CMPE 180-92

Data Structures and Algorithms in C++

September 28 Class Meeting

Department of Computer Engineering
San Jose State University

Spring 2017
Instructor: Ron Mak

www.cs.sjsu.edu/~mak

Assignment #5 Sample Solution

RomanNumeral.h

```
class RomanNumeral
{
public:
    RomanNumeral();
    RomanNumeral(string roman);
    RomanNumeral(int value);
    ~RomanNumeral();

    string get_roman()   const;
    int    get_decimal() const;

    // Overload the arithmetic operators.
    RomanNumeral operator +(const RomanNumeral& other);
    RomanNumeral operator -(const RomanNumeral& other);
    RomanNumeral operator *(const RomanNumeral& other);
    RomanNumeral operator /(const RomanNumeral& other);

    // Overload the equality operators.
    bool operator ==(const RomanNumeral& other);
    bool operator !=(const RomanNumeral& other);
```

Assignment #5 Sample Solution, *cont'd*

RomanNumeral.h

```
    // Overload the stream >> and << operators.
    friend istream& operator >>(istream& in, RomanNumeral& numeral);
    friend ostream& operator <<(ostream& out, const RomanNumeral& numeral);

private:
    string roman;       // Roman numeral as a string
    int    decimal;     // decimal value of the Roman numeral

    void to_roman();     // calculate string from decimal value
    void to_decimal();   // calculate decimal value from string
};
```

```
RomanNumeral::RomanNumeral() : roman(""), decimal(0)
{
}

RomanNumeral::RomanNumeral(string str) : roman(str)
{
    // Compute the decimal value.
    to_decimal();
}

RomanNumeral::RomanNumeral(int value) : decimal(value)
{
    // Compute the Roman numeral string.
    to_roman();
}

RomanNumeral::~RomanNumeral() {}

string RomanNumeral::get_roman() const { return roman; }

int RomanNumeral::get_decimal() const { return decimal; }
```

Assignment #5 Sample Solution, *cont'd*

```
RomanNumeral RomanNumeral::operator +(const RomanNumeral& other)
{
    int value = decimal + other.decimal;
    RomanNumeral sum(value);
    return sum;
}

RomanNumeral RomanNumeral::operator -(const RomanNumeral& other)
{
    int value = decimal - other.decimal;
    RomanNumeral diff(value);
    return diff;
}

RomanNumeral RomanNumeral::operator *(const RomanNumeral& other)
{
    int value = decimal*other.decimal;
    RomanNumeral prod(value);
    return prod;
}
```

Assignment #5 Sample Solution, *cont'd*

```
RomanNumeral RomanNumeral::operator /(const RomanNumeral& other)
{
    int value = decimal/other.decimal;
    RomanNumeral quot(value);
    return quot;
}

bool RomanNumeral::operator ==(const RomanNumeral& other)
{
    return decimal == other.decimal;
}

bool RomanNumeral::operator !=(const RomanNumeral& other)
{
    return decimal != other.decimal;
}
```

Assignment #5 Sample Solution, *cont'd*

```
istream& operator >>(istream& in, RomanNumeral& numeral)
{
    string str;
    in >> str;

    numeral.roman = str;
    numeral.to_decimal();

    return in;
}

ostream& operator <<(ostream& out, const RomanNumeral& numeral)
{
    out << "[" << numeral.decimal << ":" << numeral.roman << "]";
    return out;
}
```

Why not `numeral->roman` and `numeral->to_decimal()`?

Assignment #5 Sample Solution, *cont'd*

```
void RomanNumeral::to_roman()
{
    int temp = decimal;
    roman = "";

    while (temp >= 1000)
    {
        roman += "M";
        temp -= 1000;
    }

    if (temp >= 900)
    {
        roman += "CM";
        temp -= 900;
    }
```

```
    else if (temp >= 500)
    {
        roman += "D";
        temp -= 500;
    }
    else if (temp >= 400)
    {
        roman += "CD";
        temp -= 400;
    }

    while (temp >= 100)
    {
        roman += "C";
        temp -= 100;
    }
```

