Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Сравнение скорости работы очереди из библиотеки STL и собственной реализации»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2023, Минск

**Содержание**

1. Предисловие … 3
2. Реализация очереди… 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 очередь. Также будут проведены сравнения времени работы основных функций: добавления и удаления элемента.

**1. Реализация очереди**

Структуры и прототипы функций поместим в заголовочный файл **Myqueue.h**

#pragma once

#include <iostream>

#include<Windows.h>

#include <conio.h>

struct Node {

int num;

Node\* next = NULL;

};

struct Myqueue {

long long int size = 0;

Node\* first = NULL;

Node\* last = NULL;

};

void push(Myqueue \*&, int);

void pop(Myqueue\*&);

int front(Myqueue\*&);

int back(Myqueue\*&);

long long int size(Myqueue\*&);

bool empty(Myqueue\*&);

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

#include "Myqueue.h"

void push(Myqueue\*& q, int a) { // функция добавления элемента

if (q->first == NULL) {

q->first = q->last = new Node;

q->first->num = q->last->num = a;

}

else {

q->last = q->last->next = new Node;

q->last->num = a;

}

q->size++;

}

void pop(Myqueue\*& q) { // функция удаления элемента

if (q->first == NULL) {

exit(-1);

}

else {

Node\* t = q->first;

q->first = q->first->next;

if (q->first == NULL) {

q->last = NULL;

}

delete t;

q->size--;

}

}

int front(Myqueue\*& q) { // функция возвращения значения первого элемента

if (q->first == NULL) {

exit(-1);

}

else {

return q->first->num;

}

}

int back(Myqueue\*& q) { // функция возвращения значения последнего элемента

if (q->last == NULL) {

exit(-1);

}

else {

return q->last->num;

}

}

long long int size(Myqueue\*& q) { // функция возвращения размера очереди(количества элементов)

return q->size;

}

bool empty(Myqueue \*& q) { // функция проверки на пустоту

return q->first == NULL;

}

**2. Сравнение скорости добавления чисел**

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

#include <queue>

#include "Myqueue.h"

#include <ctime>

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n;

cin >> n;

Myqueue\* q = new Myqueue;

long long int c;

clock\_t start = clock();

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

cin >> c;

push(q, c);

}

clock\_t end = clock();

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

printf("Время Myqueue = % .5f\n", duration);

queue<int> b;

start = clock();

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

cin >> c;

b.push(c);

}

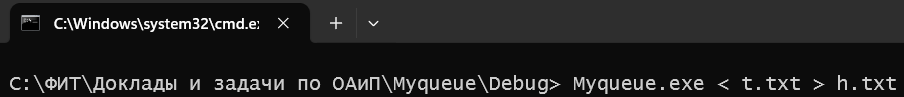
end = clock();

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

printf("Время queue = % .5f\n", duration);

}

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



Результаты:

1)100000 чисел:

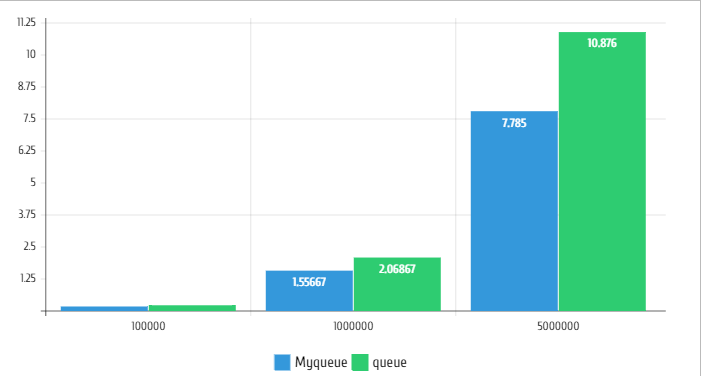
Среднее время: **Myqueue** = 0.15700 **queue** = 0.204333

2)1000000 чисел:

Среднее время: **Myqueue** = 1.55667 **queue** = 2.06867

3)5000000 чисел:

Среднее время: **Myqueue** = 7.78500 **queue** = 10.87600

На основании результатов построим график.

По графику хорошо видно, что функция **push** для добавления элементов в очередь работает быстрее в собственной реализации.

**3. Сравнение скорости удаления чисел**

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

#include <queue>

#include "Myqueue.h"

#include <ctime>

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n;

cin >> n;

Myqueue\* q = new Myqueue;

long long int c;

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

cin >> c;

push(q, c);

}

clock\_t start = clock();

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

pop(q);

}

clock\_t end = clock();

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

printf("Время Myqueue = % .5f\n", duration);

queue<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 queue = % .5f\n", duration);

}

Снова будем использовать перенаправление ввода/вывода для передачи чисел.

Результаты:

1)100000 чисел:

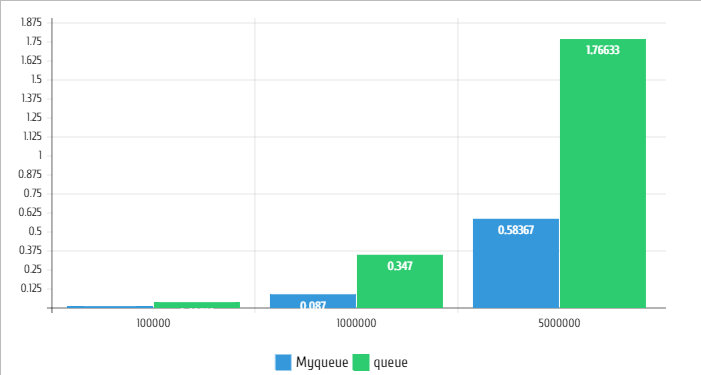
Среднее время: **Myqueue** = 0.00900 **queue** = 0.03533

2)1000000 чисел:

Среднее время: **Myqueue** = 0.08700 **queue** = 0.34700

3)5000000 чисел:

Среднее время: **Myqueue** = 0.58367 **queue** = 1.76633

На основании результатов построим график.

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

**Выводы**

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