CMPE 180-92

Data Structures and Algorithms in C++

August 24 Class Meeting

Department of Computer Engineering
San Jose State University

Fall 2017

Instructor: Ron Mak

www.cs.sjsu.edu/~mak

Basic Info

- Office hours
  - TuTh 3:00 – 4:00 PM
  - ENG 250
- Website
  - Faculty webpage: http://www.cs.sjsu.edu/~mak/
  - Class webpage: http://www.cs.sjsu.edu/~mak/CMPE180-92/
  - Syllabus
  - Assignments
  - Lecture notes

Permission Codes?

- If you need a permission code to enroll in this class, see the department's instructions at https://cmpe.sjsu.edu/content/Undergraduate-Permission-Number-Requests

- Complete the Google form at https://docs.google.com/a/sjsu.edu/forms/d/e/1FAIpQLSe9YgAea-QsgLZof-KIMmuQthoChL4micudyRukgWneiByN2A/viewform

Course Objectives

- The primary goal of this class is to learn a useful subset of C++ programming language and fundamental data structures and algorithms expressed in C++.
- You will learn best practices for developing software.
- You will acquire software development skills that are valued by employers.

Course Objectives, *cont'd*

- Not course objectives:
  - Complete knowledge of C++
    - We will briefly touch the new features of C++ 11 and 14.
  - Advanced data structures and algorithms
  - Advanced algorithm analysis

C++ Tutoring

- We hope to provide C++ tutoring during the week by the instructional student assistants (ISAs).
  - Students have found this very helpful.
  - Please take advantage of this service!
- ISAs and their schedules to be announced.

Required Textbooks

- **Problem Solving with C++**, 10^{th} edition
  - Author: Walter Savitch
  - Publisher: Pearson, 2017
  - ISBN: 978-0134448282

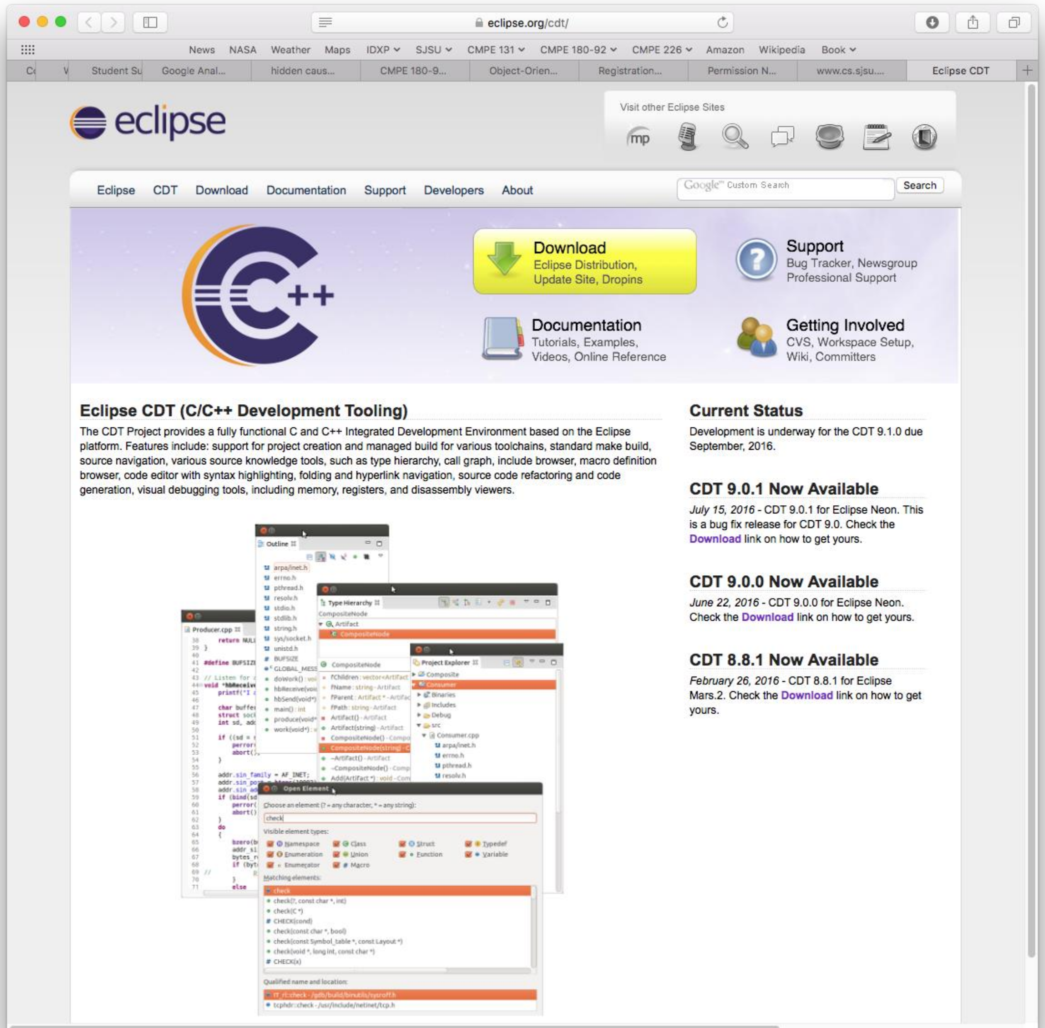
- **Data Structures Using C++**, 2^{nd} edition
  - Author: D.S. Malik
  - Publisher: Cengage Learning, 2010
  - ISBN: 978-0324782011

You are responsible for doing the chapter readings before each class, as indicated in the class schedule. In-class quizzes will be based on the readings.

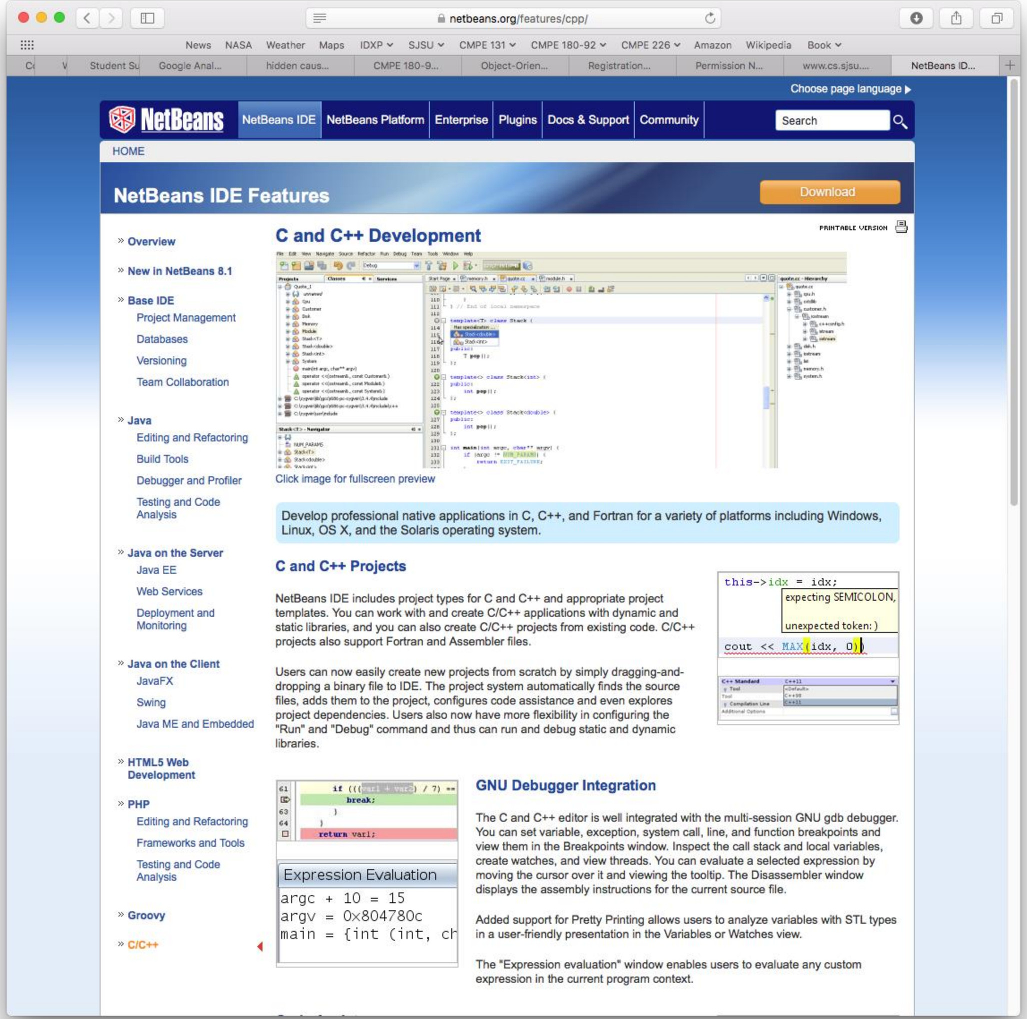
Software to Install

- Install one of the following integrated development environments (IDE) for C++ development on the Mac or Linux platform:
  - Eclipse CDT (C/C++ Development Tooling): https://eclipse.org/cdt/
  - NetBeans C and C++ Development: https://netbeans.org/features/cpp/

Software to Install, *cont'd*



Software to Install, *cont'd*



C++ on the Mac and Linux Platforms

- GNU C++ is usually pre-installed on the Mac and Linux platforms.
- No further action required!
- Avoid using Apple's Xcode on the Mac for this class.
  - You run the risk of writing programs that will not port to other platforms.

C++ on Windows

- The Windows platform has proven to be problematic for this class.
  - Difficult to install the Cygwin environment correctly.
  - Difficult to install C++ libraries successfully.
- Avoid using Microsoft's Visual C++ on Windows for this class.
  - You run the risk of writing programs that will not port to other platforms.

C++ on Windows, *cont'd*

- Run Linux in a virtual machine on Windows.
- Use Linux's pre-installed GNU C++ environment.

We will not provide support for Windows.

If you insist on running Windows,
you are on your own!

C++ on Windows, *cont'd*

- Steps:

1. Download and install the VirtualBox virtualizer: https://www.virtualbox.org/wiki/VirtualBox
2. Start VirtualBox.
3. Download a Linux .iso image (such as Debian, https://www.debian.org) and install it inside VirtualBox.
4. Start Linux from inside VirtualBox.
5. Download and install Eclipse or NetBeans on Linux.

More detailed VirtualBox instructions to come.

C++ 2011 Standard

- We will use the 2011 standard of C++.
- You must set this standard explicitly for your project in Eclipse and in NetBeans.
- On the command line:

```
g++ foo.cpp -std=c++11 -o foo
```

Set the C++ 2011 Standard in Eclipse

- Right-click on your project in the project list at the left side of the window.
- Select “Properties” from the drop-down context menu.
- In the left side of the properties window, select “C/C++ Build” ➔ “Settings”.
- In the Settings dialog, select “GCC C++ Compiler” ➔ “Dialect”.
- For “Language standard” select “ISO C++ 11”.
- Click the “Apply” button, answer “Yes”, and then click the “OK” button.

Set the C++ 2011 Standard in NetBeans

- Right-click on your project in the project list at the left side of the window.
- Select “Properties” from the drop-down context menu.
- In the left side of the properties window, select “Build” ➔ “C++ Compiler”.
- In the table, for “C++ Standard” select “C++11”.
- Click the “Apply” button and then click the “OK” button.

Assignments

- You will get lots of programming practice!
  - Multiple programming assignments per week.
  - Several small practice problems that emphasize specific skill needed to solve the main assignment.

- We will use the online CodeCheck system which will automatically check your output against a master.
  - You will be provided the URL for each assignment.
  - You can submit as many times as necessary to get the correct output.

Assignments, *cont'd*

- Assignments will be due the following week, before the next lecture.
- Solutions will be discussed at the next lecture.
- Assignments will not be accepted after solutions have been discussed in class.
  - Late assignments will receive a 0 score.

Individual Work

- You may study together.
- You may discuss the assignments together.
- But whatever you turn in must be your individual work.

Academic Integrity

- Copying another student's work or sharing your work is a violation of academic integrity.

- Violations will result in harsh penalties by the university.
  - Academic probation.
  - Disqualified for TA positions in the university.
  - Lose internship and OPT sponsorship at local companies.

- Instructors must report violations.

Moss

- Department policy is for programming assignments to be run through Stanford University's Moss application.
  - <u>M</u>easure <u>o</u>f <u>s</u>oftware <u>s</u>imilarity
  - Detects plagiarism
  - http://theory.stanford.edu/~aiken/moss/
- Moss is <u>not</u> fooled by
  - Renaming variables and functions
  - Reformatting code
  - Re-ordering functions

Example Moss output:
http://www.cs.sjsu.edu/~mak/Moss/

Quizzes

- In-class quizzes check your understanding of:
  - the required readings
  - the lectures

- Quizzes will be conducted online using Canvas.
  - Each quiz will be open for only a very short time period, around 15 minutes.
  - You are responsible for bringing a laptop or mobile device to class that can connect to the wireless.

- There will be no make-up quizzes.

Exams

- The quizzes, midterm, and final examinations will be closed book.

- Instant messaging, e-mails, texting, tweeting, file sharing, or any other forms of communication with anyone else during the exams violates academic integrity.

Exams, *cont'd*

- There can be no make-up midterm examination unless there is a documented medical emergency.
- Make-up final examinations are available only under conditions dictated by University regulations.

Final Class Grade

- 50% assignments
- 15% quizzes
- 15% midterm
- 20% final exam

- The class is graded CR/NC.

- Students who have a weighted score above the passing threshold at the end of the semester will receive the CR grade.
  - We expect least 75% of students will pass.

Fast Pace!

- This class will move forward at a fast pace.
- Lectures will consist of:
  - New PowerPoint slides by the instructor
  - PowerPoint slides from the textbook publishers
  - Program examples and live demos
  - In-class quizzes
  - Questions, answers, and discussion
- Lecture materials will be posted to the class webpage: http://www.cs.sjsu.edu/~mak/CMPE180-92/index.html

Piazza

- Besides Canvas, we will use Piazza.
  - Announcements
  - Online forum for discussions about the class

- You will receive an email invitation to join Piazza.
  - Sent to the email address that the university has on record for you.

What is C++

- An object-oriented programming (OOP) language.
  - Supports encapsulation, inheritance, polymorphism.
  - Based on the C language with added OOP features.
- A complex language!
  - Lots of features.
  - Somewhat arcane syntax.
  - Easy to make programming errors.
  - Things happen automatically at run time | that you may not expect.

A Useful Subset of C++

- We will only learn a useful subset of C++.
  - Very few people (<u>not</u> including your instructor) know all of the language.
  - Among professional C++ programmers, everybody knows a different subset, depending on experience, training, and application domains.
- It will be easy to stumble accidentally into an obscure language feature.
  - We'll have to figure out together what happened!

Our First C++ Program

- The infamous “Hello, world!” program.

helloworld.cpp

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

int main()
{
    cout << "Hello, world!" << endl;
    return 0;
}
```

```
~mak/CMPE180-92/programs: g++ helloworld.cpp -o helloworld
~mak/CMPE180-92/programs: ./helloworld
Hello, world!
```

Algorithms and Program Design

Savitch_ch_01.ppt: slides 57– 60

- Display 1.4
  - Compiling and Running a C++ Program
- Display 1.5
  - Preparing a C++ Program for Running
- Display 1.7
  - Program Design Process

Sample Program 1-8: Pods and Peas

Savitch_ch_01.ppt: slides 34 – 44

- Sample program 1.8

Break

Identifiers, Variables, and Keywords

- Identifiers are names.
- Variables represent values that can change.
  - Variables have names (variable identifiers).
  - Declare variables before you use them.
    - A declaration tells what is the variable's type (integer, real, character, etc.).
    - A declaration can also give an initial value to the variable.
- Keywords are reserved by C++ and cannot be used as identifiers.
  - Examples: `if for while`

Assignment Statements

- At run time, be sure to initialize a variable (give it a value) before you use it.
  - Either initialize the variable when you declare it.
    - Example: `int i = 5;`
  - Or execute an assignment statement.
    - Example: `i = 10;`
- Do not confuse **=** (assignment) with **==** (equality comparison).

