Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №11.4**

Дисциплина: основы алгоритмизации и программирования

Тема: “Очередь"

**Вар.20**

Выполнил работу

студент группы ИВТ-20-2Б

Кузнецов Н.Д

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2020

**Цель задачи**

1) Получить практические навыки работы с однонаправленными списками;

2) Получить практические навыки работы с двунаправленными списками;

3) Получить практические навыки работы со стеками;

4) Получить практические навыки работы с очередями.  
Для достижения поставленной цели необходимо решить следующие задачи:

* Разработать структуру однонаправленного списка.
* Организовать ввод-вывод данных.
* Реализовать функцию удаления элементов из списка.
* Разработать программу.

**Постановка задачи**

Реализовать очередь. Тип информационного поля char\*. Добавить в список элемент после элемента с заданным информационным полем.

**Анализ задачи**

1. Для решения задачи используются типы данных **bool**, **int**, класс **string** и класс **vector**.
   1. Для считывания и запоминания элементов очереди используется **vector** типа **string**.
   2. Поле data, списка так же типа string.  
      struct List

{

string data;

List\* next\_block;

List\* prev\_block;

};

* 1. Для корректного ввода длины очереди используется функция GetSize, которая не принимает значений и возвращает тип данных **int**.

1. Для хранения данных, вводимых пользователем, реализована очередь через двунаправленный список (см.п.1.2).
   1. Для заполнения очереди реализована функция Create\_list, которая в качестве параметра принимает длину очереди, возвращает указатель на первый элемент очереди.  
      List\* Create\_list(int size)

{

List\* first\_element = nullptr;

List\* current\_element;

if (size == 1)

{

cout << "Введите 0-й элемент очереди:\t";

current\_element = new List;

cin >> current\_element->data;

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

}

else

{

cout << "Введите 0-й элемент очереди:\t";

current\_element = new List;

cin >> current\_element->data;

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

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

{

cout << "Введите " << i << "-й элемент очереди:\t";

List\* new\_element = new List;

current\_element->next\_block = new\_element;

new\_element->prev\_block = current\_element;

current\_element = new\_element;

cin >> current\_element->data;

current\_element->next\_block = nullptr;

}

}

return first\_element;

}

* 1. Для удаления элемента из очереди описана функция Pop, которая принимает указатель на первый элемент и возвращает указатель на 1 элемент в очереди.

List\* Pop(List\* first\_element)

{

List\* tmp = first\_element->next\_block;

delete first\_element;

first\_element = tmp;

return first\_element;

}

* 1. Для добавления элемента в очередь реализована функция Push\_list, которая принимает указатель на первый элемент и значение, которое нужно добавить (value). Функция возвращает указатель на первый элемент.

List\* Push\_list(List\* first\_element, string value)

{

List\* current\_element = first\_element;

if (first\_element == nullptr)

{

first\_element = new List;

first\_element->data = value;

first\_element->next\_block = nullptr;

first\_element->prev\_block = nullptr;

}

else

{

{

while (current\_element->next\_block != nullptr)

current\_element = current\_element->next\_block;

}

List\* new\_element = new List;

new\_element->data = value;

new\_element->next\_block = nullptr;

new\_element->prev\_block = current\_element;

current\_element->next\_block = new\_element;

}

return first\_element;

}

* 1. Для вывода элементов очереди используется **vector**, как временное хранилище данных. Циклом while элементы поочередно выводятся на консоль, удаляются из очереди и добавляются в vector, аналогично происходит заполнение очереди обратно.

List\* Print\_list(List\*& first\_element)

{

vector <string> tmp;

int i = 0;

while (first\_element != nullptr)

{

cout << i << "-й элемент очереди:\t" << first\_element->data << endl;

i++;

tmp.push\_back(first\_element->data);

first\_element = Pop(first\_element);

}

int j = 0;

while (j < tmp.size())

{

first\_element = Push\_list(first\_element, tmp[j]);

j++;

}

return first\_element;

}

* 1. Для добавления нового элемента с указанным полем разработана функция Add\_element, которая принимает ссылку на указатель первого элемента очереди. Функция возвращает false в случае, если элемент в очереди с таким полем не найден, иначе функция возвращает true. Аналогично п.2.4 все данные из очереди переносятся в vector (**tmp**). C помощью цикла for в tmp после элемента со значение **value** добавляется новый элемент, значение которого вводит пользователь. Аналогично п.2.4 элементы из vector’a переносятся обратно в очередь.

