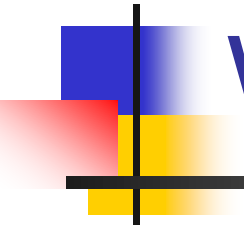
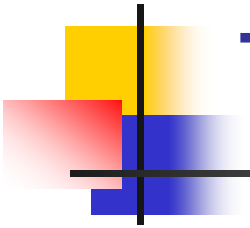


Week 6



Topics

- Nodes of a list.
- A List of nodes.
- List iterators.
- Miscellaneous

List Nodes Illustration (1)

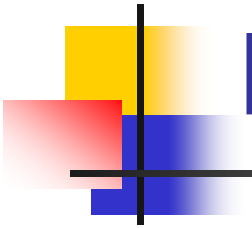
```
$ cat nodes.cpp
#include <iostream>
#include <string>
using namespace std;
template <typename T>
class Node {
        T data;
        Node<T> *next;
public:
        Node(const T &D, Node<T> *N): data(D), next(N) {}
        void show() {
                cout << data;
                if(next) { cout << ' '; next->show(); }
        }
};
```

List Nodes Illustration (2)

```
int main(int ac, char *av[]) {
    Node<string> *n = 0;
    int i;
    for(i=0; av[i]; i++) {
        n = new Node<string>( string(av[i]), n );
    }
    n->show();
    cout << endl;
    return 0;
}
```

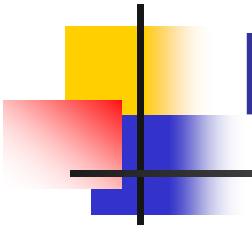
```
$ g++ nodes.cpp
$ a.out 1 22 333 44444 55555 cs1280
cs1280 55555 44444 333 22 1 a.out
```

The Problem of the Previous List Node Implementation



- The above implementation supports only insertion at the head
- The list user must maintain a pointer to point to the first node of the list

Separation of a List and Its Nodes



- It is better to model a list with two objects: the list container and the nodes.
- Only the list container will know where is the first node
- Desired operations such as add new nodes or delete nodes from a list will be member functions of the list.

List Illustration: list.cpp (1)

```
#include <iostream>
#include <string>
using namespace std;
template <typename T> class List;        // forward declaration
template <typename T>
class Node {
    T data;
    Node<T> *next;
friend class List<T>;
public:
    Node(const T & D, Node<T> * N): data(D), next(N) {}
};
```

List Illustration: list.cpp (2)

```
template <typename T>
class List {
    Node<T> *first;
public:
    List() : first(0) {}
    ~List() {
        Node<T> *q;
        for(Node<T> *p=first; p;) {
            q = p;
            p = p->next;
            delete q;
        }
    }
```

List Illustration: list.cpp (3)

```
void pushH(const T & D) {
    first = new Node<T>( D, first );
}
bool popH(T & D) {
    if(!first) return false;
    D = first->data;
    Node<T> * q = first;
    first = first->next;
    delete q;
    return true;
}
```

List Illustration: list.cpp (4)

```
void pushT(const T & D) {
    if(!first) pushH(D);
    Node<T> * q = first;
    while(q->next) q = q->next;
    q->next = new Node<T>(D, 0);
}
```

List Illustration: list.cpp (5)

```
bool popT(T & D) {
    if(!first) return false;
    Node<T> *p = first, *q = p->next;
    while(q && q->next) { p = q; q = q->next; }
    if(!q) return popH(D);
    p->next = 0;
    D = q->data;
    delete q;
    return true;
}
```

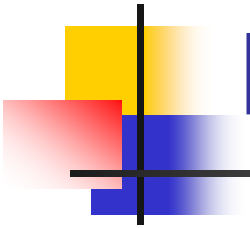
List Illustration: list.cpp (6)

```
    void show() {
        Node<T> *p = first;
        if(p) cout << p->data;
        if(p) p = p->next;
        while(p) {
            cout << p->data << endl ;
            p = p->next;
        }
    }
};
```

List Illustration: list.cpp (7)

```
int main() {
    List<string> abba;
    abba.pushH( string("Bjorn") );
    abba.pushT( string("Benny") );
    abba.pushH( string("Agnetha") );
    abba.pushT( string("Anni-Frid") );
    abba.show(); cout << “***” <<  endl;
    string who;
    if(abba.popT(who)) cout << who << ’\n’;
    if(abba.popH(who)) cout << who << ’\n’;
    if(abba.popH(who)) cout << who << ’\n’;
    if(abba.popT(who)) cout << who << ’\n’;
    return 0;
}
```

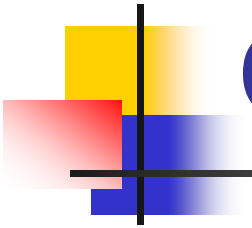
```
$g++ list.cpp
$ a.out
Agnetha
Bjorn
Benny
Anni-Frid
***
Anni-Frid
Agnetha
Bjorn
Benny
```



List Iterators

- A list is a container.
- It contains nodes, which are objects on their own.
- For a container object like a list, it is desirable to be able to traverse its objects with an iterator.

