Dynamic Arrays in Classes

Modified from Section 11.4

Classes and Dynamic Arrays

- A class can have a member variable that is a dynamic array
  - In this section you will see a class using a dynamic array as a member variable.

Program Example: A String Variable Class

- We will define the class StringVar
  - StringVar objects will be string variables
  - StringVar objects use character arrays to store data
  - StringVar objects use dynamic arrays whose size is determined when the program is running

The StringVar Constructors

- The default StringVar constructor creates an object with a maximum string length of 100
- Another StringVar constructor takes an argument of type int which determines the maximum string length of the object
- A third StringVar constructor takes a C-string argument and…
  - sets maximum length to the length of the C-string
  - copies the C-string into the object's string value

The StringVar Interface

- In addition to constructors, the class interface includes:
  - Member functions
    - int length( );
    - void input_line(istream& ins);
  - A friend function
    - friend ostream& operator << (ostream& outs,
      const StringVar& the_string);
  - Copy constructor …discussed later
  - Destructor …discussed later

Display 11.11 (1) **Display 11.11 (2)**

Display 11.11 (1/3)

DISPLAY 11.11 Program Using the StringVar Class *(part 1 of 3)*

```
1  //This is the definition for the class StringVar
2  //whose values are strings. An object is declared as follows.
3  //Note that you use (max_size), not [max_size]
4  //     StringVar the_object(max_size);
5  //where max_size is the longest string length allowed.
6  #include <iostream>
7  using namespace std;
8
9  class StringVar
10 {
11 public:
12     StringVar(int size);
13     //Initializes the object so it can accept string values up to size
14     //in length. Sets the value of the object equal to the empty string.
15
```

(continued)

Display 11.11 (2/3)

DISPLAY 11.11 **Program Using the `StringVar` Class** *(part 2 of 3)*

```
16      StringVar( );
17      //Initializes the object so it can accept string values of length 100
18      //or less. Sets the value of the object equal to the empty string.
19
20      StringVar(const char a[]);
21      //Precondition: The array a contains characters terminated with '\0'.
22      //Initializes the object so its value is the string stored in a and
23      //so that it can later be set to string values up to strlen(a) in length
24
25      StringVar(const StringVar& string_object);
26      //Copy constructor.
27
28      ~StringVar( );
29      //Returns all the dynamic memory used by the object to the freestore.
30
31      int length( ) const;
32      //Returns the length of the current string value.
33      void input_line(istream& ins);
34      //Precondition: If ins is a file input stream, then ins has been
35      //connected to a file.
36      //Action: The next text in the input stream ins, up to '\n', is copied
37      //to the calling object. If there is not sufficient room, then
38      //only as much as will fit is copied.
39      friend ostream& operator <<(ostream& outs, const StringVar& the_string);
40      //Overloads the << operator so it can be used to output values
41      //of type StringVar
42      //Precondition: If outs is a file output stream, then outs
43      //has already been connected to a file.
44
45  private:
46      char *value; //pointer to dynamic array that holds the string value.
47      int max_length; //declared max length of any string value.
48  };
49
50
51  <The definitions of the member functions and overloaded operators go here>
52
53  //Program to demonstrate use of the class StringVar.
54
55  void conversation(int max_name_size);
56  //Carries on a conversation with the user.
57
```

(continued)

The StringVar Implementation

- StringVar uses a dynamic array to store a string
  - StringVar constructors call new to create the dynamic array for member variable value
  - '\0' is used to terminate the string
  - The size of the array is not determined until the array is declared
    - Constructor arguments determine the size

Dynamic Variables

- Dynamic variables do not "go away" unless delete is used
  - Even if a local pointer variable goes away at the end of a function, the dynamic variable it pointed to remains unless delete is used
  - A user of the SringVar class could not know that a dynamic array is a member of the class, so could not be expected to call delete when finished with a StringVar object
  - We can define the destructor to delete the array

