Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе №7**

**« Структура хранения данных: односвязный линейный список с использованием указателей»**

Выполнила:

студентка ф-та ИТММ ПМИ – 381903-3

Семибабнова Анна Владимировна

Проверил:

ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

Нижний Новгород

2020

Содержание

[Введение 3](#_Toc58264021)

[Постановка задачи 4](#_Toc58264022)

[Руководство пользователя 5](#_Toc58264023)

[Руководство программиста 7](#_Toc58264024)

[Описание структуры программы 7](#_Toc58264025)

[Описание структур данных 7](#_Toc58264026)

[Описание алгоритмов 10](#_Toc58264027)

[Эксперименты 11](#_Toc58264028)

[Заключение 12](#_Toc58264029)

[Литература 13](#_Toc58264030)

[Приложения 13](#_Toc58264031)

# Введение

Структура данных, представляющая собой конечное множество упорядоченных элементов (узлов), связанных друг с другом посредством указателей, называется связным списком. Каждый элемент связного списка содержит поле с данными, а также указатель (ссылку) на следующий и/или предыдущий элемент. Эта структура позволяет эффективно выполнять операции добавления и удаления элементов для любой позиции в последовательности. Причем это не потребует реорганизации структуры, которая бы потребовалась в массиве. Минусом связного списка, как и других структур типа «список», в сравнении его с массивом, является отсутствие возможности работать с данными в режиме произвольного доступа, т. е. список – структура последовательно доступа, в то время как массив – произвольного. Последний недостаток снижает эффективность ряда операций.

По типу связности выделяют в основном односвязные и двусвязные списки, а также некоторые другие.

Каждый узел односвязного (однонаправленного связного) списка содержит указатель на следующий узел.

# Постановка задачи

Реализовать список ( односвязный ). Продемонстрировать его работу в main. Написать нескольких простых тестов на базе Google Test для проверки работоспособности программы. Опубликовать код в личном репозитории на GitHub.

# Руководство пользователя

Использование программы пользователем:

1. Зайти в интересуемый репозиторий на GitHub.
2. Перейти в нужную ветку под названием Lab7.
3. Скачать код программы. Это легко можно сделать, скачав файлы программы архивом. Для этого нужно нажать на кнопку “Code”, затем на кнопку “Download ZIP”. После этих действий появится окно проводника. Нужно сохранить данный файл на рабочий стол и извлечь из него файлы программы (нажать правой кнопкой мыши на файл, выбрать команду “Извлечь все”).

# Открыть папку с программой на рабочем столе.

1. Создать папку “build”.
2. Перейти в эту папку, нажать правой кнопкой мыши в поле проводника, нажать на “Git Bash Here” (предварительно нужно скачать программу Git с официального сайта).
3. Благодаря действиям из предыдущего пункта на экране появится консоль, в которой нужно прописать команду “cmake ../” (предварительно нужно скачать программу CMake, позволяющую автоматически собрать программу из исходного кода).

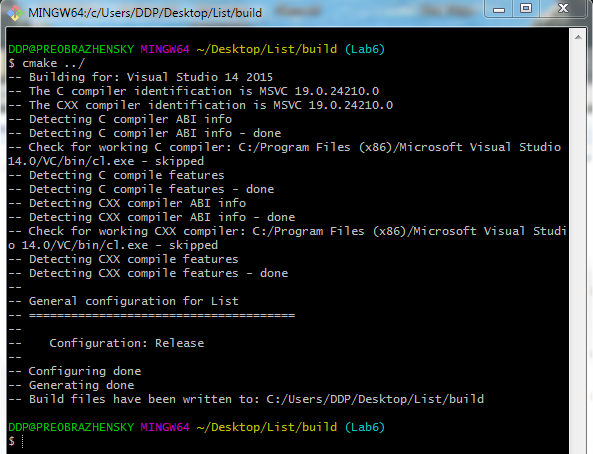


Рис1. Скриншот консоли Git Bash сборки программы

1. Теперь проект List.sln можно открыть в Visual Studio.
2. Пользователю нужно нажать на проект “SamplePrimeList” правой кнопкой мыши и выбрать пункт “Назначить автозагружаемым проектом”.
3. Затем нужно нажать на сочетание клавиш CTRL+F5 на клавиатуре. Откроется консольное приложение, демонстрирующее работу списка (вставка в начало, вставка в конец, использование итератора).