The Five Basic List Iterator Operations



- Reset to some initial position.
- Get the value at its current position.
- Set the value at its current position.
- Confirm validity of current position.
- Advance to the next position.

Iterator Illustration iter.cpp (1)

```
#include <iostream>
#include <string>
using namespace std;
template <typename T> class List; // Forward declarations
template <typename T> class ListIter;
template <typename T> class Node {
    T data;
    Node<T> *next;
friend class List<T>;
friend class ListIter<T>;
public:
    Node(const T & D, Node<T> * N): data(D), next(N) {}
};
```

Iterator Illustration iter.cpp (2)

```
template <typename T> class List {
    Node<T> *first;
friend class ListIter<T>;
public:
    List(): first(0) {}
    ~List() {
        Node<T> *q;
        for(Node<T> *p=first; p;) {
            q = p;
            p = p->next;
            delete q;
        }
    }
```

Iterator Illustration iter.cpp (3)

```
void pushH(const T & D) {
      first = new Node<T>( D, first );
}
bool popH(T & D) {
      if(!first) return false;
      D = first->data;
      Node<T> * q = first;
      first = first->next;
      delete q;
      return true;
}
```

Iterator Illustration iter.cpp (4)

```
void pushT(const T & D) {
    if(!first) pushH(D);
    Node<T> * q = first;
    while(q->next) q = q->next;
    q->next = new Node<T>(D, 0);
}
```

Iterator Illustration iter.cpp (5)

```
bool popT(T & D) {
    if(!first) return false;
    Node<T> *p = first, *q = p->next;
    while(q && q->next) { p = q; q = q->next; }
    if(!q) return popH(D);
    p->next = 0;
    D = q->data;
    delete q;
    return true;
}
```

Iterator Illustration iter.cpp (6)

```
    void show() {
        Node<T> *p = first;
        if(p) cout << p->data;
        p = p->next;
        while(p) {
            cout << p->data << endl ;
            p = p->next;
        }
    }
};
```

Iterator Illustration iter.cpp (7)

```
template <typename T> class ListIter {
    List<T> &list;
    Node<T> *current, *previous;
public:
    ListIter(List<T> & L): list(L) {}
    void operator & () {previous = 0; current = list.first;} //~begin
    T operator () () { return current->data; }     // dereference op
    void operator = (T D) { current->data = D; }
    bool operator ! () { return current != 0; }
    void operator ++ () {
        previous = current;
        current = current->next;
    }
};
```

Iterator Illustration iter.cpp (8)

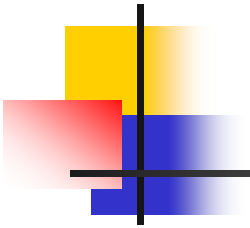
```
int main() {
    List<string> abba;
    abba.pushH( string("Bjorn") );
    abba.pushT( string("Benny") );
    abba.pushH( string("Agnetha") );
    abba.pushT( string("Anni-Frid") );
    abba.show();
    cout << “***” << endl;
    string who;
    if(abba.popT(who)) cout << who << endl;
    if(abba.popH(who)) cout << who << endl;
    if(abba.popH(who)) cout << who << endl;
    if(abba.popT(who)) cout << who << endl;
```

Iterator Illustration iter.cpp (9)

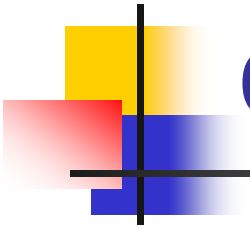
```
    ListIter<string> it(abba);
    abba.pushH( string("Bjorn") );
    abba.pushT( string("Benny") );
    abba.pushH( string("Agnetha") );
    abba.pushT( string("Anni-Frid") );
    cout << "\ntraveral by iterator\n\n";
    for( &it; !it; ++it ) {
        it = it() + "*";
        cout << it() << endl;
    }
    return 0;
}
```

```
Agnetha
Bjorn
Benny
Anni-Frid
***
Anni-Frid
Agnetha
Bjorn
Benny
traveral by iterator
Agnetha*
Bjorn*
Benny*
Anni-Frid*
```

Enforcing the Required Iterator Operations

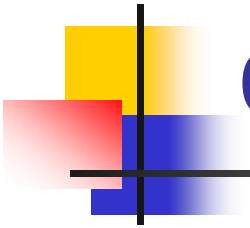


- To enforce the implementation of the five basic operations with the given interface, the ListIter class can be made a derived class of an abstract class BaseIter.



class BaseIter

```
template <typename T>
class BaseIter {
public:
    virtual void operator & () = 0;
    virtual T operator () () = 0;
    virtual void operator = (T) = 0;
    virtual bool operator ! () = 0;
    virtual void operator ++ () = 0;
};
```



class ListIter

```
template <typename T>
class ListIter : public BaseIter<T> {
    // implemented as the previous version without using BaseIter
};
```

Environment Variables(1)

- The main() functions can be called in four ways:

```
int main() { /* */ }
int main(int ac) { /* */ }
int main(int ac, char *av[]) { /* */ }
int main(int ac, char *av[], char *ev[] ) { /* */ }
```

- Both the character arrays av[] and ev[] are null-terminated.
- In particular, av[ac] is null.
- Under unix, ev[] contains the environment variables.

