Ch 17.2

Wednesday, February 07, 2007
12:23 AM

17.2 Templates for Data Abstraction

Syntax for class templates

```
template<class T>
class ClassName
{
    private:
        T var1;
        :
        T varN;
        T *ptr1;
          :
        T *ptrN;
        T arr1[50];
          :
        T arrN[50];
    public:
        ClassName();
        ClassName(T v, T *p, T a[]);
        T get_element(int index);
};
```

Can use template wildcard for:

member variable type for normal
vars, pointers and arrays
return type for member/friend func
parameter type for funcs

Syntax for declaring template class obj

```
ClassName <datatype> objectName;
Example: vector <int> v;
```

Syntax for bodies of member/friend functions:

```
template <class T>
ClassName <T> :: ClassName (T v, T*p, T a[])
{
    functionBody
}
```

Syntax for passing template class to function

```
returnType funcName (ClassName<datatype> & c);
-or-
template<class T>
returnType funcName (ClassName <T> & c);
```

Syntax for typedef;

```
typedef ClassName <datatype> NewName;

NewName varName;
```

Example from Book - Array Class pp 908-13

Header file:

```
//DISPLAY 17.4 Interface for the Class Template GenericList
//This is the header file genericlist.h. This is the interface for the
//class GenericList. Objects of type GenericList can be a list of items
//of any type for which the operators << and = are defined.
//All the items on any one list must be of the same type. A list that
//can hold up to max items all of type Type_Name is declared as follows:
//          GenericList<Type_Name> the_object(max);
#ifndef GENERICLIST_H
#define GENERICLIST_H
#include <iostream>
using namespace std;
```

```
namespace listsavitch
{
        template<class ItemType>
        class GenericList
        {
        public:
                GenericList(int max);
                //Initializes the object to an empty list that can hold up to
                //max items of type ItemType.
                ~GenericList( );
                //Returns all the dynamic memory used by the object to the
                freestore.

        int length( ) const;
        //Returns the number of items on the list.

        void add(ItemType new_item);
        //Precondition: The list is not full.
        //Postcondition: The new_item has been added to the list.

        bool full( ) const;
        //Returns true if the list is full.


void erase( );
        //Removes all items from the list so that the list is empty.

        friend ostream& operator <<(ostream& outs,
                const GenericList<ItemType>& the_list);
        //Overloads the << operator so it can be used to output the
        //contents of the list. The items are output one per line.
        //Precondition: If outs is a file output stream, then outs has
        //already been connected to a file.
        private:
                ItemType *item; //pointer to the dynamic array that holds
                the list.
                int max_length; //max number of items allowed on the list.
                int current_length; //number of items currently on the list.
        };
}//listsavitch
#endif //GENERICLIST_H
```

Source file:

```
//DISPLAY 17.6 Implementation of GenericList
//This is the implementation file: genericlist.cpp
//This is the implementation of the class template named GenericList.
//The interface for the class template GenericList is in the
//header file genericlist.h.
#ifndef GENERICLIST_CPP
#define GENERICLIST_CPP
#include <iostream>
#include <cstdlib>
#include "genericlist.h"//This is not needed when used as we are using this file,
        //but the #ifndef in genericlist.h makes it safe.
using namespace std;

namespace listsavitch
namespace listsavitch
{
        //Uses cstdlib:
        template<class ItemType>
        GenericList<ItemType>::GenericList(int max) : max_length(max),
                current_length(0)

        {
                item = new ItemType[max];
        }

        template<class ItemType>
        GenericList<ItemType>::~GenericList( )


{
        delete [] item;
        }

        template<class ItemType>
        int GenericList<ItemType>::length( ) const
        {
                return (current_length);
        }

        //Uses iostream and cstdlib:
        template<class ItemType>
        void GenericList<ItemType>::add(ItemType new_item)
        {
                if ( full( ) )
                {
                     cout << "Error: adding to a full list.\n";
                     exit(1);
                }
                else
                {
                     item[current_length] = new_item;
                     current_length = current_length + 1;
                }
        }

        template<class ItemType>
        bool GenericList<ItemType>::full( ) const
        {
                return (current_length == max_length);
        }

        template<class ItemType>
        void GenericList<ItemType>::erase( )
        {
                current_length = 0;
        }

        //Uses iostream:
        template<class ItemType>
        ostream& operator <<(ostream& outs, const GenericList<ItemType>&
        the_list)


{
        for (int i = 0; i < the_list.current_length; i++)
```

```
                                outs << the_list.item[i] << endl;

                return outs;
                }
}//listsavitch
#endif // GENERICLIST_CPP Notice that we have enclosed all the template
                // definitions in #ifndef... #endif.
```

Main file:

```
//DISPLAY 17.5 Program Using the GenericList Class Template
//Program to demonstrate use of the class template GenericList.
#include <iostream>
#include "genericlist.h"
#include "genericlist.cpp"
using namespace std;
using namespace listsavitch;

int main( )
{
                GenericList<int> first_list(2);
                first_list.add(1);
                first_list.add(2);
                cout << "first_list = \n"
                                << first_list;

                GenericList<char> second_list(10);
                second_list.add('A');
                second_list.add('B');
                second_list.add('C');
                cout << "second_list = \n"
                                << second_list;

                return 0;
}
```