Chapter 7

Constructors and Vectors

Learning Objectives

- Constructors
  - Definitions
  - Calling
- More Tools
  - const parameter modifier
  - Inline functions
  - Static member data
- Vectors
  - Introduction to vector class

Constructors

- Initialization of objects
  - Initialize some or all member variables
  - Other actions possible as well
- A special kind of member function
  - Automatically called when object declared
- Very useful tool
  - Key principle of OOP

Constructor Definitions

- Constructors defined like any member function
  - Except:
    1. Must have same name as class
    2. Cannot return a value; *not even void!*

Constructor Definition Example

- Class definition with constructor:
  - class DayOfYear

```
{
public:
    DayOfYear(int monthValue, int dayValue);
            //Constructor initializes month & day
    void input();
    void output();
    …
private:
    int month;
    int day;
}
```

Constructor Notes

- Notice name of constructor: DayOfYear
  - Same name as class itself!
- Constructor declaration has no return-type
  - Not even void!
- Constructor in public section
  - It's called when objects are declared
  - If private, could never declare objects!

Calling Constructors

- Declare objects:

```
DayOfYear date1(7, 4),
          date2(5, 5);
```

- Objects are created here
  - Constructor is called
  - Values in parents passed as arguments to constructor
  - Member variables month, day initialized:
    date1.month → 7 date2.month → 5
    date1.dat → 4 date2.day → 5

Constructor Equivalency

- Consider:
  - date1.DayOfYear(7, 4);   // ILLEGAL!
    date2.DayOfYear(5, 5);   // ILLEGAL!
- Seemingly OK…
  - CANNOT call constructors like other member functions!

Constructor Code

- Constructor definition is like all other member functions:
  ```
  DayOfYear::DayOfYear(int monthValue, int dayValue)
  {
          month = monthValue;
          day = dayValue;
  }
  ```
- Note same name around ::
  - Clearly identifies a constructor
- Note no return type
  - Just as in class definition

Alternative Definition

- Previous definition equivalent to:

```
DayOfYear::DayOfYear(int monthValue, int dayValue)
                    : month(monthValue), day(dayValue)
{…}
```

- Second line called "Initialization Section"
- Body left empty

Constructor Additional Purpose

- Not just initialize data
- Body doesn't have to be empty
  - In initializer version
- Validate the data!
  - Ensure only appropriate data is assigned to class private member variables
  - Powerful OOP principle

Constructor with No Arguments

- Can be confusing
- Standard functions with no arguments:
  – Called with syntax: callMyFunction();
    - Including empty parentheses
- Object declarations with no "initializers":
  – DayOfYear date3;   // This way!
  – DayOfYear date3();   // NO!
    - What is this really?
    - Compiler sees a function declaration/prototype!
    - Yes! Look closely!

Explicit Constructor Calls

- Can also call constructor AGAIN
  - After object declared
    - Recall: constructor was automatically called then
- Convenient method of *setting member variables*
- Method quite different from standard member function call

Explicit Constructor Call Example

- Such a call returns "anonymous object"
  - Which can then be assigned
  - **In Action**:
    DayOfYear holiday(7, 4);
    - Constructor called at object's declaration
    - Now to "re-initialize":
      holiday = DayOfYear(5, 5);
      - Explicit constructor call
      - Returns new "anonymous object"
      - Assigned back to current object

Default Constructor

- Defined as: constructor w/ no arguments
- One should always be defined
- Auto-Generated?
  - Yes & No
  - If no constructors AT ALL are defined → Yes
  - If any constructors are defined → No
- If no default constructor:
  - Cannot declare: MyClass myObject;
    - With no initializers

Class Type Member Variables

- Class member variables can be any type
  - Including objects of other classes!
  - Type of class relationship
    - Powerful OOP principle
- Need special notation for constructors
  - So they can call "back" to member object’s constructor

Class Member Variable Example: **Display 7.1** A Class Member Variable (1 of 5)

Display 7.3 A Class Member Variable

```cpp
#include <iostream>
#include<cstdlib>
using namespace std;

class DayOfYear
{
public:
    DayOfYear(int monthValue, int dayValue);
    DayOfYear(int monthValue);
    DayOfYear( );
    void input( );
    void output( );
    int getMonthNumber( );
    int getDay( );
private:
    int month;
    int day;
    void testDate( );
};
```

