CS1428 Review

Part I: Chapters 1-5

CS 2308
Spring 2014

Jill Seaman

Structure of a C++ Program

- Hello world:

```
//This program outputs a message to the screen
#include <iostream>
using namespace std;

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

- In general:

```
//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       float         bool
int         double        char
long        long double
```

- **Variable Declaration** statement

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

- **Variable Initialization** statement:

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

Constants

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

```
1
75
-2
```
```
12.45
-3.8
6.25e-5
```
```
true
false
```
```
'A'
'2'
```

- **Named Constants**:
  variable whose value cannot be changed

```
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.

5

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
```

6

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?):

  !
  + - (unary)
  * / %
  + - (binary)
  < > <= >=
  == !=
  &&
  ||

```
!(y == 10) || y == 20 && x > 3 * z
```

7

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;

8

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;
```
right hand side must be a variable!

```
cin >> x;
cout << "Enter the height and width: ";
cin >> height >> width;
```

9

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, statement 1 is executed
  - if expression is false, statement2 is executed

- the else is optional:

```
if (expression)
    statement
```

- nested if else

```
if (expression1)
    statement1
else if (expression2)
    statement2
else if (expression3)
    statement3
else
    statement4
```

10

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

11

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";
}
```

12

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) {                    Check for file open errors
    cout << "error opening file" << endl;
    return (0);
}
int x;
fin >> x;          right hand side must be a variable!
cout << "x is " << x << endl;
fin.close();
```

13

File Input: read to end of file
pp 279-281 in Gaddis, 7th ed.

- `fin>>x` returns true when a value is successfully read, false otherwise.

```
ifstream fin;
fin.open("data.txt");
int x;
while (fin >> x) {
    cout << "next number is " << x << endl;
}
fin.close();
```

NOTE:
DO NOT USE: fin.eof()

- How the while loop works:

  - executes `fin >> x`
  - If a value can be read in, it's assigned to x, and it returns true.
  - If a value cannot be read in (ie nothing else in file), nothing happens except that it returns false.

14

Type conversions

- Implicit

  - assignment:
    ```
    int x;
    double d = 3.1415;
    x = d;
    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.

    Order of types:
    ```
    double
    float
    long
    int
    ```

- Explicit
  ```
  int x, y;
  ...
  float avg = static_cast<float>(x)/y;
  ```

  or
  ```
  float avg = x/(float)y;   //c-style notation
  ```

15