Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**Списки**

**Отчет по лабораторной работе №3**

**По дисциплине**

**«Структуры и алгоритмы обработки данных в ЭВМ»**

Выполнил: студент гр. 439-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Зозуля Е.Д.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Проверил: ассистент каф. АСУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Яблонский Я.В.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Томск 2020

# Задание на лабораторную работу

Вариант 9.

Подготовить текстовый файл, содержащий не менее 15 целых чисел. Прочитать данные из этого файла и сформировать односвязный список. Изменить последовательность указателей так, чтобы отрицательные числа находились в начале списка. Распечатать исходный и итоговый списки. После завершения работы со списком освободите занимаемую им динамическую память..

# Алгоритм решения задачи

Формируется первый список, который остаётся неотсортированным.

Этот список копируется во второй. После каждый отрицательный элемент поочерёдно удаляется и добавляется в начало второго списка.

# Листинг программы

main.cpp

#include <fstream>

#include <iostream>

#include <ctime>

#include "List.h"

using namespace std;

using namespace Mine;

void moveNegativeToTheBeginning(List &SecondList, const List &FirstList)

{

ListNode \*h = FirstList.getHead();

SecondList.insertionInEnd(h->info);

while (h->next != NULL)

{

h = h->next;

SecondList.insertionInEnd(h->info);

}

//Скопировали первый список во второй

ListNode \*pointer1 = SecondList.getHead();

int i=0;

while(pointer1->info<0)//проходим первые отрицательные элементы (если они есть)

{

i++;

pointer1 = pointer1->next;

}

while (pointer1->next != NULL)//убираем оставшиеся отрицательные элементы (если они есть)

{

if(pointer1->next->info<0)

{

SecondList.insertionInFront(SecondList.removeByIndex(i+1));

}

else

{

pointer1=pointer1->next;

}

i++;

}

}

int main()

{

setlocale(LC\_CTYPE, "RUSSIAN");

srand(static\_cast<unsigned int>(time(0)));

List FirstList;

ofstream Gen;

Gen.open("Input.txt");

for (int i = 0; i < 15; i++)

{

Gen << rand() % 20 - 10;

Gen << "\n";

}

Gen.close();

ifstream FileIn("Input.txt");

int rNum;

while (!FileIn.eof())

{

FileIn >> rNum;

FirstList.insertionInEnd(rNum);

}

//Считали числа из файла в первый список

List SecondList;

moveNegativeToTheBeginning(SecondList, FirstList);

cout<<"Список FirstList:\n";

FirstList.PrintList();

cout<<"Список SecondList:\n";

SecondList.PrintList();

system("pause");

return 0;

}

List.h

namespace Mine

{

struct ListNode

{

int info;

ListNode \*next;

};

class List

{

public:

List(); //Конструктор

~List(); //Деструктор

int viewFirstItem() const; //прочитать первый элемент списка

int removeFirstItem(); //удалить первый элемент списка

void insertionInFront(int value); //добавить новый элемент в начало списка

void insertionInFront(ListNode\* &elem);//добавить существующий элемент в начало списка

int viewLastItem() const; //прочитать последний элемент списка

int removeLastItem(); //удалить последний элемент списка

void insertionInEnd(int value); //добавить новый элемент в конец списка

void insertionInEnd(ListNode\* &elem); //добавить существующий элемент в конец списка

void insertionByIndex(int value, int index); //добавить новый элемент в список после элемента с индексом

void insertionByIndex(ListNode\* &elem, int index); //добавить существующий элемент в список после элемента с индексом

int viewByIndex(int index) const; //прочитать элемент списка по индексу

int removeByIndex(int index); //удалить элемент списка по индексу

bool isEmpty() const; //проверить список на пустоту

void PrintList() const; //вывести содержимое списка на экран

ListNode\* getHead() const;

ListNode\* getTail() const;

private:

ListNode \*Head;

ListNode \*Tail;

};

}

List.cpp

#include "List.h"

#include <iostream>

namespace Mine

{

List::List()

{

Head = NULL;

Tail = NULL;

}

List::~List()

{

while(Head!=NULL)

{

ListNode \*p;

p = Head;

Head = Head->next;

delete p;

}

Head = Tail = nullptr;

}

int List::viewFirstItem() const

{

if(isEmpty())

{

std::cout << "Попытка чтения из пустого списка"<< std::endl;

throw "Попытка чтения из пустого списка";

}

return (Head->info);

}

int List::removeFirstItem()

{

if(isEmpty())

{

std::cout << "Попытка удаления из пустого списка"<< std::endl;

throw "Попытка удаления из пустого списка";

}

int value = Head->info;

ListNode \*pointer = Head;

Head = Head->next;

delete pointer;

return value;

}

void List::insertionInFront(int value)

{

ListNode \*newNode; //создаём указатель на НОВУЮ запись

newNode = new ListNode; //выделяем память

newNode->info = value; //записываем информацию в новое поле

if(isEmpty())

{

newNode->next = NULL;

Head = Tail = newNode;