```
i = 10;         // assign the value of 10 to variable i
if (i == 10)    // test whether or not i is equal to 10
```

Input and Output

- Input stream
  - Data read by the program at run time.
  - Standard input stream: `cin` (default: the keyboard).
  - Example:

```
cin >> x >> y;
```

extraction operator

    - Extract (read) the next two values from the keyboard and assign them to `x` and `y`, respectively.

- Output stream
  - Written by the program at run time.
  - Standard output stream: `cout` (default: the display).
  - Example:

```
cout << "x equals " << x << endl;
```

    - Insert (write) to the display.

insertion operator

`#include` and `using namespace`

- `#include <iostream>`
  - Read in the definitions of `cin` and `cout`.
- `using namespace std;`
  - Make the names `cin` and `cout` that reside in the <u>standard namespace</u> `std` available to the program.
  - Many other names reside in the standard namespace.

Formatting Real Numbers for Output

- Call methods of `cout` to format real numbers.
- `cout.setf(ios::fixed);`
  - Use fixed-point notation (not scientific).
- `cout.setf(ios::showpoint);`
  - Always show the decimal point.
- `cout.precision(2);`
  - How many decimal places.

Input From `cin`

- ```
  cin >> v1 >> v2 >> v3;
  ```
  - Read values into multiple variables.
  - The input values should be separated by spaces.

- The values are not read
  until you press the return key.
  - Therefore, you can backspace
    and make corrections.
  ```

Some Basic Data Types

- A data type determines
  - what kind of data values
  - what operations are allowed
- Data type **`int`** for integer values without decimal points.
  - Examples: **`0 2 45 -64`**
- Data type **`short`** for small integer values.
- Data type **`long`** for very large integer values.

Some Basic Data Types, *cont'd*

- Data type `double` for real numbers.
  - Fixed-point notation: `34.1 23.0034 -1.0 89.9`
  - Scientific notation: `3.67e17 5.89E-6 -7.23e+12`

- Data type `float` for less precision and smaller magnitude.

- Data type `char` for individual characters.
  - Examples: `'a' 'Z'`
  - Use only single quotes for character constants in a program.

Some Basic Data Types, *cont'd*

- Data type **`bool`** for the Boolean values **`true`** and **`false`**.
- The Boolean value **`false`** is stored as the integer 0.
- The Boolean value **`true`** is stored as the integer 1.

`cin` Skips Input Blanks

- The statements

```
char ch1, ch2;
cin >> ch1 >> ch2;
```

when given the input `A  B`
will set **`ch1`** to **`'A'`** and **`ch2`** to **`'B'`**.

> **`cin`** uses blanks and line feeds to <u>separate</u> input data values, but otherwise it <u>skips</u> the blanks and line feeds.

String Type

- `#include <string>`
  - Required if your program uses strings.

- Enclose string values with double quotes in your program.
  - Example: `"Hello, world!"`

- To input a string from `cin` that includes spaces, all in one line:

```
string str;
getline(cin, str);
```

Type Compatibilities and Conversions

- `int pi = 3.14;`
  - `double` → `int` is invalid. You cannot set a `double` value into an `int` variable .
- Some valid conversions:
  - `int` → `double`
  - `char` → `int`
  - `int` → `char`
  - `bool` → `int`
  - `int` → `bool`

Any nonzero integer value is stored as true.
Zero is stored as false.

Arithmetic

- Arithmetic operators: `+ - * / %`
- Integer `/` result if both operands are integer.
  - Quotient only.
- Use the modulo operator `%` to get a remainder.
- Double `/` result (includes fractional part) if either or both operands are double.

Operator Shorthand

- `n += 5` shorthand for `n = n + 5`
- `n -= 5` shorthand for `n = n - 5`
- `n *= 5` shorthand for `n = n*5`
- `n /= 5` shorthand for `n = n/5`
- `n %= 5` shorthand for `n = n%5`

The `if` Statement

- Example `if` statement:

```
if (n <= 0)
{
    cout << "Please enter a positive number." << endl;
}
```

- Example `if else` statement:

```
if (hours > 40)
{
    gross_pay = rate*40 + 1.5*rate*(hours - 40);
}
else
{
    gross_pay = rate*hours;
}
```

while Loops

- Example while loop:

```
while (count_down > 0)
{
    cout << "Hello ";
    count_down = count_down - 1;
}
```

- Example do while loop:

```
do
{
    cout << "Hello ";
    count_down = count_down - 1;
} while (count_down > 0)
```

Named Constants

- It’s good programming practice
  to give names to constants:

```
const double PI = 3.1415626;
```

- Easier for humans to read the program.
- Easier to modify the program.

- Convention: Use ALL_CAPS
  for the names of constants.

Boolean Operators

- Relational operators: `==  !=  <  <=  >  >=`
- And: `&&`
- Or: `||`
- Not: `!`

- <u>Short-circuit</u> operation: `p && q`
  - `q` is not evaluated if `p` is false

- <u>Short-circuit</u> operation: `p || q`
  - `q` is not evaluated if `p` is true

Precedence Rules

Savitch_ch_02.ppt: slide 101

Precedence Rules

The unary operators +, –, ++, – –, and ! .

The binary arithmetic operations *, /, %

The binary arithmetic operations +, –

The Boolean operations <, >, <=, >=

The Boolean operations ==, !=

The Boolean operations &&

The Boolean operations ||

Highest precedence (done first)

Lowest precedence (done last)

Enumeration Types

- A data type with values defined by a list of constants of type `int`
  - Examples:

```
enum Direction {NORTH, SOUTH, EAST, WEST};

enum MonthLength{JAN_LENGTH = 31,
                 FEB_LENGTH = 28,
                 MAR_LENGTH = 31,
                   …
                 DEC_LENGTH = 31};
```

Nested `if` Statements

- Example:

```
if (net_income <= 15000)
{
    tax_bill = 0;
}
else if ((net_income > 15000) && (net_income <= 25000))
{
    tax_bill = (0.05*(net_income - 15000));
}
else // net_income > $25,000
{
    five_percent_tax = 0.05*10000;
    ten_percent_tax = 0.10*(net_income - 25000);
    tax_bill = (five_percent_tax + ten_percent_tax);
}
```

The `switch` Statement

- Use a **`switch`** statement instead of nested **`if`** statements to compare a single integer value for equality.
  - Note the need for the **`break`** statements.
  - Note the **`default`** case at the bottom.

```
int digit;
...
switch(digit)
{
    case 1: digit_name = "one";   break;
    case 2: digit_name = "two";   break;
    case 3: digit_name = "three"; break;
    case 4: digit_name = "four";  break;
    case 5: digit_name = "five";  break;
    case 6: digit_name = "six";   break;
    case 7: digit_name = "seven"; break;
    case 8: digit_name = "eight"; break;
    case 9: digit_name = "nine";  break;

    default: digit_name = ""; break;
}
```

The Increment and Decrement Operators

- **`++n`**
  - Increase the value of **`n`** by 1.
  - Use the increased value.

- **`n++`**
  - Increase the value of **`n`** by 1.
  - Use the value before the increase.

The Increment and Decrement Operators, *cont'd*

- `--n`
  - Decrease the value of `n` by 1.
  - Use the decreased value.

- `n--`
  - Decrease the value of `n` by 1.
  - Use the value before the decrease.

`for` Loops

- Example:

```
int sum = 0;

for (int n = 1; n <= 10; n++)
{
    sum = sum + n;
}

cout << "The sum of the numbers 1 to 10 is "
     << sum << endl;
```

Note that variable `n` is local to the loop body.

for Loops, *cont'd*

- The **for** loop uses the same components as the **while** loop, but in a more compact form.

```
for (n = 1; n <= 10; n++)
```

Initialization Action

Update Action

Boolean Expression

The `break` Statement

- Use the `break` statement to exit a loop before “normal” termination.
- Do not overuse!
  - Well-designed loops should end normally.
- This use of `break` is different from the necessary use of `break` in a `switch` statement.

Loop Considerations

- Choosing the right kind of loop to use
- Designing loops
- How to control a loop
- How to exit from a loop
- Nested loops
- Debugging loops

Practice Problems for Week 1

- Five problems in CodeCheck:
  - https://play.codecheck.ws/files?repo=fall2017&problem=w1-1
  - https://play.codecheck.ws/files?repo=fall2017&problem=w1-2
  - https://play.codecheck.ws/files?repo=fall2017&problem=w1-3
  - https://play.codecheck.ws/files?repo=fall2017&problem=w1-4
  - https://play.codecheck.ws/files?repo=fall2017&problem=w1-5
- Submit as many times as you need to get correct results for each problem.
  - No penalties for multiple submissions.
- Download all five signed zip files and submit them together into Canvas: **Week 1 Practice**
  - Do not rename the zip files.

Main Assignment for Week 1

- Assignment #1 will give you practice with C++ control statements and nested loops.
  - Write-up:
  - Input file:
  - CodeCheck URL: http://codecheck.it/codecheck/files/17082218369vfzb082gwl1ggr02jbdp0cn6

- Follow carefully the instructions on how to use CodeCheck and how to submit into Canvas.

CMPE 180-92

Data Structures and Algorithms in C++

August 31 Class Meeting

Department of Computer Engineering
San Jose State University

Fall 2017

Instructor: Ron Mak

www.cs.sjsu.edu/~mak

Basic Info

- Office hours
  - TuTh 3:00 – 4:00 PM
  - ENG 250
- Website
  - Faculty webpage: http://www.cs.sjsu.edu/~mak/
  - Class webpage: http://www.cs.sjsu.edu/~mak/CMPE180-92/
  - Syllabus
  - Assignments
  - Lecture notes

Assignment #1: Sample Solution

- First start with a "draft" of your program.
  - Test that you can read the individual fields of the input file.

- Then incrementally add to the draft until you have the complete solution.
  - Always build on working code.

- Do not attempt to write an entire program all at once and then try to get it to work.

Predefined Functions

- C++ includes predefined functions.
  - AKA “built-in” functions
  - Example: Math function `sqrt`
- Predefined functions are stored in libraries.
  - Your program will need to include the appropriate library header files to enable the compiler to recognize the names of the predefined functions.
  - Example: `#include <cmath>`
    in order to use predefined math functions like `sqrt`

Predefined Functions, *cont'd*

Savitch_ch_04.ppt: slides 8 – 12, 72

Some Predefined Functions

Name	Description	Type of Arguments	Type of Value Returned	Example	Value	Library Header
`sqrt`	square root	*double*	*double*	`sqrt(4.0)`	2.0	`cmath`
`pow`	powers	*double*	*double*	`pow(2.0,3.0)`	8.0	`cmath`
`abs`	absolute value for *int*	*int*	*int*	`abs(-7)` `abs(7)`	7 7	`cstdlib`
`labs`	absolute value for *long*	*long*	*long*	`labs(-70000)` `labs(70000)`	70000 70000	`cstdlib`
`fabs`	absolute value for *double*	*double*	*double*	`fabs(-7.5)` `fabs(7.5)`	7.5 7.5	`cmath`
`ceil`	ceiling (round up)	*double*	*double*	`ceil(3.2)` `ceil(3.9)`	4.0 4.0	`cmath`
`floor`	floor (round down)	*double*	*double*	`floor(3.2)` `floor(3.9)`	3.0 3.0	`cmath`

Random Numbers

- To generate (pseudo-) random numbers using the predefined functions, first include two library header files:

```
#include <cstdlib>
#include <ctime>
```

- "Seed" the random number generator:

```
srand(time(0));
```

  - If you don't seed, you'll always get the same "random" sequence.

Random Numbers, *cont'd*

- Each subsequent call

```
rand();
```

returns a "random" number ≥ 0
and < **RAND_MAX**.

- Use **+** and **%** to scale to a desired number range.
  - Example: Each execution of the expression

```
rand()%6 + 1
```

returns a random number
with the value 1, 2, 3, 4, 5, or 6.

Type Casting

- Suppose integer variables `i` and `j` are initialized to 5 and 2, respectively.
- What is the value of the division `i/j` ?
- What if we wanted to have a quotient of type double?
  - We want to keep the fraction.

Type Casting, *cont'd*

- One way is to convert one of the operands (say `i`) to double.
  - Then the quotient will be type double.

```
double quotient = static_cast<double>(i)/j;
```

- Why won't the following work?

```
double quotient = static_cast<double>(i/j);
```

Programmer-Defined Functions

- In addition to using the predefined functions, you can write your own functions.
- Programmer-defined functions are critical for good program design.
- In your C++ program, you can call a programmer-defined function only after the function has been declared or defined.

Function Declarations

- A function declaration specifies:
  - The function name.
  - The number, order, and data types of its formal parameters.
  - The data type of its return value.

- Example:

```
double total_cost(double unit_cost, int count);
```

Function Definitions

- After you've declared a function, you must define it.
  - Write the code that is executed whenever the function is called.
  - A `return` statement terminates execution of the function and returns a value to the caller.

- Example:

```
double total_cost(double unit_cost, int count)
{
    double total = count*unit_cost;
    return total;
}
```

Function Calls

- Call a function that you wrote just as you would call a predefined function.

- Example:

```
int how_many;
double how_much;
double spent;

how_many = 5;
how_much = 29.99;
spent = total_cost(how_much, how_many);
```

Void Functions

- A void function performs some task but does not return a value.
- Therefore, its `return` statement terminates the function execution but does not include a value.
  - A return statement is not necessary for a void function if the function terminates "naturally" after it finishes executing the last statement.
- Example void function definition:

```
void print_TF(bool b)
{
    if (b) cout << "T";
    else   cout << "F";
}
```

Void Functions, *cont'd*

- A call to a void function cannot be part of an expression, since the function doesn't return a value.
- Instead, call a void function as a statement by itself.
- Example:

```
bool flag = true;
print_TF(flag);
```

Top-Down Design, *cont'd*

- Top-down design is an important software engineering principle.
- Start with the topmost subproblem of a programming problem.
  - Write a function for solving the topmost subproblem.
- Break each subproblem into smaller subproblems.
  - Write a function to solve each subproblem.
  - This process is called stepwise refinement.

Top-Down Design, *cont'd*

- The result is a hierarchical decomposition of the problem.
- AKA functional decomposition

Top-Down Design Example

- Write a program that inputs from the user that are positive integer values less than 1000.
- Translate the value into words.
- Example:
  - The user enters 482
  - The program writes “four hundred eighty-two”
- Repeat until the user enters a value ≤ 0.

Top-Down Design Example, *cont'd*

- What is the topmost problem?
  - Read numbers entered by the user until the user enters a value ≤ 0.
  - Translate each number to words.
- How to translate a number into words?
  - Break the number into separate digits.
  - Translate the digits into words such as *one*, *two*, ..., *ten*, *eleven*, *twelve*, ..., *twenty*, *thirty*, etc.

Refinement 1

- Loop to read and print the numbers.
- Call a translate function,
  but it doesn't do anything yet.

A Convention for Functions

- Put function declarations before the main.
  - If you give your functions good names, the declarations show the structure of your program.
- Put function definitions after the main.

Refinement 2

- Refine the translate function to handle some simple cases:
  - `translateOnes`: 1 through 9
  - `translateTeens`: 11 through 19

Refinement 3

- The translate function takes a 3-digit number and separates out the hundreds digit.
- Translate the hundreds digit.
  - **`translateHundreds`**
  - Do this simply by translating the hundreds digits as we did a ones digit, and append the word *hundred*.

Refinement 3

- Translate the last two digits:
  - We can already translate a teens number.
  - Otherwise, break apart the two digits into a tens digit and a ones digit.
    - **`translateTens`**: 10, 20, 30, ..., 90
    - We can already translate a ones digit.

Refinement 4

- Add a hyphen between *twenty*, *thirty*, etc. and a ones word.

Refinement 5

- Break a 6-digit number into a 3-digit first part and a 3-digit second part.
- Translate the first part and append the word *thousand*.
- Translate the second part.

Break

Scope and Local Variables

- Any variable declared inside a function is local to that function.
  - The scope of the variable is that function.
  - The variable is not accessible
    from outside the function.
  - A variable with the same name declared inside another function is a different variable.

- The same is true for any variable declared inside the main function.

Block Scope

- You can declare variables inside of a block.
  - A block of code is delimited by `{` and `}`.
- The variables are local to the block.

Global Constants and Variables

- If a constant or a variable is declared outside of and before the main and the function definitions, then that constant or variable is accessible by the main and any function.
- Global variables are <u>not recommended</u>.

Overloading Function Names

- A function is characterized by both its name and its parameters.
  - Number and data types of the formal parameters.

- You can overload a function name by defining another function with the same name but different parameters.
  - When you call a function with that name, the arguments of the call determine which function you mean.

Overloading Function Names, *cont'd*

- Example declarations:

```
double average(double n1, double n2);
double average(double n1, double n2, double n3);
```

- Example calls:

```
double avg2 = average(x, y);
double avg3 = average(x, y, z);
```

- Be careful with automatic type conversions of arguments when overloading function names.
  - See the Savitch text and slides.

