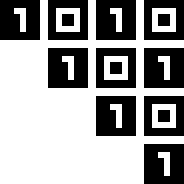
Министерство науки и высшего образования РФ

ФГБОУ ВО «Тамбовский государственный технический университет»

Кафедра

«САПР»



Отчет по лабораторной работе №2

по дисциплине «Алгоритмизация, модели данных и основы ИИ»

Выполнил: студент группы БВТ241

Белов А.К.

Проверил: преподаватель

Патутин К. И.

Тамбов 2025

**Цель работы:**42345  
**Заданиe:**456456  
**Решение:**  
 Связный список состоит из класса контейнера и классов элементов. каждый имеет полое с данными и указатели на прошлый и следующий элемент, нулевой явзяется экземпляром класса, без него работа невозможна. имеются функции добавления и удаленияб показа зависимости двух параметров  
**Code: ConsoleAlgorithms.cpp**  
#define \_CRT\_SECURE\_NO\_WARNINGS   
  
#include <iostream>  
#include <string>  
#include <sstream>  
#include <vector>  
  
#include <windows.h>  
#include <cstdlib>  
  
#include "filework.h"  
#include "laba\_1.h"  
#include "laba\_2.h"  
#include "laba\_3.h"  
#include "laba\_4.h"  
#include "laba\_5.h"  
  
#define LABSCOUNT 4  
  
#define TEST\_MODE  
  
void MenuLab4()  
{  
 system("cls");  
 cout << "Цель работы. Освоить основы работы со связными списками в языке С++." << mcl::endl;  
 cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << mcl::endl;  
  
 int select = 1;  
 while (!(GetAsyncKeyState(VK\_SPACE) & 0x8000))  
 {  
 if (GetAsyncKeyState(VK\_TAB) & 0x8000)  
 {  
 select++;  
 if (select > 1)  
 select = 1;  
 Sleep(70);  
 }  
 for (int l = 1; l <= 1; l++)  
 {  
 if (select == l)  
 {  
 cout(0, l + 5) << " >Task #" \* select\_color << l << mcl::endl;  
 }  
 else {  
 cout(0, l + 5) << "task #" \* data\_color << l << " " << mcl::endl;  
 }  
 }  
 }  
 switch (select)  
 {  
 case 1:  
 system("cls");  
 Sleep(500);  
 l4task1();  
 break;  
 default:  
 break;  
 }  
  
}  
void MenuLab3()  
{  
 system("cls");  
 cout << "Цель работы. Освоить основы работы с классами в языке С++." << mcl::endl;  
 cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << mcl::endl;  
  
 int select = 1;  
 while (!(GetAsyncKeyState(VK\_SPACE) & 0x8000))  
 {  
 if (GetAsyncKeyState(VK\_TAB) & 0x8000)  
 {  
 select++;  
 if (select > 1)  
 select = 1;  
 Sleep(70);  
 }  
 for (int l = 1; l <= 1; l++)  
 {  
 if (select == l)  
 {  
 cout(0, l + 5) << " >Task #" \* select\_color << l << mcl::endl;  
 } else {  
 cout(0, l + 5) << "task #" \* data\_color << l << " " << mcl::endl;  
 }  
 }  
 }  
 switch (select)  
 {  
 case 1:  
 system("cls");  
 Sleep(500);  
 l3task1();  
 break;  
 default:  
 break;  
 }  
   
}  
  
void MenuLab2()  
{  
 system("cls");  
 cout << "Цель работы.Освоить основы работы с базой данных в языке С++." << mcl::endl;  
 cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << mcl::endl;  
  
 int select = 1;  
 while (!(GetAsyncKeyState(VK\_SPACE) & 0x8000))  
 {  
 if (GetAsyncKeyState(VK\_TAB) & 0x8000)  
 {  
 select++;  
 if (select > 1)  
 select = 1;  
 Sleep(70);  
 }  
 for (int l = 1; l <= 1; l++)  
 {  
 if (select == l)  
 {  
 cout(0, l + 5) << " >Task #" \* select\_color << l << mcl::endl;  
 } else {  
 cout(0, l + 5) << "task #" \* data\_color << l << " " << mcl::endl;  
 }  
 }  
 }  
  
 switch (select)  
 {  
 case 1:  
 system("cls");  
 Sleep(500);  
 l2task1();  
 break;  
 default:  
 break;  
 }  
  
