Chapter 3 – Variables and Arithmetic Operations
First Program – volume of a box

/******************************************************************************
/*  Program chapter1
/*
/*
/*  This program computes the volume of a box
/*
/*******************************************************************************/
#include <iostream>
using namespace std;

int main()
{
    // Declare and initialize objects
double length(20.75), width(11.5), height(9.5), volume;

    // Calculate volume.
volume = length * width * height;

    // Print the volume.
cout << "The volume is " << volume << endl;

    // Exit program.
    return 0;
}

/*******************************************************************************/
# C++ Data Types

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Example of a constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>char</td>
<td>‘5’</td>
</tr>
<tr>
<td>int</td>
<td>25</td>
</tr>
<tr>
<td>double</td>
<td>25.0</td>
</tr>
<tr>
<td>string</td>
<td>“hello” //must include &lt;string&gt;</td>
</tr>
</tbody>
</table>
Identifiers are used to name entities in C++.

Rules for construction of identifiers
- Start with a letter or underscore _
- Consist of letters digits and underscore
- Can not be a reserved word.
- Only first 31 characters used to distinguish it from other identifiers.
- Case sensitive
Variable Declarations

Declarations define memory locations, including type of data to be stored, identifier, and possibly an initial value.

General Form:

```
data_type identifier_list;
```

Examples:

```
double length(20.75), width(11.5), volume;
int numberOfFeetInYard(3);
```
Symbolic Constants

- Used to name values which do not change during the execution of the program.
- Are always initialized at declaration.
- Used wherever an expression is allowed.

General Form:

```
const data_type identifier = value;
```
Assignment Statements

- Used to assign a value to a variable

General Form:

```
identifier = expression;
```

- Example 1 - initialization

```
double sum = 0;    // sum
```

- Example 2

```
int x;
x = 5;            // x
```

- Example 3

```
char ch;
ch = 'a';          // ch
```
Assignment Statements - continued

- Example 3
  ```
  int x, y, z;
  x=y=0;
  z=2;
  ```

- Example 4
  ```
  y=z;
  ```
Arithmetic Operators

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Modulus: %
  - Modulus returns remainder of division between two integers
  - Example
    \[5\%2\] returns a value of 1
Integer Division

- Division between two integers results in an integer.
- The result is truncated, not rounded
- Example:
  - $5/3$ is equal to 1
  - $3/6$ is equal to 0
Priority of Operators

1. Parentheses Inner most first
2. Unary operators Right to left
   (+ -)
3. Binary operators Left to right
   (* / %)
4. Binary operators Left to right
   (+ -)
Self-test - Evaluate

- $7 + 3 \times 5 - 2$
- $4 + 7 / 3$
- $8 \% 3 \times 6$
- $(7 + 3) \times 5 - 2$
Increment and Decrement Operators

- **Increment Operator** `++`
  - post increment `x++;`
  - pre increment `++x;`

- **Decrement Operator** `--`
  - post decrement `x--;`
  - pre decrement `--x;`

- For examples assume `k=5` prior to executing the statement.
  - `m = ++k;` both `m` and `k` become 6
  - `n = k--;` `n` becomes 5 and `k` becomes 4
<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parentheses: ()</td>
<td>Innermost first</td>
</tr>
<tr>
<td>2</td>
<td>Unary operators</td>
<td>Right to left</td>
</tr>
<tr>
<td></td>
<td>+ - ++ -- (type)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Binary operators</td>
<td>Left ot right</td>
</tr>
<tr>
<td></td>
<td>* / %</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Binary operators</td>
<td>Left ot right</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Assignment operator</td>
<td>Right to left</td>
</tr>
<tr>
<td></td>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>
Simple I/O - cin

- cin
  - is an istream object
  - streams input from standard input
  - uses the >> (input operator)

General Form:

```
cin >> identifier >> identifier;
```

Note: Data entered from the keyboard must be compatible with the data type of the variable.
Simple Output - cout

- **cout**
  - is an ostream object
  - streams output to standard output
  - uses the `<<` (output) operator

General Form:
```
cout << expression << expression;
```

Note: An expression is any C++ expression (string constant, identifier, formula or function call)
//Example1 for input and output
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int i, j;
double x;
string units = " cm";
cin >> i >> j;
cin >> x;
cout << "output \n";
cout << i << ‘,’ << j << ‘,’ << endl
    << x << units << endl;
return 0;
}
// Input stream:
1 2 4.5 cm
output
1,2,4.5 cm
//Example 2 of input and output
#include <iostream>
using namespace std;
int main()
{
    int i, j;
    double x, y;
    cin >> i >> j >> x >> y;
    cout << "First output " << endl;
    cout << i << ',' << j << ',' << x << ',' << y << endl;
    cin >> x >> y >> i >> j;
    cout << "Second output" << endl;
    cout << i << ',' << j << ',' << x << ',' << y << endl;
    return 0;
}  //Input stream is:
1 2
3.4 5
2 3 3 7
Characters and input

- `>>` discards leading whitespace
- `get()` method used to input whitespace characters
- Example:
  ```
  int x;
  char y;
  cin >> x >> y;
  cin >> x;
  cin.get(y);
  ```

Problem: Distance between two points

- Compute the distance between two points.
- Method for solving it:
  - Input?
  - Output?
  - Walk-through an example
  - Stepwise solution (pseudo code)
  - Code
  - Test
  - Verify
Math Functions

- Need cmath or cstlib headers
  
  ```cpp
  #include <cmath> or #include <cstlib>
  ```

- General form: \( name(parameters) \)

- Note what type and form parameters take
  - Trig functions use radian measure not angle

- Table 3.11 lists math library functions