Call-by-Value

- By default, arguments to a function are passed by value.
- A copy of the argument's value is passed to the function.
- Any changes that the function makes to the parameters do not affect the calling arguments.
  - Example: The faulty swap function.

Call-by-Value, *cont'd*

```
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

- Why doesn't this function do what was intended?

Demo

Call-by-Reference

- If you want the function to be able to change the value of the caller's arguments, you must use call-by-reference.

- The address of the actual argument is passed to the function.
  - Example: The proper exchange function.

Call-by-Reference, *cont'd*

```
void exchange(int& a, int& b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

- Why is this code better?

```
void exchange(int& a, int& b)
```

Demo

Procedural Abstraction

- Design your function such that the caller does not need to know how you implemented it.
- The function is a “black box”.

Procedural Abstraction, *cont'd*

- The function's name, its formal parameters, and your comments should be sufficient for the caller.
- Preconditions: What must be true when the function is called.
- Postconditions: What will be true after the function completes its execution.

Testing and Debugging Functions

- There are various techniques to test and debug functions.
- You can add temporary `cout` statements in your functions to print the values of local variables to help you determine what the function is doing.
- With the Eclipse or the NetBeans IDE, you can set breakpoints, watch variables, etc.

assert

- Use the `assert` macro during development to check that a function's preconditions hold.
  - You must first `#include <cassert>`
  - Example:

```
assert(y != 0);
quotient = x/y;
```

- Later, when you are sure that your program is debugged and you are going into production, you can logically remove all the asserts by defining `NDEBUG` before the include:

```
#define NDEBUG
#include <cassert>
```

Assignment #2: Functional Decomposition

- Practice decomposing a program top-down by using functions.
- The solution for Assignment #1, as suggested by the program outline in CodeCheck, was long main containing much duplicated code.
- For Assignment #2, write a new version of the program, but this time with user-defined functions.

Assignment #2: *cont'd*

- The resulting program should have a <u>hierarchical decomposition</u>.
- Choose good function names and use parameters wisely.
- Your final program should be have correct output <u>and</u> be easy to read.
- The official assignment write-up will appear in Canvas tomorrow.

Week 2 Practice Problems

- Look for practice problems in Canvas.
- They should appear in a day or two.

CMPE 180-92

Data Structures and Algorithms in C++

September 7 Class Meeting

Department of Computer Engineering
San Jose State University

Fall 2017
Instructor: Ron Mak

www.cs.sjsu.edu/~mak

SILICON VALLEY'S FIRST CHOICE FOR NEW ENGINEERING HIRES
— Silicon Valley Business Journal, 2013

San José State UNIVERSITY

Assignment #2: Sample Solution

Streams

- I/O (input/output) for a program can be considered a stream of characters.
  - Represented in a program by a stream variable.
- An input stream into your program can be
  - characters typed at the keyboard
  - characters read from a file
- An output stream from your program can be
  - characters displayed on the screen
  - characters written to a file

File I/O

- In order for a program to read from a data file, it must first connect a stream variable to the file.

```
#include <fstream>
using namespace std;
...
ifstream in_stream;   // input  file stream variable
ofstream out_stream;  // output file stream variable
...
in_stream.open("infile.dat");     // connect to the input file
out_stream.open("outfile.dat");   // connect to the output file
...
// Read three integer values from the input file.
int value1, value2, value3;
in_stream >> value1 >> value2 >> value3;

// Write to the output file.
out_stream << "Value #1 is " << value1
           << " and Value #2 is " << value2 << endl;
```

File I/O, *cont'd*

- Close a stream when you're done with reading or writing it.

```
in_stream.close();
out_stream.close();
```

- Closing a stream releases the associated file for use by another program.

Stream Name vs. File Name

- Do not confuse the name of a program's stream variable with the name of the file.
  - The stream variable's name internal to the program.
  - The file's name is external to the program.
- Calling a stream's `open` method connects the stream to the file.
- A stream is an object.
  - `open` and `close` are functions we can call on the object.

We'll learn about C++ classes and objects later.

Formatting Output

- Formatting a value that is being output includes
  - determining the width of the output field
  - deciding whether to write numbers in fixed-point notation or in scientific notation
  - setting how many digits after the decimal point

- To format output to **`cout`**, call its member functions:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

  - Use fixed-point notation instead of scientific notation.
  - Always include the decimal point in the output.
  - Only two significant digits are required in the output.

Output Manipulators

- Manipulator function **`setw`** sets the width of an output field.
- Manipulator function **`setprecision`** sets the number of places after the decimal point.
- Embed calls to manipulators in output statements.
  - Examples:

```
#include <iomanip>
using namespace std;
...
cout << "Value 1 = " << setw(10) << value1 << end;
cout << "$" << setprecision(2) << amount << endl;
```

Passing Streams to Functions

- Pass stream objects to functions
  only via call-by-reference.
  - Example:

```
void copyFile(ifstream& source,
              ofstream& destination);
```

Character I/O

- Recall that the operator `>>` used on `cin` skips blanks.

- To read all characters from an input stream, including blanks, use the `get` method:

```
char ch;
...
cin.get(ch);
```

- Use the `put` method to output any character to an output stream.

Predefined Character Functions

- Some very useful Boolean functions that test a character:
  - `isupper(ch)`
  - `islower(ch)`
  - `isalpha(ch)`
  - `isdigit(ch)`
  - `isspace(ch)`
  - `toupper(ch)`
  - `tolower(ch)`

The **`eof`** Function

- Boolean function **`eof`** tests whether or not an input stream has read the entire file.
  - eof = end of file
  - Example: `if (in_stream.eof()) ...`
- Function **`eof`** returns true only <u>after</u> an attempt was made to read past the end of file.

Quiz

Break

Arrays

- An array variable can have multiple values.
- All values must be the same data type.

- Declare an array variable by indicating how many elements.
  - Example: `int a[6];`

- Use subscripts to access array elements.
- Subscript values for an array can range from 0 ... n-1 where n is equal to the number of elements in the array.

Initialize an Array

- You can initialize an array when you declare it:

```
int ages[] = {12, 9, 7, 2};
```

  - If you initialize an array this way, you can leave off the array size.

- You can initialize the array with assignments:

```
int ages[4];
ages[0] = 12;
ages[1] = 9;
ages[2] = 7;
ages[3] = 2;
```

- Or with a loop:

```
int ages[4];
for (int i = 0; i < 4; i++) ages[i] = 0;
```

Array Function Parameters

- To pass an entire array to a function, indicate that a parameter is an array with `[]`.
  - Example:

```
void sort(double a[], int size);
```

- Also pass the array size.
- Arrays are implicitly passed by reference.
- Make the array parameter `const` to indicate that the function does not change the array.
  - Example:

```
double average(const double a[], int size);
```

Assignment #3.a. Prime Numbers

- Use the Sieve of Eratosthenes to generate an array of prime numbers under 100:

```
Primes:

  .  2  3  .  5  .  7  .  .  .
 11  . 13  .  .  . 17  . 19  .
  .  . 23  .  .  .  .  . 29  .
 31  .  .  .  .  . 37  .  .  .
 41  . 43  .  .  . 47  .  .  .
  .  . 53  .  .  .  .  . 59  .
 61  .  .  .  .  . 67  .  .  .
 71  . 73  .  .  .  .  . 79  .
  .  . 83  .  .  .  .  . 89  .
  .  .  .  .  .  . 97  .  .  .
```

  - See: https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

Multidimensional Arrays

- A multidimensional array is an array of arrays.
  - Example: A two-dimensional array:

```
char page[30][100];
```

  - Each element of `page` is itself an array of 100 characters.

- Use multiple subscripts to access an element of a multidimensional array.
  - Example: `page[i][j]` to access the `j`th character of the `i`th row.
  - What is `page[k]`?

Assignment #3.b. Spirals

- Print a sequence of integers in a counter-clockwise spiral that is enclosed in a square matrix *n*-by-*n*.
  - The 2-dimensional array has *n* rows and *n* columns.
- Start with a given value in the center of the matrix.
  - The starting value is not necessarily 1.
- Arrange subsequent values in a counter-clockwise spiral that grows outward until it fills the matrix.

Assignment #3.b. Spirals, *cont'd*

- Example spirals
  - Size 5, starting value 1:

```
17  16  15  14  13
18   5   4   3  12
19   6   1   2  11
20   7   8   9  10
21  22  23  24  25
```

  - Size 9, starting value 11:

```
75  74  73  72  71  70  69  68  67
76  47  46  45  44  43  42  41  66
77  48  27  26  25  24  23  40  65
78  49  28  15  14  13  22  39  64
79  50  29  16  11  12  21  38  63
80  51  30  17  18  19  20  37  62
81  52  31  32  33  34  35  36  61
82  53  54  55  56  57  58  59  60
83  84  85  86  87  88  89  90  91
```

C Strings

- Traditional C programs used arrays of characters to represent strings:

```
char greeting[] = "Hello, world!";
```

- A C string is always terminated by the null character `\0`.

- Therefore, the array size was one greater than the number of characters in the string.
  - The `greeting` character array above has size 14.

C Strings, *cont'd*

- You cannot assign a string value to a C string array variable:
  - Illegal: `greeting = "Good-bye!";`
- Instead, you use the **`strcpy`** ("string copy") function: `strcpy(greeting, "Good-bye!");`
- **Warning:** Do not copy past the end of the destination string!

C Strings, *cont'd*

- To compare two C strings, use the **`strcmp`** ("string compare") function:

```
strcmp(str1, str2);
```

- It returns:
  - a negative value if **`str1`** comes alphabetically before **`str2`**
  - zero if they contain the same characters
  - a positive value if **`str1`** comes alphabetically after **`str2`**.

The Standard `string` Class

- C++ programs use the standard `string` class:

```
#include <string>
using namespace std;
```

- You can initialize `string` variables when you declare them:

```
string noun, s1, s2, s3;
string verb("go");
```

- You can assign to `string` variables:

```
noun = "computer";
```

The Standard `string` Class, *cont'd*

- String concatenation:

```
s1 = s2 + " and " + s3;
```

- String comparisons with `== != < <= > >=`
  - Lexicographic comparisons as expected.
- Strings automatically grow and shrink in size.
  - A string keeps track of its own size.
- Use the member function `at` to safely access a character of a string: `s1.at(i)`
  - `s1[i]` is dangerous if you go beyond the length.

The Standard `string` Class, *cont'd*

- Many useful member functions :
  - `str.length()`
  - `str.at(i)`
  - `str.substr(position, length)`
  - `str.insert(pos, str2)`
  - `str.erase(pos, length)`
  - `str.find(str1)`
  - `str.find(str1, pos)`
  - `str.find_first_of(str1, pos)`
  - `str.find_first_not_of(str1, pos)`

Vectors

- A vector is a kind of array whose length can dynamically grow and shrink.
  - Vectors are part of the C++ Standard Template Library (STL).

An array on steroids!

- Like an array, a vector has a base type, and all its elements are of that type.

- Different declaration syntaxes from arrays:

```
vector<double> salaries;
vector<bool> truthTable(10);
vector<int> ages = {12, 9, 7, 2};
```

Vectors, *cont'd*

- Index into a vector like an array: `ages[2]`

- Use with a standard for loop:

```
for (int i = 0; i < ages.size(); i++)
{
    cout << ages[i] << endl;
}
```

- Or with a ranged for loop:

```
for (int age : ages)
{
    cout << age << endl;
}
```

Vectors, *cont'd*

- Append new values to the end of a vector:

```
salaries.push_back(100000.0);
salaries.push_back(75000.0);
salaries.push_back(150000.0);
salaries.push_back(200000.0);
```

- Vector assignment: **`v1 = v2`**;
  - Element-by-element assignment of values.
  - The size of **`v1`** can change to match the size of **`v2`**.

Vectors, *cont'd*

- Size of a vector: The current number of elements that the vector contains: `v.size()`

- Capacity of a vector: The number of elements for which memory is currently allocated: `v.capacity()`

  - Change the size: `v.resize(24)`
  - Explicitly set the capacity: `v.reserve(32)`
  - Bump up the capacity by 10: `v.reserve(v.size() + 10)`

Assignment #3.c. Prime Spirals

- Repeat Assignment #3.b, except use vectors instead of arrays.

- Instead of printing the numbers in the spiral, print dots and hashes instead.
  - Print a hash (**#**) if the position corresponds to a prime number.
  - Print a dot (**.**) if the position corresponds to a composite number.

- Curious patterns may emerge in the matrix!

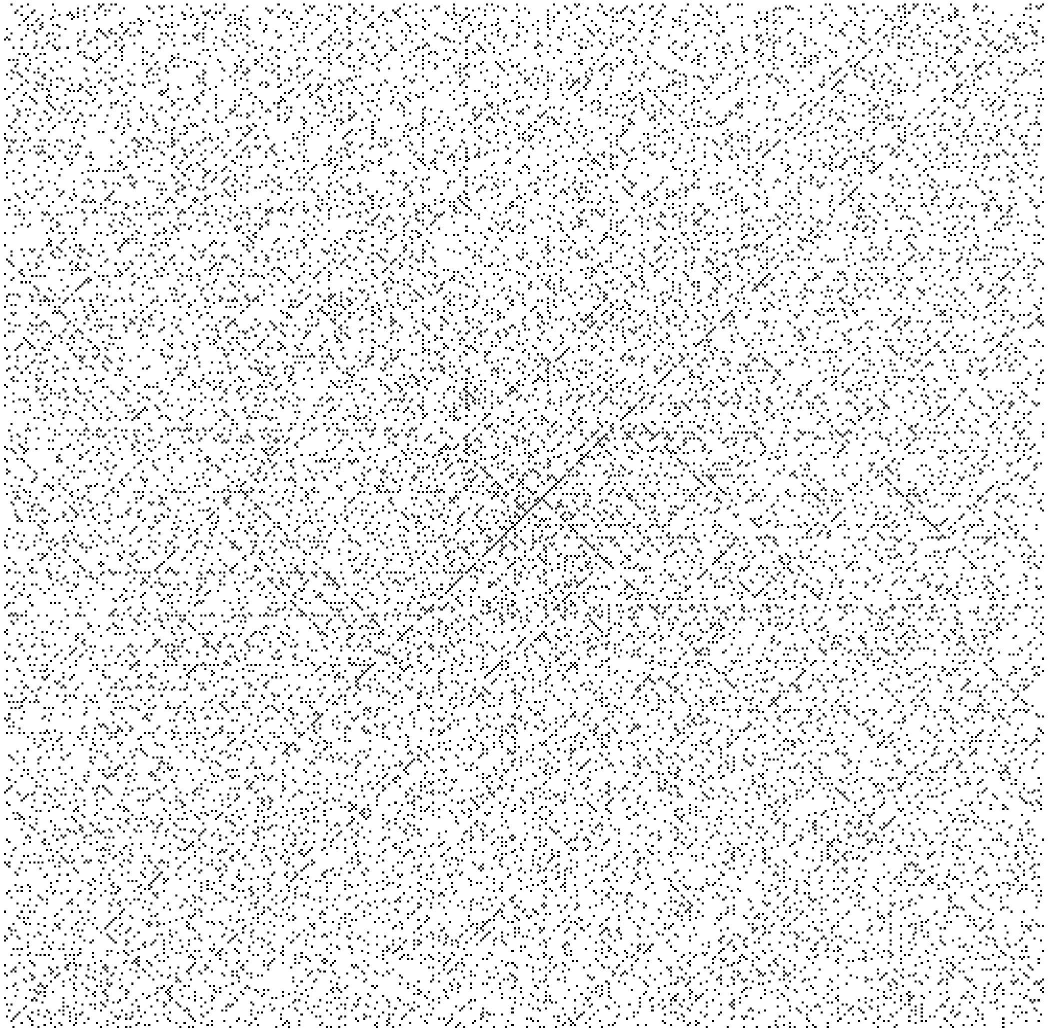
Assignment #3.c. Prime Spirals, *cont'd*

- Example
  - Size 25, starting at 11:

```
#.........#.....#.#.....#
.....#...#.......#.......
....#.......#...#........
.......#.............#...
........#.....#.....#.#..
.#.#.........#.#...#.....
#...#.....#.#...........#
.........#.#...#.#.#.#.#.
....#.#...#.#...#........
.#...#.#.#...#.#.....#...
..#...........#...#...#..
.............#...#.....#.
#.#.....#.#.#.....#......
...........#.#.#.....#.#.
#...#.#...#.....#...#.#..
.#.#.....#.....#.........
........#.....#.....#...#
...#.........#...#.#...#.
....#.#.....#.....#......
.......#...#.#...#.......
#...#.........#.......#..
...#.....#.......#.......
..#.#.....#...#.....#...#
.....#.#.................
..#...#.#...........#....
```

Assignment #3.c. Prime Spirals, *cont'd*

- Are there patterns in the prime numbers?