Assignment #5 Sample Solution, *cont'd*

```
    if (temp >= 90)
    {
        roman += "XC";
        temp -= 90;
    }
    else if (temp >= 50)
    {
        roman += "L";
        temp -= 50;
    }
    else if (temp >= 40)
    {
        roman += "XL";
        temp -= 40;
    }

    while (temp >= 10)
    {
        roman += "X";
        temp -= 10;
    }
```

```
    if (temp >= 9)
    {
        roman += "IX";
        temp -= 9;
    }
    else if (temp >= 5)
    {
        roman += "V";
        temp -= 5;
    }
    else if (temp >= 4)
    {
        roman += "IV";
        temp -= 4;
    }

    while (temp >= 1)
    {
        roman += "I";
        temp--;
    }
}
```

Assignment #5 Sample Solution, *cont'd*

```
void RomanNumeral::to_decimal()
{
    int length = roman.length();
    decimal = 0;

    // Scan the Roman numeral string from left to right
    // and add the corresponding character values.
    for (int i = 0; i < length; i++)
    {
        switch (roman[i])
        {
            case 'M':
                decimal += 1000;
                break;

            case 'D':
                decimal += 500;
                break;
```

Assignment #5 Sample Solution, *cont'd*

```
            case 'C':
                if (i+1 < length)
                {
                    switch (roman[i+1])
                    {
                        case 'D':  // CD
                            decimal += 400;
                            i++;
                            break;

                        case 'M':  // CM
                            decimal += 900;
                            i++;
                            break;

                        default:
                            decimal += 100;
                            break;
                    }
                }
                else decimal += 100;
                break;
```

San José State University

Assignment #5 Sample Solution, *cont'd*

```
            case 'L':
                decimal += 50;
                break;

            case 'X':
                if (i+1 < length)
                {
                    switch (roman[i+1])
                    {
                        case 'L':  // XL
                            decimal += 40;
                            i++;
                            break;

                        case 'C':  // XC
                            decimal += 90;
                            i++;
                            break;

                        default:
                            decimal += 10;
                            break;
                    }
                }
                else decimal += 10;
                break;
```

Assignment #5 Sample Solution, *cont'd*

```
            case 'V':
                decimal += 5;
                break;

            case 'I':
                if (i+1 < length)
                {
                    switch (roman[i+1])
                    {
                        case 'V':  // IV
                            decimal += 4;
                            i++;
                            break;

                        case 'X':  // IX
                            decimal += 9;
                            i++;
                            break;

                        default:
                            decimal++;
                            break;
                    }
                }
                else decimal++;
                break;
        }
    }
}
```

Arrays of Objects

- An array of **`Birthday`** objects:

```
Birthday celebrations[10];
```

- A dynamic array of **`Birthday`** objects:

```
Birthday *parties = new Birthday[count];
```

- When you create an array of objects, the default constructor is called for each element.
- Therefore, a class that can be the base type of an array must have a default constructor.

Destructors

- A destructor is a member function of a class that is called automatically whenever an object of the class is destroyed.
  - An object is destroyed automatically when it goes out of scope.
  - An object that was dynamically created with **`new`** and is later explicitly destroyed with **`delete`**.
- The name of the destructor is the name of the class, preceded by a tilde **`~`**
  - It has no return type and no parameters.

Destructors, *cont'd*

- C++ generates a default destructor that does nothing.
- But you can write your own destructor.

Destructors, *cont'd*

Birthday3.h

```
class Birthday
{
public:
    // Constructors
    Birthday();
    Birthday(int y, int m, int d);

    // Destructor
    ~Birthday();
    ...
}
```

```
Birthday::~Birthday()
{
    // Empty body
}
```

Birthday3.cpp

- Use the body of the destructor that you write to:
  - Delete any objects that the class dynamically allocated.
  - Close any open files.
  - etc.

