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 массивах

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 массивов.

Выполнение работы предполагает решение следующих задач:

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

Руководство пользователя

Пользователю нужно запустить файл ListTwoArrays.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 ListTwoArrays
- 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 List:

```
1. List(int _Mem = DEFAULT_MEM) – конструктор по умолчанию;
1. List(List<T>& _list) – конструктор с параметрами;
2. ~List() - деструктор;
3. int GetMem() – получить количество выделенной памяти;
4. int GetCount() – получить количество элементов в списке;
5. T GetData(int pos) – получить значение в списке по индексу;
6. bool isEmpty() – проверка на пустоту;
7. bool isFull() – проверка на переполненность;
8. void ins_Front(T Val) – вставка элемента в начало;
9. void ins_Back(T Val) – вставка элемента в конец;
10. void del_Front() – удалить элемент из начала;
11. void del_Back() - удалить элемент из конца списка;
12. void del_List() – удалить список;
13. void ins_Value(int Val, int pos) – вставить значение по индексу;
14. void del_Value(int pos) – удалить значение по индексу;
15. void print() – вывести элементы в консоль;
16. bool search(int data) – проверка есть ли переданный элемент в списке;
17. List<T>* findSpecialElements(int K) – получить список в котом каждый элемент
    делится нацело на число K;
18. void WriteFile(std::string path) – вывод списка в файл;
19. void ReadFile(std::string path) – чтение списка с файла;
```

Класс ListIterator:

```
1. ListIterator(List<T>& _list, int _index) - конструктор;
2. ~ListIterator() – деструктор;
3. bool CheckNext() – выделена ли память под следующее звено;
4. void GoNext() – сдвинуть указатель на 1 позицию вперёд;
5. ListIterator<T> operator++(int) – сдвинуть указатель на 1 позицию вперёд;
6. T GetData() – получить значение текущего указателя;
```

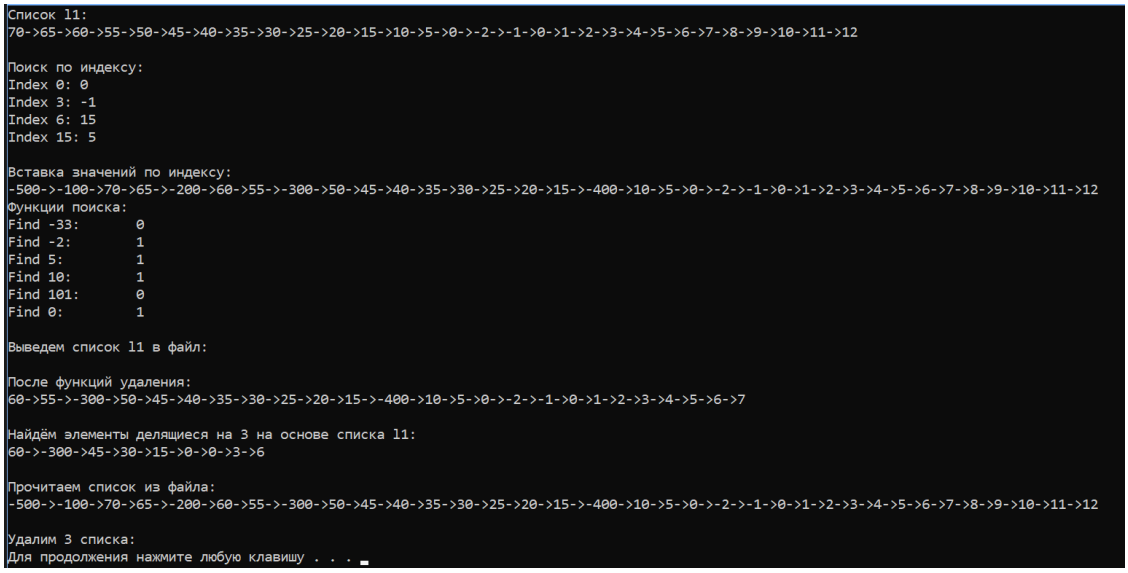
Описание алгоритмов

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 с использованием массивов. 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