}  
  
void MenuLab1()  
{  
 system("cls");  
 cout << "Цель работы.Освоить основы работы с базовой системой ввода - вывода в языке С++." << mcl::endl;  
 cout << "1. Дан текстовый файл. Удалить из него третью строку. Результат записать в другой файл;" << mcl::endl;  
 cout << "2. Имеется текстовый файл, в каждой строке которого первые два символа \nявляются буквами.Получить слово, образованное первыми буквами каждой \nстроки;" << mcl::endl;  
 cout << "3. Дан текстовый файл.Переписать его строки в обратном порядке(справа \nналево) в другой файл.Порядок строк во втором файле должен совпадать с \nпорядком строк в заданном файле;" << mcl::endl;  
 cout << "4. Имеется типизированный файл, в котором записаны восемнадцать целых \nчисел.Переписать все положительные числа файла в массив в том же \nпорядке;" << mcl::endl;  
 cout << "5. Дан типизированный файл, элементами которого являются отдельные \nслова.Найти количество слов, начинающихся на букву ‘м’(кириллица);" << mcl::endl;  
 cout << "6. Имеется типизированный файл, элементами которого являются целые \nчисла.Все четные числа этого файла записать во второй файл, а нечетные – в \nтретий файл.Порядок следования чисел сохраняется." << mcl::endl;  
 int select = 1;  
 while (!(GetAsyncKeyState(VK\_SPACE) & 0x8000))  
 {  
 if (GetAsyncKeyState(VK\_TAB) & 0x8000)  
 {  
 select++;  
 if (select > 6)  
 select = 1;  
 Sleep(70);  
 }  
 for (int l = 1; l <= 6; l++)  
 {  
 if (select == l)  
 {  
 cout(0, l + 15) << " >Task #" \* select\_color << l << mcl::endl;  
 }  
 else  
 {  
 cout(0, l + 15) << "task #" \* data\_color << l << " " << mcl::endl;  
 }  
 }  
 }  
 system("cls");  
 switch (select)  
 {  
 case 1:  
 l1task1();  
 break;  
 case 2:  
 l1task2();  
 break;  
 case 3:  
 l1task3();  
 break;  
 case 4:  
 l1task4();  
 break;  
 case 5:  
 l1task5();  
 break;  
 case 6:  
 l1task6();  
 break;  
 default:  
 break;  
 }  
}  
  
void MainMenu()  
{  
 int select = 1;  
 cout << "Hello! \n This program was made for the assembly of laboratory works." \* data\_color << mcl::endl;  
 cout << "Use and to switching and select " << mcl::endl;  
 cout << "Now you can select one of " << LABSCOUNT << " labs:" << mcl::endl;  
 while (!(GetAsyncKeyState(VK\_SPACE) & 0x8000))  
 {  
 if (GetAsyncKeyState(VK\_TAB) & 0x8000)  
 {  
 select++;  
 if (select > LABSCOUNT)  
 select = 1;  
 Sleep(90);  
 }  
 for (int l = 1; l <= LABSCOUNT; l++)  
 {  
 if (select == l)  
 {  
 cout(0, l + 4) << " >laboratory work #" \* select\_color << l << mcl::endl;  
 }  
 else  
 {  
 cout(0, l + 4) << "laboratory work #" \* data\_color << l << " " << mcl::endl;  
 }  
 }  
 }  
 cout(0, LABSCOUNT + 5) << "You select to veiw laboratory work #" \* data\_color << select << mcl::endl;  
 Sleep(300);  
  
 switch (select)  
 {  
 case 1:  
 MenuLab1();  
 break;  
 case 2:  
 MenuLab2();  
 break;  
 case 3:  
 MenuLab3();  
 break;  
 case 4:  
 MenuLab4();  
 break;  
 default:  
 cout(0, LABSCOUNT + 5) << "Wrong select" \* error\_color << mcl::endl;  
 break;  
 }  
}  
  
int new\_booking::last\_id = 0;  
int person::last\_id = 0;  
  
  
  
int main()  
{  
 SetConsoleCP(1251);  
 SetConsoleOutputCP(1251);  
  
  
#ifdef TEST\_MODE  
   
   
 selection Selection;  
 exchange Exchange;  
 insert Insert;  
 shell Shell;  
 merger Merger;  
 fast Fast;  
 heap Heap;  
 radix Radix;  
  
  
 sort\* sorts[8] = {&Selection , &Exchange , &Insert   
 , &Shell , &Merger , &Fast , &Heap , &Radix };  
  
  
 arr\_for\_sort random\_arr\_30(SMALL\_ARR\_SIZE);  
 arr\_for\_sort good\_arr\_30(SMALL\_ARR\_SIZE);  
 arr\_for\_sort bad\_arr\_30(SMALL\_ARR\_SIZE);  
 for (int i = 0; i < SMALL\_ARR\_SIZE; i++)  
 {  
 good\_arr\_30[i] = i;  
 bad\_arr\_30[i] = SMALL\_ARR\_SIZE-i;  
 }  
  
 arr\_for\_sort random\_arr\_100000(BIG\_ARR\_SIZE);  
 arr\_for\_sort good\_arr\_100000(BIG\_ARR\_SIZE);  
 arr\_for\_sort bad\_arr\_100000(BIG\_ARR\_SIZE);  
 for (int i = 0; i < BIG\_ARR\_SIZE; i++)  
 {  
 good\_arr\_100000[i] = i;  
 bad\_arr\_100000[i] = BIG\_ARR\_SIZE - i;  
 }  
  
  
 for (int i = 0; i < 8; i++)  
 {  
 cout << "\n=====================================================\t\t" << mcl::nsep;  
 sorts[i]->Info();  
 arr\_for\_sort \_random\_arr\_30 = random\_arr\_30;  
 arr\_for\_sort \_good\_arr\_30 = good\_arr\_30;  
 arr\_for\_sort \_bad\_arr\_30 = bad\_arr\_30;  
 SortCheck(\_random\_arr\_30, sorts[i]);  
 SortCheck(\_good\_arr\_30, sorts[i]);  
 SortCheck(\_bad\_arr\_30, sorts[i]);  
 }  
  