Destructors

- A destructor is a member function that is called automatically when an object of the class goes out of scope
  - A class can only have one destructor with no arguments.
  - The name of the destructor is distinguished from the default constructor by the tilde symbol ~
  - The destructor (usually) contains code to delete all dynamic variables created by the object
    - Memory used by non-dynamic variables and arrays will be automatically deallocated

Display 11.12 (1/2)

DISPLAY 11.12 **Implementation of `StringVar`** *(part 1 of 2)*

```
1  //This is the implementation of the class StringVar.
2  //The definition for the class StringVar is in Display 11.11.
3  #include <cstdlib>
4  #include <cstddef>
5  #include <cstring>
6
7  //Uses cstddef and cstdlib:
8  StringVar::StringVar(int size) : max_length(size)
9  {
10     value = new char[max_length + 1];//+1 is for '\0'.
11     value[0] = '\0';
12 }
13
14 //Uses cstddef and cstdlib:
15 StringVar::StringVar( ) : max_length(100)
16 {
17     value = new char[max_length + 1];//+1 is for '\0'.
18     value[0] = '\0';
19 }
20
21 //Uses cstring, cstddef, and cstdlib:
22 StringVar::StringVar(const char a[]) : max_length(strlen(a))
23 {
24     value = new char[max_length + 1];//+1 is for '\0'.
25     strcpy(value, a);
26 }
27 //Uses cstring, cstddef, and cstdlib:
28 StringVar::StringVar(const StringVar& string_object)
29                         : max_length(string_object.length( ))
30 {
31     value = new char[max_length + 1];//+1 is for '\0'.
32     strcpy(value, string_object.value);
33 }
34 StringVar::~StringVar( )
35 {
36     delete [] value;
37 }
38
39 //Uses cstring:
40 int StringVar::length( ) const
41 {
42     return strlen(value);
43 }
44
45 //Uses iostream:
```

Copy constructor (discussed later in this chapter)

Destructor

(continued)

Display 11.12 (2/2)

DISPLAY 11.12 Implementation of StringVar *(part 2 of 2)*

```
46   void StringVar::input_line(istream& ins)
47   {
48       ins.getline(value, max_length + 1);
49   }
50
51   //Uses iostream:
52   ostream& operator <<(ostream& outs, const StringVar& the_string)
53   {
54       outs << the_string.value;
55       return outs;
56   }
```

Copy Constructor

- The copy constructor is a constructor which an takes an object of the same class, and creates an object by initializing it with the input object
- The copy constructor is called automatically
  - When a class object is declared and initialized by an object of the same class
  - When a function returns a value of the class type
  - When an argument of the class type is plugged in for a call-by-value parameter
- If there is no copy constructor defined for the class, C++ uses the default copy constructor which copies each data member

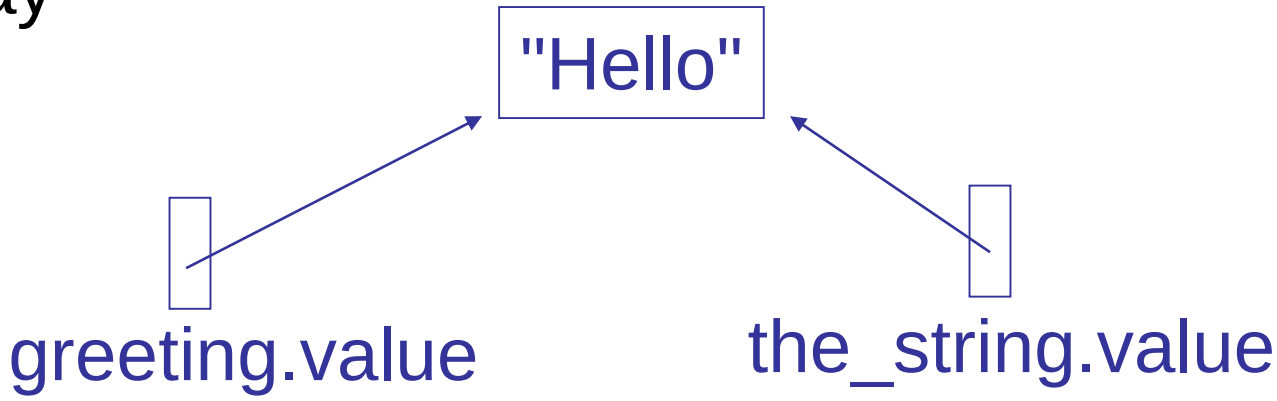
The Need For a Copy Constructor

- void show_string(StringVar the_string) {…}
  StringVar greeting("Hello");
  show_string(greeting);
  cout << greeting << endl;

