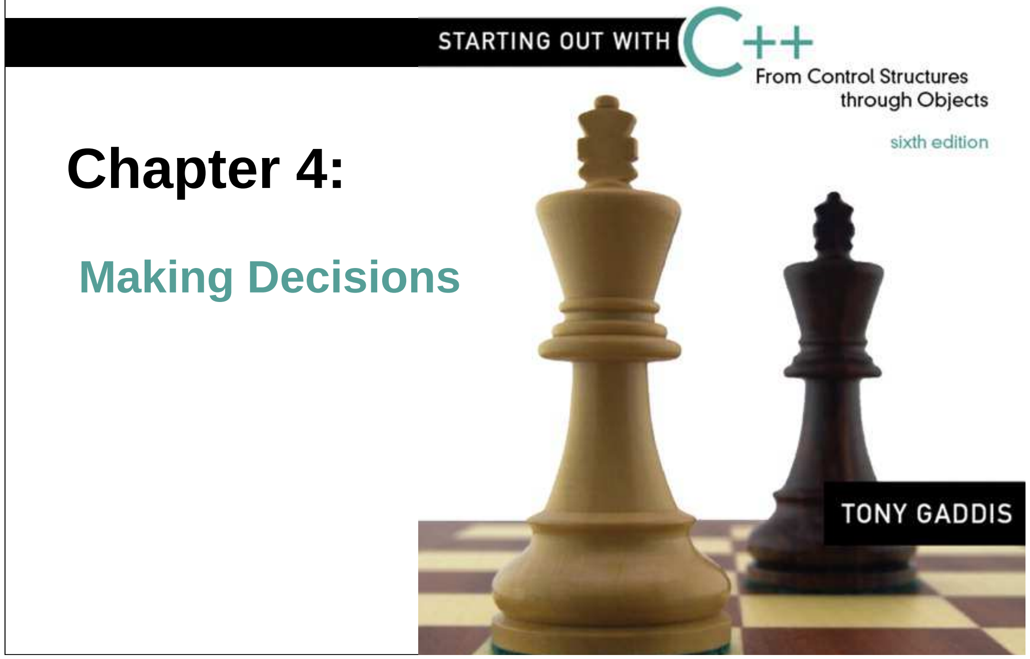
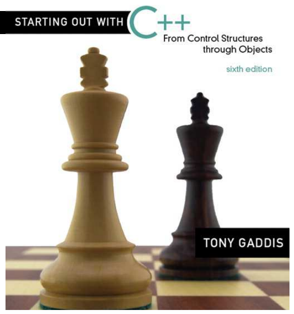


Chapter 4:

Making Decisions



4.1

Relational Operators

Simple Program Scheme

- So far our programs follow a simple scheme
  - Gather input from the user
  - Perform one or more calculations
  - Display the results on the screen

```
int numPtr;
double totPrice, unitPrice=135.29;

cout << "Enter # of printers purchased: ";
cin >> numPtr;
totPrice = numPtr*unitPrice;
cout << "Total price = " << totPrice << endl;
```

Simple Program Scheme

- Simple program scheme follows a predefined path – one sequence of actions
- Most programs can follow different paths by comparing values and making decisions

  If the # of printer (numPtr) < 5
      totPrice = numPtr * 135.29 (regular price)
  If numPtr ≥ 5
      totPrice = numPtr * 125.29 (discounted price)

  - Need to use **relational operators** (<, ≥, …)
  - Need to use **`if`** statement

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

- Used to test conditions (`true` or `false`)
  - Format: `exp1` rop `exp2`
  - Value: `true / false`
- Examples:

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

if `x` is 10, then

`x == 10`	is `true`,
`x+1 != 8`	is `true`, and
`x/2 == 8`	is `false`

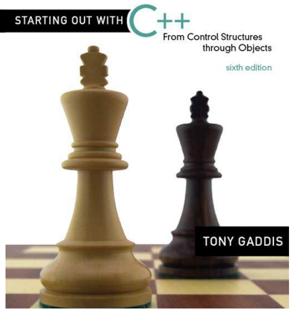
Relational Expressions

- Can be assigned to a variable or displayed on the screen:

```
result = x <= y;
```

- Relational expressions have higher precedence than the assignment operator
- Assigns `0` for `false`, `1` for `true`
- Do not confuse `=` with `==`
- It helps to use parentheses

```
cout << (x <= y);
value = (x == y);
```



