МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)
Институт информационных технологий, математики и механики

ЛАБОРАТОРНАЯ РАБОТА
«Поразрядная сортировка»

Выполнил: студент группы 381603-4

Леонтьев Вадим Евгеньевич

Подпись _____

Руководитель:

Барышева Ирина Викторовна

Подпись _____

Нижний Новгород
2017

Оглавление

Введение

Линейный список — это динамическая структура данных, каждый элемент которой посредством указателя связывается со следующим элементом.

Реальное многообразие структур данных базируется всего на двух основных способах получения адреса хранимого элемента: вычисление (массив) и хранение (указатель). До сих пор основной компонентой структуры данных являлся массив (обычный массив, массив указателей). Если же попытаться построить структуру данных, исходя только из указателей, то получается цепочка (последовательность) элементов, содержащих указатели друг на друга. В простейшем случае она может быть линейной (список), в более сложных случаях – ветвящейся (деревья, графы). Итак, список – линейная последовательность элементов, каждый из которых содержит указатели (ссылается) на своих соседей (рисунок 1).

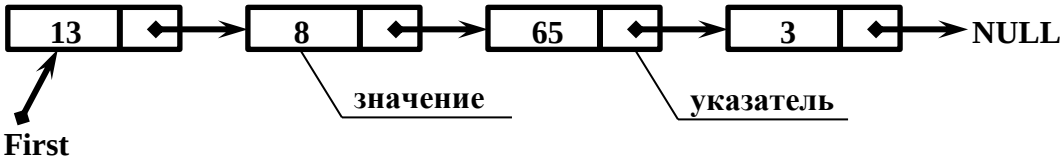

Рисунок 1. Схема линейного списка данных.

Сразу же отметим основную особенность: физическое размещение в памяти элементов списка не имеет никакого значения, все определяется наличием ссылок на него в других элементах и извне. У массива всегда есть «начало». У списка по определению отсутствует фиксированная привязка к памяти. Перечислим основные релятивистские свойства списка:

– элемент списка доступен в программе через указатель. «Смысл» этого указателя отражает функциональное назначение элемента списка в программе: первый, последний, текущий, предыдущий, новый и т.п. Между указателем и элементом списка имеется такая же взаимосвязь, как между индексом в массиве и элементом массива;

– в программе список задается посредством заголовка – указателя на первый элемент списка;

– порядок следования элементов определяется последовательностью связей между элементами. Изменение порядка следования элементов (вставка, удаление) осуществляются изменением переустановкой указателей на соседние элементы.

– логический (порядковый) номер элемента списка также задается его естественной нумерацией в цепочке элементов;

– список является структурой данных с последовательным доступом. Для получения n-го по счету элемента необходимо последовательно пройти по цепочке от элемента, на который имеется указатель (например, от заголовка);

– список удобен для использования именно как динамическая структура данных: элементы списка обычно создаются как динамические переменные, а связи между ними устанавливаются программно (динамически);

– список обладает свойством локальности изменений: при вставке/удалении элемента изменения касаются только текущего и его соседей. Вспомним массив: при вставке/удалении его элементов происходит сдвиг всех элементов, от текущего до конца.

Отсюда следует, что преимущества списков проявляются в таких структурах данных, где операции изменения порядка превалируют над операциями доступа и поиска, что и разбирается в данной лабораторной работе.

В зависимости от метода доступа к элементам линейного списка различают разновидности линейных списков называемые стеком, очередью и двусторонней очередью.

Очередь - это линейный список, где элементы удаляются из начала списка, а добавляются в конце списка (как обыкновенная очередь в магазине).

Поразрядная сортировка – алгоритм сортировки, выполняющийся за линейное время.

Постановка задачи

Задание на проект

Целью данной работы является демонстрация работы поразрядной сортировки на линейных списках.

Исходные данные: Строка содержащие все числа, которые нужно сортировать; Строка поэтапно сортируемая в результате полной сортировки (для наглядного просмотра действия алгоритма сортировки).