The class **DayOfYear** *is the same as in Display 7.1, but we have repeated all the details you need for this discussion.*

Class Member Variable Example: **Display 7.3** A Class Member Variable (2 of 5)

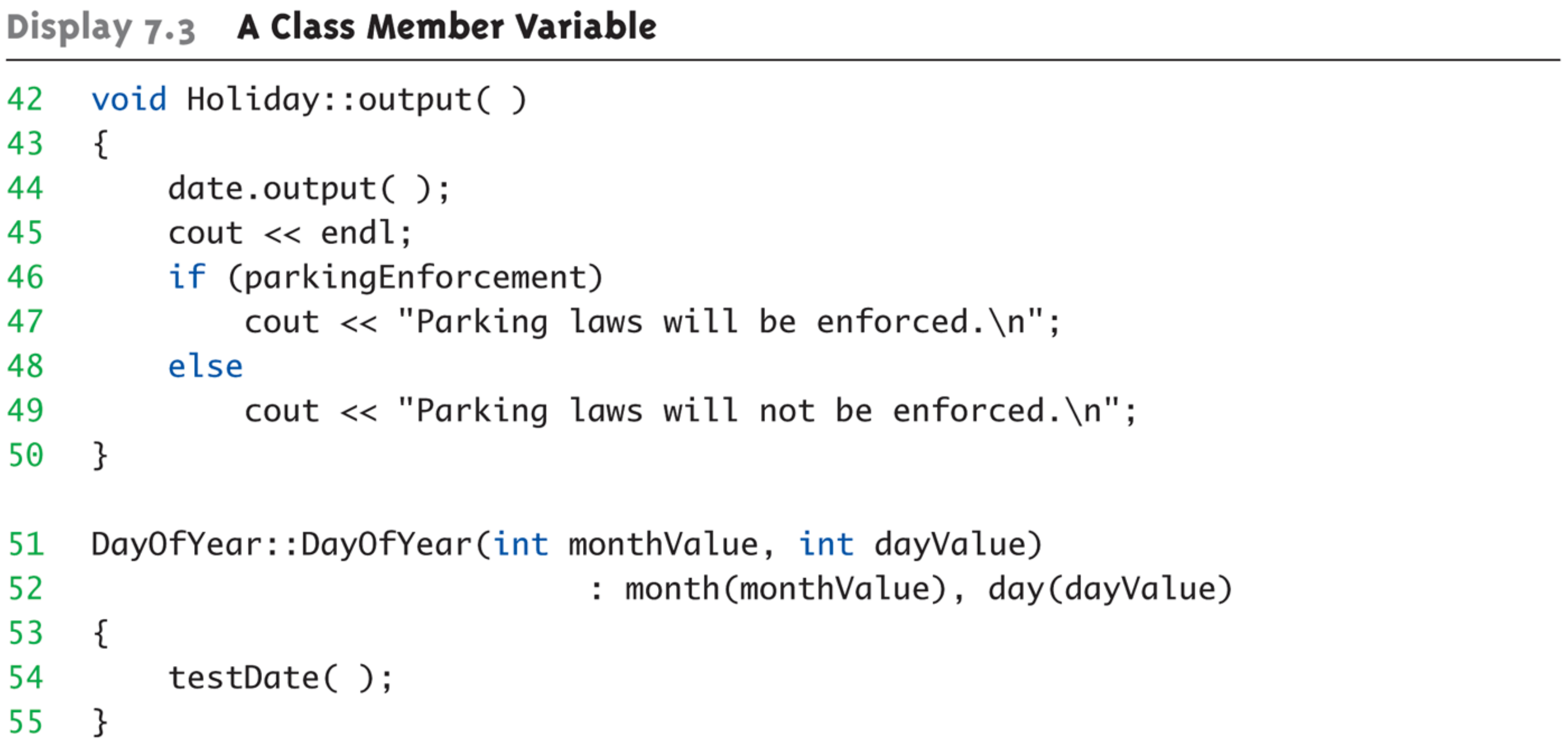
```
class Holiday
{
public:
    Holiday( );//Initializes to January 1 with no parking enforcement
    Holiday(int month, int day, bool theEnforcement);
    void output( );
private:
    DayOfYear date;
    bool parkingEnforcement;//true if enforced
};

int main( )
{
    Holiday h(2, 14, true);
    cout << "Testing the class Holiday.\n";
    h.output( );

    return 0;
}

Holiday::Holiday( ) : date(1, 1), parkingEnforcement(false)
{/*Intentionally empty*/}

Holiday::Holiday(int month, int day, bool theEnforcement)
                    : date(month, day), parkingEnforcement(theEnforcement)
{/*Intentionally empty*/}
```

