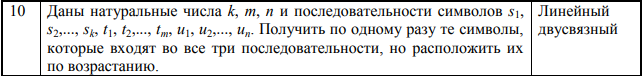
1. Цель работы

Целью работы является изучение структур, данных «линейный список» и «циклический список», а также получение практических навыков их реализации.

2. Вариант задания



Для выполнения задание потребуется организовать три списка:

* Линейный двусвязный список s из k элементов и содержащийx исходные символы;
* Линейный двусвязный список t из m элементов и содержащийx исходные символы;
* Линейный двусвязный список u из n элементов и содержащийx исходные символы;

Совершить обход трех списков и одинаковые символы добавить в результирующий массив.

Отсортировать результирующий массив.

3. Листинг программы, реализующей поставленную задачу с использованием заданных структур данных

Теперь можно разработать программу, которая должна выполнять нашу задачу.

Noda.h

#pragma once

#ifndef NODE\_H

#define NODE\_H

class Node {

public:

Node(char ch);

char ch;

Node\* next;

Node\* prev;

};

#endif

Имплементирую в cpp файле.

Noda.cpp

#include "Node.h"

Node::Node(char numb) {

this->ch = numb;

this->next = nullptr;

this->prev = nullptr;

}

В Node у нас храниться ссылка на следующий и предыдущий элемент а так же сам char символ.

LinkingList.h

#pragma once

#ifndef LINKINGLIST\_H

#define LINKINGLIST\_H

#include "Node.h"

class LinkingList {

public:

LinkingList();

void add(char ch);

void remove(int index);

void display();

Node\* get(int index);

Node\* getLast();

bool containsChar(char ch);

~LinkingList();

private:

Node\* head;

Node\* end;

};

#endif

Имплементирую в cpp файле.

#include <iostream>

#include "LinkingList.h"

LinkingList::LinkingList() {

head = nullptr;

}

void LinkingList::add(char ch) {

Node\* node = new Node(ch);

if (head == nullptr) {

head = node;

end = node;

}

else {

node->prev = end;

end->next = node;

end = node;

}

}

Node\* LinkingList::get(int index) {

Node\* current = head;

int currentIndex = 0;

while (current != nullptr && currentIndex < index) {

current = current->next;

currentIndex++;

}

return current;

}

Node\* LinkingList::getLast() {

return end;

}

void LinkingList::remove(int index) {

Node\* nodeToRemove = get(index);

if (nodeToRemove != nullptr) {

if (nodeToRemove->prev != nullptr) {

nodeToRemove->prev->next = nodeToRemove->next;

}

else {

head = nodeToRemove->next;

}

if (nodeToRemove->next != nullptr) {

nodeToRemove->next->prev = nodeToRemove->prev;

}

if (nodeToRemove == end) {

end = nodeToRemove->prev;

}

delete nodeToRemove;

}

}

bool LinkingList::containsChar(char ch) {

Node\* current = head;

while (current != nullptr) {

if (current->ch == ch) {

return true;

}

else {

current = current->next;

}

}

return false;

}

LinkingList::~LinkingList(){

Node\* current = head;

while (current != nullptr) {

Node\* next = current->next;

delete current;

current = next;

}

}

void LinkingList::display() {

Node\* current = head;

while (current != nullptr) {

std::cout << current->ch << " ";

current = current->next;

}

std::cout << std::endl;

}

В нашей имплементации есть методы:

add() – добавления

remove(int) – удаление по индексу

get(int) – получение по индексу

containsChar(char) – проверка на символ

display() – вывести на экран коллекцию

getLast() – получить последний элемент

1) Создание трех списков и их заполнение:

LinkingList\* listK = new LinkingList();

LinkingList\* listM = new LinkingList();

LinkingList\* listN = new LinkingList();

int K;

int M;

int N;

int min;

std::cout << "Введите К: ";

std::cin >> K;

std::cout << "Введите M: ";

std::cin >> M;

std::cout << "Введите N: ";

std::cin >> N;

if (K < M) {

if (K < N) {

min = K;

}

else

{

min = N;

}

}

else {

if (M < N) {

min = M;

}

else

{

min = N;

}

}

if(min==0){

std::cout << "Результата нет";

delete listK;

delete listM;

delete listN;

return;

}

char inputChar;

std::cout << "Введите сивлолы s :";

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

{

std::cin >> inputChar;

listK->add(inputChar);

}

std::cout << "Введите сивлолы t :";

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

{

std::cin >> inputChar;

listM->add(inputChar);

}

std::cout << "Введите сивлолы u :";

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

{

std::cin >> inputChar;

listN->add(inputChar);

}

Создание результирующего массива, поиск вхождений символов, сортировка результирующего массива;

char\* result = new char[min];

Node\* node = listK->get(0);

int count = 0;

while (node != nullptr) {

if (listM->containsChar(node->ch) && listN->containsChar(node->ch)) {

result[count] = node->ch;

count++;

}

node = node->next;

}

if (count == 0) {

std::cout << "Результата нет";

delete listK;

delete listM;

delete listN;

delete[] result;

return;

}

std::sort(result, result + count);

