Unit 1b – Processing Information using Expressions

Mark Redekopp
VARIABLES AND ASSIGNMENT
Variable Review: I Do Declare

• (Unlike Python) you must do a one-time declaration of a variable before using it
  – Like renting an apartment or storage unit
• If not initialized via assignment ('='), variables will NOT default to a value like 0, but will contain random data/garbage.
  – Good practice to initialize your variables
• C++ is a strongly-typed language which means...
  – You cannot change what type of value the variable stores); this is because in C++ a variable name corresponds to a reserved, fixed-size memory location that only fits that specific type

```c++
#include <iostream>
using namespace std;
int main() {
    int x; // it is declared
    // Error: x assigned before it is declared
    int x; // uninitialized variables
    // will have a (random) garbage value until we initialize it
    x = x+1; // BAD. X is uninitialized still
    x = 1; // Initialize x's value to 1

    double y = 3.14;
    x = "pi is"; // Error: x declared as int cannot be assigned a string
    cout << x << " " << y << endl;
    return 0;
}
```

C++ is "strongly-typed" and requires variables to be declared before being used.

```python
def main():
    x = 5  # x stores an integer
    y = 3.14
    x = "pi is"  # x changes to store a string
    print(x, y)
```

Python does not require explicitly declaring and typing a variable.
C/C++ Variables

- Variables have a:
  - **type** [int, char, unsigned int, float, double, etc.]
  - **name/identifier** that the programmer will use to reference the value in that memory location [e.g. x, myVariable, num_dozens, etc.]
    - Identifiers must start with [A-Z, a-z, or an underscore '_'] and can then contain any alphanumeric character [0-9, A-Z, a-z, _] (but no punctuation other than underscores)
    - Use descriptive names (e.g. numStudents, doneFlag)
    - Avoid cryptic names (myvar1, a_thing)
  - **location** [the address in memory where it is allocated]
  - **Value**

- Reminder: You must declare a variable before using it

```c
int quantity = 4;
double cost = 5.75;
cout << quantity*cost << endl;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantity</td>
<td>1008412</td>
<td>4</td>
</tr>
<tr>
<td>cost</td>
<td>287144</td>
<td>5.75</td>
</tr>
</tbody>
</table>
C/C++ Variable Types

- A **type** indicates how many bits / bytes of **storage** (memory) are required and how to **interpret** the number being stored.

- **Integer types**
  - Are **signed** (numbers can be positive or negative) by default, or **unsigned** (positive-only...including 0)
  - A **character** (more on this later)

- **Floating point types**: Very large 6.02E23 & very small numbers 6.626E-34)
  - A **float** or **double**

- **String/Text types**
  - A single **char** (1 character)
  - **character arrays** (C-Strings) / **string** (C++ string type)

- **Boolean type**
  - **bool** (true / false)

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

int main()
{
    int a = -1;
    unsigned int b = 2;
    char c = 'A';  // 'A'=65
    float d1 = 1.5;
    double d2 = 3.14;
    char e[6] = "Hello";
    string f = "Goodbye";
    bool g = true;
    // ...
}
```
Know Your Common Variable Types

- Variables are declared by listing their type and providing a name.
- They can be given an initial value using the '=' operator.

<table>
<thead>
<tr>
<th>C Type</th>
<th>Usage</th>
<th>Bytes</th>
<th>Bits</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>Text character</td>
<td>1</td>
<td>8</td>
<td>ASCII characters -128 to +127</td>
</tr>
<tr>
<td></td>
<td>Small integral value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>True/False value</td>
<td>1</td>
<td>8</td>
<td>true / false</td>
</tr>
<tr>
<td>int</td>
<td>Integer values</td>
<td>4</td>
<td>32</td>
<td>-2 billion to +2 billion</td>
</tr>
<tr>
<td>unsigned int</td>
<td></td>
<td></td>
<td></td>
<td>0 to +4 billion</td>
</tr>
<tr>
<td>double</td>
<td>Rational/real values</td>
<td>8</td>
<td>64</td>
<td>±16 significant digits * 10^±308</td>
</tr>
<tr>
<td>string</td>
<td>Arbitrary text</td>
<td>1 byte per char</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

```cpp
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
    int w = -400;
    double x = 3.7;
    char y = 'a';
    bool z = false;
    cout << w << " " << x << " ";
    cout << y << " " << z << endl;
    return 0;
}
```
When To Introduce a Variable

• When a value will be supplied and/or change at run-time (as the program executes)

• When a value is computed/updated at one time and used (many times) later

• To make the code more readable by another human

double a = (x+34) * (n*6.25);
// readability of above vs. below
double height = x + 34;
double width = n * 6.25;
double area = height * width;
What Variables Might Be Needed

