Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

Факультет вычислительной математики и кибернетики

Отчёт по лабораторной работе

Список на указателях

Выполнил:
студент ф-та ИИТММ гр. 381908-01
Козел С. А.

Проверил:
ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Нижний Новгород
2020 г.

Содержание

Введение

Связный список – это базовая структура данных, состоящая из узлов. Каждый узел хранит в себе собственные данные и указатель на следующий узел. Данная структура данных имеет большое преимущество перед обычными массивами, так как удаление/добавление элемента осуществляется переустановкой указателей, при этом сами данные не копируются, что обеспечивает высокую скорость (O (1)) вставки новых элементов.

Постановка задачи

Цель данной работы - разработка структуры данных для хранения списков с использованием указателей.

Выполнение работы предполагает решение следующих задач:

1. Реализация структуры Node.
2. Реализация класса списка List.
3. Реализация класса итератора для списка ListIterator.
4. Реализация метода для получения из списка элементов, нацело делящихся на число K.
5. Реализация методов для ввода/вывода структуры данных в файл
6. Публикация исходных кодов в личном репозиторий на GitHub.

Руководство пользователя

Пользователю нужно запустить файл List_two_arrays.exe.

Откроется консольное приложение для тестирования списков.

Программа покажет функциональность каждой функции по средствам вывода данных в консоль.

Руководство программиста

Описание структуры программы

Программа состоит из следующих модулей:

- Приложение List_two_arrays
- Статическая библиотека Node:
  - Node.h – описание звена списка
- Статическая библиотека List:
  - List.h – описание класса списка
- Статическая библиотека ListIterator:
  - ListIterator.h – описание класса итератора для списка
- Приложение main:
  - main.cpp – тестирование работы программы

Описание структур данных

Класс Node:

```
1. Node(int _data = 0, Node* pNext = nullptr); - конструктор;
```

Класс List:

```
1. List() - конструктор;
2. ~List() - деструктор;
3. int GetListLenght() – получить размер списка;
4. bool IsEmpty() – проверка на пустоту списка;
5. void InsFirst(int Val) – вставка в начало;
6. void InsLast(int Val) – вставка в конец;
7. void DelFirst() – удаление элемент из начала;
8. void DelLast() – удаление элемента с конца;
9. void DelList() – удаление всего списка;
10. void InsValue(int Val, int pos) – вставка значение по индексу;
11. void DelValue(int pos) – удаление значения по индексу;
12. void print() – вывод списка в консоль;
13. bool search(int data) – проверка содержится ли переданный элемент в списке;
14. int get(int pos) – получить элемент по позиции;
15. bool IsSort() – проверка отсортирован ли список по возрастанию;
16. List* findSpecialElements(int K) - Возвращает список в котором каждый элемент
    делится на цело на K;
17. void WriteFile(std::string path) – вывод списка в файл;
18. void ReadFile (std::string path) – чтение списка с файла;
```

Класс ListIterator:

```
1. ListIterator() - конструктор;
2. ListIterator(Node* n) – конструктор с параметром;
3. void init(Node* n) – функция инициализации;
4. bool check_current() – проверка не пуст ли текущий указатель;
5. bool check_next() – проверка выделена ли память под следующий указатель;
6. void go_next() – сдвинуть указатель на следующую позицию;
7. int  get_value() – получить значение по текущему указателю;
8. Node* del_cur(Node* n) – удалить переданный указатель;
9. Node* insert(int val) – вставить новое звено;
10. void Del_It() – удалить итератор;
```