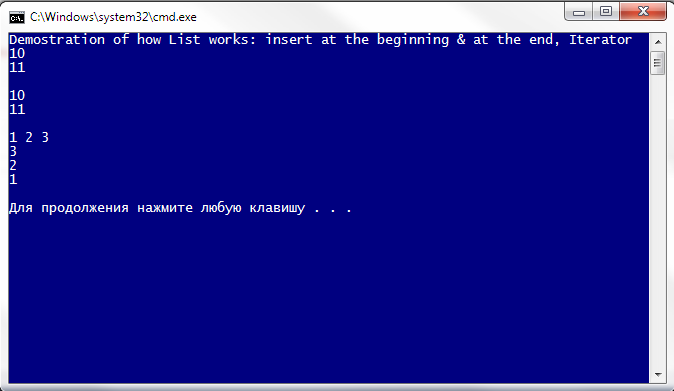


Рис.2 Скриншот интерфейса программы

# Руководство программиста

## Описание структуры программы

Решение программы состоит из 6 проектов.

В проектах ALL\_BUILD, gtest и ZERO\_CHECK определен вспомогательный код, необходимый для работы тестовой системы и всей программы в целом.

В проекте List содержится 1 модуль – List.h. В нем определены шаблонные классы TListElem, TList и TListIterator, методы и их определения.

В проекте SamplePrimeList содержится 1 модуль – SamplePrimeList.cpp. В нем объявлена функция main с примером работы списка.

В проекте test\_List содержатся 2 модуля: test\_List.cpp, test\_main.cpp. В них содержится набор тестов и вызов этих тестов для проверки корректности работы программы.

## Описание структур данных

В программе определены следующие шаблонные классы: TListElem, TList.

Внутри класса TListElem определены следующие protected-поля:

* T data – данные элемента;
* TListElem\* next – указатель на следующий элемент;
* TListElem\* prev – указатель на предыдущий элемент;

Также определены следующие public-методы:

* TListElem(T \_data) – конструктор-инициализатор;
* ~TListElem() – деструктор;
* T GetData() – метод для получения данных;
* TListElem\* GetNext() – метод для получения указателя на следующий элемент;
* TListElem\* GetPrev() – метод для получения указателя на предыдущий элемент;
* void SetData(T \_data) – метод для задания данных;
* void SetNext(TListElem\* \_next) – метод для задания указателя на следующий элемент;
* void SetPrev(TListElem\* \_prev) – метод для задания указателя на предыдущий элемент;
* friend ostream& operator<< (ostream& ostr, const TListElem<T1>& A) – оператор вывода;
* friend istream& operator >> (istream& istr, TListElem<T1>& A) – оператор ввода.

Внутри класса TList определены следующие protected-поля:

* TListElem<T>\* root – указатель на головной элемент списка;
* TListElem<T>\* end – указатель на хвостовой элемент списка;
* int count ­– количество элементов в списке;

Также определены следующие public-методы:

* TList() – конструктор по умолчанию;
* TList(TList<T>& \_v) – конструктор копирования;
* ~TList() – деструктор;
* TList<T>& operator =(TList<T>& \_v) – оператор =;
* void InsFirst(T d) – метод вставки в начало;
* void InsLast(T d) – метод вставки в конец;
* void Ins(TListElem<T>\* e, T d) – метод вставки на указанную позицию;
* TListElem<T>\* GetFirst() – метод получения указателя на первый элемент списка;
* TListElem<T>\* GetLast() – метод получения указателя на последний элемент списка;
* void DelFirst() – метод удаления элемента с начала;
* void DelLast() – метод удаления элемента с конца;
* void Del(TListElem<T>\* e) – метод удаления элемента с указанной позиции;
* bool IsEmpty(void) const – метод, проверяющий, пустой ли список;
* friend ostream& operator<< (ostream& ostr, const TList<T1> &A) – оператор вывод;
* friend istream& operator >> (istream& istr, TList<T1> &A) – оператор ввода;
* int GetCount() – метод получения количества элементов в списке;
* void reverse() – метод, разворачивающий список;
* void cleanse() – метод, удаляющий одинаковые элементы в списке.

## Описание алгоритмов

# В данной лабораторной работе отсутствуют алгоритмы, требующие подробного описания.

# Эксперименты

Все тесты работают успешно, следовательно, программа прошла проверку и работает правильно.

