cisc1110 fall 2010 lecture II.1

- simple data types
- storing data
- integers and floating-point numbers
- variables
- assignment statements
- math operators
- outputting variables
- floating-point divide versus integer divide

storing data

- think of the computer’s memory as a bunch of boxes
- inside each box, there is a number
- you give each box a name
  ⇒ which, in C++ is called “declaring a variable”
- example:
  
  program code:
  ```
  int x;
  ```
  
  computer’s memory:
  ```
  x
  ```

- in the example program code above, the name of the box is `x`
- preceding the name `x` is the word `int`, which is a “data type”
- data types are necessary to define what kind of data can be stored in the box
- different kinds of data take up different amounts of space in the computer’s memory
- when you declare a variable, you need to specify its data type so that the computer knows how much memory to set aside for that variable

integers and floating-point numbers

- note that for integers, unless you are writing embedded programs (i.e., code to run on a small device with limited memory), then you don’t need to worry too much about the different size integers — just use `int`
- `double` can store up to 15 significant digits
- `float` can store up to 7 significant digits

Remember:

- integers are for storing whole numbers
  i.e., numbers without any fractional components
  e.g., 1 or −3 or 10 or 987 or 143, 234, etc.
- floating-point numbers are for storing real numbers
  i.e., numbers with decimal places
  e.g., 1.23 or −12.4598 or 1, 234, 567, 890.0987654321

- the data type also tells the computer what kinds of “operations” can be performed with the data
- data types can be “numeric”, “character” or “boolean”
- numeric data types store numbers of various sizes:
  - `int`: integer, for storing whole numbers (typically $-2^{31} \ldots 2^{31} - 1$)
  ⇒ the smallest integer is: $-2^{31} = -2,147,483,648$
  ⇒ the largest integer is: $2^{31} - 1 = 2,147,483,647$
  - `short`: short integer, for storing small integers ($-2^{15} \ldots 2^{15} - 1$)
  - `byte`: one-word integer, for storing even smaller integers ($-128 \ldots 127$)
  - `long`: integer, for storing whole numbers ($-2^{31} \ldots 2^{31} - 1$)
  - `float`: floating-point, for storing real numbers ($\approx -3.4 \times 10^{38} \ldots 3.4 \times 10^{38}$)
  - `double`: floating-point, for storing larger real numbers ($\approx -1.7 \times 10^{308} \ldots 1.7 \times 10^{308}$)
- character data types store alphanumeric characters:
  - `char`: character, for storing one alphanumeric character
- boolean data types store true/false values:
  - `bool`: boolean, for storing 0 (false) or 1 (true)
variables

- each variable has:
  - a name
  - a data type
  - a value

- naming rules:
  - names may contain letters and/or numbers
  - but cannot begin with a number
  - names may also contain underscore (_)
  - can be of any length
  - cannot use C++ keywords (also called identifiers)
  - remember: C++ is case-sensitive!!

assigning values

- \(=\) is the assignment operator

- example:
  
<table>
<thead>
<tr>
<th>program code</th>
<th>computer's memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>int x; // declaration x = 19; // assignment</td>
<td></td>
</tr>
<tr>
<td>or int x = 19; // declaration and assignment together</td>
<td></td>
</tr>
</tbody>
</table>

- another example:

<table>
<thead>
<tr>
<th>program code</th>
<th>computer's memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>double y; // declaration y = 87.34; // assignment</td>
<td></td>
</tr>
<tr>
<td>or double y = 87.34; // declaration and assignment together</td>
<td></td>
</tr>
</tbody>
</table>

doing math

- \(+\) unary plus
- \(-\) unary minus
- \(+\) addition
- \(-\) subtraction
- \(\times\) multiplication
- \(/\) division
- \(\%\) modulo

  example:

  | int x, y; x = -5; y = x + 7; y = y + 3; x = x + -2; y = x / 19; what are x and y equal to? |

  modulo means “remainder after integer division”

increment and decrement operators

- increment operator: \(++\)
  meaning: add one and assign
  example: \(i++\);
  is the same as: \(i = i + 1;\)

- decrement operator: \(--\)
  meaning: subtract one and assign
  example: \(i--;\)
  is the same as: \(i = i - 1;\)
assignment operators

<table>
<thead>
<tr>
<th>operator</th>
<th>meaning</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>add and assign</td>
<td>i += 3;</td>
</tr>
<tr>
<td>-=</td>
<td>subtract and assign</td>
<td>i -= 3;</td>
</tr>
<tr>
<td>*=</td>
<td>multiply and assign</td>
<td>i *= 3;</td>
</tr>
<tr>
<td>/=</td>
<td>divide and assign</td>
<td>i /= 3;</td>
</tr>
<tr>
<td>%=</td>
<td>modulo and assign</td>
<td>i %= 3;</td>
</tr>
</tbody>
</table>

outputting variables

- you can output the value of a variable using cout
- for example:
  ```cpp
  int i;
i = 7;
cout << "the value of i is " << i << endl;
  ```
- notice the use of the endl function. this is a built-in C++ function that produces a newline. it is the same as "\n", so we could have written the following, which would do the same thing as the last line, above:
  ```cpp
  cout << "the value of i is " << i << "\n";
  ```
- you can output character variables in the same way:
  ```cpp
  char c;
c = 'E';
cout << "the value of c is " << c << endl;
  ```

floating-point divide versus integer divide

- when you divide integers, if the result is not a whole number, then the result is converted to an integer by truncating the decimal part
- if you want to use the decimal part, then divide floating-point values
- consider the following two examples on the next two pages

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

int main() {
    int x = 10;
    int y = 2;
    int w = 3;
    int z;

    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
z = x / y;
    cout << "z = " << z << endl;
    z = x / w;
    cout << "z = " << z << endl;
z = x / 6;
    cout << "z = " << z << endl;
}
```
• second, a floating-point example:

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

int main() {
    double x = 10;
    double y = 2;
    double w = 3;
    double z;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    z = x / y;
    cout << "z = " << z << endl;
    z = x / w;
    cout << "z = " << z << endl;
    z = x / 6;
    cout << "z = " << z << endl;
} // end of main()
```

The integer output is:

- `x = 10`
- `y = 2`
- `z = 5`
- `z = 3`
- `z = 1`

And the floating-point output is:

- `x = 10`
- `y = 2`
- `z = 3.33333`
- `z = 1.66667`

• here’s another example:

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

int main() {
    int x = 10;
    int w = 3;
    double z;
    z = x / w;
    cout << "z = " << z << endl;
    z = x / (double)w;
    cout << "z = " << z << endl;
    z = (double)x / w;
    cout << "z = " << z << endl;
    z = x / 6.0;
    cout << "z = " << z << endl;
} // end of main()
```

And the output is:

- `z = 3`
- `z = 3.333333`
- `z = 3.333333`
- `z = 1.66667`

The `(double)` operator coerces (i.e., converts) the integer values into floating-point, so that the result of the math is computed in floating-point and the precision is retained. The last operation shows the use of a floating-point constant (6.0) which has the same affect.