

#include <iostream>

#include <Windows.h>

struct Queue

{

// Рэалізацыя ўсіх функцый вынесена са структуры

// Канструктар з параметрам - ініцыялізацыя чаргі з зададзеным памерам

Queue(int size);

// Канструктар капіявання - стварэнне копіі існуючай чаргі

Queue(const Queue& other);

// Дэструктар - вызваленне выдзеленай памяці

~Queue();

// Праверка, ці з'яўляецца чарга поўнай

bool isFull() const;

// Праверка, ці з'яўляецца чарга пустой

bool isEmpty() const;

// Даданне элемента ў чаргу

bool Enqueue(int item);

// Выманне элемента з чаргі

int\* Dequeue();

// Вывад ўсіх элементаў чаргі

void PrintAll() const;

// Вяртае колькасць элементаў у чарзе

size\_t SizeCount() const;

// Выдаляе зададазеную колькасць элементаў з пачатку чаргі

size\_t RemoveFront(size\_t amount);

// Знайсці максімальны элемент

int\* Max() const;

int Head; // Індэкс галавы чаргі

int Tail; // Індэкс хваста чарзе

int Size; // Памер чаргі

int\* Data; // Паказальнік на масіў дадзеных

};

// Канструктар з параметрам - ініцыялізацыя чаргі з зададзеным памерам

Queue::Queue(int size)

{

Head = Tail = 0; // Ініцыялізуем паказальнікі на пачатак і канец чаргі

Size = size + 1; // Усталёўваем памер чаргі на адзінку больш зададзенага для рэалізацыі кальцавой структуры

Data = new int[Size]; // Вылучаем памяць пад масіў дадзеных

}

// Канструктар капіравання - стварэнне копіі існуючай чаргі

Queue::Queue(const Queue& other) : Size(other.Size), Head(other.Head), Tail(other.Tail)

{

Data = new int[Size]; // Вылучаем памяць пад новы масіў дадзеных

// Капіруем дадзеныя з існуючай чаргі ў новую

for (int i = Head; i != Tail; i = (i + 1) % Size)

{

Data[i] = other.Data[i];

}

}

// Дэструктар - вызваленне выдзеленай памяці

Queue::~Queue()

{

delete[] Data; // Вызваляем памяць масіва дадзеных

}

// Праверка, ці з'яўляецца чарга поўнай

bool Queue::isFull() const

{

return (Head % Size == (Tail + 1) % Size);

}

// Праверка, ці з'яўляецца чарга пустой

bool Queue::isEmpty() const

{

return Head == Tail;

}

// Даданне элемента ў чаргу

bool Queue::Enqueue(int item)

{

if (!isFull()) // Калі чарга не поўная

{

Data[Tail] = item; // Дадаем элемент у канец чаргі

Tail = (Tail + 1) % Size; // Зрушваем паказальнік на канец чаргі

return true; // Вяртаем true, каб паказаць паспяховае даданне

}

return false; // Вяртаем false, калі чарга поўная

}

// Выманне элемента з чаргі

int\* Queue::Dequeue()

{

if (!isEmpty()) // Калі чарга не пустая

{

int\* item = &Data[Head]; // Атрымліваем паказальнік на элемент, які мы здабываем

Head = (Head + 1) % Size; // Зрушваем паказальнік на пачатак чаргі

return item; // Вяртаем паказальнік на выняты элемент

}

return nullptr; // Калі чарга была пустая, вяртаем nullptr

}

// Вывад ўсіх элементаў чаргі

void Queue::PrintAll() const

{

int current = Head;

// Праходзім па чарзе і выводім ўсе яе элементы

while (current != Tail)

{

std::cout << Data[current] << " ";

current = (current + 1) % Size;

}

std::cout << std::endl;

}

// Вяртае колькасць элементаў у чарзе

size\_t Queue::SizeCount() const

{

size\_t size = 0;

int current = Head;

// Падлічваем колькасць элементаў у чарзе

while (current != Tail)

{

size++;

current = (current + 1) % Size;

}

return size;

}

// Выдаляе зададзеную колькасць элементаў з пачатку чаргі

size\_t Queue::RemoveFront(size\_t amount)

{

size\_t count = 0;