Destructors, *cont'd*

- Just to confirm that the destructor is called:

Birthday3.cpp

```
Birthday::~Birthday()
{
    cout << "*** Destructor called for " << *this << endl;
}
```

Destructors, *cont'd*

```
#include <iostream>
#include "Birthday3.h"

int main()
{
    Birthday *pbd0 = new Birthday();             // call default constructor
    Birthday *pbd1 = new Birthday(1981, 9, 2);   // call constructor
    Birthday *pbd2 = new Birthday(1992, 5, 8);   // call constructor

    pbd0->print();
    pbd1->print();
    (*pbd2).print();
    cout << *pbd0 << ", " << *pbd1 << ", " << *pbd2 << endl;

    cout << endl;
    cout << years_apart(*pbd1, *pbd2) << "
    cout << *pbd1 - *pbd2 << " years apart

    delete pbd0;
    delete pbd1;
    delete pbd2;
}
```

```
0/0/0
9/2/1981
5/8/1992
0/0/0, 9/2/1981, 5/8/1992

11 years apart
11 years apart
*** Destructor called for 0/0/0
*** Destructor called for 9/2/1981
*** Destructor called for 5/8/1992
```

Confirm Calling Constructors and Destructors

Birthday4.cpp

```
Birthday::Birthday() : year(0), month(0), day(0)
{
    cout << "*** Default constructor called" << endl;
}

Birthday::Birthday(int y, int m, int d) : year(y), month(m), day(d)
{
    cout << "*** Constructor called for " << *this << endl;
}

Birthday::~Birthday()
{
    cout << "*** Destructor called for " << *this << endl;
}
```

Vectors of Objects

BirthdayTester4.cpp

```
#include <iostream>
#include <vector>
#include "Birthday4.h"

int main()
{
    cout << "Creating Birthday variables ..." << endl;
    Birthday bd0;
    Birthday bd1(1981, 9, 2);
    Birthday bd2(1992, 5, 8);
```

```
Creating Birthday variables ...
*** Default constructor called
*** Constructor called for 9/2/1981
*** Constructor called for 5/8/1992
```

Vectors of Objects, *cont'd*

BirthdayTester4.cpp

```
cout << endl << "Creating Birthday vector ..." << endl;
vector<Birthday> birthdays;

cout << "...  push_back(bd0) ..." << endl;
birthdays.push_back(bd0);
cout << "...  push_back(bd1) ..." << endl;
birthdays.push_back(bd1);
cout << "...  push_back(bd2) ..." << endl;
birthdays.push_back(bd2);
```

```
Creating Birthday vector ...
...  push_back(bd0) ...
...  push_back(bd1) ...
*** Destructor called for 0/0/0
...  push_back(bd2) ...
*** Destructor called for 9/2/1981
*** Destructor called for 0/0/0
```

Oops!
Where did the destructor calls come from?

Vectors of Objects, *cont'd*

BirthdayTester4.cpp

```
    cout << endl << "Updating Birthday vector ..." << endl;
    birthdays[0].set_year(2010);
    birthdays[1].set_year(2011);
    birthdays[2].set_year(2012);

    cout << endl << "Printing Birthday variables ..." << endl;
    cout << bd0 << ", " << bd1 << ", " << bd2 << endl;

    cout << endl << "Printing Birthday vector ..." << endl;
    cout << birthdays[0] << ", " << birthdays[1] << ", "
         << birthdays[2] << endl;
```

```
Updating Birthday vector ...

Printing Birthday variables ...
0/0/0, 9/2/1981, 5/8/1992

Printing Birthday vector ...
0/0/2010, 9/2/2011, 5/8/2012
```

Vectors of Objects, *cont'd*

BirthdayTester4.cpp

```
cout << endl << "Creating pointer vector ..." << endl;
vector<Birthday *> bdptrs;
bdptrs.push_back(new Birthday());
bdptrs.push_back(new Birthday(3001, 9, 2));
bdptrs.push_back(new Birthday(3002, 5, 8));

cout << endl << "Printing pointer vector ..." << endl;
cout << *bdptrs[0] << ", " << *bdptrs[1] << ", "
     << *bdptrs[2] << endl;
```

```
Creating pointer vector ...
*** Default constructor called
*** Constructor called for 9/2/3001
*** Constructor called for 5/8/3002

Printing pointer vector ...
0/0/0, 9/2/3001, 5/8/3002
```

Vectors of Objects, *cont'd*

BirthdayTester4.cpp

```
    cout << endl << "Deleting birthdays from pointer vector ..."
         << endl;
    for (int i = 0; i < bdptrs.size(); i++) delete bdptrs[i];
    cout << "Done deleting from pointer vector!"
         << endl << endl;
}
```

```
Deleting birthdays from pointer vector ...
*** Destructor called for 0/0/0
*** Destructor called for 9/2/3001
*** Destructor called for 5/8/3002
Done deleting from pointer vector!

*** Destructor called for 5/8/2012
*** Destructor called for 9/2/2011
*** Destructor called for 0/0/2010
*** Destructor called for 5/8/1992
*** Destructor called for 9/2/1981
*** Destructor called for 0/0/0
```

Can you justify all the destructor calls?

