

## 1. Постановка задачи

Создать родительский класс "Очередь" с функциями инициализации очереди, добавления элемента в очередь и извлечения элемента из очереди.

Создать метод создания копии очереди. Результатом должен стать новый экземпляр класса "Очередь", состоящий из элементов (копий элементов) исходной очереди. Порядок следования элементов должен быть сохранен.

Создать функцию слияния двух очередей. Результатом должна быть очередь, состоящая из элементов первой очереди и второй очереди. Порядок следования элементов должен быть сохранен.

На основе родительского класса "Очередь" создать дочерний класс "Очередь1" с функциями нахождения и отображения на экране требуемого в соответствии с вариантом задания значения.

Вариант 17. Нахождение первого элемента, большего среднего арифметического.

Важно учитывать при реализации:

- Необходимо выполнить разделение на h и cpp файлы для каждого класса. h файлы содержат определение, cpp файлы содержат реализацию. Функция main обязана располагаться в отдельном cpp файле.

- Элемент очереди содержит данные (целое число) и ссылку на предыдущий элемент. Элемент реализовать с помощью класса или структуры.

- Реализовать динамическое выделение памяти для очереди/элемента очереди и динамическое удаление из памяти при извлечении элемента очереди.

- Заранее число элементов очереди неизвестно, очередь заполняется постепенно пользователем программы.

- Не принимается очередь, реализованная с помощью STL контейнеров или в виде статического массива.

- Данные очереди обязаны находиться в области доступа private базового класса.

- Реализовать пользовательское меню: 1 – Добавление элемента очереди; 2 – Извлечение элемента очереди; 3 – Вывод очереди на экран; 4 – Вычисление требуемого значения (в соответствии с вариантом задания); 5 – Создание копии очереди; 6 – Слияние оригинальной очереди с копией и вывод результата на экран; 7 – Выход из программы.

Реализовать задание для трех режимов доступа при наследовании:

- public,
- protected,
- private.

Объяснить разницу работы программы при разных модификаторах доступа при наследовании. Продемонстрировать работу программы.

## 2. Формализация

Исходя из требований, программа будет иметь следующую структуру:

Файл lab1.cpp с определяющим хедером lab1.h будут содержать интерфейс пользователя. К ним будут подключены родительский класс MyQueue и три класса наследника MyQueuePubl, MyQueueProt, MyQueuePriv, реализующие наследование с разными модификаторами доступа от класса MyQueue. Дочерние классы будут реализовывать метод printTask, объединяющий в себе функции возвращения нужного по заданию значения и вывода его на экран. Для линковки будет использована система CMake.

### 3. Исходный код

main/main.cpp

```
using namespace std;

#include "../libs/libs.h"
#include "../lab1/lab1.h"

int main(){
    lab1();

    return 0;
}
```

libs/libs.h

```
#ifndef LIBS_H_
#define LIBS_H_
#pragma once
#include <iostream>

#endif
```

lab1/lab1.h

```
#ifndef LAB1_H_
#define LAB1_H_

int lab1();
#include "../libs/libs.h"
#include "MyQueue.h"
#include "MyQueuePubl.h"
#include "MyQueueProt.h"
#include "MyQueuePriv.h"

#endif
```

lab1/lab1.cpp

```
#include "lab1.h"
using namespace std;

int lab1(){

    int symb = 0;
    bool flag = false; //false = original, true = copy
```

```

MyQueuePubl *orig = new MyQueuePubl();
MyQueuePubl *cpy, *uni;
system("cls");

do{
    cout<<"-----\n\n";
    cout<<"Queue inheritance logic program\n\n";
    cout<<"-----\n\n";

    cout<<"List of commads:\n";
    cout<<"1. Add new element to current queue\n";
    cout<<"2. Pop element from current queue\n";
    cout<<"3. Print curent queue\n";
    cout<<"4. Print task value from current queue\n";
    cout<<"5. Creaty copy of current queue\n";
    cout<<"6. Unite copy and original queues\n";
    cout<<"7. End\n";

    cout<<"Creating copy makes current queue copied
queue.\n";
    cout<<"Uniting makes current queue original queue.\n";

    cout<<"Please enter num of command to use it:"<<endl;
    cin>>symb;
    if(symb==EOF) symb = 7;
    system("cls");

    switch(symb){
        case 1:{
            cout<<"Integer number: ";
            cin>>symb;
            if(!flag) orig->add(symb);
            else cpy->add(symb);
        }
        break;
        case 2:{
            if(!flag) cout<<orig->pop()<<endl;
            else cout<<cpy->pop()<<endl;
        }
        break;
        case 3:{
            if(!flag) orig->print();
            else cpy->print();
        }
    }
}