CMPE 180-92

Data Structures and Algorithms in C++

September 14 Class Meeting

Department of Computer Engineering
San Jose State University

Spring 2017

Instructor: Ron Mak

www.cs.sjsu.edu/~mak

Assignment #3 Sample Solutions

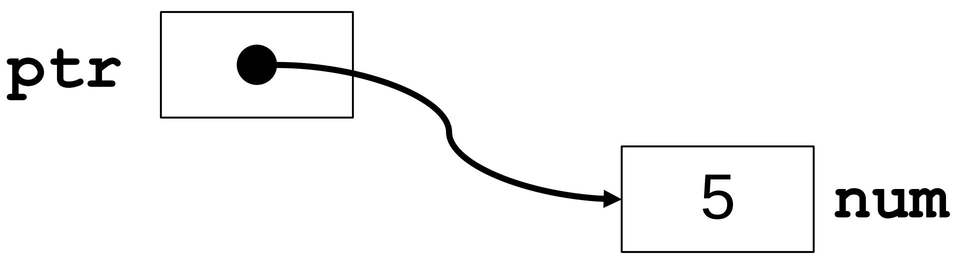
Pointers

- Pointers are an extremely powerful feature of C and C++ programs.
  - You would not be a competent C or C++ programmer if you did not know how to use pointers effectively.
- Pointers can also be extremely dangerous.
  - Many runtime errors and program crashes are due to misbehaving pointers.
  - Pointers are a prime cause of memory errors.

An `int` vs. Pointer to an `int`

- A graphical representation of an **`int`** variable named **`num`** and its value:

- A graphical representation of a pointer variable named **`ptr`** that points to an **`int`** value of a variable named **`num`**:

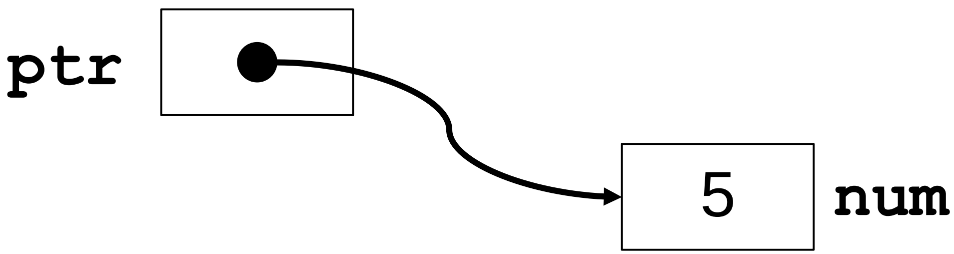


Declaring and Assigning Pointers

- After the following statements are executed:

```
int  num = 5;
int *ptr = &num;
```

- We have this situation:



Pointers are Addresses

- To declare that a variable is a pointer, use a `*` before the variable name:

```
int *ptr;
double *ptr2;
```

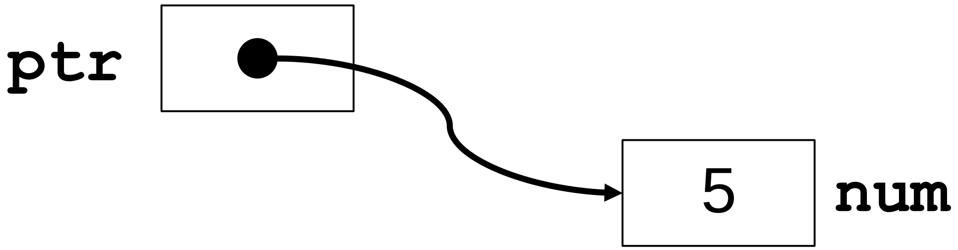
  - `ptr` can point to an `int` value
  - `ptr2` can point to a `double` value

`&` is the address-of operator

- The statement `ptr = #` assigns the address of variable `num` to pointer variable `ptr`
  - Make `ptr` point to the address of variable `num`.

The Dereferencing Operator

```
int  num = 5;
int *ptr = &num;
```



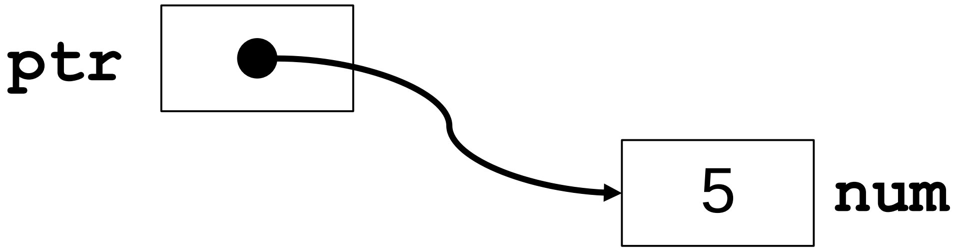
- To get the value that pointer `ptr` is pointing to:

```
*ptr
```

- Now the `*` is the dereferencing operator.
  - "Follow the pointer to get what it's pointing to."
- We can use `*ptr` in an expression.
  - Example: `*ptr + 2` gives the value 7.

The Dereferencing Operator, *cont'd*

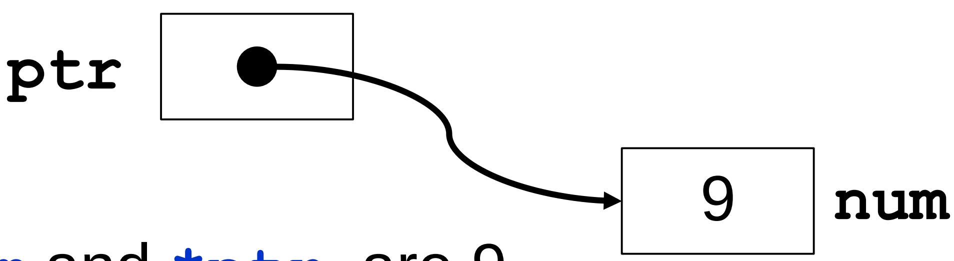
```
int  num = 5;
int *ptr = &num;
```



- In the above example, both `*ptr` and `num` refer to the same value 5.

- What happens if we execute the statement?

```
*ptr = 9;
```



  - Now both `num` and `*ptr` are 9.

A Pointer Declaration Warning

- You can declare several pointer variables in one line:

```
double *ptr1, *ptr2, *ptr3;
```

- How many pointer variables do we have?

```
double* ptr1, ptr2, ptr3;
```

  - Only `ptr1` is a pointer to a double value. `ptr2` and `ptr3` are simple double variables.

Break

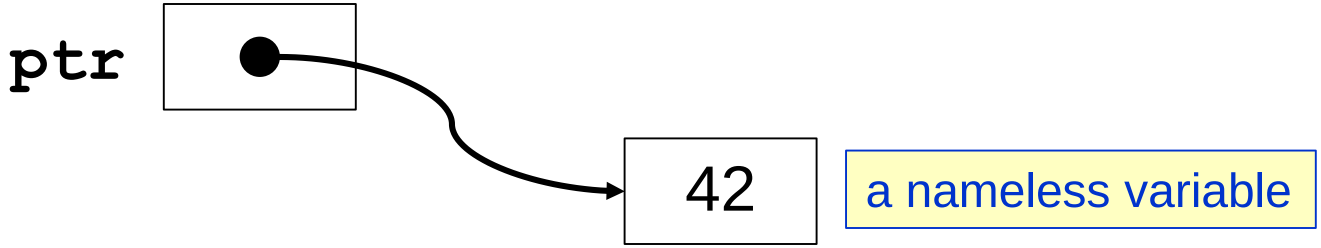
The `new` Operator

- So far, all our variables have names and are created automatically when we declare them:

```
int num;
```

- We can also create nameless variables.
  - The `new` operator returns a pointer to the variable it just created.
  - This is ideal for pointer variables.

```
int *ptr = new int(42);
```



The `delete` Operator

- If your program creates nameless variables, then it must remove them from memory when the program no longer needs them.
  - Delete from memory the nameless variable that `ptr` points to.

```
delete ptr;
```

- If your program doesn't get rid of all the nameless variables it created, those variables clutter up memory, and therefore you are said to have a memory leak.

Pointer Parameters

- We can pass a pointer by value to a function:

```
void foo(int *ptr1, double *ptr2);
```

  - We can change the value of the variable that `ptr1` points to.

- We can also pass a pointer by reference:

```
void bar(int* &ptr1, double* &ptr2);
```

  - We can change what variable `ptr1` points to.
  - Ugly syntax!

typedef

- Use `typedef`s to simplify pointer notation:

```
typedef int    *IntPtr;
typedef double *DoublePtr;
```

- Now you can use `IntPtr` in place of `int *` and `DoublePtr` in place of `double *`

```
void foo(IntPtr ptr1, DoublePtr ptr2);
```

```
void bar(IntPtr& ptr1, DoublePtr& ptr2);
```

Using Pointers to Pass-by-Reference

- C programmers used pointers to pass parameters by reference.
  - Example:

```
function baz(int *parm);
```

- A call to the function needed the address of the corresponding argument:

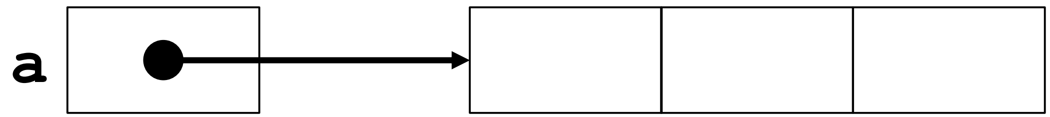
```
int arg;
baz(&arg);
```

  - Because `parm` points back to the actual argument, the function can use `*parm` to change the value of the actual argument.

Pointers and Arrays

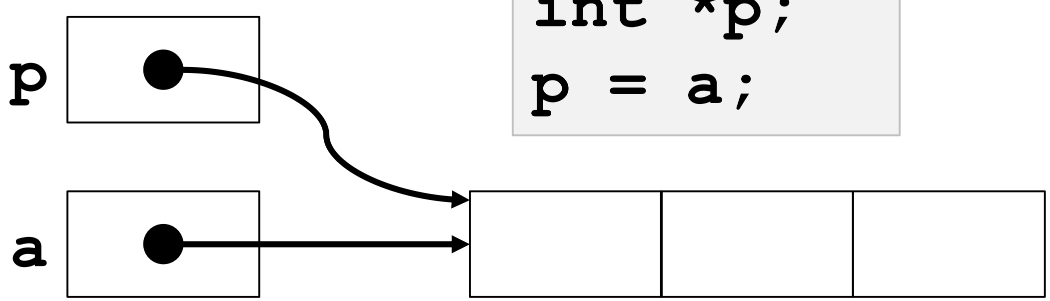
- An array variable is actually a pointer variable.

```
int a[3];
```



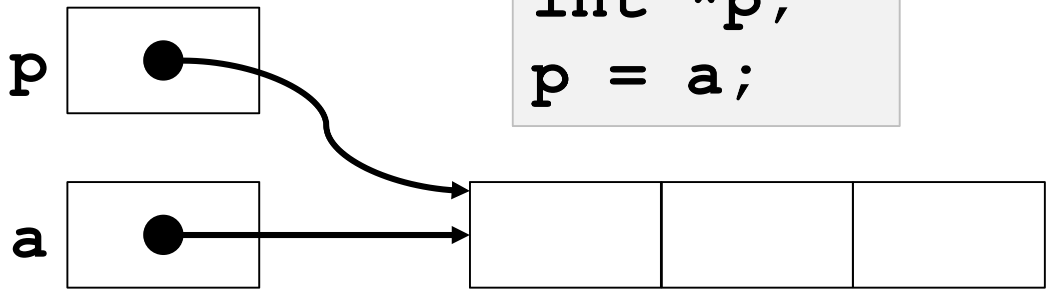
- The array/pointer variable points to the <u>first element</u> of the array.

```
int a[3];
int *p;
p = a;
```



Pointer Arithmetic

```
int a[3];
int *p;
p = a;
```

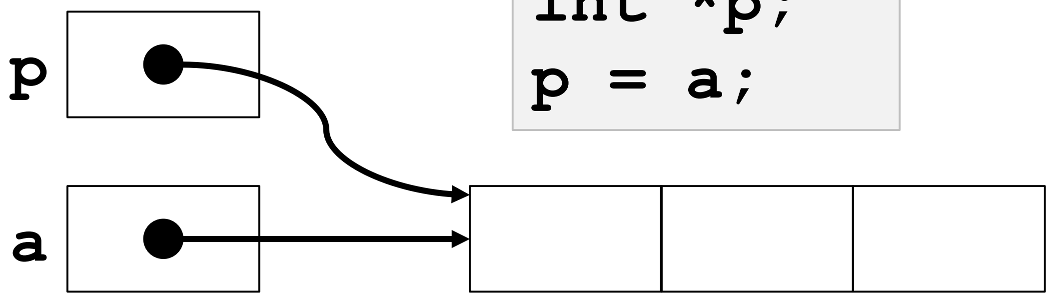


- The following expressions all access the third array element:

```
a[2]
p[2]
*(p+2)
*(a+2)
```

What is `*p+2` ?

Pointer Arithmetic, *cont'd*




```
int a[3];
int *p;
p = a;
```

- Use a pointer to iterate through an array.
  - In the above example, `p` initially points to the first element of the array.
  - Then `p++` points to the second element.
  - And next, `p++` points to the third element.

Dynamic Arrays

- Up until now, whenever we declared an array, we explicitly gave its size.
  - Example: `int a[10];`
- But suppose we don't know until run time how many elements we need.
  - Example: At run time, your program reads in a count of names, and then the names. You want to create an array that can hold exactly that many names.
- You can use a dynamic array (instead of a vector).

Dynamic Arrays, *cont'd*

- If the size of the array you want is in variable `n` whose value you don't know until run time, use the `new` operator to create an array of size `n`.

- Use a pointer variable to point to the first element of the dynamic array.

```
string *names = new string[n];
```

- When you're done with the array, use the special form of the `delete` operator to remove the array from memory: `delete [] names;`

`char*` and `char**`

- Recall that C programs didn't have C++ style strings, but instead had arrays of characters.
- The declaration

```
char *cstr;
```

  is for a dynamic character array, a C-string.

- If you have a dynamic array of C-strings, you need a pointer to a pointer of characters:

```
char **cstr_array;
```

Assignment #4. Big Pi

- You will compute and print the first 1,000 decimal digits of pi.
  - Algorithm: Nonic convergence at https://en.wikipedia.org/wiki/Borwein's_algorithm
- You will use the Multiple-Precision Integers and Rationals (MPIR) library.
  - http://mpir.org/
  - The library is distributed as C source files.
- Use the library to create and work with numbers with arbitrarily long precision.

Assignment #4. Big Pi, *cont'd*

- You will learn how to download the source files, compile them, and configure, build, and install the MPIR library.

- Useful skills to have, because you will most likely need to use other libraries in the future.
  - graphics libraries
  - circuit simulation libraries
  - numerical computing libraries
  - etc.

Assignment #4. Big Pi, *cont'd*

- Building and installing the MPIR library is straightforward on Linux and Mac OS.

- Therefore, if you are on Windows, use VirtualBox to run Linux as a virtual machine.
  - VirtualBox: https://www.virtualbox.org/wiki/VirtualBox
  - Debian Linux: https://www.debian.org/
  - Ubuntu Linux: https://www.ubuntu.com/

- Download and install a Linux disk image (**.iso** file) into VirtualBox.

Assignment #4. Big Pi, *cont'd*

- Please work together to help each other to build and install MPIR.
- Programs must be individual work, as usual.
- Extra credit: Compute one million digits of pi.