// Выдаляем элементы з пачатку чаргі зададзеную колькасць разоў

while (!isEmpty() && amount-- != 0)

{

Dequeue();

count++;

}

return count; // Вяртаем колькасць вынятых элементаў

}

int\* Queue::Max() const

{

// Паказальнік this прадстаўляе паказальнік на бягучы аб'ект,

// для якога была выклікана функцыя

if (this->isEmpty()) // Калі чарга пустая, то вяртаем нулявы паказальнік

return nullptr;

Queue temp(\*this); // Ствараем копію чаргі, каб не змяняць зыходную

int\* max = &temp.Data[temp.Head]; // За пачатковы лік бярэм першы элемент

while (!temp.isEmpty()) // Пакуль чарга не пустая

{

int\* element = temp.Dequeue(); // Атрымліваем элемент з чаргі

if (\*element > \*max) // Захоўваем яго, калі ён большы за максімальны

{

max = &this->Data[temp.Head - 1]; // Захоўваем паказальнік на максімальны элемент з бягучага аб'екта, выкарыстоўваючы паказальнік на пачатак чаргі Head з копіі аб'екта. Толькі для таго, каб пасля выхаду з вобласці функцыі не выдаляўся элемент, на які паказвае max

}

}

return max;

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// Увод памеру чаргі

int queueSize;

std::cout << "Увядзіце памер чаргі: ";

std::cin >> queueSize;

// Стварэнне чаргі

Queue myQueue(queueSize);

int choice;

do {

// Вывад меню

std::cout << "\n====== Меню чаргі =======\n";

std::cout << "1. Дадаць элемент у чаргу\n";

std::cout << "2. Выняць элемент з чаргі\n";

std::cout << "3. Вывесці ўсе элементы чаргі\n";

std::cout << "4. Вывесці памер чаргі\n";

std::cout << "5. Выдаліць азначаную колькасць элементаў з пачатку чаргі\n";

std::cout << "6. Знайсці максімальны элемент у чарзе\n";

std::cout << "0. Выхад\n";

std::cout << "===========================\n";

std::cout << "Абярыце дзеянне: ";

std::cin >> choice;

switch (choice) {

case 1: {

int value;

std::cout << "Увядзіце значэнне для дадання: ";

std::cin >> value;

if (myQueue.Enqueue(value)) {

std::cout << "Элемент паспяхова дададзены ў чаргу.\n";

}

else {

std::cout << "Чарга поўная, немагчыма дадаць элемент.\n";

}

break;

}

case 2: {

int\* item = myQueue.Dequeue();

if (item != nullptr) {

std::cout << "Выняты элемент: " << \*item << "\n";

}

else {

std::cout << "Чарга пустая, немагчыма атрымаць элемент.\n";

}

break;

}

case 3:

std::cout << "Элементы ў чарзе: ";

myQueue.PrintAll();

break;

case 4:

std::cout << "Памер чаргі: " << myQueue.SizeCount() << "\n";

break;

case 5: {

size\_t amount;

std::cout << "Увядзіце колькасць элементаў для выдалення: ";

std::cin >> amount;

size\_t removed = myQueue.RemoveFront(amount);

std::cout << "Выдалена элементаў з пачатку чаргі: " << removed << "\n";

myQueue.PrintAll();

break;

}

case 6:

{

int\* item = myQueue.Max();

if (item != nullptr) {

std::cout << "Максімальны элемент у чарзе: " << \*item << "\n";

}

else {

std::cout << "Чарга пустая, немагчыма знайсці максімальны элемент.\n";

}

break;

}

case 0:

std::cout << "Выхад з праграмы.\n";

break;

default:

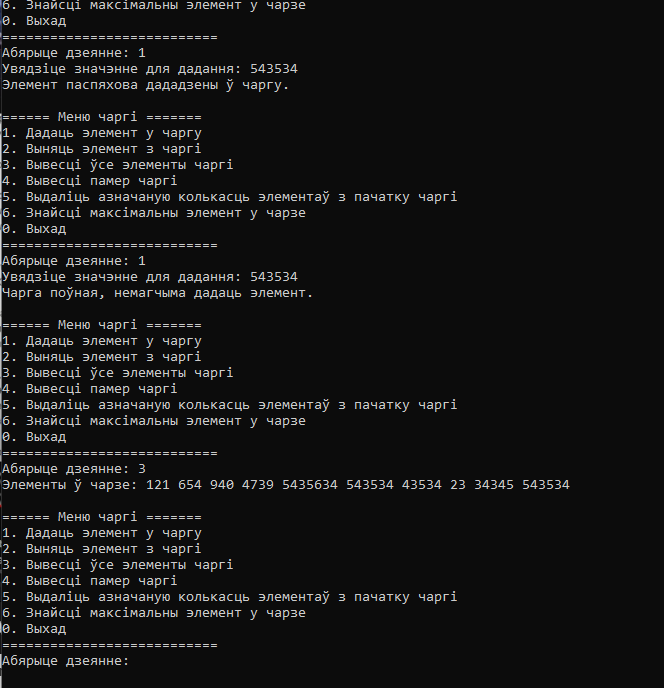
std::cout << "Некарэктны выбар, паспрабуйце зноў.\n";

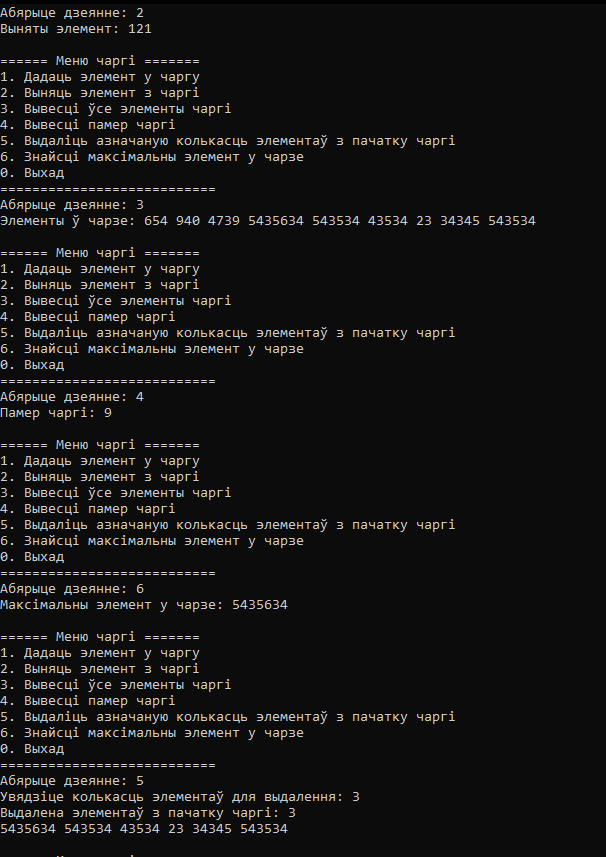
}

} while (choice != 0);

return 0;

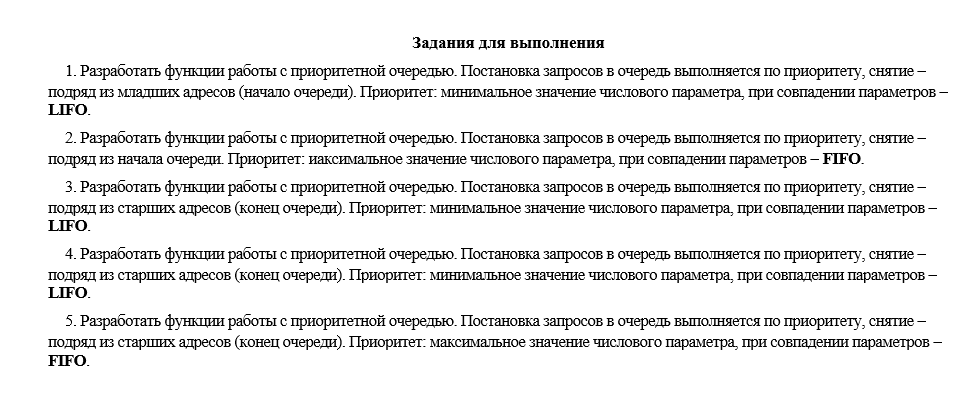
}





5. Дап. Заданні





5.1.1

#include <iostream>

#include <string>

#include <Windows.h>

using namespace std;

struct Item {

int data; // Значэнне элемента

Item\* next; // Спасылка на наступны элемент

};

// Аб'яўляем два глабальныя паказальнікі: head - на першы элемент чаргі,

// tail - на апошні элемент чаргі.