Описание алгоритма:

1. Делим строку на слова, получая массив строк содержащих переменные;
2. Ищем максимум среди длин элементов массива переменных;
3. Для каждого элемента массива переменных добавляем « 0 » и целью равенства длинн каждой из переменных;
4. Для каждого из максимальных символов (разряда), начиная с последнего:
   a) для каждого элемента массива переменных:
      I. выделим i-ый символ в переменную L;
      II. положим эту переменную в очередь и номером L;
      III. положим эту переменную в L-ый столбец DataGritView;
   b) если нажата кнопка «to step», сделаем задержку;
   c) положим к=0;
   d) Для каждой очереди:
      I. Пока очередь не пуста, занесём значение из неё в очередные элементы массива переменных;
   e) отчистить DataGritView;

Обсуждение структуры данных

Очередь – динамическая структура данных в которой операция взять и положить организованна по правилу – «первый вошедший – первым уходит». Способы организации:

1. Очередь с перепаковкой
   Структура хранения – массив
   – Берём первый элемент, а все остальные сдвигаем на 1 элемент
   – Добавляем элементы только в конец
   – При этом необходим контроль пустоты (по количеству элементов)
2. Кольцевой буфер
   Структура хранения – массив
   – Берём элемент с номером First
   – Добавляем после Last (рисунок 2)

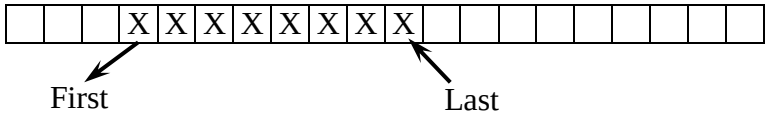

Рисунок 2. Схема структуры очереди «Кольцевой буфер»

– Контроль пустоты и полноты осуществляется по счётчику Count (количество элементов, заполняющих массив). Возможные варианты такой структуры приведены на рисунке 3.

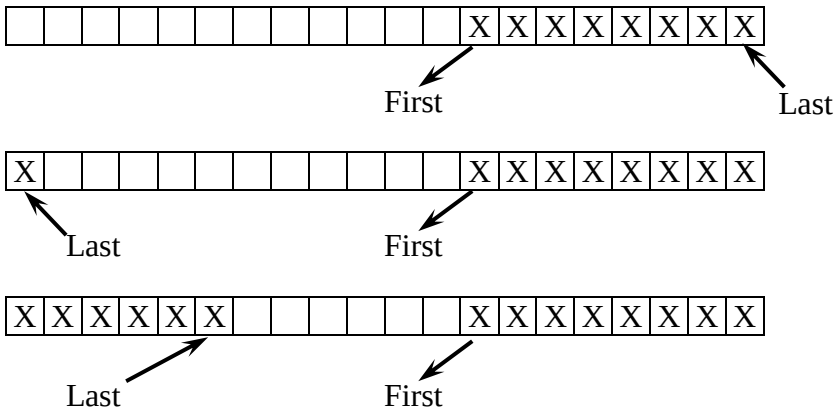

Рисунок 3. Возможные случаи «Кольцевого буфера»

3. Линейный список

Структура хранения – линейный список с ограниченным доступом (рисунок 1)

– Берём info из First

– Добавляем после Last

– Необходим контроль пустоты

Цель данного проекта – демонстрация работы поразрядной сортировки на линейном списке. Линейный список выбран не случайно. В моей лабораторной работе сортируются числа, а соответственно сортировать будем по 10и разрядам (от 0 до 9). В наихудшем варианте может получится такое, что все N переменных будут занимать N строк под одним разрядом (примерно как на рисунке 7), а это значит, что в структуре хранения «массив» нам понадобится место в памяти N*N, при котором большое количество данных сделает невозможным продолжение работы программы. Линейный список устраняет такую проблему, так как памяти на каждой итерации работы программы под данные требуется ровно под N переменных (с указателями на следующий элемент).

Поразрядная сортировка выполняется за линейное время, количество итераций которой равняется максимальной длины «слова», введённых в качестве исходных данных. Из-за большой эффективности использования памяти на больших входных данных и за возможность быстро отсортировать данные была выбрана структура очереди на линейном списке.

Структура проекта

Для решения поставленной задачи разработан проект со структурой построения классов UML (Рисунок 4).

