Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №11.2**

Дисциплина: Основы алгоритмизации и программирования

Тема: “Динамические структуры данных”

Вариант 15

Выполнил:

Студент гр. ИВТ-20-2б

Чувашев Максим Алексеевич

Проверил:

Доцент кафедры ИТАС  
Полякова О.А

Пермь, 2021

**Цель работы**

Получить практические навыки работы со стеками;

**Постановка задачи**

1. Сформировать стек. Тип информационного поля указан в варианте. (Тип информационного поля double. Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0.)
2. Распечатать полученную структуру.
3. Выполнить обработку структуры в соответствии с заданием.
4. Распечатать полученный результат.

**Анализ задачи**

1. Используемые типы данных.
   1. Структура Stack используется для хранения информационного поля типа int, а также для хранения адреса предыдущего элемента стека.

struct Stack

{

double inf; // информационное поле типа double

Stack\* prev; // поле содержащее адрес предыдущего элемента

};

* 1. Тип int используется для хранения размера стека.

int size;

* 1. Тип double используется в переменной которая является информационным полем нашего стека.

double inf; // информационное поле типа double

* 1. Тип stack<double> используется для хранения стека в реализации стека через библиотеки STL.

stack<double> st;

1. Действия над используемыми данными.
   1. С данными типа int, а именно с размером, происходит ввод этих данных пользователем, а так же осуществляется проверка с помощью цикла while.

cin >> \*size;

while (\*size < 1)

* 1. С данными типа double, происходит проверка на отрицательность и инициализация пользователем данной переменной.

if (ptr->inf < 0)

* 1. С данными типа Stack происходит:
     1. Создание стека
     2. Добавление элементов с 0 после отрицательных
     3. Вывод стека на экран
  2. С данными типа stack<double> происходит:
     1. Создание стека
     2. Добавление элементов с 0после отрицательных
     3. Вывод стека на экран

1. Вид данных.
   1. Стек реализован в программе с помощью структуры и STL.
2. Структура.
   1. В программе используется структура Stack, которая реализует стек. Поле inf имеет тип double и является информационным полем. Поле prev, хранит адрес предыдущего элемент начиная сверху.

struct Stack

{

double inf; // информационное поле типа double

Stack\* prev; // поле содержащее адрес предыдущего элемента

};

1. Ввод и вывод.
   1. Ввод и вывод всех данных осуществляется в консоль через операторов cin и cout соответственно.

cout << "Введите размер стека: \t";

cin >> \*size;

1. Действия для решения задачи.
   1. Реализация структуры Stack.

struct Stack

{

double inf; // информационное поле типа double

Stack\* prev; // поле содержащее адрес предыдущего элемента

};

* 1. Функция, которая задает размер стека, одинакова как при реализации через структуру и через библиотеки STL.

void InitSize(int \*size) // функция для ввода размера стека

{

cout << "Введите размер стека: \t";

cin >> \*size;

while (\*size < 1)

{

cout << "Введен некорректный размер стека!" << endl;

cout << "Введите размер стека: \t";

cin >> \*size;

}

}

* 1. Функция, которая создает стек.
     1. Реализация через структуру. Функция создает, проходится по всему размеру массива, создает элементы стека, просит пользователя ввести информационное поле в каждом элемент стека, а в поле, которое хранит адрес, хранится адрес на предыдущий элемент. Подробное описание работы кода указано в реализации кода.

Stack\* InitSTACK(int size) // функция создания стека

{

Stack\* top; // создание указателя на верхний элемент стека

top = new Stack; // выделение динамической памяти для стека

cout << "Введите 1 элемент стека:\t";

cin >> top->inf; // ввод пользоватемем значения информационного поля

top->prev = NULL; // указатель на предыдущий элемент верхнего элемента указывает на NULL

for (int i = 2; i <= size; i++) // проходимся от 2 до размера массива

{

Stack\* h = new Stack; // выделяем динамическую память под следующие элементы стека

cout << "Введите "<< i << " элемент стека:\t";

cin >> h->inf; // заполнение информационного поля очередного элемента стека

h->prev = top; // указатель на предыдущий элемент указывает на запомненный элемент

top = h; // верхним становится новый элемент, переходим к нему

}

return top; // возвращаем указатель на верхний элемент стека

}

* + 1. Реализация через библиотеки STL. Функция проходится по всему размеру стека, просит пользователя ввести элемент, а затем данный элемент кладется в конец стека. В результате последний элемент, который поместиться в стек, станет самым верхним элементом.