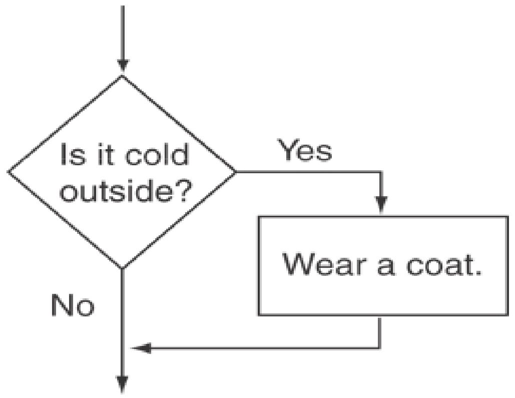
4.2

The `if` Statement

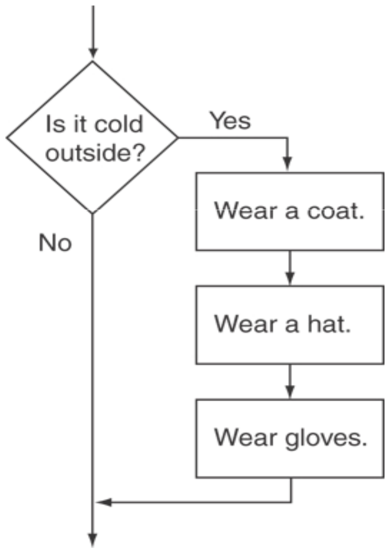
The `if` Statement

- Allow programs to make decisions
- Allows statements to be conditionally executed or skipped over
- Models the way we mentally evaluate situations:
  - "If it is raining, take an umbrella."
  - "If it is cold outside, wear a coat."

Flowchart for Evaluating a Decision



Flowchart for Evaluating a Decision



The `if` Statement

- General Format:

```
if (boolean expression)
     statement;
```

- Example:

```
if ( age >= 18 )
   cout << "You can vote" << endl;
```

`if` statement – what happens

To evaluate:

```
if (boolean expression)
    statement;
```

- If the *`boolean expression`* is `true`, then *`statement`* is executed.
- If the *`boolean expression`* is `false`, then *`statement`* is skipped.

Program 4-2

```
 1  // This program averages three test scores
 2  #include <iostream>
 3  #include <iomanip>
 4  using namespace std;
 5
 6  int main()
 7  {
 8     int score1, score2, score3;   // To hold three test scores
 9     double average;               // To hold the average score
10
```

(Program Continues)

Program 4-2 *(continued)*

```
11     // Get the three test scores.
12     cout << "Enter 3 test scores and I will average them: ";
13     cin >> score1 >> score2 >> score3;
14
15     // Calculate and display the average score.
16     average = (score1 + score2 + score3) / 3.0;
17     cout << fixed << showpoint << setprecision(1);
18     cout << "Your average is " << average << endl;
19
20     // If the average is greater than 95, congratulate the user.
21     if (average > 95)
22        cout << "Congratulations! That's a high score!\n";
23     return 0;
24  }
```

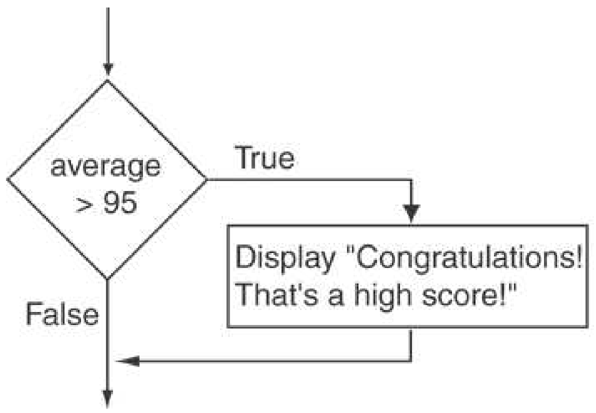
Program Output with Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 80 90 70 [Enter]
Your average is 80.0
```

Program Output with Other Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 100 100 100 [Enter]
Your average is 100.0
Congratulations! That's a high score!
```

Flowchart for Lines 21 and 22



`if` statement notes

- Do not place `;` after `(`*`boolean expression`*`)`
- Place *`statement`*`;` on a separate line after `(`*`boolean expression`*`)`, indented:

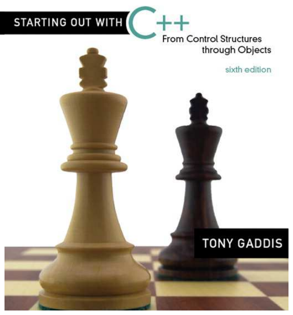
```
    if (score > 90)
        grade = 'A';
```

- Be careful about testing `float`s and `double`s for equality (not recommended)
- Don't confuse == with =

```
    if ( average = 100 )        // wrong
        cout << "Congratulations!";
```

- `0` is `false`; any other value is `true`

```
// This program calculates the total price
// of the printers purchased.
#include <iostream>
using namespace std;
void main()
{
    int numPtr;
    double totPrice, uPrice1=135.29, uPrice2=125.29;

    cout << "Enter # of printers purchased: ";
    cin >> numPtr;

    if ( numPtr < 5 )
        totPrice = numPtr*uPrice1;
    if ( numPtr >= 5 )
        totPrice = numPtr*uPrice2;

    cout << "Total price = " << totPrice << endl;
}
```



