Week 2
Branching & Looping
Gaddis: Chapters 4 & 5
CS 5301
Fall 2018
Jill Seaman

Week 2
Branching & Looping
Gaddis: Chapters 4 & 5
CS 5301
Fall 2018
Jill Seaman

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;
  int 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;

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

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";
  }
Nested if/else

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

```java
if (score >= 90)
    grade = 'A';
else if (score >= 80)
    grade = 'B';
else if (score >= 70)
    grade = 'C';
else if (score >= 60)
    grade = 'D';
else
    grade = 'F';
```

- The braces are optional on this side

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

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

Nested if/else

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

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

Logical Operators

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

- examples T/F?:
  ```java
  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)
  ```

Operator precedence:

- `!` / `%` / `+` / `-` / `<=` / `==` / `=!` / `&&` / `||`
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:
```c
switch (ch) {
  case 'a':
  case 'A': cout << "Option A";
    break;
  case 'b':
  case 'B': cout << "Option B";
    break;
  default: cout << "Invalid choice";
}
```

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

```c
// Example of input validation
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
}
```

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++; ++x;</td>
<td>x = x + 1;</td>
</tr>
<tr>
<td>--</td>
<td>x--; --x;</td>
<td>x = x - 1;</td>
</tr>
</tbody>
</table>

while loops

- while

```c
while (expression)
  statement
```  

- 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
```
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.

for loops

- for:
  - equivalent to:
    - Good for implementing count-controlled loops:
      - initialize; test; update

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

pattern: for (initialize; test; update)

do-while loops

- do while:
  - statement may be a compound statement (a block: {statements})
  - 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);
```

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;
```
Sentinel controlled loop

- A sentinel controlled loop continues to process data until reaching a special value (called the sentinel) that signals the end.

  ```
  get the first data item
  while item is not the sentinel
    process the item
  get the next data item
  ```

- The first item is retrieved before the loop starts. This is called the priming read, since it gets the process started.

- If the first item is the sentinel, the loop terminates and no data is processed.

Example: summing using a sentinel

```cpp
float total = 0.0; //Accumulator
float miles;
cout << "Enter the miles you rode (-1 to quit): ";
cin >> miles;
while (miles != -1)
{
    total = total + miles;
    cout << "Enter the miles you rode (-1 to quit): ";
cin >> miles;
}
cout << "Total miles ridden: " << total << endl;
```

Nested loops

- When one loop appears in the body of another

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

  Output:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

continue and break Statements

- Use break to terminate execution of a loop

  - When used in a nested loop, terminates the inner loop only.

  ```cpp
  ```

- Use continue to go to end of current loop and prepare for next repetition

  ```cpp
  ```

- while, do-while loops: go immediately to the test, repeat loop if test passes

  ```cpp
  ```

- for loop: immediately perform update step, then test, then repeat loop if test passes

  ```cpp
  ```
Example problem: Future Value

- Money deposited in a bank account earns interest annually. How much will the account be worth 10 years from now?
- Inputs: the principal and the annual interest rate
- Output: value of the investment in 10 years
- Relationship between Inputs and Outputs:

  Value after one year is given by this formula:
  \[ \text{principal} \times (1 + \text{apr}) \]
  This needs to be done 10 times.

Example problem: Future Value

- Design:

<table>
<thead>
<tr>
<th>Print an introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input the amount of the principal (\text{principal})</td>
</tr>
<tr>
<td>Input the annual percentage rate (\text{apr})</td>
</tr>
<tr>
<td>Repeat 10 times:</td>
</tr>
<tr>
<td>\text{principal} = \text{principal} \times (1 + \text{apr})</td>
</tr>
<tr>
<td>Output the value of principal</td>
</tr>
</tbody>
</table>

Example problem: Future Value

- Code:

```c
int main() {
    cout << fixed << setprecision(2);
    double principal, apr;

    // Print an introduction
    cout << "This program calculates the future ";
    cout << "value of a 10-year investment." << endl;

    // Input the amount of the principal and interest
    cout << "Enter the initial principal: ";
    cin >> principal;
    cout << "Enter the annual interest rate: ";
    cin >> apr;

    // Repeat 10 times:
    for (int i=1; i<=10; i++)
        principal = principal * (1 + apr);

    // Output the value of principal
    cout << "The value in 10 years is: " << principal << endl;
}
```