Учреждение образования

«БЕЛОРУССКИЙГОСУДАРСТВЕННЫЙТЕХНОЛОГИЧЕСКИЙУНИВЕРСИТЕТ»

Н. Н. Пустовалова, Н. В. Пацей

# ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

**в 2-х частях**

**Часть 2**

**Лабораторный практикум**

**Лабораторная работа № 8. Полустатические структуры данных: очереди**

**Лабораторная работа № 8. Полустатические структуры данных: очереди**

|  |  |
| --- | --- |
| **Задание** | **Программа** |
| 1. В программе, приведенной справа, демонстрируется реализация очереди на основе *односвязного списка.*  Внести изменения в программу с тем, чтобы выводились не только буквы, но и слова. |  |
| 2. Изучить способы манипуляции с элементами очереди, реализованной на основе *односвязного списка* выполнив программу, приведенную в правой части. |  |

#include<iostream>

using namespace std;

struct Number

{

int info;

Number\* next;

};

void create(Number\*\* begin, Number\*\* end, int p); //формирование элементов оче-реди

void view(Number\* begin); //вывод элементов очереди

Number\* minElem(Number\* begin); //определение минимального эле-мента

void DeltoMin(Number\*\* begin, Number\*\* p); //удаление до минимального эле-мента

int main()

{

Number\* begin = NULL, \* end, \* t;

t = new Number;

int p, size;

cout << "\nEnter size queue = ";

cin >> size;

cout << "Enter number = ";

cin >> p;

t->info = p; //первый элемент

t->next = NULL;

begin = end = t;

for (int i = 1; i < size; i++) //создание очереди

{

cout << "Enter number= ";

cin >> p;

create(&begin, &end, p);

}

cout << "\nelements of queue: \n";

if (begin == NULL) //вывод на экран

{

cout << "No elements" << endl;

}

else

{

view(begin);

}

t = minElem(begin); //определение минимального

cout << "minimum=" << t->info << endl;

DeltoMin(&begin, &t); //удаление до минимального

cout << "\nnew Queue:\n";

view(begin);

return 0;

}

void create(Number\*\* begin, Number\*\* end, int p) //Формирование элементов оче-реди

{

Number\* t = new Number;

t->next = NULL;

if (\*begin == NULL)

{

\*begin = \*end = t;

}

else

{

t->info = p;

(\*end)->next = t;

\*end = t;

}

}

void view(Number\* begin) //Вывод элементов очереди

{

Number\* t = begin;

if (t == NULL)

{

cout << "Number is empty\n";

return;

}

else

{

while (t != NULL)

{

cout << t->info << endl;

t = t->next;

}

}

}

Number\* minElem(Number\* begin) //Определение минимального элемента

{

Number\* t = begin, \* mn = nullptr;

int min;

if (t == NULL)

{

cout << "Number is empty\n";

return 0;

}

else

{

min = t->info;

while (t != NULL)

{

if (t->info <= min)

{

min = t->info;

mn = t;

}

t = t->next;

}

}

return mn;

}

void DeltoMin(Number\*\* begin, Number\*\* p) //Удаление до минимального элемента

{

Number\* t;

t = new Number;

while (\*begin != \*p)

{

t = \*begin;

\*begin = (\*begin)->next;

delete t;

}

}

|  |  |
| --- | --- |
| 3. В правой части приведен *проект*, в котором реализация очереди осуществлена на основе *динамического* *массива элементов, представленных в виде структур*.  Изменить главную функцию, включив операторы работы с функциями добавления, извлечения и вывода различных элементов. |  |

**Главный модуль:**

#include <iostream>

#include "MyQueue.h"

using namespace std;

struct myQue

{

int a;

char b;

};

void printQueue(Queue& s) // Вывод на экран с очисткой очереди

{

while (!(s.isEmpty()))

{

cout << ((myQue\*)peekQueue(s))->a << " " << ((myQue\*)peekQueue(s))->b << endl;

delQueue(s);

}

}

int main()