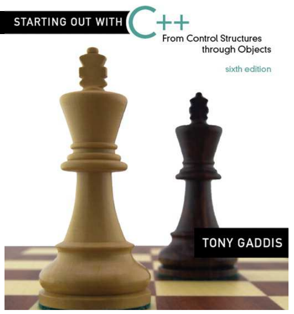
4.3

Flags

Flags

- A variable that signals a condition (vs. expression)
- Usually implemented as a `bool` variable
- As with other variables in functions, must be assigned an initial value before it is used

```
bool highScore = false;
…
if ( average > 95 )
 highScore = true;
…
if ( highScore )
 cout << "Congratulation! That is a high score!";
```



4.4

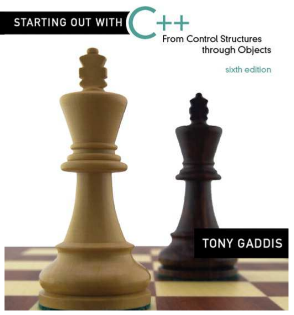
Expanding the `if` Statement

Expanding the `if` Statement

- To execute more than one statement as part of an `if` statement, enclose them in `{ }`

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

- `{ }` creates a <u>block</u> of code (Can't be omitted)
- If the condition is false, the whole block will be skipped



4.5

The `if/else` Statement

The `if/else` Statement

- Provides two possible paths of execution
- Performs one statement or block if the *`expression`* is true, otherwise performs another statement or block.
  - "If it rains, I will stay home. If not I will go to a movie."
  - "If you divide a number by 2 and the remainder is 0, it is an even number. Otherwise it is an odd number."

The `if/else` Statement

General Format:

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

or

```
if (expression)
{
  statement1;
  statement2;
  …
}
else
{
  statement1;
  statement2;
  …
}
```

A block of statements

`if/else` – what happens

To evaluate:

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

- If the *`expression`* is `true`, then *`statement1`* is executed and *`statement2`* is skipped.
- If the *`expression`* is *`false`*, then *`statement1`* is skipped and *`statement2`* is executed.

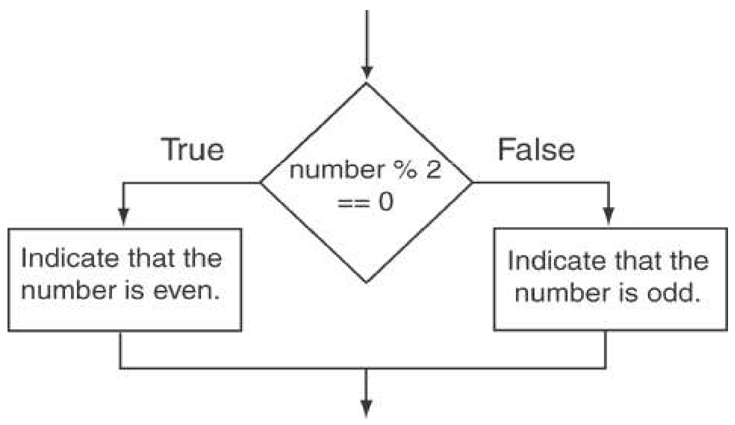
Program 4-8

```
 1  // This program uses the modulus operator to determine
 2  // if a number is odd or even. If the number is evenly divisible
 3  // by 2, it is an even number. A remainder indicates it is odd.
 4  #include <iostream>
 5  using namespace std;
 6
 7  int main()
 8  {
 9     int number;
10
11     cout << "Enter an integer and I will tell you if it\n";
12     cout << "is odd or even. ";
13     cin >> number;
14     if (number % 2 == 0)
15        cout << number << " is even.\n";
16     else
17        cout << number << " is odd.\n";
18     return 0;
19  }
```

Program Output with Example Input Shown in Bold

```
Enter an integer and I will tell you if it
is odd or even. 17 [Enter]
17 is odd.
```

Flowchart for Lines 14 through 18


True
number % 2 == 0
False
Indicate that the number is even.
Indicate that the number is odd.