 SortCheck(random\_arr\_100000, &Heap);  
 SortCheck(good\_arr\_100000, &Heap);  
 SortCheck(bad\_arr\_100000, &Heap);  
  
  
#endif  
  
#ifndef TEST\_MODE  
  
 MainMenu();  
  
#endif  
  
  
  
}  
 **Code: laba\_5.cpp  
  
Code: laba\_5.h**#pragma once  
#include <iostream>  
#include <cstring>  
#include <windows.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <cstdlib>  
#include <conio.h>  
#include <sysinfoapi.h>  
  
#include "MultiColorLine.h"  
  
  
DWORD last\_time;  
  
#define TIME\_IT last\_time = GetTickCount();  
#define COMPARE\_TIME (int)(GetTickCount() - last\_time)  
  
#define SMALL\_ARR\_SIZE 30  
#define BIG\_ARR\_SIZE 100000  
  
enum debug\_lvl  
{  
 only\_sort\_result,   
 sort\_process,   
 inheritance\_process  
};  
  
 debug\_lvl debug = only\_sort\_result;  
  
  
enum sorts  
{  
  
 selection\_sort,  
 exchange\_sort,  
 insert\_sort,  
 shell\_sort,  
 merger\_sort,  
 fast\_sort,  
 heap\_sort,  
 radix\_sort  
};  
  
  
class array {  
public:  
 array(int len = 1)  
 {  
 data = nullptr;  
 Create(len);  
 }  
 array(array& another)  
 {  
 Create(another.size);  
 for(int i = 0; i < size; i++)  
 {   
 data[i] = another[i];  
 }  
 }  
 int& operator[](int index)  
 {  
 return data[index];  
 }  
  
 int\* data;  
 int size;  
 void Create(int len)  
 {  
 if ((data != nullptr)&&(size != len))  
 delete data;  
 data = new int[len];  
 size = len;  
 }  
 void Copy(int\* start, int len)  
 {  
 Create(len);  
 for (int i = 0; i < len; i++)  
 {  
 data[i] = start[i];  
 }  
 }  
 ~array()  
 {  
 if (data != nullptr)  
 delete[] data;  
 data = nullptr;  
 }  
 void Print()  
 {  
 for (int i = 0; i < size; i++)  
 {  
 cout << " | " << data[i] << mcl::nsep;  
 }  
 cout << " |" << mcl::endl;  
 }  
};  
  
  
class arr\_for\_sort : public array {  
public:  
 arr\_for\_sort(int len = 0) : array(len)  
 {  
 Random(len);  
 }  
 void Random(int len)  
 {  
 Create(len);  
 for (int i = 0; i < len; i++)  
 {  
 data[i] = rand() % 1000;  
 }  
 }  
 void Swap(int index\_1, int index\_2)  
 {  
 int tmp = data[index\_1];  
 data[index\_1] = data[index\_2];  
 data[index\_2] = tmp;  
 }  
 void Insert(int index\_from, int index\_to)  
 {  
 if (index\_to < index\_from)  
 {  
 int tmp = data[index\_from];  
 for (int i = index\_from; i >= index\_to; i--)  
 {  
 data[i] = data[i-1];  
 }  
 data[index\_to] = tmp;  
 }  
 else  
 {  
 int tmp = data[index\_from];  
 for (int i = index\_from; i <= index\_to; i++)  
 {  
 data[i] = data[i+1];  
 }  
 data[index\_to] = tmp;  
 }  
 }  
 bool Check(int index\_1, int index\_2)  
 {  
 return data[index\_1] > data[index\_2];  
 }  
 void Append(int num)  
 {  
 arr\_for\_sort tmp = \*this;  
 Create(size + 1);  
 for (int i = 0; i < size-1; i++)  
 {  
 data[i] = tmp[i];  
 }  
 data[size - 1] = num;  
 }  
 int Min(int i)  
 {  
 int min = 2000000, ind = size;  
 for (; i < size; i++)  
 {  
 if (min > data[i]) {  
 min = data[i];  
 ind = i;  
 }  
 }  
 return ind;  
 }  
 int Max(int i)  
 {  
 int max = 0, ind = size;  
 for (; i < size; i++)  
 {  
 if (max < data[i]) {  
 max = data[i];  
 ind = i;  
 }  
 }  
 return ind;  
 }  
 int Size()  
 {  
 return size;  
 }  
  
private:  
  
};  
  
  
class sort{  
public:  
 sort(sorts my\_type) : my\_type(my\_type)  
 { }  
//protected:  
 virtual void Sort(arr\_for\_sort& arr) { };  
 virtual void Info() {};  
private:  
 sorts my\_type;  
  
