Basic C++
(What you should already know)
Chapters 1-5
CS 2308
Spring 2017
Jill Seaman

Structure of a C++ Program

• Hello world:

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

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

• In general:

```cpp
#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>long</td>
<td>double</td>
</tr>
</tbody>
</table>

• **Variable Declaration** statement

```cpp
datatype identifier;
```

• **Variable Initialization** statement

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

Constants

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

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

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

Logical Operations, precedence

- **logical operators (values and results are bool):**
  ```
  ! not
  && and
  || or
  ```
- **operator precedence (which happens first?):**
  ```
  ! (unary)
  * / %
  + - (binary)
  < > <= >=
  == !=
  &&
  ||
  ```

More assignment statements

- **Compound assignment**
  ```
  x += e;  x = x + e;
  x -= e;  x = x - e;
  x *= e;  x = x * e;
  x /= e;  x = x / e;
  ```
- **increment, decrement**
  ```
  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;
    ```
  - the type of expression on the right will be converted to type of variable on left, possibly losing information.

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

Order of types:

- double
- float
- long
- int
- char

Basic Input/Output

- **Output** (cout and `<<`)
  ```
  cout << expression;
  cout << expr1 << expr2;
  ```

- **Input** (cin and `>>`)
  ```
  cin >> variable;
  cin >> var1 >> var2;
  ```

Control structures: if else

- **if and else**
  ```
  if (expression)
  statement1
  else
  statement2
  ```
  if expression is true, statement1 is executed
  if expression is false, statement2 is executed
  the else is optional:
  nested if else:
  ```
  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})

- **for**
  ```
  for (expr1; expr2; expr3)
  statement
  ```
  for (expr1; expr2; expr3)

- **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:**
  - 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*)

  ```
  switch (expression) {
    case constant: statements ...
    case constant: statements
    default: 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;  // Check for file open errors
  cout << "x is " << x << endl;
  fin.close();
  ```