• Calculator App
  —

• Video playback (YouTube player)
  —
What Variables Might Be Needed

- Calculator App
  - Current number input, current result
- Video playback (YouTube player)
  - Current URL, full screen, volume level
Scope

• "Scope" of a variable refers to the
  – Visibility (who can access it) and
  – Lifetime of a variable (how long is the memory reserved

• For now, there are 2 scopes we will learn
  – Global: Variables are declared outside of any function and are visible to all the code/functions in the program
  – Local: Variables are declared inside of a function and are only visible in that function and die when the function ends

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

// Global Variable
int x=1;

int add_x(int input)
{
    // y and z NOT visible (in scope) here
    // but x is since it is global
    return (input + x);
} // input dies here

int main()
{
    // y and z are "local" variables
    int y, z=5; // y is garbage, z is five

    z = add_x(z);
    y += z;     // BAD!! Why?
    cout << x << " " << y << endl;
    return 0;
} // y and z die here
```
Assignment operator (=)

- Syntax:

  \[
  \text{variable} = \text{expression};
  \]

  \begin{align*}
  \text{(LHS)} & \quad \text{(RHS)} \\
  \text{LHS} = \text{Left Hand-Side}, & \quad \text{RHS} = \text{Right Hand Side}
  \end{align*}

- Should be read: Place the value of \text{<expression>}
  into memory location of \text{<variable>}

  \begin{align*}
  - & \quad z = x + y - (2*z); \\
  - & \quad \text{If variable appears on both sides, we use the old/current value of the variable on the RHS}
  \end{align*}

- **Note:** Without assignment values are computed and then forgotten

  \begin{align*}
  - & \quad x + 5; \quad \text{// will take x's value add 5 but NOT update x (just throws the result away)} \\
  - & \quad x = x + 5; \quad \text{// will actually updated x (i.e. requires an assignment)}
  \end{align*}
Assignment Means Copy

- Assigning a variable makes a **copy**
  - It leaves the source variable unchanged
  - Is performed immediately and takes effect before the next statement
- Challenge: Swap the value of 2 variables

```c
int main()
{
    int x = 5, y = 3;
    x = y;    // copy y into x
    // y still has 3
    return 0;
}
```

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

    // now consider swapping
    // the value of 2 variables
    a = b;
b = a;

    return 0;
}
```
More Assignments

• Assigning a variable makes a **copy**
  – It leaves the source variable unchanged

• Example: Swap the value of 2 variables
  – Easiest method: Use a 3rd temporary variable to save one value and then replace that variable

• Challenge: 4swap exercise

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

    // let's try again
    temp = a;
    a = b;
    b = temp;

    cout << a << " " << b << endl;
    return 0;
}
```
Assignment (\(=\)) Operator

- To update or change a value in a variable we use the assignment operator (=)

- Syntax:
  - variable = expression;
    (Left-Side) (Right-side)

- Semantics:
  - Place the resulting value of 'expression' in the memory location associated with 'variable'
  - Does not mean "compare for equality" (e.g. is \(w\) equal to 300?)
    - That is performed by the \(==\) operator

```cpp
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
    int w; // variables don't have to be initialized when declared
    char x; // be initialized when declared

    w = 300;
    x = 'a';
    cout << w << " " << x << endl;

    w = -75;
    x = '!' ;
    cout << w << " " << x << endl;
    return 0;
}
```

Order of evaluation: right to left

```
variable = expression;
```

Assignment is one of the most common operations in programs
Assignment & Expressions

- `=' means assign/copy, NOT "compare for equality" (e.g. is dozens equal to 3...that is performed by the '==' operator)
- Variables can be used in expressions and be operands for arithmetic and logic
- See inset below on how to interpret a variable's usage based on which side of the assignment operator it is used

```
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
    int dozens = 3;
    double gpa = 2.0;

    int num = 12 * dozens;
    gpa = (2 * 4.0) + (4 * 3.7); // gpa updated to 22.8
    gpa = gpa / 6; // integer or double division?

    cout << dozens << " dozen is " << num << " items." << endl;
    cout << "Your gpa is " << gpa << endl;
    return 0;
}
```

Order of evaluation: right to left

