**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе № 3**

**по дисциплине «Программирование»**

**Тема: Линейные структуры данных: стек, очередь, дек**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Быков И. В. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

1. **Цель работы.**

Познакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование стека, очереди и дека для решения задач.

1. **Постановка задачи.**

Вариант 3. Рассматриваются следующие типы данных:

**type** *имя* = (*Анна,* *...,* *Яков*);

*дети* = **array** [*имя,* *имя*] **of** *Boolean*;

*потомки* = **file of** *имя*.

Задан массив *Д* типа *дети* (*Д* [*x*, *y*] = *true*, если человек по имени *y* является ребенком человека по имени *x*). Для введенного пользователем имени *И* записать в файл *П* типа *потомки* имена всех потомков человека с именем *И* в следующем порядке: сначала - имена всех его детей, затем - всех его внуков, затем - всех правнуков и т. д.

В задании требуется использовать очередь.

1. **Основные теоретические положения.**

Стек, очередь и дек представляют собой разновидности линейного списка.

Ссылочная реализация стека и очереди в динамической памяти в основном аналогична ссылочной реализации линейных списков. Упрощение связано с отсутствием необходимости работать с текущим элементом списка. Идеи такой реализации ясны из рисунка 3.3.

Для ссылочной реализации дека естественно использовать двунаправленный список.

*Стек*:

*Верхушка стека*

*Начало*:

*Конец*:

*Очередь*:

*Рис. 3.3. Ссылочное представление стека и очереди*

Поскольку для стека, очереди и дека доступ к элементам осуществляется только через начало и конец последовательности, то эти структуры данных допускают и эффективную непрерывную реализацию на базе вектора. При этом используется одномерный массив Mem: array [0..n] of α и переменная Верх :  -1..n.

***Mem:***

***n***

***2***

***1***

***0***

***Верх***

**X X X X X X . . .**

*Рис. 3.4. Непрерывная реализация ограниченного стека на базе вектора*

Для пустого стека Верх = = -1, для целиком заполненного стека Верх = = n. Вершина стека доступна как Mem [Верх], операция Pop реализуется как Верх:= Верх −1, а операция Push (p, s) как {Верх = Верх + 1; Mem [Верх] = p } при -1 ≤ Верх < n.

Непрерывная реализация ограниченной очереди на базе вектора требует, в отличие от стека, двух переменных Начало и Конец.

***1***

***0***

***2***

***n***

**X X X X X X X**

**X X X X X X**

***Mem:***

***Конец***

***Начало***

*Рис. 3.5. Непрерывная реализация ограниченной очереди на базе вектора*

Особенностью такого представления является наличие ситуации, когда последовательность элементов очереди по мере их добавления может выходить за границу вектора, продолжаясь с его начала (вектор имитирует здесь так называемый кольцевой буфер). Эта ситуация изображена на рис. 3.6.

***0***

***1***

***2***

***n***

***Mem:***

**X X X X X X X X X X X**

***Начало***

***Конец***

*Рис. 3.6. Непрерывное представление очереди в кольцевом буфере*

Переменная **Начало** может принимать значение **Конец+1** в случаях как пустой, так и полной очереди. Чтобы различать эти ситуации, надо ввести еще одну переменную **Длина**. Для пустой очереди Длина = 0, а для полной очереди Длина = n+1.

1. **Спецификация программы.**

Программа предназначена для поиска всех потомков заданной персоны. Потомки должны выводиться в порядке: дети, внуки, правнуки и.т.д. Информация о потомках (массив string) и их родстве (двумерный массив bool) известна заранее и хранится в текстовых документах.

Программа написана на языке C++ с использованием компилятора mingw32- g++. Входными данными для программы являются:

элемент типа string – имя, которому ищем потомков.

1. **Начальные данные.**

В качестве начальных данных используем массив имен:

Ivan Oleg Petr Sergey Egor Kolya Ilya Anna Masha Katya Elena Vika

Gena

и двумерный массив родства (*Д* [*x*, *y*] = *true*, если человек по имени *y* является ребенком человека по имени *x*):

0 1 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 1 0 0 0 0 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 1 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 0 0 0 0 1 0

0 0 0 1 1 0 0 0 0 0 0 0 0

0 1 1 0 0 0 0 0 0 0 1 1 1

0 0 0 0 0 0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 0 0 0 0

Каждому имени задается идентификатор равный его индексу в массиве names.