void StackInit(stack<double>& st, int size) // функция создания стека

{

double a; // выделение память под верменную переменную

for (int i = 0; i < size; i++) // проходимся от 0 до размера стека

{

cout << "Введите " << i + 1 << " элемент стека:\t";

cin >> a; // пользователь указывает значение информационного поля для каждого элемента стека

st.push(a); // элемент кладется в стек

}

}

* 1. Функция, которая выводит стек в консоль.
     1. Реализация через структуру. Сначала делается проверка на существование стека. Далее программа проходится с помощью цикла while, выводит информационное поле каждого элемента стека.

void PrintStack(Stack\* top) // фукнция печати стека в консоль

{

if (top == NULL) // проверка на существование стека

{

cout << "Стек пуст!" << endl;

}

else

{

Stack\* ptr = top; // создаем временную переменную, и приравниваем ее к верхнему элементу стека

while (ptr != NULL) // проходимся по всем элементам стека, пока не пройдемся по всем элементам стека

{

cout << ptr->inf << " "; // выводим информационное поле элемента стека в консоль

ptr = ptr->prev; // переходим к предыдущему элементу стека

}

cout << endl;

}

}

* + 1. Реализация через библиотеки STL. Сначала функция проверяет стек на существование. Далее, функция проходится по всему размеру стека, выводит верхний элемент стека, далее достает верхний элемент стека, и так до тех пор, пока элементы существуют.

void PrintStack(stack<double> st, int size) // функция печати всех элементов стека

{

cout << endl << endl;

if (size == 0) // проверка размера стека на 0

{

cout << "Стек пуст!\n";

}

else

{

for (int i = 0; i < size; i++) // прозодимся во всем элементам стека

{

cout << st.top() << " "; // выводим каждый элемент стека в консоль

st.pop(); // достаем верхний элемент из стека

}

}

}

* 1. Функция, изменяющая стек.
     1. Реализация через структуру. Функция создает временную переменную, которая хранит самый верхний элемент. Так же создает еще одну временную переменную, которая будет хранить старую переменную. Функция проходится по всем элементам стека, и, если информационное поле, которое меньше 0 заходит в цикл. Создается новый элемент стека, информационное поле которого равно 0. Подробная работа функции описана в комментариях к коду.

void EditStack(Stack\*& top) // функция изменения стека

{

Stack\* ptr = top; // создаем временную переменную и приравниваем ее к верхнему элементу стека

Stack\* old = nullptr; // создаем временную переменную которая будет запоминать старый элемент

while (ptr != NULL) // проверям на существование текущего элемента

{

if (ptr->inf < 0) // если информационное поле элемента стека меньше нуля

{

Stack\* h = new Stack; // выделяем динамическую память под новую переменную

h->inf = 0; // в данной переменной информационное поле будет равно 0

if (old != nullptr) { // проверяем равняется ли запомненная перменная nullptr

old->prev = h; // в поле которое указывает на предыдущий элемент указываем на новый элемент

}

else {

top = h; // верхним элементом становится новый элемент

}

h->prev = ptr; // в поле, которое указываем на предыдущий элемент, указываем на следующий элемент

}

old = ptr; // переходим к следующему элементу стека

ptr = ptr->prev; // передвигаем верменный указатель на предыщущий элемент

}

}

* + 1. Реализация через библиотеки STL. Функция, создает временный стек, который будет хранить развернутый стек. Функция проходится с помощью цикла for проверяет на отрицательность, и кладет новый элемент, значение которого равно 0. И кладет его после отрицательного элемента.

void EditStack(stack<double> &st, int \*size) // функция редактирования стека

{

stack<double> Dst; // выделтение память под временный стек

for (int i = 0; i < \*size; i++) // проходимя по всему стеку и заполняем новый стек, только данный стек будет развернут по отношению к первому