Program 4-9

```
// This program asks the user for two numbers, num1 and num2.
// num1 is divided by num2 and the result is displayed.
// Before the division operation, however, num2 is tested
// for the value 0. If it contains 0, the division does not
// take place.
#include <iostream>
using namespace std;

int main()
{
   double num1, num2, quotient;

```

(Program Continues)

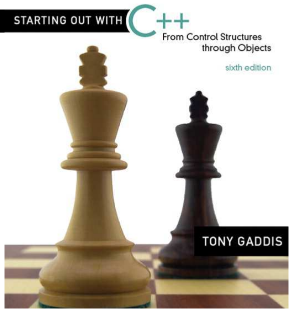
Program 4-9 *(continued)*

```
    // Get the first number.
    cout << "Enter a number: ";
    cin >> num1;

    // Get the second number.
    cout << "Enter another number: ";
    cin >> num2;

    // If num2 is not zero, perform the division.
    if (num2 == 0)
    {
       cout << "Division by zero is not possible.\n";
       cout << "Please run the program again and enter\n";
       cout << "a number other than zero.\n";
    }
    else
    {
       quotient = num1 / num2;
       cout << "The quotient of " << num1 << " divided by ";
       cout<< num2 << " is " << quotient << ".\n";
    }
    return 0;
}
```

block

Program Output with Example Input Shown in Bold

```
Enter a number: 10 [Enter]
Enter another number: 0 [Enter]
Division by zero is not possible.
Please run the program again and enter
a number other than zero.
```



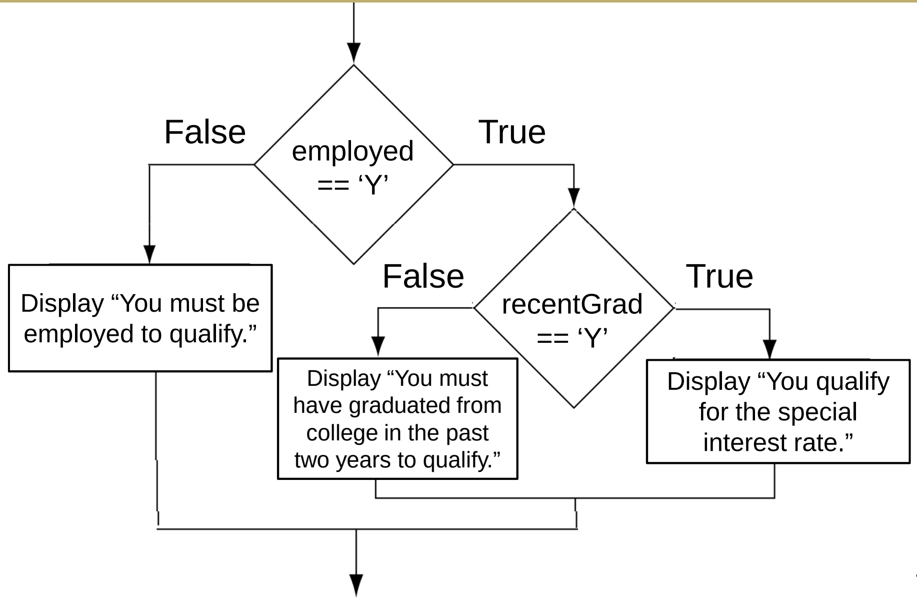
4.6

Nested `if` Statements

Nested `if` Statements

- An `if` statement can be nested inside another `if` statement
- Nested `if` statements can be used to test more than one condition
- Example:
  - A banking program determines if a customer qualifies for a special low interest loan based on two conditions:
    1) Currently employed?
    2) Recently graduated from college?

Flowchart for a Nested `if` Statement


False
employed == 'Y'
True
Display "You must be employed to qualify."
False
recentGrad == 'Y'
True
Display "You must have graduated from college in the past two years to qualify."
Display "You qualify for the special interest rate."

Nested `if` Statements – 1

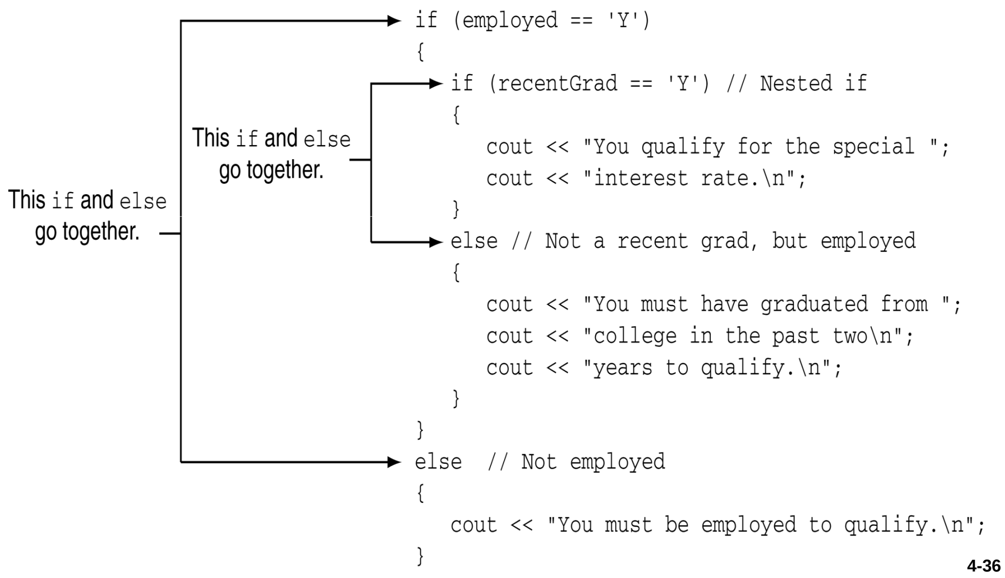
```
// Determine the user's loan qualifications.
if (employed == 'Y')
{
   if (recentGrad == 'Y') //Nested if
   {
      cout << "You qualify for the special ";
      cout << "interest rate.\n";
   }
}
```

If the customer does not qualify for the loan, the program does not print out a message to notify the user.

