.

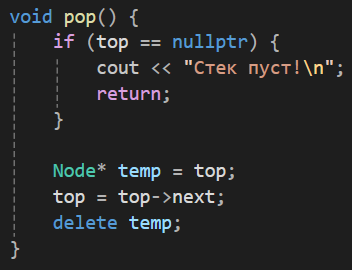


Рисунок 1.5 – Метод для извлечения элемента из стека

Метод peek нужен для получения данных вершины стека. Этот метод не принимает параметров. Возвращает целое число, которое является данными вершины стека. Проверяет, не пуст ли стек. Если стек пуст, выводит сообщение об ошибке и возвращает -1. Если стек не пуст, возвращает данные вершины стека, что изображено на рисунке 1.6.

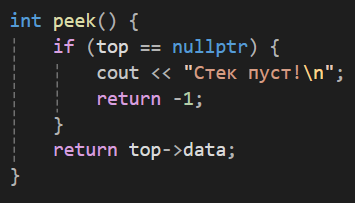


Рисунок 1.6 – Метод для получения данных вершины стека

isEmpty() - метод для проверки пустоты стека. Возвращает логическое значение, которое указывает, пуст ли стек. Если вершина стека равна nullptr, то стек пуст и возвращает true. Если вершина стека не равна nullptr, то возвращается false. Это представлено на рисунке 1.7.

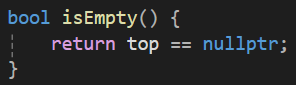


Рисунок 1.6 – Метод для проверки пустоты стека

Transfer - метод для переноса элементов из одного стека в другой. Принимает два параметра: ссылки на объекты класса Stack, которые являются исходным и целевым стеками. В цикле извлекаются элементы из исходного стека и добавляются в целевой стек, пока исходный стек не станет пустым.

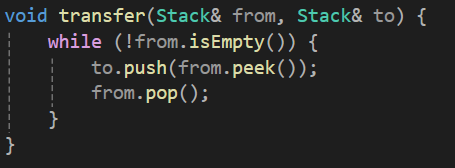


Рисунок 1.7 – Метод для переноса элементов из одного стека в другой

removeNegativeAndPrintPositive - метод для удаления отрицательных и вывода положительных чисел в стеке. Принимает один параметр: ссылку на объект класса Stack, который является исходным стеком. Создает новый объект класса Stack, являющийся стеком положительных элементов. В цикле извлекает элементы из исходного стека и проверяет их знак. Если элемент положительный, добавляет его в стек положительных элементов. После того, как исходный стек станет пустым, переносит элементы из стека положительных элементов обратно в исходный стек. Затем создает объект класса ofstream, который используется для записи в файл report.txt. В цикле извлекает элементы из исходного стека и выводит их на экран и в файл report.txt. После файл report.txt закрывается.

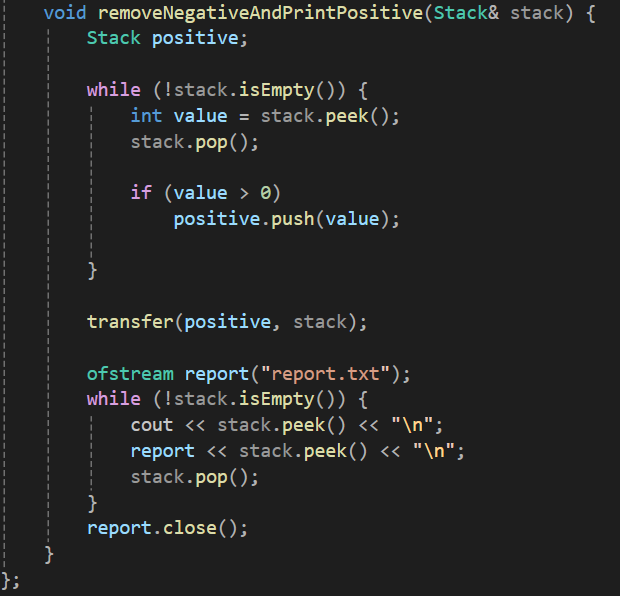


Рисунок 1.8 – Метод для удаления отрицательных и вывода положительных чисел в стеке

В этом блоке кода определяется функция main, которая является точкой входа в программу. Функция main принимает два параметра: argc и argv. Параметр argc хранит количество аргументов командной строки, включая имя программы. Параметр argv хранит массив строк, содержащих аргументы командной строки. Первый аргумент argv[0] - это имя программы, второй аргумент argv[1] - это режим работы программы.