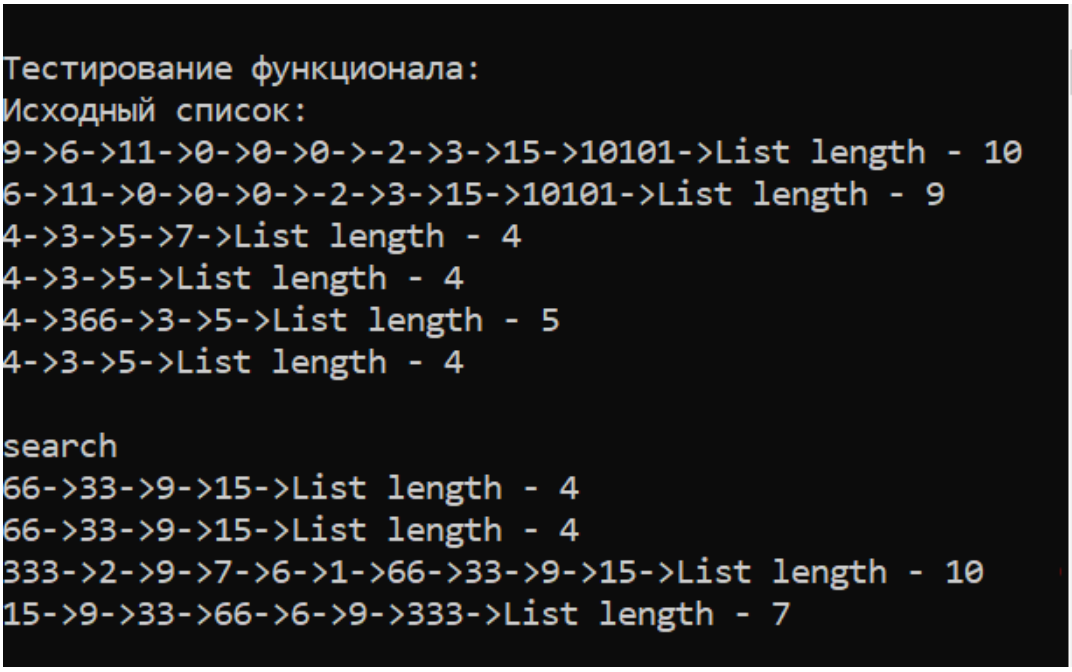
Описание алгоритмов

Принцип работы списка

Инициализация – добавление первого звена в список, который будет являться корнем.

Добавление узла - указатель на предыдущие звено перепривязывается на указатель добавляемого узла, добавляемый узел указывает либо на следующий элемент, либо на null, если он добавляется в конец.

Эксперименты



Заключение

При выполнении данной работы мною была полностью изучена и успешно реализована структура данных список. Полученный опыт является очень полезным и нужным, так как списки используются повсеместно.

Литература

1. https://prog-cpp.ru/data-ols/
2. https://ru.wikipedia.org/wiki/Связный_список#Односвязный_список_(однонаправленный_связный_список)
3. https://codelessons.ru/cplusplus/spisok-list-v-s-polnyj-material.html#2

Приложения

Приложение 1

Node.h

```
class Node
{
public:
    int data;
    Node* Next;

    Node(int _data = 0, Node* pNext = nullptr);
};

Node::Node(int _data, Node* pNext)
{
    this->data = _data;
    this->Next = pNext;
}
```