```

```

        break;
        case 4:{
            if(!flag) orig->printTask();
            else cpy->printTask();
        }
        break;
        case 5:{
            if(!flag){
                flag = true;
                cpy=static_cast<MyQueuePubl*>
                    (orig->copy());
            }
            else cout<<"You already use copy!\n";
        }
        break;
        case 6:{
            if(!flag) cout<<"You already use original!\n";
            else{
                flag = false;

                uni=static_cast<MyQueuePubl*>
                    (unite(orig,cpy));
                delete orig;
                delete cpy;
                orig=static_cast<MyQueuePubl*>
                    (uni->copy());
                delete uni;
                orig->print();

            }
        }
        break;
        case 7:{
            delete orig;
            if(flag) delete cpy;
            symb = EOF;
            std::cout<<"End."<<std::endl;

        }
        break;
    }

}while(symb != EOF);

```

```
        return 0;
    }
}
```

lab1/MyQueue.h

```
#ifndef MY_QUEUE_H
#define MY_QUEUE_H

#include "../libs/libs.h"
class MyQueue
{
private:
    class Node{
    public:
        int value;
        Node *next;
        Node(int value);
    };

    Node *head;

public:
    MyQueue();
    bool add(int value);
    int get();
    int pop();
    bool print();
    MyQueue* copy();
    bool isEmpty();
    ~MyQueue();
};
```

```
MyQueue* unite(MyQueue* first, MyQueue *second);
```

```
#endif
```

lab1/MyQueue.cpp

```
#include "MyQueue.h"

MyQueue::Node::Node(int value = 0){
    this->value = value;
```

```

        this->next = nullptr;
    }

    MyQueue::MyQueue(){
        head = nullptr;
    }

    MyQueue::~MyQueue(){
        if(isEmpty()) return;
        Node *finder = head;

        Node *deleter = finder;

        while(finder->next!=nullptr){
            deleter = finder;
            finder = finder->next;
            delete deleter;
        }
        delete finder;
    }

    bool MyQueue::add(int value){
        if(isEmpty()){
            head = new Node(value);
            return true;
        }
        Node *finder = head;

        while(finder->next!=nullptr) finder = finder->next;

        Node *element = new Node(value);
        finder->next = element;
        return true;
    }

    int MyQueue::get(){
        if(isEmpty())
            return head->value;
        else
            return -1;
    }

    int MyQueue::pop(){
        if(isEmpty()) return -1;
        int result = head->value;

```

```

        Node *finder = head;
        head = head->next;
        delete finder;
        return result;
    }

    bool MyQueue::print(){
        if(isEmpty()){
            std::cout<<"Queue is empty!"<<std::endl;
            return false;
        }
        Node *finder = head;
        while(finder->next!=nullptr){
            std::cout<< finder->value<<" ";
            finder = finder->next;
        }
        std::cout<<finder->value<<std::endl;
        return true;
    }

    bool MyQueue::isEmpty(){
        if(head == nullptr) return true;
        else return false;
    }

    MyQueue* MyQueue::copy(){
        if(isEmpty()) return new MyQueue();
        MyQueue* result = new MyQueue();

        Node *finder = head;
        while(finder->next!=nullptr){
            result->add(finder->value);
            finder = finder->next;
        }
        result->add(finder->value);

        return result;
    }

    MyQueue* unite(MyQueue* first, MyQueue *second){
        MyQueue *result = first->copy();
        MyQueue *tmp = second->copy();
        while(!tmp->isEmpty()){
            result->add(tmp->pop());
        }
    }

```

```

    }
    delete tmp;
    return result;
}

```

lab1/MyQueuePubl.h

```

#ifndef MY_QUEUE_PUBL_H
#define MY_QUEUE_PUBL_H

#include "MyQueue.h"

class MyQueuePubl : public MyQueue{
    using MyQueue::MyQueue;
    public:
        int printTask();

};

#endif