Покажем структуру родства:

Анна 7

Иван 0

Маша 8

Петр 2

Олег 1

Сергей 3

Егор 4

Илья 6

Коля 5

Катя 9

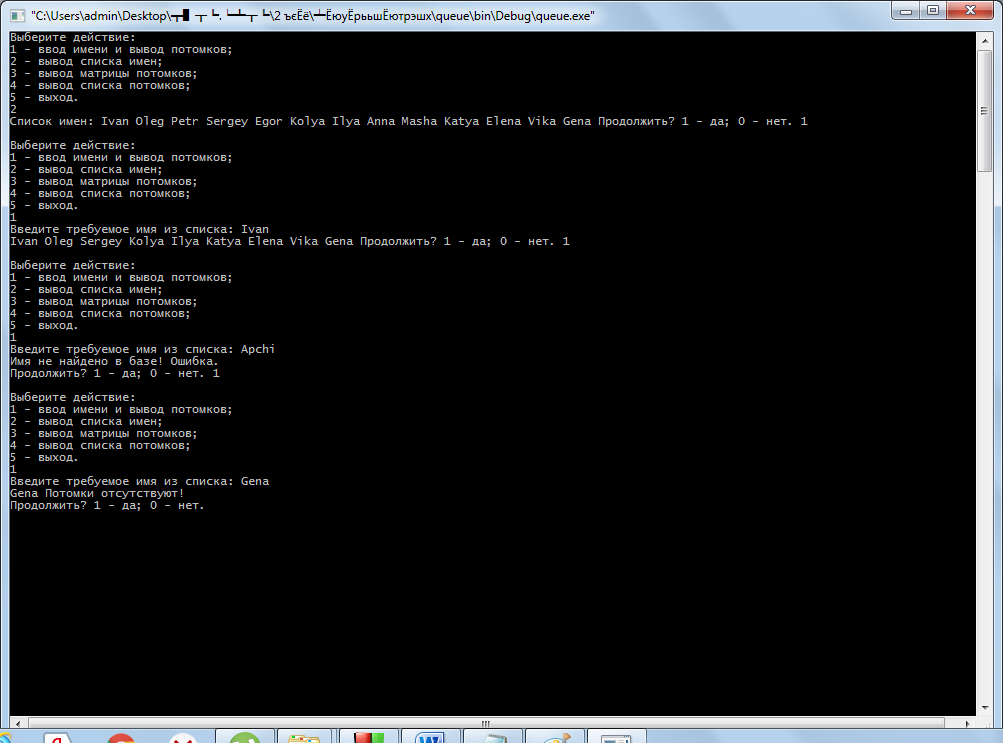
Елена 10

Вика 11

Гена 12

Ограничением на задание двумерного массива является проверка на d[i,i] !=0, !((d[i,j] = 1 || d[j,i] = 1) && (d[i,j]=d[j,i]) + потомок человека не должен являться его отцом (дедушкой и т.д.)

1. **Пример диалога с пользователем.**

****

1. **Реализация.**

Классы:

1. Очередь:

template <typename T>

class Queue

{

private:

T \*queuePtr; // указатель на очередь

const int size; // максимальное количество элементов в очереди

int begin, // начало очереди

end; // конец очереди

int elemCT; // счетчик элементов

public:

Queue(int = 10); // конструктор по умолчанию

Queue(const Queue<T> &); // конструктор копирования

~Queue(); // деструктор

void enqueue(const T &); // добавить элемент в очередь

T dequeue(); // удалить элемент из очереди

void printQueue(); // вывод очереди

bool isEmpty(); // проверка на пустоту

};

1. Класс names.

template <typename S>

class myNames

{

private:

S \*name; // массив имен

int \*numName; // массив их идентификаторов

bool \*\*arrayOfChildren; // двумерный массив родства

const int size; // размер

int countOfNames; // текущее количество имен

public:

myNames(int = 10); // конструктор

~myNames(); // деструктор

void delName(int my\_name); // удаление имени

void addNames(); // добавление имени

int getCnt();

S getName(int i);

int getNumName(S i);

bool realiseArray(); //создание 2мерного массива

bool getArray(int i, int j);

void printArray();

void printArray2();

int findNumName(S \_name); // нахождение идентификатора по имени

S findName(int \_numName); // нахождение имени по идентификатору

};

Функции:

bool findChld(int my\_num\_name, myNames<string> &\_names, Queue<string> &\_queue, std::ostream& out);

*Функция, которая выводит потомков заданного лица*

*предусловие: my\_num\_name – идентификатор заданного лица*

*\_names – класс имен*

*\_queue – очередь*

*возвращает true, если потомки есть, false – в обратном случае.*

1. **Тестирование.**

|  |  |
| --- | --- |
| Ivan | Oleg Sergey Kolya Ilya Katya Elena Vika Gena |
| Oleg | Kolya Ilya Elena Vika Gena |
| Petr | Kolya Elena |
| Sergey | Ilya Katya Vika Gena |
| Egor | Katya Vika Gena |
| Kolya | Elena |
| Ilya | Vika Gena |
| Anna | Sergey Egor Ilya Katya Vika Gena |
| Masha | Oleg Petr Elena Vika Gena Kolya Ilya |
| Katya | Vika Gena |
| Elena | Потомков нет |
| Vika | Gena |
| Gena | Потомков нет |

**Выводы****.**

В результате выполнения лабораторной работы мною были получены основные сведения о стеке и очереди. Я реализовал собственную очередь и использовал ее для решения задачи.