Приложение 2

List.h

```
class List
{
protected:
    Node* pFirst;
    Node* pLast;
    ListIterator it;
    int ListLen;
public:
    List();
    ~List();

    int GetListLenght();        // Размер списка
    bool IsEmpty();             // Проверка на пустоту

    void InsFirst(int Val);     // Вставить в начало
    void InsLast(int Val);      // Вставить в конец

    void DelFirst();            // Удалить элемент в начале
    void DelLast();             // Удалить элемент в конце
    void DelList();             // Удалить список

    void InsValue(int Val, int pos);    // Вставить значение по позиции
    void DelValue(int pos);             // Удалить значение по позиции

    void print();                       // Вывести список в консоль

    bool search(int data);              // Проверка есть ли элемент в списке
    int  get(int pos);                  // Получить значение по позиции
    bool IsSort();                      // Проверка отсортирован ли список по возрастанию

    List* findSpecialElements(int K);   // Возвращает список в котором каждый элемент делится на цело на K

    void WriteFile(std::string path);   // Вывести список в файл
    void ReadFile (std::string path);   // Считать список с файла
};


List::List()
{
    pFirst = nullptr;
    pLast = nullptr;
    ListLen = 0;
}


 List::~List() { DelList(); }


 int List::GetListLenght() { return ListLen; }


bool List::IsEmpty()
{
    if (ListLen == 0) { return true; }
    else { return false; }
}


void List::InsFirst(int Val)
{
    it.init(pFirst);
    if (!it.check_current())
    {
        pFirst = it.insert(Val);
        pLast = pFirst;
    }
    else
    {
        pFirst = it.insert(Val);
    }
    ListLen++;
}
```

```cpp
void List::InsLast(int Val)
{
    it.init(pFirst);
    if (!it.check_current())
    {
        pFirst = it.insert(Val);
        pLast = pFirst;
    }
    else
    {
        Node* tmp = new Node(Val);
        pLast->Next = tmp;
        pLast = tmp;
    }
    ListLen++;
}


void List::DelFirst()
{
    it.init(pFirst);
    if (!it.check_current())
    {
        return;
    }
    if (pFirst == pLast)
    {
        delete pFirst;
        pFirst = nullptr;
        pLast = nullptr;
        ListLen--;
        return;
    }
    pFirst = it.del_cur(pFirst);
    ListLen--;
}


void List::DelLast()
{
    it.init(pFirst);
    if (!it.check_current())
    {
        return;
    }
    if (pFirst == pLast)
    {
        delete pFirst;
        pFirst = nullptr;
        pLast = nullptr;
        ListLen--;
        return;
    }

    Node* tmp = pFirst;
    while (tmp->Next->Next != nullptr)
    {
        tmp = tmp->Next;
    }
    delete tmp->Next;
    tmp->Next = nullptr;
    pLast = tmp;
}


void List::DelList()
{
    it.Del_It();
    pFirst = nullptr;
    pLast = nullptr;
    ListLen = 0;
}
```

```cpp
void List::InsValue(int Val, int pos)
{
    if (pos < 0)
    {
        return;
    }
    if (pos == 0)
    {
        InsFirst(Val);
        return;
    }
    if (pos >= ListLen)
    {
        InsLast(Val);
        return;
    }
    int count = 0;
    Node* tmp = pFirst;
    it.init(tmp);
    while (count != pos - 1)
    {
        it.go_next();
        count++;
    }
    Node* newNode = new Node(Val, tmp->Next);
    it.insert(Val);
    tmp->Next = newNode;
    ListLen++;
}


void List::DelValue(int pos)
{
    if (pos < 0 || pos >= ListLen)
    {
        return;
    }
    if (pos == 0)
    {
        DelFirst();
        return;
    }
    if (pos == ListLen - 1)
    {
        DelLast();
        return;
    }
    Node* tmp = pFirst;
    it.init(tmp);
    int count = 0;
    while (count != pos - 1)
    {
        it.go_next();
        count++;
    }
    Node* tmp2 = tmp->Next->Next;
    delete tmp->Next;
    tmp->Next = tmp2;
    ListLen--;
}

void List::print()
{
    it.init(pFirst);
    if (it.check_current())
    {
        while (it.check_next())
        {
            std::cout << it.get_value() << "->";
            it.go_next();
        }
        std::cout << it.get_value() << "->";
    }
    std::cout << "List length - " << GetListLenght() << "\n";
}
```

```cpp
bool List::search(int data)
{
    if (ListLen == 0)
    {
        return false;
    }
    if (ListLen == 1)
    {
        if (pFirst->data == data)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    it.init(pFirst);
    while (it.check_next())
    {
        if (it.get_value() == data)
        {
            return true;
        }
        it.go_next();
    }
    if (it.get_value() == data)
    {
        return true;
    }
    return false;
}


int List::get(int pos)
{
    if (pos < 0 || pos >= ListLen)
    {
        throw std::out_of_range("Input error: invalide input value");
    }
    it.init(pFirst);
    int count = 0;
    while (count != pos)
    {
        it.go_next();
        count++;
    }
    return it.get_value();

}

bool List::IsSort()
{
    for (int i = 0; i < ListLen - 1; i++)
    {
        if (get(i) > get(i + 1))
        {
            return false;
        }
    }
    return true;
}

inline List* List::findSpecialElements(int K)
{
    List* res = new List();
    it.init(pFirst);
    if (it.check_current())
    {
        Node* tmp = pFirst;
        while (it.check_next())
        {
            if (it.get_value() % K == 0)
            {
                res->InsFirst(it.get_value());
            }
            it.go_next();
        }
        // Последнее значение
        if (it.get_value() % K == 0)
        {
            res->InsFirst(it.get_value());
        }
    }
    return res;
}
```

```cpp
inline void List::WriteFile(std::string path)
{
    std::ofstream file(path, std::ios::trunc);
    if (file.is_open())
    {
        it.init(pFirst);
        if (it.check_current())
        {
            while (it.check_next())
            {

                file << it.get_value() << "->";
                it.go_next();
            }
            file << it.get_value();
        }
    }
    else
    {
        std::cout << "Files is not open!\n";
    }
    file.close();
}

inline void List::ReadFile(std::string path)
{
    std::ifstream file(path, std::ios::in);
    if (file.is_open())
    {
        std::string buf;

        int val[250];
        int i_val = 0;
        std::string str;
        while (std::getline(file, buf))
        {
            for (int i = 0; i < buf.size(); i++)
            {
                if (buf[i] == '-' && buf[i + 1] == '>')
                {
                    val[i_val] = std::atoi(str.c_str());
                    InsLast(val[i_val]);
                    i_val++;
                    str.clear();
                    i++;
                    continue;
                }
                str += buf[i];
            }
            val[i_val] = std::atoi(str.c_str());
            InsLast(val[i_val]);
        }
    }
    else
    {
        std::cout << "Files is not open!\n";
    }
    file.close();
}
```

