Unit 4b

Assignment Idioms
More C++ Statements
Division/Modulo Idioms
Unit Objectives

• Predict the value of variables based on a sequence of assignments
• Apply the swap idiom
• Utilize casting
• Utilize math library functions
Review of Data Types

- **bool**
  - true or false values

- **int or unsigned int**
  - Integer values

- **char**
  - A single ASCII character
  - Or a small integer (but just use 'int')

- **double**
  - A real number (usually if a decimal/fraction is needed) but also for very large numbers

- **string**
  - Multiple text characters, ending with the null ('\0' = 00) character
Common Idioms and Potential Pitfalls

ASSIGNMENT AND ORDERING
Temporal/Sequential Nature of Assignment

• It is critical to realize that assignment:
  – Does **NOT** create a permanent relationship that causes one variable to update if another does
  – Uses the variable values **at the time the line of code is executed**
  – Copies (not moves) data to the destination variable

• So the result of assignment statements depend on the order (timing) in which they are executed because one statement may affect the next

```cpp
int main()
{
    int x = 5;
    // Performs a one-time update of y to 2*5+1=11
    int y = 2 * x + 1;
    // This assignment will NOT cause y to be re-evaluated
    x = 7;
    // y is still 11 and not 15
    cout << "y = " << y << endl;
    // Copies the value of x into y
    y = x;
    // both x and y are 7 now
    cout << x << " " << y << endl;
    return 0;
}
```
Problem Solving Idioms

- An idiom is a colloquial or common mode of expression
  - Example: "raining cats and dogs"
- Programming has common modes of expression that are used quite often to solve problems algorithmically
- We have developed a repository of these common programming idioms. We STRONGLY suggest you
  - Reference them when attempting to solve programming problems
  - Familiarize yourself with them and their structure until you feel comfortable identifying them

### Rule / Exception Idiom

- **Name**: Rule/Exception
- **Description**: Perform a default action and then use an *if* to correct for an exceptional case
- **Structure**: Code for some default action (i.e., the rule) is followed by code for the exceptional case

```cpp
// Default action
if( /* Exceptional Case */ )
{
    // Code for exceptional case
}
```

- **Example(s)**:
  - Base pay plus bonus for certain exceptional employees

```cpp
bool earnedBonus = /* set somehow */;
int bonus = /* set somehow */;

int basePay = 100;
if( earnedBonus == true )
{
    basePay += bonus;
}
```

- **Notes**: This can be implemented with an *if/else* where an *else* implements the other.
Shifting and Rotation Assignment Idioms

- The **shifting idiom** shifts data among variables usually replacing/dropping some elements to make room for new ones
  - The key pattern is some elements get *dropped/overwritten* and other elements are *reassigned/moved*
  - It is important to **start by assigning the variable to be replaced/dropped** and then move in order to variables receiving newer data
  - Examples: Top k items (high score list)

- The **rotation idiom** reorders or rearranges data among variables without replacing/dropping elements
  - Swap is simply a rotation of 2 elements
  - The key pattern is **all elements are kept** but just reordered
  - It is usually necessary to declare and **maintain some temporary variable** to avoid elements getting dropped/overwritten
Shifting Idiom Ex. (Insertion)

• Suppose a business represents each client with a 3-digit integer ID (and -1 to mean "free")
  – Lower IDs are given to more important clients
  – Client's with lower ID's always get the appointment time they want
  – Suppose client 105 calls and wants a 2 p.m. appointment, will the highlighted code below work?

• Shifting or rotation?
  – Are we adding/dropping values or keeping all the originals?

• Recall that statements execute one at a time in sequential order
  – Earlier statements complete fully before the next starts

```c
int main()
{
    // Original appointment schedule
    // Lower client ID gets earlier appointment
    int apt_1pm = 100;
    int apt_2pm = 120;
    int apt_3pm = 140;
    int apt_4pm = -1;

    // Now client 105 wants a 2 p.m. appointment
    apt_2pm = 105;
    apt_3pm = apt_2pm;
    apt_4pm = apt_3pm;

    return 0;
}
```
Shifting Idiom Ex. (Insertion)