Функция main выполняет следующие действия:

* устанавливает кодировку консоли для ввода и вывода данных на русском языке;
* создает объект класса Stack, который представляет стек целых чисел;
* создает переменную x, которая используется для ввода чисел;
* создает переменную human, которая определяет режим работы программы. Если аргумент командной строки равен t, то human принимает значение true, иначе false;
* создает объект класса ofstream, который используется для записи введенных чисел в файл data.txt;
* если режим работы программы для человека, то:
  + выводит подсказку, что нужно ввести последовательность чисел, заканчивающуюся нулем;
  + в цикле считывает числа с клавиатуры, пока не введен ноль, и добавляет их в стек и в файл data.txt;
  + закрывает файл data.txt;
  + выводит подсказку, что будут выведены положительные числа в стеке;
  + вызывает метод removeNegativeAndPrintPositive, который удаляет из стека все отрицательные числа и выводит на экран и в файл report.txt все положительные числа в стеке;
* если режим работы программы для программы, то:
  + в цикле считывает числа из входного потока, пока не введен ноль, и добавляет их в стек и в файл data.txt;
  + закрывает файл data.txt;
  + вызывает метод removeNegativeAndPrintPositive, который удаляет из стека все отрицательные числа и выводит на экран и в файл report.txt все положительные числа в стеке;
  + вызывает функцию system(“pause”), чтобы приостановить выполнение программы и дать пользователю возможность просмотреть результаты.

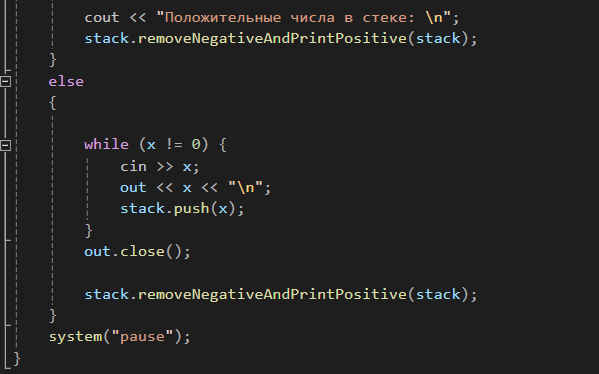
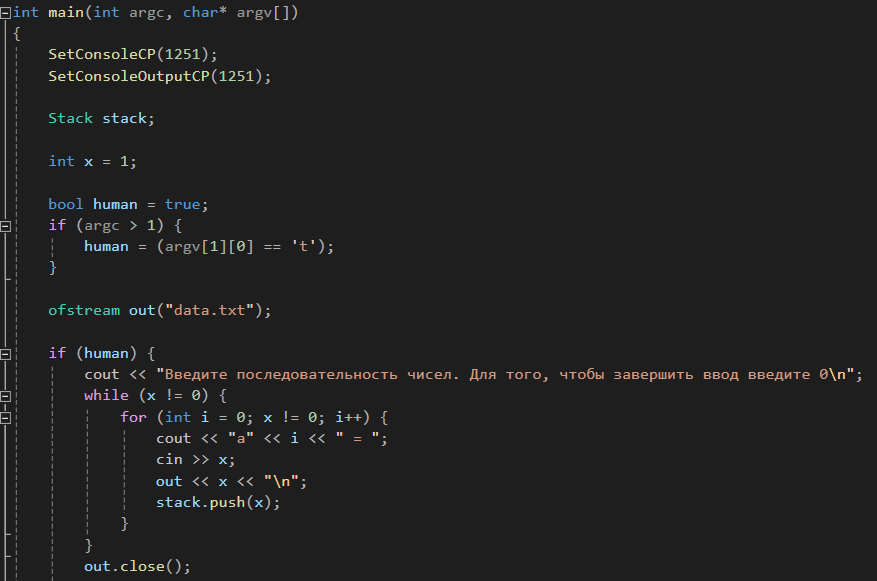


Рисунок 1.9 – Функция main

## Процесс тестирования и полученных результатов

Цель тестирования.

