Ordered List ADT

Array-Based Implementation

- ✓ Inheritance
- ✓ OO Programming

Specification of the Generic Ordered List ADT

Ordered?

Ordered By Key

```
// DataType: data item
// KeyType: key field



class Account {
  public:
      int accountNum;                 // (Key) Account number
      float balance;                  // Account balance
      int getKey () const
          { return accountNum; }      // Returns the key
};
```

Design of the Generic Ordered List ADT via Inheritance: Class Diagram

Array-Based Representation

Implementation of the Generic Ordered List ADT

OrderedList.h
(OrderedList.cpp)
test4.cpp

Inherit from Generic List ADT

ListArray.h
(ListArray.cpp)

- ✓ Potential Inheritance = “is a”
- ✓ Virtual function in a base class
- ✓ Overridden (redefined) in derived classes
- ✓ Late (Dynamic) binding
- ✓ protected

```
template < typename DataType >
class List
{
  public:

    static const int MAX_LIST_SIZE = 10;   // Default maximum list size

    // Default constructor
    List ( int maxNumber = MAX_LIST_SIZE );

    // Copy constructor
    List ( const List& source );

    // Overloaded assignment operator
    const List& operator= ( const List& source );

    // Destructor
    virtual ~List ();
```

```
// List manipulation operations


// Insert after cursor
virtual void insert ( const DataType& newDataItem )
throw ( logic_error );

// Remove data item marked by the cursor
void remove () throw ( logic_error );

// Replace data item marked by the cursor
virtual void replace ( const DataType& newDataItem )
throw ( logic_error );

// Clear list
void clear ();
```

```
// List status operations

bool isEmpty () const;                  // List is empty

bool isFull () const;                   // List is full
```

```
// List iteration operations

void gotoBeginning ()                          // Go to beginning
    throw ( logic_error );

void gotoEnd ()                                // Go to end
    throw ( logic_error );

bool gotoNext ()                               // Go to next data item
    throw ( logic_error );

bool gotoPrior ()                              // Go to prior data item
    throw ( logic_error );

DataType getCursor () const
    throw ( logic_error );                     // Return data item
```

```
virtual void showStructure () const;


void moveToNth ( int n )          // Move data item to pos. n
    throw ( logic_error );

bool find ( const DataType& searchDataItem )     // Find data item
    throw ( logic_error );
```

```
  protected:
    int maxSize,
        size,                // Actual number of data item in the list
        cursor;              // Cursor array index
    DataType *dataItems;     // Array containing the list data item
};
```

```
template < typename DataType, typename KeyType >

// DataType : data item
// KeyType : key field

class OrderedList : public List<DataType>
{
  public:

    static const int DEFAULT_MAX_LIST_SIZE = 10;

    // Constructor
    OrderedList ( int maxNumber = DEFAULT_MAX_LIST_SIZE );
```

```
    // Modified (or new) list manipulation operations

    virtual void insert ( const DataType &newDataItem )
    throw ( logic_error );

    virtual void replace ( const DataType &newDataItem )
    throw ( logic_error );

    bool retrieve ( const KeyType& searchKey, DataType &searchDataItem );

    void showStructure () const;

private:

    bool binarySearch ( KeyType searchKey, int &index );

};
```

```
class Account {
  public:
      int accountNum;                 // (Key) Account number
      float balance;                  // Account balance
      int getKey () const
          { return accountNum; }  // Returns the key
};



int main()
{
    OrderedList<Account, int> accounts(20);  // List of accounts

    Account acct;                            // A single account
```

```
template < typename DataType, typename KeyType >
OrderedList<DataType,KeyType>:: OrderedList ( int maxNumber )

// Creates an empty list by calling the List ADT constructor.

  : List<DataType>(maxNumber)

{}
```

```
template < typename DataType, typename KeyType >
bool OrderedList<DataType,KeyType>:: binarySearch ( KeyType searchKey,
int &index )

// Binary search routine. Searches a list for the data item with
// key searchKey. If the data item is found, then returns true with
// index returning the array index of the entry containing searchKey.
// Otherwise, returns false with index returning the array index of the
// entry containing the largest key that is smaller than searchKey
// (or -1 if there is no such key).

{









}
```

```
template < typename DataType, typename KeyType >
bool OrderedList<DataType,KeyType>:: binarySearch ( KeyType searchKey,
int &index )

// Binary search routine. Searches a list for the data item with
// key searchKey. If the data item is found, then returns true with
// index returning the array index of the entry containing searchKey.
// Otherwise, returns false with index returning the array index of the
// entry containing the largest key that is smaller than searchKey
// (or -1 if there is no such key).

{
    int low  = 0,            // Low index of current search range
        high = size - 1;     // High index of current search range

    bool result;             // Result returned
```

```
while ( low <= high )
{
    index = ( low + high ) / 2;                    // Compute midpoint

    if ( searchKey < dataItems[index].getKey() )

       high = index - 1;                           // Search lower half

    else if ( searchKey > dataItems[index].getKey() )

       low = index + 1;                            // Search upper half

    else

       break;                                      // searchKey found
}
```

```
    if ( low <= high )

       result = true;        // searchKey found

    else
    {
       index = high;         // searchKey not found, adjust index
       result = false;
    }

    return result;
}
```

```
template < typename DataType, typename KeyType >
void OrderedList<DataType,KeyType>:: showStructure () const

// Outputs the keys of the data items in a list. If the list is
// empty, outputs "Empty list". This operation is intended for
// testing and debugging purposes only.

{
    if ( size == 0 )
       cout << "Empty list" << endl;
    else
    {
       cout << "size = " << size
            <<  "   cursor = " << cursor << endl;

       for ( int j = 0 ; j < maxSize ; ++j )
           cout << j << "\t";

       cout << endl;
```

```
for ( int j = 0 ; j < size ; ++j ) {

          if( j == cursor ) {
                cout << "[" << dataItems[j].getKey() << "]\t";

          } else {
                cout << dataItems[j].getKey() << "\t";
          }
       }
       cout << endl;
    }
}
```