Item\* head = nullptr;

Item\* tail = nullptr;

// Функцыя isNull правярае, ці пустая чарга.

bool isNull() {

return (head == nullptr);

}

// Функцыя deletFirst выдаляе першы элемент з чаргі.

void deletFirst() {

if (isNull()) {

// Чарга пустая.

cout << "Чарга пустая" << endl;

}

else {

Item\* p = head; // Часовы паказальнік, каб не было уцечкі памяці

head = head->next;

delete p;

}

}

// Функцыя getFromHead атрымлівае значэнне першага элемента

// чаргі без яго выдалення.

void getFromHead() {

if (isNull()) {

// Чарга пустая.

cout << "Чарга пустая" << endl;

}

else {

cout << "Пачатак = " << head->data << endl;

}

}

// Функцыя insertToQueue дадае новы элемент x у чаргу.

void insertToQueue(int x) {

Item\* p = new Item;

p->data = x;

p->next = nullptr;

if (isNull()) {

// Калі чарга пустая, новы элемент становіцца

// і першым, і апошнім.

head = tail = p;

}

else {

Item\* prev = nullptr;

Item\* current = head;

// Шукаем месца для ўстаўкі новага элемента, перабіраючы ўсе элементы чаргі.

while (current != nullptr && x > current->data) { // Пакуль элемент х большы за бягучыя элементы ці чарга не скончылась

prev = current;

current = current->next;

}

if (prev == nullptr) {

// Устаўка ў пачатак чаргі (больш высокі прыярытэт).

p->next = head;

head = p;

}

else {

// Устаўка пасля элемента з ніжэйшым прыярытэтам.

if (current == nullptr) { //Калі трэба ўставіць у самы канец

tail->next = p;

tail = p;

}

else { // Калі недзде ў сярэдзіне

p->next = current;

prev->next = p;

} }

}

}

// Функцыя printQueue выводзіць усе элементы чаргі.

void printQueue() {

Item\* p = new Item;

if (isNull()) {

// Чарга пустая.

cout << "Чарга пустая" << endl;

}

else {

cout << "Чарга = ";

p = head;

while (!isNull()) { // Пакуль чарга не пустая, выводзім усе яе элементы

if (p != nullptr) {

cout << p->data << " "; cout << "-> ";

p = p->next;

}

else {

cout << "NULL" << endl;

return;

}

}

}

}

// Функцыя clrQueue чысціць чаргу, выдаляючы ўсе элементы.

void clrQueue() {

while (!isNull()) deletFirst();

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int i = 1, choice = 1, z;

while (choice != 0) {

cout << "1 - дадаць элемент" << endl;

cout << "2 - атрымаць элемент з пачатку" << endl;

cout << "3 - атрымаць элемент з пачатку" << endl;

cout << "4 - вывесці элементы" << endl;

cout << "5 - ачысціць чаргу" << endl;

cout << "0 - выхад" << endl;

cout << "Абярыце дзеянне: "; cin >> choice;

switch (choice)

{

case 1: cout << "Увядзіце элемент: "; cin >> z;

insertToQueue(z); printQueue(); break;

case 2: getFromHead(); break;

case 3: deletFirst(); break;

case 4: printQueue(); break;

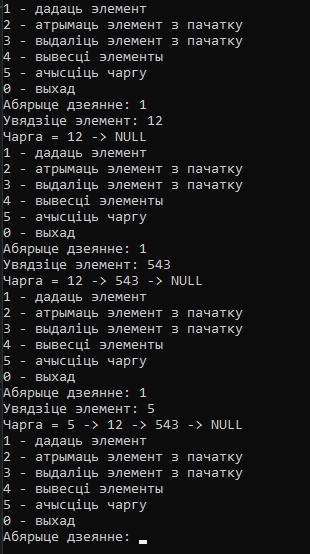
case 5: clrQueue(); break;

