**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**КАФЕДРА САПР**

отчет

**по лабораторной работе №2**

**по дисциплине «Программирование»**

Тема: **«Представление строки символов с маркером, ее обработка»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0336 |  | Насекин С.А. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2022

Основное задание: в однонаправленном линейном списке удалить n элементов, начиная с элемента с номером k; удалить список.

Условия:

1) Реализация на языке С++.

2) Использовать только примитивные типы данных.

Способ решения:

1) Создать однонаправленный список.

2) Прочитать файл input.txt с занесением содержимого в список.

3) Разработать алгоритм удаления n элементов, начиная с элемента с номером k.

4) Сохранить результаты в файл.

5) Удалить список.

Для создания однонаправленного списка нужно иметь две структуры или класса.

Структура Node:

struct Node {

char \*str; // строка

char marker = '@'; //маркер

int length = 0; // длина

Node \*next;

~Node() {

str = nullptr;

marker = 0;

}

};

Класс LinkedList:

class LinkedList {

Node \*last;

Node \*head;

const char\* empt = "";

public:

int countElem;

\\ методы, конструктор, деструктор.

}

Добавление элемента в список осуществляется методом addString:

void addString(char source[1000]) {

countElem++;

Node \*node = new Node;

int size = 0;

while (source[size] != '\000') size++;

node->str = new char [size];

for(int i = 0; i <= size; i++){

if(i < size){

node->str[i] = source[i];

node->length++;

} else{

node->str[i] = node->marker;

}

}

node->next = nullptr;

if (isEmpty()) { // Если список пуст создаем первый узел

head = node;

last = node;

} else { // иначе добавляем в конец списка

Node \*current = last;

current->next = node;

last = node;

}

}

Для проверки списка на пустоту используется метод isEmpty:

bool isEmpty() {

return head == nullptr;

}

Для удаления элемента по индексу используется метод removeByIndex:

bool removeByIndex(int i) {

if (countElem < i) {

printf("Can't found by index: %d, Total %d elements", i, countElem);

return false;

}

Node \*current = head;

int index = 1;

while (current->next != nullptr) {

if (i == 1) { // если номер элемента == 1 (голова списка)

Node \*remove = current;

head = current->next;

print(remove);

delete remove;

countElem--;

return true;

}

if (index + 1 == i) {

Node \*remove = current->next;

if (remove == last) { // если удаляемый элемент конец списка

last = current;

print(remove);

delete remove;

countElem--;

return true;

}

current->next = current->next->next;

print(remove);

delete remove;

countElem--;

return true;

}

index++;

current = current->next;

}

return true;

}

Для удаления n элементов, начиная с элемента k используется функция removeFromTo:

void removeFromTo(int i, int k)

{

if (countElem < i) {

printf("Can't found by index: %d, Total %d elements\n", i, countElem);

return;

}

if ((i + k) >countElem ) {

printf("Can't delete: %d, Total %d elements after index: %d\n ", k, countElem - i, i);

return;

}

if (i < 1) {

printf("Can't found by index: %d, The first element: %d \n", i, 1);

return;

}

if (k < 0) {

printf("Can't remove: %d\n", k);

return;

}

for (int j = 0; j <= k-1; ++j) {

removeByIndex(i);

}

}

Для печати строки элемента используется метод print:

void print(Node \*node) {

printf("Node with string remove:\n");

for (int i = 0; node->str[i] != node->marker; i++) {

std::cout << (node->str[i]);

}

std::cout << '\n';

}

Для возврата строки по номеру элемента используется метод getStr:

char \* getStr(int i) {

Node \*current = head;

int currentIndex = 1;

while (current != nullptr) {

if (currentIndex == i) {

return current->str;

}

currentIndex++;

current = current->next;

}

return const\_cast<char \*>(empt);

}

Деструкторы LinkedList и Node для удаления списка через оператор delete:

~Node() {

str = nullptr;

marker = 0;

}

~LinkedList(){

if (isEmpty()) return;

Node \*curr = head;

while (curr != nullptr) {

Node \*remove = curr;

curr = curr->next;

head = curr;

delete remove;

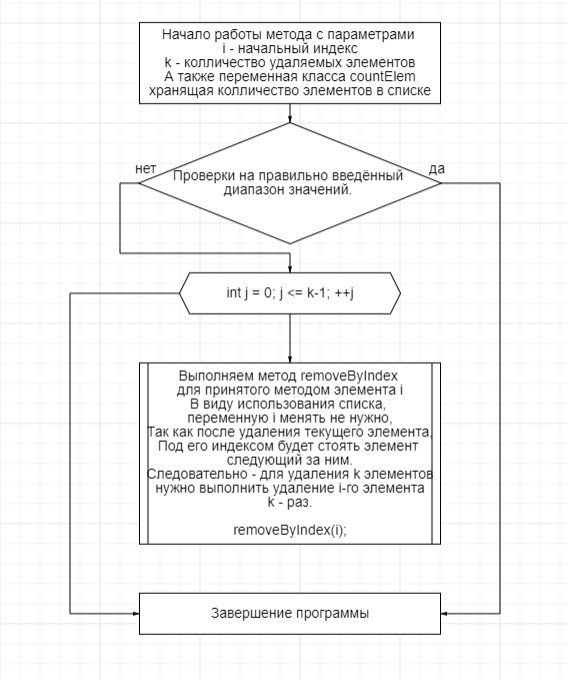
}

countElem = 0;

