ENGINEERING PROGRAMMING I

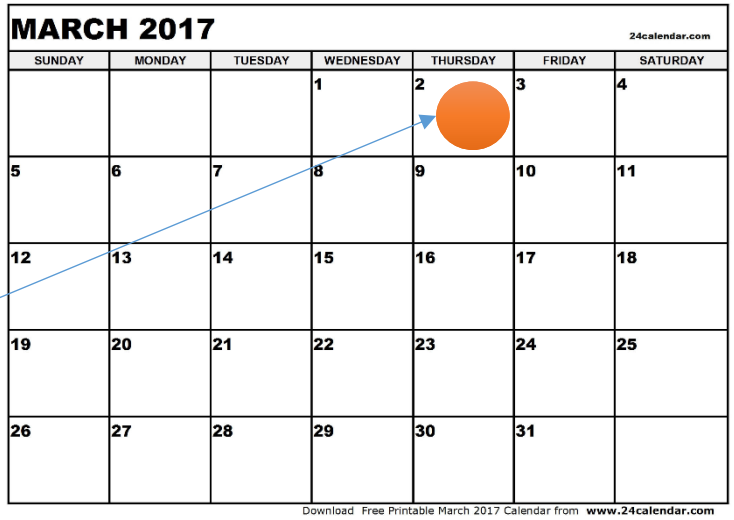
Chapter 04

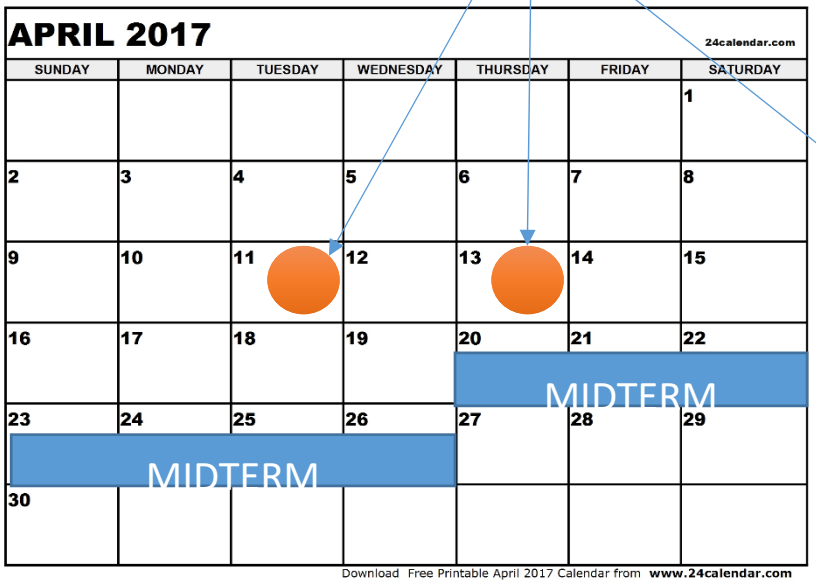
2017

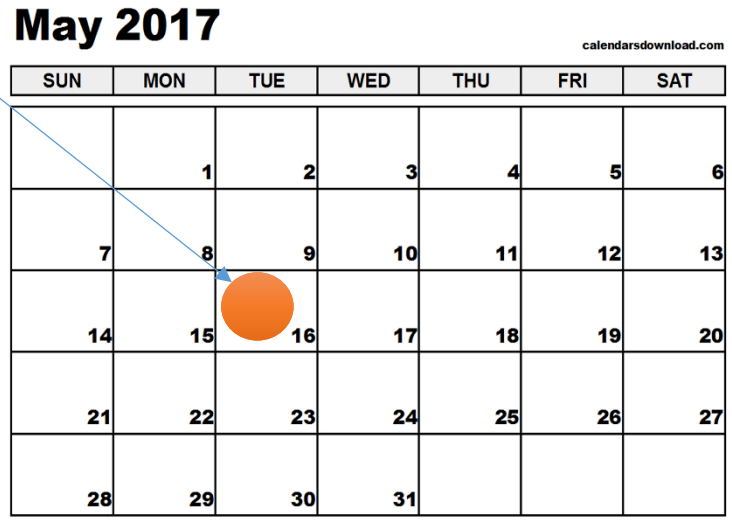
Administrivia

- Classes cancelled
  - Mar 2 (Thu)
  - April 11 (Tue), April 13 (Thu)
  - May 16 (Tue)
  - Total 200 minutes.