```

lab1/MyQueuePubl.cpp

```

#include "MyQueuePubl.h"

int MyQueuePubl::printTask(){
    MyQueue *cop = this->copy();
    int len = 0;
    int sum = 0;
    while(!cop->isEmpty()){
        len++;
        sum+=cop->pop();
    }
    delete cop;

    cop = this->copy();
    double ar = 0;
    if(len!=0) ar = sum/len;
    else{
        std::cout<<"Queue is empty!"<<std::endl;
        return false;
    }
    int tmp;
    while(!cop->isEmpty()){
        tmp = cop->pop();
    }
}

```



```

        if(tmp>ar){
            std::cout<<"Task value:"<<tmp<<std::endl;
            break;
        }
    }
    delete cop;
    return tmp;
}

```

lab1/MyQueueProt.h

```

#ifndef MY_QUEUE_PROT_H
#define MY_QUEUE_PROT_H

#include "MyQueue.h"

class MyQueueProt : protected MyQueue{
    using MyQueue::MyQueue;
    public:
        int printTask();
};

#endif

```

lab1/MyQueueProt.cpp

```

#include "MyQueueProt.h"

int MyQueueProt::printTask(){
    MyQueue *cop = this->copy();
    int len = 0;
    int sum = 0;
    while(!cop->isEmpty()){
        len++;
        sum+=cop->pop();
    }
    delete cop;

    cop = this->copy();
    double ar = 0;
    if(len!=0) ar = sum/len;
    else{
        std::cout<<"Queue is empty!"<<std::endl;
        return false;
    }
}

```

```

        int tmp;
        while(!cop->isEmpty()){
            tmp = cop->pop();
            if(tmp>ar){
                std::cout<<"Task value:"<<tmp<<std::endl;
                break;
            }
        }
        delete cop;
        return tmp;
    }
}

```

lab1/MyQueuePriv.h

```

#ifndef MY_QUEUE_PRIV_H
#define MY_QUEUE_PRIV_H

#include "MyQueue.h"

class MyQueuePriv : private MyQueue{
    using MyQueue::MyQueue;
public:
    int printTask();
};

#endif

```

lab1/MyQueuePriv.cpp

```

#include "MyQueuePriv.h"

int MyQueuePriv::printTask(){
    MyQueue *cop = this->copy();
    int len = 0;
    int sum = 0;
    while(!cop->isEmpty()){
        len++;
        sum+=cop->pop();
    }
    delete cop;

    cop = this->copy();
    double ar = 0;
    if(len!=0) ar = sum/len;
    else{

```

```

        std::cout<<"Queue is empty!"<<std::endl;
        return false;
    }
    int tmp;
    while(!cop->isEmpty()){
        tmp = cop->pop();
        if(tmp>ar){
            std::cout<<"Task value:"<<tmp<<std::endl;
            break;
        }
    }
    delete cop;
    return tmp;
}

```

CMakeLists.txt

```

cmake_minimum_required(VERSION 3.23.28.3)

project(cmakeproject)

add_executable(main
    main/main.cpp
    lab1/lab1.cpp
    lab1/MyQueue.cpp
    lab1/MyQueuePubl.cpp
    lab1/MyQueueProt.cpp
    lab1/MyQueuePriv.cpp
)