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

{

std::cout << result[i] << " ";

}

delete listK;

delete listM;

delete listN;

delete[] result;

}

Для выполнения поставленной задачи поиску одинаковых символов во всех трех последовательностей. При этом необходимо расположить их по возрастанию. Я воспользовался обычным линейным поиском взывая метод containsChar() я прохожусь по одному из списков в поисках совпадений в других трех примерная сложность O(n3), что значит при больший n это займет порядочное время. Алгоритм сортировки был взят из стандартной библиотеки, как я узнал из источников из интернета реализация является интросортировкой (IntroSort), скорость такого алгоритма O(n log n). К сожалению алгоритмы линейного поиска в этом случае показывает не самый хороший результат, но для данной задачи, где n вводиться с клавиатуры, он подходит благодаря своей простоте.

4. Контрольные примеры:

Тест проверяет 3 вхождения из 3-х :

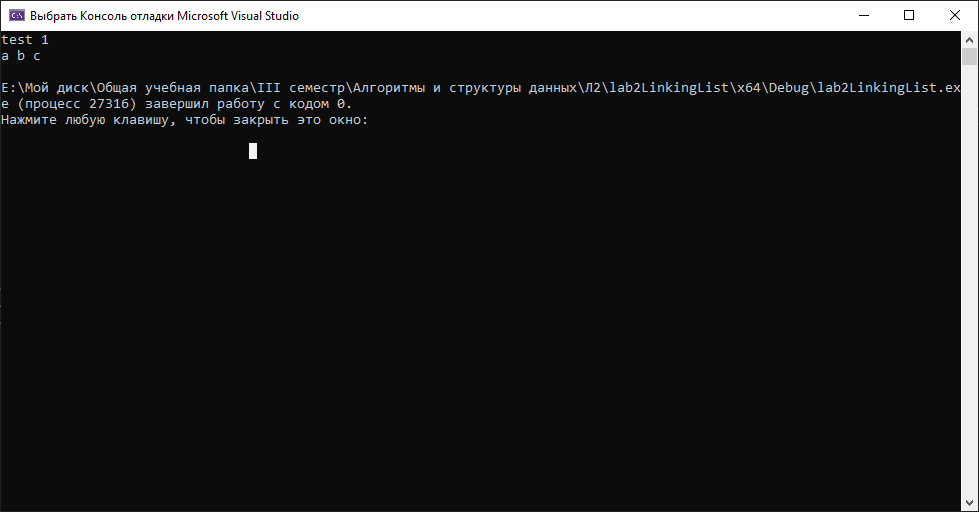
listK = {{‘c’},{‘b’},{‘a’}}

listM = {{‘b’},{‘c’},{‘a’}}

listN = {{‘c’},{‘a’},{‘b’}}

Результат:

a b c



Как мы видим результат такой какой мы и ожидали.

Тест проверка 1 вхождение из 3-х

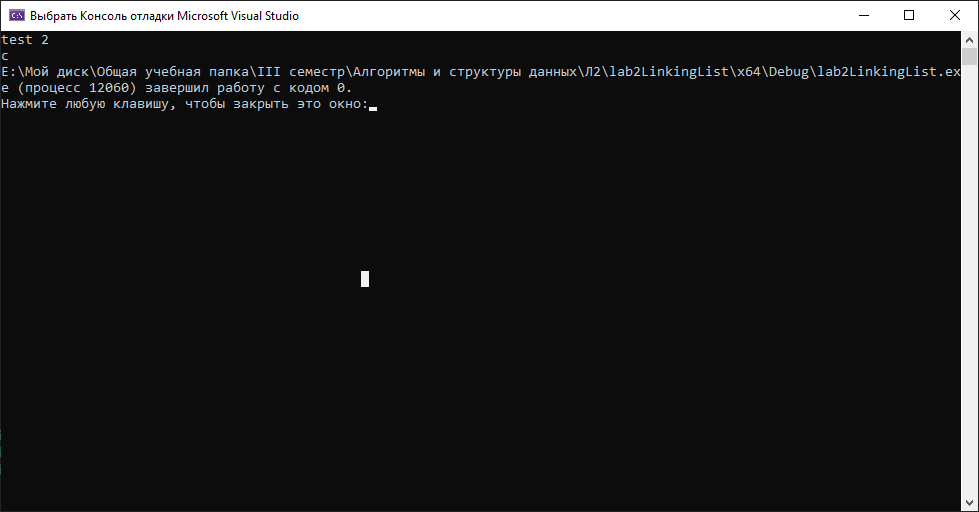
listK = {{‘c’},{‘b’},{‘a’}}

listM = {{‘x’},{‘c’},{‘e’}}

listN = {{‘c’},{‘x’},{‘b’}}

Результат:

c



Как мы видим результат такой какой мы и ожидали.

5. Вывод по работе

В ходе выполнение знание я изучил линейные и циклические списки которые позволяет эффективно хранить и обрабатывать данные, особенно в случаях, когда требуется частое добавление и удаление элементов. Также я научились работать с указателями и осознал принципы работы этой структурой данных. В результате выполнения работы я, улучшили свои навыки программирования и разработки алгоритмов.