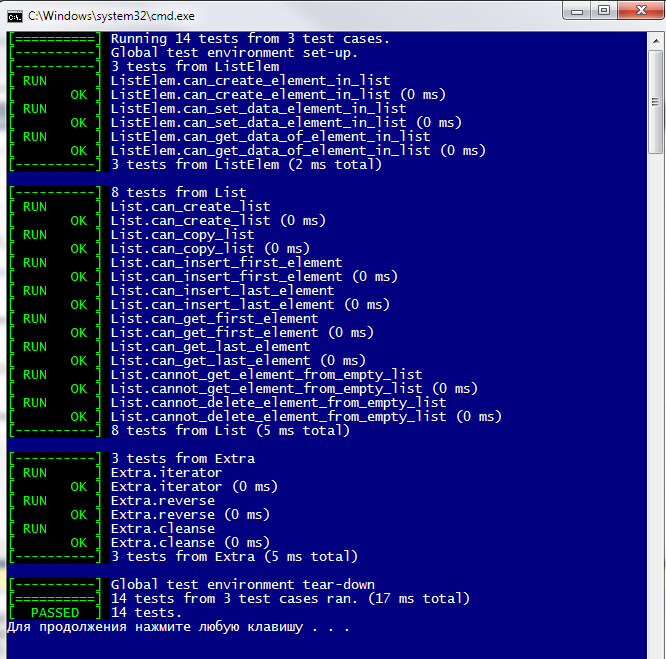


Рис 3. Скриншот выполненных тестов

# Заключение

Таким образом, в рамках данной лабораторной работы была успешно создана программа, которая позволяет работать со списком. Программа полностью выдержала проверку и готова к использованию.

# Литература

1. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003.
2. Вирт Н. Алгоритмы + структуры данных = программы. — М.: Мир, 1985. — С. 406.
3. Официальный сайт программы PMBK. – Режим доступа <http://www.pmbk.ru/pr/chto-takoe-massivy-v-programmirovanii.html>
4. Официальный сайт Bestreferat. – Режим доступа <https://www.bestreferat.ru/referat-183091.html>
5. Официальный сайт Habr. – Режим доступа <https://habr.com/ru/post/339656/>
6. Официальный сайт Microsoft. – Режим доступа <https://docs.microsoft.com/ru-ru/cpp/c-language/main-function-and-program-execution?view=vs-2019>

# Приложения

## Код программы

**List.h**

#include <iostream>

using namespace std;

template <class T>

class TListElem

{

protected:

T data;

TListElem\* next;

TListElem\* prev;

public:

TListElem(T \_data);

~TListElem();

T GetData();

TListElem\* GetNext();

TListElem\* GetPrev();

void SetData(T \_data);

void SetNext(TListElem\* \_next);

void SetPrev(TListElem\* \_prev);

template <class T1>

friend ostream& operator<< (ostream& ostr, const TListElem<T1>& A);

template <class T1>

friend istream& operator >> (istream& istr, TListElem<T1>& A);

template<class T1>

friend class TList;

};

template <class T>

class TList

{

protected:

TListElem<T>\* root;

TListElem<T>\* end;

int count;

public:

TList();

TList(TList<T>& \_v);

~TList();

TList<T>& operator =(TList<T>& \_v);

void InsFirst(T d);

void InsLast(T d);

void Ins(TListElem<T>\* e, T d);

TListElem<T>\* GetFirst();

TListElem<T>\* GetLast();

void DelFirst();

void DelLast();

void Del(TListElem<T>\* e);

bool IsEmpty(void) const;

//bool IsFull(void) const;

template <class T1>

friend ostream& operator<< (ostream& ostr, const TList<T1> &A);

template <class T1>

friend istream& operator >> (istream& istr, TList<T1> &A);

int GetCount();

//extra

template<class T2>

friend class TListIterator;

void reverse();

void cleanse();

};

template<class T1>

inline ostream& operator<<(ostream& ostr, const TListElem<T1>& A)

{

ostr << A.data;

return ostr;

}

template<class T1>

inline istream& operator >> (istream& istr, TListElem<T1>& A)

{

istr >> A.data;

return istr;

}

template <class T1>

ostream& operator<< (ostream& ostr, const TList<T1> &A)

{

TListElem<T1>\* i = A.root;

while (i != 0)

{

ostr << \*i << endl;

i = i->GetNext();

}

return ostr;

}

template <class T1>

istream& operator >> (istream& istr, TList<T1> &A) {

int count;

istr >> count;

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

{

T1 d;

istr >> d;

A.InsLast(d);

}

return istr;

}

template<class T>

inline TList<T>::TList()

{

root = 0;

end = 0;

count = 0;

}

template <class T>

TList<T>::TList(TList<T>& \_v)

{

root = nullptr;

count = \_v.count;

TListElem<T>\* i = \_v.root;

TListElem<T>\* j = this->root;

TListElem<T>\* p = 0;

while (i != 0)

{

j = new TListElem<T>(\*i);

j->SetNext(0);

if (p != 0)

{

p->SetNext(j);

j->SetPrev(p);

}

p = j;

if (root == nullptr)

root = j;

end = j;

i = i->next;

