Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Сравнение скорости работы стека из библиотеки STL и собственной реализации через односвязный список»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2023, Минск

**Содержание**

1. Предисловие … 3
2. Реализация стека с помощью односвязного списка… 4
3. Сравнение скорости добавления чисел … 5
4. Сравнение скорости удаления чисел … 7
5. Выводы … 8

**Предисловие**

В данном реферате будет реализована такая полустатическая структура данных, как стек, на основе односвязного списка. Также будут проведены сравнения времени работы основных функций: добавления и удаления элемента.

**1. Реализация стека с помощью односвязного списка**

Структуру и прототипы функций поместим в заголовочный файл **Mystack.h**.

#pragma once

struct Mystack {

long long int num;

Mystack\* next = nullptr;

};

void push(Mystack\*&, long long int );

void pop(Mystack\*&);

int top(Mystack\*&);

bool empty(Mystack\*&);

int size(Mystack\*);

Сами функции поместим в отдельный файл **Mystack\_func.cpp**.

#include <iostream>

#include<Windows.h>

#include <string>

#include <fstream>

#include “Mystack.h”

using namespace std;

void push(Mystack\*& st, long long int n) {

Mystack\* newst = new Mystack;

newst->num = n;

newst->next = st;

st = newst;

}

void pop(Mystack\*& st) {

if (st == NULL) {

exit(-1);

}

else {

Mystack\* newst = st;

st = st->next;

delete newst;

}

}

int top(Mystack\*& st) {

if (st == NULL) {

exit(-1);

}

else {

return st->num;

}

}

bool empty(Mystack\*& st) {

return (st == NULL);

}

int size(Mystack \*st) {

long long int s = 0;

while (st) {

s++;

st = st->next;

}

return s;

}

**2. Сравнение скорости добавления чисел**

Будем сравнивать скорость добавления чисел в собственный стек и стек из библиотеки STL. Проведем 3 исследования с разным количеством чисел. Для более точного результата будем делать по 3 попытки и находить среднее время.

#include <iostream>

#include<Windows.h>

#include <conio.h>

#include <stack>

#include “Mystack.h”

#include <ctime>

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n;

cin >> n;

long long int c;

Mystack \*a = nullptr;

clock\_t start = clock();

for (int i = 0; i < n; i++) {

cin >> c;

push(a, c);

}

clock\_t end = clock();

double duration = (float)(end – start) / CLOCKS\_PER\_SEC;

printf(“Время Mystack = %.5f\n”, duration);

stack<int> b;

start = clock();

for (int i = 0; i < n; i++) {

cin >> c;

b.push(c);

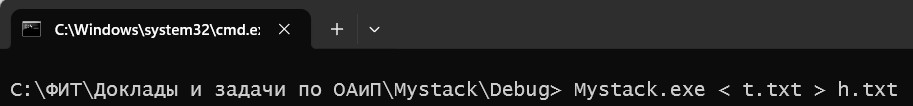
}

end = clock();

duration = (float)(end – start) / CLOCKS\_PER\_SEC;

printf(“Время stack = %.5f\n”, duration);

}

Для более удобной работы будем передавать файл с числами, используя перенаправление ввода/вывода.

Результаты:

1)100000 чисел:

Среднее время: **Mystack** = 0.15567 **stack** = 0.20467

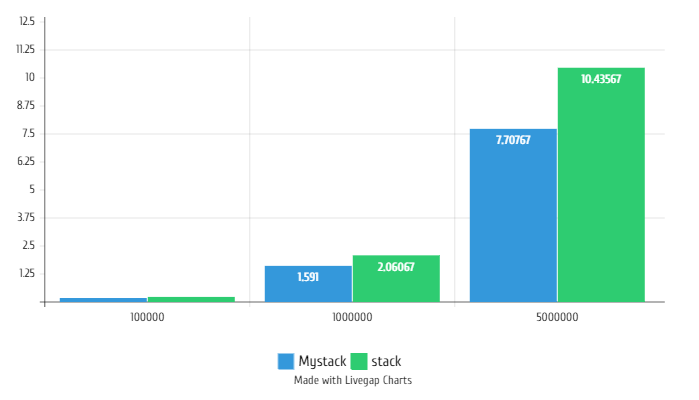
2)1000000 чисел:

Среднее время: **Mystack** = 1.59100 **stack** = 2.06067

3)5000000 чисел:

Среднее время: **Mystack** = 7.70767 **stack** = 10.43567

На основании результатов построим график.



Как видно, функция **push** для добавления элементов в стек работает быстрее в собственной реализации.

**3. Сравнение скорости удаления чисел**

Будем сравнивать скорость удаления чисел из собственного стека и стека из библиотеки STL. Проведем 3 исследования с разным количеством чисел. Для более точного результата будем делать по 3 попытки и находить среднее время.

#include <iostream>

#include<Windows.h>

#include <conio.h>

#include <stack>

#include “Mystack.h”

#include <ctime>

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n;

cin >> n;

long long int c;

Mystack \*a = nullptr;

for (int i = 0; i < n; i++) {

cin >> c;

push(a, c);

}

clock\_t start = clock();

for (int i = 0; i < n; i++) {

pop(a);

}

clock\_t end = clock();

double duration = (float)(end – start) / CLOCKS\_PER\_SEC;

printf(“Время Mystack = %.5f\n”, duration);

stack<int> b;

for (int i = 0; i < n; i++) {

cin >> c;

b.push(c);

}

start = clock();

for (int I = 0; I < n; i++) {

b.pop();

}

end = clock();

duration = (float)(end – start) / CLOCKS\_PER\_SEC;

printf(“Время stack = %.5f\n”, duration);

}

Снова будем использовать перенаправление ввода/вывода для передачи чисел.

Результаты:

1)100000 чисел:

Среднее время: **Mystack** = 0.00900 **stack** = 0.03133

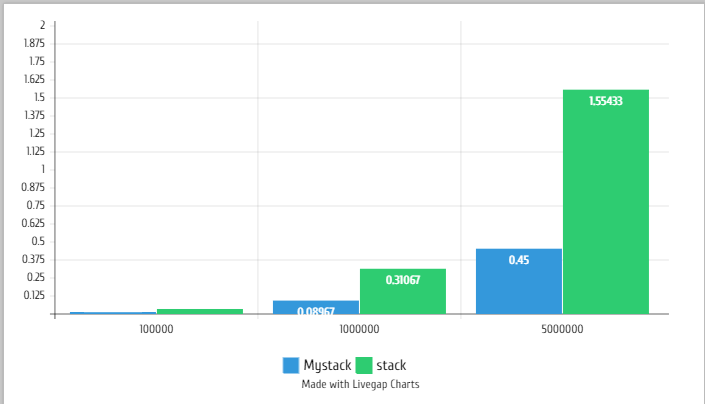
2)1000000 чисел:

Среднее время: **Mystack** = 0.08967 **stack** = 0.31067

3)5000000 чисел:

Среднее время: **Mystack** = 0.45000 **stack** = 1.55433

На основании результатов построим график.



Как видно, функция **pop** для удаления элементов из стека работает быстрее в собственной реализации.

**Выводы**

На основании проведенных исследований было выявлено, что функции в собственной реализации стека работают быстрее, чем функции для стека из библиотеки STL.