**Шараев Владислав Николаевич 020602**

**Лабораторная работа №15**

**Вариант 14**

***Условие:*** Поменять местами первый и максимальный элементы списка.

***Текст программы:***

#define \_CRT\_SECURE\_NO\_WARNINGS // Предпроцессор для отключения предупреждений при работе со scanf и printf

#include <iostream>

#include <cstdio>

#include <ctime>

using namespace std;

struct List { // Структура элемента очереди

List\* left = 0; // Указатель на левый элемент

List\* right = 0; // Указатель на правый элемент

int value; // Значение элемента

};

void NewList(List\*\*, List\*\*); // Создание нового списка

void PushAfterList(List\*, int); // Добавление элемента справа от указанного

void PushBeforeList(List\*, int); // Добавление элемента слева от указанного

void PrintList(List\*, List\*); // Вывод содержимого списка на экран

void FindItemList(List\*, List\*, int); // Поиск элемента списка

void DivideIntoTwoLists(List\*, List\*, List\*\*, List\*\*, List\*\*, List\*\*); // Разделение на два списка

void UniteLists(List\*\*, List\*\*, List\*, List\*, List\*, List\*); // Объединение двух списков

void SortList(List\*\*, List\*\*); // Сортировка списка

int DeleteItem(List\*); // Удаление элемента из списка

void DeleteList(List\*\*, List\*\*); // Удаление списка

void SwapElementsList(List\*, List\*); // Поменять местами первый и максимальный элементы списка

int main()

{

setlocale(LC\_ALL, "Russian"); // Поддержка кириллицы

srand(time(0)); // Инициализация псевдослучайных чисел

int n;

int toFind = 14; // Число, которое необходимо найти

List\* QLeft, \*QRight; // Указатели на левую и правую границы списка

NewList(&QLeft, &QRight); // Создание нового списка

printf("Введите количество элементов: "); // Чтение количества элементов

scanf("%d", &n);

for (int i = 0; i < n; i++) // Заполнение списка элементами

{

int temp = rand() % 101 - 50; // Генерация чисел в диапазоне [-50;50]

PushBeforeList(QRight, temp); // Добавление в список

}

printf("\nСодержимое списка: \n"); // Вывод содержимого списка до сортировки

PrintList(QLeft, QRight);

printf("\n");

FindItemList(QLeft, QRight, toFind); // Вывод результата поиска числа в списке

SwapElementsList(QLeft, QRight); // Меняем местами первый и максимальный элементы списка

printf("\nСодержимое списка после смены местами: \n");

PrintList(QLeft, QRight); // Вывод содержимого списка после изменения мест

SortList(&QLeft, &QRight); // Сортировка списка

printf("\nСодержимое списка после сортировки: \n");

PrintList(QLeft, QRight); // Вывод содержимого списка после сортировки

printf("\n");

DeleteList(&QLeft, &QRight); // Удаление списка

return 0;

}

void NewList(List\*\* left, List\*\* right) // Создание нового списка

{

\*left = new List; // Создание новых элементов списка

\*right = new List;

(\*left)->left = 0; // Связываем указатели на начало и конец списка

(\*left)->right = \*right;

(\*right)->left = \*left;

(\*right)->right = 0;

}

void PushAfterList(List\* prev, int value) // Добавление элемента справа от указанного

{

List\* item = new List; // Создание нового элемента

item->value = value; // Присвоение значения новому элементу

item->left = prev; // Связывание элемента со списком

item->right = prev->right;

prev->right = item;

item->right->left = item;

}

void PushBeforeList(List\* next, int value) // Добавление элемента слева от указанного

{

List\* item = new List; // Создание нового элемента

item->value = value; // Присвоение значения новому элементу

item->right = next; // Связывание элемента со списком

item->left = next->left;

next->left = item;

item->left->right = item;

}

void PrintList(List\* begin, List\* end) // Вывод содержимого списка на экран

{

List\* item = begin->right; // Итератор для прохода по списку

while (item != end) // Пока не достигнем конца списка

{

printf("%d ", item->value);

item = item->right; // Переходим на следующий элемент (справа)

}

printf("\n");