No class

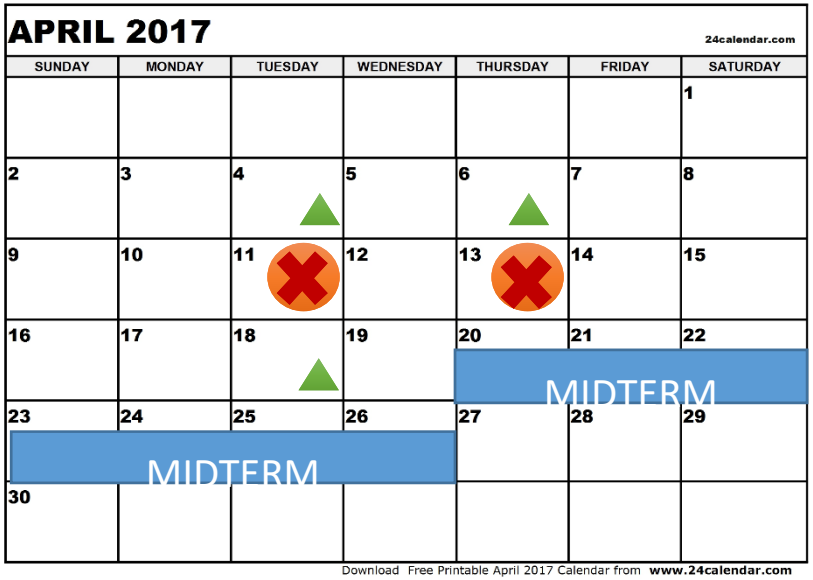


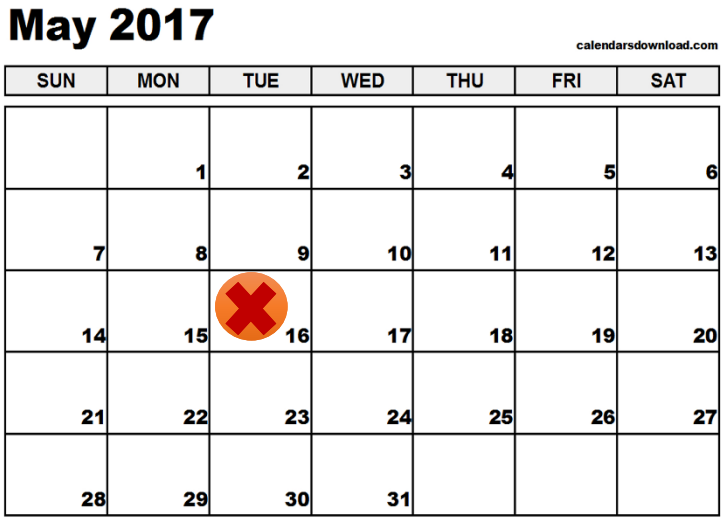




Administrivia

- Classes cancelled
  - Mar 2 (Thu)
  - April 11 (Tue), April 13 (Thu)
  - May 16 (Tue)
  - 200 mins/25mins = 8 times.
  - 75 mins class (1pm-2:15pm) until Apr 6.