CMPE 180-92

Data Structures and Algorithms in C++

September 21 Class Meeting

Department of Computer Engineering
San Jose State University

Fall 2017

Instructor: Ron Mak

www.cs.sjsu.edu/~mak

Assignment #4 Sample Solution

```
#include <iostream>
#include <iomanip>
#include <mpir.h>
#include <stdlib.h>
#include <string.h>

using namespace std;

const int MAX_ITERATIONS = 100;
const int PLACES         = 1000;        // desired decimal places
const int PRECISION      = PLACES + 1;  // +1 for the digit 3 before the decimal

const int BASE       = 10;  // base 10 numbers
const int BIT_COUNT  = 8;   // bits per machine word

const int BLOCK_SIZE = 10;                      // print digits in blocks
const int LINE_SIZE  = 100;                     // digits to print per line
const int LINE_COUNT = PLACES/LINE_SIZE;        // lines to print
const int GROUP_SIZE = 5;                       // line grouping size
```

Assignment #4 Sample Solution

```
void cube_root(mpf_t& x, const mpf_t a)
{
    // Use Halley's method:
    // https://en.wikipedia.org/wiki/Cube_root

    // Multiple-precision variables
    mpf_t x_prev;  mpf_init(x_prev);
    mpf_t temp1;   mpf_init(temp1);
    mpf_t temp2;   mpf_init(temp2);
    mpf_t two_a;   mpf_init(two_a);
    mpf_t x_cubed; mpf_init(x_cubed);

    // Constant 3
    mpf_t three; mpf_init(three); mpf_set_str(three, "3", BASE);

    // Set an initial estimate for x.
    mpf_div(x, a, three);   // x = a/3

    int n = 0;  // iteration counter
```

Assignment #4 Sample Solution, *cont'd*

$$x_{n+1} = x_n \left(\frac{x_n^3 + 2a}{2x_n^3 + a} \right)$$

```
    // Loop until two consecutive values are equal
    // or up to MAX_ITERATIONS times.
    do
    {
        mpf_set(x_prev, x);

        mpf_mul(x_cubed, x, x);
        mpf_mul(x_cubed, x_cubed, x);        // x_cubed = x^3
        mpf_add(two_a, a, a);                // two_a = 2a
        mpf_add(temp1, x_cubed, two_a);      // temp1 = x^3 + 2a
        mpf_add(temp2, x_cubed, x_cubed);    // temp2 = 2x^3
        mpf_add(temp2, temp2, a);            // temp2 = 2x^3 + a
        mpf_div(temp1, temp1, temp2);        // temp1 = (x^3 + 2a)/(2x^3 + a)
        mpf_mul(x, x, temp1);                // x = x((x^3 + 2a)/(2x^3 + a))

        n++;
    } while ((mpf_cmp(x, x_prev) != 0) && (n < MAX_ITERATIONS));
}
```

Assignment #4 Sample Solution, *cont'd*

```
void compute_pi(mpf_t& pi)
{
    // Use a nonic algorithm:
    // https://en.wikipedia.org/wiki/Borwein's_algorithm

    // Multiple-precision constants.
    mpf_t one;          mpf_init_set_str(one,           "1", BASE);
    mpf_t two;          mpf_init_set_str(two,           "2", BASE);
    mpf_t three;        mpf_init_set_str(three,         "3", BASE);
    mpf_t nine;         mpf_init_set_str(nine,          "9", BASE);
    mpf_t twenty_seven; mpf_init_set_str(twenty_seven, "27", BASE);

    mpf_t one_third; mpf_init(one_third);
    mpf_div(one_third, one, three);
```

Assignment #4 Sample Solution, *cont'd*

```
// Multiple-precision variables
mpf_t a;          mpf_init(a);
mpf_t r;          mpf_init(r);
mpf_t s;          mpf_init(s);
mpf_t t;          mpf_init(t);
mpf_t u;          mpf_init(u);
mpf_t v;          mpf_init(v);
mpf_t w;          mpf_init(w);
mpf_t power3;     mpf_init(power3);
mpf_t prev_a;     mpf_init(prev_a);

// Temporaries
mpf_t temp1; mpf_init(temp1);
mpf_t temp2; mpf_init(temp2);
```

Assignment #4 Sample Solution, *cont'd*

Start by setting

$$a_0 = \frac{1}{3}$$
$$r_0 = \frac{\sqrt{3}-1}{2}$$
$$s_0 = (1 - r_0^3)^{1/3}$$

```
// Initialize a
mpf_set(a, one_third);       // a = 1/3

// Initialize r
mpf_sqrt(temp1, three);      // temp1 = sqrt(3)
mpf_sub(temp1, temp1, one); // temp1 = sqrt(3) - 1
mpf_div(r, temp1, two);      // r = (sqrt(3) - 1)/2

// Initialize s
mpf_mul(temp1, r, r);
mpf_mul(temp1, temp1, r);    // temp1 = r^3
mpf_sub(temp1, one, temp1); // temp1 = 1 - r^3
cube_root(s, temp1);         // s = cbrt(1 - r^3)

// Initialize power3
mpf_set(power3, one_third);
```

Assignment #4 Sample Solution, *cont'd*

```
// Loop until two consecutive values are equal
// or up to MAX_ITERATIONS times. Iterate at least twice.
int n = 0;
do
{
    // Save the previous a for later comparison.
    mpf_set(prev_a, a);                      // prev_a = a

    mpf_div(temp1, one, prev_a);

    // Compute t
    mpf_add(temp1, r, r);                    // temp1 = 2r
    mpf_add(t, one, temp1);                  // t = 1 + 2r

    // Compute u
    mpf_add(temp1, one, r);                  // temp1 = 1 + r
    mpf_mul(temp2, r, r);                    // temp2 = r^2
    mpf_add(temp1, temp1, temp2);            // temp1 = 1 + r +r^2
    mpf_mul(temp1, nine, temp1);
    mpf_mul(temp1, r, temp1);                // temp1 =  9r(1 + r + r^2)
    cube_root(u, temp1);                     // u = cbrt(9r(1 + r + r^2))
```

Then iterate

$$
\begin{aligned}
t_{n+1} &= 1 + 2r_n \\
u_{n+1} &= (9r_n(1 + r_n + r_n^2))^{1/3} \\
v_{n+1} &= t_{n+1}^2 + t_{n+1}u_{n+1} + u_{n+1}^2 \\
w_{n+1} &= \frac{27(1 + s_n + s_n^2)}{v_{n+1}} \\
a_{n+1} &= w_{n+1}a_n + 3^{2n-1}(1 - w_{n+1}) \\
s_{n+1} &= \frac{(1 - r_n)^3}{(t_{n+1} + 2u_{n+1})v_{n+1}} \\
r_{n+1} &= (1 - s_{n+1}^3)^{1/3}
\end{aligned}
$$

Assignment #4 Sample Solution, *cont'd*

```
// Compute v
mpf_mul(temp1, t, t);          // temp1 = t^2
mpf_mul(temp2, t, u);          // temp2 = tu
mpf_add(temp1, temp1, temp2);  // temp1 = t^2 + tu
mpf_mul(temp2, u, u);          // temp2 = u^2
mpf_add(v, temp1, temp2);      // v = t^2 + tu + u^2

// Compute w
mpf_add(temp1, one, s);        // temp1 = 1 + s
mpf_mul(temp2, s, s);          // temp2 = s^2
mpf_add(temp1, temp1, temp2);  // temp1 = 1 + s + s^2
mpf_mul(temp1, temp1, twenty_seven);    // temp1 = 27(1 + s + s^2)
mpf_div(w, temp1, v);          // w = (27(1 + s + s^2))/v

// Compute next a
mpf_mul(temp1, w, a);          // temp1 = wa
mpf_sub(temp2, one, w);        // temp2 = 1 - w
mpf_mul(temp2, power3, temp2); // temp2 =  (3^(2n-1))(1 - w)
mpf_add(a, temp1, temp2);      // a = wa + (3^(2n-1))(1 - w)
```

Then iterate

$$t_{n+1} = 1 + 2r_n$$
$$u_{n+1} = (9r_n(1 + r_n + r_n^2))^{1/3}$$
$$v_{n+1} = t_{n+1}^2 + t_{n+1}u_{n+1} + u_{n+1}^2$$
$$w_{n+1} = \frac{27(1 + s_n + s_n^2)}{v_{n+1}}$$
$$a_{n+1} = w_{n+1}a_n + 3^{2n-1}(1 - w_{n+1})$$
$$s_{n+1} = \frac{(1 - r_n)^3}{(t_{n+1} + 2u_{n+1})v_{n+1}}$$
$$r_{n+1} = (1 - s_{n+1}^3)^{1/3}$$

Assignment #4 Sample Solution, *cont'd*

```
// Compute next s
mpf_sub(temp2, one, r);          // temp2 = 1 - r
mpf_mul(temp1, temp2, temp2);
mpf_mul(temp1, temp1, temp2);    // temp1 = (1 - r)^3
mpf_add(temp2, t, u);
mpf_add(temp2, temp2, u);        // temp2 = t + 2u
mpf_mul(temp2, temp2, v);        // temp2 = (t + 2u)v
mpf_div(s, temp1, temp2);        // s = ((1 - r)^3)/((t + 2u)v)

// Compute next r
mpf_mul(temp1, s, s);
mpf_mul(temp1, temp1, s);        // temp1 = s^3
mpf_sub(temp1, one, temp1);      // temp1 = 1 - s^3
cube_root(r, temp1);             // r = (1 - s^3)^(1/3)
```

Then iterate

$$
\begin{aligned}
t_{n+1} &= 1 + 2r_n \\
u_{n+1} &= (9r_n(1 + r_n + r_n^2))^{1/3} \\
v_{n+1} &= t_{n+1}^2 + t_{n+1}u_{n+1} + u_{n+1}^2 \\
w_{n+1} &= \frac{27(1 + s_n + s_n^2)}{v_{n+1}} \\
a_{n+1} &= w_{n+1}a_n + 3^{2n-1}(1 - w_{n+1}) \\
s_{n+1} &= \frac{(1 - r_n)^3}{(t_{n+1} + 2u_{n+1})v_{n+1}} \\
r_{n+1} &= (1 - s_{n+1}^3)^{1/3}
\end{aligned}
$$

Assignment #4 Sample Solution, *cont'd*

```
        // Compute next power of 3
        mpf_mul(power3, power3, nine);  // power3 = 3^(2n-1)

        n++;
    } while (   ((n < 2) || (mpf_eq(a, prev_a, PRECISION) == 0))
             && (n < MAX_ITERATIONS));

    // Compute pi = 1/a
    mpf_div(pi, one, a);
}
```

Then iterate

$$
\begin{aligned}
t_{n+1} &= 1 + 2r_n \\
u_{n+1} &= (9r_n(1 + r_n + r_n^2))^{1/3} \\
v_{n+1} &= t_{n+1}^2 + t_{n+1}u_{n+1} + u_{n+1}^2 \\
w_{n+1} &= \frac{27(1 + s_n + s_n^2)}{v_{n+1}} \\
a_{n+1} &= w_{n+1}a_n + 3^{2n-1}(1 - w_{n+1}) \\
s_{n+1} &= \frac{(1 - r_n)^3}{(t_{n+1} + 2u_{n+1})v_{n+1}} \\
r_{n+1} &= (1 - s_{n+1}^3)^{1/3}
\end{aligned}
$$

Assignment #4 Sample Solution, *cont'd*

```
/**
 * Print the decimal places of a multiple-precision number x.
 * @param pi the multiple-precision number to print.
 */
void print(const mpf_t& pi)
{
    mp_exp_t exp;  // exponent (not used)

    // Convert the multiple-precision number x to a C string.
    char *str = NULL;
    char *s = mpf_get_str(str, &exp, BASE, PRECISION, pi);
    char *p = s+1;  // skip the 3 before the decimal point

    cout << endl;
    cout << "3.";

    char block[BLOCK_SIZE + 1];  // 1 extra for the ending \0
```

Assignment #4 Sample Solution, *cont'd*

```
    // Loop for each line.
    for (int i = 1; i <= LINE_COUNT; i++)
    {
        // Loop to print blocks of digits in each line.
        for (int j = 0; j < LINE_SIZE; j += BLOCK_SIZE)
        {
            strncpy(block, p+j, BLOCK_SIZE);
            block[BLOCK_SIZE] = '\0';
            cout << block << " ";
        }

        cout << endl << "   ";

        // Print a blank line for grouping.
        if (i%GROUP_SIZE == 0) cout << endl << "   ";

        p += LINE_SIZE;
    }

    free(s);
}
```

Structures

- A structure represents a collection of values that can be of different data types.

- We want to treat the collection as a single item.
  - Example:

structure tag

```
struct Employee
{
    int id;
    string first_name;
    string last_name;
    double salary;
};
```

members

Structures are Types

```
struct Employee
{
    int id;
    string first_name;
    string last_name;
    double salary;
};
```

john

id	98765
first_name	“John”
last_name	“Johnson”
salary	75000.00

- A structure is a type:

```
Employee mary, john;

mary.id = 12345;
mary.first_name = "Mary";
mary.last_name  = "Poppins";
mary.salary = 150000.25;

mary.salary = 1.10*mary.salary;
```

mary

id	12345
first_name	“Mary”
last_name	“Poppins”
salary	150000.25

Scope of Structure Member Names

- Two different structure types can contain members with the same name:

```
struct Employee
{
    int id;
    ...
};
```

```
struct Student
{
    int id;
    ...
};
```

- To access the value of one of the structure's members, use a member variable such as **`mary.salary`**

Structure Variables

- If you have two variables of the same structure type, you can assign one to the other:

```
john = mary;
```

- This is equivalent to:

```
john.id         = mary.id;
john.first_name = mary.first_name;
john.last_name  = mary.last_name;
john.salary     = mary.salary;
```

Structure Variables, *cont'd*

- An array of employees:

```
Employee team[10];

team[4].id = 39710;
team[4].first_name = "Sally";
```

- Pass a structure variable to a function:

```
void foo(Employee emp1, Employee& emp2);
```

- Return a structure value:

```
Employee find_employee(int id);
```

Structure Variables, *cont'd*

- Pointer to a structure:

```
Employee *emp_ptr;

emp_ptr = new Employee;
(*emp_ptr).id = 192837;
emp_ptr->salary = 95000.00;
```

- Nested structures:

```
struct Employee
{
    int id;
    string first_name;
    string last_name;
    double salary;
    Birthday bday;
};
```

```
struct Birthday
{
    int month, day, year;
};
```

```
Employee tom;
tom.bday.year = 1992;
```

Break

Object-Oriented Programming

- Object-oriented programming (OOP) is about
  - encapsulation
  - inheritance
  - polymorphism

Combine variables and functions into a single class.

- Work with values called objects.
  - Objects have member functions that operate on the objects.
  - Example: A string is an object. Strings have a length method, so that if `str` is a string variable, then `str.length()` is the length of its string value.

Classes

- A class is a data type whose values are objects.
  - Like structure types, you can define your own class types.
- A class type definition includes both member variables and declarations of member functions.
  - Example:

```
class Birthday
{
public:
    int month, day, year;
    void print();
};
```

Defining Member Functions

- Define member functions <u>outside</u> of the class definition:

```
class Birthday
{
public:
    int month, day, year;
    void print();
};

void Birthday::print()
{
    cout << month << "/" << day << "/" << year << endl;
}
```

- Scope resolution operator `::`

Public and Private Members

- Members of a class are either public or private.
- Private members of a class can be accessed only by member functions of the same class.
- You can provide public getters and setters for any private member variables.
  - AKA accessors and mutators
- A member function (public or private) can be labelled `const`.
  - It will not modify the value of any member variable.

Public and Private Members, *cont'd*

Birthday1.cpp

```
class Birthday
{
public:
    void set_year(int y);
    void set_month(int m);
    void set_day(int d);

    int get_year()  const;
    int get_month() const;
    int get_day()   const;
    void print()    const;

private:
    int year, month, day;
};
```

Public and Private Members, *cont'd*

Birthday1.cpp

```
int Birthday::get_year()  const { return year; }
int Birthday::get_month() const { return month; }
int Birthday::get_day()   const { return day; }

void Birthday::set_year(int y)  { year = y; }
void Birthday::set_month(int m) { month = m; }
void Birthday::set_day(int d)   { day = d; }

void Birthday::print() const
{
    cout << month << "/" << day << "/" << year << endl;
}
```

Public and Private Members, *cont'd*

Birthday1.cpp

```
int main()
{
    Birthday bd;
    bd.set_year(1990);
    bd.set_month(9);
    bd.set_day(2);
    bd.print();
}
```

```
9/2/1990
```

Constructors

- A class can define special member functions called constructors that initialize the values of member variables.

- A constructor has the same name as the class itself.
  - It has no return type, not even void.
  - The default constructor has no parameters.

- A constructor is called automatically whenever an object of the class is declared.

