Programming Choice Statements with Boolean (bool) Variables

Larry Caretto
Computer Science 106

Computing in Engineering and Science

February 28, 2006

California State University Northridge

Outline

- Review last week
  - Simple if statements
  - if-else-if statements
- Boolean (bool) variables
- Programming with bool variables
- Input validation
- DeMorgan's Laws
- Nested if statements

California State University Northridge 2

Review if Statements

- Implementation of choice statements in most high-level languages uses an if statement
- The C++ format is

if (***<condition>***)
{
<statements done if condition true>
}

California State University Northridge 3

Review if-else Statements

- Executes different statement blocks if condition is true or false

if (***<condition>***)
{
<statements done if condition true>
}
else
{
<statements done if condition false>
}
<Next statement after one block done>

California State University Northridge 4

Review if – else – if Structure

if (***<condition1>***)
{
<statements done if condition1 true>
}
else if (***<condition2>***)
{
<statements done if condition2 true>
}
// Place additional conditions here
// Continue on next chart

California State University Northridge 5

Review if – else – if Structure II

// Continued from previous chart
else if (***<conditionN>***)
{
<statements done if conditionN true>
}
else // optional to have this final else
{
<statements done if all conditions false>
}
<Next statement after any block done>

California State University Northridge 6

Review if – else – if Operation

- In this structure only one block of code – the code associated with the first true condition – is executed
- Conditions are scanned from top to bottom until the first true condition is found
- The code associated with that condition is executed and control is transferred to the first statement after the final block in the if – else – if structure

Boolean (bool) Variables

- Variables of type bool can hold values of expressions that are true or false
- Can be used to hold results of relational expressions
- Useful for testing complex conditions
- Variable name can give meaning to condition
- Use leap year algorithm as example

Leap Year Example

- A year is a leap year if
  - It is evenly divisible by four
  - But is not evenly divisible by 100
  - Except years evenly divisible by 400 are leap years
- Calendar programs need an algorithm to determine if a year is a leap year
- What is condition for N to be evenly divisible by M

N % M == 0

Leap Year Pseudocode

If the year is not evenly divisible by four
    The year is **not** a leap year; quit
But, if the year is evenly divisible by 400
    The year **is** a leap year; quit
But, if the year is evenly divisible by 100
    The year is **not** a leap year; quit
If no statements above are true
    The year **is** a leap year

Example – Leap Year

```
bool leap
if ( year % 4 != 0 )
{   leap = false; }
else if ( year % 400 == 0 )
{   leap = true; }
else if ( year % 100 == 0 )
{   leap = false; }
else
{   leap = true; }
```


Simpler Leap Year Example

```
bool leap = year % 4 == 0 &&
    ( year % 100 != 0 || year % 400 == 0 );
```

- This single statement gives the same result as code on previous slide
- Check for following test cases
  - 2000 is a leap year
  - 2008 is a leap year
  - 2006 is not a leap year
  - 2100 is not a leap year

Data Validation

- Programs should test input data to make sure they are reasonable
  - Lengths should be positive
  - Physical variables have known scales
  - Accounting systems expect transactions in certain ranges
  - Age as a variable is nonnegative and less than some arbitrary age (150 years?)
- Can test maximum and minimum

Data Validation Example

```
int xMin = -3, xMax = 22;
cout <<
"Enter a value for x between "
  << xMin << " and " << xMax;
int x;
cin >> x;
bool badData =
  x < xMin || x > xMax;
```


Programming Data Validation

- Will later show how to use loops
- Keep sending error message and requesting new data while user enters incorrect data
- With only if statements halt execution if user enters bad data
  - Can also use if–else–if to give user two or three tries to enter correct data then quit
  - Wait for looping to show easier way

Data Validation Example II

```
if ( badData )
{
    cout << "Your entry for x = "
         << x << " is out of range"
         << "\nMinimum x = "<< xMin
         << ", Maximum x = " << xMax
         << ".\nProgram will halt.";
    return EXIT_FAILURE
}
```


What is goodData

- Contrast badData on last chart with goodData definition below

```
bool badData = x < xMin || x >
  xMax;
bool goodData = x >= xMin && x <=
  xMax;
```

- How are these conditions related?

```
goodData = !badData;
```

- General relations: DeMorgan's Law

DeMorgan's Laws

- Have two bool variables, a and b, that can have values of true or false
- Combinations of conditions for a and b satisfy both of the following
- !(a && b) = !a || !b
- !(a || b) = !a && !b
- Can construct a truth table to verify this by looking at all possible conditions

!(a && b) = !a || !b

a	b	!a	!b	a && b
true	true	false	false	true
true	false	false	true	false
false	true	true	false	false
false	false	true	true	false
			!a || !b	!(a && b)
			false	false
			true	true
			true	true
			true	true

!(a || b) = !a && !b

a	b	!a	!b	
true	true	false	false	
true	false	false	true	
false	true	true	false	
false	false	true	true	
	a || b	!(a || b)	!a && !b	
	true	false	false	
	true	false	false	
	true	false	false	
	false	true	true	

Review Data Validation

- Apply DeMorgan's Law to Validation

```
badData = x < Min || x > Max;
goodData = x >= Min && x <= Max;
goodData = !badData;
goodData = !(x < Min || x > Max);
DeMorgan: !(a || b ) = !a && !b
goodData = !(x < Min) && !(x>Max)
goodData = x >= Min && x <= Max
```