Chapter 4: Making Decisions

Starting Out with C++
Early Objects
Seventh Edition

by Tony Gaddis, Judy Walters,
and Godfrey Muganda

Addison-Wesley
is an imprint of
PEARSON

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

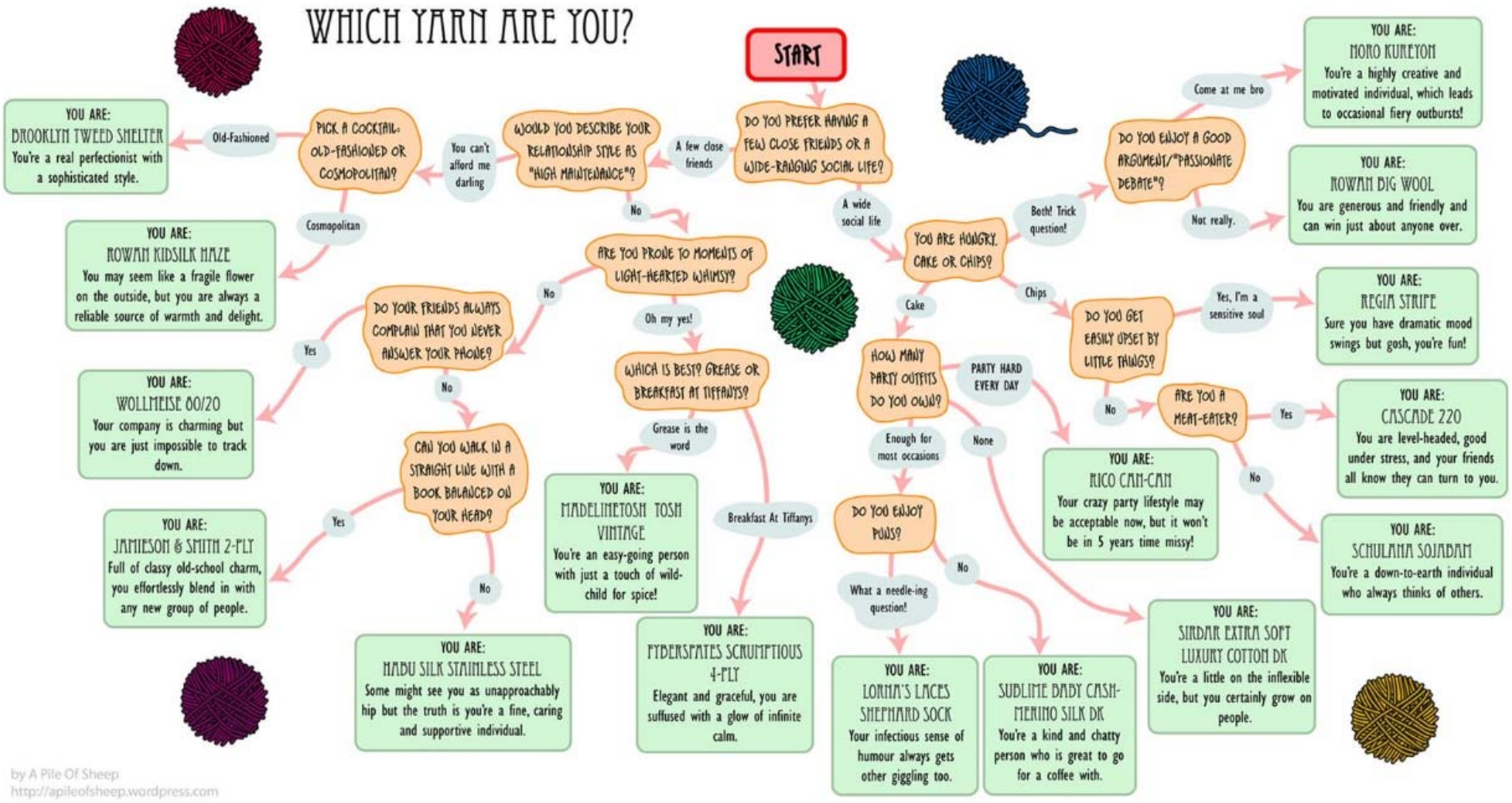
Topics

- 4.1 Relational Operators
- 4.2 The if Statement
- 4.3 The if/else Statement
- 4.4 The if/else if Statement
- 4.5 Menu-Driven Programs
- 4.6 Nested if Statements
- 4.7 Logical Operators

Topics (continued)

- 4.8 Validating User Input
- 4.9 More about Variable Definitions and Scope
- 4.10 Comparing Characters and Strings
- 4.11 The Conditional Operator
- 4.12 The switch Statement
- 4.13 Enumerated Data Types
- 4.14 Testing for File Open Errors

Personality Test: Flow Chart



by A Pile Of Sheep
http://apileofsheep.wordpress.com

Program#2: Tip Calculator

```
Welcome to the Tip Calculator.
How much was your bill?
> 20
How many people are in your party?
> 2
You each need to pay $12.00.
```

You go out to dinner with a group of friends.
Write a simple program to calculate the tip and
Divide the total equally among friends.
If your bill is more than $50, you should pay a 15% tip.
Otherwise, 20% tip.

Need Conditional Branches

Relational Operators

- Used to compare numbers to determine relative order
- Operators:

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Relational Expressions

- Relational expressions are Boolean
  - i.e., evaluate to true or false

- Examples:

  `12 > 5` is **`true`**
  `7 <= 5` is **`false`**

  if **X** is 10, then

  `x == 10` is **`true`**,
  `x != 8` is **`true`**, and
  `x == 8` is **`false`**

Relational Expressions

- Can be assigned to a variable

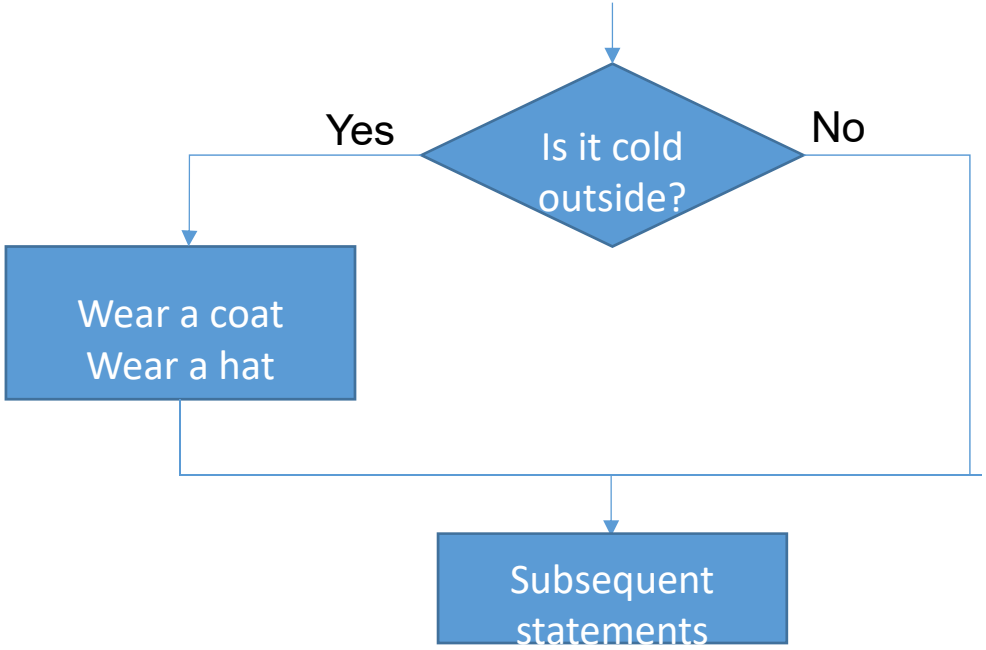
```
bool result = (x <= y);
```

- Assigns 0 for false, 1 for true
- Do not confuse = (assignment) and == (equal to)

The `if` Statement

- Allows statements to be conditionally executed or skipped over

- Models the way we mentally evaluate situations
  - "If it is cold outside,
  - wear a coat and wear a hat."



Format of the if Statement

```
if (condition)
{
   statement1;
   statement2;
       …
   statementn;
}
```

•The block inside the braces is called the body of the if statement. If there is only 1 statement in the body, the { } may be omitted.

How the `if` Statement Works

- If (condition) is true, then the statement(s) in the body are executed.

- If (condition) is false, then the statement(s) are skipped.

