Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе № 11**

ОЧЕРЕДИ

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь 2023**

**Цель работы**

Получить практические навыки работы с очередями.

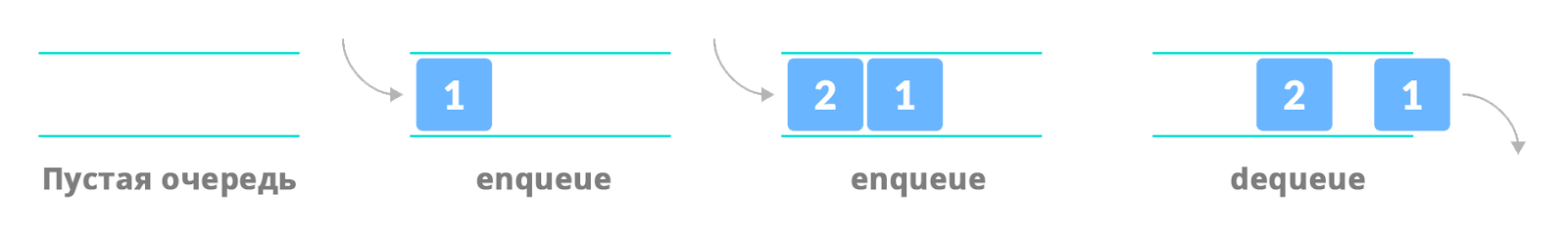
**Постановка задачи**

Записи в линейном списке содержат ключевое поле типа \*char (строка символов). Реализовать все методы очереди.

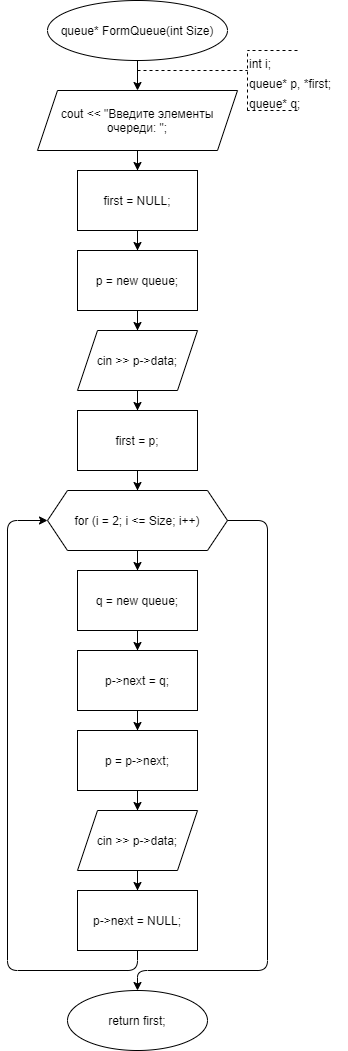
**Анализ задачи**

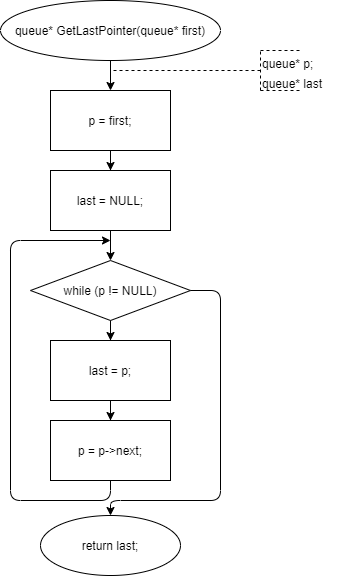
Реализовано с использованием односвязных списков