```

#### 4. Результаты работы программы

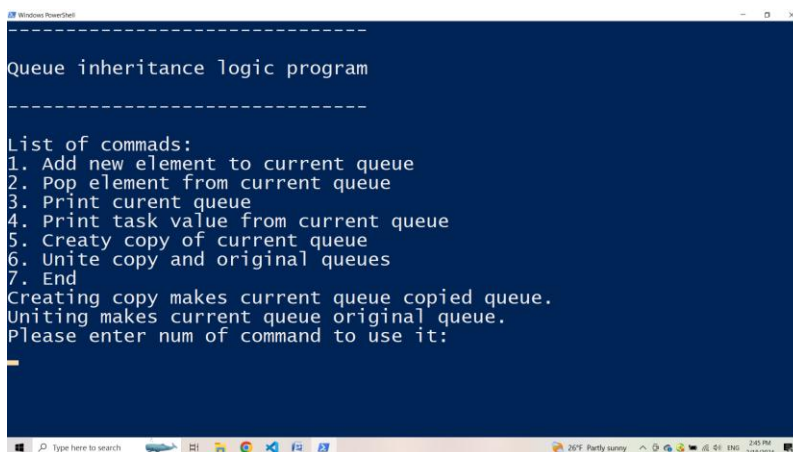
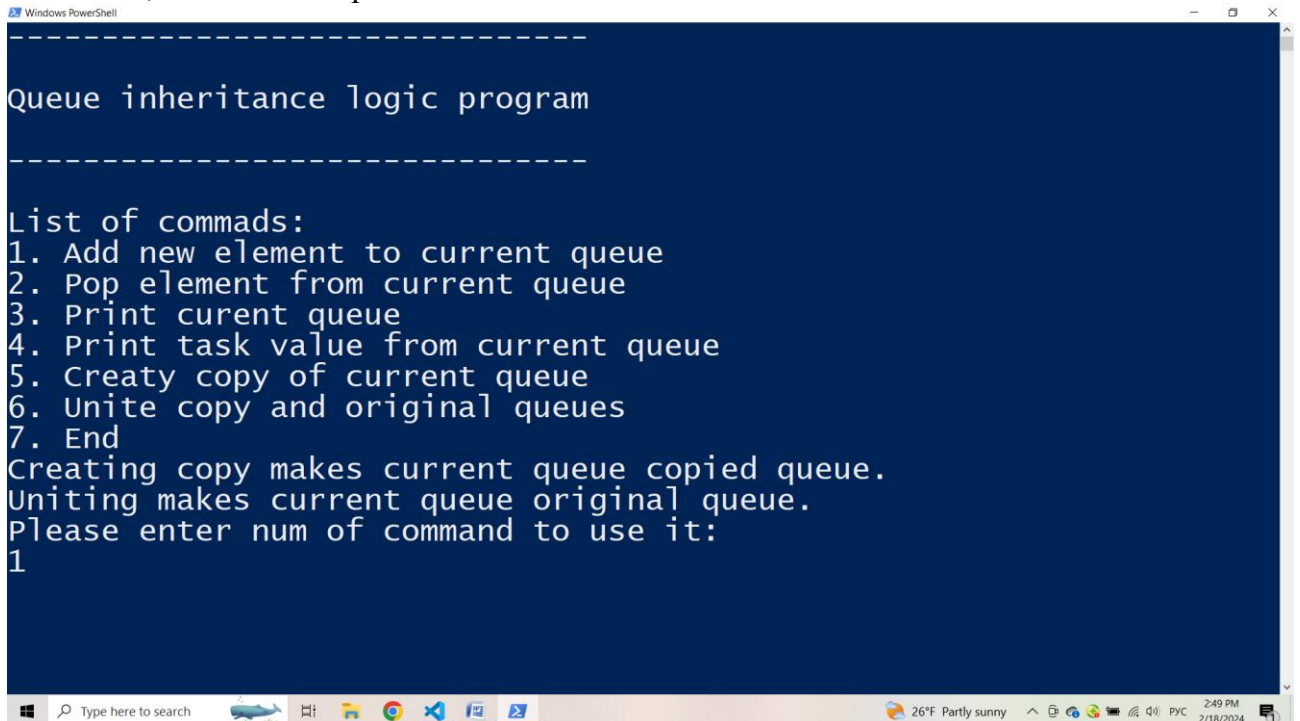