Example if Statements

```
if (score >= 60)
   cout << "You passed.\n";

if (score >= 90)
{
   grade = 'A';
   cout << "Wonderful job!\n";
}
```

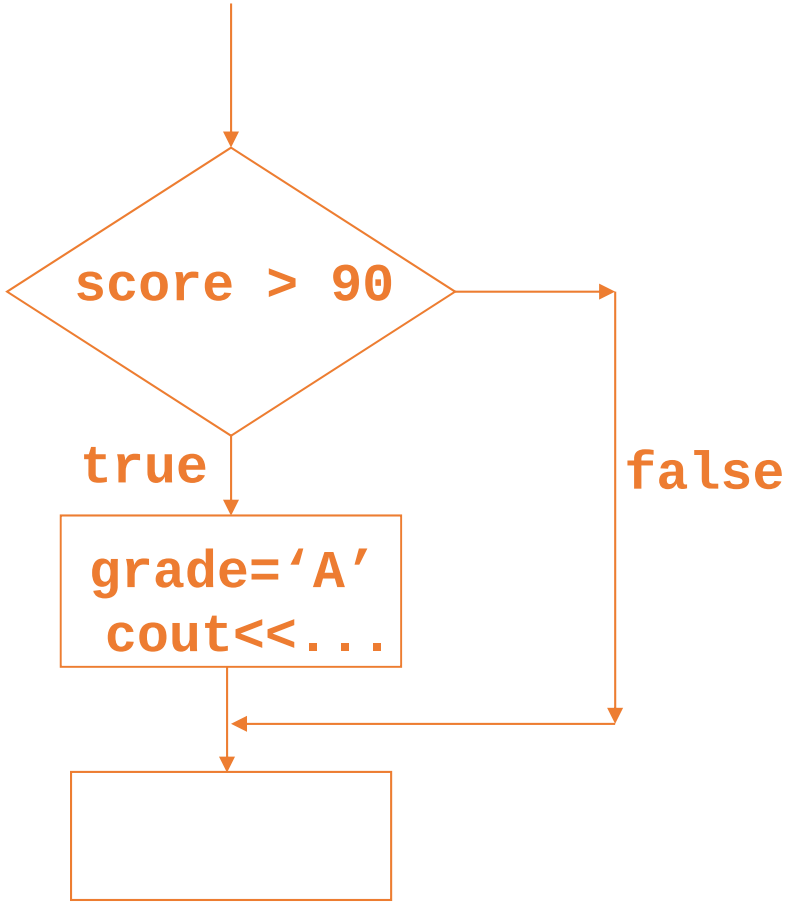
`if` Statement Notes

- Do not place ; after (condition)
- Don't forget the { } around a multi-statement body
- Place each statement; on a separate line after (condition), indented
- 0 is false; any other value is true

Example if Statements

```
if (score >= 90)
{
   grade = 'A';
   cout << "Wonderful job!\n";
}
```

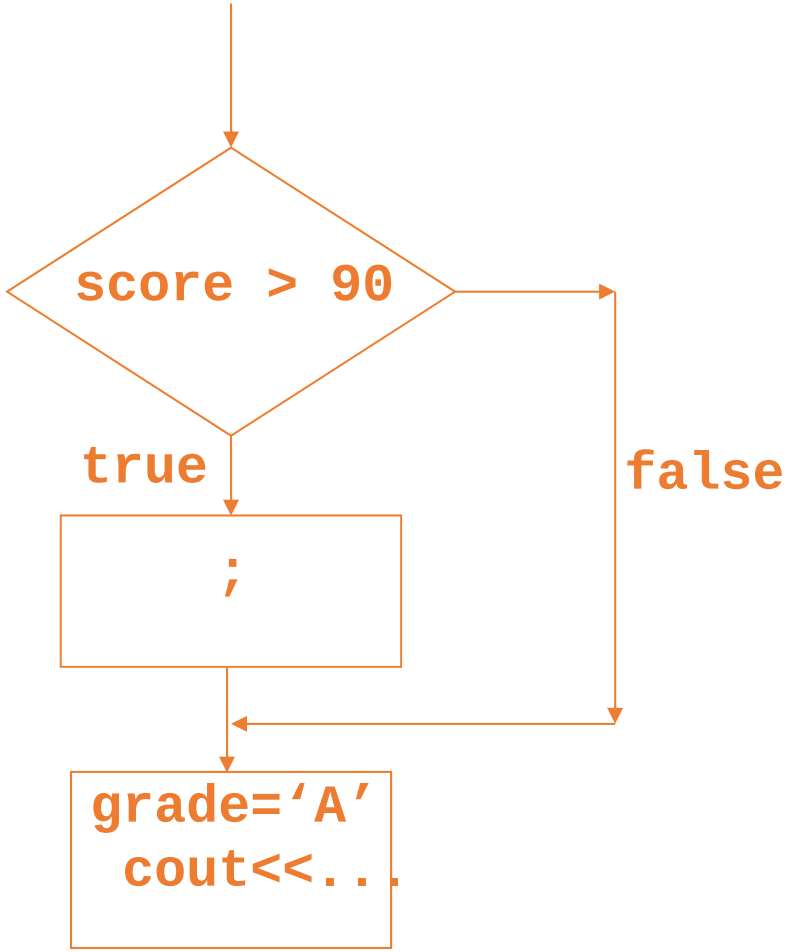
if Statement Flow of Control



Example if Statements

```
if (score >= 90);
{
   grade = 'A';
   cout << "Wonderful job!\n";
}
```

if Statement Flow of Control



What is `true` and `false`?