- When function show_string is called, greeting is copied into the_string
  - the_string.value is set equal to greeting.value. Two pointers are pointer to the same memory location!
  - When ‘show_string’ is finished, the destructor is called for the local object ‘the_string’

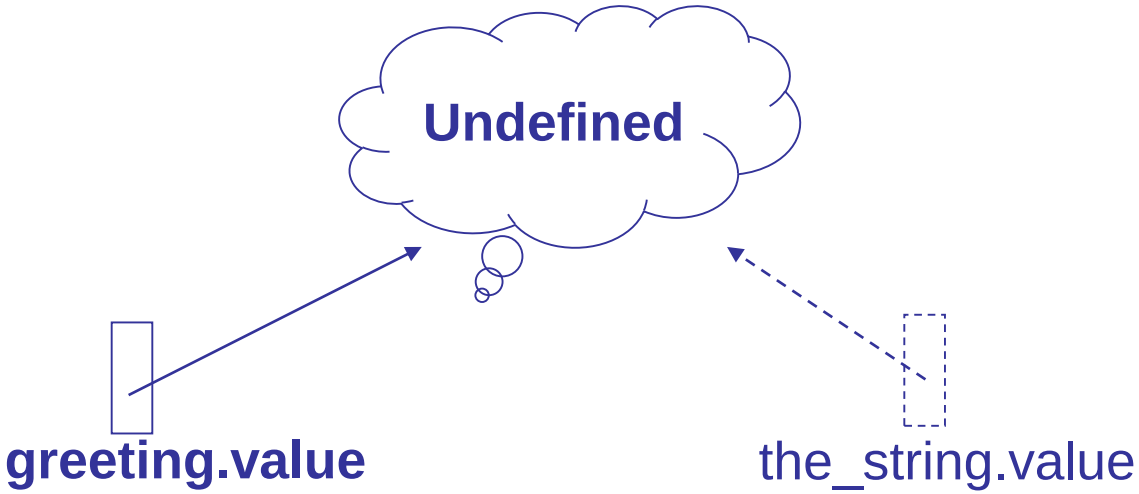
The Need For a Copy Constructor

- Since greeting.value and the_string.value are pointers, they now point to the same dynamic array



The Need For a Copy Constructor

- When show_string ends, the destructor for the_string executes, returning the dynamic array pointed to by the_string.value to the freestore



- greeting.value now points to memory that has been given back to the freestore!

The Need For a Copy Constructor

- Two problems now exist for object greeting
  - Attempting to output greeting.value is likely to produce an error
    - In some instances all could go OK
  - When greeting goes out of scope, its destructor will be called
    - Calling a destructor for the same location twice is likely to produce a system crashing error (double free)

Copy Constructors

- The problem with using call-by-value parameters with pointer variables is solved by a user-defined copy constructor.
- A copy constructor is a constructor with one parameter of the same type as the class
  - The parameter is a call-by-reference parameter
  - The parameter is usually a constant parameter
  - The constructor creates a complete, independent copy of its argument

StringVar Copy Constructor

- This code for the StringVar copy constructor
  - Creates a new dynamic array for a copy of the argument
  - StringVar::StringVar(const StringVar& string_object)
    : max_length(string_object.length())
    {
    value = new char[max_length+ 1];
    strcpy(value, string_object.value);
    }

Copy Constructor Demonstration

- Using the same example, but with a copy constructor defined
  - greeting.value and the_string.value point to different locations in memory