{

Queue q1 = createQueue(4);

myQue a1 = { 1, 'q' }, a2 = { 2, 'w' }, a3 = { 3, 'e' };

enQueue(q1, &a1);

enQueue(q1, &a2);

enQueue(q1, &a3);

myQue\* a4 = new myQue;

a4->a = 4;

a4->b = 'r';

enQueue(q1, a4);

printQueue(q1);

return 0;

}

**Модуль с функциями:**

#include "MyQueue.h"

Queue createQueue(int n) // Выделить ресурс для очереди

{

return \*(new Queue(n));

};

Queue createQueue(const Queue& pq) // Создать очередь

{

Queue\* rc = new Queue(pq.Size - 1);

rc->Head = pq.Head;

rc->Tail = pq.Tail;

for (int i = 0; i < pq.Size; i++)

{

rc->Data[i] = pq.Data[i];

}

return \*rc;

}

bool Queue::isFull() const // Очередь заполненa?

{

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

}

bool Queue::isEmpty()const // Очередь пустa?

{

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

}

bool enQueue(Queue& q, void\* x) // Добавить элемент x

{

bool rc = true;

if (rc = !q.isFull())

{

q.Data[q.Tail] = x;

q.Tail = (q.Tail + 1) % q.Size;

}

else

{

rc = false;

}

return rc;

};

void\* delQueue(Queue& q) // Удалить элемент

{

void\* rc = (void\*)MYQUEUE1\_EQE;

if (!q.isEmpty())

{

rc = q.Data[q.Head];

q.Head = (q.Head + 1) % q.Size;

}

else

{

rc = nullptr;

}

return rc;

}

void\* peekQueue(const Queue& q) // Получить первый элемент очереди

{

void\* rc = (void\*)MYQUEUE1\_EQE;

if (!q.isEmpty())

{

rc = q.Data[q.Head];

}

else

{

rc = nullptr;

}

return rc;

}

int clearQueue(Queue& q) // Очистить очередь

{

int rc;

if ((q.Tail - q.Head) >= 0)

{

rc = q.Tail - q.Head;

}

else

{

rc = (q.Size - q.Head + q.Tail + 1);

}

q.Tail = q.Head = 0;

return rc; // количество элементов до очистки

}

void releaseQueue(Queue& q) // Освободить ресурсы очереди

{

delete[] q.Data;

q.Size = 1;

q.Head = q.Tail = 0;

}

**Файл с заголовками:**

#define MYQUEUE1\_EQE 0x0000 // возврат в случае пустоты очереди

struct Queue // Блок управления очередью

{

int Head; // голова очереди

int Tail; // хвост очереди

int Size; // размер очереди (макс. колич.+1)

void\*\* Data; // хранилище данных очереди

Queue(int size) // физический размер очереди

{

Head = Tail = 0;

Data = new void\* [Size = size + 1];

}

bool isFull() const; // очередь заполненa ?

bool isEmpty()const; // очередь пустa ?

};

Queue createQueue(int n); // n – макс. количество

Queue createQueue(const Queue& pq); // создать очередь по образцу

bool enQueue(Queue& q, void\* x); // добавить x

void\* delQueue(Queue& q); // удалить элемент

void\* peekQueue(const Queue& q); // получить первый элемент

int clearQueue(Queue& q); // очистить очередь

void releaseQueue(Queue& q); // освободить ресурсы

|  |  |
| --- | --- |
| 4. Создать проект, демонстрирующий работу с ***очередью***. В соответствии со своим вариантом выполнить задание из таблицы, представленной ниже. Разработать меню и реализовать все операции с очередью через функции. Максимальный размер очереди ввести с клавиатуры.  **Вар.1:** Создать очередь для символов и функции для ввода, вывода и удаления элементов очереди. Ввести эталонный символ. Вводить символы с экрана в очередь до встречи эталонного. После встречи с эталонным, вывести всю очередь, удалить два элемента и посчитать оставшееся количество элементов очереди. |  |

#include <iostream>

using namespace std;

struct queue

{

char symbol;

queue\* next;

};

void printLine()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

void showMenu()