List.h

```
#define DEFAULT_MEM 100

template<class T>
class List
{
private:
    int* pLinks;        // Указатели
    T*   Data;          // Значения
    int  Head;          // Указатель первый элемент
    int  Mem;           // Выделенная память
    int  Count;         // Кол-во элементов в списке

public:
    List(int _Mem = DEFAULT_MEM);
    List(List<T>& _list);
    ~List();

    int  GetMem();                      // Выделенная память
    int  GetCount();                    // Кол-во элементов в списке
    T    GetData(int pos);              // Вернуть значение в списке по позиции

    bool isEmpty();                     // Проверка на пустоту
    bool isFull();                      // Проверка на переполненность

    void ins_Front(T Val);              // Вставка в начало
    void ins_Back(T Val);               // Вставка в конец

    void del_Front();                   // Удалить в начале
    void del_Back();                    // Удалить в конце
    void del_List();                    // Удалить список

    void ins_Value(int Val, int pos);   // Вставить значение по позиции
    void del_Value(int pos);            // Удалить  значение по позиции

    void print();                       // Вывести элементы в консоль

    bool search(int data);              // Найти значение в списке по значению

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

    void WriteFile(std::string path);   // Запись в файл
    void ReadFile(std::string path);    // Чтение с файла

    friend class ListIterator<T>;       // Для доступа к полям класса
};
```

```cpp
template<class T>
List<T>::List(int _Mem)
{
    Mem = _Mem;
    Count = 0;
    pLinks = new int[_Mem];
    Data = new T[_Mem];
    for (int i = 0; i < _Mem; i++)
    {
        pLinks[i] = -5;
        Data[i] = 0;
    }
    Head = -1;
}

template<class T>
inline List<T>::List(List<T>& _list)
{
    Mem = _list.Mem;
    Count = _list.Count;
    pLinks = new int[Mem];
    Data = new int[Mem];
    for (int i = 0; i < _list.Mem; i++)
    {
        pLinks[i] = _list.pLinks[i];
        Data[i] = _list.Data[i];
    }
    Head = _list.Head;
}

template<class T>
inline List<T>::~List()
{
    delete[] pLinks;
    delete[] Data;
    Mem = 0;
}

template<class T>
inline int List<T>::GetMem()
{
    return this->Mem;
}

template<class T>
inline int List<T>::GetCount()
{
    return this->Count;
}

template<class T>
inline T List<T>::GetData(int pos)
{
    if (pos >= 0 && pos < Count)
    {
        return Data[pos];
    }
    else { throw std::logic_error("Going beyond the array boundaries!"); }
}
```

```cpp
template<class T>
inline bool List<T>::isEmpty()
{
    return Head == -1;
}

template<class T>
inline bool List<T>::isFull()
{
    return Count == Mem;
}

template<class T>
inline void List<T>::ins_Front(T Val)
{
    if (isFull())
        throw "List is Full";

    if (isEmpty())
    {
        Head = 0;
        this->Data[Head] = Val;
        pLinks[Head] = -1;
    }
    else
    {
        int index = 0;
        while (pLinks[index] != -5)
            index++;

        this->Data[index] = Val;
        pLinks[index] = Head;
        Head = index;
    }
    Count++;
}

template<class T>
inline void List<T>::ins_Back(T Val)
{
    if (isFull())
        throw "List is Full";

    if (isEmpty())
    {
        Head = 0;
        this->Data[Head] = Val;
        pLinks[Head] = -1;
    }
    else
    {
        int last = Head;
        for (int i = 0; i < Count; i++)
        {
            if (pLinks[last] != -1)
                last = pLinks[last];
            else
                break;
        }
        int current = 0;
        for (int i = 0; i < Mem; i++)
        {
            if (pLinks[i] == -5)
            {
                current = i;
                break;
            }
        }
        this->Data[current] = Val;
        pLinks[last] = current;
        pLinks[current] = -1;
    };

    Count++;
}
```

```cpp
template<class T>
inline void List<T>::del_Front()
{
    if (isEmpty())
        throw "List is empty";

    if (pLinks[Head] == -1)
    {
        pLinks[Head] = -5;
        Head = -1;
    }
    else
    {
        int first = pLinks[Head];
        pLinks[Head] = -5;
        Head = first;
    }
    Count--;
}

template<class T>
inline void List<T>::del_Back()
{
    if (isEmpty())
        throw "List is empty";

    if (pLinks[Head] == -1)
    {
        pLinks[Head] = -5;
        Head = -1;
    }
    else
    {
        int last = Head;
        int prev = 0;
        for (int i = 0; i < Count; i++)
            if (pLinks[last] != -1)
            {
                prev = last;
                last = pLinks[last];
            }
            else
            {
                pLinks[last] = -5;
                pLinks[prev] = -1;
            }
    }
    Count--;
}

template<class T>
inline void List<T>::del_List()
{
    for (int i = 0; i < Count; i++)
    {
        pLinks[i] = -5;
        Data[i] = 0;
    }
    Head = -1;
    Count = 0;
}
```

```cpp
template<class T>
inline void List<T>::ins_Value(int Val, int pos)
{
    if (isFull())
        throw "List is full";

    if (pos < 0 || pos > Count)
        throw "Going beyond the array boundaries!";

    if (pos == 0)
        ins_Front(Val);
    else
    {
        int current = 0;
        while (pLinks[current] != -5)
            current++;

        int previous = Head;
        int counter = 0;

        while (counter != pos - 1)
        {
            if (pLinks[previous] == -1)
                break;
            previous = pLinks[previous];
            counter++;
        }

        this->Data[current] = Val;
        pLinks[current] = pLinks[previous];
        pLinks[previous] = current;
        Count++;
    }
}

template<class T>
inline void List<T>::del_Value(int pos)
{
    if (pos < 0 || pos >= Count)
        throw "Going beyond the array boundaries!";

    if (pos == 0)
    {
        del_Front();
    }
    else
    {
        int del = Head;
        int prev = 0;

        int i = 0;
        while (i != pos)
        {
            if (i == pos - 1)
                prev = del;
            del = pLinks[del];
            i++;
        }

        pLinks[prev] = pLinks[del];
        pLinks[del] = -5;
        Count--;
    }
}


template<class T>
inline void List<T>::print()
{
    if (isEmpty())
        throw "List is empty";

    ListIterator<int> it(*this, Head);
    while(it.CheckNext())
    {
        std::cout << it.GetData() << "->";
        it++;
    }
    std::cout << it.GetData();
}
```

```cpp
template<class T>
inline bool List<T>::search(int data)
{
    if (isEmpty())
        throw "List is empty";

    ListIterator<int> it(*this, Head);
    while (it.CheckNext())
    {
        if (it.GetData() == data)
            return true;
        it++;
    }

    return false;
}

template<class T>
inline List<T>* List<T>::findSpecialElements(int K)
{
    List<T>* res = new List<T>(Count);

    ListIterator<int> it(*this, Head);
    while (it.CheckNext())
    {
        if (it.GetData() % K == 0)
        {
            res->ins_Back(it.GetData());
        }
        it++;
    }

    return res;
}

template<class T>
inline void List<T>::WriteFile(std::string path)
{
    if (isEmpty())
        throw "List is empty";

    std::ofstream file(path, std::ios::trunc);
    if (file.is_open())
    {
        ListIterator<int> it(*this, Head);
        while (it.CheckNext())
        {
            file << it.GetData() << "->";
            it++;
        }
        file << it.GetData();
    }
    else { std::cout << "FILE IS NOT OPEN!\n"; }
    file.close();
}

template<class T>
inline void List<T>::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());
                    ins_Back(val[i_val]);
                    i_val++;
                    str.clear();
                    i++;
                    continue;
                }
                str += buf[i];
            }
        }
        ins_Back(std::atoi(str.c_str()));
    }
    else{ std::cout << "FILE IS NOT OPEN!\n"; }
    file.close();
}
```