Vectors of Objects, *cont'd*

BirthdayTester4.cpp

```
cout << endl << "Creating Birthday vector ..." << endl;
vector<Birthday> birthdays;

cout << "...  push_back(bd0) ..." << endl;
birthdays.push_back(bd0);
cout << "...  push_back(bd1) ..." << endl;
birthdays.push_back(bd1);
cout << "...  push_back(bd2) ..." << endl;
birthdays.push_back(bd2);
```

```
Creating Birthday vector ...
...  push_back(bd0) ...
...  push_back(bd1) ...
*** Destructor called for 0/0/0
...  push_back(bd2) ...
*** Destructor called for 9/2/1981
*** Destructor called for 0/0/0
```

Oops!
Where did the destructor calls come from?

Vectors of Objects, *cont'd*

BirthdayTester4.cpp

```
cout << endl << "Creating Birthday vector ..." << endl;
vector<Birthday> birthdays;
birthdays.reserve(10);

cout << "...  push_back(bd0) ..." << endl;
birthdays.push_back(bd0);
cout << "...  push_back(bd1) ..." << endl;
birthdays.push_back(bd1);
cout << "...  push_back(bd2) ..." << endl;
birthdays.push_back(bd2);
```

```
Creating Birthday vector ...
...  push_back(bd0) ...
...  push_back(bd1) ...
...  push_back(bd2) ...
```

Quiz and Break

- Canvas: Quizzes/Quiz 4 – 2017 Sep 28
  - 30 minutes until
- Come back at

Copy Constructor

- Every class has a copy constructor.
  - C++ supplies a default copy constructor.
  - It may not do what you want, so you can write one.
- A copy constructor has only one parameter, a constant reference to the same class.
- A copy constructor is called when:
  - A new object is created and initialized using another object of the same type.
  - An object is passed by value to a function.
  - An object is returned by a function.

Copy Constructor, *cont'd*

Birthday5.h

```
class Birthday
{
public:
    // Constructors
    Birthday();
    Birthday(int y, int m, int d);
    Birthday(const Birthday& bd);  // copy constructor
    ...
}
```

Copy Constructor, *cont'd*

Birthday5.cpp

```
Birthday::Birthday() : year(0), month(0), day(0)
{
    cout << "*** Default constructor called @ " << this << endl;
}

Birthday::Birthday(int y, int m, int d) : year(y), month(m), day(d)
{
    cout << "*** Constructor called for " << *this << " @ "<< this << endl;
}

Birthday::Birthday(const Birthday& bd)
{
    cout << "*** Copy constructor called for " << bd << " @ "<< this << endl;
    *this = bd;
}

Birthday::~Birthday()
{
    cout << "*** Destructor called for " << *this << " @ "<< this << endl;
}
```

Copy Constructor, *cont'd*

BirthdayTester5.cpp

```
int main()
{
    cout << "Creating Birthday variables ..." << endl;
    Birthday bd0;
    Birthday bd1(1981, 9, 2);
    Birthday bd2(1992, 5, 8);
```

```
Creating Birthday variables ...
*** Default constructor called @ 0x7fff4fd160e0
*** Constructor called for 9/2/1981 @ 0x7fff4fd160d0
*** Constructor called for 5/8/1992 @ 0x7fff4fd160b8
```