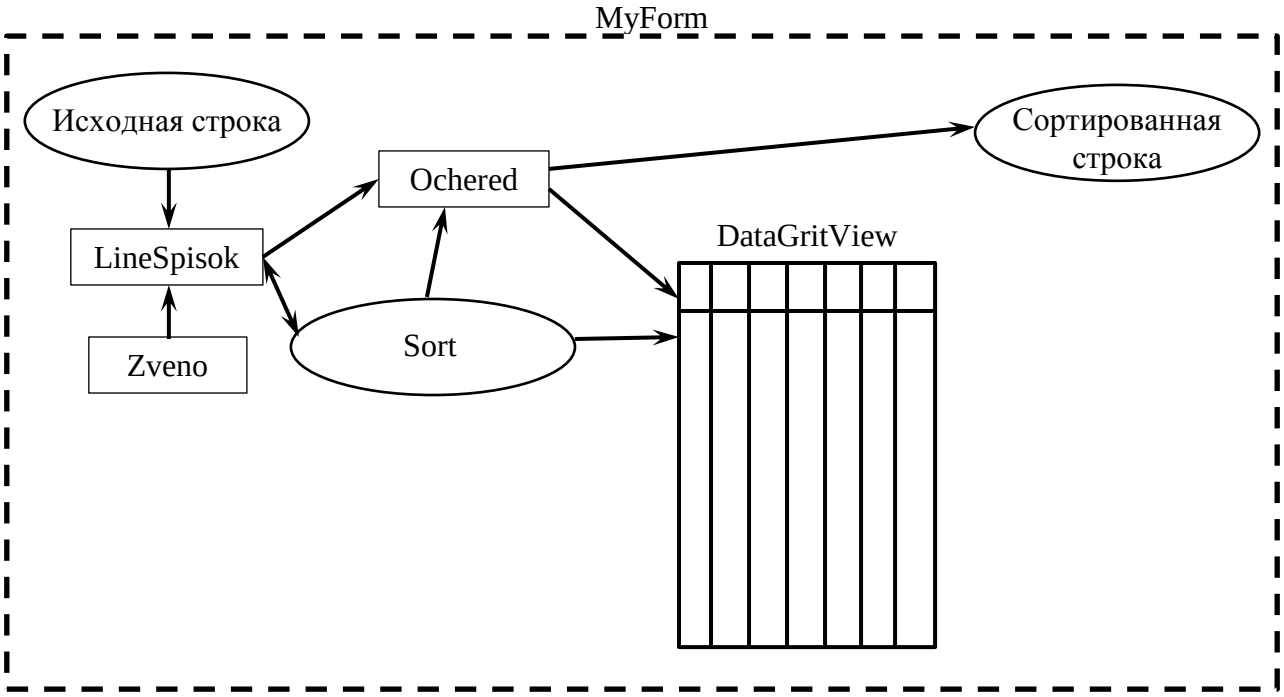

Рисунок 4. Схема проекта.

Проект располагается в форме (Windows Form) описание которой находится в Приложении «Windows Form». Лабораторная работа состоит из классов:

1. Zveno, состоящее из полей информация и указателя на следующий элемент. Этот класс сформировывает один элемент линейного списка (класс описан в приложении «Класс Zveno»).
2. LineSpisok служит для формирования из звеньев линейный список (описание класса находится в приложении «Класс LineSpisok»), и имеет указатель на первое звено списка, с возможностью последующего добавления звеньев в список.
3. Ochered, который спроектирован для организации поразрядной сортировки с использование динамической структуры «Очередь». Описание класса приведено в приложении «Класс Ochered»

Метод Sort, который сортирует линейный список данных, создавая, в конечном итоге, отсортированный линейный список.

На схеме проекта изображено как элементы среды Windows Form взаимодействуют с абстрактными классами, а именно:

Строка переменных, которую ввёл пользователь, преобразуется в список переменных.

В зависимости от нажатой кнопки (рисунок 6), при помощи метода Sort будет создан отсортированный список (с возможностью выводить промежуточные данные в DataGritView). Кнопки будут влиять на то – увидит ли пользователь промежуточные результаты сортировки, или же сразу захочет получить отсортированные данные. В конечном итоге отсортированный список (или промежуточные данные) будет преобразован в строку отсортированных элементов в окне программы.

Инструкция для пользователя

После запуска программы появиться окно программы (рисунок 5).

