Basic C++

Chapters 1-5

CS 2308
Fall 2016

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`, `long double`
  - `bool`

- **Variable Declaration** statement
  ```cpp
datatype identifier;
```

- **Variable Initialization** statement:
  ```cpp
datatype identifier = constant;  // ex: int count = 0;
```

Constants

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12.45</td>
<td>true</td>
<td>'A'</td>
</tr>
<tr>
<td>75</td>
<td>-3.8</td>
<td>false</td>
<td>'2'</td>
</tr>
<tr>
<td>-2</td>
<td>6.25e-5</td>
<td></td>
<td></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.

Logical Operations, precedence

- logical operators (values and results are bool):
  ```
  ! not
  && and
  || or
  ```
  ```
  x < 10 && x > 0
  y == 10 || y == 20
  !(a == b)
  ```

- operator precedence (which happens first?):
  ```
  !  +  *  /  %
  + - (binary)
  < > <= >=
  == !=
  && ||
  ```
  ```
  !(y == 10) || y == 20 && x > 3 * 2
  ```

Arithmetic and Relational Operations

- arithmetic operators:
  ```
  + addition
  - subtraction
  * multiplication
  / division
  % modulo
  ```
  ```
  x + 10
  7 % 2
  8 + 5 * 10
  ```
  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
  ```
  ```
  7 < 25
  89 == x
  x % 2 != 0
  8 + 5 * 10 <= 100 * n
  ```

More assignment statements

- Compound assignment
  ```
  operator usage equivalent syntax:
  += x += e;    x = x + e;
  -= x -= e;    x = x - e;
  *= x *= e;    x = x * e;
  /= x /= e;    x = x / e;
  ```

- increment, decrement
  ```
  operator usage equivalent syntax:
  ++ x++;    ++x;    x = x + 1;
  -- x--;    --x;    x = x - 1;
  ```
Type conversions

- **Implicit**
  - assignment:
    ```
    int x;
    double d = 3.1415;
    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
  ```

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;
  ```

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

- **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:
  ```
  cout << "Hello";
  ```
  - **To define string variables:**
  ```
  string firstName, lastName;
  ```
- **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 << "Count is: " << count << endl;
  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();
  ```