Copy Constructor, *cont'd*

BirthdayTester5.cpp

```
cout << endl << "Creating Birthday vector ..." << endl;
vector<Birthday> birthdays;

cout << "...  push_back(bd0) ..." << endl;
birthdays.push_back(bd0);
cout << "...  push_back(bd1) ..." << endl;
birthdays.push_back(bd1);
cout << "...  push_back(bd2) ..." << endl;
birthdays.push_back(bd2);
```

```
Creating Birthday vector ...
...  push_back(bd0) ...
*** Copy constructor called for 0/0/0 @ 0x7fb672402550
...  push_back(bd1) ...
*** Copy constructor called for 9/2/1981 @ 0x7fb67240256c
*** Copy constructor called for 0/0/0 @ 0x7fb672402560
*** Destructor called for 0/0/0 @ 0x7fb672402550
...  push_back(bd2) ...
*** Copy constructor called for 5/8/1992 @ 0x7fb672402598
*** Copy constructor called for 9/2/1981 @ 0x7fb67240258c
*** Copy constructor called for 0/0/0 @ 0x7fb672402580
*** Destructor called for 9/2/1981 @ 0x7fb67240256c
*** Destructor called for 0/0/0 @ 0x7fb672402560
```

Wow! Where did all those extra constructor and destructor calls come from?

San José State University

Copy Constructor, *cont'd*

BirthdayTester5.cpp

```
cout << endl << "Creating pointer vector ..." << endl;
vector<Birthday *> bdptrs;
bdptrs.push_back(new Birthday());
bdptrs.push_back(new Birthday(3001, 9, 2));
bdptrs.push_back(new Birthday(3002, 5, 8));
```

```
Creating pointer vector ...
*** Default constructor called @ 0x7fb672402550
*** Constructor called for 9/2/3001 @ 0x7fb672600000
*** Constructor called for 5/8/3002 @ 0x7fb672600020
```

Copy Constructor, *cont'd*

BirthdayTester5.cpp

```
cout << endl << "Deleting birthdays from pointer vector ..."
     << endl;
for (int i = 0; i < bdptrs.size(); i++) delete bdptrs[i];
cout << "Done deleting from pointer vector!" << endl << endl;
```

```
Deleting birthdays from pointer vector ...
*** Destructor called for 0/0/0 @ 0x7fb672402550
*** Destructor called for 9/2/3001 @ 0x7fb672600000
*** Destructor called for 5/8/3002 @ 0x7fb672600020
Done deleting from pointer vector!

*** Destructor called for 5/8/2012 @ 0x7fb672402598
*** Destructor called for 9/2/2011 @ 0x7fb67240258c
*** Destructor called for 0/0/2010 @ 0x7fb672402580
*** Destructor called for 5/8/1992 @ 0x7fff4fd160b8
*** Destructor called for 9/2/1981 @ 0x7fff4fd160d0
*** Destructor called for 0/0/0 @ 0x7fff4fd160e0
```

“Extra” Constructor and Destructor Calls

- Why is my program running so slowly?
- C++ does many operations “behind your back”.
- You may not expect “extra” calls to constructors and destructors.

Copy Constructor, *cont'd*

BirthdayTester5.cpp

```
cout << endl << "Creating Birthday vector ..." << endl;
vector<Birthday> birthdays;

cout << "...  push_back(bd0) ..." << endl;
birthdays.push_back(bd0);
cout << "...  push_back(bd1) ..." << endl;
birthdays.push_back(bd1);
cout << "...  push_back(bd2) ..." << endl;
birthdays.push_back(bd2);
```

```
Creating Birthday vector ...
...  push_back(bd0) ...
*** Copy constructor called for 0/0/0 @ 0x7fb672402550
...  push_back(bd1) ...
*** Copy constructor called for 9/2/1981 @ 0x7fb67240256c
*** Copy constructor called for 0/0/0 @ 0x7fb672402560
*** Destructor called for 0/0/0 @ 0x7fb672402550
...  push_back(bd2) ...
*** Copy constructor called for 5/8/1992 @ 0x7fb672402598
*** Copy constructor called for 9/2/1981 @ 0x7fb67240258c
*** Copy constructor called for 0/0/0 @ 0x7fb672402580
*** Destructor called for 9/2/1981 @ 0x7fb67240256c
*** Destructor called for 0/0/0 @ 0x7fb672402560
```

Wow! Where did all those extra constructor and destructor calls come from?