Целью тестирования данной программы является проверка ее корректности, надежности и удобства использования. Для этого необходимо проверить, что программа:

Результаты тестирования.

Результаты тестирования данной программы показывают, что программа работает корректно, надежно и удобно. Программа прошла все тесты

Вот примеры тестов и их результатов:

**Выводы.**

В ходе выполнения лабораторной работы была разработана программа, которая позволяет:

# Список использованной литературы

1. Павловская Т.А., Чаевников В.В., Юрков Н.К. Программирование на языке С++. Электронное методическое пособие. СПб.: Издательство СПбГЭТУ «ЛЭТИ». 2014.
2. Практикум по процедурному программированию на языке C++ С. А. Ивановский, А. А. Лисс, В. П. Самойленко, О. М. Шолохова. Практикум по процедурному программированию на языке C++: учеб. пособие. СПб.: Издательство СПбГЭТУ«ЛЭТИ», 2016.
3. Справочник по стандартной библиотеке C++ (STL) // Справочный сайт learn.microsoft.com URL: https://learn.microsoft.com/ru-ru/cpp/standard-library/cpp-standard-library-reference?view=msvc-170 (дата обращения 11.02.2024).
4. Документация по языку C++ // Справочный сайт learn.microsoft.com URL: https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170 (дата обращения 11.02.2024).
5. Файловые потоки. Открытие и закрытие // Интернет платформа metanit.com URL: https://metanit.com/cpp/tutorial/8.2.php (дата обращения 11.02.2024).

# Приложение

## Код написанной программы

#include <iostream>

#include <windows.h>

using namespace std;

// Структура узла связанного списка

struct Node {

int data; // Значение, хранящееся в узле

Node\* next; // Указатель на следующий узел

};

// Класс Queue, реализующий структуру данных очередь

class Queue {

private:

Node\* head; // Указатель на первый узел очереди

Node\* tail; // Указатель на последний узел очереди

int size; // Количество элементов в очереди

public:

Queue() {

head = tail = nullptr; // Инициализация пустой очереди

size = 0;

}

~Queue() {

clear(); // Освобождение памяти при удалении объекта

}

void queue(int data) { // Функция для добавления элемента в очередь

Node\* temp = new Node(); // Создание нового узла

temp->data = data; // Присваивание значения новому узлу

temp->next = nullptr; // Установка следующего указателя на nullptr

if (head == nullptr) { // Если очередь пуста

head = temp; // Новый узел становится головой

}

else {

tail->next = temp; // Новый узел добавляется в конец очереди

}

tail = temp; // Обновление указателя на последний узел

size++; // Увеличение размера очереди

}

int unqueue() { // Функция для извлечения элемента из очереди

if (head == nullptr) {

throw runtime\_error("Стек пуст!"); // Выбрасывается исключение, если очередь пуста

}

Node\* temp = head; // Сохранение указателя на первый узел

int data = temp->data; // Получение значения первого узла

head = head->next; // Перемещение головы на следующий узел

delete temp; // Освобождение памяти первого узла

size--; // Уменьшение размера очереди

return data; // Возвращение значения извлеченного элемента

}

void insertBeforeNegative() { // Функция для вставки 1 перед отрицательными элементами