{

printLine();

cout << "Меню:\n" << "1 - Ввод нового символа в очередь\n" << "2 - Удаление символа из очереди\n";

cout << "3 - Вывести очередь\n" << "4 - Ввести эталонный символ\n" << "5 - Вывод меню\n" << "0 - Выход из программы\n";

printLine();

//после эталонного: вывод, удалить 2 эл, вывод, посчитать оставшиеся элементы

}

void inputNewEl(queue\*& end, queue\*& begin)

{

queue\* newEl = new queue;

cout << "Введите новый символ очереди: ";

cin >> newEl->symbol;

newEl->next = nullptr;

if (end == nullptr)

{

end = newEl;

begin = newEl;

}

else

{

end->next = newEl;

end = newEl;

}

}

int countQuantity(queue\* begin)

{

int size = 0;

if (begin != nullptr)

{

while (begin != nullptr)

{

size++;

begin = begin->next;

}

}

else

{

cout << "Очередь пустая\n";

}

return size;

}

void deleteEl(queue\*& begin, int quantityEl)

{

if (quantityEl <= countQuantity(begin))

{

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

{

queue\* temp = begin;

begin = begin->next;

delete temp;

}

}

else

{

cout << "Ошибка, очередь короче 2 символов, условие не может быть выполненно\n";

}

}

void showQueue(queue\* begin)

{

if (countQuantity(begin) != 0)

{

while (begin != nullptr)

{

cout << begin->symbol << " ";

begin = begin->next;

}

cout << "\n";

}

}

void inputReferenceSymbol(queue\*& begin, queue\*& end)

{

char refSymbol;

queue\* newEl;

cout << "Введите эталонный символ: ";

cin >> refSymbol;

cout << "Эталонный символ успешно записан, для того чтобы окончить ввод введите эталонный символ повторно\n";

do

{

newEl = new queue;

cout << "Введите новый символ очереди: ";

cin >> newEl->symbol;

newEl->next = nullptr;

if (end == nullptr)

{

end = newEl;

begin = newEl;

}

else

{

end->next = newEl;

end = newEl;

}

} while (newEl->symbol != refSymbol);

cout << "Вся очередь:\n";

showQueue(begin);

cout << "Очередь после удаления 2 символов:\n";

deleteEl(begin, 2);

showQueue(begin);

cout << "Количество оставшихся элементов после удаления: " << countQuantity(begin) << endl;

}

int main()

{

queue\* begin, \* end;

end = begin = nullptr;

int userInput, quantityEl;

setlocale(LC\_CTYPE, "Russian");

showMenu();

cin >> userInput;

while (userInput != 0)

{

switch (userInput)

{

case 1:

inputNewEl(end, begin);

break;

case 2:

cout << "Введите сколько элементов хотите удалить: ";

cin >> quantityEl;

deleteEl(begin, quantityEl);

break;

case 3:

showQueue(begin);

break;

case 4:

inputReferenceSymbol(begin, end);

break;

case 5:

showMenu();

break;

case 0:

break;

default:

cout << "Ошибка ввода\n";

break;

}

cout << "Что дальше?\n";

cin >> userInput;

}

}

|  |  |
| --- | --- |
| 5.1.В [приложении 1](#prilogenie1) приведен проект, в котором реализована очередь на основе односвязного *списка* с *приоритетным* включением. На основе данного проекта разработать функции, которые предлагается создать в данном приложении.  1. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие − подряд из младших адресов (начало очереди). Приоритет: минимальное значение числового параметра, при совпадении параметров − **LIFO**. |  |

#include <iostream>

using namespace std;

struct Item

{

int value;

Item\* next;

};

Item\* head, \* tail;

void addNewEl()

{

Item\* newValue = new Item;

Item\* before, \* after;

cout << "Введите новое значение: ";

cin >> newValue->value;

if (head == nullptr)

{

head = tail = newValue;

head->next = nullptr;

}

else if (newValue->value < head->value)

{

newValue->next = head;

head = newValue;

}

else

{

before = head;

after = before->next;

while (after != nullptr)

{

if (newValue->value > after->value)

{

before = after;

}

after = after->next;

}

if (before->next == nullptr)

{

newValue->next = nullptr;

tail->next = newValue;

tail = tail->next;

