#include<iostream>

using namespace std;

enum CMD {

CMD\_erase,

CMD\_insert,

CMD\_length,

CMD\_empty\_check,

CMD\_delete\_first,

CMD\_add\_first,

CMD\_add\_back,

CMD\_delete\_back,

CMD\_clear,

CMD\_show,

CMD\_show\_list,

CMD\_replace,

CMD\_reverse,

CMD\_insert\_list\_by\_index,

CMD\_insert\_list\_back,

CMD\_insert\_list\_beginnig,

CMD\_check\_list\_in,

CMD\_find\_first\_occur,

CMD\_find\_last\_occur,

CMD\_swap,

CMD\_exit,

CMD\_none

};

class SingleLinkedList //Односвязный список

{

private:

class Node // Узел

{

public:

int data;

Node\* next;

public:

Node(int data)

{

this->data = data;

this->next = nullptr;

}

};

public:

Node\* head, \* tail;

public:

SingleLinkedList()

{

this->head = this->tail = nullptr;

}

~SingleLinkedList()

{

ClearList();

}

void ClearList()

{

while (head != nullptr)

{

DeleteFirst();

}

}

bool EmptyCheck() // Проверка на пустоту списка

{

return (head == nullptr);

}

void DeleteFirst() // Удалние первого

{

if (EmptyCheck()) return;

if (head == tail) // Проверка, что в списке всего один элемент

{

delete tail;

head = tail = nullptr;

return;

}

Node\* node = head;

head = node->next; //перемещаем указатель head на следующий и освобождаем память первого элемента

delete node;

}

void DeleteBack()

{

if (EmptyCheck()) return;

if (head == tail) // Если в списке всего 1 элемент

{

delete tail;

head = tail = nullptr;

return;

}

Node\* node = head;

for (; node->next != tail; node = node->next); // перемещаемся к предпоследнему элементу

node->next = nullptr;

delete tail;

tail = node;

}

void AddBack(int data)

{

Node\* node = new Node(data); // Cоздание элемента

if (head == nullptr) head = node; // Если добавляеся первый элемент, то head ссылается на этот элемент

if (tail != nullptr) tail->next = node; //

tail = node; // перемещение tail в конец списка

}

void AddFirst(int data)

{

Node\* node = new Node(data);

node->next = head;

head = node;

if (tail == nullptr) tail = node; // Если добавляется первый элемент, то выпонляем tail = node

}

Node\* GetByIndex(int index) // функция класса, возвращающая указатель на элемент

{

if (index < 0) return nullptr;

Node\* node = head;

unsigned cntr = 0;

while (node && cntr != index && node->next) // существует ли объект node, неравенство индекса и счетчика и указатель next текущего объекта был не равен nullptr

{

node = node->next;

cntr++;

}

return (cntr == index) ? node : nullptr;

}

void Insert(int index, int data)

{

Node\* left = GetByIndex(index);

if (left == nullptr) return;

Node\* right = left->next;

Node\* node = new Node(data);

left->next = node;

node->next = right;

if (right == nullptr) tail = node;

}

void Erase(int index)

{

if (EmptyCheck()) return;

if (index < 0) return;

if (index == 1)

{

DeleteFirst();

return;

}

Node\* left = GetByIndex(index - 2);

Node\* node = left->next;

if (node == nullptr) return;

Node\* right = node->next;

left->next = right;

if (node == tail) tail = left;

delete node;

}

void ShowByIndex(int index)

{

if (EmptyCheck()) return;

Node\* node = GetByIndex(index-1);

int d = (node != nullptr) ? node->data : 0;

std::cout << d << std::endl;

}

unsigned LengthOfList()

{

unsigned cntr = 0;

for (Node\* node = head; node != nullptr; node = node->next)

{

cntr++;

}

return cntr;

}

void ShowList()

{

if (EmptyCheck()) return;

for (Node\* node = head; node != nullptr; node = node->next)

{

cout << node->data << " ";

}

}

void Replace(int index, int data)

{

if (EmptyCheck()) return;

Node \*node = GetByIndex(index-1);

node->data = data;

}

void Reverse()

{

if (EmptyCheck()) return;

int cntr = LengthOfList() - 1;

int temp;

Node\* node = head;

int saved\_cntr = cntr;

for (cntr; cntr >= 0; cntr--)

{

node = GetByIndex(cntr);

temp = (node != nullptr) ? node->data : 0;

AddBack(temp);

}

for (int i = 0; i <= saved\_cntr; i++)

{

DeleteFirst();

}

}

void InsertListBack()

{

int num\_of\_nodes;

int data;

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

cout << "Введите элементы\n";

for (int i = 0; i < num\_of\_nodes; i++)

{

cin >> data;

AddBack(data);

}

}

void InsertListBeginnig()

{

int num\_of\_nodes;

int data;

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

cout << "Введите элементы\n";

cin >> data;

AddFirst(data);

for (int i = 0; i < num\_of\_nodes - 1; i++)