member variable of a class type

Invocations of constructors from the class DayOfYear*.*

(continued)

Class Member Variable Example: **Display 7.3** A Class Member Variable (3 of 5)



Display 7.3 A Class Member Variable

```
void Holiday::output( )
{
    date.output( );
    cout << endl;
    if (parkingEnforcement)
        cout << "Parking laws will be enforced.\n";
    else
        cout << "Parking laws will not be enforced.\n";
}

DayOfYear::DayOfYear(int monthValue, int dayValue)
                          : month(monthValue), day(dayValue)
{
    testDate( );
}
```

Class Member Variable Example: **Display 7.3** A Class Member Variable (4 of 5)

```
//uses iostream and cstdlib:
void DayOfYear::testDate( )
{
    if ((month < 1) || (month > 12))
    {
        cout << "Illegal month value!\n";
        exit(1);
    }
    if ((day < 1) || (day > 31))
    {
        cout << "Illegal day value!\n";
        exit(1);
    }
}

//Uses iostream:
void DayOfYear::output( )
{
    switch (month)
    {
        case 1:
            cout << "January "; break;
        case 2:
            cout << "February "; break;
        case 3:
            cout << "March "; break;
              .
              .
              .
```

The omitted lines are in Display 6.3, but they are obvious enough that you should not have to look there.

Class Member Variable Example: **Display 7.3** A Class Member Variable (5 of 5)

Display 7.3 A Class Member Variable

```
            case 11:
                cout << "November "; break;
            case 12:
                cout << "December "; break;
            default:
                cout << "Error in DayOfYear::output. Contact software vendor.";
        }

        cout << day;
    }
```

SAMPLE DIALOGUE

```
Testing the class Holiday.
February 14
Parking laws will be enforced.
```

Parameter Passing Methods

- Efficiency of parameter passing
  - Call-by-value
    - Requires copy be made → Overhead
  - Call-by-reference
    - Placeholder for actual argument
    - Most efficient method
  - Negligible difference for simple types
  - For class types → clear advantage
- Call-by-reference desirable
  - Especially for "large" data, like class types

Vectors

- Vector Introduction
  - Recall: arrays are fixed size
  - Vectors: "arrays that grow and shrink at run time"
    - During program execution
  - Formed from Standard Template Library (STL)