bool Add\_element(List\*& first\_element)

{

string value;

vector <string> tmp;

bool check = false;

cout << "Введите значение, после которого нужно добавить элемент:\t";

cin >> value;

while (first\_element != nullptr)

{

tmp.push\_back(first\_element->data);

first\_element = Pop(first\_element);

}

for (int i = 0; i < tmp.size(); i++)

{

if (tmp[i] == value)

{

check = true;

string temp;

cout << "Введите значение нового элемента в очереди:\t";

cin >> temp;

tmp.insert(tmp.begin() + i + 1, temp);

}

}

for (int j = 0; j < tmp.size(); j++)

{

first\_element = Push\_list(first\_element, tmp[j]);

}

return check;

}

1. Для хранения данных реализована очередь по принципу работы двунаправленного списка.
2. Структура очереди имеет 3 поля, поле информации (**string**), 2 указателя: на следующий и на прошлый элемент очереди.

struct List

{

string data;

List\* next\_block;

List\* prev\_block;

};

1. Для ввода-вывода используются операторы cin, cout, для этого подключена соответствующая библиотека (см.п.2.4)
2. В функции **main** переменной **size** присваивается результат выполнения функции **GetSize**, далее создается указатель на первый элемент очереди и ему присваивается результат выполнения функции **Create\_list**, в которую передается переменная **size**. Далее очередь выводится пользователю на консоль. Оператором if проверяется результат выполнения функции **Add\_element**. В случае истины пользователю выводится в консоль измененная очередь, в противном же случае, пользователю выводится сообщение о том, что элемента с введенным полем нет.  
   …   
   setlocale(LC\_ALL, "ru");

cout << "Введите размер очереди:\t";

int size = GetSize();

List\* que = Create\_list(size);

cout << "============================\n";

que = Print\_list(que);

cout << "============================\n";

if (Add\_element(que))

{

cout << "============================\n";

Print\_list(que);

cout << "============================\n";

}

else

{

cout << "============================\n";

cout << "Такого элемента в очереди нет!\n";

cout << "============================\n";

}  
 …

**Код через struct**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

struct List

{

string data;

List\* next\_block;

List\* prev\_block;

};

int GetSize()

{

int size;

while (!(cin >> size) || (cin.peek() != '\n') || (size < 1))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число!\t";

}

}

return size;

}

List\* Create\_list(int size)

{

List\* first\_element = nullptr;

List\* current\_element;

if (size == 1)

{

cout << "Введите 0-й элемент очереди:\t";

current\_element = new List;

cin >> current\_element->data;

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

}

else

{

cout << "Введите 0-й элемент очереди:\t";

current\_element = new List;

cin >> current\_element->data;

first\_element = current\_element;

current\_element->next\_block = nullptr;

current\_element->prev\_block = nullptr;

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

{

cout << "Введите " << i << "-й элемент очереди:\t";

List\* new\_element = new List;

current\_element->next\_block = new\_element;

new\_element->prev\_block = current\_element;

current\_element = new\_element;

cin >> current\_element->data;

current\_element->next\_block = nullptr;

}

}

return first\_element;

}

List\* Pop(List\* first\_element)

{

List\* tmp = first\_element->next\_block;

delete first\_element;

first\_element = tmp;

return first\_element;

}

List\* Push\_list(List\* first\_element, string value)

{

List\* current\_element = first\_element;

if (first\_element == nullptr)

{

first\_element = new List;

first\_element->data = value;

first\_element->next\_block = nullptr;

first\_element->prev\_block = nullptr;

}

else

{

{

while (current\_element->next\_block != nullptr)

current\_element = current\_element->next\_block;

}

List\* new\_element = new List;

new\_element->data = value;

new\_element->next\_block = nullptr;

new\_element->prev\_block = current\_element;

current\_element->next\_block = new\_element;

}

return first\_element;

}

List\* Print\_list(List\*& first\_element)

{

vector <string> tmp;

int i = 0;

while (first\_element != nullptr)

{

cout << i << "-й элемент очереди:\t" << first\_element->data << endl;

i++;

tmp.push\_back(first\_element->data);

first\_element = Pop(first\_element);

}

int j = 0;

while (j < tmp.size())

{

first\_element = Push\_list(first\_element, tmp[j]);

j++;

}

return first\_element;

}

bool Add\_element(List\*& first\_element)