Extra Implementation

- Programming Exercise 2 (p. 53)
  merge
- Programming Exercise 3 (p. 54)
  subset
- Flexible Insert
  insert2
- Binary search vs Linear search

```
template < typename DataType, typename KeyType >
void OrderedList<DataType,KeyType>:: merge ( const
OrderedList<DataType,KeyType>& fromL ) throw ( logic_error )

// Merges the data items in list fromL into a list. Assumes that
// none of the data items in fromL occur in L already. The merge
// is done in a single pass through both lists.

{









}
```

```
template < typename DataType, typename KeyType >
bool OrderedList<DataType,KeyType>:: subset ( const
OrderedList<DataType,KeyType> &subList )

// Returns true if every key in list subList also occurs in a list --
// that is, if the keys in subList are a subset of the keys in the list.
// Otherwise, returns false.

{









}
```

```
template < typename DataType, typename KeyType >
void OrderedList<DataType,KeyType>:: insert2 ( const DataType
&newDataItem )
// flexible insert

{









}
```

Application of the Generic Ordered List ADT

Programming Exercise1 (p. 52)

```
#include <iostream>
#include <fstream>
using namespace std;

#include "OrderedList.cpp"

// Reads a series of packets, formats them into a message, and
// outputs the message.
```

```
class Packet {
  public:
    static const int PACKET_SIZE = 6;  // Number of characters in a packet
                                       // including null ('\0') terminator,
                                       // but excluding the key (1st char in each line).

    int position;                      // (Key) Packet's position w/in message

    char body[PACKET_SIZE];            // Characters in the packet

    int getKey () const
        { return position; }           // Returns the key field
};
```

```
int main()
{
    ifstream messageFile ("message.dat");    // Message file
    OrderedList<Packet,int> message;         // Message
    Packet currPacket;                       // Message packet
    int j;                                   // Loop variable

    // Read in the packets



    // Output the message packet-by-packet.



    return 0;
}
```

message.dat

```
3rrect
4ly co
6d.  C
9ions!
1Packe
5mbine
7ongra
2ts co
8tulat
```

Analysis Exercise 1 (p. 55)