Приложение 2

ListIterator.h

```h
template<class T>
class List;

template<class T>
class ListIterator
{
private:
    List<T>& list;
    int index;
public:
    ListIterator(List<T>& _list, int _index) : list(_list), index(_index) {}
    ~ListIterator() {}

    bool CheckNext()
    {
        return (list.pLinks[index] != -1);
    }
    void GoNext()
    {
        if (!CheckNext())
            throw "List is end!";
        index = list.pLinks[index];
    }

    ListIterator<T> operator++(int)
    {
        GoNext();
        return (*this);
    }

    T GetData()
    {
        return list.Data[index];
    }
};
```

Приложение 3

Main.cpp

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

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

    List<int> l1;

    for (int i = 0; i < 15; i++)
    {
        l1.ins_Front(i * 5);
        l1.ins_Back(i - 2);
    }

    std::cout << "Список l1: \n";
    l1.print();
    std::cout << "\n\n";

    std::cout << "Поиск по индексу: \n";
    std::cout << "Index 0: "  << l1.GetData(0)  << "\n";
    std::cout << "Index 3: "  << l1.GetData(3)  << "\n";
    std::cout << "Index 6: "  << l1.GetData(6)  << "\n";
    std::cout << "Index 15: " << l1.GetData(15) << "\n";
    std::cout << "\n";

    std::cout << "Вставка значений по индексу: \n";
    l1.ins_Value(-100, 0);
    l1.ins_Value(-200, 3);
    l1.ins_Value(-300, 6);
    l1.ins_Value(-400, 15);
    l1.ins_Value(-500, 0);
    l1.print();
    std::cout << "\n";

    std::cout << "Функции поиска: \n";
    std::cout << "Find -33: " << "\t" << l1.search(-33) << "\n";
    std::cout << "Find -2: "  << "\t" << l1.search(2)   << "\n";
    std::cout << "Find 5: "   << "\t" << l1.search(5)   << "\n";
    std::cout << "Find 10: "  << "\t" << l1.search(10)  << "\n";
    std::cout << "Find 101: " << "\t" << l1.search(101) << "\n";
    std::cout << "Find 0: "   << "\t" << l1.search(0)   << "\n";
    std::cout << "\n";

    std::cout << "Выведем список l1 в файл: \n";
    l1.WriteFile(PATH);
    std::cout << "\n";

    std::cout << "После функций удаления: \n";
    l1.del_Back();
    l1.del_Back();
    l1.del_Back();
    l1.del_Back();
    l1.del_Back();
    l1.del_Front();
    l1.del_Front();
    l1.del_Front();
    l1.del_Front();
    l1.del_Front();
    l1.print();
    std::cout << "\n\n";

    std::cout << "Найдём элементы делящиеся на 3 на основе списка l1: \n";

    List<int> l2;
    l2 = *l1.findSpecialElements(3);
    l2.print();
    std::cout << "\n\n";

    std::cout << "Прочитаем список из файла: \n";
    List<int> l3;
    l3.ReadFile(PATH);
    l3.print();
    std::cout << "\n\n";

    std::cout << "Удалим 3 списка: \n";
    l1.del_List();
    l2.del_List();
    l3.del_List();

    system("pause");
    return 0;
}
```