{

string value;

vector <string> tmp;

bool check = false;

cout << "Введите значение, после которого нужно добавить элемент:\t";

cin >> value;

while (first\_element != nullptr)

{

tmp.push\_back(first\_element->data);

first\_element = Pop(first\_element);

}

for (int i = 0; i < tmp.size(); i++)

{

if (tmp[i] == value)

{

check = true;

string temp;

cout << "Введите значение нового элемента в очереди:\t";

cin >> temp;

tmp.insert(tmp.begin() + i + 1, temp);

}

}

for (int j = 0; j < tmp.size(); j++)

{

first\_element = Push\_list(first\_element, tmp[j]);

}

return check;

}

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Введите размер очереди:\t";

int size = GetSize();

List\* que = Create\_list(size);

cout << "============================\n";

que = Print\_list(que);

cout << "============================\n";

if (Add\_element(que))

{

cout << "============================\n";

Print\_list(que);

cout << "============================\n";

}

else

{

cout << "============================\n";

cout << "Такого элемента в очереди нет!\n";

cout << "============================\n";

}

}

**Код через STL**

#include <iostream>

#include <queue>

#include <iterator>

#include <vector>

#include <string>

using namespace std;

queue <string> que\_list;

int GetSize()

{

int size;

cout << "Введите длину очереди:\t";

//cin.ignore();

while (!(cin >> size) || (cin.peek() != '\n') || size < 1)

{

cin.clear();

while (cin.get() != '\n');

cout << "Введите корректное число:\t";

}

return size;

}

void Create\_que(int size)

{

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

{

string tmp;

cout << "Введите " << i << "-й элемент очереди:\t";

cin >> tmp;

que\_list.push(tmp);

}

}

void Print\_que()

{

vector <string> tmp;

int i = 0;

while (!que\_list.empty())

{

cout << i << "-й элемент очереди:\t" << que\_list.front() << endl;

tmp.push\_back(que\_list.front());

que\_list.pop();

i++;

}

for (int j = 0; j < tmp.size(); j++)

{

que\_list.push(tmp[j]);

}

}

bool Add\_element()

{

string value;

vector <string> tmp;

bool check = false;

cout << "Введите значение, после которого нужно добавить элемент:\t";

cin >> value;

while (!que\_list.empty())

{

tmp.push\_back(que\_list.front());

que\_list.pop();

}

for (int i = 0; i < tmp.size(); i++)

{

if (tmp[i] == value)

{

check = true;

que\_list.push(tmp[i]);

string temp;

cout << "Введите значение нового элемента в очереди:\t";

cin >> temp;

que\_list.push(temp);

}

else

{

que\_list.push(tmp[i]);

}

}

return check;

}

int main()

{

setlocale(LC\_ALL, "ru");

int size = GetSize();

Create\_que(size);

cout << "================================================" << endl;

cout << "Изначальная очередь:" << endl;

Print\_que();

cout << "================================================" << endl;

if (Add\_element())

{

cout << "================================================" << endl;

cout << "Очередь после добавления:" << endl;

Print\_que();

cout << "================================================" << endl;

}

else

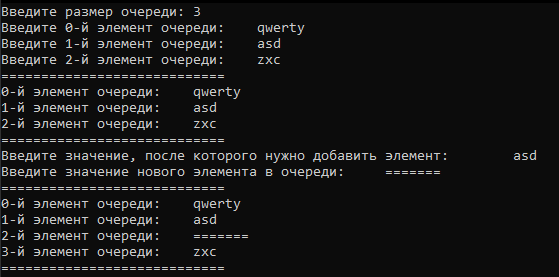
{

cout << "Такого элемента в очереди нет" << endl;

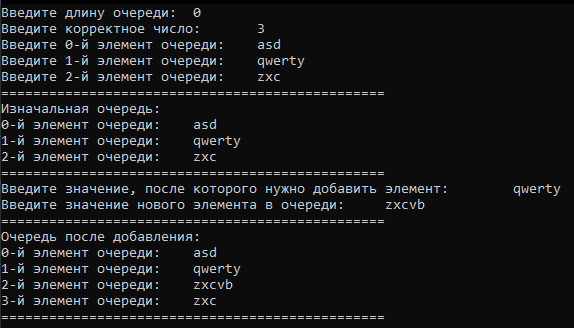
}

}

**Работа кода через struct**



**Работа код через STL**



**Блок-схема**

