Chapter 5
Selection Statements

Mr. Dave Clausen
La Cañada High School

Objectives

- Construct and evaluate Boolean expressions
- Understand how to use selection statements to make decisions
- Design and test **if** statements
- Design and test **if-else** statements
- Design and test **switch** statements

Boolean Expressions

- Selection Statements
  - a control statement that help the computer make decisions
  - Certain code is executed when the condition is true, other code is executed or ignored when the condition is false.
- Control Structure
  - controls the flow of instructions that are executed.

New Data Type

- Simple data types that we already know
  - int, double, char
- New simple type
  - bool (Boolean data type: **true** or **false**)
  - bool is_prime = **false**;
  - if (is_prime = = **true**) //not necessary to test
  - if (is_prime) //preferred: is_prime contains true of false

Boolean Data Type

BOOLTEST_Dev.cpp

- In older systems
  - 0 represents false
  - the number 1 represents true
  - assign values of **true** or **false** in lowercase letters

Relational Operators, and Boolean Expressions

- Relational Operators
  - operations used on same data types for comparison
    - equality, inequality, less than, greater than
- Simple Boolean Expression
  - two values being compared with a single relational operator
  - has a value of **true** or **false**

Relational Operators

Arithmetic Operation	Meaning	Relational Operator
=	Is equal to	= =
<	Is less than	<
>	Is greater than	>
≤	less than or equal to	<=
≥	greater than or equal to	>=
≠	Is not equal to	!=

Simple Boolean Expressions

7 = = 7	true
-3.0 = = 0.0	false
4.2 > 3.7	true
-18 < -15	true
13 < 0.013	false
-17.32 != -17.32	false
a = = a	true

Order of Operations

1. ( )
2. *, /, %
3. +, -
4. = =, <, >, <=, >=, !=

Comparing Strings

- Compared according to ASCII values
  - upper case < lower case (i.e. ‘A’ < ‘a’ is true)
- When comparing 2 strings where one is a subset of the other (one string is shorter & contains all the same letters)
  - shorter string < longer string
  - i.e. “Alex” < “Alexander”

Confusing = and = =

- = = means equality in a comparison
- = means assigning a value to an identifier
- side effects are caused if we confuse the two operators

P209EX1Dev.cpp

Compound Boolean Expressions

- Logical Operators
  - And && (two ampersands) Conjunction
  - Or || (two pipe symbols) Disjunction
  - Not ! (one exclamation point) Negation
- Use parentheses for each simple expression and the logical operator between the two parenthetical expressions.
  - i.e. ((grade >= 80) && (grade < 90))

Truth Tables for AND / OR

Expression1 (E1)	Expression2 (E2)	E1&&E2	E1||E2
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Truth Table for NOT
Order of Priority for Booleans

Expression	Not E
(E)	!E
true	false
false	true

- Order Of Priority in Boolean Expressions
  - 1. !
  - 2. &&
  - 3. ||

Order of Operations including Booleans

- 1. ( )
- 2. !
- 3. *, /, %
- 4. +, -
- 5. <, <=, >, >=, = =, !=
- 6. &&
- 7. ||

Complements

Operation	Complement (equivalent)
! <	>=
! <=	>
! >	<=
! >=	<

DeMorgan's Laws

! (A && B) is the same as: !A || !B

Not (true AND true)

Not (true)

false

Not(true) OR Not(true)

false OR false

false

!(A || B) is the same as: !A && !B

Not(true OR true)

Not (true)

false

Not(true) AND Not(true)

false AND false

false

If Statements

- Format for if statements:

  if (<Boolean expression>)

  <statement>

  – Parentheses are <u>required</u> around the Boolean Expression.

```
sum = 0.0;
cin>>number;
if (number > 0.0)
   sum = sum + number;
cout<<"the sum is: "<<sum<<endl;
```

Compound Statements

- Use the symbols { and } to group several statements as a single unit.
- Simple statements within the compound statements end with semicolons.
- Compound Statements are sometimes called a statement block.
- Use compound statements in an if statement if you want several actions executed when the Boolean expression is true.

MINMAXDev.cpp

if …else Statements

- Format for if…else statements

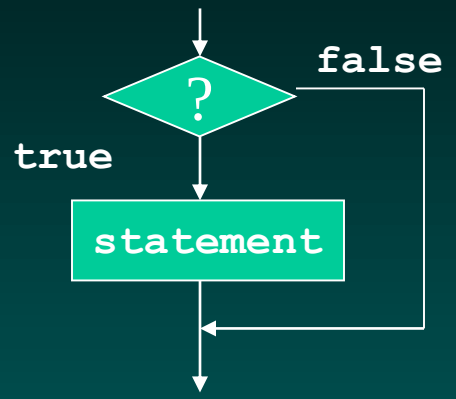
```
if (<Boolean expression>)
{
    <true statement>
    //end of if option
}
else
{
    <false statement>
    //end of else option
}
```