};  
  
class selection : public sort  
{  
public:  
 selection() : sort(selection\_sort)  
 { }  
 void Sort(arr\_for\_sort& arr) override  
 {  
 for(int i = 0; i < arr.Size()-1; i++)  
 {   
 arr.Swap(arr.Min(i), i);  
 if (debug >= sort\_process)  
 arr.Print();  
 }  
 }  
 void Info() override  
 {  
 cout << "selection sort \n\t- sorts by moving the minimum element to its place" << mcl::endl;  
 }  
private:  
};  
  
class exchange : public sort  
{  
public:  
 exchange() : sort(exchange\_sort)  
 { }  
 void Sort(arr\_for\_sort& arr) override  
 {  
 bool ready = true;  
 while (ready)  
 {  
 ready = false;  
 for (int i = 0; i < arr.Size() - 1; i++)  
 {  
 if (arr.Check(i, i + 1))  
 {  
 arr.Swap(i, i + 1);  
 ready = true;  
 }  
 }  
 if (debug >= sort\_process)  
 arr.Print();  
 }  
 }  
 void Info() override  
 {   
 cout << "exchange sort \n\t- sorts by exchange until condition is met for all elements" << mcl::endl;  
 }  
private:  
};  
  
class insert : public sort  
{  
public:  
 insert() : sort(insert\_sort)  
 {   
 step = 1;  
 start = 0;  
 }  
 void Sort(arr\_for\_sort& arr) override  
 {  
 for (int i = start+step, j = start+step; i < arr.Size(); i += step, j = i - step)  
 {  
 for (; (j >= 0)&&(arr.Check(j, j + step)); j -= step)  
 {  
 arr.Swap(j, j + step);  
   
 if (debug >= sort\_process)  
 arr.Print();  
 }  
 if (debug >= sort\_process)  
 arr.Print();   
 }  
 }  
 void Info() override  
 {  
 cout << "insert sort \n\t- sorts by bringing each element to its place, at any stage the left part is already sorted" << mcl::endl;  
 }  
protected:  
 void SetStep(int new\_step)  
 {  
 step = new\_step;  
 }  
 void SetStart(int new\_start)  
 {  
 start = new\_start;  
 }  
private:  
 int step, start;  
};  
  
class shell : public sort, private insert  
{  
public:  
 shell() : sort(shell\_sort)  
 {  
  
 }  
 void Sort(arr\_for\_sort& arr) override  
 {  
 int step = 0;  
 for (int i = arr.Size(); i > 0; i--)  
 {  
 if(GetStep(log2(i), arr.Size()) == step)  
 { continue; }  
 step = GetStep(log2(i), arr.Size());  
 for (int j = 0; j < step; j++)  
 {  
 SetStep(step);  
 SetStart(j);  
 insert::Sort(arr);  
 }  
 if (debug >= sort\_process)  
 arr.Print();  
 }  
 }  
 void Info() override  
 {  
 cout << "shell sort \n\t- sorts using insertion sort at different stages with different steps" << mcl::endl;  
 }  
private:  
 int GetStep(int i, int n)  
 {  
 int res;  
 if (i % 2 == 0)  
 res = 9 \* pow(2, i) - 9 \* pow(2, i / 2) + 1;  
 else  
 res = 8 \* pow(2, i) - 6 \* pow(2, (i + 1) / 2) + 1;  
 if (res \* 3 > n)  
 return GetStep(i - 1, n);  
 return res;  
 }  
};  
  
  
class merger : public sort  
{  
public:  
 merger() : sort(merger\_sort)  
 {  
  
 }  
 void Sort(arr\_for\_sort& arr) override  
 {  
 int size;  
 arrs = Fragment(arr, size);  
 arrs = Sort(arrs, size);  
 Defragment(arrs, arr, size);  
 }  
 void Info() override  
 {  
 cout << "merger sort \n\t- splits into arrays and after merging the ones that satisfy the condition, goes into recursion" << mcl::endl;  
 }  
private:  
 array\*\* arrs;  
  
 array\*\* Merge(array\*\* arrays, int& size\_new\_arr, int size)  
 {  
 array\*\* new\_arrs;  
 size\_new\_arr = (size + 1) / 2;  
 new\_arrs = new array \* [size\_new\_arr];  
 if (debug >= sort\_process)  
 {  
 cout << "before merge " << size << "to" << size\_new\_arr << mcl::endl;  
 PrintFragments(arrays, size);  
 }  
  
 for (int i = 0; i < size\_new\_arr; i++)  
 {  
 int j\_a\_max = arrays[i \* 2]->size, j\_b\_max = (size > i \* 2 + 1) ?  
 arrays[i \* 2 + 1]->size : 0;  
 if (debug >= sort\_process)  
 cout << "create arr for " << j\_a\_max << "+" << j\_b\_max << " elements:" << mcl::endl;  
 int size\_ = j\_a\_max + j\_b\_max;  
 new\_arrs[i] = new array(size\_);  
 int j\_a = 0, j\_b = 0;  
  