Environment Variables(2)

```
$ cat main.cpp
#include <iostream>
using namespace std;
int main(int ac, char *av[], char *ev[] ) {
    int c = -1;
    while( ev[++c] ) cout << ev[c] << endl;
    cout << c << " environment variables\n";
    return 0;
}
```

Environment Variables(3)

```
$ g++ main.cpp
$ a.out
MANPATH=/opt/SUNWspro/man:/usr/dt/man:/usr/local/man:/usr/local/teTeX/man:/usr/man:/usr/openwin/share/man:/usr/local/X11/man:/opt/sfw/man:/usr/sfw/man:/usr/local/share/man:/usr/local/man
TERM=xterm
SHELL=/bin/bash
...
33 environment variables
```

Which Constructor? (1)

```
#include <iostream>
using namespace std;
class C {
    public:
    C(char c){ cout << "char\t" << c << endl; }
    C(int i=1280){ cout << "int\t" << i << endl; }
    C(double d){ cout << "double\t" << d << endl; }
};
```

Which Constructor? (2)

```
int main() {
    C x;
    C c = 'a';      // construction, not assignment
    C i = 0; C d = 0.0;   // named objects
    C('b'); C(1); C(1.0); // anonymous objects
    return 0;
}
```

```
$ a.out
int 1280
char a
int 0
double 0
char b
int 1
double 1
```

Parameter Passing by Value

```
$ cat byv.cpp
#include <iostream>
using namespace std;
void setZ(int x) {
      cout << x << endl; x = 0; cout << x << endl;
}
int main() {
      setZ(1);
      int x = 2;
      setZ(x);
      cout << x << endl;
      return 0;
}
```

```
$ g++ byv.cpp
$ a.out
1
0
2
0
2
```

Parameter Passing by Reference

```
$ cat byr.cpp
#include <iostream>
using namespace std;
void setZ(int & x) {
      cout << x << endl; x = 0; cout << x << endl;
}
int main() {
      // setZ(1); compilation error
      int x = 2;
      setZ(x);
      cout << x << endl;
      return 0;
}
```

```
$ g++ byr.cpp
$ a.out
2
0
0
```

Parameter Passing by Pointer

```
$ cat byp.cpp
#include <iostream>
using namespace std;
void setZ(int * x) {
      cout << *x << endl; *x = 0; cout << *x << endl;
}
int main() {
      // setZ(1); compilation error
      int x = 2;
      setZ(&x);
      cout << x << endl;
      return 0;
}
```

```
$ g++ byp.cpp
$ a.out
2
0
0
```

Return by Reference (1)

```
$ cat ret.cpp
#include <iostream>
using namespace std;
class C {
public:
    int x;
    C(int X=1) :x(X) {}
    void show() { cout << x << endl; }
};
```

Return by Reference (2)

```
C G(1280);
C & f() { C x(12345); return x; }          // return a local object
C & g() { return C(67890); } // return an anonymous local object
C & h() { return G; }                      // return a global object
C & i() { return *new C(999); } // return an anonymous object
int main() {
    f().show();
    g().show();
    h() = 1281; G.show();
    i().show();
    return 0;
}
```

Return by Reference (3)

$ g++ ret.cpp

ret.cpp: In function ‘C& f()’:

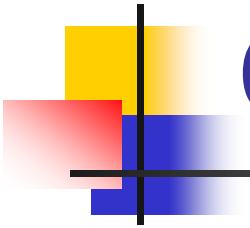
ret.cpp:11: warning: reference to local variable ‘x’ returned

ret.cpp: In function ‘C& g()’:

ret.cpp:12: error: invalid initialization of non-const reference of type ’C&’ from a temporary of type ’C’

Fix the above error (hint: which function’ return by reference is accepted by the compiler?) , we have:

```
$ a.out
12345
67890
1281
999
```



Cascading cout (1)

- The expression

  cout << expr;

  evaluates to

  cout;

- The operator << is left associative. Thus

  cout << expr1 << expr2;

  is equivalent to

  (cout << expr1) << expr2;

Cascading cout (2)

```
$cat cout.cpp
#include <iostream>
using namespace std;
int main() {
  int x=1, y=3;
  cout << x << y << endl;
  cout << (x << y) << endl;
  return 0;
}
```

```
$g++ cout.cpp
$a.out
13
8
```

cin.clear()

```
#include <iostream>
using namespace std;
int main() {
	int i;
	while(cin >>i) {// ctrl-d for eof on unix
		cout << i << endl;
	}
	cin.clear();
	while(cin >>i) cout << i << endl;
	return 0;
}
```

```
$ g++ clear.cpp
$ a.out
1
1
^D
2
2
```

What if cin.clear(); is commented off?