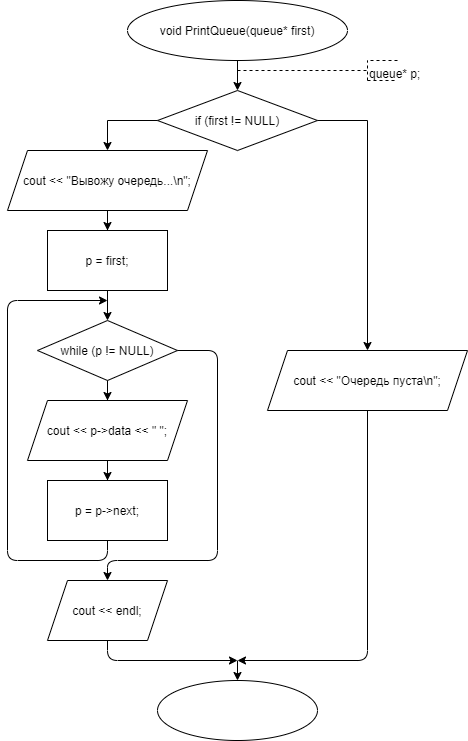
Очередь подчиняется принципу FIFO — Firts In First Out («первый пришел — первый вышел»). Первый элемент в очереди выходит первым.