if …else Example

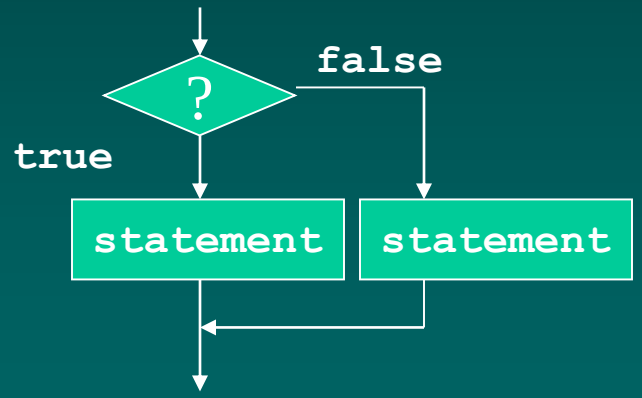
```
cout<<"Please enter a number and press <Enter>. ";
cin>>number;
if (number < 0)
{
   neg_count = neg_count + 1;
   cout<<setw(15) << number<<endl;
  //end if option
}
else
{
   non_neg_count = non_neg_count + 1;
   cout << setw(30) << number << endl;
 //end of else option
}
```

Behavior of Selection Statements

```
if (<Boolean expression>)
{
   <statement 1>
   .
   <statement n>
}
```




```
if (<Boolean expression>)
   <statement>
else
{
   <statement 1>
   .
   <statement n>
}
```



Robust Programs

- Robust Programs are protected from:
  - most possible crashes
  - bad data
  - unexpected values
- For student programs
  - balance “error trapping” and code efficiency
  - our text assumes valid data is entered when requested.

Nested if statements

- Nested if statement (avoid this for this class)
  - an if statement used within another if statement where the "true" statement or action is.

```
if (score >=50)
    if (score>=69.9)
        cout<<blah, blah, blah      //true for score>=69.9 and score>=50
    else
        cout<<blah, blah, blah      //false score>=69.9 true score >=50
else
   cout<<blah, blah, blah           //false for score >=50
```

Extended if statements

- Extended if statements are Nested if statements where another if statement is used with the else clause of the original if statement. (use this for this class)

```
if  (condition 1)
    action1
else if (condition2)
        action2
      else
          action3
```

Avoid Sequential Selection

- This is not a good programming practice.
  - Less efficient
  - only one of the conditions can be true
    - this is called mutually exclusive conditions

```
if (condition1)          //avoid this structure
    action1              //use nested selection
if (condition2)
    action2
if (condition3)
    action3
```

SALESDev.cpp

Program Testing

- Use test data that tests every branch or selection in the program.
- Test the if statement(s)
- Test the else statement(s)
- Test the border, edge, extreme cases.
- Test carefully nested and extended selection statements and their paths.

Switch Statements

- Allows for multiple selection that is easier to follow than nested or extended if statements.

```
switch (age)    //age is of type int
{
   case 18:     <statement1>
                break;
   case 19:     <statement2>
                break;
   case 20:     <statement3>
                break;
   default:     <default statement>
}
```

Switch: Flow of Execution

- The selector (argument) for switch must be of an ordinal type (not double)
  - switch (age)
  - The variable “age” is called the selector in our example.
  - If the first instance of the variable is found among the labels, the statement(s) following this value is executed until reaching the next break statement.
  - Program control is then transferred to the next statement following the entire switch statement.
  - If no value is found, the default statement is executed.

Switch Statement Example 2

```
switch (grade)                    //grade is of type char
{
    case 'A' :
    case 'B' :      cout<<"Good work!"<<endl;
                    break;
    case 'C' :      cout<<"Average work"<<endl;
                    break;
    case 'D' :
    case 'F' :      cout<<"Poor work"<<endl;
                    break;
    default  :      cout<<grade<<" is not a valid letter grade.";
                    break;
}
```

Assertions

- Assertions
  - A statement about what we expect to be true at a certain point in the program where the assertion is placed.
  - Could be used with selection, functions, and repetition to state what you expect to happen when certain conditions are true.
  - Could be comments, should be Boolean expressions that could be evaluated by the compiler.
  - Are used as preconditions and post conditions
  - Can be used as a proof of your program's correctness
  - Can help you "debug" your program. If there is an error, the program halts with an error message and the line number where the error is located.

Implementing Assertions

- Use #include <cassert>
  - Example 1:

    assert (number_of_students != 0);

    class_average = sum_of_scores / number_of_students;
  - The parameter of assert is a Boolean Expression.
  - The compiler verifies if the assertion is true or false
    - if false, the program halts with an error message:

    Assertion Failed: number_of_students !=0, file "filename", line 48
    - this tells you that your assertion is false

Assertion Example #2

- Assertions used as pre & post conditions:

```
assert((num1>=0) && (num2>=0));  //precondition
if (num1 < num2)
{
    temp = num1;
    num1 = num2;
    num2 = temp;
}
assert((num1>=num2) && (num2>=0));  //postcondition
```

Assert as a Debugging Tool

- Executable assertions when false, terminate your program and tell you the line number where the error is located.
- This is valuable during the software debugging and testing phases.
- When your program is working correctly, you do not need to remove all the assertions manually.
- Add the preprocessor directive:

#define NDEBUG BEFORE

#include <cassert> to ignore all assert functions.

//NDEBUG means No debug.

assertDev.cpp