}

void FindItemList(List\* begin, List\* end, int value) // Поиск элемента списка

{

List\* item = begin->right; // Итератор для прохода по списку

for (int i = 0; item != end; item = item->right, i++) // i - номер позиции элемента, если он есть; с каждым шагом берем следующий элемент (правый)

{

if (item->value == value)

{

printf("Значение %d найдено на позиции номер %d\n", value, i);

return;

}

}

printf("Значение %d не найдено в списке\n", value);

}

void DivideIntoTwoLists(List\* begin, List\* end, List\*\* leftL, List\*\* rightL, List\*\* leftR, List\*\* rightR) // Разделение списка на два

{

NewList(leftL, rightL); // Создаем два новых списка

NewList(leftR, rightR);

List\* item = begin->right; // Итератор для прохода по исходному списку

while (item != end) // Пока не достигнем конца

{

PushBeforeList(\*rightL, DeleteItem(item));

item = begin->right;

if (item != end)

{

PushBeforeList(\*rightR, DeleteItem(item));

item = begin->right;

}

}

delete begin;

delete end;

}

void UniteLists(List\*\* begin, List\*\* end, List\* leftL, List\* rightL, List\* leftR, List\* rightR) // Объединение двух списков

{

NewList(begin, end); // Новый список, полученные из предыдущих двух

List\* itemL = leftL->right; // Итераторы для двух списков

List\* itemR = leftR->right;

while ((itemL != rightL) && (itemR != rightR)) // Пока не достигнем конца обоих списков

{

if (itemL->value < itemR->value) // Выбираем из какого списка взять следующий элемент

{

PushBeforeList(\*end, DeleteItem(itemL));

itemL = leftL->right;

}

else

{

PushBeforeList(\*end, DeleteItem(itemR));

itemR = leftR->right;

}

}

while (itemL != rightL) // Если один список был меньше другого, то все оставшиеся элементы добавятся в конец

{

PushBeforeList(\*end, DeleteItem(itemL));

itemL = leftL->right;

}

delete leftL;

delete rightL;

while (itemR != rightR) // Аналогично предыдущему while

{

PushBeforeList(\*end, DeleteItem(itemR));

itemR = leftR->right;

}

delete leftR;

delete rightR;

}

void SortList(List\*\* begin, List\*\* end) // Сортировка списка

{

List \*leftL, \*rightL, \*leftR, \*rightR; // Указатели для разделения на два списка

if ((\*begin)->right->right == \*end) return; // Если список состоит из 1 элемента, то ничего не делаем

DivideIntoTwoLists(\*begin, \*end, &leftL, &rightL, &leftR, &rightR); // Разделение на два списка

SortList(&leftL, &rightL); // Рекурсивно идем по двум новым спискам

SortList(&leftR, &rightR);

UniteLists(begin, end, leftL, rightL, leftR, rightR); // Объединяем 2 списка

}

int DeleteItem(List\* item) // Удаляем элемент списка, возвращаем его значение

{

int val = item->value; // Запоминаем значение

item->left->right = item->right; // Связываем левый и правый элементы

item->right->left = item->left;

delete item; // Очищаем память

return val; // Возвращаем значение

}

void DeleteList(List\*\* begin, List\*\* end) // Удаление списка

{

List\* item = (\*begin)->right; // Итератор для прохода по списку

while (item != \*end) // Пока не достигнем конца списка

{

DeleteItem(item); // Удаление элемента из списка

item = (\*begin)->right; // Переходим к следующему элементу (справа)

}

delete\* begin; // Очищаем указатели на границы списка

delete\* end;

\*begin = \*end = 0;

printf("Список удален\n");

}

void SwapElementsList(List\* begin, List\* end) // Меняем местами первый и максимальный элементы списка

{

List\* item = begin->right->right; // Итератор для прохода по списку

List\* max = begin->right; // Указатель на элемент с максимальным значением

while (item != end) // Пока не достигнем конца списка

{

if (item->value > max->value) // Если нашли значение большее, чем в max, то запоминаем его

{

max = item;

}

item = item->right; // Переходим к следующему элементу (справа)

}

swap(max->value, begin->right->value); // Меняем местами значения

}

***Результат работы программы:***