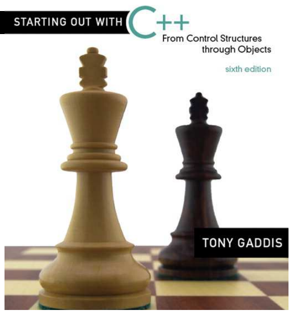
Nested `if` Statements – 2

```
// Determine the user's loan qualifications.
if (employed == 'Y')
{
   if (recentGrad == 'Y') // Nested if
   {
      cout << "You qualify for the special ";
      cout << "interest rate.\n";
   }
   else // Not a recent grad, but employed
   {
      cout << "You must have graduated from ";
      cout << "college in the past two\n";
      cout << "years to qualify.\n";
   }
}
else // Not employed
{
   cout << "You must be employed to qualify.\n";
}
```

Use Proper Indentation!



```
if (employed == 'Y')
{
   if (recentGrad == 'Y') // Nested if
   {
      cout << "You qualify for the special ";
      cout << "interest rate.\n";
   }
   else // Not a recent grad, but employed
   {
      cout << "You must have graduated from ";
      cout << "college in the past two\n";
      cout << "years to qualify.\n";
   }
}
else  // Not employed
{
   cout << "You must be employed to qualify.\n";
}
```



4.7

The `if/else if` Statement

The `if/else if` Statement

- A special nested `if` statement where the `else` part is another `if/else` statement
- Tests a series of conditions until one is found to be true
- Often simpler than using nested `if/else` statements
- Can be used to model thought processes such as:

  "If it is raining, take an umbrella,
  else, if it is windy, take a hat,
  else, take sunglasses"

`if/else if` format

```
if (expression_1)
     stmt_1;              // or block_1
else if (expression_2)
     stmt_2;              // or block_2
     ……                 // other else ifs
else if (expression_n)
     stmt_n;              // or block_n
else
     stmt_def;          // or block_def
```

How does `if/else if` work?

Program Example

```
    // Determine the letter grade.
    if (testScore < 60)
       cout << "Your grade is F.\n";
    else if (testScore < 70)
       cout << "Your grade is D.\n";
    else if (testScore < 80)
       cout << "Your grade is C.\n";
    else if (testScore < 90)
       cout << "Your grade is B.\n";
    else
       cout << "Your grade is A.\n";
```

Using a Trailing `else` to Catch Errors

The trailing else clause is optional, but is best used to catch errors

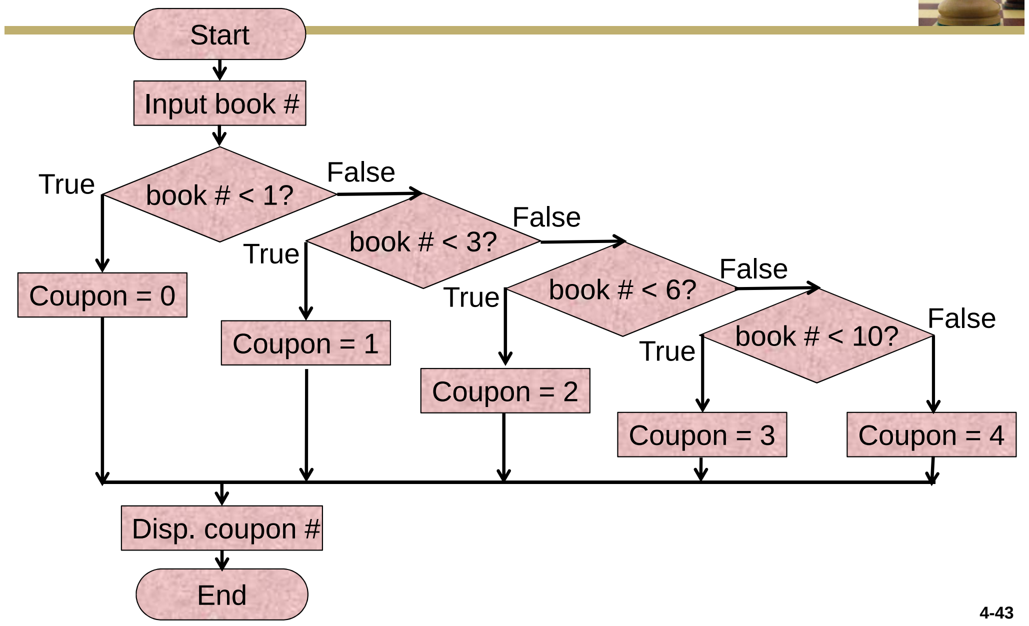
```
// Determine the letter grade.
if (testScore < 60)
   cout << "Your grade is F.\n";
else if (testScore < 70)
   cout << "Your grade is D.\n";
else if (testScore < 80)
   cout << "Your grade is C.\n";
else if (testScore < 90)
   cout << "Your grade is B.\n";
else if (testScore <= 100)
   cout << "Your grade is A.\n";
else
   cout << "We do not give scores higher than 100.\n";
```