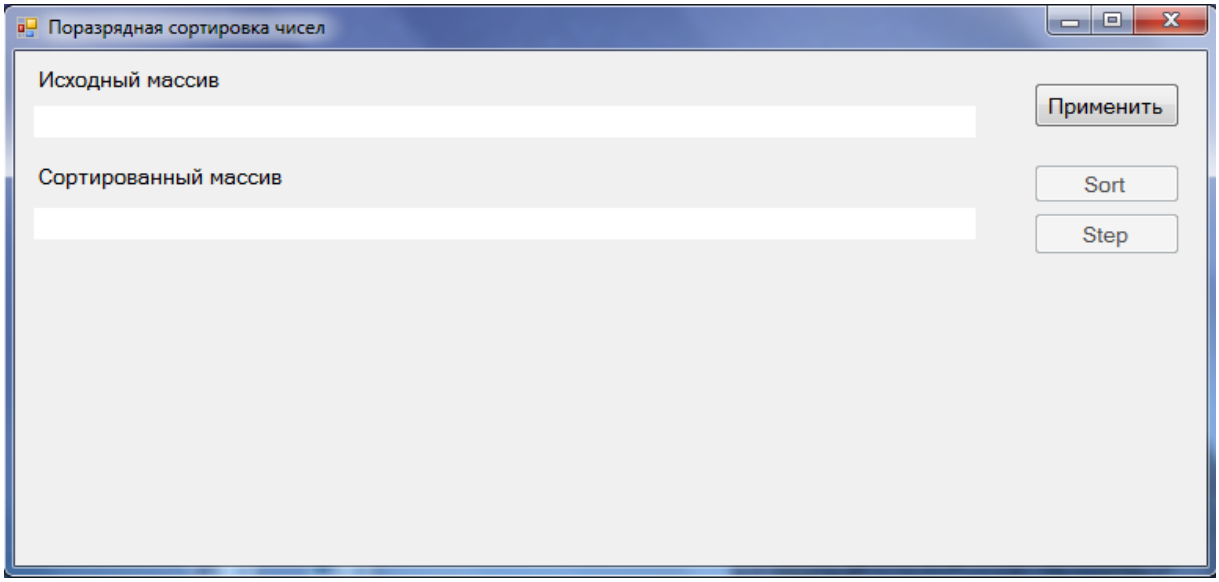

Рисунок 5. Окно программы

В сроку - «Исходный массив» необходимо ввести числовые данные, которые необходимо отсортировать. После чего нажать на кнопку «Применить». Этой кнопкой будет сформирован линейный список из полученной строки и таблица, для просмотра промежуточных результатов сортировки данных (рисунок 6).