{

Dst.push(st.top()); // первый элемент становится последним в новом стеке и так далее

st.pop(); // вынимаем верхний элемент старого стека, в итоге старый стек станет пустым

}

for (int i = 0; i < \*size; i++) // проходимся по всем элементам нового стека

{

if (Dst.top() < 0) // проверяем все элементы стека на отрицательность

{

st.push(Dst.top()); // кладем верхний элемент нового стека в старый стек

st.push(0); // после этого кладем новый элемент равный 0

}

else

{

st.push(Dst.top()); // кладем верхний элемент нового стека в старый стек

}

Dst.pop(); // достаем из нового стека верхний элемент

}

\*size = st.size(); // кладем верхний элемент нового стека в старый стек изменяем значение размера стека на получившийся размер

}

**Код программы реализации через структуры**

#include<iostream>

using namespace std;

struct Stack

{

double inf; // информационное поле типа double

Stack\* prev; // поле содержащее адрес предыдущего элемента

};

void InitSize(int \*size) // функция для ввода размера стека

{

cout << "Введите размер стека: \t";

cin >> \*size;

while (\*size < 1)

{

cout << "Введен некорректный размер стека!" << endl;

cout << "Введите размер стека: \t";

cin >> \*size;

}

}

Stack\* InitSTACK(int size) // функция создания стека

{

Stack\* top; // создание указателя на верхний элемент стека

top = new Stack; // выделение динамической памяти для стека

cout << "Введите 1 элемент стека:\t";

cin >> top->inf; // ввод пользоватемем значения информационного поля

top->prev = NULL; // указатель на предыдущий элемент верхнего элемента указывает на NULL

for (int i = 2; i <= size; i++) // проходимся от 2 до размера массива

{

Stack\* h = new Stack; // выделяем динамическую память под следующие элементы стека

cout << "Введите "<< i << " элемент стека:\t";

cin >> h->inf; // заполнение информационного поля очередного элемента стека

h->prev = top; // указатель на предыдущий элемент указывает на запомненный элемент

top = h; // верхним становится новый элемент, переходим к нему

}

return top; // возвращаем указатель на верхний элемент стека

}

void PrintStack(Stack\* top) // фукнция печати стека в консоль

{

if (top == NULL) // проверка на существование стека

{

cout << "Стек пуст!" << endl;

}

else

{

Stack\* ptr = top; // создаем временную переменную, и приравниваем ее к верхнему элементу стека

while (ptr != NULL) // проходимся по всем элементам стека, пока не пройдемся по всем элементам стека

{

cout << ptr->inf << " "; // выводим информационное поле элемента стека в консоль

ptr = ptr->prev; // переходим к предыдущему элементу стека

}

cout << endl;

}

}

void EditStack(Stack\*& top) // функция изменения стека

{

Stack\* ptr = top; // создаем временную переменную и приравниваем ее к верхнему элементу стека

Stack\* old = nullptr; // создаем временную переменную которая будет запоминать старый элемент

while (ptr != NULL) // проверям на существование текущего элемента

{

if (ptr->inf < 0) // если информационное поле элемента стека меньше нуля

{

Stack\* h = new Stack; // выделяем динамическую память под новую переменную

h->inf = 0; // в данной переменной информационное поле будет равно 0

if (old != nullptr) { // проверяем равняется ли запомненная перменная nullptr

old->prev = h; // в поле которое указывает на предыдущий элемент указываем на новый элемент

}

else {

top = h; // верхним элементом становится новый элемент

}

h->prev = ptr; // в поле, которое указываем на предыдущий элемент, указываем на следующий элемент

}

old = ptr; // переходим к следующему элементу стека

ptr = ptr->prev; // передвигаем верменный указатель на предыщущий элемент

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

int size;

InitSize(&size);

Stack\* top = InitSTACK(size);

PrintStack(top);

EditStack(top);

PrintStack(top);

}

**Код программы реализации через библиотеки STL**

#include <iostream>

#include <stack> // подключаем библиотеку stack

using namespace std;

void InitSize(int\* size) // функция для ввода размера стека

{

cout << "Введите размер стека: \t";

cin >> \*size;

while (\*size < 1)