- An expression whose value is 0 is considered false.
- An expression whose value is non-zero is considered true.
- An expression need not be a comparison – it can be a single variable or a mathematical expression.

Flag

- A variable that signals a condition
- Usually implemented as a bool
- Meaning:
  - `true`: the condition exists
  - `false`: the condition does not exist
- The flag value can be both set and tested with if statements

Flag Example

Example:

```
bool validMonths = true;
        …
if (months < 0)
    validMonths = false;
        …
if (validMonths)
    moPayment = total / months;
```

Comparisons with floating-point numbers

- It is difficult to test for equality when working with floating point numbers.

- It is better to use
  - greater than, less than tests, or
  - test to see if value is very close to a given value

The `if/else` Statement

- Allows a choice between statements depending on whether (condition) is true or false
- Format:

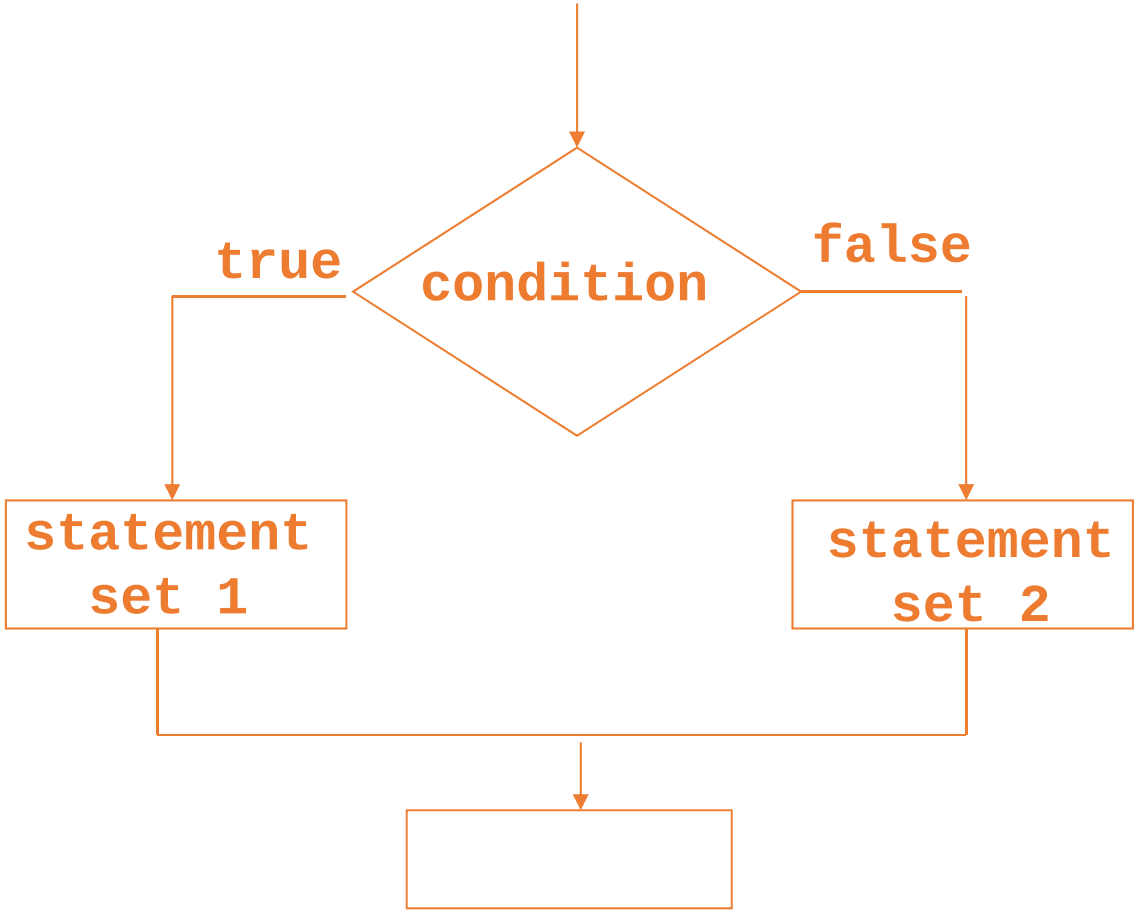
```
if (condition)
{
     statement set 1;
}
else
{
     statement set 2;
}
```

How the `if/else` Works

- If (condition) is `true`, statement set 1 is executed and statement set 2 is skipped.

- If (condition) is `false`, statement set 1 is skipped and statement set 2 is executed.

if/else Flow of Control



Example `if/else` Statements

```
if (score >= 60)
  cout << "You passed.\n";
else
  cout << "You did not pass.\n";

if (intRate > 0)
{  interest = loanAmt * intRate;
   cout << interest;
}
else
    cout << "You owe no interest.\n";
```

The `if/else if` Statement

- Chain of if statements that test in order until one is found to be true
- Also models thought processes
  - "If it is raining, take an umbrella,
  - else, if it is windy, take a hat,
  - else, if it is sunny, take sunglasses."

if/else if Format

```
if (condition 1)
{   statement set 1;
}
else if (condition 2)
{   statement set 2;
}
        …
else if (condition n)
{   statement set n;
}
```

Using a Trailing `else`

- Used with if/else if statement when all of the conditions are false
- Provides a default statement or action
- Can be used to catch invalid values or handle other exceptional situations

Example `if/else if` with Trailing else

```
if (age >= 21)
   cout << "Adult";
else if (age >= 13)
   cout << "Teen";
else if (age >= 2)
   cout << "Child";
else
   cout << "Baby";
```

