ОТЧЕТ  
ПО ЛАБОРАТОРНОЙ РАБОТЕ №7  
ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ» Вариант №4

Факультет: АВТФ Преподаватель: Новицкая Ю.В.

Группа: АВТ-208

Студент(ы): Нестеренко М.К.

Паршин Н.С.

Цель работы:

Ознакомиться с созданием шаблонов функций и классов. Изучить написание контейнерных шаблонных классов и их применение для построения различных структур данных.

Задание:

Задание представляет собой типовую задачу по разработке шаблонов стандартных структур данных. Протестировать структуру данных. В качестве хранимых объектов использовать встроенные типы С++ (int и char).

**Вариант 4**

Структура данных: циклическая очередь (динамический массив). Способ хранения объектов: ссылки на объекты. Размерность структуры данных: параметр шаблона. Операция: включение в конец очереди. Операция: исключение из очереди.

Исходные коды модулей проекта:

/**/cyclic\_queue\_funcs.h**

#pragma once

#include "../Header\_files/cyclic\_queue.h"

template <typename T>

cyclic\_queue<T>::cyclic\_queue() {

queue = nullptr;

size = 0;

front = -1;

rear = -1;

}

template <typename T>

cyclic\_queue<T>::cyclic\_queue(size\_t Size) {

queue = new T\*[Size];

size = Size;

front = -1;

rear = -1;

}

template <typename T>

cyclic\_queue<T>::~cyclic\_queue() {

if (isEmpty()) {

delete[] queue;

return;

}

for (int i = front; i <= rear; i++) {

delete queue[i];

}

delete[] queue;

}

template <typename T>

bool cyclic\_queue<T>::isFull() {

if (front == 0 && rear == size-1) {

return true;

}

return false;

}

template <typename T>

bool cyclic\_queue<T>::isEmpty() {

if (front == -1) {

return true;

}

return false;

}

template <typename T>

void cyclic\_queue<T>::enQueue(T \*item) {

if (isEmpty()) {

queue[0] = item;

front = 0;

rear = 0;

return;

}

if (isFull()) {

cout << "Queue is full, unable to add element\n";

return;

}

rear = (rear + 1) % size;

queue[rear] = item;

}

template <typename T>

void cyclic\_queue<T>::deQueue() {

if (isEmpty()) {

cout << "Queue is already empty\n";

return;

}

if (front == rear) {

front = -1;

rear = -1;

return;

}

front = (front + 1) % size;

}

template <typename T>

size\_t cyclic\_queue<T>::getSize() {

return size;

}

template <typename T>

T\*\* cyclic\_queue<T>::getQueue() {

return queue;

}

template <typename T>

void cyclic\_queue<T>::print() {

if (isEmpty()) {

cout << "Queue is clear\n";

return;

}

for (int i = front; i <= rear; i++) {

cout << queue[i] << endl;

}

}

**// cyclic\_queue\_funcs.h**

#pragma once

#include "../Header\_files/cyclic\_queue.h"

template <typename T>

cyclic\_queue<T>::cyclic\_queue() {

queue = nullptr;

size = 0;

front = -1;

rear = -1;

}

template <typename T>

cyclic\_queue<T>::cyclic\_queue(size\_t Size) {

queue = new T\*[Size];

size = Size;

front = -1;

rear = -1;

}

template <typename T>

cyclic\_queue<T>::~cyclic\_queue() {

if (isEmpty()) {

delete[] queue;

return;

}

for (int i = front; i <= rear; i++) {

delete queue[i];

}

delete[] queue;

}

template <typename T>

bool cyclic\_queue<T>::isFull() {

if (front == 0 && rear == size-1) {

return true;

}

return false;

}

template <typename T>

bool cyclic\_queue<T>::isEmpty() {

if (front == -1) {

return true;

}

return false;

}

template <typename T>

void cyclic\_queue<T>::enQueue(T \*item) {

if (isEmpty()) {

queue[0] = item;

front = 0;

rear = 0;

return;

}

if (isFull()) {

cout << "Queue is full, unable to add element\n";

return;

}

rear = (rear + 1) % size;

queue[rear] = item;

}

template <typename T>

void cyclic\_queue<T>::deQueue() {

if (isEmpty()) {

cout << "Queue is already empty\n";

return;

}

if (front == rear) {

front = -1;

rear = -1;

return;

}

front = (front + 1) % size;

}

template <typename T>

size\_t cyclic\_queue<T>::getSize() {

return size;

}

template <typename T>

T\*\* cyclic\_queue<T>::getQueue() {

return queue;

}

template <typename T>

void cyclic\_queue<T>::print() {

if (isEmpty()) {

cout << "Queue is clear\n";

return;

}

for (int i = front; i <= rear; i++) {

cout << queue[i] << endl;

}

}

**// main.cpp**

#include "../Header\_files/cyclic\_queue.h"

#include "../Header\_files/cyclic\_queue\_funcs.h"

using namespace std;

void char\_test();

void int\_test();

int main() {

while (true) {

cout << "Menu\n" << "1. Launch string test\n" << "2. Launch int test\n" << "-1 for exit\n";

int flag;

cin >> flag;

if (flag == -1) {

break;

}

switch (flag)

{

case -1:

break;

case 1:

char\_test();

break;

case 2:

int\_test();

break;

default:

break;

}

}

return 0;

}

void char\_test() {

cout << "How long queue for strings?: ";

size\_t string\_queue\_size;

cin >> string\_queue\_size;

cyclic\_queue<char> string\_queue = cyclic\_queue<char>(string\_queue\_size);

char \*tmp = new char[50];

cout << "Enter string: ";

cin >> tmp;

string\_queue.enQueue(tmp);

string\_queue.print();

cout << "One more: ";

char \*tmp\_1 = new char[100];

cin >> tmp\_1;

string\_queue.enQueue(tmp\_1);

string\_queue.print();

string\_queue.deQueue();

string\_queue.deQueue();

string\_queue.print();

}

void int\_test() {

cout << "How long queue for integers?: ";

size\_t int\_queue\_size;

cin >> int\_queue\_size;

cyclic\_queue<int> string\_queue = cyclic\_queue<int>(int\_queue\_size);

int tmp;

cout << "Enter number: ";

cin >> tmp;

string\_queue.enQueue(&tmp);

string\_queue.print();

cout << "One more: ";

int tmp\_1;

cin >> tmp\_1;

string\_queue.enQueue(&tmp\_1);

string\_queue.print();

string\_queue.deQueue();

string\_queue.deQueue();

string\_queue.print();

}

Выводы:

В ходе выполнения лабораторной работы на тему "Универсальность. Применение шаблонов функций и классов" были изучены основные принципы использования шаблонов функций в программировании и классах. Шаблоны позволяют создавать универсальные конструкции, которые могут работать с разными типами данных без необходимости создания отдельной функции или класса для каждого конкретного типа.

Одним из основных преимуществ шаблонов функций является возможность создавать одну функцию, которая может обрабатывать различные типы данных. Шаблоны классов позволяют создавать универсальные классы, которые могут работать с различными типами данных. В ходе выполнения работы было обнаружено, что использование шаблонов функций и классов значительно упрощает и ускоряет процесс программирования, так как позволяет избежать дублирования кода и унифицировать его структуру. Также применение шаблонов позволяет создавать более гибкие и масштабируемые программы, которые могут быть легко адаптированы для работы с различными типами данных.

Таким образом, использование шаблонов функций и классов является неотъемлемой частью современного программирования, обеспечивая универсальность, гибкость и эффективность разработки программного обеспечения.