}

}

template <class T>

TList<T>::~TList()

{

if (this->root != 0)

{

TListElem<T>\* i = this->root;

TListElem<T>\* p = 0;

while (i != 0)

{

p = i;

i = i->GetNext();

delete p;

}

this->root = 0;

this->end = 0;

count = 0;

}

}

template <class T>

TList<T>& TList<T>::operator =(TList<T>& \_v)

{

if (this == &\_v)

return \*this;

if (this->root != 0)

{

TListElem<T>\* i = this->root;

TListElem<T>\* p = 0;

while (i != 0)

{

p = i;

i = i->GetNext();

delete p;

}

this->root = 0;

this->end = 0;

count = 0;

}

root = nullptr;

count = \_v.count;

TListElem<T>\* i = \_v.root;

TListElem<T>\* j = this->root;

TListElem<T>\* p = 0;

while (i != 0)

{

j = new TListElem<T>(\*i);

j->SetNext(0);

if (p != 0)

{

p->SetNext(j);

j->SetPrev(p);

}

p = j;

if (root == nullptr)

root = j;

end = j;

i = i->next;

}

}

template<class T>

inline void TList<T>::InsFirst(T d)

{

TListElem<T>\* tmp = new TListElem<T>(d);

tmp->SetNext(root);

root = tmp;

if (end == 0)

end = tmp;

count++;

}

template<class T>

inline void TList<T>::InsLast(T d)

{

//if (end == root && root != 0)

//root->SetNext(tmp);

if (IsEmpty())

InsFirst(d);

else

{

TListElem<T>\* tmp = new TListElem<T>(d);

if (end != 0)

end->SetNext(tmp);

tmp->SetPrev(end);

tmp->SetNext(0);

end = tmp;

count++;

}

}

template<class T>

inline void TList<T>::Ins(TListElem<T>\* e, T d)

{

TListElem<T>\* tmp = new TListElem<T>(d);

tmp->SetNext(e->GetNext());

tmp->SetPrev(e);

e->GetNext()->SetPrev(tmp);

e->SetNext(tmp);

count++;

}

template<class T>

inline bool TList<T>::IsEmpty(void) const

{

return count == 0;

}

template<class T>

inline TListElem<T>\* TList<T>::GetFirst()

{

if ((\*this).IsEmpty())

throw logic\_error("empty\_list");

return root;

}

template<class T>

inline TListElem<T>\* TList<T>::GetLast()

{

if ((\*this).IsEmpty())

throw logic\_error("empty\_list");

return end;

}

template<class T>

inline void TList<T>::DelFirst()

{

if (root == 0) throw logic\_error("invalid argument");

TListElem<T>\* i = root;

root = root->GetNext();

delete i;

count--;

}

template<class T>

inline void TList<T>::DelLast()

{

if ((\*this).IsEmpty()) throw logic\_error("empty list");

TListElem<T>\* j = root;

if (j->GetNext() != 0) {

while (j->GetNext()->GetNext() != 0) {

j = j->GetNext();

}

delete j->GetNext();

j->SetNext(0);

return;

}

delete j;

root = 0;

}

template<class T>

inline void TList<T>::Del(TListElem<T>\* e)

{

e->GetPrev()->SetNext(e->GetNext());

e->GetNext()->SetPrev(e->GetPrev());

count--;

delete e;

}

template<class T>

inline int TList<T>::GetCount()

{

return count;

}

template<class T>

inline void TList<T>::reverse()

{

TListElem<T>\* temp = root;

while (temp != nullptr)

{

TListElem<T>\* temp2 = temp->GetNext();

temp->SetNext(temp->GetPrev());

temp->SetPrev(temp2);

temp = temp2;

}

temp = root;

root = end;

end = temp;

}

template<class T>

inline TListElem<T>::TListElem(T \_data)

{

data = \_data;

next = 0;

prev = 0;

}

template<class T>

inline TListElem<T>::~TListElem()

{

next = 0;

prev = 0;

}

template<class T>

inline T TListElem<T>::GetData()

{

return data;

}

template<class T>

inline TListElem<T>\* TListElem<T>::GetNext()

{

return next;

}

template<class T>

inline TListElem<T>\* TListElem<T>::GetPrev()

{

return prev;

}

template<class T>

inline void TListElem<T>::SetData(T \_data)

{

data = \_data;

}

template<class T>

inline void TListElem<T>::SetNext(TListElem<T>\* \_next)

{

next = \_next;

}

template<class T>

inline void TListElem<T>::SetPrev(TListElem<T>\* \_prev)

{

prev = \_prev;