**Приложение А. Исходный код.**

**А.1. queue.h**

#ifndef QUEUE\_H\_INCLUDED

#define QUEUE\_H\_INCLUDED

#include <cassert>

#include <iostream>

using namespace std;

template <typename T>

class Queue

{

private:

T \*queuePtr; // указатель на очередь

const int size; // максимальное количество элементов в очереди

int begin, // начало очереди

end; // конец очереди

int elemCT; // счетчик элементов

public:

Queue(int = 10); // конструктор по умолчанию

Queue(const Queue<T> &); // конструктор копирования

~Queue(); // деструктор

void enqueue(const T &); // добавить элемент в очередь

T dequeue(); // удалить элемент из очереди

void printQueue(); // вывод очереди

bool isEmpty(); // проверка на пустоту

};

template<typename T>

bool Queue<T>::isEmpty()

{

if (begin == end)

return true;

return false;

}

// конструктор по умолчанию

template<typename T>

Queue<T>::Queue(int sizeQueue) :

size(sizeQueue), // инициализация константы

begin(0), end(0), elemCT(0)

{

// дополнительная позици поможет нам различать конец и начало очереди

queuePtr = new T[size + 1];

}

// конструктор копии

template<typename T>

Queue<T>::Queue(const Queue &otherQueue) :

size(otherQueue.size) , begin(otherQueue.begin),

end(otherQueue.end), elemCT(otherQueue.elemCT),

queuePtr(new T[size + 1])

{

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

queuePtr[ix] = otherQueue.queuePtr[ix]; // копируем очередь

}

// деструктор класса Queue

template<typename T>

Queue<T>::~Queue()

{

delete [] queuePtr;

}

// функция добавления элемента в очередь

template<typename T>

void Queue<T>::enqueue(const T &newElem)

{

assert( elemCT < size );

queuePtr[end++] = newElem;

elemCT++;

if (end > size)

end -= size + 1;

}

// функция удаления элемента из очереди

template<typename T>

T Queue<T>::dequeue()

{

assert( elemCT > 0 );

T returnValue = queuePtr[begin++];

elemCT--;

if (begin > size)

begin -= size + 1;

return returnValue;

}

template<typename T>

void Queue<T>::printQueue()

{

cout << "Очередь:";

if (end == 0 && begin == 0)

cout << " пустая\n";

else

{

for (int ix = begin; ix < end; ix++)

cout << queuePtr[ix] << " ";

cout << endl;

}

}

#endif // QUEUE\_H\_INCLUDED

**A.2. names.h**

#ifndef NAMES\_H\_INCLUDED

#define NAMES\_H\_INCLUDED

#include <fstream>

template <typename S>

class myNames

{

private:

S \*name; // массив имен

int \*numName; // массив их идентификаторов

bool \*\*arrayOfChildren; // двумерный массив родства

const int size; // размер

int countOfNames; // текущее количество имен

public:

myNames(int = 10); // конструктор

~myNames(); // деструктор

void delName(int my\_name); // удаление имени

void addNames(); // добавление имени

int getCnt();

S getName(int i);

int getNumName(S i);

bool realiseArray(); //создание 2мерного массива

bool getArray(int i, int j);

void printArray();

void printArray2();

int findNumName(S \_name); // нахождение идентификатора по имени

S findName(int \_numName); // нахождение имени по идентификатору

};

template <typename S>

myNames<S>::myNames(int sizeNames) :

size(sizeNames), countOfNames(0)

{

name = new S[size + 1];

numName = new int[size + 1];

arrayOfChildren = new bool \* [size + 1];

for (int i = 0; i < size + 1; i++)

arrayOfChildren[i] = new bool [size + 1];

}

template <typename S>

myNames<S>::~myNames()

{

delete [] name;

delete [] numName;

}

template <typename S>

void myNames<S>::addNames()

{

ifstream names( "names.txt");

while (names.peek()!= EOF) // גגמה טלום טח פאיכא

{

names >> name[countOfNames];

countOfNames++;

}

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

numName[i] = i;

}

template <typename S>

int myNames<S>::getCnt()

{

return countOfNames;

}

template <typename S>

S myNames<S>::getName(int i)

{

return name[i];

}

template <typename S>

int myNames<S>::getNumName(S iname)

{

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

{

if (iname == name[i])

return i;

}

return -1;

}

template <typename S>

bool myNames<S>::realiseArray()

{

bool flag = true;

ifstream in( "children.txt");