}

else

{

newNode->next = Head;

Head = newNode;

}

}

void List::insertionInFront(ListNode\* &elem)

{

if(isEmpty())

{

elem->next = NULL;

Head = Tail = elem;

}

else

{

elem->next = Head;

Head = elem;

}

}

int List::viewLastItem() const

{

if(isEmpty())

{

std::cout << "Попытка чтения из пустого списка"<< std::endl;

throw "Попытка чтения из пустого списка";

}

return (Tail->info);

}

int List::removeLastItem()

{

if(isEmpty())

{

std::cout << "Попытка удаления из пустого списка"<< std::endl;

throw "Попытка удаления из пустого списка";

}

int value = Tail->info;

ListNode \*pointer = Tail;

ListNode \*i = Head;

while (i->next != Tail)

{

i = i->next;

}

i->next = NULL;

Tail = i;

delete pointer;

return value;

}

void List::insertionInEnd(int value)

{

ListNode \*newNode; //создаём указатель на НОВУЮ запись

newNode = new ListNode; //выделяем память

newNode->info = value; //записываем информацию в новое поле

if(isEmpty())

{

newNode->next = NULL;

Head = Tail = newNode;

}

else

{

newNode->next = NULL;

Tail->next = newNode;

Tail = newNode;

}

}

void List::insertionInEnd(ListNode\* &elem)

{

if(isEmpty())

{

elem->next = NULL;

Head = Tail = elem;

}

else

{

elem->next = NULL;

Tail->next = elem;

Tail = elem;

}

}

void List::insertionByIndex(int value, int index)

{

if(isEmpty())

{

std::cout << "Попытка поиска несуществующего элемента при записи по индексу"<< std::endl;

throw "Попытка поиска несуществующего элемента при записи по индексу";

}

ListNode \*p = Head;

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

{

if(p->next == NULL)

{

std::cout << "Попытка поиска несуществующего элемента при записи по индексу"<< std::endl;

throw "Попытка поиска несуществующего элемента при записи по индексу";

}

p = p->next;

}

ListNode \*newNode; //создаём указатель на НОВУЮ запись

newNode = new ListNode; //выделяем память

newNode->info = value; //записываем информацию в новое поле

newNode->next = p->next;

p->next = newNode;

}

void List::insertionByIndex(ListNode\* &elem, int index)

{

if(isEmpty())

{

std::cout << "Попытка поиска несуществующего элемента при записи по индексу"<< std::endl;

throw "Попытка поиска несуществующего элемента при записи по индексу";

}

ListNode \*p = Head;

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

{

if(p->next == NULL)

{

std::cout << "Попытка поиска несуществующего элемента при записи по индексу"<< std::endl;

throw "Попытка поиска несуществующего элемента при записи по индексу";

}

p = p->next;

}

elem->next = p->next;

p->next = elem;

}

int List::viewByIndex(int index) const

{

if(isEmpty())

{

std::cout << "Попытка чтения из пустого списка"<< std::endl;

throw "Попытка чтения из пустого списка";

}

ListNode \*p = Head;

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

{

if(p->next == NULL)

{

std::cout << "Попытка чтения несуществующего элемента"<< std::endl;

throw "Попытка чтения несуществующего элемента";

}

p = p->next;

}

return(p->info);

}

int List::removeByIndex(int index)

{

if(index<0)

{

std::cout << "Попытка удаления несуществующего элемента"<< std::endl;

throw "Попытка удаления несуществующего элемента";

}

if(isEmpty())

{

std::cout << "Попытка удаления из пустого списка"<< std::endl;

throw "Попытка удаления из пустого списка";

}

ListNode \*pointer;

if (index==0)

{

pointer = Head;

Head = Head->next;

}

else

{

ListNode \*p = Head;

for(int i=0; i<index-1; i++)

{

if(p->next == NULL)

{

std::cout << "Попытка поиска несуществующего элемента при удалении по индексу"<< std::endl;

throw "Попытка поиска несуществующего элемента при удалении по индексу";

}

p = p->next;

}

if(p->next == NULL)

{

std::cout << "Попытка чтения несуществующего элемента"<< std::endl;

throw "Попытка чтения несуществующего элемента";

}

pointer = p->next;

p->next=pointer->next;

}

return(pointer->info);

delete pointer;

pointer = nullptr;

}

bool List::isEmpty() const

{

return (Head == NULL);

}

void List::PrintList() const

{

ListNode \*p = Head;

while (p != NULL)

{

printf("%d; ", p->info);

p = p->next;

}

printf("\n");

}

ListNode\* List::getHead() const

{

return(Head);

}

ListNode\* List::getTail() const

{

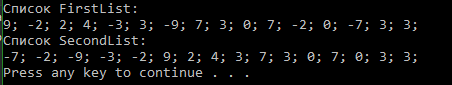
return(Tail);

}

}

# Пример решения

Скриншоты успешной работы программы, входные данные тестового примера, выходные данные.



# Вывод

Было изучено АТД список и использовано для решения задачи.