 for (; j\_a + j\_b < size\_;)  
 {  
 if (j\_a == j\_a\_max) {  
 new\_arrs[i]->data[j\_a + j\_b] = arrays[i \* 2 + 1]->data[j\_b];  
 j\_b++;  
 }  
 else if (j\_b == j\_b\_max) {  
 new\_arrs[i]->data[j\_a + j\_b] = arrays[i \* 2]->data[j\_a];  
 j\_a++;  
 }  
 else if (arrays[i \* 2 + 1]->data[j\_b] >= arrays[i \* 2]->data[j\_a]) {  
 new\_arrs[i]->data[j\_a + j\_b] = arrays[i \* 2]->data[j\_a];  
 j\_a++;  
 }  
 else {  
 new\_arrs[i]->data[j\_a + j\_b] = arrays[i \* 2 + 1]->data[j\_b];  
 j\_b++;  
 }  
 }  
 if (debug >= sort\_process)  
 PrintFragment(new\_arrs[i]);  
  
 }  
 if (debug >= sort\_process)  
 {  
 cout << "after merge " << size << " to " << size\_new\_arr << mcl::endl;  
 PrintFragments(new\_arrs, size\_new\_arr);  
 }  
 return new\_arrs;  
 }  
 array\*\*& Sort(array\*\* arrays, int size)  
 {  
 int size\_new\_arr = size;  
 while (size\_new\_arr != 1)  
 {  
 array\*\* tmp = Merge(arrays, size\_new\_arr, size);  
 for (int i = 0; i < size; i++)  
 delete arrays[i];  
 delete[] arrays;  
 arrays = tmp;  
 size = size\_new\_arr;  
 if (debug >= sort\_process)  
 PrintFragments(arrays, size\_new\_arr);  
 }  
 return arrays;  
 }  
 array\*\* Fragment(arr\_for\_sort& arr, int& size)  
 {  
 size = arr.Size();  
 array\*\* arrays = new array \* [size];  
 for (int i = 0; i < size; i++)  
 {  
 arrays[i] = new array(1);  
 arrays[i]->Copy(arr.data + i, 1);  
 }  
 return arrays;  
 }  
 void Defragment(array\*\*& arrays, arr\_for\_sort& arr, int size)  
 {  
 for (int i = 0; i < size; i++)  
 {  
 arr[i] = arrays[0]->data[i];  
 }  
 }  
 void PrintFragment(array\*& fragment)  
 {  
 fragment->Print();  
 }  
 void PrintFragments(array\*\* fragments, int len)  
 {  
 for (int i = 0; i < len; i++)  
 {  
 cout << "[" << i << "] = " << mcl::nsep;  
 PrintFragment(fragments[i]);  
 }  
 }  
};  
  
class fast : public sort  
{  
public:  
 fast() : sort(fast\_sort)  
 {  
  
 }  
 void Sort(arr\_for\_sort& arr) override  
 {  
  
 Swap(arr, arr.Size() / 2, arr.Size(), 0);  
 }  
  
 void Info() override  
 {  
 cout << "fast sort \n\t- sorts relative to the pivot and starts recursion" << mcl::endl;  
 }  
private:  
 void Swap(arr\_for\_sort& arr, int middle, int size, int start)  
 {  
   
 middle = start + (size + 1) / 2 - size % 2;  
 if (debug >= sort\_process)  
 {  
 cout << "\_\_recursion\_\_" << mcl::endl;  
 PrintSegment(arr, middle, size, start);  
 }  
 int center = middle;  
 bool check\_1, check\_2;  
 for (int i = 1; i <= (size-1)/2; i++)  
 {  
  
 check\_1 = arr.Check(center - i, middle);  
 check\_2 = arr.Check(middle, center + i);  
 if (check\_1 && check\_2)  
 {  
 if (debug >= sort\_process)  
 cout << "Swap" << mcl::endl;  
  
 arr.Swap(center - i, center + i);  
 }  
 else if (check\_1 || check\_2)  
 {  
 if (check\_1)  
 {  
 if (debug >= sort\_process)  
 cout << "Insert" << mcl::endl;  
 arr.Insert(center - i, middle);  
 middle--;  
 }  
 if (check\_2)  
 {  
 if (debug >= sort\_process)  
 cout << "Insert" << mcl::endl;  
 arr.Insert(center + i, middle);  
 middle++;  
 }  
 }  
 if (debug >= sort\_process)  
 PrintSegment(arr, middle, size, start);  
  