San José State University

“Extra” Constructor and Destructor Calls, *cont’d*

BirthdayTester5.cpp

```
cout << endl << "Creating Birthday vector ..." << endl;
vector<Birthday> birthdays;
birthdays.reserve(10);
```

```
Creating Birthday vector ...
...  push_back(bd0) ...
*** Copy constructor called for 0/0/0 @ 0x7f8359c02550
...  push_back(bd1) ...
*** Copy constructor called for 9/2/1981 @ 0x7f8359c0255c
...  push_back(bd2) ...
*** Copy constructor called for 5/8/1992 @ 0x7f8359c02568
```

How a Vector Grows

- When a vector needs to grow in order to insert or append more elements, C++ doesn't simply lengthen the vector in place.
- Instead, C++ allocates a new, longer vector and copies the elements from the old vector to the new vector.
- Therefore, "extra" copy constructor calls to populate the new vector and "extra" destructor calls to deallocate the old vector.

Namespaces

- A namespace is a collection of identifiers.
  - Names of variables, functions, classes, etc.
- When we use a namespace, it opens a scope for those identifiers.
  - In other words, we can use those names.
  - Example:

```
using namespace std;
```

Now we can use the names in the standard namespace.

Namespaces, *cont'd*

- When have separate compilations, different programmers can write different source files.
- How do we ensure that names used by one programmer do not conflict with names used by another programmer?
- Each programmer can define his or her own namespace and put names into it.

Namespaces, *cont'd*

```
namespace rons_namespace
{
    void function foo();
    ...
}
```

- If another programmer wants to use names defined in **`rons_namespace`**:

```
using namespace rons_namespace;
```

  - Use **`rons_namespace`** in subsequent code.

Namespaces, *cont'd*

```
namespace rons_namespace
{
    void function foo();
    ...
}
```

- Use the scope resolution operator `::` to use only a specific name from a namespace.
  - Example:

```
rons_namespace::foo();
```

- Also:

```
using rons_namespace::foo;
...
foo();
```

Search an Array: Linear Search

- Search for a value in an array of *n* elements.
  - The array is not sorted in any way.
- What choices do we have?
  - Look at all the elements one at a time.
- On average, you have to examine half of the array.

Search an Array: Binary Search

- Now assume the array is sorted.
  - Smallest value to largest value.
- First check the middle element.
- Is the target value you're looking for smaller than the middle element?
  - If so, search the first half of the array.
- Is the target value you're looking for larger than the middle element?
  - If so, search the second half of the array.

Binary Search, *cont'd*

- The binary search keeps cutting in half the part of the array it's searching.
  - Next search either the first half or the second half.
  - Eventually, you'll either find the target value, or conclude that the value is not in the array.
- The order of growth of the number of steps in a binary search is expressed $O(\log_2 n)$ Big-O notation
  - To search 1000 elements, it takes < 10 steps.
  - Computer science logarithms are base 2 by default.

Iterative Binary Search

- It's easy to write an iterative binary search:

```
int search(int value, vector<int> v, int low, int high)
{
    while (low <= high) {
        int mid = (low + high)/2;

        if (value == v[mid]) {
            return mid;
        }
        else if (value < v[mid]) {
            high = mid-1;
        }
        else {
            low = mid+1;
        }
    }

    return -1;
}
```

Get the midpoint of the subrange.

Found the target value?

Search the first half next.

Search the second half next.

The target value is not in the array.

BinarySearchIterative.cpp

Assignment #6. Book Catalog

- Create a catalog of book records (objects) as a vector sorted by ISBN.
- Insert new books into the correct positions of the catalog .
- Remove books from the catalog.
- Search for books by ISBN, category, and author.
  - Use linear and binary searches.
- Print reports of books by category or by author.

Assignment #6. Book Catalog, *cont'd*

- Keyboard input formats:
  - Insert a new book into the catalog:

    + *ISBN* , *lastname* , *firstname* , *title*,*category*

    Comma-separated values (CSV)

  - Remove a book from the catalog:

    – *ISBN*

  - Print all the book records sorted by ISBN:

    ?

Valid categories:
- fiction
- history
- technical

Assignment #6. Book Catalog, *cont'd*

- Print all the book records in sorted order that match the search criteria:

```
? isbn=ISBN
? category=category
? author=last name
```

Prompt: `Command:`

Binary searches by ISBN.
Linear searches by category and by author's last name.

- Overload the >> and << operators to facilitate reading and writing book records.

- Due Thursday, October 5
  - Assignment write-up and input data to come.