}

template<class T2>

inline TListIterator<T2>::~TListIterator()

{

}

template<class T2>

inline bool TListIterator<T2>::CanGoFwd()

{

return (current->GetNext() != nullptr);

}

template<class T2>

inline bool TListIterator<T2>::CanGoBck()

{

return (current->GetPrev() != nullptr)

}

template<class T2>

inline void TListIterator<T2>::GoFwd()

{

if (!CanGoFwd())

throw - 1;

current = current->GetNext();

}

template<class T2>

inline void TListIterator<T2>::GoBck()

{

if (!CanGoBck())

throw - 1;

current = current->GetPrev();

}

template<class T2>

inline bool TListIterator<T2>::operator==(const TListIterator<T2>& \_v)

{

return (current == \_v.current);

}

template<class T2>

inline TListIterator<T2> TListIterator<T2>::operator++(int)

{

GoFwd();

return (\*this);

}

template<class T2>

inline TListIterator<T2> TListIterator<T2>::operator--(int)

{

GoBck();

return (\*this);

}

template<class T2>

inline TListIterator<T2>& TListIterator<T2>::operator=(const TListIterator<T2>& \_v)

{

link = \_v.link;

current = \_v.current;

}

template<class T2>

inline T2 TListIterator<T2>::GetData()

{

if (current == nullptr)

throw - 1;

return current->GetData();

}

**SamplePrimeList.h**

#include "List.h"

int main()

{

cout << "Demostration of how List works: insert at the beginning & at the end, Iterator" << endl;

TList<int> a;

a.InsFirst(10);

a.InsLast(11);

cout << a << endl;

TList<int> b(a);

cout << b << endl;

TList<int> c;

c.InsLast(1);

c.InsLast(2);

c.InsLast(3);

c.InsLast(1);

c.InsLast(2);

c.cleanse();

}

**test\_main.cpp**

#include <gtest.h>

int main(int argc, char \*\*argv)

{

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

**test\_List.cpp**

#include "List.h"

#include <gtest.h>

TEST(ListElem, can\_create\_element\_in\_list)

{

ASSERT\_NO\_THROW(TListElem<int> e(9));

}

TEST(ListElem, can\_set\_data\_element\_in\_list)

{

TListElem<int> e(9);

ASSERT\_NO\_THROW(e.SetData(10));

}

TEST(ListElem, can\_get\_data\_of\_element\_in\_list)

{

TListElem<int> e(5);

EXPECT\_EQ(e.GetData(), 5);

}

TEST(List, can\_create\_list)

{

ASSERT\_NO\_THROW(TList<int>());

}

TEST(List, can\_copy\_list)

{

TList<int> a;

a.InsLast(10);

TList<int> b(a);

EXPECT\_EQ(10, b.GetLast()->GetData());

}

TEST(List, can\_insert\_first\_element)

{

TList<int> f;

ASSERT\_NO\_THROW(f.InsFirst(5));

}

TEST(List, can\_insert\_last\_element)

{

TList<int> l;

ASSERT\_NO\_THROW(l.InsLast(5));

}

TEST(List, can\_get\_first\_element)

{

TList<int> f;

int o = 9;

int p = 3;

f.InsFirst(o);

f.InsLast(p);

EXPECT\_EQ(9, f.GetFirst()->GetData());

}

TEST(List, can\_get\_last\_element)

{

TList<int> l;

int o = 9;

int p = 3;

l.InsFirst(o);

l.InsLast(p);

EXPECT\_EQ(3, l.GetLast()->GetData());

}

TEST(List, cannot\_get\_element\_from\_empty\_list)

{

TList<int> e;

ASSERT\_ANY\_THROW(e.GetLast());

}

TEST(List, cannot\_delete\_element\_from\_empty\_list)

{

TList<int> e;

ASSERT\_ANY\_THROW(e.DelLast());

}

TEST(Extra, iterator)

{

TList<int> a;

a.InsLast(1);

a.InsLast(2);

a.InsLast(3);

TListIterator<int> i(a, a.GetFirst());

i++;

EXPECT\_EQ(2, i.GetData());

}

TEST(Extra, reverse)

{

TList<int> a;

a.InsLast(1);

a.InsLast(2);

a.InsLast(4);

a.reverse();

EXPECT\_EQ(4, a.GetFirst()->GetData());

}

TEST(Extra, cleanse)

{

TList<int> a;

a.InsLast(1);

a.InsLast(3);

a.InsLast(4);

a.InsLast(2);

a.InsLast(4);

a.cleanse();

EXPECT\_EQ(4, a.GetCount());

EXPECT\_EQ(2, a.GetLast()->GetData());

}