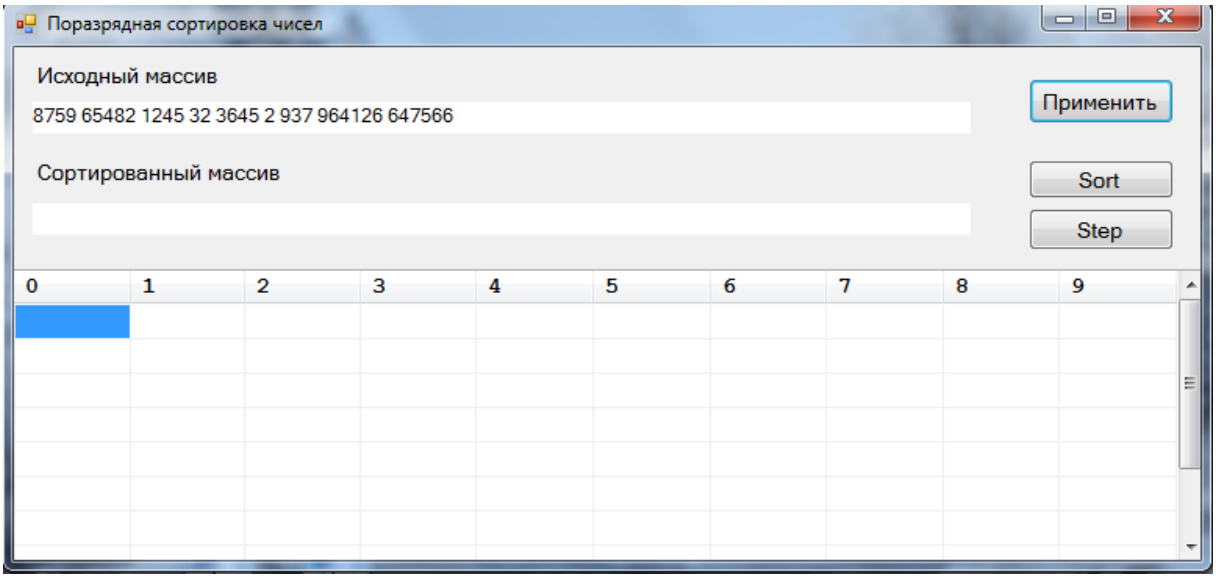

Рисунок 6. Окно программы с введёнными данными

Затем, чтобы увидеть промежуточные результаты сортировки нужно нажать на кнопку «Step», а для мгновенного получения отсортированной строки нажать на кнопку «Start» (рис. 7).

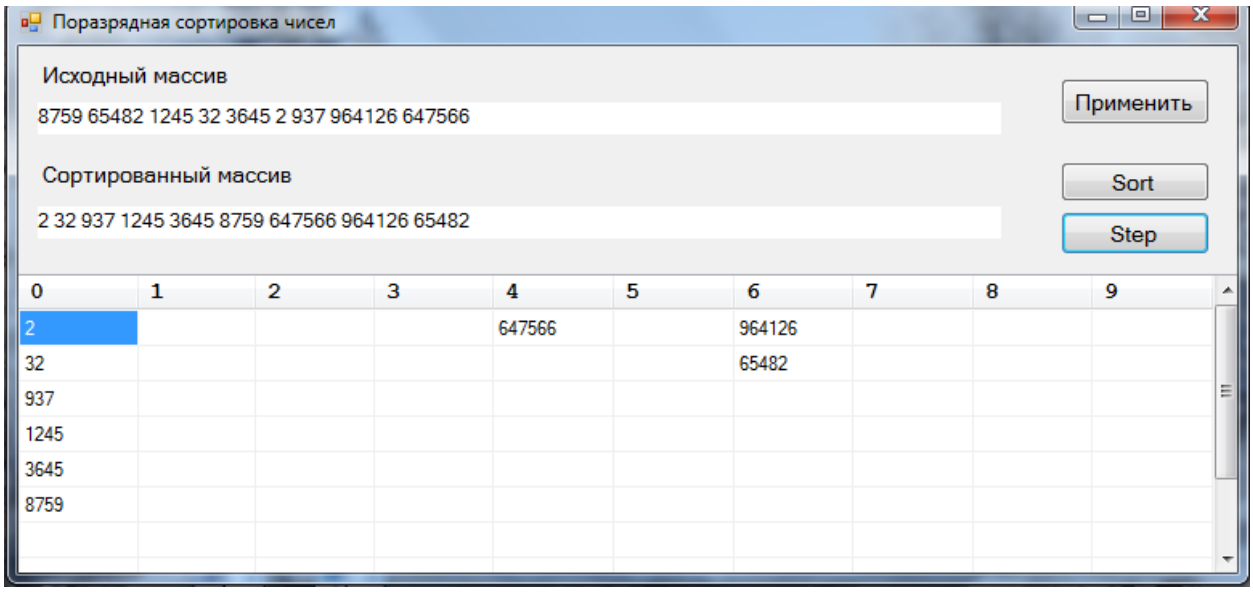

Рисунок 7. Окно программы с возможностью просмотра промежуточного результата.

По завершению программы в строке «Сортированный массив» появится строка отсортированных исходных данных по возрастанию (рисунок 8).