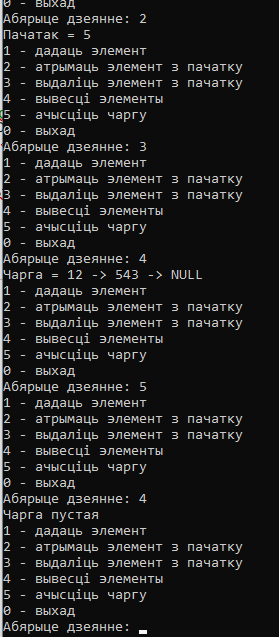
}

}

return 0;

}





5.1.2

Адрозненне толькі ў адной функцыі

У радку while (current != nullptr && x <= current->data):  
// Функцыя insertToQueue дадае новы элемент x у чаргу.

void insertToQueue(int x) {

Item\* p = new Item;

p->data = x;

p->next = nullptr;

if (isNull()) {

// Калі чарга пустая, новы элемент становіцца

// і першым, і апошнім.

head = tail = p;

}

else {

Item\* prev = nullptr;

Item\* current = head;

// Шукаем месца для ўстаўкі новага элемента, перабіраючы ўсе элементы чаргі.

while (current != nullptr && x <= current->data) { // Пакуль элемент х большы за бягучыя элементы ці чарга не скончылась

prev = current;

current = current->next;

}

if (prev == nullptr) {

// Устаўка ў пачатак чаргі (больш высокі прыярытэт).

p->next = head;

head = p;

}

else {

// Устаўка пасля элемента з ніжэйшым прыярытэтам.

if (current == nullptr) { //Калі трэба ўставіць у самы канец

tail->next = p;

tail = p;

}

else { // Калі недзде ў сярэдзіне

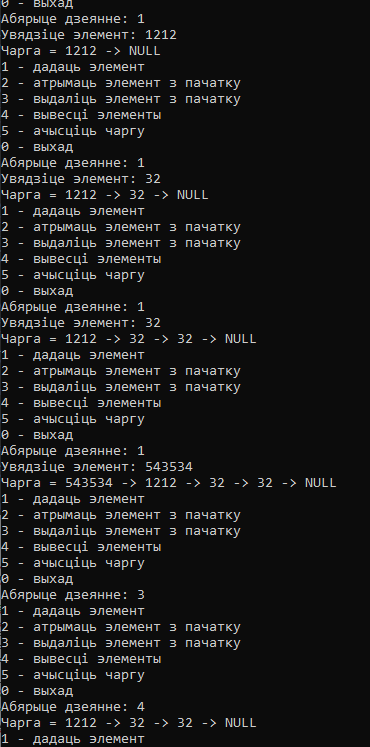
p->next = current;

prev->next = p;

} }

}

}



5.1.3, 5.1.14 (5.1.4 адзін у адзін за 5.1.3)

Адрозненне ад 5.1.1 толькі ў 2 функцыях:  
// Функцыя getFromHead атрымлівае значэнне апошнягя элемента

// чаргі без яго выдалення.

void getFromHead() {

if (isNull()) {

// Чарга пустая.

cout << "Чарга пустая" << endl;

}

else {

cout << "Канец = " << tail->data << endl; // Праз паказальнік tail, а не head

}

}

// Функцыя deletFirst выдаляе апошні элемент з чаргі.

void deletFirst() {

if (isNull()) {

// Чарга пустая.

cout << "Чарга пустая" << endl;

}

else {

Item\* p = nullptr;

Item\* current = head; // Часовы паказальнік, каб не было уцечкі памяці

while (!isNull())

{

// Калі наступны элемент - апошні

if (current->next == tail) {

current->next = nullptr; // Выдаляем яго

p = tail; // Захоўваем выдалены

tail = current; // Абнаўляем tail на новы канец

break;

}

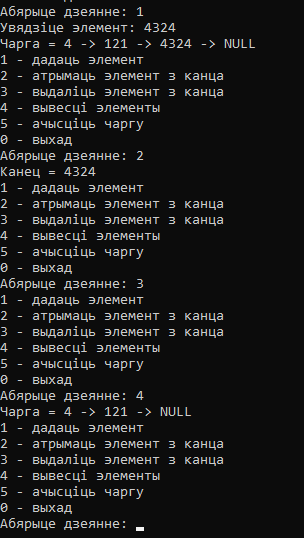
current = current->next;