This trailing `else` catches invalid test scores

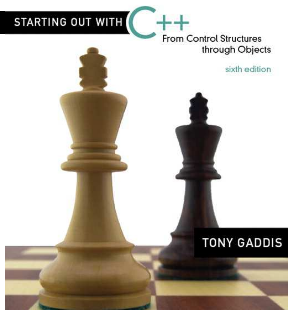
Bookstore Coupon Example

- A bookstore gives a customer discount coupons based on how many books the customer buys. If the customer does not buy any book, he/she doesn't get any coupon. If he/she buys 1 to 2 books, he/she gets 1 coupon. If he/she buys 3 to 5 books, he/she gets 2 coupons. If he/she buys 6 to 9 books, he/she gets 3 coupons. If the customer buys 10 or more books, he/she gets 4 coupons. Write a program to determine the number of coupons a customer gets.

Bookstore Coupon Example



```
#include <iostream>
using namespace std;
void main()
{
     int numBooks, numCoupons;
     cout << "How many books are sold? ";
     cin >> numBooks;
     if (numBooks < 1)
       numCoupons = 0;
     else if (numBooks < 3)
       numCoupons = 1;
     else if (numBooks < 5)
       numCoupons = 2;
     else if (numBooks < 10)
       numCoupons = 3;
     else
       numCoupons = 4;
     cout << "# of coupons = " << numCoupons << endl;
}
```



4.8

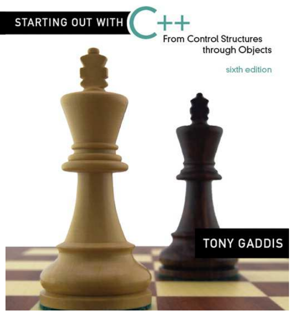
Menus

Menus

- Menu-driven program: program execution controlled by user selecting from a list of actions
- Menu: list of choices on the screen
- Menus can be implemented using `if/else if` statements

Menu-driven program organization

- Display a list of numbered or lettered choices for actions
- Prompt user to make selection
- Test user selection in *`expression`* using `if / else if`
  - if a match, then execute code for action
  - if not, then go on to next *`expression`*
- Program 4-15



4.9

Logical Operators

Logical Operators

- Used to connect two or more relational expressions into one (for testing compound conditions), or reverse the logic of an expression
- Operators, meaning, and 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, and false becomes true

Logical Operators - 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

The && Operator in Program 4-16

```
// Determine the user's loan qualifications.
if (employed == 'Y')
{
   if (recentGrad == 'Y') //Nested if
   {
      cout << "You qualify for the special ";
      cout << "interest rate.\n";
   }
}
```

```
// Determine the user's loan qualifications.
if (employed == 'Y' && recentGrad == 'Y')
{
   cout << "You qualify for the special ";
   cout << "interest rate.\n";
}
```

The || Operator in Program 4-17

The customer qualifies for the loan if his/her income is more than or equal to $35,000 **or** he/she has worked more than five years.

```
// Determine the user's loan qualifications.
if (income >= 35000 || years > 5)
   cout << "You qualify.\n";
```

The ! Operator in Program 4-18

If it is not true that the customer's income is more than or equal to $35,000 **or** has worked more than five years, he/she does not qualify for the loan.

```
    // Determine the user's loan qualifications.
    if (!(income >= 35000 || years > 5))
    {
       cout << "You must earn at least $35,000 or have\n";
       cout << "been employed for more than 5 years.\n";
    }
```

Logical Operators - notes

- ! has highest priority, followed by &&, then ||
- && and || rank lower than relational operators
- Use parentheses to avoid errors

```
int x=5, y=10, z=15;

x > 10 || y == 12 && !(z < 5)
```
F

- Must provide complete expression

```
temp < 0 || > 100
temp < 0 || temp > 100
```
(wrong)
(correct)