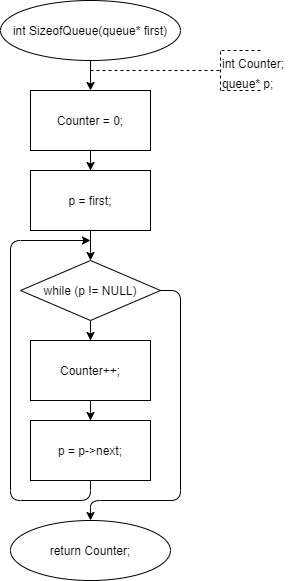


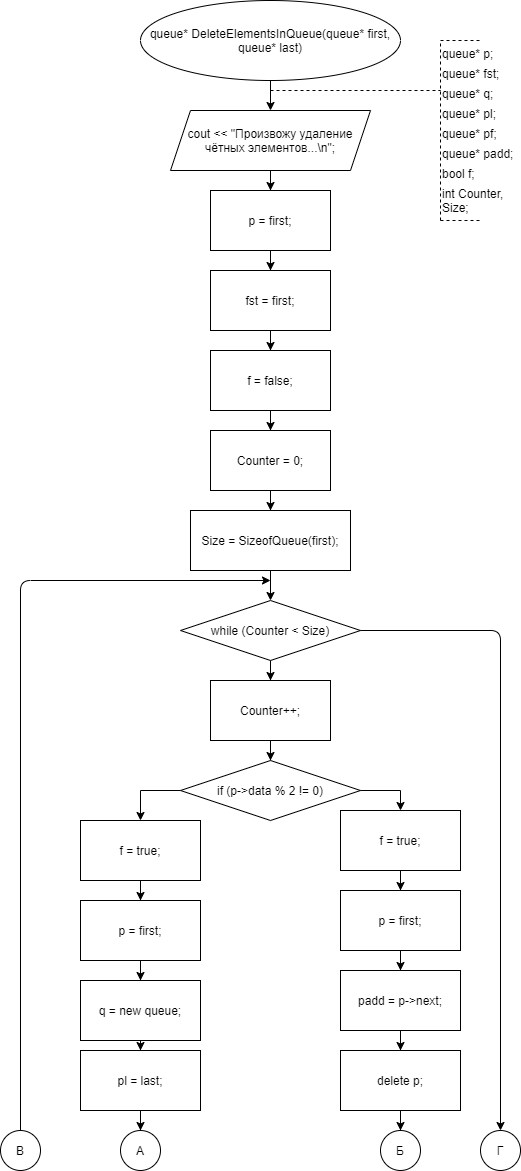
**Блок-схема программы**

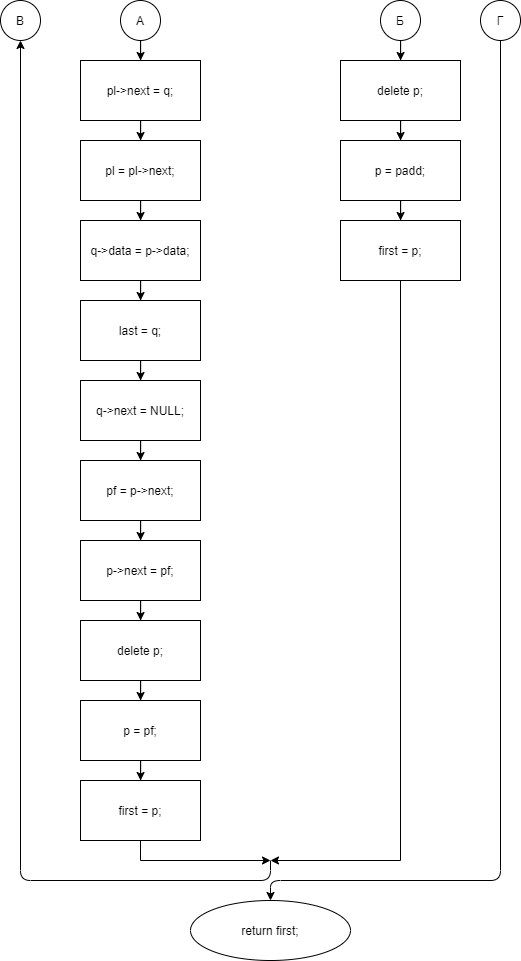


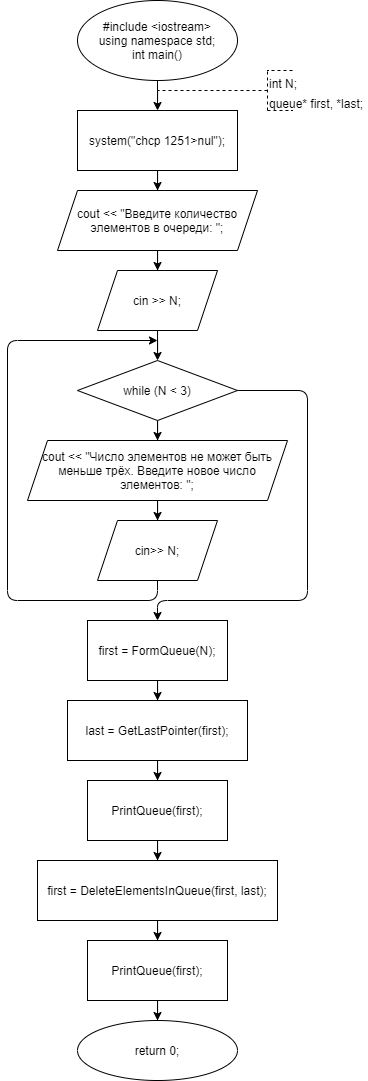












**Реализация задачи на языке С++**

/\*18. Записи в линейном списке содержат ключевое поле типа \*char (строка символов)\*/

#include <iostream>

#include <string>

#include <fstream>

#include <math.h>

using namespace std;

struct Queue // структура односвязного очереди

{

char data[100]; //поле, содержащее информацию, хранящуюся в одном узле

Queue\* next; //поле, хранящее указатель на следующий узел

};

int N;

Queue\* head;

Queue\* tail;

Queue\* Pop() // Удаление из начала очереди

{

Queue\* delElem = head;

head = head->next;

delElem->next = nullptr;

N--;

return delElem;

}

void Push(Queue\* elem) // Добавление в конец очереди

{

tail->next = elem;

tail = tail->next;

N++;

}

void create\_queue()

{

cout << "Метод создания очереди вызван" << endl;

DeleteQueue();

head = nullptr; // пока нет элементов, помимо первого, нет и ссылки на следующий элемент

Queue\* node = new Queue; // выделение памяти под узел

cout << "Введите элементы очереди: ";

cin.getline(node->data, 100);

cin.getline(node->data, 100); // заполнение первого узла данными

head = node; //присваиваем данные первому узлу

tail = head;

int n = N;

N = 1;

for (int i = 1; i < n; i++)