}

delete p;

}

}



5.1.5

// Функцыя insertToQueue дадае новы элемент x у чаргу.

void insertToQueue(int x) {

Item\* p = new Item;

p->data = x;

p->next = nullptr;

if (isNull()) {

// Калі чарга пустая, новы элемент становіцца

// і першым, і апошнім.

head = tail = p;

}

else {

Item\* prev = nullptr;

Item\* current = head;

// Шукаем месца для ўстаўкі новага элемента, перабіраючы ўсе элементы чаргі.

while (current != nullptr && x <= current->data) { // Пакуль элемент х большы за бягучыя элементы ці чарга не скончылась

prev = current;

current = current->next;

}

if (prev == nullptr) {

// Устаўка ў пачатак чаргі (больш высокі прыярытэт).

p->next = head;

head = p;

}

else {

// Устаўка пасля элемента з ніжэйшым прыярытэтам.

if (current == nullptr) { //Калі трэба ўставіць у самы канец

tail->next = p;

tail = p;

}

else { // Калі недзде ў сярэдзіне

p->next = current;

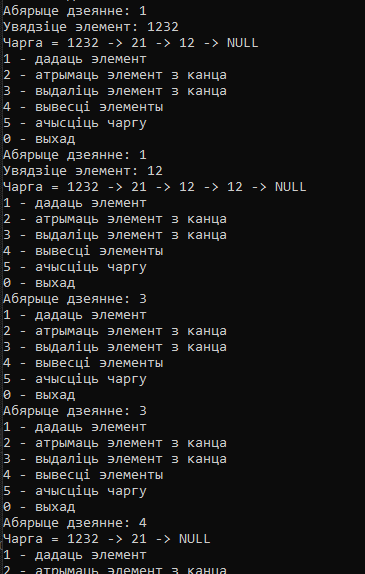
prev->next = p;

}

}

}

}



5.2



Акрамя функцыі void Queue::CycleMove() астатнія захавалісь з 4 задання, толькі тып дадзеных зменены з int на double

#include <iostream>

#include <Windows.h>

struct Queue

{

// Рэалізацыя ўсіх функцый вынесена са структуры

// Канструктар з параметрам - ініцыялізацыя чаргі з зададзеным памерам

Queue(int size);

// Дэструктар - вызваленне выдзеленай памяці

~Queue();

// Праверка, ці з'яўляецца чарга поўнай

bool isFull() const;

// Праверка, ці з'яўляецца чарга пустой

bool isEmpty() const;

// Даданне элемента ў чаргу

bool Enqueue(double item);

// Выманне элемента з чаргі

double\* Dequeue();

// Вывад ўсіх элементаў чаргі

void PrintAll() const;

// Вяртае колькасць элементаў у чарзе

size\_t SizeCount() const;

// Знайсці максімальны элемент

double\* Max() const;

void CycleMove();

int Head; // Індэкс галавы чаргі

int Tail; // Індэкс хваста чарзе

int Size; // Памер чаргі

double\* Data; // Паказальнік на масіў дадзеных

};

// Канструктар з параметрам - ініцыялізацыя чаргі з зададзеным памерам

Queue::Queue(int size)

{

Head = Tail = 0; // Ініцыялізуем паказальнікі на пачатак і канец чаргі

Size = size + 1; // Усталёўваем памер чаргі на адзінку больш зададзенага для рэалізацыі кальцавой структуры

Data = new double[Size]; // Вылучаем памяць пад масіў дадзеных

}

// Дэструктар - вызваленне выдзеленай памяці

Queue::~Queue()

{

delete[] Data; // Вызваляем памяць масіва дадзеных

}

// Праверка, ці з'яўляецца чарга поўнай

bool Queue::isFull() const

{

return (Head % Size == (Tail + 1) % Size);

}

// Праверка, ці з'яўляецца чарга пустой

bool Queue::isEmpty() const

{

return Head == Tail;

}

// Даданне элемента ў чаргу

bool Queue::Enqueue(double item)

{

if (!isFull()) // Калі чарга не поўная

{

Data[Tail] = item; // Дадаем элемент у канец чаргі

Tail = (Tail + 1) % Size; // Зрушваем паказальнік на канец чаргі

return true; // Вяртаем true, каб паказаць паспяховае даданне

}

return false; // Вяртаем false, калі чарга поўная

}

// Выманне элемента з чаргі

double\* Queue::Dequeue()