{

cin >> data;

Insert(i, data);

}

}

void Swap()

{

if (EmptyCheck()) return;

int first;

int second;

int temp\_data;

Node\* first\_node;

Node\* second\_node;

Node\* temp;

cout << "Введите индекс первого элемента \n";

cin >> first;

cout << "Введите индекс второго элемента \n";

cin >> second;

first\_node = GetByIndex(first-1);

second\_node = GetByIndex(second-1);

temp = GetByIndex(first - 1);

temp\_data = temp->data;

Replace(first, second\_node->data);

Replace(second, temp\_data);

}

void InsertListByIndex()

{

int index;

int num\_of\_nodes;

int data;

cout << "Введите индекс \n";

cin >> index;

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

cout << "Введите элементы\n";

for (int i = 0, j = index - 1; i < num\_of\_nodes; i++, j++)

{

cin >> data;

Insert(j, data);

}

}

};

bool СheckListInList(SingleLinkedList& main\_list)

{

if (main\_list.EmptyCheck()) return false;

bool flag = false;

int occurence = 0;

int num\_of\_list\_in\_list;

int in\_index = 0;

int num\_of\_nodes;

int data;

cout << "Введите список\n";

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

SingleLinkedList list;

cout << "Введите элементы\n";

for (int i = 0; i < num\_of\_nodes; i++)

{

cin >> data;

list.AddBack(data);

}

num\_of\_list\_in\_list = main\_list.LengthOfList() / num\_of\_nodes;

for (int i = 0; i < num\_of\_list\_in\_list; i++)

{

for (in\_index; in\_index < num\_of\_nodes; in\_index++)

{

if (list.GetByIndex(in\_index)->data == main\_list.GetByIndex(in\_index)->data)

{

occurence++;

}

}

if (occurence == num\_of\_nodes)

{

flag = true;

return flag;

}

in\_index += num\_of\_list\_in\_list;

occurence = 0;

}

return flag;

}

void FindFirstOccur(SingleLinkedList& main\_list)

{

bool flag = true;

int num\_of\_list\_in\_list;

int in\_index = 0;

int num\_of\_nodes;

int data;

cout << "Введите список\n";

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

SingleLinkedList list;

cout << "Введите элементы\n";

for (int i = 0; i < num\_of\_nodes; i++)

{

cin >> data;

list.AddBack(data);

}

num\_of\_list\_in\_list = main\_list.LengthOfList() / num\_of\_nodes;

for (int i = 0; (i < num\_of\_list\_in\_list) && flag; i++)

{

for (in\_index; (in\_index < num\_of\_nodes)&&flag; in\_index++)

{

if (list.GetByIndex(in\_index)->data == main\_list.GetByIndex(in\_index)->data)

{

cout << "Элемент\n";

cout << list.GetByIndex(in\_index)->data;

cout << "\n Индекс\n";

cout << in\_index;

flag = false;

}

}

}

}

void FindLastOccur(SingleLinkedList& main\_list)

{

int num\_of\_list\_in\_list;

int in\_index = 0;

int num\_of\_nodes;

int data = 0;

cout << "Введите список\n";

cout << "Введите количество элементов\n";

cin >> num\_of\_nodes;

SingleLinkedList list;

cout << "Введите элементы\n";

for (int i = 0; i < num\_of\_nodes; i++)

{

cin >> data;

list.AddBack(data);

}

num\_of\_list\_in\_list = main\_list.LengthOfList() / num\_of\_nodes;

for (int i = 0; i < num\_of\_list\_in\_list; i++)

{

for (in\_index; in\_index < num\_of\_nodes; in\_index++)

{

if (list.GetByIndex(in\_index)->data == main\_list.GetByIndex(in\_index)->data)

{

data = list.GetByIndex(in\_index)->data;

}

}

in\_index += num\_of\_list\_in\_list;

}

cout << "Элемент\n";

cout << data;

cout << "\n Индекс\n";

cout << in\_index;

}

void ShowMenu()

{

cout << "МЕНЮ ДЕЙСТВИЙ\n\n";

cout << "1. Добавить элемент в конец списка\n";

cout << "2. Добавить элемент в начало списка\n";

cout << "3. Удалить последний элемент\n";

cout << "4. Удалить первый элемент\n";

cout << "5. Добавить элемент по индексу\n";

cout << "6. Получить элемент по индексу\n";

cout << "7. Удалить элемент по индексу\n";

cout << "8. Получить размер списка\n";

cout << "9. Удалить все элементы списка\n";

cout << "10. Заменить элемент\n";

cout << "11. Проверить на пустоту списка\n";

cout << "12. Замена порядока элементов в списке на обратный\n";

cout << "13. Вставить другой список в список, начиная с индекса\n";

cout << "14. Вставить другой список в конец\n";

cout << "15. Вставить другой список в начало\n";

cout << "16. Проверить на содержание другого списка в списке\n";

cout << "17. Найти первое вхождение другого списка в список\n";

cout << "18. Найти последнее вхождение другого списка в список\n";

cout << "19. Обменять два элемента списка по индексам\n";

cout << "120. Показать список\n";

cout << "0. Выход\n";