 }  
 if (size % 2 == 0)  
 {  
 if (arr.Check(start, middle))  
 {  
 arr.Insert(start, middle);  
 middle--;  
 }  
 }  
 if (debug >= sort\_process)  
 PrintSegment(arr, middle, size, start);  
  
 int len\_1 = middle - start;  
 int len\_2 = (start + size - 1) - middle;  
 if (len\_1 > 1)  
 {  
 Swap(arr, len\_1 / 2 + start, len\_1, start);  
 }  
 if (len\_2 > 1)  
 {  
 Swap(arr, (len\_2) / 2 + middle+1, len\_2, middle+1);  
 }  
 }  
 void PrintSegment(arr\_for\_sort& arr, int middle, int size, int start)  
 {  
 cout << " middle: " << middle << " size: " << size << " start: " << start << mcl::endl;  
  
 for (int i = start; i < start + size; i++)  
 {  
 if (i == middle)  
 {  
 cout << " > " << arr[i] << mcl::nsep;  
 }  
 else  
 {  
 cout << " | " << arr[i] << mcl::nsep;  
 }  
 }   
 cout << " | " << mcl::endl;  
  
 }  
};  
  
  
class heap : public sort  
{  
public:  
 heap() : sort(heap\_sort)  
 {  
  
 }  
 void Sort(arr\_for\_sort& arr) override  
 {  
 for (max\_index = arr.Size(); max\_index > 0; max\_index--)  
 {  
 Check(arr, 0);  
 arr.Swap(0, max\_index - 1);  
 if (debug >= sort\_process)  
 arr.Print();  
 }  
 }  
 void Info() override  
 {   
 cout << "heap sort \n\t- sorts using a binary tree folded into an array" << mcl::endl;  
 }  
  
private:  
 void Check(arr\_for\_sort& arr, int index)  
 {  
 int branch\_l = index\*2 + 1, branch\_r = index\*2 + 2;  
 if (branch\_l < max\_index)  
 {  
 Check(arr, branch\_l);  
 if (arr.Check(branch\_l, index))  
 arr.Swap(branch\_l, index);  
 }  
 if (branch\_r < max\_index)  
 {  
 Check(arr, branch\_r);  
 if (arr.Check(branch\_r, index))  
 arr.Swap(branch\_r, index);  
 }  
 }  
 int max\_index;  
};  
  
  
class radix : public sort  
{  
public:  
 radix() : sort(radix\_sort)  
 {  
  
 }  
 void Sort(arr\_for\_sort& arr) override  
 {  
 for (int i = 0; i < MaxRadix(arr); i++)  
 {  
 Merge(Сrushing(arr, i), arr);  
 }  
 }  
 void Info() override  
 {  
 cout << "radix sort \n\t- sorts by rank" << mcl::endl;  
 }  
  
private:  
 int MaxRadix(arr\_for\_sort& arr)  
 {  
 int max = arr[arr.Max(0)];  
 int i = 0;  
 for (; max > 0; i++, max = max / 10);  
 return i;  
 }  
 void Merge(arr\_for\_sort\* arrs, arr\_for\_sort& arr)  
 {  
 int k = 0;  
 for (int i = 0; i < 10; i++)  
 {  
 for (int j = 0; j < arrs[i].Size(); j++)  
 {  
 arr[k] = arrs[i][j];  
 k++;  
 }  
 }  
 delete[] arrs;  
 }  
 arr\_for\_sort\* Сrushing(arr\_for\_sort& arr, int radix)  
 {  
 arr\_for\_sort\* arrs = new arr\_for\_sort[10];  
 for (int i = 0; i < arr.Size(); i++)  
 {  
 arrs[GetRadix(arr[i], radix)].Append(arr[i]);  
 }  
 return arrs;  
 }  
 int GetRadix(int num, int radix)  
 {  
 return ((int)(num / pow(10, radix)) % 10);  
 }  
};  
  
  
void SortCheck(arr\_for\_sort& some\_good\_arr, sort\* some\_sort)  
{  
 cout << "before: " << mcl::nsep;  
 some\_good\_arr.Print();  
 TIME\_IT  
 some\_sort->Sort(some\_good\_arr);  
 int time\_is = COMPARE\_TIME;  
 cout << "after: " << mcl::nsep;  
 some\_good\_arr.Print();  
 if (time\_is < 1)  
 cout << "time: <1ms." << mcl::endl;  
 else  
 cout << "time: " << time\_is << "ms." << mcl::endl;  
} **Code: \filework.cpp**  
  
#include"filework.h"  
  
FILE\* open(const char\* nameFile, const char\* arg)  
{  
 FILE\* f;  
 ;  
 if (fopen\_s(&f, nameFile, arg) == EINVAL) {  
 cout << "Error opening file. The file may be missing." \* error\_color << mcl::endl;  
 }  
 else {  
 cout << "file \"" \* success\_color << nameFile << "\" was open." << mcl::endl;  
 }  
 return f;  
}  
  
bool readline(FILE\* file, char\* buff, buff\_size size)  
{  
 cout << "Reading.. " \* process\_color << mcl::nsep;  
 if ((!feof(file)) && (fgets(buff, size, file) != NULL))  
 {  
 char tmp[4] = { buff[0], buff[1], buff[2], '\0'};  
 cout << "\"" \* process\_color << tmp << "..\"\n\r" << "Successful reading" \* success\_color << mcl::endl;  
 return true;  
 }  
 else {  
 if (feof(file))  
 {  
 cout << "End of file." \* error\_color << mcl::endl;  
 }  
 else {  
 cout << "An error occurred while reading from the file." \* error\_color << mcl::endl;  
 }  
 return false;  
 }  
}  
  