{

if (!isEmpty()) // Калі чарга не пустая

{

double\* item = &Data[Head]; // Атрымліваем паказальнік на элемент, які мы здабываем

Head = (Head + 1) % Size; // Зрушваем паказальнік на пачатак чаргі

return item; // Вяртаем паказальнік на выняты элемент

}

return nullptr; // Калі чарга была пустая, вяртаем nullptr

}

// Вывад ўсіх элементаў чаргі

void Queue::PrintAll() const

{

int current = Head;

// Праходзім па чарзе і выводім ўсе яе элементы

while (current != Tail)

{

std::cout << Data[current] << " ";

current = (current + 1) % Size;

}

std::cout << std::endl;

}

// Вяртае колькасць элементаў у чарзе

size\_t Queue::SizeCount() const

{

size\_t size = 0;

int current = Head;

// Падлічваем колькасць элементаў у чарзе

while (current != Tail)

{

size++;

current = (current + 1) % Size;

}

return size;

}

double\* Queue::Max() const

{

if (isEmpty()) // Калі чарга пустая, то вяртаем нулявы паказальнік

return nullptr;

double\* max = &Data[Head]; // За пачатковы лік бярэм першы элемент

for (int i = Head; i != Tail; i = (i + 1) % Size)

{

if (\*max < Data[i]) {

max = &Data[i];

}

}

return max;

}

double\* Queue::Max() const

{

if (isEmpty()) // Калі чарга пустая, то вяртаем нулявы паказальнік

return nullptr;

double\* max = &Data[Head]; // За пачатковы лік бярэм першы элемент

for (int i = Head; i != Tail; i = (i + 1) % Size)

{

if (\*max < Data[i]) {

max = &Data[i];

}

}

return max;

}

void Queue::CycleMove()

{

if (isEmpty() || SizeCount() == 1) {

// Няма чаго зрушваць, калі чарга пустая ці ўтрымоўвае толькі адзін элемент

return;

}

double max = \*Max();

while (Data[Head] != max) // Пакуль першым элементам у чарге не з'яўляецца максімальны

{

// Захоўваем апошні элемент, бо ён страціцца падчас выканання цыклу for

double lastElement = Data[(Tail - 1 + Size) % Size];

// Выконваем цыклічны зрух направа, лічачы цыклічныя індэксы

for (size\_t i = (Tail - 1 + Size) % Size; i!= Head; i = (i - 1 + Size) % Size) {

Data[i] = Data[(i - 1 + Size) % Size];

}

// Змяшчаем захаваны апошні элемент у пачатак чаргі

Data[Head] = lastElement;

}

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// Увод памеру чаргі

int queueSize;

std::cout << "Увядзіце памер чаргі: ";

std::cin >> queueSize;

// Стварэнне чаргі

Queue myQueue(queueSize);

int choice;

do {

// Вывад меню

std::cout << "\n====== Меню чаргі =======\n";

std::cout << "1. Дадаць элемент у чаргу\n";

std::cout << "2. Выняць элемент з чаргі\n";

std::cout << "3. Вывесці ўсе элементы чаргі\n";

std::cout << "6. Зрушыць у права\n";

std::cout << "0. Выхад\n";

std::cout << "===========================\n";

std::cout << "Абярыце дзеянне: ";

std::cin >> choice;

switch (choice) {

case 1: {

double value;

std::cout << "Увядзіце значэнне для дадання: ";

std::cin >> value;

if (myQueue.Enqueue(value)) {

std::cout << "Элемент паспяхова дададзены ў чаргу.\n";

}

else {

std::cout << "Чарга поўная, немагчыма дадаць элемент.\n";

}

break;

}

case 2: {

double\* item = myQueue.Dequeue();

if (item != nullptr) {

std::cout << "Выняты элемент: " << \*item << "\n";

}

else {

std::cout << "Чарга пустая, немагчыма атрымаць элемент.\n";

}

break;

}

case 3:

std::cout << "Элементы ў чарзе: ";

myQueue.PrintAll();

break;

case 6:

{

myQueue.PrintAll();

myQueue.CycleMove();

myQueue.PrintAll();

break;

}

case 0:

std::cout << "Выхад з праграмы.\n";

break;

default:

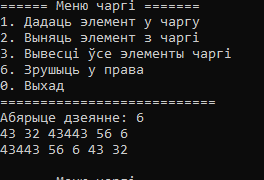
std::cout << "Некарэктны выбар, паспрабуйце зноў.\n";

}

} while (choice != 0);

return 0;

}



5.3



#include <iostream>

#include <fstream>

#include <string>

// Структура для прадстаўлення элемента чаргі

struct QueueNode {

char data;

QueueNode\* next;

// Канструктар для зручнасці стварэння элемента

QueueNode(char value) : data(value), next(nullptr) {}

};

// Структура для прадстаўлення чаргі

struct Queue {

QueueNode\* Head; // Паказальнік на пачатак чаргі

QueueNode\* Tail; // Паказальнік на канец чаргі

// Канструктар, які ініцыялізуе паказальнікі

Queue() : Head(nullptr), Tail(nullptr) {}

// Дэструктар для вызвалення памяці пры выдаленні чаргі

~Queue() {

// Пакуль чарга не пустая, выдаляем элементы

while (!isEmpty()) {

dequeue();

}

}

// Функцыя для праверкі, ці пустая чарга

bool isEmpty() {

return Head == nullptr;

}

// Функцыя для дадання элемента ў канец чаргі

void enqueue(char value) {

// Ствараем новы вузел

QueueNode\* newNode = new QueueNode(value);

// Калі чарга пустая, новы элемент становіцца і пачаткам, і канцом

if (isEmpty()) {

Head = Tail = newNode;

}

else {

// Інакш дадаем новы элемент у канец і абнаўляем паказальнік на канец

Tail->next = newNode;

Tail = newNode;

}

}

// Функцыя для выдалення элемента з пачатку чаргі

char dequeue() {

// Правяраем, ці не з'яўляецца чарга пустой

char item = '\0'; // Значэнне па змаўчанні

if (isEmpty()) {

std::cerr << "Чарга пустая. Немагчыма выканаць аперацыю dequeue." << std::endl;

return item;

}

// Атрымліваем паказальнік на першы элемент

QueueNode\* temp = Head;

item = Head->data;

// Абнаўляем паказальнік на пачатак чаргі

Head = Head->next;

// Выдаляем першы элемент і вяртаем значэнне

delete temp;

return item;

}

};

int main() {

Queue charsQueue; // Чарга для сімвалаў

Queue digitsQueue; // Чарга для лічбаў

std::ifstream filef("f.txt");

// Адкрываем і правяраем на паспяховасць адкрыцця файл для чытання

if (!filef.is\_open() || filef.fail() || filef.bad()) {

std::cerr << "Не атрымалася адкрыць файл\n";

return 1;

}

std::ofstream fileg("g.txt");

// Адкрываем і правяраем на паспяховасць адкрыцця файл для запісу

if (!fileg.is\_open() || fileg.fail() || fileg.bad()) {

std::cerr << "Не атрымалася адкрыць файл\n";

return 1;

}

std::string line;

while (std::getline(filef, line)) // Пакуль атрымаецца лічыць з файла радок

{

for (auto it = line.begin(); it != line.end(); it++) // выкарыстоўваючы ітэратары на пачатак і канец радка .begin() і .end() адпаведна, перабіраем усе сімвалы ў радку

{

if ('0' <= \*it && \*it <= '9') // Калі гэты сімвал - лічба

{

digitsQueue.enqueue(\*it); // Дадаем яго ў чаргу для лічбаў

}

else

{

charsQueue.enqueue(\*it); // Інакш у чаргу для сімвалаў

}

}

while (!charsQueue.isEmpty()) // Пакуль чарга не пустая

{

fileg << charsQueue.dequeue(); // Вымаемы першы элемент чаргі і запісваем яго ў файл

}

fileg << ' ';

while (!digitsQueue.isEmpty()) // Пакуль чарга не пустая

{

fileg << digitsQueue.dequeue(); // Вымаемы першы элемент чаргі і запісваем яго ў файл

}

fileg << '\n';

}

return 0;

}