Constructors, *cont'd*

Birthday1.cpp

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

    ...
}

Birthday::Birthday() : year(0), month(0), day(0)
{
    // Default constructor with an empty body
}

Birthday::Birthday(int y, int m, int d) : year(y), month(m), day(d)
{
    // Empty body
}
```

Constructors, *cont'd*

Birthday1.cpp

```
int main()
{
    Birthday bd1;                  // call default constructor
    Birthday bd2(2000, 9, 2);   // call constructor

    bd1.print();
    bd2.print();
}
```

```
0/0/0
9/2/2000
```

- Do not write: `Birthday bd1();`
  - That is a declaration of a function named **`bd1`** that returns a value of type **`Birthday`**.

Constructors, *cont'd*

- If you provided no constructors for a class, the C++ compiler will generate a default constructor that does nothing.
- However, if you provided at least one constructor for the class, the compiler will not generate a default constructor.

Constructors, *cont'd*

- Suppose you are provided this constructor only:

```
Birthday(int y, int m, int d);
```

- Then the following object declaration is illegal:

```
Birthday bd1;
```

Friend Functions

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

    int get_year()  const;
    int get_month() const;
    int get_day()   const;

    void set_year(int y);
    void set_month(int m);
    void set_day(int d);

    void print() const;

private:
    int year, month, day;
};
```

- Write a function that is external to the class (i.e., not a member function) that compares two birthdays for equality.

Friend Functions, *cont'd*

Birthday1.cpp

```
bool equal(const Birthday& bd1, const Birthday& bd2)
{
    return     (bd1.get_year()  == bd2.get_year())
            && (bd1.get_month() == bd2.get_month())
            && (bd1.get_day()   == bd2.get_day());
}
```

- Function **`equal`** must call the accessor (getter) methods because **`year`**, **`month`**, and **`day`** are private member variables.
- Make function **`equal`** a friend of class **`Birthday`** to allow the function to access the <u>private</u> member variables directly.

Friend Functions, *cont'd*

Birthday2.cpp

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

    int get_year()  const;
    int get_month() const;
    int get_day()   const;

    void set_year(int y);
    void set_month(int m);
    void set_day(int d);

    void print() const;

    friend bool equal(const Birthday& bd1, const Birthday& bd2);

private:
    int year, month, day;
};
```

Because it is a friend of the class, function **equal** can now access private members.

```
bool equal(const Birthday& bd1,
           const Birthday& bd2)
{
    return     (bd1.year  == bd2.year)
            && (bd1.month == bd2.month)
            && (bd1.day   == bd2.day);
}
```

Have both friend functions and accessor functions.

Operator Overloading

- How many years apart are two birthdays?
- We can write a function **`years_apart`** that takes two birthdays and subtracts their years:

Birthday2.cpp

```
class Birthday
{
public:
    ...
    friend bool equal(const Birthday& bd1, const Birthday& bd2);
    friend int years_apart(const Birthday& bd1, const Birthday& bd2);
    ...
};
```

```
int years_apart(const Birthday& bd1, const Birthday& bd2)
{
    return abs(bd1.year - bd2.year);
}
```

Operator Overloading, *cont'd*

- Overload the subtraction operator
  and make `operator -` a friend function.

Birthday2.cpp

```
class Birthday
{
public:
    ...
    friend bool equal(const Birthday& bd1, const Birthday& bd2);
    friend int years_apart(const Birthday& bd1, const Birthday& bd2);
    friend int operator -(const Birthday& bd1, const Birthday& bd2);
    ...
};
```

```
int operator -(const Birthday& bd1, const Birthday& bd2)
{
    return abs(bd1.year - bd2.year);
}
```

Operator Overloading, *cont'd*

Birthday2.cpp

```
int main()
{
    Birthday bd1;                // call default constructor
    Birthday bd2(1990, 9, 2);    // call constructor
    Birthday bd3(2001, 5, 8);    // call constructor

    bd1.print();
    bd2.print();

    cout << years_apart(bd2, bd3) << endl;
    cout << bd2 - bd3 << endl;
}
```

```
0/0/0
9/2/1990
11
11
```

Overload <<

- You can overload the stream insertion operator.
- Suppose you want a **`Birthday`** object to be output in the form month/day/year.

Birthday2.cpp

```
class Birthday
{
public:
    ...
    friend ostream& operator <<(ostream& outs, const Birthday& bd);
    ...
};
```

```
friend ostream& operator <<(ostream& outs, const Birthday& bd)
{
    outs << bd.month << "/" << bd.day << "/" << bd.year << endl;
    return outs;
}
```

Overload <<, *cont'd*

Birthday2.cpp

```
int main()
{
    Birthday bd1;                  // call default constructor
    Birthday bd2(1990, 9, 2);   // call constructor
    Birthday bd3(2001, 5, 8);   // call constructor

    cout << bd1 << ", " << bd2 << ", " << bd3 << endl;
}
```

```
0/0/0, 9/2/1990, 5/8/2001
```

Overload >>

- You want to input birthdays in the format

  {*year***,** *month***,** *day***}**

  - Example: `{1993, 9, 2}`

- Overload the stream extraction operator.

Birthday2.cpp

```
class Birthday
{
public:
    ...
    friend istream& operator >>(istream& ins, Birthday& bd);
    ...
};
```

Overload >>, *cont'd*

```
istream& operator >>(istream& ins, Birthday& bd)
{
    int y, m, d;
    char ch;

    ins >> ch;
    if (ch == '{')
    {
        ins >> y;

        ins >> ch;
        if (ch == ',')
        {
            ins >> m;

            ins >> ch;
            if (ch == ',')
            {
                ins >> d;

                ins >> ch;
                if (ch == '}')
                {
                    bd.year  = y;
                    bd.month = m;
                    bd.day   = d;
                }
            }
        }
    }

    return ins;
}
```

Error checking needed!

Birthday2.cpp

```
int main()
{
    Birthday bd1;
    Birthday bd2;

    cout << "Enter two birthdays: ";
    cin >> bd1 >> bd2;
    cout << bd1 << ", " << bd2 << endl;
}
```

```
Enter two birthdays: {1953, 9, 2}   {1957, 4, 3}
9/2/1953, 4/3/1957
```

Abstract Data Types

- A data type specifies:
  - what values are allowed
  - what operations are allowed
- An abstract data type (ADT):
  - allows its values and operations to be used
  - hides the implementation of values and operations
- Example: The predefined type `int` is an ADT.
  - You can use integers and the operators `+ - * / %`
  - But you don't know how they're implemented.

Abstract Data Types, *cont'd*

- To make your class an ADT, you must separate:
  - The specification of how a type is <u>used</u>.
  - The details of how the type is <u>implemented</u>.
- To ensure this separation:
  - Make all member variables private.
  - Make public all the member functions that a programmer needs to use, and fully specify how to use each one.
  - Make private all helper member functions.

Is the **`Birthday`** class an ADT?

Separate Compilation

- Put each class declaration in a separate .h header file.
  - By convention, name the file after the class name.
  - Any other source file that uses the class would **`#include`** the class header file.
- Put the <u>implementations</u> of the member functions into a .cpp file.
  - By convention, name the file after the class name.
- A class header file is the interface that the class presents to users of the class.

Separate Compilation, *cont'd*

Birthday.h

```
#ifndef BIRTHDAY_H_
#define BIRTHDAY_H_

using namespace std;

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

    // Destructor
    ~Birthday();

    int get_year() const;
    int get_month() const;
    int get_day() const;

    void set_year(int y);
    void set_month(int m);
    void set_day(int d);

    void print();

    friend bool equal(const Birthday& bd1, const Birthday& bd2);
    friend int years_apart(const Birthday& bd1, const Birthday& bd2);
    friend int operator -(const Birthday& bd1, const Birthday& bd2);
    friend ostream& operator <<(ostream& outs, const Birthday& bd);
    friend istream& operator >>(istream& ins, Birthday& bd);

private:
    int year, month, day;
};

#endif
```

Separate Compilation, *cont'd*

Birthday3.cpp

```
#include <iostream>
#include <cstdlib>
#include "Birthday.h"

using namespace std;

Birthday::Birthday() : year(0), month(0), day(0)
{
    // Default constructor with an empty body
}

Birthday::Birthday(int y, int m, int d) : year(y), month(m), day(d)
{
    // Empty body
}

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

int Birthday::get_year()  const { return year; }
int Birthday::get_month() const { return month; }
int Birthday::get_day()   const { return day; }

void Birthday::set_year(int y)  { year = y; }
void Birthday::set_month(int m) { month = m; }
void Birthday::set_day(int d)   { day = d; }
```

Separate Compilation, *cont'd*

Birthday.cpp

```
void Birthday::print()
{
    cout << month << "/" << day << "/" << year << endl;
}

int operator -(const Birthday& bd1, const Birthday& bd2)
{
    return abs(bd1.year - bd2.year);
}

ostream& operator <<(ostream& outs, const Birthday& bd)
{
    outs << bd.month << "/" << bd.day << "/" << bd.year;
    return outs;
}

istream& operator >>(istream& ins, Birthday& bd)
{
    ...
}
```

Separate Compilation, *cont'd*

BirthdayTester.cpp

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

int main()
{
    Birthday bd1;                   // call default constructor
    Birthday bd2(1990, 9, 2);   // call constructor
    Birthday bd3(2001, 5, 8);   // call constructor

    bd1.print();
    bd2.print();

    cout << bd2 - bd3 << endl;
    cout << bd1 << ", " << bd2 << ", " << bd3 << endl;

    cout << endl;
    cout << "Enter two birthdays: ";
    cin >> bd1 >> bd2;
    cout << bd1 << ", " << bd2 << endl;
}
```

Assignment #5. Roman Numerals

- Define a C++ class **`RomanNumeral`** that implements arithmetic operations with Roman numerals, and reading and writing Roman numerals.
  - See https://en.wikipedia.org/wiki/Roman_numerals

- Private member variables **`string roman`** and **`int decimal`** store the Roman numeral string (such as **`"MCMLXVIII"`**) and its integer value (1968).

Assignment #5. Roman Numerals, *cont'd*

- Private member functions **`to_roman`** and **`to_decimal`** convert between the string and integer values of a **`RomanNumeral`** object.

- One constructor has an integer parameter, and another constructor has a string parameter.
  - Construct a Roman numeral object by giving either its integer or string value.

- Public getter functions return the object's string and integer values.

Assignment #5. Roman Numerals, *cont'd*

- Override the arithmetic operators `+ - * /`
  - Roman numerals perform integer division.

- Override the equality operators `== !=`

- Override the stream operators `>>` and `<<`
  - Input a Roman numeral value as a string, such as **`MCMLXVIII`**
  - Output a Roman numeral value in the form

    `[`*integer value*`:`*roman string*`]`

    such as **`[1968:MCMLXVIII]`**

Assignment #5. Roman Numerals, *cont'd*

- A test program inputs and parses a text file containing simple two-operand arithmetic expressions with Roman numerals:

```
MCMLXIII + LIII
MMI - XXXIII
LIII * XXXIII
MMI / XXXIII
```

- It performs the arithmetic and output the results:

```
[1963:MCMLXIII] + [53:LIII] = [2016:MMXVI]
[2001:MMI] - [33:XXXIII] = [1968:MCMLXVIII]
[53:LIII] * [33:XXXIII] = [1749:MDCCXLIX]
[2001:MMI] / [33:XXXIII] = [60:LX]
```

Assignment #5. Roman Numerals, *cont'd*

- File RomanNumeral.h contains the class declaration.
- File RomanNumeral.cpp contains the class implementation.
- File RomanNumeralTester.cpp contains two functions to test the class.

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 <u>default constructor</u> is called for each element.

- Therefore, a class that can be the base type of an array <u>must</u> 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;
}
```

Demo

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?

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?

“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 <u>new, longer vector</u> and <u>copies the elements</u> 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 <u>all</u> the book records in sorted order that <u>match</u> the search criteria:

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

Prompt: **`Command:`**

<u>Binary searches</u> by ISBN.
<u>Linear searches</u> 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.

CMPE 180-92

Data Structures and Algorithms in C++

October 5 Class Meeting

Department of Computer Engineering
San Jose State University

Fall 2017

Instructor: Ron Mak

www.cs.sjsu.edu/~mak

Assignment #6 Sample Solution

Book.h

```
class Book
{
public:
    /**
     * Book categories.
     */
    enum class Category { FICTION, HISTORY, TECHNICAL, NONE };

    /**
     * Default constructor.
     */
    Book();

    /**
     * Constructor.
     */
    Book(string isbn, string last, string first, string title, Category category);

    /**
     * Destructor.
     */
    ~Book();
```

Assignment #6 Sample Solution, *cont'd*

Book.h

```
/**
 * Getter.
 * @return the book's ISBN.
 */
string get_isbn() const;

/**
 * Getter.
 * @return the author's last name.
 */
string get_last() const;

/**
 * Getter.
 * @return the author's first name.
 */
string get_first() const;
```

Assignment #6 Sample Solution, *cont'd*

Book.h

```
/**
 * Getter.
 * @return the book's title.
 */
string get_title() const;

/**
 * Getter.
 * @return the book's category.
 */
Category get_category() const;
```

Assignment #6 Sample Solution, *cont'd*

Book.h

```
/**
 * Overloaded input stream extraction operator for a book.
 * Reads from a CSV file.
 * @param istream the input stream.
 * @param book the book to input.
 * @return the input stream.
 */
friend istream& operator >>(istream& ins, Book& emp);

/**
 * Overloaded output stream insertion operator for a book.
 * @param ostream the output stream.
 * @param book the book to output.
 * @return the output stream.
 */
friend ostream& operator <<(ostream& outs, const Book& emp);
```

Assignment #6 Sample Solution, *cont'd*

Book.h

```
private:
    string isbn;          // ISBN
    string last;          // author's last name
    string first;         // author's first name
    string title;         // book title
    Category category;    // book category
};


/**
 * Overloaded output stream insertion operator for a book category.
 * Doesn't need to be a friend since it doesn't access any
 * private members.
 * @param ostream the output stream.
 * @param book the category to output.
 * @return the output stream.
 */
ostream& operator <<(ostream& outs, const Book::Category& category);
```

Assignment #6 Sample Solution, *cont'd*

Book.cpp

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <stdio.h>
#include "Book.h"

using namespace std;

Book::Book()
    : isbn(""), last(""), first(""), title(""),
      category(Category::NONE)
{}

Book::Book(string isbn, string last, string first, string title,
           Category category)
    : isbn(isbn), last(last), first(first), title(title),
      category(category)
{}

Book::~Book()
{}
```

Assignment #6 Sample Solution, *cont'd*

Book.cpp

```
string Book::get_isbn()  const { return isbn; }
string Book::get_last()  const { return last; }
string Book::get_first() const { return first; }
string Book::get_title() const { return title; }

Book::Category Book::get_category() const { return category; }
```

Assignment #6 Sample Solution, *cont'd*

Book.cpp

```
istream& operator >>(istream& ins, Book& book)
{
    ins.get();  // skip the blank after the command

    getline(ins, book.isbn,  ',');
    getline(ins, book.last,  ',');
    getline(ins, book.first, ',');
    getline(ins, book.title, ',');

    string catstr;
    getline(ins, catstr);

    book.category = Book::Category::NONE;

    if      (catstr == "fiction")   book.category = Book::Category::FICTION;
    else if (catstr == "history")   book.category = Book::Category::HISTORY;
    else if (catstr == "technical") book.category = Book::Category::TECHNICAL;

    return ins;
}
```

Assignment #6 Sample Solution, *cont'd*

```
ostream& operator <<(ostream& outs, const Book::Category& category)
{
    switch (category)
    {
        case Book::Category::FICTION:   outs << "fiction";      break;
        case Book::Category::HISTORY:   outs << "history";      break;
        case Book::Category::TECHNICAL: outs << "technical";    break;
        case Book::Category::NONE:      outs << "none";         break;
    }

    return outs;
}
```

Book.cpp

Assignment #6 Sample Solution, *cont'd*

Book.cpp

```
ostream& operator <<(ostream& outs, const Book& book)
{
    outs << "Book{ISBN=" << book.isbn << ", last=" << book.last
         << ", first=" << book.first << ", title=" << book.title
         << ", category=" << book.category << "}";
    return outs;
}
```

BookApp.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <iomanip>
#include "Book.h"

using namespace std;

// Status codes.
enum class StatusCode {OK, DUPLICATE, NOT_FOUND, INVALID_COMMAND};

/**
 * Execute a command.
 * @param command the command.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 */
StatusCode execute(const char command, istream &input,
                   vector<Book>& catalog);
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
/**
 * Insert a new book into the catalog at the appropriate position
 * to maintain sort order by ISBN.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 * @param index set to the catalog index of the new record.
 * @return the status code of this operation.
 */
StatusCode insert(istream &input, vector<Book>& catalog, int &index);

/**
 * Remove a book from the catalog.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 * @param book set to the removed book.
 * @return the status code of this operation.
 */
StatusCode remove(istream &input, vector<Book>& catalog, Book& book);
```

BookApp.cpp