}

CMD user\_cmd\_input()

{

CMD incoming\_cmd = CMD\_none;

unsigned user\_input;

cin >> user\_input;

switch (user\_input)

{

case 0:

incoming\_cmd = CMD\_exit;

break;

case 1:

incoming\_cmd = CMD\_add\_back;

break;

case 2:

incoming\_cmd = CMD\_add\_first;

break;

case 3:

incoming\_cmd = CMD\_delete\_back;

break;

case 4:

incoming\_cmd = CMD\_delete\_first;

break;

case 5:

incoming\_cmd = CMD\_insert;

break;

case 6:

incoming\_cmd = CMD\_show;

break;

case 7:

incoming\_cmd = CMD\_erase;

break;

case 8:

incoming\_cmd = CMD\_length;

break;

case 9:

incoming\_cmd = CMD\_clear;

break;

case 10:

incoming\_cmd = CMD\_replace;

break;

case 11:

incoming\_cmd = CMD\_empty\_check;

break;

case 12:

incoming\_cmd = CMD\_reverse;

break;

case 13:

incoming\_cmd = CMD\_insert\_list\_by\_index;

break;

case 14:

incoming\_cmd = CMD\_insert\_list\_back;

break;

case 15:

incoming\_cmd = CMD\_insert\_list\_beginnig;

break;

case 16:

incoming\_cmd = CMD\_check\_list\_in;

break;

case 17:

incoming\_cmd = CMD\_find\_first\_occur;

break;

case 18:

incoming\_cmd = CMD\_find\_last\_occur;

break;

case 19:

incoming\_cmd = CMD\_swap;

break;

case 20:

incoming\_cmd = CMD\_show\_list;

break;

default:

cout << "Invalid input!\n";

incoming\_cmd = CMD\_none;

}

return incoming\_cmd;

}

int main()

{

setlocale(LC\_ALL, "Russian");

CMD user\_CMD = CMD\_none;

SingleLinkedList list;

int index;

int data;

bool flag = true;

while (flag)

{

system("cls");

ShowMenu();

user\_CMD = user\_cmd\_input();

switch (user\_CMD)

{

case CMD\_erase:

system("cls");

cout << "Введите номер элемента\n";

cin >> index;

list.Erase(index);

break;

case CMD\_insert:

system("cls");

cout << "Введите номер элемента\n";

cin >> index;

cout << "Введите значение\n";

cin >> data;

list.Insert(index-1, data);

break;

case CMD\_length:

system("cls");

cout << list.LengthOfList() << "\n";

cout << "\n\n";

system("pause");

break;

case CMD\_empty\_check:

system("cls");

(list.EmptyCheck()) ? cout << " Список пустой\n" : cout << "Список не пустой\n";

cout << "\n\n";

system("pause");

break;

case CMD\_delete\_first:

list.DeleteFirst();

break;

case CMD\_add\_first:

system("cls");

cout << "Введите значение\n";

cin >> data;

list.AddFirst(data);

break;

case CMD\_add\_back:

system("cls");

cout << "Введите значение\n";

cin >> data;

list.AddBack(data);

break;

case CMD\_delete\_back:

list.DeleteBack();

break;

case CMD\_clear:

list.ClearList();

break;

case CMD\_show:

system("cls");

cout << "Введите номер элемента\n";

cin >> index;

list.ShowByIndex(index);

cout << "\n\n";

system("pause");

break;

case CMD\_show\_list:

system("cls");

list.ShowList();

cout << "\n\n";

system("pause");

break;

case CMD\_replace:

system("cls");

cout << "Введите номер элемента\n";

cin >> index;

cout << "Введите значение\n";

cin >> data;

list.Replace(index, data);

break;

case CMD\_reverse:

list.Reverse();

break;

case CMD\_insert\_list\_by\_index:

system("cls");

list.InsertListByIndex();

break;

case CMD\_insert\_list\_back:

system("cls");

list.InsertListBack();

break;

case CMD\_insert\_list\_beginnig:

system("cls");

list.InsertListBeginnig();

break;

case CMD\_check\_list\_in:

system("cls");

if (СheckListInList(list))

{

cout << "Список есть в списке";

}

else

{

cout << "Список не входит в список";

}

cout << "\n\n";

system("pause");

break;

case CMD\_find\_first\_occur:

system("cls");

FindFirstOccur(list);

cout << "\n\n";

system("pause");

break;

case CMD\_find\_last\_occur:

system("cls");

FindLastOccur(list);

cout << "\n\n";

system("pause");

break;

case CMD\_swap:

system("cls");

list.Swap();

break;

case CMD\_exit:

flag = false;

break;

case CMD\_none:

break;

}

}

}