个人资料

Microsues

访问： 80522次
积分： 1205
等级： BLOG 4
排名： 千里之外

原创： 35篇 转载： 19篇
译文： 0篇 评论： 28条

文章搜索

文章分类

C/C++ (22)
CSAPP读书笔记 (7)
MFC (2)
winsock编程 (5)
数据结构 (12)
杂记 (2)
算法 (2)
C#高级编程读书笔记 (2)
C++primer复习笔记 (1)

文章存档

2011年07月 (9)
2011年06月 (9)
2011年05月 (2)
2011年03月 (1)
2011年01月 (8)

展开

阅读排行

链表——用C++模板实现 (8138)
2011年.NET面试题总结--基 (4746)
使用jrtplib写的一个简单服 (3924)
struct/class/union内存对齐原 (3656)
在VC9.0中实现C++模板类 (3617)

链表——用C++模板实现

标签： c++ null insert string list 测试

2010-12-29 20:02 8145人阅读 评论(2) 收藏 举报

分类： 数据结构（11）

LinkedList.h

```
[cpp] view plain copy print ?
01. //单向链表的实现
02. //查找、插入和移除某个元素、复制、清空单向链表的时间复杂度均为O(n)
03. //设计类时，应尽量减少类的成员函数之间的依赖关系
04. #ifndef LINKEDLIST_H
05. #define LINKEDLIST_H
06. template<class DT>
07. struct Node
08. {
09.     DT info;
10.     Node<DT> *next;
11. };
12. template<class DT>
13. class LinkedList
14. {
15. public:
16.     LinkedList();
17.     LinkedList(const LinkedList<DT> &aplist);
18.     ~LinkedList();
19.     LinkedList<DT>& operator=(const LinkedList<DT> &rlist);
20.     void insert(const DT &element);  //在链表的头部之前插入
21.     void insert_end(const DT &element);  //在链表的尾部插入
22.     bool first(DT &listEl);    //得到对头的数据
23.     inline bool getNext(DT &listEl);  //得到当前指针所指的下一个数据
24.     bool find(const DT &element);    //查找一个数据
25.     bool retrieve(DT &element);      //检索一个数据
26.     bool replace(const DT &newElement);   //更改一个数据
27.     bool remove(DT &element);
28.     bool isEmpty() const;
29.     void makeEmpty();
30. private:
31.     Node<DT> *start;  //指向头结点
32.     Node<DT> *current;   //指向当前的结点
33.     inline void deepCopy(const LinkedList<DT> &original);  //"深复制"
34. };
35. #endif
```