```
/**
 * Match books.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 * @return a vector of the indices of the matching books.
 */
vector<int> match(istream &input, vector<Book>& catalog);

/**
 * Match the book in the catalog with the given ISBN.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 * @return a vector of the index of the matching book.
 */
vector<int> match_by_isbn(const string last,
                          const vector<Book>& catalog);
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
/**
 * Match the books in the catalog with the given author's last name.
 * Use a linear search.
 * @param last the author's last name.
 * @param catalog the book vector.
 * @return a vector of the indices of the matching books.
 */
vector<int> match_by_author(const string last,
                            const vector<Book>& catalog);

/**
 * Match the books in the catalog in the given category.
 * Use a linear search.
 * @param catstr the category.
 * @param catalog the book vector.
 * @return a vector of the indices of the matching books.
 */
vector<int> match_by_category(string catstr,
                              const vector<Book>& catalog);
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
/**
 * Match all the books in the catalog.
 * Use a linear search.
 * @param last the author's last name.
 * @param catalog the book vector.
 * @return a vector of the indices of the matching books.
 */
vector<int> match_all(const vector<Book>& catalog);

/**
 * Process an invalid command.
 * @param istream the input data stream.
 * @return the status code.
 */
StatusCode invalid_command(istream &input);
```

BookApp.cpp

```
/**
 * Find the book in the catalog with the given ISBN.
 * Use a binary search.
 * @param isbn the ISBN.
 * @param catalog the vector of book records.
 * @return the vector index of the book if found, else return -1.
 */
int find(const string isbn, const vector<Book>& catalog);

/**
 * Print an error message.
 * @param status the status code.
 */
void print_error_message(StatusCode status);

const string INPUT_FILE_NAME = "commands.in";
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
/**
 * The main. Open the command input file and loop to process commands.
 */
int main()
{
    // Open the input file.
    ifstream input;
    input.open(INPUT_FILE_NAME);
    if (input.fail())
    {
        cout << "Failed to open " << INPUT_FILE_NAME << endl;
        return -1;
    }

    vector<Book> catalog;  // book catalog

    char command;
    input >> command;  // read the first command
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
    /**
     * Loop to read commands until the end of file.
     */
    while (!input.fail())
    {
        cout << endl << command << " ";

        StatusCode status = execute(command, input, catalog);
        if (status != StatusCode::OK) print_error_message(status);

        input >> command;
    }

    return 0;
}
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
StatusCode execute(const char command, istream &input,
                   vector<Book>& catalog)
{
    int index;
    StatusCode status;
    Book book;

    // Execute the command.
    switch (command)
    {
        case '+':
            status = insert(input, catalog, index);
            book = catalog[index];
            cout << "Inserted at index " << index << ": "
                 << book << endl;
            break;

        case '-':
            status = remove(input, catalog, book);
            cout << "Removed " << book << endl;
            break;
```

San José State University

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
        case '?':
        {
            vector<int> matches = match(input, catalog);
            for (int i : matches) cout << catalog[i] << endl;
            status = StatusCode::OK;
            break;
        }

        default:
            status = invalid_command(input);
            break;
    }

    return status;
}
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
StatusCode insert(istream &input, vector<Book>& catalog, int& index)
{
    // Read the book information.
    Book book;
    input >> book;

    string isbn = book.get_isbn();

    // Loop to find the proper insertion point.
    index = 0;
    while (    (index < catalog.size())
           && (isbn > catalog[index].get_isbn())) index++;
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
    // Check the insertion point.
    if (index >= catalog.size())
    {
        catalog.push_back(book);         // append at the end
        return StatusCode::OK;
    }
    else if (isbn == catalog[index].get_isbn())
    {
        return StatusCode::DUPLICATE;    // duplicate
    }
    else
    {
        catalog.insert(catalog.begin() + index, book);  // insert
        return StatusCode::OK;
    }
}
```

Assignment #6 Sample Solution, *cont'd*

```
StatusCode remove(istream &input, vector<Book>& catalog, Book& book)
{
    string isbn;
    input >> isbn;

    // Look for the book record with a matching ISBN.
    int index = find(isbn, catalog);
    if (index == -1)
    {
        book = Book(isbn, "", "", "", Book::Category::NONE);
        return StatusCode::NOT_FOUND;
    }

    // Remove the matching book from the catalog.
    book = catalog[index];
    catalog.erase(catalog.begin() + index);
    return StatusCode::OK;
}
```

BookApp.cpp

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
vector<int> match(istream &input, vector<Book>& catalog)
{
    vector<int> matches;

    string str;
    getline(input, str);

    if (str == "")
    {
        matches = match_all(catalog);
    }

    else if (str.find("isbn=") != str.npos)
    {
        string isbn = str.substr(str.find("=") + 1);
        matches = match_by_isbn(isbn, catalog);
    }
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
    else if (str.find("author=") != str.npos)
    {
        string last = str.substr(str.find("=") + 1);
        matches = match_by_author(last, catalog);
    }

    else if (str.find("category=") != str.npos)
    {
        string category = str.substr(str.find("=") + 1);
        matches = match_by_category(category, catalog);
    }

    return matches;
}
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
vector<int> match_by_isbn(const string isbn,
                          const vector<Book>& catalog)
{
    vector<int> matches;

    cout << "Book with ISBN " << isbn << ":" << endl;

    int index = find(isbn, catalog);
    if (index != -1) matches.push_back(index);

    return matches;
}
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
vector<int> match_by_author(const string last,
                            const vector<Book>& catalog)
{
    vector<int> matches;

    cout << "Books by author " << last << ":" << endl;

    // Do a linear search.
    for (int i = 0; i < catalog.size(); i++)
    {
        Book book = catalog[i];
        if (last == book.get_last()) matches.push_back(i);
    }

    return matches;
}
```

Assignment #6 Sample Solution, *cont'd*

```
vector<int> match_by_category(string catstr, const vector<Book>& catalog)
{
    vector<int> matches;

    Book::Category category = catstr == "fiction"   ? Book::Category::FICTION
                            : catstr == "history"   ? Book::Category::HISTORY
                            : catstr == "technical" ? Book::Category::TECHNICAL
                            :                         Book::Category::NONE;


    cout << "Books in category " << category << ":" << endl;

    // Do a linear search.
    for (int i = 0; i < catalog.size(); i++)
    {
        Book book = catalog[i];
        if (category == book.get_category()) matches.push_back(i);
    }


    return matches;
}
```

BookApp.cpp

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
vector<int> match_all(const vector<Book>& catalog)
{
    vector<int> matches;

    cout << "All books in the catalog:" << endl;

    for (int i = 0; i < catalog.size(); i++) matches.push_back(i);
    return matches;
}

StatusCode invalid_command(istream &input)
{
    // Read and ignore the rest of the input line.
    string ignore;
    getline(input, ignore);

    return StatusCode::INVALID_COMMAND;
}
```

Assignment #6 Sample Solution, *cont'd*

```
int find(const string isbn, const vector<Book>& catalog)
{
    // Do a binary search.
    int low = 0;
    int high = catalog.size();

    while (low <= high)
    {
        int mid = (low + high)/2;
        Book book = catalog[mid];

        if (isbn == book.get_isbn())
        {
            return mid;     // found
        }
        else if (isbn < book.get_isbn())
        {
            high = mid - 1; // search lower half
        }
        else
        {
            low  = mid + 1; // search upper half
        }
    }

    return -1;  // not found
}
```

BookApp.cpp

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
void print_error_message(StatusCode status)
{
    switch (status)
    {
        case StatusCode::DUPLICATE:
            cout << "*** Duplicate ISDN ***" << endl;
            break;

        case StatusCode::NOT_FOUND:
            cout << "*** Book not found ***" << endl;
            break;

        case StatusCode::INVALID_COMMAND:
            cout << "*** Invalid command ***" << endl;
            break;

        default:  break;
    }
}
```

A “Safe” Array Type: Version 1

- We will develop a new array type that is “safe”.
  - It will allocate the array dynamically.
  - It will check all subscript values to ensure that they are in the legal range (0 ≤ index < array length).
- We’ll start with an integer array.

A “Safe” Array Type: Version 1, *cont’d*

SafeArray1.h

```
class SafeArray
{
public:
    SafeArray();
    SafeArray(int len);
    ~SafeArray();

    int get_length() const;

    int at(int i) const;
    void set(int i, int value);

    void operator =(const SafeArray& rhs);

private:
    int *elements;
    int length;
};
```

A “Safe” Array Type: Version 1, *cont’d*

SafeArray1.cpp

```
SafeArray::SafeArray() : elements(nullptr), length(0)
{
}

SafeArray::SafeArray(int len) : elements(nullptr), length(len)
{
    elements = new int[length];
}

SafeArray::~SafeArray()
{
    if (elements != nullptr) delete[] elements;
}

int SafeArray::get_length() const { return length; }

int SafeArray::at(int i) const
{
    assert((i >= 0) && (i < length));
    return elements[i];
}
```

A “Safe” Array Type: Version 1, *cont’d*

SafeArray.cpp

```
void SafeArray::set(int i, int value)
{
    assert((i >= 0) && (i < length));
    elements[i] = value;
}

void SafeArray::operator =(const SafeArray& rhs)
{
    if (elements != nullptr) delete[] elements;

    length = rhs.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = rhs.elements[i];
    }
}
```

A “Safe” Array Type: Version 1, *cont’d*

SafeArrayTests1.cpp

```
int main()
{
    SafeArray a1(10), a2;
    //SafeArray a3;

    for (int i = 0; i < 10; i++) a1.set(i, 10*i);

    a2 = a1;
    a1.set(4, -a1.at(4));

    cout << "a1 ="; print(a1);
    cout << "a2 ="; print(a2);

    //a3 = a2 = a1;
    return 0;
}

void print(SafeArray& a)
{
    for (int i = 0; i < a.get_length(); i++) cout << " " << a.at(i);
    cout << endl;
}
```

```
a1 = 0 10 20 30 -40 50 60 70 80 90
a2 = 0 10 20 30 40 50 60 70 80 90
```

A “Safe” Array Type: Version 1, *cont’d*

- What happens if you try to chain assignments?

```
SafeArray a1(10), a2;
SafeArray a3;

...

a3 = a2 = a1;
```

```
../SafeArrayTests.cpp:20:8: error: no viable overloaded '='
    a3 = a2 = a1;
    ~~ ^ ~~~~~~~~
../SafeArray.h:16:10: note: candidate function not viable:
  cannot convert argument of incomplete type 'void' to 'const SafeArray'
    void operator =(const SafeArray& rhs);
         ^
1 error generated.
```

A “Safe” Array Type: Version 2

SafeArray2.h

```
class SafeArray
{
public:
    SafeArray();
    SafeArray(int len);
    ~SafeArray();

    int get_length() const;

    int at(int i) const;
    void set(int i, int value);

    SafeArray& operator =(const SafeArray& rhs);

private:
    int *elements;
    int length;
};
```

A "Safe" Array Type: Version 2, *cont'd*

```
SafeArray& SafeArray::operator =(const SafeArray& rhs)
{
    if (elements != nullptr) delete[] elements;

    length = rhs.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = rhs.elements[i];
    }

    return *this;
}
```

SafeArray2.cpp

A "Safe" Array Type: Version 2, *cont'd*

SafeArrayTests2.cpp

```
int main()
{
    SafeArray a1(10), a2, a3;

    for (int i = 0; i < 10; i++) a1.set(i, 10*i);

    a3 = a2 = a1;
    a1.set(4, -a1.at(4));

    cout << "a1 ="; print(a1);
    cout << "a2 ="; print(a2);
    cout << "a3 ="; print(a3);

    return 0;
}
```

```
a1 = 0 10 20 30 -40 50 60 70 80 90
a2 = 0 10 20 30 40 50 60 70 80 90
a3 = 0 10 20 30 40 50 60 70 80 90
```

A "Safe" Array Type: Version 2, *cont'd*

- What happens the program executes

```
a1 = a1;
```

```
SafeArray& SafeArray::operator =(const SafeArray& rhs)
{
    if (elements != nullptr) delete[] elements;

    length = rhs.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = rhs.elements[i];
    }

    return *this;
}
```

SafeArray2.cpp

A “Safe” Array Type: Version 3

- The solution:

```
SafeArray& SafeArray::operator =(const SafeArray& rhs)
{
    if (this == &rhs) return *this;
    if (elements != nullptr) delete[] elements;

    length = rhs.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = rhs.elements[i];
    }

    return *this;
}
```

SafeArray3.cpp

Break

A “Safe” Array Type: Version 4

- The `at` and `set` member functions are awkward to use.
- Why can’t we use subscripts on a smart array as if it were a regular array?
- We can overload the subscript operator `[]`
  - We want the subscripts to be usable on <u>either side</u> of an assignment.
  - Example:

```
a1[4] = -a1[4];
```

A “Safe” Array Type: Version 4, *cont’d*

SafeArray4.h

```
class SafeArray
{
public:
    SafeArray();
    SafeArray(int len);
    ~SafeArray();

    int get_length() const;

    int at(int i) const;
    void set(int i, int value);

    SafeArray& operator =(const SafeArray& rhs);
    int& operator [](int i) const;

private:
    int *elements;
    int length;
};
```

A “Safe” Array Type: Version 4, *cont’d*

```
int& SafeArray::operator [](int i) const
{
    assert((i >= 0) && (i < length));
    return elements[i];
}
```

SafeArray4.cpp

A “Safe” Array Type: Version 4, *cont’d*

SafeArrayTests4.cpp

```
int main()
{
    SafeArray a1(10), a2, a3;

    for (int i = 0; i < 10; i++) a1[i] = 10*i;

    a3 = a2 = a1;
    a1[4] = -a1[4];

    cout << "a1 ="; print(a1);
    cout << "a2 ="; print(a2);
    cout << "a3 ="; print(a3);

    return 0;
}

void print(SafeArray& a)
{
    for (int i = 0; i < a.get_length(); i++) cout << " " << a[i];
    cout << endl;
}
```

```
a1 = 0 10 20 30 -40 50 60 70 80 90
a2 = 0 10 20 30 40 50 60 70 80 90
a3 = 0 10 20 30 40 50 60 70 80 90
```

A “Safe” Array Type: Version 4, *cont’d*

- What if we passed the smart array object <u>by value</u> instead of by reference?