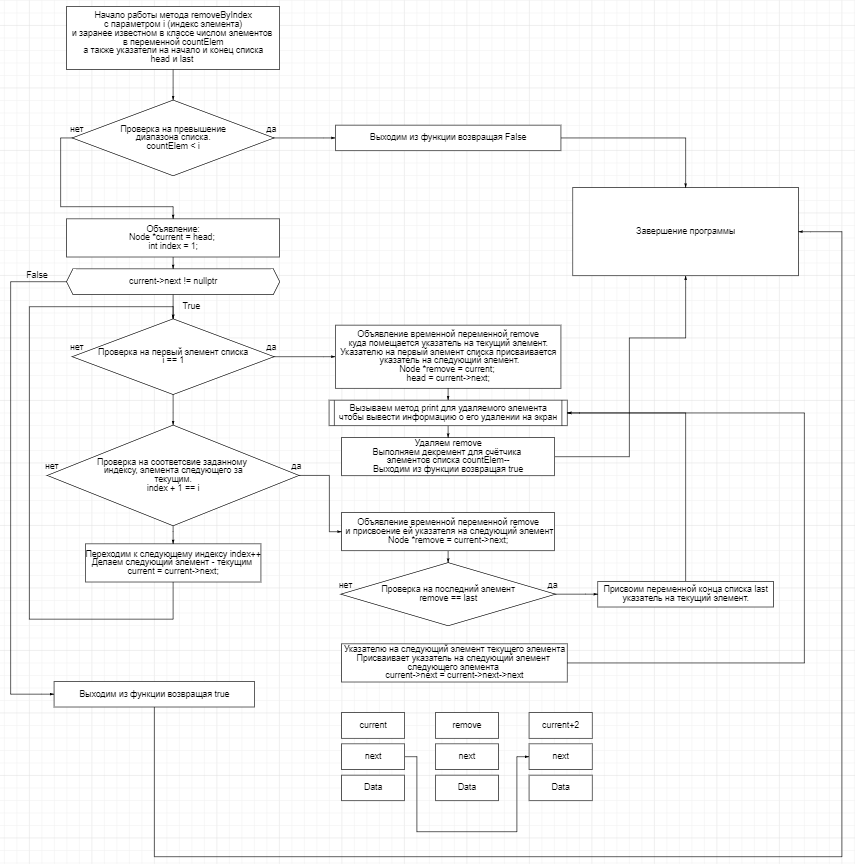
last = nullptr;

}

Блок-схема для метода removeByIndex.



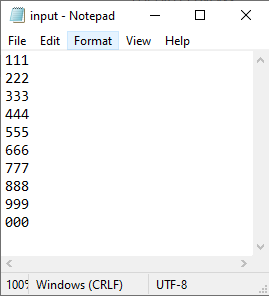
Блок-схема метода removeByIndex.

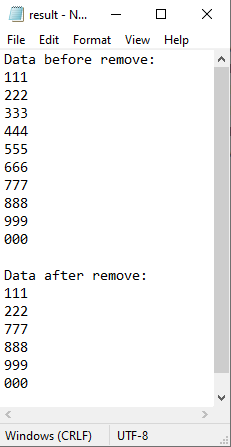


Тестовые данные:

int startIndex = 3; Индекс с которого начнётся удаление (включительно).

int countElements = 4; Количество удаляемых элементов.

 Файл input.txt

Файл result.txt после запуска программы.