LinkedList.cpp

```
[cpp] view plain copy print ?
01. #include "LinkedList.h"
02. template<class DT>
03. LinkedList<DT>::LinkedList()
04. {
```

```
    start = current = NULL;
}
template<class DT>
LinkedList<DT>::LinkedList(const LinkedList<DT> &aplist)
{
    deepCopy(aplist);
}
template<class DT>
LinkedList<DT>::~LinkedList()
{
    makeEmpty();
}
template<class DT>
LinkedList<DT>& LinkedList<DT>::operator =(const LinkedList<DT> &rlist)
{
    if(this == &rlist)
        return this;
    makeEmpty();
    deepCopy(rlist);
    return this;
}
//在链表的表头插入DT &element，插入后没有当前位置
template<class DT>
void LinkedList<DT>::insert(const DT &element)
{
    current = NULL;
    Node<DT> *NNode = new Node<DT>;
    NNode->info = element;
    NNode->next = start;
    start = NNode;
}
////在链表的尾部插入
template<class DT>
void LinkedList<DT>::insert_end(const DT &element)
{
    current = NULL;
    Node<DT> *NNode = new Node<DT>;
    NNode->info = element;
    NNode->next = NULL;
    Node<DT> *temp;
    DT item;
    if(start == NULL)
    {
        start = NNode;
        temp = start;
        return;
    }
    temp = start;
    while(temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = NNode;
}
//把链表中的第一个元素存放在listEl中，并且把第一个元素的位置置为current
template<class DT>
bool LinkedList<DT>::first(DT &listEl)
{
    if(start == NULL)
        return false;
    current = start;
    listEl = start->info;
    return true;
}
//把链表中current的下个元素放在listEl中，current指向下一个元素
template<class DT>
bool LinkedList<DT>::getNext(DT &listEl)
{
    if(current == NULL)
        return false;
    if(current->next == NULL)
    {
        current = NULL;
        return false;
    }
    listEl = current->next->info;
    current = current->next;
    return true;
}
```

```
84. //如果找到elment，返回true,current在getNext()中设置
85. template<class DT>
86. bool LinkedList<DT>::find(const DT &element)
87. {
88.     DT item;
89.     if(!first(item))
90.         return false;  //检查是否为空
91.     do{
92.         if(item == element)
93.             return true;
94.     }while(getNext(item));
95.     return false;
96. }
97. template<class DT>
98. bool LinkedList<DT>::retrieve(DT &element)
99. {
100.     if(!find(element))
101.         return false;
102.     element = current->info;
103.     return true;
104. }
105. template<class DT>
106. bool LinkedList<DT>::replace(const DT &newElement)
107. {
108.     if(current == NULL)
109.         return false;
110.     current->info = newElement;
111.     return true;
112. }
113. template<class DT>
114. bool LinkedList<DT>::remove(DT &element)
115. {
116.     current = NULL;
117.     if(start == NULL)
118.         return false;
119.     Node<DT> *ptr = start;
120.     if(start->info == element)
121.     {
122.         element = start->info;
123.         start = ptr->next;
124.         delete ptr;
125.         return true;
126.     }
127.     while(ptr->next != NULL){
128.         if(ptr->next->info == element)
129.         {
130.             Node<DT> *tempptr = ptr->next;
131.             element = tempptr->info;
132.             ptr->next = tempptr->next;
133.             delete tempptr;
134.             return true;
135.         }
136.         ptr = ptr->next;
137.     }
138.     return false;
139. }
140. template<class DT>
141. bool LinkedList<DT>::isEmpty() const
142. {
143.     return start == NULL;
144. }
145. template<class DT>
146. void LinkedList<DT>::makeEmpty()
147. {
148.     while(start != NULL)
149.     {
150.         current = start;
151.         start = start->next;
152.         delete current;
153.     }
154.     current = NULL;
155. }
156. template<class DT>
157. void LinkedList<DT>::deepCopy(const LinkedList<DT> &original)
158. {
159.     start = current = NULL;
160.     if(original.start == NULL)
161.         return;
162.     Node<DT> *copyptr = start = new Node<DT>;
```

```
    Node<DT> *originalptr = original.start;
    copyptr->info = originalptr->info;
    if(originalptr == original.current)
        current = copyptr;
    while(originalptr->next  != NULL)
    {
        copyptr->next = new Node<DT>;  //这一句有难度啊！
        originalptr = originalptr->next;
        copyptr = copyptr->next;
        copyptr->info = originalptr->info;
        if(originalptr == original.current)
            current = copyptr;
    }
    copyptr->next = NULL;
}
//#endif
```

测试用例：

```
[cpp] view plain  copy  print  ?
#include<iostream>
#include "LinkedList.cpp"
#include<string>
using namespace std;
void main()
{
    string str[5] = {"hello","world","welcome","to","beijing"};
    //初始化一个链表
    LinkedList<string> strLL;
    for(int ix=0;ix<5;++ix)
//      strLL.insert(str[ix]);          //调用insert()，依次插入在链表的头部
        strLL.insert_end(str[ix]);
    //遍历链表
    cout<<"遍历链表: ";
    string listEl;
    if(strLL.first(listEl))
        cout<<listEl<<" ";    //获得current
    while(strLL.getNext(listEl))
        cout<<listEl<<" ";
    cout<<endl;
    LinkedList<string> strLL2(strLL);  //调用构造函数初始化
    LinkedList<string> strLL3 = strLL;  //调用重载“=”操作符
    strLL2.first(listEl);
    cout<<"ListEl = "<<listEl<<endl;
    strLL3.first(listEl);
    cout<<"ListEl = "<<listEl<<endl;
    if(strLL2.find(str[3]))             //调用find()
        cout<<"str[3] = "<<str[3]<<endl;
    if(strLL3.retrieve(str[2]))         //调用retrieve()
        cout<<"str[2] = "<<str[2]<<endl;
    if(strLL.remove(str[1]))
    {
        strLL.first(listEl);
        cout<<"ListEl after remove is "<<listEl<<endl;
    }
    if(strLL.isEmpty())
        cout<<"List is Empty!"<<endl;
    strLL.makeEmpty();
    strLL2.makeEmpty();
    strLL3.makeEmpty();
    cout<<"hello"<<endl;
}
```