Static Array Class

Array Class Example

The Class Interface

The Class Interface

Array Constructors

Array Constructors

Copy Constructor

Assignment Operator

Insertion Functions

Deletion Functions

Search Functions

Miscellaneous Functions

Driver

©1995-2000 Barnette ND, McQuain WD

Array Example

This chapter presents a sample implementation of a static array, encapsulated in a C++ class.

The primary goals of this implementation are:

- to provide a reasonable set of member functions
- to protect against array bounds errors
- to provide proper deep copy operations

The implementation provides an encapsulation of a dynamically allocated, but fixed-size, array of `int`s. The implementation could easily be modified to handle a generic type, as was done with the sample linked list class covered earlier.

The implementation also includes an "extra" member function that will initialize the array to pseudo-random values using the standard library functions `srand()` and `rand()`. This is primarily useful for testing, and is included here primarily to provide an excuse to discuss the use of those functions.

Warning: the Array class given in this chapter is intended for instructional purposes. The given implementation may contain a number of flaws.

Caveat emptor.

©1995-2000 Barnette ND, McQuain WD

The Class Interface

```
#include <cstdlib>        // for NULL
#include <limits>         // for INT_MIN
#include <ctime>          // for time()

class ArrayClass {
private:
   int  Capacity;  // dimension of list array
   int  Usage;     // number of elements now in list
   int  Current;   // current location in list (unused)
   int* List;      // the list

   void ShiftTailUp(int Start);
   void ShiftTailDown(int Start);
   void Swap(int& First, int& Second);

public:
   // empty list of size zero
   ArrayClass();

   // empty list of size initCapacity
   ArrayClass(int initCapacity);

   // list of size initCapacity, each cell stores Value
   ArrayClass(int initCapacity, int Value);

   // copy constructor and assignment overload
   ArrayClass(const ArrayClass& Source);

   ArrayClass& operator=(const ArrayClass& Source);

   // fill list with random integers in range [0, Max)
   void Randomize(int Max);

   // retrieve array capacity (dimension)
   int  getCapacity() const;

   // retrieve number of elements in array
   int  getUsage() const;
                                  // . . . continues . . .
```


©1995-2000 Barnette ND, McQuain WD

The Class Interface

```
// . . . continued . . .

   // check if array is full
   bool isFull() const;

   // check if array is empty
   bool isEmpty() const;

   // insert newValue at tail of list
   bool InsertAtTail(int newValue);

   // insert newValue at index Idx
   bool InsertAtIndex(int newValue, int Idx);

   // delete element at index Idx (shift)
   bool DeleteAtIndex(int Idx);

   // delete all copies of Value from list
   bool DeleteValue(int Value);

   // retrieve value at index Idx
   int  Retrieve(int Idx) const;

   // find index (if any) at which Value occurs
   int  FindValue(int Value) const;

   // deallocate array and set Capacity to 0
   void Clear();

   // reverse order of elements in list
   void Reverse();

   // deallocate array
   ~ArrayClass();

};
```


©1995-2000 Barnette ND, McQuain WD

Array Constructors

```
//////////////////////////////////////////////////////
// Default constructor:  provides an empty list with no
// storage space.  Would become useful if the insert
// method were changed to create/extend the list as
// needed.
//
ArrayClass::ArrayClass() {

   Capacity =    0;
   Usage    =    0;
   Current  =    0;
   List     = NULL;
}


//////////////////////////////////////////////////////
// Provides a list of the specified size, initializing
// the entries to INT_MIN.
//
// Parameters:
//    initCapacity   the capacity of the list to be
//                   created
//

ArrayClass::ArrayClass(int initCapacity) {

   Capacity = initCapacity;
   Usage    =  0;
   Current  =  0;

   if (Capacity <= 0) {
      List = NULL;
      Capacity = 0;
   }
   else {
      List = new int[initCapacity];

      for (int Idx = 0; Idx < Capacity; Idx++)
         List[Idx] = INT_MIN;
   }
}
```


©1995-2000 Barnette ND, McQuain WD

Array Constructors

```
//////////////////////////////////////////////////////
// Provides a list of the specified size, initializing
// the entries to the specified value.
//
// Parameters:
//    initCapacity   the capacity of the list to be
//                   created
//    Value          value to be stored in each cell
//
ArrayClass::ArrayClass(int initCapacity, int Value) {

   Capacity = initCapacity;
   Usage    =  0;
   Current  =  0;

   if (Capacity <= 0) {
      List = NULL;
      Capacity = 0;
   }
   else {
      List = new int[initCapacity];

      for (int Idx = 0; Idx < Capacity; Idx++)
         List[Idx] = Value;
   }
}
}
```


©1995-2000 Barnette ND, McQuain WD

Copy Constructor

```
//////////////////////////////////////////////////////
// Provides a list that is a copy of the parameter.
//
// Parameters:
//    oldArray   ArrayClass object to be copied
//
ArrayClass::ArrayClass(const ArrayClass& Source) {

   Capacity = oldArray.getCapacity();
   Usage    = oldArray.getUsage();

   if (Capacity <= 0) {
      List = NULL;
      Capacity = 0;
   }
   else {
      List = new int[Capacity];

      for (int Idx = 0; Idx < Capacity; Idx++)
         List[Idx] = oldArray.Retrieve(Idx);
   }
}
```


©1995-2000 Barnette ND, McQuain WD

Assignment Operator

```
//////////////////////////////////////////////////////
// Provides a list that is a copy of the parameter.
//
// Parameters:
//    Source   ArrayClass object to be copied
//
ArrayClass& ArrayClass::operator=(const ArrayClass&
                                        Source) {

   if (this != &Source) {

      Clear();

      Capacity = Source.getCapacity();
      Usage    = Source.getUsage();
      Current  = Source.Current;

      if (Capacity <= 0) {
         List = NULL;
         Capacity = 0;
      }
      else {
         List = new int[Capacity];

         for (int Idx = 0; Idx < Capacity; Idx++) {
            List[Idx] = Source.Retrieve(Idx);
         }
      }
   }
   return (*this);
}
```