  
file\_data::file\_data(file\_size size, buff\_size lens) : size{ size }, lens{ lens }  
{  
 strings = new char\* [size];  
 for (int i = 0; i < size; i++)  
 {  
 strings[i] = new char[lens] {0};  
 }  
}  
file\_data::~file\_data()  
{  
 for (int i = 0; i < size; i++)  
 {  
 delete[] strings[i];  
 }  
 delete[] strings;  
}  
  
  
file\_data\* read(FILE\* file)  
{  
 file\_data\* file\_ = new file\_data(FILE\_LEN, BUFF\_LEN);  
 for (int i = 0; i < file\_->size; i++)  
 {  
 if (!readline(file, file\_->strings[i], file\_->lens))  
 {  
 break;  
 }  
 }  
 cout << "Successful reading all file." \* success\_color << mcl::endl;  
 return file\_;  
}  
  
bool write(FILE\* file, file\_data\* data)  
{  
 for (int i = 0; i < data->size; i++)  
 {  
 if (fputs(data->strings[i], file) == EOF) {  
 cout << "An error occurred while writing to the file." \* error\_color << mcl::endl;  
 return false;  
 }  
 }  
 cout << "Successful writing all file." \* success\_color << mcl::endl;  
 return true;  
}  
  
void close(FILE\* file)  
{  
 fclose(file);  
 cout << "file was closed." \* success\_color << mcl::endl;  
}  
 **Code: \MultiColorLine.cpp**  
#include "MultiColorLine.h"  
  
  
void mcl::setConsoleColor(HANDLE hConsole, int textColor, int backgroundColor)  
{  
 SetConsoleTextAttribute(hConsole, (backgroundColor << 4) | textColor);  
}  
  
mcl::cmd\_color::cmd\_color() : textColor(0), backgroundColor(0)  
{ }  
mcl::cmd\_color::cmd\_color(int textColor, int backgroundColor) : textColor(textColor), backgroundColor(backgroundColor)  
{ }  
  
  
mcl::letter::letter(double x, cmd\_color color) : string(std::to\_string(x)), color(color)  
{ }  
mcl::letter::letter(int x, cmd\_color color) : string(std::to\_string(x)), color(color)  
{ }  
mcl::letter::letter(std::string s, cmd\_color color) : string(s), color(color)  
{ }  
  
mcl::letter mcl::operator \* (std::string string, cmd\_color color) {  
 return mcl::letter(string, color);  
}  
mcl::letter mcl::operator \* (double number, cmd\_color color) {  
 return letter(number, color);  
}  
mcl::letter mcl::operator \* (int number, cmd\_color color) {  
 return letter(number, color);  
}  
mcl::letter mcl::operator \* (const char\* string, cmd\_color color) {  
 return mcl::letter(string, color);  
}  
enum tag  
{  
 tab,  
 endl,  
 nsep  
};  
  
mcl::print::print() : last\_color(7, 0)  
{  
 hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);  
}  
void mcl::print::add(letter str)  
{  
 string.push\_back(str);  
}  
void mcl::print::out(tag tag\_)  
{  
 for (auto it : string)  
 {  
#ifdef text\_color  
 setConsoleColor(hConsole, it.color.textColor, it.color.backgroundColor);  
#endif // text\_color  
 std::cout << it.string;  
 }  
#ifdef text\_color  
 setConsoleColor(hConsole, 7, 0);  
#endif // text\_color  
  
 switch (tag\_)  
 {  
 case tab:  
 std::cout << "\t";  
 break;  
 case endl:  
 std::cout << std::endl;  
 break;  
 case space:  
 std::cout << " ";  
 break;  
 case nsep:  
 break;  
 default:  
 break;  
 }  
 string.clear();  
}  
void mcl::print::SetPos(short x, short y)  
{  
 SetConsoleCursorPosition(hConsole, COORD{ x, y });  
}  
  
mcl::print& mcl::print::operator()(short x, short y)  
{  
 SetPos(x, y);  
 return \*this;  
}  
  
  
mcl::print& mcl::operator << (print& out, tag tag\_) {  
 out.out(tag\_);  
 return out;  
}  
  
mcl::print& mcl::operator << (print& out, letter str) {  
 out.add(str);  
 out.last\_color = str.color;  
 return out;  
}  
  
mcl::print& mcl::operator << (print& out, std::string str) {  
 out.add(letter(str, out.last\_color));  
 return out;  
}  
  
mcl::print& mcl::operator << (print& out, const char\* str) {  
 out.add(letter(str, out.last\_color));  
 return out;  
}  
  
mcl::print& mcl::operator << (print& out, double num) {  
 out.add(letter(std::to\_string(num), out.last\_color));  
 return out;  
}  
  
mcl::print& mcl::operator << (print& out, int num) {  
 out.add(letter(std::to\_string(num), out.last\_color));  
 return out;  
}  
  
mcl::print cout;  
mcl::cmd\_color error\_color(4, 0);  
mcl::cmd\_color success\_color(2, 0);  
mcl::cmd\_color process\_color(3, 0);  
mcl::cmd\_color data\_color(15, 0);  
mcl::cmd\_color select\_color(15, 8);  
 **Code: \filework.h**#pragma once  
  
