

Week 2

Branching & Looping

Gaddis: Chapters 4 & 5

CS 5301
Spring 2017

Jill Seaman

1

Relational Operators

- relational operators (result is bool):

== Equal to (do not use =)
!= Not equal to
> Greater than
< Less than
>= Greater than or equal to
<= Less than or equal to

```
int x=90;  
int n=6;  
▶ 7 < 25  
▶ 89 == x  
▶ x % 2 != 0  
▶ 8 + 5 * 10 <= 10 * n
```

- operator precedence:

```
*/%  
+-  
<><=>=  
== !=  
=
```

Which operation happens first? next? ...

```
int x, y;  
  
... x < y -10 ...  
... x * 5 >= y + 10 ...  
  
bool t1 = x > 7;  
bool t2 = x * 5 >= y + 10;
```

2

if/else

- if and else

```
if (expression)  
    statement1  
else  
    statement2
```

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

```
double rate, monthlySales;  
.  
.  
if (monthlySales > 3000)  
    rate = .025;  
else  
    rate = .029;
```

- the else is optional:

```
if (expression)  
    statement
```

- if expression is true, statement is executed, otherwise statement is skipped

3

Block or compound statement

- a set of statements inside braces:

```
{  
    int x;  
    cout << "Enter a value for x: " << endl;  
    cin >> x;  
}
```

- This allows us to use multiple statements when by rule only one is allowed.

```
int number;  
cout << "Enter a number" << endl;  
cin >> number;  
if (number % 2 == 0)  
{  
    number = number / 2;  
    cout << "0";  
}  
else  
{  
    number = (number + 1) / 2;  
    cout << "1";  
}
```

4

Nested if/else

- if-else is a statement. It can occur as a statement inside of another if-else statement.

```
if (testScore < 60)
    grade = 'F';
else {
    if (testScore < 70)
        grade = 'D';
    else {
        if (testScore < 80)
            grade = 'C';
        else {
            if (testScore < 90)
                grade = 'B';
            else
                grade = 'A';
        }
    }
}
```

This is equivalent to the code on the left. It is just formatted differently

```
if (testScore < 60)
    grade = 'F';
else if (testScore < 70)
    grade = 'D';
else if (testScore < 80)
    grade = 'C';
else if (testScore < 90)
    grade = 'B';
else
    grade = 'A';
```

5

Logical Operators

- logical operators (values and results are bool):

! not
&& and
|| or

!a is true when a is false
a && b is true when **both** a and b are true
a || b is true when **either** a or b is true

- operator precedence:

```
!
* / %
+ -
< > <= >=
== !=
&&
||
```

- examples T/F?:

```
int x=6;
int y=10;
a. x == 5 && y <= 3
b. x > 0 && x < 10
c. x == 10 || y == 10
d. x == 10 || x == 11
e. !(x > 0)
f. !(x > 6 || y == 10)
```

6

switch statement

- 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";
}
```

7

Input Validation

- Input validation: inspecting input data to determine whether it is acceptable
- Invalid input is an error that should be treated as an exceptional case.
 - The program can ask the user to re-enter the data
 - The program can exit with an error message

```
cout << "Enter a score between 0 and 100: ";
cin >> score;
if (score < 0 || score > 100) {
    cout << "That is an invalid score." << endl;
} else {
    //do something with score here
}
```

8

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;

9

while loops

- while

```
while (expression)
    statement
```

statement may be a compound statement (a block: {statements})

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

- Example:

```
int number;
cout << "Enter a number, 0 when finished: ";
cin << number;
while (number != 0)
{
    cout << "You entered " << number << endl;
    cout << "Enter the next number: ";
    cin >> number;
}
cout << "Done" << endl;
```

- output:

```
Enter a number, 0 when finished: 22
You entered 22
Enter the next number: 5
You entered 5
Enter the next number: 0
Done
```

10

two kinds of loops

- conditional loop

- * execute as long as a certain condition is true

- count-controlled loop:

- * executes a specific number of times

- initialize counter to zero (or other start value).
- test counter to make sure it is less than count.
- update counter during each iteration.

```
int number = 1;
while (number <= 3)
{
    cout << "Student" << number << endl;
    number = number + 1; // or use number++
}
cout << "Done" << endl;
```

number is a "counter", it keeps track of the number of times the loop has executed.

11

for loops

- for:

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

statement may be a compound statement (a block: {statements})

- * equivalent to:

```
expr1;
while (expr2) {
    statement
    expr3;
}
```

- Good for implementing count-controlled loops:

pattern: for (initialize; test; update)

```
for (int number = 1; number <= 3; number++)
{
    cout << "Student" << number << endl;
}
cout << "Done" << endl;
```

12

do-while loops

- do while:

```
do  
    statement  
while (expression);
```

statement may be a
compound statement
(a block: {statements})

statement is executed.
if expression is true, then repeat

- The test is at the end, statement ALWAYS executes at least once.

```
int number;  
do {  
    cout << "Enter a number, 0 when finished: ";  
    cin << number;  
    cout << "You entered " << number << endl;  
} while (number != 0);
```

13

Keeping a running total (summing)

- Example:

```
int days;  
float total = 0.0; //Accumulator  
  
cout << "How many days did you ride your bike? ";  
cin >> days;  
  
for (int i = 1; i <= days; i++)  
{  
    float miles;  
    cout << "Enter the miles for day " << i << ": ";  
    cin >> miles;  
    total = total + miles;  
}  
  
cout << "Total miles ridden: " << total << endl;
```

14

Sentinel controlled loop

- Use a special value to signify end of the data:

```
float total = 0.0; //Accumulator  
float miles;  
  
cout << "Enter the miles you rode each day, ";  
cout << "one number per line.\n";  
cout << "Then enter -1 when finished.\n";  
  
cin >> miles;  
  
while (miles != -1)  
{  
    total = total + miles;  
    cin >> miles;  
}  
  
cout << "Total miles ridden: " << total << endl;
```

- Sentinel value must NOT be a valid value

15

Nested loops

- When one loop appears in the body of another
- For every iteration of the outer loop, we do all the iterations of the inner loop

```
for (row=1; row<=3; row++) //outer  
{  
    for (col=1; col<=3; col++) //inner  
        cout << row * col << " ";  
    cout << endl;
```

Output:

1	2	3
2	4	6
3	6	9

16

continue and break Statements

- Use `break` to terminate execution of a loop
- When used in a nested loop, terminates the inner loop only.

- Use `continue` to go to end of **current** loop and prepare for next repetition
- `while`, `do-while` loops: go immediately to the test, repeat loop if test passes
- `for` loop: immediately perform update step, then test, then repeat loop if test passes

17

Sample Problem

- A software company sells a package that retails for \$99. Quantity discounts are given according to the following table.

Quantity	Discount
10-19	20%
20-49	30%
50-99	40%
100 or more	50%

- Write a program that asks for the number of units sold and computes the total cost of the purchase.
- Input Validation: Make sure the number of units is greater than 0. Otherwise output an error message.¹⁸