

Iterators

- 1 Double Linked and Circular Lists
node UML diagram
implementing a double linked list
the need for a deep copy
- 2 Iterators on List
nested classes for iterator
function objects

MCS 360 Lecture 12
Introduction to Data Structures
Jan Vershelde, 20 September 2010

Iterators

1 Double Linked and Circular Lists

node UML diagram

implementing a double linked list

the need for a deep copy

2 Iterators on List

nested classes for iterator

function objects

Linked Lists

A linked list is a sequence of nodes connected by pointers.

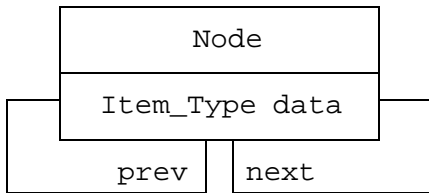
We distinguish four types:

- 1 single linked list: one pointer to next node
- 2 double linked list: pointer to next and previous
- 3 circular list: last next points to first node
- 4 ordered list: if data type admits order

The `list` in the Standard Template Library (STL) is double linked. The `slist` in the STL is a single linked list.

Unified Modeling Language

The node UML diagram of a double linked list:



The type of the `data` is templated,

- `next` points to the *next* node in the list,
- `prev` points to the *previous* node in the list.

doubly linking nodes

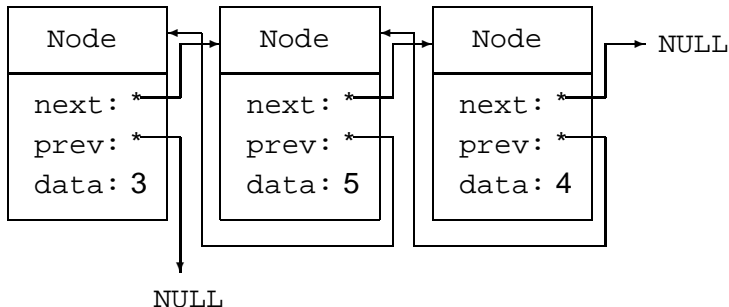
Double Linked and Circular Lists

node UML diagram
implementing a
double linked list
the need for a deep
copy

Iterators on List

nested classes for
iterator
function objects

A linked list to store 3, 5, 4 looks like:



At the expense of one extra pointer per node,
reverse transversal becomes efficient.

a circular list

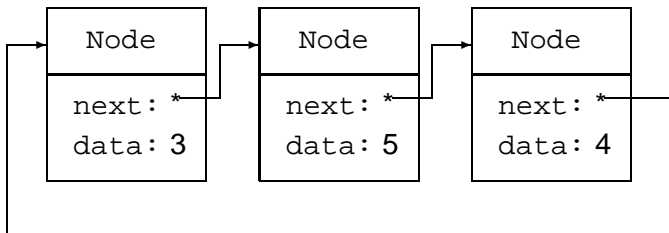
Double Linked
and Circular
Lists

node UML diagram
implementing a
double linked list
the need for a deep
copy

Iterators on
List

nested classes for
iterator
function objects

A circular list to store 3, 5, 4 looks like:



Traversing a circular list is like modular addition,
e.g.: n times $nd = nd \rightarrow next$ on a node nd
in a list of length n leads to the same node nd .

Iterators

1 Double Linked and Circular Lists

node UML diagram

implementing a double linked list

the need for a deep copy

2 Iterators on List

nested classes for iterator

function objects

file mcs360_double_list.h

Double Linked
and Circular
Lists

node UML diagram
implementing a
double linked list

the need for a deep
copy

Iterators on
List

nested classes for
iterator
function objects

```
#ifndef MCS360_DOUBLE_LIST_H
#define MCS360_DOUBLE_LIST_H
#define NULL 0

namespace mcs360_double_list
{
    template <typename T>
    class List
    {
        private:
            #include "mcs360_double_node.h"
            Node *first; // points to first node
            Node *last; // points to last node

        public: // operations omitted
    };
}

#include "mcs360_double_list.tc"
#endif
```

file mcs360_double_node.h

Double Linked
and Circular
Lists

node UML diagram
implementing a
double linked list

the need for a deep
copy

Iterators on
List

nested classes for
iterator
function objects

```
#ifndef DNODE_H
#define DNODE_H

struct Node
{
    T data; // T is template parameter
    Node *next; // pointer to next node
    Node *prev; // pointer to previous node

    Node(const T& item, Node* next_ptr = NULL,
          Node* prev_ptr = NULL) :
        data(item), next(next_ptr),
        prev(prev_ptr) {}
};
#endif
```