©1995-2000 Barnette ND, McQuain WD

Insertion Functions

```
//////////////////////////////////////////////////////
// Inserts the given value at the tail of the list, if
// there is space.
//
// Parameters
//     newValue   value to be inserted
// Returns
//     true       if element insertion is successful
//     false      otherwise
//
bool ArrayClass::InsertAtTail(int newValue) {

   if ( isFull() ) return false;

   List[Usage] = newValue;
   Usage++;
   return true;
}

//////////////////////////////////////////////////////
// Inserts the given value at the given list index, if
// there is space.
//
// Parameters
//     newValue   value to be inserted
//     Idx        index at which element is to be
//                inserted
// Returns
//     true       if element insertion is successful
//     false      otherwise
//
bool ArrayClass::InsertAtIndex(int newValue, int Idx) {

   if (Usage == Capacity) return false;
   if (Idx > Usage || Idx < 0) return false;

   ShiftTailUp(Idx);
   List[Idx] = newValue;
   Usage++;
   return true;
}
```


©1995-2000 Barnette ND, McQuain WD

Deletion Functions

```
//////////////////////////////////////////////////////
// Deletes element at specified index, if possible.
//
// Parameters
//     Idx        index of item to be deleted
// Returns
//     true       if deletion is carried out
//     false      otherwise
//
bool ArrayClass::DeleteAtIndex(int Idx) {

   if (Idx >= Usage || Idx < 0) return false;

   ShiftTailDown(Idx);
   Usage--;
   return true;
}
//////////////////////////////////////////////////////
// Deletes all elements that have specified value.
//
// Parameters
//     Value      value of items to be deleted
// Returns
//     true       if at least one deletion is carried
//                out
//     false      otherwise
//
bool ArrayClass::DeleteValue(int Value) {

   bool Found = false;
   if (Usage == 0) return Found;

   int Idx = 0;
   while (Idx < Usage) {
      if (List[Idx] == Value) {
         DeleteAtIndex(Idx);
         Found = true;
      }
      Idx++;
   }
   return Found;
}
```


©1995-2000 Barnette ND, McQuain WD

Search Functions

```
//////////////////////////////////////////////////////
// Returns element at specified index, if possible.
//
// Parameters
//     Idx        index of item to be deleted
// Returns
//     if index is valid, value at Idx in List
//     -1 otherwise
//
int ArrayClass::Retrieve(int Idx) const {

   if (Idx >= Usage || Idx < 0)
      return -1;
   else
      return List[Idx];
}

//////////////////////////////////////////////////////
// Returns index of first occurrence of specified value
// in List.
//
// Parameters
//     Value      value to be located in List
// Returns
//     if value occcurs in List, index of first
//                occurrence
//     -1 otherwise
//
int ArrayClass::FindValue(int Value) const {

   if (Usage == 0) return -1;

   int Idx = 0;
   while (Idx < Usage) {
      if (List[Idx] == Value)
         return Idx;
      Idx++;
   }
   return -1;
}
```


©1995-2000 Barnette ND, McQuain WD

Miscellaneous Functions

```
//////////////////////////////////////////////
// Deletes memory allocated for List.
//
ArrayClass::~ArrayClass() {

   Capacity = 0;
   Usage    = 0;
   if (List != NULL) {
      delete [] List;
      List = NULL;
   }
}
```

```
//////////////////////////////////////////////////////
// Stores randomly generated int values into list;
// values are in range 0 to Max-1.
//
// Parameters
//     Max    determines range of values stored
//
void ArrayClass::Randomize(int Max) {

   srand( (unsigned)time( NULL ) );

   for (int Idx = 0; Idx < Capacity; Idx++) {
      List[Idx] = rand() % Max;
   }
   Usage = Capacity;
}
```

```
//////////////////////////////////////////////////
// Reverses the order of the elements in the List
//
void ArrayClass::Reverse() {

   for (int Idx = 0; Idx < Usage/2; Idx++) {
      Swap(List[Idx], List[Usage - 1 - Idx]);
   }
}
```


©1995-2000 Barnette ND, McQuain WD

Driver

```
#include <iostream>
#include <iomanip>

#include "ArrayClass.h"

using namespace std;

void PrintList(const ArrayClass& pList);

void main() {

   ArrayClass myArray(10, 5);

   int Limit = myArray.getCapacity();
   cout << "myArray capacity is: "
        << Limit << endl;

   myArray.Randomize(1000);
   PrintList(myArray);

   ArrayClass myArrayClone(myArray);

   myArrayClone.Reverse();
   PrintList(myArrayClone);

   myArrayClone = myArray;
   PrintList(myArrayClone);
}

void PrintList(const ArrayClass& pList)

   int k = 0;
   for (k = 0; k < pList.getCapacity();
      cout << setw(3) << k
           << setw(8)
           << pList.Retrieve(k)
           << endl;
   }
   cout << endl;
}
```

```
myArray capacity is: 10
  0     987
  1     465
  2     958
  3     932
  4     947
  5     280
  6      30
  7      44
  8     520
  9     694

  0     694
  1     520
  2      44
  3      30
  4     280
  5     947
  6     932
  7     958
  8     465
  9     987

  0     987
  1     465
  2     958
  3     932
  4     947
  5     280
  6      30
  7      44
  8     520
  9     694
```


©1995-2000 Barnette ND, McQuain WD