Week 1

Operators and Data Types, I/O

Gaddis: Chapters 1, 2, 3

CS 5301
Spring 2014

Jill Seaman

---

Structure of a C++ Program

- **Hello world:**

```cpp
// This program outputs a message to the screen
#include <iostream>
using namespace std;

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

- **In general:**

```cpp
// This is a comment
#include <includefile> ...
using namespace std;

int main() {
    statements ...
}
```

---

Variables, Data Types

- **Variable**: portion of memory that stores a value
- **Identifier**: name of a program element
- **Fundamental data types**
  - short
  - int
  - long
  - float
  - double
  - bool
  - char

- **Variable Declaration** statement

```
data type identifier;
```

```
float hours;
```

- **Variable Initialization** statement:

```
data type identifier = constant;
```

```
int count = 0;
```

---

Integer types

- Integers are whole numbers such as 12, 7, and -99

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>-23,768 to 32,767</td>
</tr>
<tr>
<td>int</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
</tbody>
</table>

- **char** type stores characters such as ‘A’, ‘@’, and ‘9’
  - The ascii code value (an integer) of the character is stored in memory.
Floating-point types
(and bool)

- Floating point types store real numbers such as 12.45 and -3.8
- They are stored using scientific notation.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>float</code></td>
<td>±3.4E-38 to ±3.4E38</td>
</tr>
<tr>
<td><code>double</code></td>
<td>±1.7E-308 to ±1.7E308</td>
</tr>
<tr>
<td><code>long double</code></td>
<td>±1.7E-308 to ±1.7E308</td>
</tr>
</tbody>
</table>

- `bool` type stores values that are true or false
  - false is 0, true is 1.

Constants

- **Literals** (specific value of a given type)

<table>
<thead>
<tr>
<th></th>
<th>Float</th>
<th>Double</th>
<th>LongDouble</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>12.45</td>
<td>true</td>
</tr>
<tr>
<td>75</td>
<td>-3.8</td>
<td>false</td>
<td>‘A’</td>
</tr>
<tr>
<td>-2</td>
<td>6.25e-5</td>
<td></td>
<td>‘2’</td>
</tr>
</tbody>
</table>

- **Named Constants**: variable whose value cannot be changed

```cpp
const datatype identifier = constant;
```

```
const double TAX_RATE = 0.0675;
```

Assignment statement, expressions

- To change the value of a variable:

```cpp
variable = expression; count = 10;
```

- The lefthand side must be a variable
- The righthand side is an expression of the right type

- What is an expression?
  - an expression has a type and evaluates to a value
    - literal
    - named constant
    - variable
    - arithmetic expression
    - etc.

Arithmetic Operations

- **arithmetic operators**:

  ```cpp
  x + 10
  7 * 2
  8 - 5 * 10
  (3 * 10) / 2
  ```

- **% modulo (remainder)**

- **Integer division**:
  ```cpp
  14 / 3 = 4 r. 2 (because 4*3+2 = 14)
  14/3 => 4 in C++
  14%3 => 2 in C++
  14.0/3.0 => 4.6666667 in C++
  ```
Operator precedence

- In an expression with multiple operators, which one happens first?
- Use this order for different operators:
  - + - (unary)
  - * / %
  - + - (binary)
  - < <= > >=
  - == !=
  - && ||
- We will study relational and logical operators next week.

Use this order for multiple occurrences of the same operator
  - - (unary negation) associates right to left
  - *, /, %, +, - associate left to right

Basic Input/Output

- **Output (cout and <<)**
  ```
  cout << expression;
  cout << expr1 << expr2;
  cout << "hello";
  cout << "Count is: " << count << endl;
  ```
- **Input (cin and >>)**
  ```
  cin >> variable;
  cin >> var1 >> var2;
  cout << "Enter the height and width: ";
  cin >> height >> width;
  cout << "The height is " << height << endl;
  ```
  right hand side must be a variable!

Formatting output

- Goal: control how output displays for numeric data
- These require #include<iomanip>
- `setw(x)`: print next value in a field at least x spaces wide (right justified, padded with spaces).
  ```
  cout << setw(6) << 1234 << setw(6) << 5 << endl;  
  cout << setw(6) << 5 << setw(6) << 1234 << endl; 
  ```
- `fixed`: always use decimal notation (not scientific)
- `setprecision(x)`: when used with fixed, print floating point values using x digits after the decimal
  ```
  cout << fixed << setprecision(2); 
  cout << 3.14159 << endl; 
  float x = 20; 
  cout << x << endl; 
  ```

The string class

- **string literals**: represent sequences of chars:
  ```
  cout << "Hello";
  ```
- To define string variables:
  ```
  string firstName, lastName; 
  ```
- Operations include:
  - `=` for assignment
  - `.size()` function for length
  - `[n]` to access one character in the nth position
  ```
  string name = "George";
  cout << name.size() << " "; 
  cout << name[2] << endl; 
  ```
Type conversions

- Implicit
  - assignment:
    ```
    int x;
    double d = 3.1415;
    x = d;
    cout << x << endl;
    ```
  - binary operations:
    ```
    int x = 10;
    double d = 2.3;
    cout << x + d << endl;
    ```

- Explicit
  ```
  int x, y;
  ...
  float avg = static_cast<float>(x)/y;
  ```
  ```
  or
  float avg = x/(float)y; //c-style notation
  ```

Order of types:
- long double
- double
- float
- long
- int
- char

Programming Style

- The visual organization of the source code
- Purpose: improve the readability of the source code
- Includes the use of spaces, tabs, and blank lines
- Includes naming of variables, constants.
- Includes where to use comments.
- Common elements to improve readability:
  - Braces { } aligned vertically
  - Indentation of statements within a set of braces
  - Lines shorter than 80 characters.