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

for (int j = 0; j < countOfNames; j++)

in >> arrayOfChildren[i][j];

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

for (int j = 0; j < countOfNames; j++)

{

if (arrayOfChildren[i][i] == 1)

flag = false;

if ((arrayOfChildren[i][j] == 1 || arrayOfChildren[j][i] == 1) && (arrayOfChildren[i][j] == arrayOfChildren[j][i]))

flag = false;

}

return flag;

}

template <typename S>

bool myNames<S>::getArray(int i, int j)

{

return arrayOfChildren[i][j];

}

template <typename S>

void myNames<S>::printArray()

{

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

{

for (int j = 0; j < countOfNames; j++)

{

cout << arrayOfChildren[i][j] << " ";

}

cout << endl;

}

}

template <typename S>

void myNames<S>::printArray2()

{

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

{

for (int j = 0; j < countOfNames; j++)

{

if (arrayOfChildren[i][j])

cout << "Descendant [ "<< name[i] << " ][ " << name[j] << " ] = " << arrayOfChildren[i][j] << endl;

}

}

}

template <typename S>

int myNames<S>::findNumName(S \_name)

{

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

if (\_name == name[i])

return numName[i];

return -1;

}

template <typename S>

S myNames<S>::findName(int \_numName)

{

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

if (\_numName == numName[i])

return name[i];

return "error";

}

template <typename S>

void myNames<S>::delName(int row)

{

for (int k = row; k < countOfNames - 1; k++)

{

for (int z = 0; z < countOfNames; z++)

{

arrayOfChildren[k][z] = arrayOfChildren[k + 1][z];

arrayOfChildren[z][k] = arrayOfChildren[z][k + 1];

}

}

bool flag = false;

int num = 0;

S my\_name = name[row];

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

if (my\_name == name[i])

{

flag = true;

num = i;

}

if (flag)

{

for (int i = num; i < countOfNames - 1; i++)

name[i] = name[i + 1];

countOfNames--;

}

}

#endif // NAMES\_H\_INCLUDED

**A.3. main.cpp**

#include "queue.h"

#include <clocale>

#include "names.h"

#define MAX 1000

using namespace std;

int readAction();

bool findChld(int my\_num\_name, myNames<string> &\_names, Queue<string> &\_queue, std::ostream& out);

int main()

{

ofstream out("output.txt");

setlocale(LC\_CTYPE, "rus");

myNames<string> \_names(MAX);

Queue<string> namesQueue(MAX);

string my\_name;

int my\_num\_name;

bool num;

int flag;

\_names.addNames();

if (!\_names.realiseArray())

{

cout << "Ошибка при составлении матрицы потомков!";

return -1;

}

while (true)

{

switch (readAction())

{

case 1:

cout << "Введите требуемое имя из списка: ";

cin >> my\_name;

my\_num\_name = \_names.findNumName(my\_name);

if (my\_num\_name == -1)

{

cout << "Имя не найдено в базе! Ошибка." << endl;

break;

}

num = findChld(my\_num\_name, \_names, namesQueue, cout);

if (!num)

cout << "Потомки отсутствуют!" << endl;

break;

case 2:

cout << "Список имен: ";

for (int i = 0; i < \_names.getCnt(); i++)

cout << \_names.getName(i) << " ";

break;

case 3:

cout << "Матрица потомков: " << endl;

\_names.printArray();

cout << endl;

break;

case 4:

cout << "Список потомков: " << endl;

\_names.printArray2();

break;

case 5:

return 0;

}

cout << "Продолжить? 1 - да; 0 - нет. ";

cin >> flag;

cout << endl;

if (flag == 0)

return 0;

}

}

bool findChld(int my\_num\_name, myNames<string> &\_names, Queue<string> &\_queue, std::ostream& out)

{

bool cnt = false;

\_queue.enqueue(\_names.getName(my\_num\_name));

bool used[\_names.getCnt()];

for (int i = 0; i < \_names.getCnt(); i++)

used[i] = 0;

int cur = my\_num\_name;

while(!\_queue.isEmpty())

{

cur = \_names.getNumName(\_queue.dequeue());

out << \_names.getName(cur) << " ";

for (int i = 0; i < \_names.getCnt(); i++)

if ((\_names.getArray(cur, i) == true) && !used[i])

{

cnt = true;

\_queue.enqueue(\_names.getName(i));

used[i] = 1;

}

}

return cnt;

}

int readAction()

{

while (true)

{

cout <<"Выберите действие:\n1 - ввод имени и вывод потомков;\n2 - вывод списка имен;\n3 - вывод матрицы потомков;\n4 - вывод списка потомков;\n5 - выход.\n";

int action;

cin >> action;

if (action > 5 || action < 1)

{

printf("Ошибка!\n");

continue;

}

return action;

}

}