Приложение 3

ListIterator.h

```
class ListIterator {
public:
    Node* NodeList;

    ListIterator();
    ListIterator(Node* n);

    void init(Node* n);

    bool check_current();
    bool check_next();

    void go_next();
    int  get_value();

    Node* del_cur(Node* n);

    Node* insert(int val);

    void Del_It();

};


ListIterator::ListIterator()
{
    NodeList = nullptr;
}

inline ListIterator::ListIterator(Node* n)
{
    if (n != nullptr)
    {
        NodeList = n;
    }
}

inline void ListIterator::init(Node* n)
{
    if (n != nullptr)
    {
        NodeList = n;
    }
}

inline bool ListIterator::check_current()
{
    if (NodeList != nullptr)
    {
        return true;
    }
    else { return false; }
}
```

```cpp
inline bool ListIterator::check_next()
{
    if (NodeList->Next != nullptr)
    {
        return true;
    }
    else { return false; }
}

inline void ListIterator::go_next()
{
    if (check_current())
    {
        if (check_next())
        {
            NodeList = NodeList->Next;
        }
    }
}

inline int ListIterator::get_value()
{
    if (check_current())
    {
        return NodeList->data;
    }
}

inline Node* ListIterator::del_cur(Node* n)
{
    NodeList = n;
    Node* tmp = NodeList;

    NodeList = NodeList->Next;
    delete tmp;
    return NodeList;
}

inline Node* ListIterator::insert(int val)
{
    return new Node(val, NodeList);
}

inline void ListIterator::Del_It()
{
    NodeList = nullptr;
}
```

Приложение 4

main.cpp

```
#define PATH "C:\\out.txt"

int main()
{
    setlocale(LC_ALL, "RUS");

    std::cout << "\nТестирование функционала: \n";
    List list1;
    List list2;

    List s;

    s.InsFirst(15);
    s.InsFirst(3);
    s.InsFirst(-2);
    s.InsFirst(0);
    s.InsFirst(0);
    s.InsFirst(0);
    s.InsFirst(11);
    s.InsFirst(6);
    s.InsFirst(9);
    s.InsLast(10101);

    std::cout << "Исходный список: \n";
    s.print();
    s.DelFirst();
    s.print();
    s.DelList();
    s.InsFirst(3);
    s.InsFirst(4);
    s.InsLast(5);
    s.InsLast(7);
    s.print();
    s.DelLast();
    s.print();
    s.InsValue(366, 2);
    s.print();
    s.DelValue(2);
    s.print();

    if (s.search(3))
        std::cout << "\nsearch\n";
    else { std::cout << "\nNot find\n"; }


    list1.InsFirst(15);
    list1.InsFirst(9);
    list1.InsFirst(33);
    list1.InsFirst(66);

    list1.WriteFile(PATH);

    list2.ReadFile(PATH);
    list2.print();

    list2.InsFirst(1);
    list2.InsFirst(6);
    list2.InsFirst(7);
    list2.InsFirst(9);
    list2.InsFirst(2);
    list2.InsFirst(333);

    list1.print();

    list2.print();

    List find;
    find = *list2.findSpecialElements(3);
    find.print();

    return 0;
}
```