The struct of C is a class where all is public.

public methods

In file `mcs360_double_list.h`:

```
public:
```

```
List();  
List(T item);  
void append(T item);  
void write_forward();  
void write_backward();
```

These methods form a very basic, first implementation.

constructors

in the file `mcs360_double_list.tc`

```
namespace mcs360_double_list
{
    template <typename T>
    List<T>::List()
    {
        first = NULL;
        last = NULL;
    }

    template <typename T>
    List<T>::List(T item)
    {
        first = new Node(item);
        last = first;
    }
}
```

a double linked list

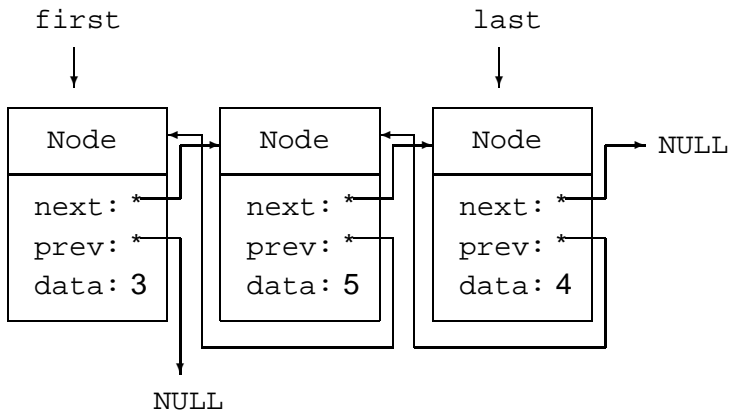
Double Linked
and Circular
Lists

node UML diagram

implementing a
double linked listthe need for a deep
copyIterators on
Listnested classes for
iterator

function objects

Appending to a list that stores 3, 5, 4:



the append() method

Double Linked
and Circular
Lists

node UML diagram

implementing a
double linked listthe need for a deep
copyIterators on
Listnested classes for
iterator

function objects

```
template <typename T>
void List<T>::append(T item) {
    if(first == NULL) {
        first = new Node(item);
        last = first;
    }
    else if(first == last) {
        first->next = new Node(item);
        first->next->prev = first;
        last = first->next;
    } else {
        last->next = new Node(item);
        last->next->prev = last;
        last = last->next;
    }
}
```

traversing forward and backward

Double Linked and Circular Lists

node UML diagram

implementing a
double linked list

the need for a deep
copy

Iterators on List

nested classes for
iterator

function objects

```
template <typename T>
void List<T>::write_forward()
{
    Node *ptr = first;
    while(ptr != NULL) {
        std::cout << "->" << ptr->data;
        ptr = ptr->next;
    }
}

template <typename T>
void List<T>::write_backward()
{
    Node *ptr = last;
    while(ptr != NULL) {
        std::cout << "->" << ptr->data;
        ptr = ptr->prev;
    }
}
```

Iterators

1 Double Linked and Circular Lists

node UML diagram

implementing a double linked list

the need for a deep copy

2 Iterators on List

nested classes for iterator

function objects

testing the class

Double Linked
and Circular
Lists

node UML diagram
implementing a
double linked list
the need for a deep
copy

Iterators on
List

nested classes for
iterator
function objects

```
#include "mcs360_double_list.h"
#include <iostream>
using namespace std;
using namespace mcs360_double_list;

int main()
{
    List<int> L(3);

    L.append(5);
    L.append(4);
    cout << " forward write :";
    L.write_forward();
    cout << endl;
    cout << "backward write :";
    L.write_backward();
    cout << endl;
}
```

the need for a deep copy

```
List<int> K = L;  
L.append(1);  
cout << " writing K after K = L :";  
K.write_forward(); cout << endl;  
cout << " writing L after K = L :";  
L.write_forward(); cout << endl;
```

The assignment `K = L` is the shallow copy of pointers
`first` and `last` \Rightarrow as `L` changes, so does `K`.

Iterators

1 Double Linked and Circular Lists

node UML diagram

implementing a double linked list

the need for a deep copy

2 Iterators on List

nested classes for iterator

function objects

iterators on list

Including `write_forward` and `write_backward` in definition of our class is not proper.