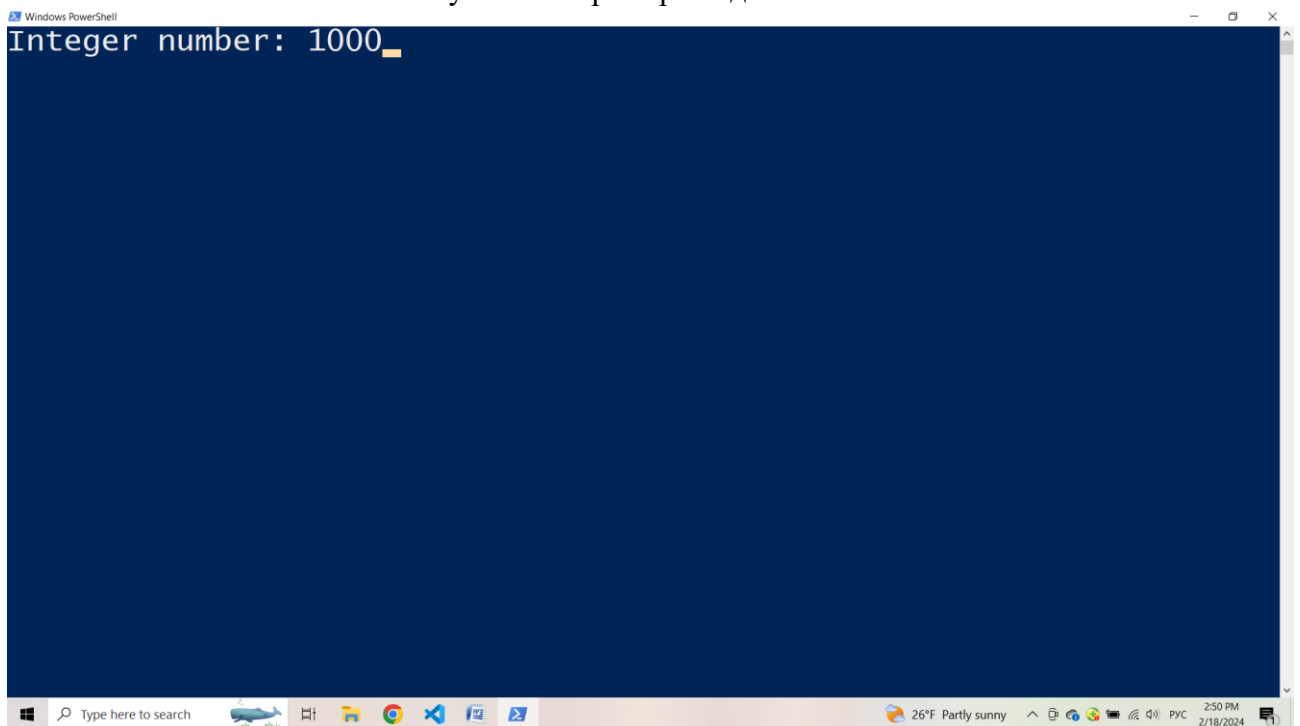
Рисунок 1 – Интерфейс программы

Пользователю доступно 7 команд. Первые 4 работают с текущей очередью, добавляют, извлекают, выводят на экран целиком или выводят лишь искомое значение соответственно.



```
-----  
Queue inheritance logic program  
-----  
  
List of commads:  
1. Add new element to current queue  
2. Pop element from current queue  
3. Print curent queue  
4. Print task value from current queue  
5. Creaty copy of current queue  
6. Unite copy and original queues  
7. End  
Creating copy makes current queue copied queue.  
Uniting makes current queue original queue.  
Please enter num of command to use it:  
1
```

Рисунок 2 – Пример ввода значений



```
Integer number: 1000_
```

Рисунок 3 – программа запрашивает вводимое число

Добавим таким же образом числа 2000 и 3000. В результате наша очередь будет иметь следующий вид.

```
Windows PowerShell
1000 2000 3000
-----
Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:
_
```

Рисунок 4 – Очередь после добавления

Извлечем элемент из очереди с помощью pop.

```
Windows PowerShell
1000
-----
Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:
_
```

Рисунок 5 – Использование pop извлекает элемент и выводит его на экран

```
Windows PowerShell
2000 3000
-----
Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:

```

Рисунок 6 – очередь после извлечения элемента.

Добавим значения 1001, 1002, 3000, 4000, 5000. Теперь очередь имеет следующий вид.

```
Windows PowerShell
2000 3000 1001 1002 3000 4000 5000
-----
Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:

```

Рисунок 7 – очередь с 7ю элементами.

Среднее арифметическое очереди  $Sr$ :

$$Sr = \frac{2000 + 3000 + 1001 + 1002 + 3000 + 4000 + 5000}{7} = 2714,7$$

Первым элементом, большим чем  $Sr$  должен быть элемент 3000.

```
Windows PowerShell
Task value:3000
-----

Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:
```

Рисунок 8 – Вывод значения по заданию

Создадим копию исходной очереди.

```
Windows PowerShell
2000 3000 1001 1002 3000 4000 5000
-----

Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:

```

Рисунок 9 – программа создала копию исходной очереди.

Попробуем еще раз вызвать создание копии.

```
Windows PowerShell
You already use copy!
-----
Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:

```

Рисунок 10 – вывод сообщения об использовании копии.

Добавим в копию элемент 27 и произведем слияние копии и оригинала.

```
Windows PowerShell
Integer number: 27
-----
Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:

```

Рисунок 11 – добавление нового элемента

Слияние очередей снова переключит нас на пользование оригинальной очередью и сотрет все данные копии.



```
Windows PowerShell
2000 3000 1001 1002 3000 4000 5000 2000 3000 1001 1002 3000 4000 50
00 27
-----
Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:
_
```

Рисунок 12 – слияние очередей.