Nested `if` Statements

- An **`if`** statement that is part of the **`if`** or **`else`** part of another **`if`** statement
- Can be used to evaluate > 1 data item or condition

```
if (score < 100)
{
   if (score > 90)
      grade = 'A';
}
```

Notes on Coding Nested ifs

- An else matches the nearest if that does not have an else

```
if (score < 100)
   if (score > 90)
       grade = 'A';
   else ...  // goes with second if,
             // not first one
```

- Proper indentation aids comprehension

Logical Operators

- Used to create relational expressions from other relational expressions

Operators	Meaning	Explanation
&&	AND	New relational expression is true if both expressions are true
||	OR	New relational expression is true if either expression is true
!	NOT	Reverses the value of an expression; true expression becomes false, false expression becomes true

Logical Operator Examples

- `int x = 12, y = 5, z = -4;`

`(x > y) && (y > z)`	`true`
`(x > y) && (z > y)`	`false`
`(x <= z) || (y == z)`	`false`
`(x <= z) || (y != z)`	`true`
`!(x >= z)`	`false`

Logical Precedence

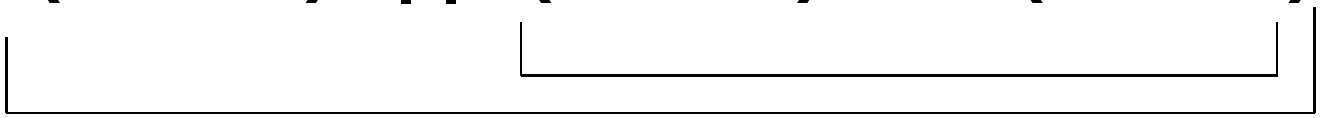
Highest `!`

`&&`

Lowest `||`

Example:

```
(2 < 3) || (5 > 6) && (8 > 7)
```

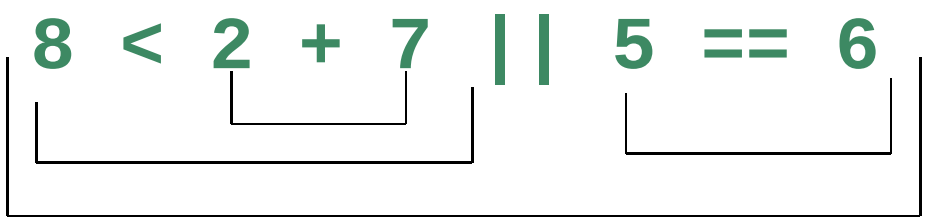


is true because AND is evaluated before OR

More on Precedence

Highest	arithmetic operators
↓	relational operators
Lowest	logical operators

Example:



is true

Checking Numeric Ranges with Logical Operators

- Used to test if a value is within a range

```
if (grade >= 0 && grade <= 100)
   cout << "Valid grade";
```

- Can also test if a value lies outside a range

```
if (grade <= 0 || grade >= 100)
   cout << "Invalid grade";
```

- Cannot use mathematical notation

```
if (0 <= grade <= 100) //Doesn't
                       //work!
```

Clarification on previous class contents: Precision error example

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

int main()
{
  float a = 1.05;
  float b = 100.0;
  float c = a*b;
  cout << c << endl;
  int d = c;
  cout << d << endl;
  return 0;
}
```

Clarification on previous class contents: a block of statements

- A **block** of statements, also called a **compound statement**, is a group of statements that is treated by the compiler as if it were a single statement.
- Blocks begin with a { symbol, end with a } symbol, and the statements to be executed are placed in between.
- Blocks can be used any place where a single statement is allowed.

```
if(a==1)
{// beginning of a compound statement
  cout << "a==1" <<endl;
  a = 2*a;
} // end of a compound statement
```

Clarification on previous class contents: a block of statements

```
if(a==1)
    a_statement_should_come_here;
else
    another_statement_should_come_here;
```

Since a block of statements is equally treated as a single statement…

```
if(a==1){
    multiple_statements_can_come_here;
    multiple_statements_can_come_here;
}
else {
    multiple_statements_can_come_here;
    multiple_statements_can_come_here;
}
```

Clarification on previous class contents: a block of statements

```
if(a==1){
    multiple_statements_can_come_here;
    multiple_statements_can_come_here;
}
else {
    multiple_statements_can_come_here;
    multiple_statements_can_come_here;
}
```

This if-else block is treated as a single statement.

Clarification on previous class contents: nested if-else statements

```
if (today_is_Saturday_or_Sunday)
        if (it_is_raining)
            Sleep_late = true;
        else
            Get_up_and_go_outside = true;

else
        Go_to_work = true;
```

We do not need { } for the inner if-else statement, because it is a single compound-statement.

Clarification on previous class contents: a block of statements

```
if(a==1){
    if(b>0) {
        c = 1;
    }
    else
        c = 2;
}
else
    if(b>0)
        c = 3;
    else
        c = 4;
```

Nested if-else blocks.

Validating User Input

- Input validation: inspecting input data to determine if it is acceptable
- Want to avoid accepting bad input
- Can perform various tests
  - Range
  - Reasonableness
  - Valid menu choice
  - Zero as a divisor

More About Variable Definitions and Scope

- Scope of a variable is the block in which it is defined, from the point of definition to the end of the block
- Variables are usually defined at beginning of function
- They may instead be defined close to first use

More About Variable Definitions and Scope

- Variables defined inside { } have local or block scope

- When in a block that is nested inside another block, you can define variables with the same name as in the outer block.

  - When the program is executing in the inner block, the outer definition is not available
  - This is generally not a good idea

Comparing Characters and Strings

- Can use relational operators with characters and string objects