Application of DeMorgan's Law to goodData = !badData gives expected result for goodData

Exercise Background

- An example of an iteration problem, shown below, computes $x = \sqrt{A}$

$$x^{(n+1)} = \frac{x^{(n)}}{2} + \frac{A}{2x^{(n)}}$$

- Iterations continue until converged, defined as $|x^{(n+1)} - x^{(n)}| \le \varepsilon_1 + \varepsilon_2 |x^{(n+1)}|$
- Note use of absolute values in computing convergence condition
- Allowed error, ε_1 and ε_2, set by user

Exercise Background II

- What is meaning of ε_1 and ε_2 in the condition $|x^{(n+1)} - x^{(n)}| \le \varepsilon_1 + \varepsilon_2 |x^{(n+1)}|$?
- Absolute error is given by ε_1
  - The error cannot be less than this regardless of the values of $x^{(n+1)}$
  - Controls iterations for small $x^{(n+1)}$
- Relative error given by ε_2
  - Governs when x is large
- Combination accounts for range of x

Numerical Example

- Use algorithm to find $x = \sqrt{A}$, with A = 2 and initial guess, $x^{(0)}$, = 1

$$x^{(n+1)} = \frac{x^{(n)}}{2} + \frac{A}{2x^{(n)}} = \frac{x^{(n)}}{2} + \frac{2}{2x^{(n)}} = \frac{x^{(n)}}{2} + \frac{1}{x^{(n)}}$$

$$x^{(1)} = \frac{x^{(0)}}{2} + \frac{1}{x^{(0)}} = \frac{1}{2} + \frac{1}{1} = 1.5 \qquad \left|x^{(1)} - x^{(0)}\right| = 0.5$$

$$x^{(2)} = \frac{x^{(1)}}{2} + \frac{1}{x^{(1)}} = \frac{1.5}{2} + \frac{1}{1.5} = 1.417 \qquad \left|x^{(2)} - x^{(1)}\right| = 0.083$$

$$x^{(3)} = \frac{x^{(2)}}{2} + \frac{1}{x^{(2)}} = \frac{1.417}{2} + \frac{1}{1.417} = 1.414 \qquad \left|x^{(3)} - x^{(2)}\right| = 0.003$$

More Accurate Results

n	$x^{(n)}$	$\lvert x^{(n+1)} - x^{(n)}\rvert$	True error
0	2		0.585786
1	1.5	0.5	0.085786
2	1.416666667	0.083333	0.002453
3	1.414215686	0.002451	2.12E-06
4	1.4142135624	2.12E-06	1.59E-12
5	1.4142135624	1.59E-12	2.22E-16

At Last, The Exercise

$$x^{(n+1)} = \frac{x^{(n)}}{2} + \frac{A}{2x^{(n)}} \quad until \left|x^{(n+1)} - x^{(n+1)}\right| \le \varepsilon_1 + \varepsilon_2 \left|x^{(n+1)}\right|$$

- We want to iterate until the solution is converged: $|x^{(n+1)} - x^{(n)}| \le \varepsilon_1 + \varepsilon_2 |x^{(n+1)}|$
- Define C++ variables for the mathematical terms in this iteration
  - xNew is $x^{(n+1)}$
  - xOld is $x^{(n)}$
  - e1 is ε_1
  - e2 is ε_2

|xNew – xOld| <= e1 + e2 |xNew|

At Last, The Exercise

$$x^{(n+1)} = \frac{x^{(n)}}{2} + \frac{A}{2x^{(n)}} \quad until \left|x^{(n+1)} - x^{(n+1)}\right| \le \varepsilon_1 + \varepsilon_2 \left|x^{(n+1)}\right|$$

- Define a bool variable, converged, that is true when |xNew – xOld| <= e1 + e2 |xNew| using fabs(x) for |x|

```
bool converged = fabs(xNew -
xOld) <= e1 + e2 * fabs(xNew);
```

- What condition is true the solution is converged or iterations > maximum

```
converged || iterations > maximum
```


Nested If Statements

- Can have one if block inside another
- Example: Find days for month number
  - If the number of the month is 4, 6, 9, or 11 the answer is 30
  - If the number of the month is 2
    - If it is a leap year, the answer is 29
    - Otherwise the answer is 28
  - For all other month numbers (1, 3, 5, 7, 8, 10, and 12) the answer is 31

Days in Month

```
if ( month == 4 || month == 6
  || month == 9 || month == 11 )
{
    days = 30;
}
else if ( month == 2 )
{
    if (leapYear) // bool var
    {
        days = 29;
    } // continue on next chart
```


Days in Month Continued

```
    else
    {
        days = 28;
    }
} // ends else if (month==2)
else
{
    days = 31;
}
```

Assignments

- Reading pages in text
  - Today – pp 179 – 180 and pp 196 – 199
  - Thursday – pp 226 – 240
  - March 7 – pp 262 – 266
- This week's homework problems
  - Pages 208 and 209, checkpoints 4.20, 4.21, 4.22, 4.23, and 4.24
- Exercise 5 due Tuesday, March 7

California State University Northridge

31