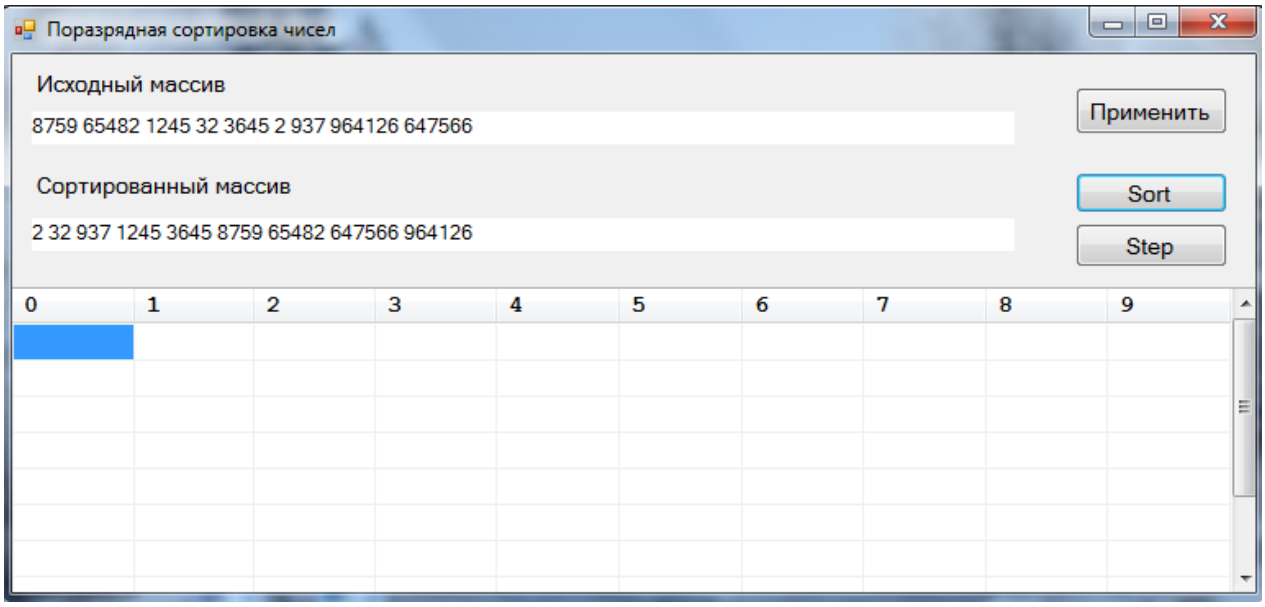

Рисунок 8. Окно программы с отсортированными исходными данными.

Программа завершила сортировку данных, а для сортировки другого массива данных нужно повторить все пункты, описанные выше.

Контрольный пример

Контрольный пример привожу на массиве чисел:

« 8759 65482 1245 32 3645 2 937 964126 647566 »

В ходе проверки при нажатии на кнопку «Step» должны появляться промежуточные результаты (рисунок 7), а при нажатии на кнопку «Start» сразу выводится результат, что и происходит.

Поставленный эксперимент (рисунок 8) показывает верное построение проекта, и правильное взаимодействие классов между собой и средой Windows Form.

Заключение

В данном проекте было продемонстрирована поразрядная сортировка на линейных списках.

В ходе выполнения работы мною были спроектированы такие структуры данных как – очередь и линейный список.

В процессе ознакомления с поставленной задачей, я проанализировал возможные структуры хранения типа – очередь, и подобрал наиболее эффективную, работа которой не будет затрагивать много времени и памяти.

Так же был составлена структура проекта, в которой было объяснено применения каждого класса и каждого метода для решения поставленной задачи.

Вычисленный контрольный пример продемонстрировал эффективную и бесперебойную работу программы, с ожидаемыми выходными данными.

При решении задачи были использованы средства среды Windows Forms и продемонстрировано применение инструментов визуального программирования для упрощения программирования пользовательского интерфейса, благодаря которой лабораторная работа эстетично выглядит.

В программе не выявлены серьёзные проблемы, и она полностью справилась с выполнением поставленной задачи.

Список литературы

1. Павловская Т. А. C/C++. Программирование на языке высокого уровня. — СПб.:Питер, 2003. – 461
2. Буч Г. Объектно-ориентированный анализ и проектирование с применением приложений. – М : «Вильямс», 2008 – 720 стр.
3. Зиборов В.В. MS Visual C++ 2010 в среде .NET.Библиотека програмиста. –СПБ.: Питер, 2012. – 320 с.
4. Шилдт Г.С. C++. Базовый курс. – М.: «Вильямс», 2010. – 624
5. Сабуров С.В. Языки программирования С и С++. _ М.: Бук_пресс, 2006. – 647 с.
6. Шень А. Программирование: теоремы и задачи. – М.: МЦНМО, 2004. – 296
7. Окулов С. М. Программирование в алгоритмах – М.: БИНОМ, 2002. – 341

Интернет-ресурсы:

1. https://habrahabr.ru/post/113625/
2. https://ru.wikipedia.org/wiki/Очередь
3. https://ru.wikipedia.org/wiki/Линейный список
4. http://ermak.cs.nstu.ru/cprog/html/063.htm
5. https://www.intuit.ru/studies/courses/40/40/lecture/1225
6. http://victor192007.narod.ru/files/cpp_d1.html
7. http://forum.sources.ru/index.php?showtopic=56949
8. https://ru.wikipedia.org/wiki/Поразрядная_сортировка

Приложение

Класс Zveno

```
#ifndef __Zveno_Spiska___
#define __Zveno_Spiska___
#include <stdio.h>
class Zveno{
int info;
Zveno* Next;
public:
//Конструктор
Zveno(int _info = 0): info(_info), Next(nullptr) {}
//Конструктор копирования
Zveno(const Zveno &tmp): info(tmp.info), Next(nullptr) {}
//Деструктор
~Zveno(void){
if(Next != nullptr){
delete Next;}}
//Оператор присваивания
Zveno& operator =(const Zveno &tmp){
if(Next != nullptr){
delete Next;
Next = nullptr;}
info = tmp.info;
return *this;}
//Получить значение info
int getInfo(void) { return info; }
//Получить адрес следующего звена
Zveno* getNext(void) { return Next; }
//Установить адрес следующего звена
void setNext(Zveno* __Next) { Next = __Next; }};
#endif //__Zveno_Spiska___
```

Класс LineSpisok

```
#ifndef __My_Line_Spisok___
#define __My_Line_Spisok___
class lineSpisok{
Zveno* first;
public:
//Конструктор
lineSpisok(void): first(nullptr){}
//Конструктор копирования
lineSpisok(const lineSpisok &tmp){
if(nullptr == tmp.first) { first = nullptr; }
else{
first = new Zveno(*(tmp.first));
Zveno* cur = tmp.first->getNext();
Zveno* last = first;
while(cur != nullptr){
```

```
last->setNext(new Zveno(*cur));
last = last->getNext();
cur = cur->getNext();}}}
//Деструктор
~lineSpisok(void){
if(first != nullptr) { delete first; }}
//Операторприсваивания
lineSpisok& operator =(const lineSpisok &tmp){
if(first != nullptr) { delete first; }
first = nullptr;
if(tmp.first != nullptr){
first = new Zveno(*(tmp.first));
Zveno* cur = tmp.first->getNext();
Zveno* last = first;
while(cur != nullptr){
last->setNext(new Zveno(*cur));
last = last->getNext();
cur = cur->getNext();}}
return *this;}
//Добавить новое звено в конец списка
void add(const int info_){
Zveno* p = new Zveno(info_);
if(nullptr == first) { first = p; }
else{
Zveno* last = first;
while(last->getNext() != nullptr) { last = last->getNext(); }
last->setNext(p);}}
//Удалить звено из списка (с пустым полем 'значения')
void del(void){
if(first != nullptr){
while((first != nullptr) && (0 == first->getInfo())){
Zveno* temp = first;
first = first->getNext();
temp->setNext(nullptr);
delete temp;}
if(first != nullptr){
Zveno* prev = first;
while(prev->getNext() != nullptr){
Zveno* cur = prev->getNext();
if(0 == cur->getInfo()){
prev->setNext(cur->getNext());
cur->setNext(nullptr);
delete cur;}
else { prev = prev->getNext(); }}}}}
//Создать новый список с упорядоченой по возрастанию 'значением'
lineSpisok newSortSpisok(const lineSpisok &sp){
lineSpisok rez;
if(sp.first != nullptr){
Zveno* p = new Zveno(*(sp.first));
Zveno* cur = sp.first->getNext();
while(cur != nullptr){
Zveno* newCur = new Zveno(*cur);
```

```
if(p->getInfo() >= newCur->getInfo()){
newCur->setNext(p);
p = newCur;}
else{
Zveno* prev = p;
Zveno* newCurent = prev->getNext();
while((newCurent != nullptr) && ((newCur->getInfo()) > newCurent->getInfo())){
prev = newCur;
newCur = newCur->getNext();}
prev->setNext(newCurent);
newCurent->setNext(newCur);}
cur = cur->getNext();}
rez.first = p;}
return rez;}};
#endif //__My_Line_Spisok___
```