{

node = new Queue; // Переход указателем на следующий элемент

cin.getline(node->data, 100); // Запись нового элемента

node->next = nullptr; // Указатель на следующий элемент равен нулю, т. к. был записан последний элемент

Push(node);

}

cout << "Очередь заполнена" << endl;

}

void PrintQueueConsole(Queue\* head)

{

while (head != NULL)

{

cout << head->data << " ";

head = head->next;

}

cout << endl;

}

fstream file;

string path = "Queue.txt";

void PrintQueueFile(Queue\* head)

{

file.open(path, fstream::out); //открытие файла для записи

if (!file.is\_open())

cout << "Ошибка открытия файла " << path << " для записи" << endl;

else

{

while (head != NULL)

{

file << head->data << endl;

head = head->next;

}

cout << "Данные записаны в файл" << endl;

}

file.close();

}

void AddOneElemEnd()

{

Queue\* elem = new Queue;

cin.getline(elem->data, 100); // Запись нового элемента

elem->next = nullptr;

Push(elem);

}

void AddKElemEnd(int k)

{

int i = 0;

char c[100];

cin.getline(c, 100);

while (i < k)

{

AddOneElemEnd();

i++;

}

}

void AddKElemBeg(int k)

{

Queue\* node = new Queue; // выделение памяти под узел

cin.getline(node->data, 100);

for (int i = 0; i < k; i++) //создание очереди из к элементов, который будет присоединяться к концу существующей очереди

{

node = new Queue;

cin.getline(node->data, 100); // Запись нового элемента

node->next = nullptr; // Указатель на следующий элемент равен нулю, т. к. был записан последний элемент

Push(node);

}

for (int i = 0; i < N - k; i++)

{

Queue\* tailElem = Pop();

Push(tailElem);

}

}

void AddKElemIndex(int pos, int k)

{

if (pos > N)

{

AddKElemEnd(k); // Вызов добавления k элементов в конец очереди

}

else if (pos == 0)

{

AddKElemBeg(k); // Вызов добавления k элементов в начало очереди

}

else

{

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

{

Queue\* tailElem = Pop();

Push(tailElem);

}

AddKElemBeg(k);

for (int i = 0; i < N - pos; i++)

{

Queue\* tailElem = Pop();

Push(tailElem);

}

}

}

void DeleteQueue()

{

while (N != 0)

{

Queue\* tmp = Pop();

delete tmp;

}

}

void DelOneElemBeg()

{

Queue\* elem = Pop();

delete elem;

}

void DelKElemBeg(int k)

{

int i = 0;

while (i < k)

{

DelOneElemBeg();

i++;

}

}

void DelKElemEnd(int k)

{

int i = 0;

while (i < N - k)

{

Queue\* tailElem = Pop();

Push(tailElem);

i++;

}

DelKElemBeg(k);

}

void DelKElemIndex(int pos, int k) // i > k

{

if (pos > N)

{

DelKElemEnd(k); // Вызов удаления k элементов из конца очереди

}

else if (pos < 1)

{

DelKElemBeg(k); // Вызов удаления k элементов из начала очереди

}

else

{

int index = abs(pos - k);

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

{

Queue\* tailElem = Pop(); // перемещение по списку до нужного индекса

Push(tailElem);

}

DelKElemBeg(k);

for (int i = 0; i < N - index; i++)

{

Queue\* tailElem = Pop(); // перемещение по списку до нужного индекса

Push(tailElem);