In our test program we would like to use

```
void write_with_iterator ( List<int> L )
{
    for(List<int>::Iterator item = L.begin();
        item != L.end(); ++item)
        cout << "->" << *item;
    cout << endl;
}
```

nested classes

With templates everything must be included in the header file, here `mcs360_double_list.h`.

```
#include <stdexcept>

namespace mcs360_double_list
{
    // omitted private part
    public: // omitted previous public

    #include "mcs360_list_iterator.h"
    friend class Iterator;
```

The class `Iterator` must have access to the private part of `List`, therefore: `friend`.

the class Iterator

```
#ifndef MCS360_LIST_ITERATOR_H
#define MCS360_LIST_ITERATOR_H

class Iterator
{
    friend class List<T>;
private:

    List<T> *parent; /* refers to the list */

    typename List<T>::Node *current;
    /* pointer to the current node in parent */

    Iterator( List<T> *L, Node *n ) :
        parent(L), current(n) {}
};
```

the public part

```
T& operator*() const // dereferencing operator
{
    if(current == NULL)
        throw std::invalid_argument
            ("cannot dereference past end()");
    return current->data;
}

Iterator& operator++() // prefix increment operator
{
    if(current == NULL)
        throw std::invalid_argument
            ("cannot go past end()");
    current = current->next;
}

bool operator!=(const Iterator &other)
{
    return current != other.current;
}
```

begin() and end()

At the end of `mcs360_list_iterator.h`, we put:

```
Iterator begin()  
{  
    Iterator r(this,this->first);  
    return r;  
}
```

```
Iterator end()  
{  
    Iterator r(this,NULL);  
    return r;  
}
```

postfix increment operator

Double Linked and Circular Lists

node UML diagram
implementing a
double linked list
the need for a deep
copy

Iterators on List

nested classes for
iterator
function objects

The postfix increment operator first makes a copy of the current value of the iterator:

```
Iterator operator++(int)
// postfix increment operator
{
    Iterator return_value = *this;
    ++(*this);
    return return_value;
}
```

Observe the prefix increment operator:

```
Iterator& operator++()
```

Iterators

1 Double Linked and Circular Lists

node UML diagram
implementing a double linked list
the need for a deep copy

2 Iterators on List

nested classes for iterator
function objects

function objects

A frequent recurring application on sequences is to select items that match a criterion.

Example: list all even numbers in a list.

To define the criterion we overload the function call operator, the `operator()`.

A class that overloads `()` is called a *function class* and an object of this class is a *function object*.

class Divisible_By

```
class Divisible_By
{
    private:

        int divisor;

    public:

        Divisible_By(int d) : divisor(d){}
        bool operator()(int x)
        {
            return x % divisor == 0;
        }
};
```

the `find_if` method

The `find_if` method of the STL list class finds the first occurrence of an item that satisfies a criterion.

```
template < typename T, typename P >  
    T find_if( T first, T last, P pred );
```

If there is no item for which `pred` returns true, then `last` is returned.

listing the even numbers

```
int main()
{
    list<int> L;
    const int n = 20;

    srand(time(0));
    for(int i=0; i<n; i++)
        L.push_back(rand() % 100);

    cout << "L : ";
    write(L);
    cout << "the even numbers of L : "
         << endl;
    even_numbers(L); cout << endl;

    return 0;
}
```

applying find_if

```
void even_numbers ( list<int> L )
{
    list<int>::iterator i = L.begin();

    while(i != L.end())
    {
        list<int>::iterator j;

        j = find_if(i,L.end(),Divisible_By(2));

        if(j == L.end()) break;

        cout << " " << *j;

        while(i != j) i++;
        i++;
    }
}
```

Summary + Assignments

Ended Chapter 4 on *Sequential Containers*.

Assignments:

- 1 Define a `copy` operation on a double linked list. Show with a test program that the copy is deep: the original list does not change if the copy changes.
- 2 Write code for to insert `item1` in a double linked list before `item2`. If `item2` does not occur in the list, then `item1` is appended to the list.
- 3 Define a prefix decrement operator on the iterator. Write a program to write a list in reverse order.
- 4 Use the STL list and vector classes and give code to convert a list into a vector and vice versa.

Lab tomorrow in EPASWL270 (*not* in SEL 2263)!