• To correctly code the shift, we must start with the variable to be dropped
• The code to the right does not follow this guideline
  – Perform each highlighted operation one at a time, marking up the diagram below to see the error that results

```c
int main()
{
    // Original appointment
    // schedule
    // Lower client ID gets
    // earlier appointment
    int apt_1pm = 100;
    int apt_2pm = 120;
    int apt_3pm = 140;
    int apt_4pm = -1;

    // Now client 105 wants
    // a 2 p.m. appointment
    apt_2pm = 105;
    apt_3pm = apt_2pm;
    apt_4pm = apt_3pm;

    return 0;
}
```
Shifting Idiom Ex. (Insertion)

- To correctly code the shift, we must start with the variable to be dropped
  - Move items in reverse order

```c
int main()
{
    // Original appointment
    // schedule
    // Lower client ID gets
    // earlier appointment
    int apt_1pm = 100;
    int apt_2pm = 120;
    int apt_3pm = 140;
    int apt_4pm = -1;

    // Now client 105 wants
    // a 2 p.m. appointment
    apt_4pm = apt_3pm;
    apt_3pm = apt_2pm;
    apt_2pm = 105;

    return 0;
}
```
Shifting Idiom Ex. (Moving-Window)

• Suppose we only want to work with the last $k$ (let $k=3$ for this example) value input by the user
  – Declare $k$ variables (i.e. $x_1$, $x_2$, $x_3$)
  – As we receive new values we drop the undesired values shifting the current values as needed via assignment operations

```
int x1 = 10, x2 = 20, x3 = 50;
```
Shifting Values (Moving Window) Idiom

• Remember, *order* of assignment is **very important** to avoid overwriting data we still need
• Start by assigning the value to be overwritten/dropped...
• Continue assigning in order until reaching the variable that should receive the new value

```c
int x1 = 10, x2 = 20, x3 = 50;
```
Rotation Idiom Ex. (Swap)

• Given two variables, swap their contents
  – Before:  a = 7, b = 9
  – Desired Result:  a = 9, b = 7

• This is rotation because we want to keep all values and just reorder them

• Since shifting requires us to start with the variable to be overwritten/dropped and we want to keep both values, no order of assignment will work without a temporary variable!

• Perform the code to the right to see the error:
  – Actual Result: a = ___, b = ___;

```cpp
int main()
{
    int a = 7, b = 9;

    // Now suppose we want to swap the values of a and b
    a = b;
    b = a;

    // What will this do?
    return 0;
}
```
Rotation Idiom Ex. (Swap)

- We need an extra, temporary location to hold the old value of one of the variables while we update it to the new value.

```c
int main()
{
    int a = 7, b = 9;

    // Now suppose we want to swap the values of a and b
    // Introduce a temp var.
    int temp = a;
    a = b;
    b = temp;

    return 0;
}
```
MORE OPERATIONS AND USING MATH LIBRARY FUNCTIONS
Shortcut Assignment Statements

• A common task is to update a variable by adding, subtracting, multiplying, etc. some value to it
  - \( x = x + 4 \);
  - \( y = y \times 2.5 \);

• C/C++ provide a shortcut for writing these statements:
  - \( x += 4 \);
  - \( y *= 2.5 \);

• The substitution is:
  - \( \text{var op= expr}; \)
  - Becomes \( \text{var = var op expr}; \)

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

int main()
{
    int x = 1;
    double y = 3.75;
    x += 5;      // x updates to 6
    y -= 2.25;   // y updates to 1.5
    x /= 3;      // x updates to 2
    y *= 2.0     // y updates to 3.0