```
if (menuChoice == 'A')
if (firstName == "Beth")
```

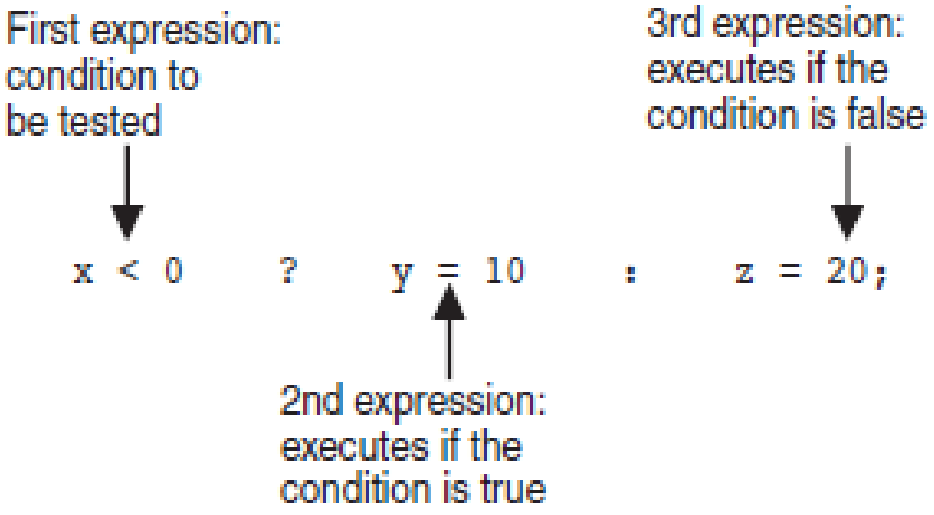
- Comparing characters is really comparing ASCII values of characters
- Comparing string objects is comparing the ASCII values of the characters in the strings. Comparison is character-by-character
- Cannot compare C-style strings with relational operators

Example

```
string str = “Hello”;
if (str == “Hello”)
   cout << “match 1” << endl;
if (“Hello” == str)
   cout << “match 2” << endl;
char cstr[] = “Hello”;
if (cstr == “Hello”)
   cout << “match 3” << endl;
if (“Hello” == cstr)
   cout << “match 4” << endl;
```

The Conditional Operator

- Can use to create short **`if/else`** statements
- Format: **`expr ? expr : expr;`**



The `switch` Statement

- Used to select among statements from several alternatives
- May sometimes be used instead of if/else if statements

switch Statement Format

```
switch (IntExpression)
{
 case exp1: statement set 1;
 case exp2: statement set 2;
 ...
 case expn: statement set n;
 default:   statement set n+1;
}
```

switch Statement Requirements

- ***IntExpression*** must be a char or an integer variable or an expression that evaluates to an integer value
- ***exp1*** through ***expn*** must be constant integer type expressions and must be unique in the switch statement
- default is optional but recommended

How the switch Statement Works

- ***IntExpression*** is evaluated

- The value of ***IntExpression*** is compared against ***exp1*** through ***expn***.

- If ***IntExpression*** matches value ***expi***, the program branches to the statement(s) following ***expi*** and continues to the end of the switch

- If no matching value is found, the program branches to the statement after default:

The `break` Statement

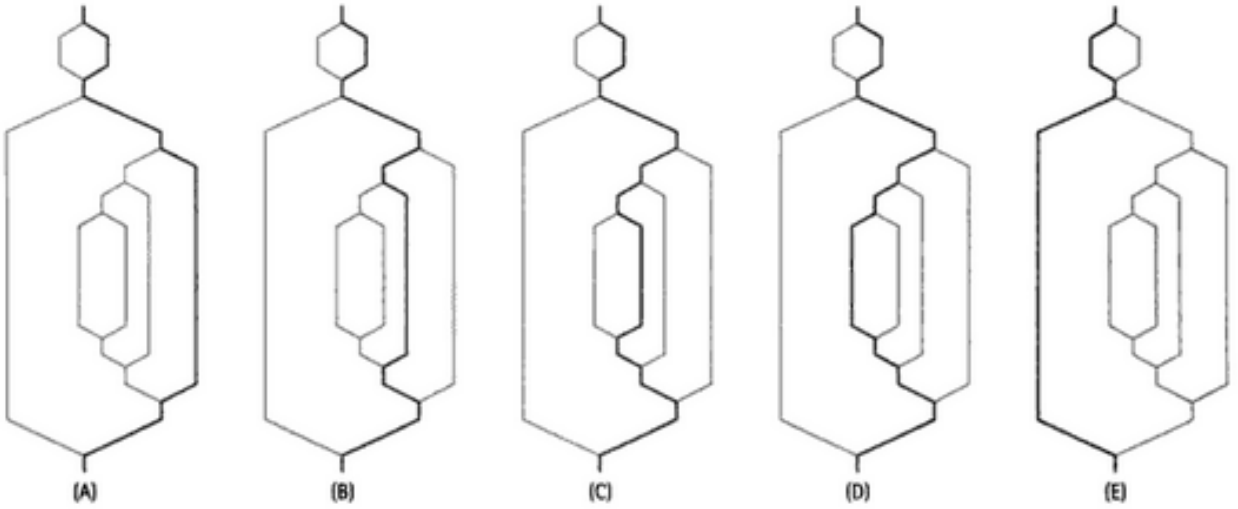
- Used to stop execution in the current block
- Also used to exit a `switch` statement
- Useful to execute a single `case` statement without executing statements following it

Example switch Statement