}

}

}

void print\_menu() {

system("cls"); // очищаем экран

cout << "ЧТО ВЫ ХОТИТЕ СДЕЛАТЬ?" << endl << endl;

cout << "\t\t\t\tOЧЕРЕДЬ" << endl << endl;

cout << "1. Создать очередь." << endl << endl;

cout << "+ Добавить k элементов в: " << endl << endl;

cout << "\t\t2. конец очереди." << endl;

cout << "\t\t3. начало очереди." << endl;

cout << "\t\t4. очередь по индексу." << endl << endl;

cout << "- Удалить k элементов из: " << endl << endl;

cout << "\t\t5. конца очереди." << endl;

cout << "\t\t6. начала очереди." << endl;

cout << "\t\t7. очереди по индексу." << endl << endl;

cout << "8. вывод очереди в консоль." << endl;

cout << "9. запись очереди в файл." << endl;

cout << "10. уничтожение очереди." << endl;

cout << "11. восстановление очереди из файла." << endl;

cout << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << endl;

cout << "12. Выход" << endl;

cout << ">> ";

}

int get\_var(int count)

{

int var;

cin >> var;

// пока ввод некорректен, сообщаем об этом и просим повторить его

while (var < 1 or var > count) {

cout << "НЕКОРРЕКТНЫЙ ВВОД! ПОПРОБУЙТЕ СНОВА " << endl; // выводим сообщение об ошибке

system("pause");

cout << ">> ";

cin >> var; // считываем строку повторно

}

return var;

}

Queue\* recovery\_Queue()

{

file.open(path, fstream::in); //открытие файла для записи

if (!file.is\_open())

cout << "Ошибка открытия файла " << path << " для записи" << endl;

else

{

Queue\* node = new Queue; // выделение памяти под узел

file.getline(node->data, 100);

head = node; //присваиваем данные первому узлу

tail = head;

int i = 1; // позиция курсора

N = 1;

while (!file.eof())

{

if (i != 1)

{

node = new Queue; // Выделение димнамической памяти под новый элемент

file.getline(node->data, 100); // Запись нового элемента

node->next = nullptr;

Push(node);

}

i++;

}

cout << "Очередь восстановлена!" << endl;

}

file.close();

return head;

}

int main()

{

system("chcp 1251>NULL");

print\_menu();

int k, pos;

int var;

head = nullptr;

tail = nullptr;

do {

print\_menu(); // выводим меню на экран

var = get\_var(12); // получаем номер выбранного пункта меню

switch (var)

{

case 1:

cout << "Введите количество элементов очереди: ";

cin >> N;

create\_queue();

break;

case 2:

cout << "Введите количество элементов, которое вы хотите добавить в конец очереди: ";

cin >> k;

cout << "Введите элементs: ";

AddKElemEnd(k);

break;

case 3:

cout << "Введите количество элементов, которое вы хотите добавить в начало очереди: ";

cin >> k;

cout << "Введите элементs: ";

AddKElemBeg(k);

break;

case 4:

cout << "Введите индекс добавления элементов в очередь: ";

cin >> pos;

cout << "Введите количество элементов, которое вы хотите добавить в очередь: ";

cin >> k;

cout << "Введите элементs: ";

AddKElemIndex(pos, k);

break;

case 5:

cout << "Введите количество элементов, которое вы хотите удалить из конца очереди: ";

cin >> k;

DelKElemEnd(k);

break;

case 6:

cout << "Введите количество элементов, которое вы хотите удалить из начала очереди: ";

cin >> k;

if (k == 1) DelOneElemBeg();

else DelKElemBeg(k);

break;

case 7:

cout << "Введите индекс удаления элементов из очереди: ";

cin >> pos;

cout << "Введите количество элементов, которое вы хотите удалить из очереди: ";

cin >> k;

DelKElemIndex(pos, k);

break;

case 8:

if (N == 0)

{

head = nullptr;

cout << "Очередь удалена. Элементы в очереди осутствуют!" << endl;

}

cout << "очередь: ";

PrintQueueConsole(head);

break;

case 9:

PrintQueueFile(head);

break;

case 10:

DeleteQueue();

if (N == 0) {

cout << "Очередь удалена!" << endl;

head = nullptr;

}

break;

case 11:

head = recovery\_Queue();

break;

}

if (var != 12)

system("pause"); // задерживаем выполнение, чтобы пользователь мог увидеть результат выполнения выбранного пункта

} while (var != 12);

return 0;

}