Класс Ochered

```
#include "TQueue.h"
TQueue::TQueue() :LinearSpk(){
count = 0;
last = NULL;}
TQueue::TQueue(const TQueue &tmp){
first = NULL;
last = NULL;
count = 0;
TRecord *cur = tmp.first;
while (cur != NULL){
this->Add(cur->getInf());
cur = cur->getNext();}}
TQueue::~TQueue(){}
TQueue & TQueue::operator = (const TQueue &tmp){
if (count != 0) delete first;
first = NULL;
last = NULL;
count = 0;
TRecord *cur = tmp.first;
while (cur != NULL){
this->Add(cur->getInf());
cur = cur->getNext();}
return *this;}
//методы
void TQueue::Add(int numb){
TRecord *tmp = new TRecord(numb);
if (count == 0){
first = tmp;
last = tmp;
count = 1;}
else{
last->setNext(tmp);
last = tmp;
count++;}}
```

```
int TQueue::Del(){
if (first == NULL) return 0;
else{
TRecord *tmp = first;
first = first->getNext();
tmp->setNext(NULL);
int numb = tmp->getInf();
delete tmp;
count--;
if (count == 0) last = NULL;
return numb;}}
int TQueue::isEmpty(){
if (count > 0) return 0;
else return 1;}
```

Метод Sort

```
void radSortStep(int *array, int step, int qNumb, DataGridView^ table){
TQueue queue[10];
int n = 0;
for (int i = 0; i < qNumb; i++){
int item = 0;
int tmp = array[i];
for (int j = 0; j < step; j++){
item = tmp % 10;
tmp = tmp / 10;}
queue[item].Add(array[i]);
table[item, queue[item].getCount() - 1]->Value = array[i];}
for (int i = 0; i < 10; i++){
while (queue[i].isEmpty() != 1){
array[n++] = queue[i].Del();}}}
```

Windows Form

```
#pragma once
#include "TQueue.h"
#include "Source.h"
namespace Sort {
// Column1
this->Column1->HeaderText = L"0";
this->Column1->Name = L"Column1";
// Column2
this->Column2->HeaderText = L"1";
this->Column2->Name = L"Column2";
// Column3
this->Column3->HeaderText = L"2";
this->Column3->Name = L"Column3";
// Column4
this->Column4->HeaderText = L"3";
this->Column4->Name = L"Column4";
// Column5
```

```
this->Column5->HeaderText = L"4";
this->Column5->Name = L"Column5";
// Column6
this->Column6->HeaderText = L"5";
this->Column6->Name = L"Column6";
// Column7
this->Column7->HeaderText = L"6";
this->Column7->Name = L"Column7";
// Column8
this->Column8->HeaderText = L"7";
this->Column8->Name = L"Column8";
// Column9
this->Column9->HeaderText = L"8";
this->Column9->Name = L"Column9";
// Column10
this->Column10->HeaderText = L"9";
this->Column10->Name = L"Column10";
#pragma endregion
public:
int qArray;
int *array;
int qNumb;
int step;
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
 button2->Enabled = true;
 button1->Enabled = true;
 step = 1;
 dataGridView1->Visible = true;
 dataGridView1->RowCount = 10;
 string name = SystemToStd(textBox1->Text);
 string *word = new string[name.length() / 2 + 1];
 name += " ";
 qArray = StringSeparation(name, word);
 array = new int[qArray];
 for (int i = 0; i < qArray; i++) {
 array[i] = atoi(word[i].c_str()); }
 int maxNumb = word[1].length();
 for (int i = 0; i < qArray; i++) {
 if (maxNumb < word[i].length()) {
 maxNumb = word[i].length();}}
 qNumb = maxNumb;
 delete[] word; }
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
 clearTable(dataGridView1);
 if (step <= qNumb) {
 radSortStep(array, step++, qArray, dataGridView1);
 textBox2->Text = gcnew System::String(arrayToStr(array, qArray).c_str()); }
 else {
  MessageBox::Show("Complete!", "", MessageBoxButtons::OK,
MessageBoxIcon::Asterisk);} }
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
 while (step <= qNumb) {
```

```
 radSortStep(array, step++, qArray, dataGridView1);
 textBox2->Text = gcnew System::String(arrayToStr(array, qArray).c_str()); }
 clearTable(dataGridView1); }
private: System::Void SortForm_Load(System::Object^  sender, System::EventArgs^  e) {
 button2->Enabled = false;
 button1->Enabled = false; }};}
```