Код программы:

#include <iostream>

#include <fstream>

using namespace std;

// Узел связного списка

struct Node {

char \*str; // строка

char marker = '@'; //маркер

int length = 0; // длина

Node \*next;

~Node() {

str = nullptr;

marker = 0;

}

};

// Класс для реализации односвязного списка

class LinkedList {

Node \*last;

Node \*head;

const char\* empt = "";

public:

int countElem;

LinkedList() {

head = nullptr;

last = nullptr;

countElem = 0;

}

~LinkedList(){

if (isEmpty()) return;

Node \*curr = head;

while (curr != nullptr) {

Node \*remove = curr;

curr = curr->next;

head = curr;

delete remove;

}

countElem = 0;

last = nullptr;

}

// Для добавления элемента в список

void addString(char source[1000]) {

countElem++;

Node \*node = new Node;

int size = 0;

while (source[size] != '\000') size++;

node->str = new char [size];

for(int i = 0; i <= size; i++){

if(i < size){

node->str[i] = source[i];

node->length++;

} else{

node->str[i] = node->marker;

}

}

node->next = nullptr;

if (isEmpty()) { // Если список пуст создаем первый узел

head = node;

last = node;

} else { // иначе добавляем в конец списка

Node \*current = last;

current->next = node;

last = node;

}

}

// проверка на пустоту списка

bool isEmpty() {

return head == nullptr;

}

// удалить элемент по заданному номеру

bool removeByIndex(int i) {

if (countElem < i) {

printf("Can't found by index: %d, Total %d elements", i, countElem);

return false;

}

Node \*current = head;

int index = 1;

while (current->next != nullptr) {

if (i == 1) { // если номер элемента == 1 (голова списка)

Node \*remove = current;

head = current->next;

print(remove);

delete remove;

countElem--;

return true;

}

if (index + 1 == i) {

Node \*remove = current->next;

if (remove == last) { // если удаляемый элемент конец списка

last = current;

print(remove);

delete remove;

countElem--;

return true;

}

current->next = current->next->next;

print(remove);

delete remove;

countElem--;

return true;

}

index++;

current = current->next;

}

return true;

}

// удалить k элементов, начиная с элемента i

void removeFromTo(int i, int k)

{

if (countElem < i) {

printf("Can't found by index: %d, Total %d elements\n", i, countElem);

return;

}

if ((i + k) >countElem ) {

printf("Can't delete: %d, Total %d elements after index: %d\n ", k, countElem - i, i);

return;

}

if (i < 1) {

printf("Can't found by index: %d, The first element: %d \n", i, 1);

return;

}

if (k < 0) {

printf("Can't remove: %d\n", k);

return;

}

for (int j = 0; j <= k-1; ++j) {

removeByIndex(i);

}

}

// распечатать строку элемента

void print(Node \*node) {

printf("Node with string remove:\n");

for (int i = 0; node->str[i] != node->marker; i++) {

std::cout << (node->str[i]);

}

std::cout << '\n';

}

// вернуть строку по номеру элемента

char \* getStr(int i) {

Node \*current = head;

int currentIndex = 1;

while (current != nullptr) {

if (currentIndex == i) {

return current->str;

}

currentIndex++;

current = current->next;

}

return const\_cast<char \*>(empt);

}

};

int main() {

cout << "Laboratory work number 2" << endl;

cout << "Completed by a student of the group 0336" << endl;

cout << "Nasekin S.A." << endl << endl;

int startIndex;

int countElements;

LinkedList \*linkedList = new LinkedList;

// запись строк из файла в связный список, где каждый узел списка представляет строку с маркером

ifstream input("..\\input.txt"); //открываем поток для чтения файла

if (input.is\_open()) {

char temp[1000];

while (input.getline(temp, 1000)) {

linkedList->addString(temp); // запись очередной строки файла в список

}

} else {

printf("Can't found input.txt");

return 1;

}

cout << "Data before remove: " << endl;

for (int i = 1; i <= linkedList->countElem; i++) {

char \*stringMarker = linkedList->getStr(i);

for (int j = 0; stringMarker[j] != '@'; j++) {

cout << stringMarker[j];

}

cout << "\n";

}

cout << "Enter start index and how many elements to remove:" << endl;

cout << "Start index: ";

cin >> startIndex;

cout << endl << "How many elements to remove: ";

cin >> countElements;

cout << endl;

ofstream result("..\\result.txt", ios::out);

if (result.is\_open()) {

result << "Data before remove:\n";

for (int i = 1; i <= linkedList->countElem; i++) {

char \*stringMarker = linkedList->getStr(i);

for (int j = 0; stringMarker[j] != '@'; j++) {

result << stringMarker[j];

}

result << "\n";

}

linkedList->removeFromTo(startIndex,countElements);

result << "\nData after remove:\n";

for (int i = 1; i <= linkedList->countElem; i++) {

char \*stringMarker = linkedList->getStr(i);

for (int j = 0; stringMarker[j] != '@'; j++) {

result << stringMarker[j];

}

result << "\n";

}

}

result.close();

cout << "Data after remove: " << endl;

for (int i = 1; i <= linkedList->countElem; i++) {

char \*stringMarker = linkedList->getStr(i);

for (int j = 0; stringMarker[j] != '@'; j++) {

cout << stringMarker[j];

}

cout << "\n";

}

delete linkedList;

return 0;

}

В ходе работы изучена структура данных однонаправленный линейный список, деструкторы для удаления объектов из памяти, работа с файлами.