{

cout << "Введен некорректный размер стека!" << endl;

cout << "Введите размер стека: \t";

cin >> \*size;

}

}

void StackInit(stack<double>& st, int size) // функция создания стека

{

double a; // выделение память под верменную переменную

for (int i = 0; i < size; i++) // проходимся от 0 до размера стека

{

cout << "Введите " << i + 1 << " элемент стека:\t";

cin >> a; // пользователь указывает значение информационного поля для каждого элемента стека

st.push(a); // элемент кладется в стек

}

}

void EditStack(stack<double> &st, int \*size) // функция редактирования стека

{

stack<double> Dst; // выделтение память под временный стек

for (int i = 0; i < \*size; i++) // проходимя по всему стеку и заполняем новый стек, только данный стек будет развернут по отношению к первому

{

Dst.push(st.top()); // первый элемент становится последним в новом стеке и так далее

st.pop(); // вынимаем верхний элемент старого стека, в итоге старый стек станет пустым

}

for (int i = 0; i < \*size; i++) // проходимся по всем элементам нового стека

{

if (Dst.top() < 0) // проверяем все элементы стека на отрицательность

{

st.push(Dst.top()); // кладем верхний элемент нового стека в старый стек

st.push(0); // после этого кладем новый элемент равный 0

}

else

{

st.push(Dst.top()); // кладем верхний элемент нового стека в старый стек

}

Dst.pop(); // достаем из нового стека верхний элемент

}

\*size = st.size(); // кладем верхний элемент нового стека в старый стек изменяем значение размера стека на получившийся размер

}

void PrintStack(stack<double> st, int size) // функция печати всех элементов стека

{

cout << endl << endl;

if (size == 0) // проверка размера стека на 0

{

cout << "Стек пуст!\n";

}

else

{

for (int i = 0; i < size; i++) // прозодимся во всем элементам стека

{

cout << st.top() << " "; // выводим каждый элемент стека в консоль

st.pop(); // достаем верхний элемент из стека

}

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

int size;

InitSize(&size);

stack<double> st;

StackInit(st, size);

PrintStack(st, size);

EditStack(st, &size);

PrintStack(st, size);

}

**Блок схема реализация через структуры**











**Блок схема реализация через библиотеки STL**







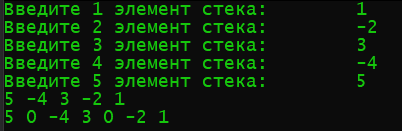
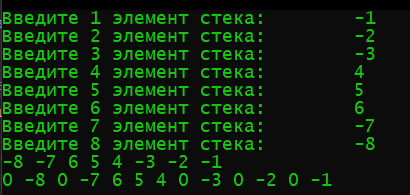
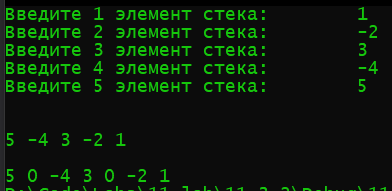
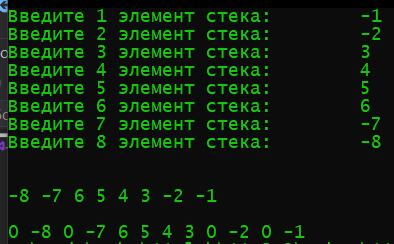
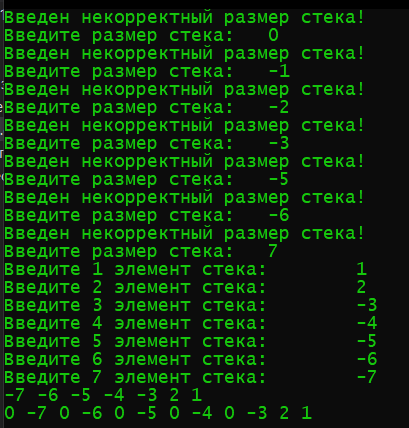








**Скриншоты результатов работы программы**

1. Нормальный ввод.
   1. Реализация через структуры
      1. 
      2. 
   2. Реализация через библиотеку STL
      1. 
      2. 
2. Ввод с ошибками
   1. Реализация через структуры
      1. 
   2. Реализация через библиотеку STL
      1. 