Node\* curr = head; // Текущий указатель, начиная с головы

Node\* prev = nullptr; // Предыдущий указатель, начиная с nullptr

while (curr != nullptr) { // Проход по всем узлам очереди

if (curr->data < 0) { // Если текущий элемент отрицательный

Node\* temp = new Node(); // Создание нового узла со значением 1

temp->data = 1;

temp->next = curr; // Новый узел указывает на текущий

if (prev == nullptr) { // Если это первый отрицательный элемент

head = temp; // Новый узел становится головой

}

else {

prev->next = temp; // Иначе, новый узел вставляется перед текущим

}

prev = temp; // Обновление предыдущего указателя

size++; // Увеличение размера очереди

}

else {

prev = curr; // Если элемент не отрицательный, обновляем предыдущий указатель

}

curr = curr->next; // Переход к следующему узлу

}

}

// Удаление из очереди всех элементов с отрицательной информационной частью

void removeNegative() {

Node\* prev = nullptr; // Указатель на предыдущий узел

Node\* curr = head; // Указатель на текущий узел

while (curr != nullptr) { // Пока не достигнут конец очереди

if (curr->data < 0) { // Если текущий узел содержит отрицательное число

Node\* temp = curr; // Сохранение указателя на текущий узел

if (prev == nullptr) { // Если текущий узел является головой

head = head->next; // Перемещение головы на следующий узел

}

else { // Если текущий узел не является головой

prev->next = curr->next; // Пропуск текущего узла

}

if (curr == tail) { // Если текущий узел является хвостом

tail = prev; // Предыдущий узел становится хвостом

}

curr = curr->next; // Переход к следующему узлу

delete temp; // Удаление текущего узла

size--; // Уменьшение размера очереди

}

else { // Если текущий узел не содержит отрицательное число

prev = curr; // Текущий узел становится предыдущим

curr = curr->next; // Переход к следующему узлу

}

}

}

int count(int data) { // Функция для подсчета количества вхождений заданного значения

int count = 0; // Счетчик вхождений

Node\* curr = head; // Текущий указатель, начиная с головы

while (curr != nullptr) { // Проход по всем узлам очереди

if (curr->data == data) { // Если значение совпадает

count++; // Увеличение счетчика

}

curr = curr->next; // Переход к следующему узлу

}

return count; // Возвращение количества вхождений

}

void clear() { // Функция для очистки очереди

Node\* curr = head; // Текущий указатель, начиная с головы

while (curr != nullptr) { // Проход по всем узлам очереди

Node\* temp = curr; // Сохранение текущего узла

curr = curr->next; // Переход к следующему узлу

delete temp; // Освобождение памяти текущего узла

}

head = nullptr; // Установка головы в nullptr

tail = nullptr; // Установка хвоста в nullptr

size = 0; // Обнуление размера очереди

}

void outPut() { // Функция для вывода очереди

Node\* curr = head; // Текущий указатель, начиная с головы

while (curr != nullptr) { // Проход по всем узлам очереди

cout << curr->data << " "; // Вывод значения текущего узла

curr = curr->next; // Переход к следующему узлу

}

cout << endl; // Перевод строки после вывода очереди

}

};

int main()

{

SetConsoleCP(1251); // Установка кодировки консоли для ввода

SetConsoleOutputCP(1251); // Установка кодировки консоли для вывода

Queue q; // Создание объекта очереди

string choice; // Переменная для хранения выбора пользователя

int data; // Переменная для хранения ввода пользователя

while (true) { // Цикл, пока пользователь не выберет выход

cin >> choice; // Ввод выбора пользователя

if (choice == "insert") {

cin >> data; // Ввод значения

q.queue(data); // Добавление элемента в очередь

}

else if (choice == "front") {

cout << q.unqueue() << endl; // Извлечение элемента из очереди

}

else if (choice == "insertBeforeNegative") { // Если выбрана вставка 1 перед отрицательными элементами

q.insertBeforeNegative();

}

else if (choice == "removeNegative") { // Если выбрано удаление отрицательных элементов

q.removeNegative();

}

else if (choice == "count") { // Если выбран подсчет вхождений

cin >> data; // Ввод значения

cout << q.count(data) << endl; // Вывод результата

}

else if (choice == "clear") { // Если выбрана очистка очереди

q.clear();

cout << "Очередь очищена" << endl; // Сообщение об успешной очистке

}

else if (choice == "stop") { // Если выбран выход

break;

}

else { // Если выбрано неверное значение

cout << "Неверный выбор, попробуйте еще раз" << endl; // Сообщение об ошибке

break;

}

}

}