Basic C++
(What you should already know)
Chapters 1-5
CS 2308/CS5301
Spring 2019
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

<table>
<thead>
<tr>
<th>short</th>
<th>float</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>double</td>
<td>char</td>
</tr>
<tr>
<td>long</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• **Variable Declaration** statement

```cpp```
datatype identifier; float hours;
```

• **Variable Initialization** statement:

```cpp```
datatype identifier = constant; int count = 0;
```

Constants

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

<table>
<thead>
<tr>
<th>1</th>
<th>75</th>
<th>12.45</th>
<th>-3.8</th>
<th>true</th>
<th>‘A’</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>-2</td>
<td>6.25e-5</td>
<td></td>
<td>false</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:
  
  ```
  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 and Relational Operations

- **arithmetic operators:**
  - addition
  - subtraction
  - multiplication
  - division
  - modulo

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x + 10</td>
<td>17</td>
</tr>
<tr>
<td>7 % 2</td>
<td>1</td>
</tr>
<tr>
<td>8 * 5 + 10</td>
<td>50</td>
</tr>
</tbody>
</table>

Watchout: Integer division!!

- **relational operators (result is bool):**
  - Equal to
  - Not equal to
  - Greater than
  - Less than
  - Greater than or equal to
  - Less than or equal to

<table>
<thead>
<tr>
<th>Example</th>
<th>Boolean Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 &lt; 25</td>
<td>False</td>
</tr>
<tr>
<td>89 == x</td>
<td>True</td>
</tr>
<tr>
<td>x % 2 != 0</td>
<td>True</td>
</tr>
<tr>
<td>8 + 5 * 10 &lt;= 100 * n</td>
<td>True</td>
</tr>
</tbody>
</table>

Logical Operations, precedence

- **logical operators (values and results are bool):**
  - not
  - && and
  - || or

<table>
<thead>
<tr>
<th>Example</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &lt; 10 &amp;&amp; x &gt; 0</td>
<td>!(x &lt; 10)</td>
</tr>
<tr>
<td>y == 10</td>
<td></td>
</tr>
<tr>
<td>!(a == b)</td>
<td>!(a == b)</td>
</tr>
</tbody>
</table>

- **operator precedence (which happens first?):**
  - ! (unary)
  - * / %
  - + - (binary)
  - < > <= >=
  - == /=
  - && ||

<table>
<thead>
<tr>
<th>Example</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(y == 10)</td>
<td></td>
</tr>
</tbody>
</table>

More assignment statements

- **Compound assignment**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Usage</th>
<th>Equivalent Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += e;</td>
<td>x = x + e;</td>
</tr>
<tr>
<td>-=</td>
<td>x -= e;</td>
<td>x = x - e;</td>
</tr>
<tr>
<td>*=</td>
<td>x *= e;</td>
<td>x = x * e;</td>
</tr>
<tr>
<td>/=</td>
<td>x /= e;</td>
<td>x = x / e;</td>
</tr>
</tbody>
</table>

- **Increment, decrement**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Usage</th>
<th>Equivalent Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>x++;</td>
<td>x = x + 1;</td>
</tr>
<tr>
<td>--</td>
<td>x--;</td>
<td>x = x - 1;</td>
</tr>
</tbody>
</table>
Type conversions

- **Implicit**
  - **assignment:**
    ```
    int x;
    double d = 3.1415;
    cout << x << endl;
    ```
    the type of expression on the right will be converted to type of variable on left, possibly losing information.
  - **binary operations:**
    ```
    int x = 10;
    double d = 2.3;
    cout << x + d << endl;
    ```
    the operand with the lower ranking type is converted to the type of the other.

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

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;
  ```
  ```
  cin >> x;
  cout << "Enter the height and width: ";
  cin >> height >> width;
  ```
  right hand side must be a variable!

Control structures: if else

- **if and else**
  ```
  if (expression) statement1
  else statement2
  ```
  statement may be a compound statement (a block: {statements})

- **if expression is true, statement1 is executed**
- **if expression is false, statement2 is executed**

- **the else is optional:**
- **nested if else**
  ```
  if (expression)
  statement
  ```
  ```
  if (expression1)
  statement1
  else if (expression2)
  statement2
  else if (expression3)
  statement3
  else
  statement4
  ```

Control structures: loops

- **while**
  ```
  while (expression) statement
  ```
  statement may be a compound statement (a block: {statements})

- **if expression is true, statement is executed, repeat**

- **for:**
  ```
  for (expr1; expr2; expr3)
  statement
  ```
  ```
  expr1;
  while (expr2) {
  statement
  expr3;
  }
  ```
  equivalent to:

- **do while:**
  ```
  do statement
  while (expression);
  ```
  statement is executed, if expression is true, then repeat
Control structures: switch

- switch stmt:
  ```
  switch (expression) {
    case constant: statements
    ...
    case constant: statements
    default: statements
  }
  ```
  - execution starts at the case labeled with the value of the expression.
  - if no match, start at default
  - use break to exit switch (usually at end of statements)

- example:
  ```
  switch (ch) {
    case 'a':
      case 'A': cout << "Option A";
      break;
    case 'b':
      case 'B': cout << "Option B";
      break;
    default: cout << "Invalid choice";
  }
  ```

The string class

- string literals: represent sequences of chars:
  ```
  string firstName, lastName;
  ```
  - to define string variables:
  ```
  string name = "George";
  ```
  - Operations include:
    - = for assignment
    - .size() member function for length
    - ==, <, ..., relational operators (alphabetical order)
    - [n] to access one character

File Input/Output

- #include <fstream>
- Output (ofstream)
  ```
  ofstream fout;
  fout.open("filename.txt");
  fout << "hello"
  fout.close();
  ```
- Input (ifstream)
  ```
  ifstream fin;
  fin.open("data.txt");
  if (!fin) {
    cout << "error opening file" << endl;
    return (0);
  }
  int x;
  fin >> x;
  cout << "x is " << x << endl;
  fin.close();
  ```