}

else

{

after = before->next;

newValue->next = after;

before->next = newValue;

}

}

}

void showQueue()

{

Item\* ptr = head;

if (ptr != nullptr)

{

while (ptr != nullptr)

{

cout << ptr->value << "-->";

ptr = ptr->next;

}

}

else

{

cout << "Очередь пуста\n";

}

cout << "NULL\n";

}

void getEl()

{

cout << "1-ый элемент в очереди: " << head->value << endl;

}

void cutEl()

{

if (head != nullptr)

{

Item\* temp = head;

cout << "Из очереди был извлечен элемент " << temp->value;

head = head->next;

delete(temp);

}

else

{

cout << "Ошибка: очередь пуста\n";

}

}

void clearQueue()

{

while (head != nullptr)

{

Item\* temp = head;

head = head->next;

delete (temp);

}

delete(head);

}

int main()

{

setlocale(LC\_CTYPE, "Russian");

int choice = 1;

head = nullptr;

tail = nullptr;

while (choice != 0)

{

cout << "1 - добавить элемент по приоритету" << endl;

cout << "2 - получить элемент с начала" << endl;

cout << "3 - извлечь элемент с начала" << endl;

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

cout << "5 - очистить очередь" << endl;

cout << "0 - выход" << endl;

cout << "Выберите действие ";

cin >> choice;

switch (choice)

{

case 1:

addNewEl();

cout << "Очередь:\n";

showQueue();

break;

case 2:

getEl();

break;

case 3:

cutEl();

cout << "Новая очередь:\n";

showQueue();

break;

case 4:

showQueue();

break;

case 5:

clearQueue();

cout << "Очередь успешно очищена\n";

break;

}

}

return 0;

}

|  |  |
| --- | --- |
| 5.2. Создать очередь с вещественными числами, и заполнить ее с клавиатуры. Выполнить циклический сдвиг элементов в очереди так, чтобы в ее начале был расположен наибольший элемент. |  |

#include <iostream>

using namespace std;

struct Item

{

double value;

Item\* next;

};

Item\* head, \* tail;

void printLine()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

void showMenu()

{

printLine();

cout << "Меню:\n" << "1 - Добавить новое значение в очередь\n";

cout << "2 - Вывести очередь\n" << "3 - Поместить наибольший элемент на 1 позицию\n";

cout << "4 - Показать меню\n" << "0 - Выход из программы\n";

printLine();

}

void addNewEl()

{

Item\* newEl = new Item;

cout << "Введите новый элемент очереди: ";

cin >> newEl->value;

newEl->next = nullptr;

if (head == nullptr)

{

head = tail = newEl;

}

else

{

tail->next = newEl;

tail = newEl;

}

}

void showQueue()

{

Item\* ptr = head;

while (ptr != nullptr)

{

cout << ptr->value << "-->";

ptr = ptr->next;

}

cout << "NULL\n";

}

int searchMaxEl()

{

int max = head->value;

Item\* ptr = head;

while (ptr != nullptr)

{

if (ptr->value > max)

{

max = ptr->value;

}

ptr = ptr->next;

}

return max;

}

void deleteElFromQueue(int max)

{

Item\* before, \* after;

before = after = head;

if (before->value == max)

{

head = head->next;

delete(before);

}

else if (after->next != nullptr)

{

after = after->next;

while (after != nullptr)

{

if (after->value == max)

{

after = after->next;

break;

}

before = after;

after = after->next;

}

before->next = after;

}

}

void moveQueue()

{

int max = searchMaxEl();

deleteElFromQueue(max);

Item\* newValue = new Item;

newValue->value = max;

newValue->next = head;

head = newValue;

}

int main()

{

setlocale(LC\_CTYPE, "Russian");

int userInput = 1;

head = nullptr;

tail = nullptr;

showMenu();

while (userInput != 0)

{

cin >> userInput;

switch (userInput)

{

case 1:

addNewEl();

break;

case 2:

showQueue();

break;

case 3:

moveQueue();

break;

case 4:

showMenu();

break;

case 0:

break;

}

cout << "Что дальше ?\n";

}

return 0;

