Навчально-науковий комплекс “Інститут прикладного системного аналізу”

при Національному технічному університеті України “КПІ”

Кафедра математичних методів системного аналізу

РОБОТИ КОМП’ЮТЕРНОГО ПРАКТИКУМУ

З КУРСУ “ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ”

Виконав: студент 2-го курсу

групи КА-04

Штріккер Д.Я.

Прийняв: Шолохов О.В.

**Мета роботи:** Навчитись коректно користуватися параметризованими класами, параметризованими функціями та механізмом виключних ситуацій. Ознайомитись з основними засобами бібліотеки STL і вміти їх правильно застосовувати.

**Завдання (варіант 8):**

* Побудувати свій контейнер uses\_priority\_queue, який мав би поведінку, еквівалентну стандартному priority\_queue із STL щодо компонування елементів.
* Побудувати функції:
  + визначення довжини черги;
  + додавання нового елемента в чергу;
  + вилучення учасників із кінця (по черзі).

**Лістинг програми:**

Github (разом із історією створення): <https://github.com/dindin28/task_4>

**Класс “uses\_priority\_queue”**

**Header file:**

#ifndef \_TASK\_4\_INCLUDE\_TASK\_4\_USES\_PRIORITY\_QUEUE\_H\_

#define \_TASK\_4\_INCLUDE\_TASK\_4\_USES\_PRIORITY\_QUEUE\_H\_

#include <functional>

#include <algorithm>

#include <vector>

template <class T, class Compare = std::less<T>>

class uses\_priority\_queue

{

public:

uses\_priority\_queue();

uses\_priority\_queue(const T \*pointer, size\_t size);

uses\_priority\_queue(const uses\_priority\_queue &copy);

uses\_priority\_queue<T, Compare> &operator=(const uses\_priority\_queue &copy);

bool empty() const;

size\_t size() const;

void push(const T value);

void pop();

T &operator[](const int index);

private:

size\_t size\_;

T \*pointer\_;

};

#include <uses\_priority\_queue\_impl.cpp> //Source file

#endif //Header Guard

**Source file:**

// This is a personal academic project. Dear PVS-Studio, please check it.

// PVS-Studio Static Code Analyzer for C, C++ and C#: http://www.viva64.com

#ifndef \_TASK\_4\_SRC\_USES\_PRIORITY\_QUEUE\_IMPL\_H\_

#define \_TASK\_4\_SRC\_USES\_PRIORITY\_QUEUE\_IMPL\_H\_

#include <task\_4/uses\_priority\_queue.h>

template <class T, class Compare>

uses\_priority\_queue<T, Compare>::uses\_priority\_queue()

: size\_(0), pointer\_(nullptr){};

template <class T, class Compare>

uses\_priority\_queue<T, Compare>::uses\_priority\_queue(const T \*pointer, size\_t size)

: size\_(size)

{

pointer\_ = new T[size];

for (int i = 0; i < size; i++)

{

pointer\_[i] = pointer[i];

}

}

template <class T, class Compare>

uses\_priority\_queue<T, Compare>::uses\_priority\_queue(const uses\_priority\_queue &copy)

: size\_(copy.size\_)

{

pointer\_ = new T[copy.size\_];

for (int i = 0; i < size\_; i++)

{

pointer\_[i] = copy.pointer\_[i];

}

}

template <class T, class Compare>

uses\_priority\_queue<T, Compare> &uses\_priority\_queue<T, Compare>::operator=(const uses\_priority\_queue &copy)

{

size\_ = copy.size\_;

for (int i = 0; i < size\_; i++)

{

pointer\_[i] = copy.pointer\_[i];

}

return \*this;

}

template <class T, class Compare>

bool uses\_priority\_queue<T, Compare>::empty() const

{

return (size\_ > 0) ? (false) : (true);

}

template <class T, class Compare>

size\_t uses\_priority\_queue<T, Compare>::size() const

{

return size\_;

}

template <class T, class Compare>

void uses\_priority\_queue<T, Compare>::push(const T value)

{

++size\_;

std::vector<T> v;

for (int i = 0; i < size\_ - 1; i++)

{

v.push\_back(pointer\_[i]);

}

v.push\_back(value);

std::sort(v.begin(), v.end(), Compare());

T \*new\_pointer = new T[v.size()];

for (int i = 0; i < v.size(); i++)

{

new\_pointer[i] = v.at(i);

}

delete[] pointer\_;

pointer\_ = new\_pointer;

} //Function (push)

template <class T, class Compare>

void uses\_priority\_queue<T, Compare>::pop()

{

if (size\_ > 0)

{

--size\_;

if (size\_ == 0)

{

delete[] pointer\_;

pointer\_ = nullptr;

}

else

{

T \*new\_pointer = new T[size\_];

for (int i = 0; i < size\_; i++)

{

new\_pointer[i] = pointer\_[i];

}

delete[] pointer\_;

pointer\_ = new\_pointer;

} //Condition (size\_ == 0)

} //Condition(size\_ > 0)

} //Function (pop)

template <class T, class Compare>

T &uses\_priority\_queue<T, Compare>::operator[](const int index)

{

return pointer\_[index];

}

#endif //Header Guard

**Main file:**

// This is a personal academic project. Dear PVS-Studio, please check it.

// PVS-Studio Static Code Analyzer for C, C++ and C#: http://www.viva64.com

#include <task\_4/uses\_priority\_queue.h>

#include <functional>

#include <iostream>

template <typename T, class Compare = std::less<T>>

void QueueDemonstration()

{

int ch = -1;

uses\_priority\_queue<T, Compare> queue;

while (ch != 0)

{

do

{

std::system("clear");

for (int i = 0; i < queue.size(); i++)

{

std::cout << i + 1 << ") " << queue[i] << std::endl;

}

std::cout << "uses\_priority\_queue::empty(): " << (queue.empty() ? "true" : "false") << std::endl;

std::cout << "uses\_priority\_queue::size(): " << queue.size() << std::endl;

std::cout << std::endl;

std::cout << "1)push()" << std::endl;

std::cout << "2)pop()" << std::endl;

std::cout << "0)exit" << std::endl;

std::cin >> ch;

} while (ch != 0 && ch != 1 && ch != 2);

switch (ch)

{

case (1):

{

std::cout << "Enter new value: ";

T variable;

std::cin >> variable;

queue.push(variable);

break;

}

case (2):

{

queue.pop();

break;

}

}

}

}

int main()

{

QueueDemonstration<int>();

return 0;

}

**Результати роботи програми:**

****

**Висновок:** Навчилися коректно користуватися параметризованими класами, параметризованими функціями та механізмом виключних ситуацій. Ознайомитись з основними засобами бібліотеки STL і правильно їх застосували. Перевантажили оператор індексації та детально ознайомилися з бібліотекою priority\_queue із STL.