```
int x = 0;
x = x + 3;
```

Semantics of variable usage:
- Right-side of assignment: Substitute/use the current value stored in the variable
- Left-side of assignment: variable is the destination location where the result of the right side will be stored
Exercises

• What is printed by the following two programs?

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

int main()
{
    int value = 1;
    value = (value + 5) * (value - 3);
    cout << value << endl;

    double amount = 2.5;
    value = 7;
    amount = value + 6 / amount;
    cout << amount << endl;
    cout << value % 3 << endl;
    return 0;
}

#include <iostream>
using namespace std;

int main()
{
    int x = 5;
    int y = 3;
    double z = x % y * 6 + x / y;
    cout << z << endl;
    z = 1.0 / 4 * (z - x) + y;
    cout << z << endl;
    return 0;
}
```
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:
  - var op= expr;
  - Becomes var = var op expr;
- **Shorthand operators** exist for most operators:
  - +=, -=, *=, /=, %, &=, ...

```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 AND USING MATH LIBRARY FUNCTIONS
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/
#include Directive

- Common usage: To include “header files” that allow us to access functions defined in a separate file or library.
- For pure C compilers, we include a C header file with its filename: `#include <stdlib.h>`
- For C++ compilers, we include a C header file without the `.h` extension and prepend a ‘c’: `#include <cstdlib>`

<table>
<thead>
<tr>
<th>C</th>
<th>Description</th>
<th>C++</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdio.h</td>
<td>C Input/Output/File access (printf, fopen, snprintf, etc.)</td>
<td>iostream</td>
<td>I/O and File streams (cin, cout, cerr)</td>
</tr>
<tr>
<td>cstdio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stdlib.h</td>
<td>rand(), Memory allocation, etc.</td>
<td>fstream</td>
<td>File I/O (ifstream, ofstream)</td>
</tr>
<tr>
<td>cstdlib</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>string.h</td>
<td>C-string library functions that operate on character arrays</td>
<td>string</td>
<td>C++ string class that defines the ‘string’ object</td>
</tr>
<tr>
<td>cstring</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>math.h</td>
<td>Math functions: sin(), pow(), etc.</td>
<td>vector</td>
<td>Array-like container class</td>
</tr>
<tr>
<td>cmath</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Casting Motivation

• **Def.** casting: *Temporarily converting the type of a data value*

• What is the result of $5 + 3/2$?
  • To achieve the correct answer for $5 + 3/2$ we could...

• Use **implicit** casting (mixed expression)
  – Could just write $5 + 3.0/2$
    • If an operator is applied to mixed type inputs, less expressive type is automatically and implicitly cast (promoted) to the more expressive (int is promoted to 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 can perform an **explicit** cast using either the C or C++ syntax
  – `x + (double) y / z; // C style casting method`
  – `x + static_cast<double>(y) / z; // C++ style casting method`

• **BE CAREFUL!!** This won't yield the 6.5 answer you expect.
  – `x + static_cast<double>(y/z); // Why not?`
Common Casting Errors

• Only changes the type **temporarily** for the sake of the expression (not a permanent type change)

• **Casting only really works on numeric types and NOT strings**
  – Different than many other languages like Python
  – When converting to/from a string, do **NOT** use casting, but functions from the string library (to_string(), stoi(), stod(), etc.)

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

int main() {
    double a = 3.6;
    int b = static_cast<int>(a) / 2;  // Works! b = 1 (casts 3.6 to 3)
    int c = 123;
    string d = static_cast<string>(c);  // Error! Doesn't compile.
    string d = to_string(c);  // Works! But only since C++11
    string e = "42";
    int f = static_cast<int>(e);  // Error! Doesn't compile.
    int f = stoi(e);  // string-to-int
    // Works! But only since C++11
    // use stod() for string-to-double
    return 0;
}
```
Statements

- C/C++ functions 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 + Expressions
  - cout << cos(0.0) + 1.5 << " is the answer." << endl;
- Return statement (immediately ends a function)
  - return expression;  // (more on this later)
Exercises

- Exercises:
  - average
  - rad2deg

- Write a program to convert temperature from Celsius to Fahrenheit \( F = \frac{9}{5} \cdot C + 32 \)
  - Use [http://cpp.sh](http://cpp.sh) or [http://onlinegdb.com](http://onlinegdb.com) (or EdStem Workspace, if available)
SOLUTIONS
Exercises

• What is printed by the following two programs?

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

int main()
{
    int value = 1;
    value = (value + 5) * (value - 3);
    cout << value << endl;

    double amount = 2.5;
    value = 7;
    amount = value + 6 / amount;
    cout << amount << endl;
    cout << value % 3 << endl;
    return 0;
}
```

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

int main()
{
    int x = 5;
    int y = 3;
    double z = x % y * 6 + x / y;
    cout << z << endl;
    z = 1.0 / 4 * (z - x) + y;
    cout << z << endl;
    return 0;
}
```

-12
9.4
1

13 // or 13.0
5 // or 5.0