}

|  |  |
| --- | --- |
| 5.3. Содержимое текстового файла **f**, разделенное на строки, переписать в текстовый файл **g**, перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением исходного взаимного порядка, как среди цифр, так и среди остальных символов строки). Использовать очереди. |  |

#include <iostream>

using namespace std;

void printLine() \

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

FILE\* fdata, \* gdata;

struct Num

{

int value;

Num\* next;

};

struct Symbol

{

char value;

Symbol\* next;

};

Num\* firstnum = nullptr;

Symbol\* firstsymb = nullptr;

void showMenu()

{

printLine();

cout << "Меню:\n" << "1 - Заполнить случайными символами файл f\n";

cout << "2 - Заполнить файл g\n" << "0 - Выход из программы\n";

printLine();

}

void fillFFile()

{

srand(time(0));

int colum, row, symbolOrNumber;

char symbol;

cout << "Укажите количество строк: ";

cin >> colum;

cout << "Укажите количество символов в каждой строке: ";

cin >> row;

if (!fopen\_s(&fdata, "filef.txt", "w"))

{

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

{

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

{

symbolOrNumber = rand() % 3;

if (symbolOrNumber == 0)

{

symbol = 65 + rand() % 26;

}

else if (symbolOrNumber == 1)

{

symbol = 97 + rand() % 26;

}

else

{

symbol = 48 + rand() % 10;

}

fwrite(&symbol, sizeof(char), 1, fdata);

}

fwrite("\n", sizeof(char), 1, fdata);

}

fclose(fdata);

}

else

{

cout << "Ошибка открытия файла f\n";

}

}

void readLine(char symbol)

{

Num\* numbers;

Symbol\* symbols;

Num\* ptrForNum = firstnum;

Symbol\* ptrForSymbol = firstsymb;

while (symbol != '\n')

{

if (symbol >= '0' && symbol <= '9')

{

if (ptrForNum == nullptr)

{

numbers = new Num;

numbers->value = symbol;

numbers->next = nullptr;

firstnum = numbers;

ptrForNum = firstnum;

}

else

{

numbers = new Num;

numbers->value = symbol;

numbers->next = nullptr;

ptrForNum->next = numbers;

ptrForNum = ptrForNum->next;

}

}

else

{

if (ptrForSymbol == nullptr)

{

symbols = new Symbol;

symbols->value = symbol;

symbols->next = nullptr;

firstsymb = symbols;

ptrForSymbol = firstsymb;

}

else

{

symbols = new Symbol;

symbols->value = symbol;

symbols->next = nullptr;

ptrForSymbol->next = symbols;

ptrForSymbol = ptrForSymbol->next;

}

}

fread(&symbol, sizeof(char), 1, fdata);

}

}

void fillGFile()

{

char symbol;

if (!fopen\_s(&fdata, "filef.txt", "r") && !fopen\_s(&gdata, "fileg.txt", "w"))

{

while (fread(&symbol, sizeof(char), 1, fdata))

{

readLine(symbol);

while (firstsymb != nullptr)

{

fwrite(&(firstsymb->value), sizeof(char), 1, gdata);

Symbol\* temp;

temp = firstsymb;

firstsymb = firstsymb->next;

delete(temp);

}

while (firstnum != nullptr)

{

fwrite(&(firstnum->value), sizeof(char), 1, gdata);

Num\* temp;

temp = firstnum;

firstnum = firstnum->next;

delete(temp);

}

fwrite("\n", sizeof(char), 1, gdata);

}

fclose(fdata);

fclose(gdata);

}

else

{

cout << "Ошибка, файл f не заполенен";

}

cout << "Файл успешно заполен\n";

}

int main()

{

int userInput = 1;

firstnum = nullptr;

firstsymb = nullptr;

setlocale(LC\_CTYPE, "Russian");

showMenu();

while (userInput != 0)

{

cin >> userInput;

switch (userInput)

{

case 1:

fillFFile();

break;

case 2:

fillGFile();

break;

case 0:

break;

default:

cout << "Ошибка ввода";

break;

}

cout << "Что дальше ?\n";

}

}