Logical Operators - notes

- If the value of an expression can be determined by evaluating just the sub-expression on left side of a logical operator, then the sub-expression on the right side will not be evaluated (*short circuit evaluation*)

```
int x=10, y=5;
if ( x > 100 && y < 20 )
   cout << "You win!";

if ( x < 100 || y > 20 )
   cout << "You lose!";
```

4.10

Checking Numeric Ranges with Logical Operators

Checking Numeric Ranges with Logical Operators

- Used to test if a value falls **inside** a range:

```
if (grade >= 0 && grade <= 100)
   cout << "Valid grade";
if (grade <= 100 && grade >= 90)
   cout << "Your grade is A";
```

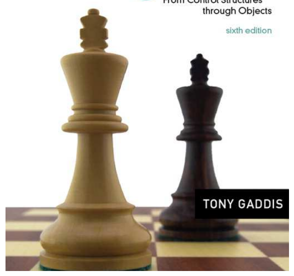
- Can also test if value falls **outside** of range:

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

- Cannot use mathematical notation:

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

STARTING OUT WITH C++ From Control Structures through Objects sixth edition TONY GADDIS



4.11

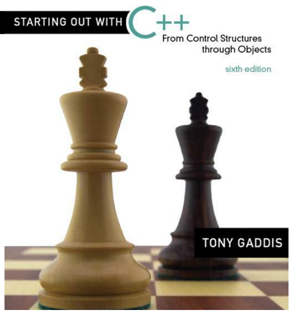
Validating User Input with Logical Operators

Validating User Input

- Input validation: inspecting input data to determine whether it is acceptable
- Bad output will be produced from bad input
- A good program should always check the validity of the input data
- Can perform various tests:
  - Range
  - Reasonableness
  - Valid menu choice
  - Divide by zero

Program with Input Validation

```
   // Get the numeric test score.
   cout << "Enter your numeric test score and I will\n";
   cout << "tell you the letter grade you earned: ";
   cin >> testScore;

   if (testScore < 0 || testScore > 100) //Input validation
   {
      // An invalid score was entered.
      cout << testScore << " is an invalid score.\n";
      cout << "Run the program again and enter a value\n";
      cout << "in the range of 0 to 100.\n";
   }
   else
   {
      // Determine the letter grade.
      if (testScore < 60)
         grade = 'F';
      else if (testScore < 70)
         grade = 'D';
      else if (testScore < 80)
         grade = 'C';
      else if (testScore < 90)
         grade = 'B';
      else if (testScore <= 100)
         grade = 'A';

      // Display the letter grade.
      cout << "Your grade is " << grade << endl;
   }
```



4.12

More About Variable Definitions and Scope

More About Variable Definitions and Scope

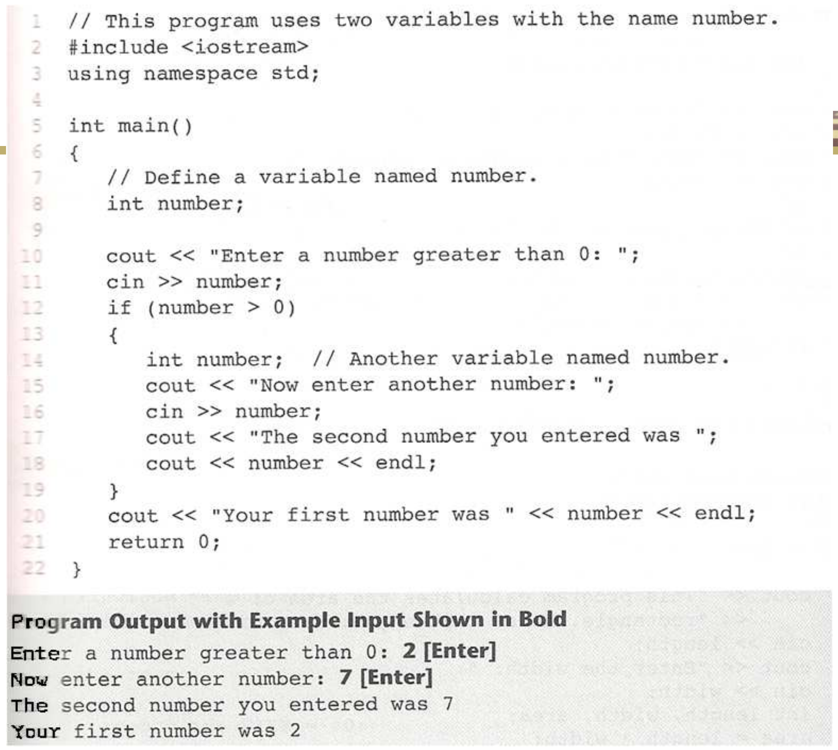
- <u>Scope</u> of a variable is the block in which it is defined, from the point of definition to the end of the block
- A block is defined by { }
- Usually defined at beginning of function
- May be defined close to its first use to make its purpose evident (especially in a long program)

```
int main()
{
   // Get the annual income.
   cout << "What is your annual income? ";
   double income;    //variable definition
   cin >> income;

   if (income >= 35000)
   {
      // Get the number of years at the current job.
      cout << "How many years have you worked at "
           << "your current job? ";
      int years;     //variable definition
      cin >> years;

      if (years > 5)
         cout << "You qualify.\n";
      else
      {
         cout << "You must have been employed for\n";
         cout << "more than 5 years to qualify.\n";
      }
   }
}
```