#include <iostream>  
#include <cstring>  
#include <windows.h>  
#include <stdio.h>  
#include <stdlib.h>  
  
#include "MultiColorLine.h"  
#include "laba\_1.h"  
#include "laba\_2.h"  
  
  
#define READ\_FILE "r"  
#define WRITE\_FILE "w"  
#define READ\_BIN\_FILE "rb"  
#define WRITE\_BIN\_FILE "wb"  
  
#define BUFF\_LEN 64  
#define FILE\_LEN 16  
  
  
  
typedef unsigned int buff\_size;  
typedef unsigned int file\_size;  
  
//template struct column<int>;  
//template struct data<int>;  
//template struct data\_str<int>;  
  
FILE\* open(const char\* nameFile, const char\* arg);  
  
void close(FILE\* file);  
  
bool readline(FILE\* file, char\* buff, buff\_size size);  
  
struct file\_data {  
 char\*\* strings;  
 const file\_size size;  
 const buff\_size lens;  
 file\_data(file\_size size, buff\_size lens);  
 ~file\_data();  
};  
  
file\_data\* read(FILE\* file);  
  
bool write(FILE\* file, file\_data\* data);  
  
/\*  
template<typename data\_type>  
void save(const char\* filename, data\_type\* p);  
  
template<typename data\_type>  
data\_type load(const char\* filename);  
\*/  
  
template<typename data\_type>  
void save(const char\* filename, data\_type\* p)  
{  
 FILE\* fp;  
 char\* c;  
 int size = sizeof(data\_type);  
 fp = open(filename, WRITE\_BIN\_FILE);  
 c = (char\*)p;  
 for (int i = 0; i < size; i++)  
 {  
 putc(\*c++, fp);  
 }  
 close(fp);  
}  
  
template<typename data\_type>  
data\_type load(const char\* filename)  
{  
 FILE\* fp;  
 char\* c;  
 int i;  
 int size = sizeof(data\_type);  
 data\_type\* ptr = (data\_type\*)malloc(size);  
  
 fp = open(filename, READ\_BIN\_FILE);  
  
  
 c = (char\*)ptr;  
 while ((i = getc(fp)) != EOF)  
 {  
 \*c = i;  
 c++;  
 }  
  
 close(fp);  
 data\_type tmp = \*ptr;  
 free(ptr);  
 return tmp;  
} **Code: \MultiColorLine.h**#pragma once  
  
#include <iostream>  
#include <string>  
#include <sstream>  
#include <vector>  
  
#include <windows.h>  
#include <cstdlib>  
  
#ifndef text\_color  
#define text\_color  
  
  
namespace mcl {  
 void setConsoleColor(HANDLE hConsole, int textColor, int backgroundColor);  
  
 struct cmd\_color  
 {  
 int textColor;  
 int backgroundColor;  
 cmd\_color();  
 cmd\_color(int textColor, int backgroundColor);  
 };  
  
 struct letter {  
 std::string string;  
 cmd\_color color;  
 letter(double x, cmd\_color color);  
 letter(int x, cmd\_color color);  
 letter(std::string s, cmd\_color color);  
 };  
  
 letter operator \* (std::string string, cmd\_color color);  
 letter operator \* (double number, cmd\_color color);  
 letter operator \* (int number, cmd\_color color);  
 letter operator \* (const char\* string, cmd\_color color);  
 enum tag  
 {  
 tab,  
 endl,  
 space,  
 nsep  
 };  
  
 class print  
 {  
 HANDLE hConsole;  
 std::vector<letter> string;  
 public:  
 cmd\_color last\_color;  
  
 print();  
 void add(letter str);  
 void out(tag tag\_);  
 void SetPos(short x, short y);  
  
 print& operator()(short x, short y);  
 };  
  
 print& operator << (print& out, tag tag\_);  
 print& operator << (print& out, letter str);  
 print& operator << (print& out, std::string str);  
 print& operator << (print& out, const char\* str);  
 print& operator << (print& out, double num);  
 print& operator << (print& out, int num);  
} ;  
  
extern mcl::print cout;  
extern mcl::cmd\_color error\_color;  
extern mcl::cmd\_color success\_color;  
extern mcl::cmd\_color process\_color;  
extern mcl::cmd\_color data\_color;  
extern mcl::cmd\_color select\_color;  
  
/\*  
0 = Черный  
1 = Синий  
2 = Зеленый  
3 = Голубой  
4 = Красный  
5 = Лиловый  
6 = Желтый  
7 = Белый  
8 = Серый  
9 = Светло-синий  
A/10 = Светло-зеленый  
B/11 = Светло-голубой  
C/12 = Светло-красный  
D/13 = Светло-лиловый  
E/14 = Светло-желтый  
F/15 = Ярко-белый  
\*/  
  
#endif