```
void print(SafeArray a)
{
    for (int i = 0; i < a.get_length(); i++)
    {
        cout << " " << a[i];
    }
    cout << endl;
}
```

SafeArrayTests4.cpp

```
a1 = 0 10 20 30 -40 50 60 70 80 90
a2 = 0 10 20 30 40 50 60 70 80 90
a3 = 0 10 20 30 40 50 60 70 80 90
SafeArray4(78650,0x7fffbe7fe3c0) malloc: *** error for object 0x7fbe64c02740:
                                         pointer being freed was not allocated
*** set a breakpoint in malloc_error_break to debug
```

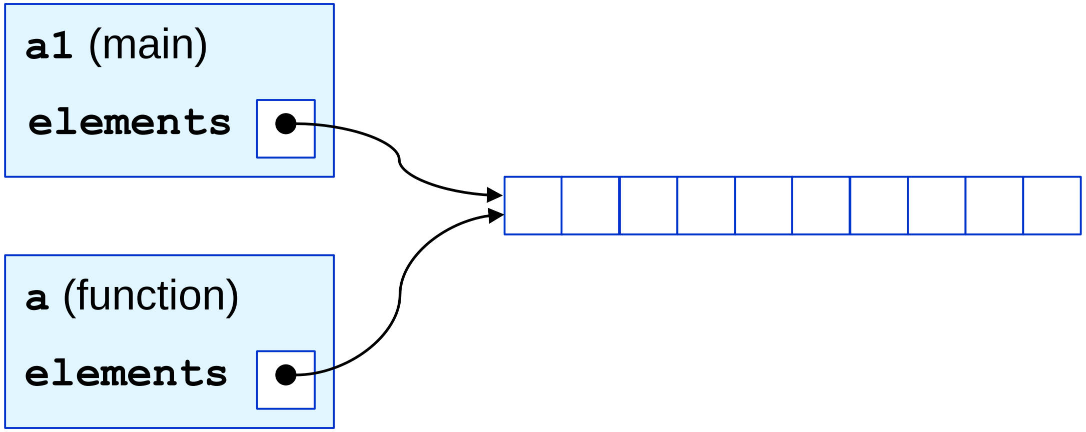
A "Safe" Array Type: Version 5

- A very unexpected side effect!
- At the end, the program attempted to delete the private dynamic array elements.
- But the dynamic array was already deleted by the destructor.
  - So who tried to delete the array again?
- Why did passing a SmartArray object by value instead of by reference to the print function cause this problem?

A “Safe” Array Type: Version 5, *cont’d*

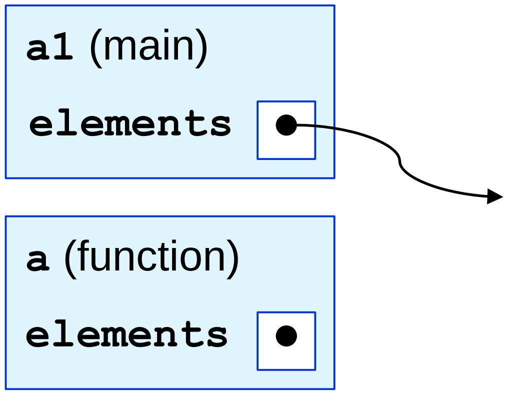
- When a **`SmartArray`** object is passed by value to the **`print`** function, a copy is made.

- This copy will point to the same dynamic array.
  - This is what the default copy constructor does.



A "Safe" Array Type: Version 5, *cont'd*

- When the print function completes and returns, its local variables go out of scope.
- The `SmartArray` object's destructor is called, which deletes the dynamic array.
  - Now variable `a1` has a dangling pointer.
  - When the program is ready to terminate, it calls `a1`'s destructor.
  - An error occurs because of the attempt to delete memory that has already been deleted.



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, which is a 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.

A “Safe” Array Type: Version 5, *cont’d*

SafeArray5.h

```
class SafeArray
{
public:
    SafeArray();
    SafeArray(int len);
    SafeArray(const SafeArray& other);  // copy constructor
    ~SafeArray();

    int get_length() const;

    SafeArray& operator =(const SafeArray& rhs);
    int& operator [](int i) const;

private:
    int *elements;
    int length;
};
```

A “Safe” Array Type: Version 5, *cont’d*

```
SafeArray::SafeArray(const SafeArray& other)
    : elements(nullptr), length(0)
{
    length = other.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = other.elements[i];
    }
}
```

SafeArray5.cpp

- Now the copy of the object has a separate copy of the contents of the `elements` array.

Shorthand for Pointer Expressions

```
class Node
{
public:
    Node(int value);
    ~Node();

    int data;
    Node *next;
};
```

```
Node *head;
```

- The expression **`head->data`** is the preferred shorthand for **`(*head).data`**

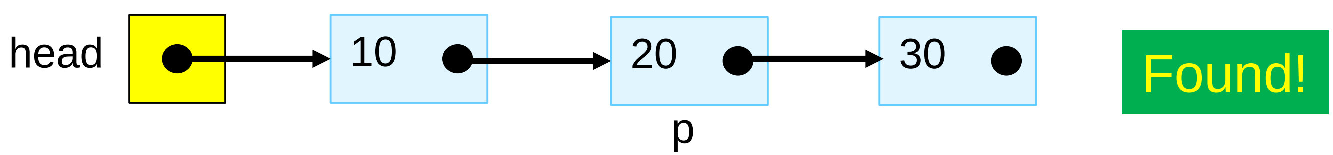
Searching a Sorted Linked List

```
Node *SortedLinkedList::find(int value) const
{
    Node *p = head;

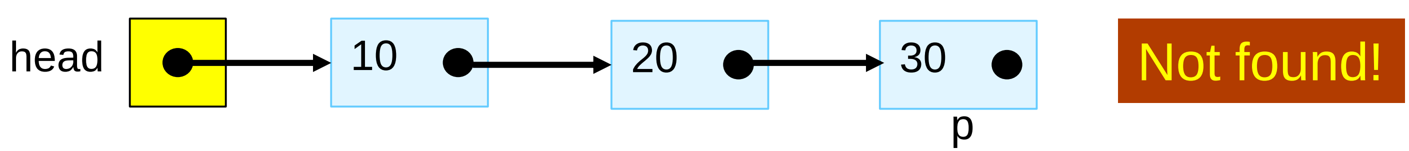
    // Search the sorted list.
    while ((p != nullptr) && (value > p->data)) p = p->next;

    if ((p != nullptr) && (value == p->data)) return p;       // found
    else                                      return nullptr; // not found
}
```

- Search for 20:



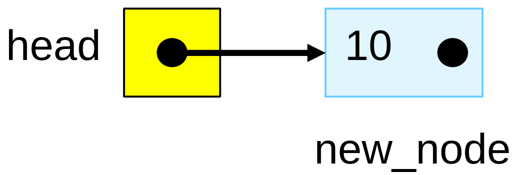
- Search for 25:



Inserting into a Sorted Linked List

- Insert the first element into a sorted linked list.

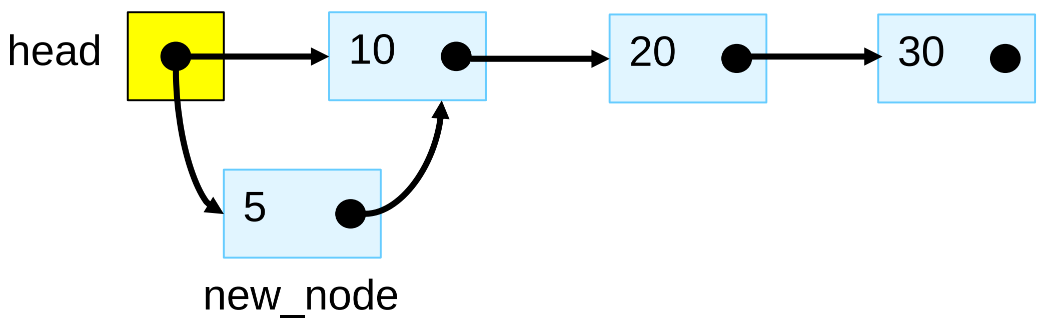
```
if (head == nullptr)
{
    head = new_node;
    return new_node;
}
```



Inserting into a Sorted Linked List, *cont'd*

- Insert at the beginning of an existing sorted linked list.

```
else if (value < head->data)
{
    new_node->next = head;
    head = new_node;
    return new_node;
}
```

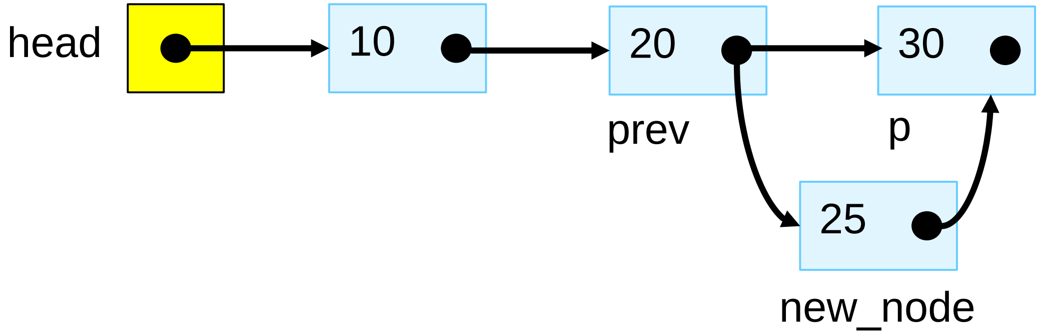


Inserting into a Sorted Linked List, *cont'd*

- Insert into the middle of a sorted linked list.

```
while ((p != nullptr) && (value >= p->data))
{
    prev = p;
    p = p->next;
}

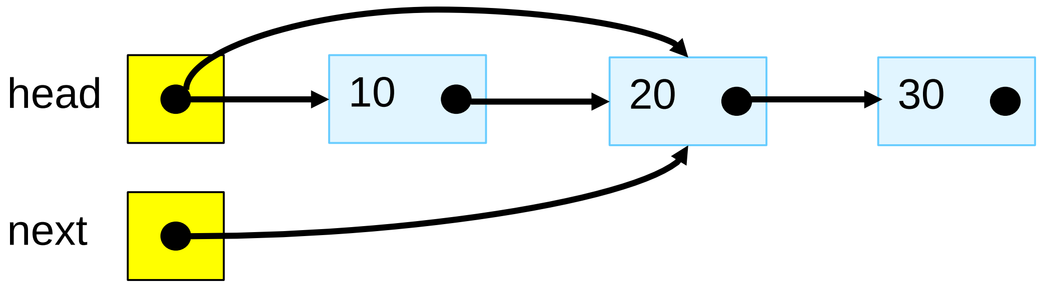
prev->next = new_node;
new_node->next = p;
return new_node;
```



Removing from a Sorted Linked List

- Remove from the head of a sorted list.

```
if (value == head->data)
{
    Node *next = head->next;
    delete head;
    head = next;
    return;
}
```

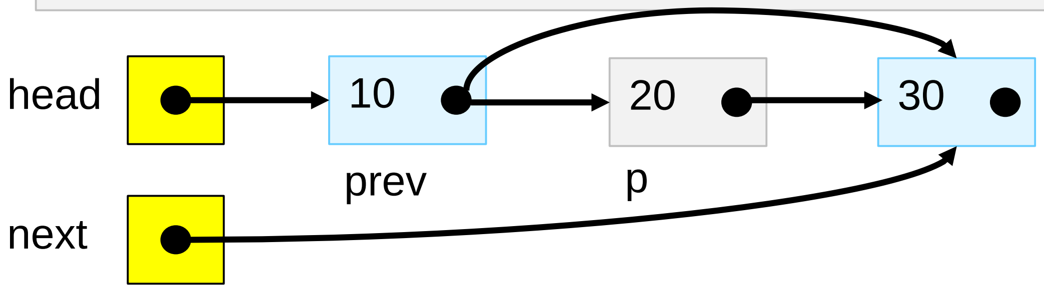


Removing from a Sorted Linked List, *cont'd*

- Remove from the middle of a sorted list.

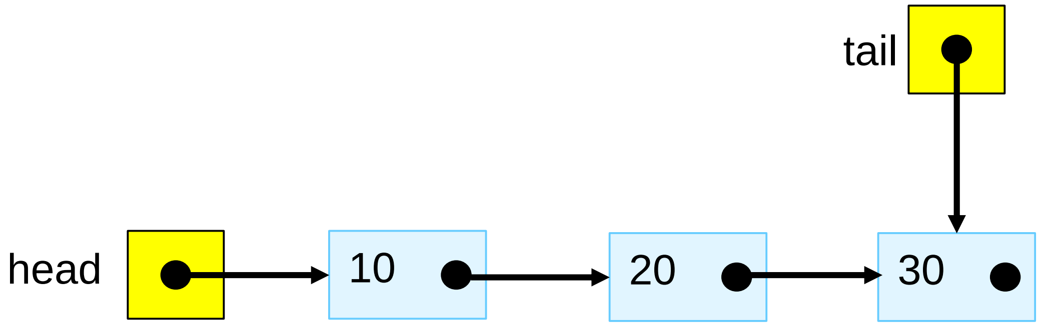
```
while ((p != nullptr) && (value > p->data))
{
    prev = p;
    p = p->next;
}

if ((p != nullptr) && (value == p->data))
{
    Node *next = p->next;
    delete p;
    prev->next = next;
}
```



Linked List Tail

- Often there are advantages for a linked list to maintain both a head pointer and a tail pointer.

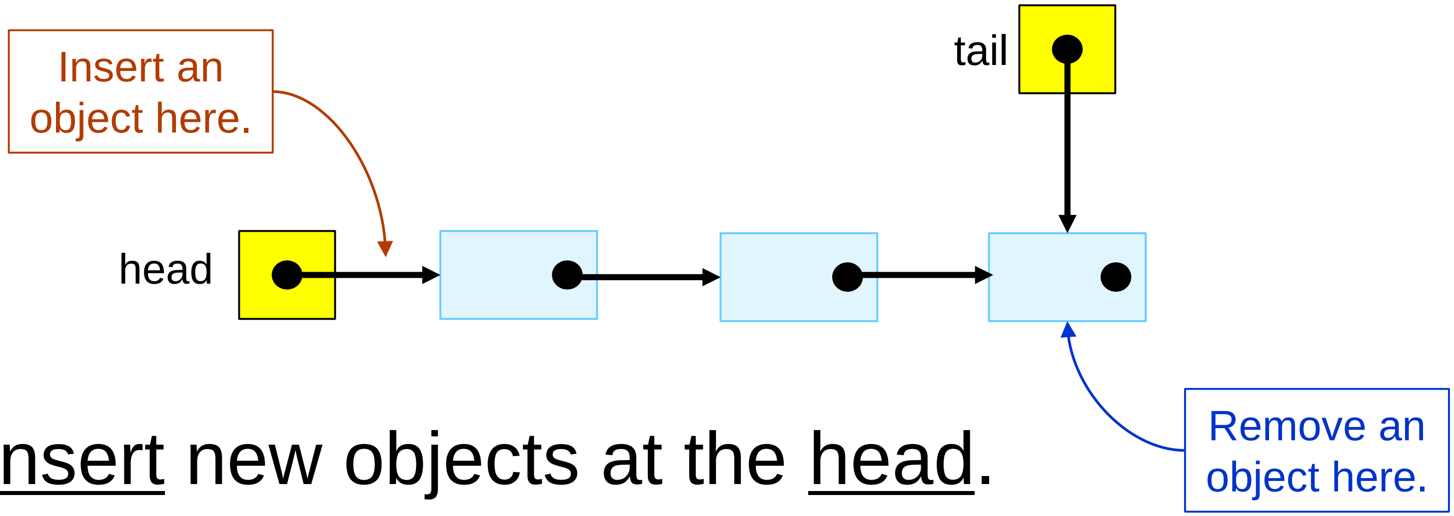


Queue

- A queue is a data structure which you can insert objects into and from which you can remove objects.
- The queue maintains the order that the objects are inserted.
- Objects are removed from the queue in the same order that they were inserted.
- This is commonly known as first-in first-out (FIFO).

Queue, *cont'd*

- We can use a linked list to implement a queue.



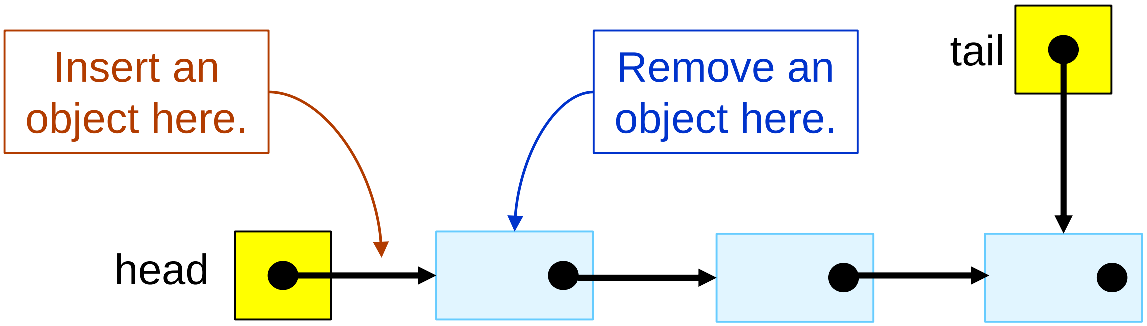
- Insert new objects at the head.
- Remove objects at the tail.
- Objects in the queue are in arrival order.
  - Not necessary for the objects to be in data order.

Stack

- A stack is a data structure into which you can insert objects and from which you can remove objects.
- The stack maintains the order that the objects are inserted.
- Objects are removed from the queue in the reverse order that they were inserted.
- This is commonly known as last-in first-out (LIFO).

Stack, *cont'd*

- We can use a linked list to implement a stack.



- Insert (push) new objects at the head.
- Remove (pop) objects at the head.

Midterm Next Week

- Combination of multiple-choice, short answer, and short programming (such as a function or a class declaration).
- Covers
  - all lectures through today
  - Savitch book chapters 1 – 13
  - assignments 1 – 7
- Closed book and laptop
- 75 minutes

Assignment #7

- Practice with linked lists.
  - Write-up and data files in Canvas by Friday.
- Read from text files containing data about books by various authors.
  - Each book has an ISBN, its author's last and first names, and the book title.
  - Each text file contains books from one category, already sorted by ISBN.
- Create separate linked lists of books from each category; i.e., a linked list per input text file.

Assignment #7, *cont'd*

- Print each category list of books.
- Merge all the separate category lists into a single book list, sorted by ISBN.
- Print the merged list.
- Split the merged list into two sublists, one sublist for authors with last names starting with A – M and the second sublist for authors with last names starting with N – Z,
- Print the two sublists.