// This program has three layers of blocks

Still More About Variable Definitions and Scope

- Variables defined inside { } have <u>local</u> or <u>block</u> scope
- When inside a block within another block, can define variables with the same name as in the outer block
  - When in inner block, outer definition is not available
  - Not a good idea



```
// This program uses two variables with the name number.
#include <iostream>
using namespace std;

int main()
{
   // Define a variable named number.
   int number;

   cout << "Enter a number greater than 0: ";
   cin >> number;
   if (number > 0)
   {
      int number;  // Another variable named number.
      cout << "Now enter another number: ";
      cin >> number;
      cout << "The second number you entered was ";
      cout << number << endl;
   }
   cout << "Your first number was " << number << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter a number greater than 0: 2 [Enter]
Now enter another number: 7 [Enter]
The second number you entered was 7
Your first number was 2
```

STARTING OUT WITH C++ From Control Structures through Objects sixth edition

TONY GADDIS

4.15

The switch Statement

The switch Statement

- Used to make decisions like `if/else if` statements
- Uses the value of a variable or expression to determine where the program will branch
- In some cases, preferred to `if/else if` statements (e.g. menu system)

switch statement format

```
switch (IntExpr)
{
  case ConstExpr-1:
      // place one or more statements here
  case ConstExpr-2:
      // place one or more statements here
     ...
  case ConstExpr-n:
      // place one or more statements here
  default:
     // place one or more statements here
}
```

Program 4-28

```
// The switch statement in this program tells the user something
// he or she already knows: what they just entered!
#include <iostream>
using namespace std;

int main()
{
   char choice;

   cout << "Enter A, B, or C: ";
   cin >> choice;
   switch (choice)
   {
      case 'A': cout << "You entered A.\n";
                break;
      case 'B': cout << "You entered B.\n";
                break;
      case 'C': cout << "You entered C.\n";
                break;
      default:  cout << "You did not enter A, B, or C!\n";
   }
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: B [Enter]
You entered B.
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: F [Enter]
You did not enter A, B, or C!
```

switch statement requirements

1) *`IntExpr`* must be an integer variable or an expression that evaluates to an integer value
2) *`ConstExpr-1`* through *`ConstExpr-n`* must be constant integer expressions or literals, and must be unique in the `switch` statement
3) `default` is optional but recommended

switch statement – how it works

1) *`IntExpr`* is evaluated
2) The value of *`IntExpr`* is compared against *`ConstExpr-1`* through *`ConstExpr-n`*.
3) If *`IntExpr`* matches value *`ConstExpr-i`*, the program branches to the statement following *`ConstExpr-i`* and continues to the `break` statement or end of the `switch` statement
4) If no matching value is found, the program branches to the statement after `default:`

break statement

- Used to exit a `switch` statement
- If it is left out, the program "falls through" the remaining statements in the `switch` statement until a `break` statement is encountered or the end of `switch` statement is reached
- Sometimes the `break` statement is left out on purpose

Program 4-30

```
// This program is carefully constructed to use the "fallthrough"
// feature of the switch statement.
#include <iostream>
using namespace std;

int main()
{
   int modelNum;  // Model number

   // Get a model number from the user.
   cout << "Our TVs come in three models:\n";
   cout << "The 100, 200, and 300. Which do you want? ";
   cin >> modelNum;

   // Display the model's features.
   cout << "That model has the following features:\n";
   switch (modelNum)
   {
      case 300: cout << "\tPicture-in-a-picture.\n";
      case 200: cout << "\tStereo sound.\n";
      case 100: cout << "\tRemote control.\n";
                break;
      default:  cout << "You can only choose the 100,";
                cout << "200, or 300.\n";
   }
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 100 [Enter]
That model has the following features:
   Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 200 [Enter]
That model has the following features:
   Stereo sound.
   Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 300 [Enter]
That model has the following features:
   Picture-in-a-picture.
   Stereo sound.
   Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 500 [Enter]
That model has the following features:
You can only choose the 100, 200, or 300.
```

Using `switch` with a menu

- `switch` statement is a natural choice for menu-driven program:
  - display the menu
  - then, get the user's menu selection
  - use user input as *`IntExpr`* in `switch` statement
  - use menu choices as *`ConstExpr`* in `case` statements
- View program 4-32