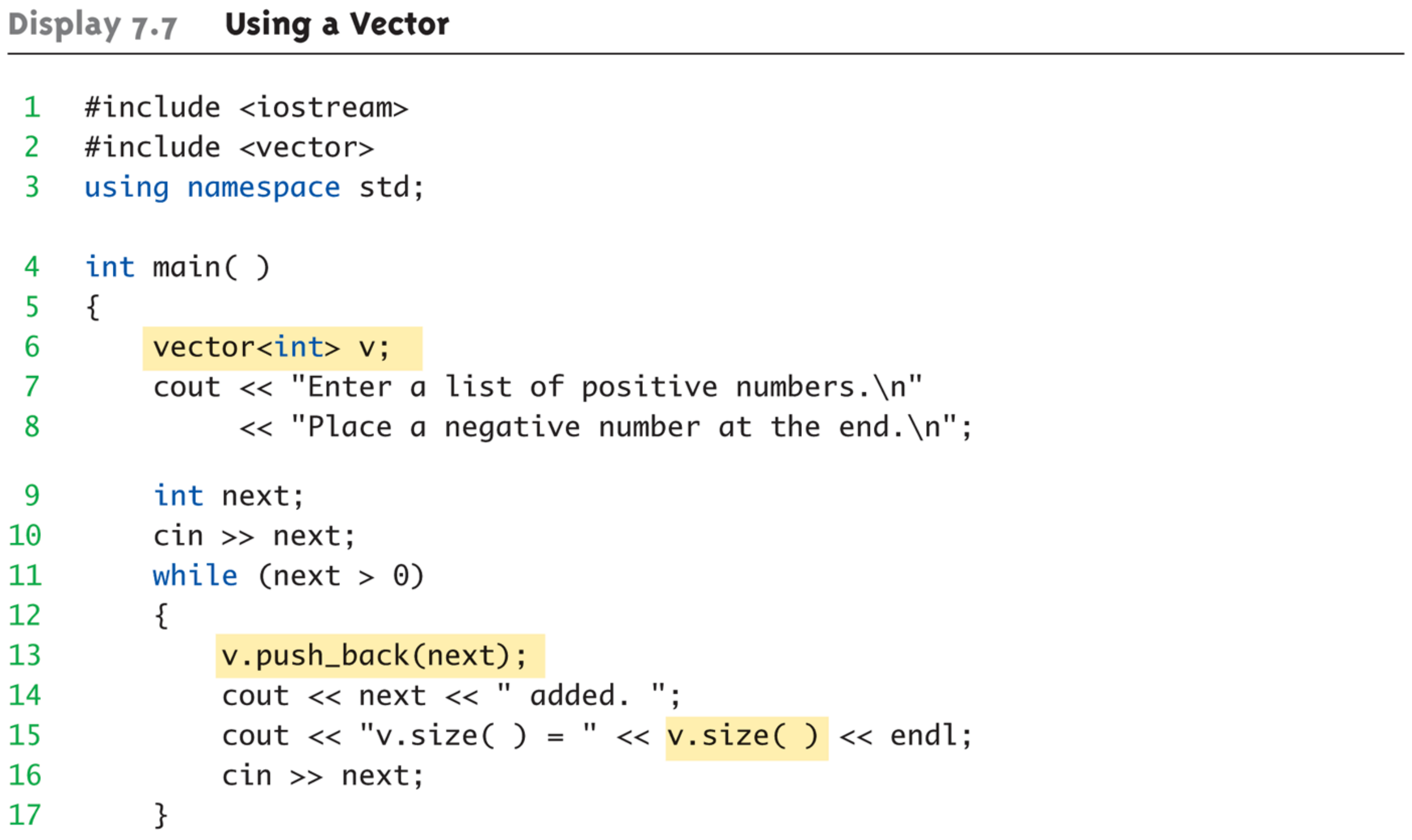
Vector Basics

- Similar to array:
  - Has base type
  - Stores collection of base type values
- Declared differently:
  - Syntax: vector<Base_Type>
    - Indicates template class
    - Any type can be "plugged in" to Base_Type
    - Produces "new" class for vectors with that type
  - Example declaration:
    `vector<int> v;`

Vector Use

- vector<int> v;
  - "v is vector of type int"
  - Calls class default constructor
    - Empty vector object created
- Indexed like arrays for access
- But to add elements:
  - Must call member function push_back
- Member function size()
  - Returns current number of elements

Vector Example:
Display 7.7 Using a Vector (1 of 2)



Display 7.7 **Using a Vector**

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

int main( )
{
    vector<int> v;
    cout << "Enter a list of positive numbers.\n"
         << "Place a negative number at the end.\n";

    int next;
    cin >> next;
    while (next > 0)
    {
        v.push_back(next);
        cout << next << " added. ";
        cout << "v.size( ) = " << v.size( ) << endl;
        cin >> next;
    }
```

Vector Example: **Display 7.7** Using a Vector (2 of 2)

```
        cout << "You entered:\n";
        for (unsigned int i = 0; i < v.size( ); i++)
            cout << v[i] << " ";
        cout << endl;

        return 0;
    }
```

SAMPLE DIALOGUE

```
Enter a list of positive numbers.
Place a negative number at the end.
2 4 6 8 -1
2 added. v.size = 1
4 added. v.size = 2
6 added. v.size = 3
8 added. v.size = 4
You entered:
2 4 6 8
```