Попробуем еще раз произвести слияние. Должно быть предупреждение о том, что копия пуста – мы уже используем оригинальную очередь.

```
Windows PowerShell
You already use original!
-----
Queue inheritance logic program
-----

List of commads:
1. Add new element to current queue
2. Pop element from current queue
3. Print curent queue
4. Print task value from current queue
5. Creaty copy of current queue
6. Unite copy and original queues
7. End
Creating copy makes current queue copied queue.
Uniting makes current queue original queue.
Please enter num of command to use it:
_
```

Рисунок 13 – Предупреждение об объединении с пустой очередью.

Завершим работу программы.

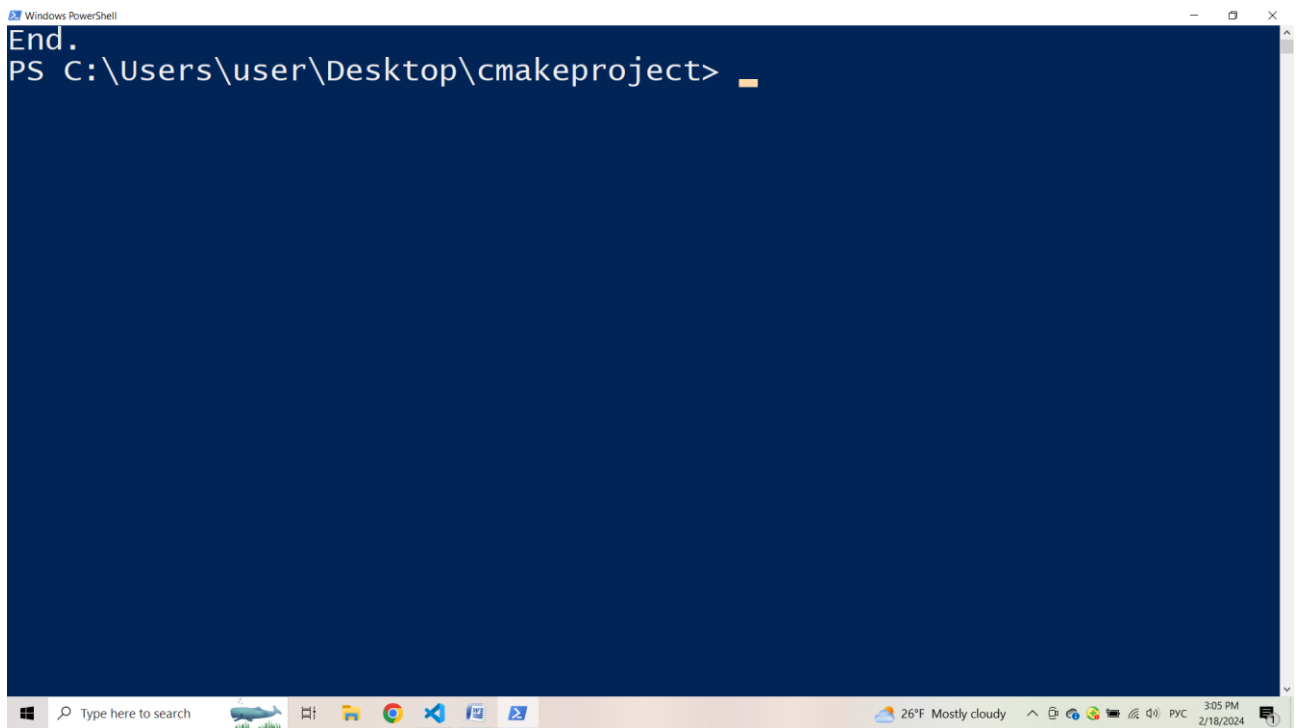


Рисунок 14 – конец работы программы.

## 5. Выводы

В процессе работы я научился работать с наследованием на языке программирования C++. Объектно реализовал класс очередь, и создал наследников с разными типами наследования. Изучил отличия наследования с разными модификаторами доступа. Реализовал пользовательский интерфейс. Результатом работы стала программа для работы с двумя очередями – оригиналом и копией. Программа работает успешно, так как ожидаемые результаты работы совпадают с фактическими.