    return 0;
}
```
Post-Increment/Decrement

• Adding 1 to a variable (e.g. `x += 1`) and subtracting 1 from a variable (e.g. `x -= 1`) are extremely common operations (especially when we cover loops).

• The `++` and `--` operators offer a shortcut to "increment-by-1" or "decrement-by-1"
  - Performs `( x += 1)` or `( x -= 1)`
  - `x++;` // If x was 2 it will be updated to 3 (x = x + 1)
  - `x--;` // If x was 2 it will be updated to 1 (x = x - 1)

• Note: There are some nuances to this operator and an alternative known as pre-increment/decrement that we will discuss in future lectures but this is sufficient for now.
Casting Motivation

- To achieve the correct answer for $5 + 3 / 2$ we could...
- Make everything a double
  - Write $5.0 + 3.0 / 2.0$ [explicitly use doubles]
- Use **implicit** casting (mixed expression)
  - Could just write $5 + 3.0 / 2$
    - If operator is applied to mixed type inputs, less expressive type is automatically promoted to more expressive (int => double)
- But what if instead of constants we have variables
  - int x=5, y=3, z=2;
    x + y/z;  // Won't work & you can't write y.0
- We need a way to explicitly cast a variable to a different type for the sake of a computation
Casting

• To cast a variable, place the type to which you want to cast in parentheses BEFORE the variable.

• Casting is the only way to convert a variable to a different numeric type.
  
  - `x + (double) y / z;` // z will be implicitly cast to a double.

• This won't work.
  
  - `x + (double) (y / z);` // the integer division in parens goes first.

• Notes:
  
  - Only changes the type temporarily for the sake of the expression (not a permanent type change).
  
  - Only works on numeric types and not strings.

  • Can't cast an integer/double to a character or string.
  
  • `double x = 1.6; int y = (int) x / 2;` // fine!

  • `int x = 123; string y = (string) x;` // doesn't work

  • `int x = (string) "123";` // doesn't work
Math & Other Library Functions

- C++ predefines a variety of functions for you. Here are a few of them:
  - `sqrt(x)`: returns the square root of x (in `<cmath>`)
  - `pow(x, y)`: returns \(x^y\), or x to the power y (in `<cmath>`)
  - `sin(x)/cos(x)/tan(s)`: returns the sine of x if x is in radians (in `<cmath>`)
  - `abs(x)`: returns the absolute value of x (in `<cstdlib>`)
  - `max(x, y)` and `min(x, y)`: returns the maximum/minimum of x and y (in `<algorithm>`)  

- You call these by writing them similarly to how you would use a function in mathematics [using parentheses for the inputs (aka) arguments]
- Result is replaced into bigger expression
- Must `#include` the correct library
  - #includes tell the compiler about the various pre-defined functions that your program may choose to call

```cpp
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

int main()
{
    // can call functions
    // in an assignment
    double res = cos(0); // res = 1.0

    // can call functions in an
    // expression
    res = sqrt(2) / 2; // res = 1.414/2

    cout << max(34, 56) << endl; // outputs 56

    return 0;
}
```

http://www.cplusplus.com/reference/cmath/
Statements

• C/C++ programs are composed of statements
• Most common kinds of statements end with a semicolon
• Declarations (e.g. `int x=3;`)
• Assignment + Expression (suppose `int x=3; int y;`)
  – `x = x * 5 / 9;`  // compute the expression & place result in x
  // `x = (3*5)/9 = 15/9 = 1`
• Assignment + Function Call (+ Expression)
  – `x = cos(0.0) + 1.5;`
  – `sin(3.14);`  // Must save or print out the result (x = sin(3.14), etc.)
• `cin, cout` statements
  – `cout << cos(0.0) + 1.5 << " is the answer." << endl;`
• Return statement (immediately ends a function)
  – `return value;`
  – More on this in Unit 6
I/O Manipulators

- Manipulators control HOW `cout` handles certain output options and how `cin` interprets the input data (but print nothing themselves)
  - Must `#include <iomanip>`
- Common examples
  - `setw(n)`: Separate consecutive outputs by `n` spaces
  - `setprecision(n)`: Use `n` digits to display doubles (both the integral + decimal parts)
  - `fixed`: Uses the precision for only the digits after the decimal point
  - `boolalpha`: Show Booleans as `true` and `false` rather than 1 and 0, respectively
- Separated by `<<` or `>>` and used inline with actual data
- Other than `setw`, manipulators continue to apply to other output until changed

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

int main()
{
    double pi = 3.14159;

    cout << pi << endl;  // Prints: 3.14159

    cout << setprecision(2) << fixed << pi << endl;  // Prints: 3.14

    return 0;
}
```


See "iomanip" in-class exercise to explore various options
Exercises

• Exercises:
  – cpp/cin/average
  – cpp/cin/rad2deg

• Write a program to convert temperature from Celsius to Fahrenheit \[ F = \frac{9}{5} \cdot C + 32 \]
  – Use http://cpp.sh or http://onlinegdb.com