```
switch (gender)
{
  case 'f': cout << "female";
            break;
  case 'm': cout << "male";
            break;
  default : cout << "invalid gender";
}
```

Testing and Debugging Branches

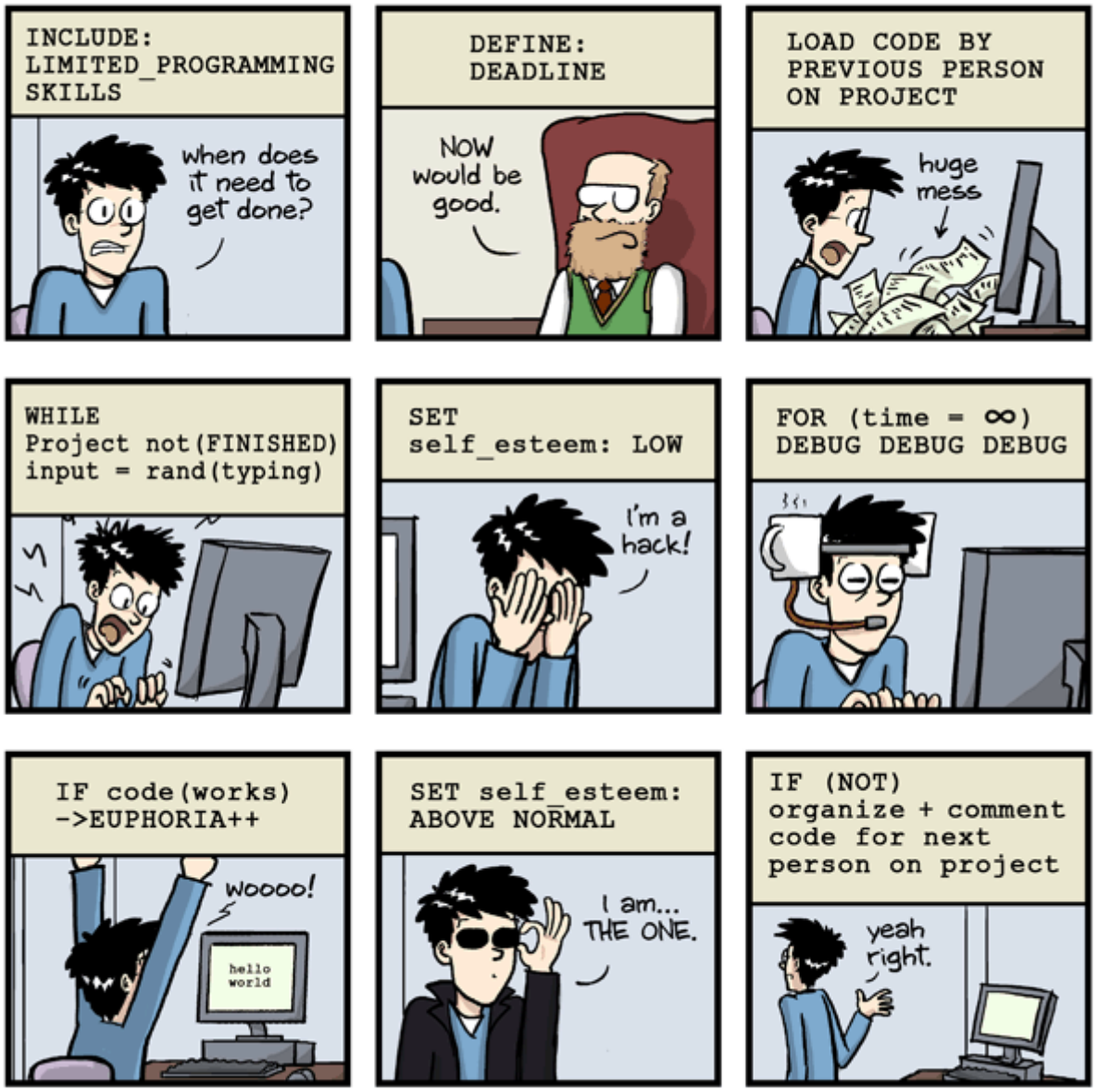
- When your program has conditional branches, you should test your code for all possible control flows.
  - Testing all possible values is impossible. (Data coverage)
  - Examining the code and looking for ranges of values for which processing is identical. (Code coverage)



Write a C++ program

- Write a C++ program that takes a number in the range of 0 to 6 and a second number in the range of 1 to 366 as input.
- The first number represents the day of the week on which the year begins, where 0 is Sunday, and so on. The second number indicates the day of the year. The program then outputs the name of the day of the week corresponding to the day of the year.
- The program should output an error message if the numbers entered are not in the required ranges. The prompt and the error message should make it clear to the user how the numbers must be entered. Be sure to use proper formatting and appropriate comments in your code. The output should be labeled clearly and formatted neatly.

PROGRAMMING FOR NON-PROGRAMMERS



JORGE CHAM © 2014

WWW.PHDCOMICS.COM