Relational Operators

Relational operators

\[ \begin{align*}
\text{>, <, >=, <=, ==, !=} \\
\end{align*} \]

if/else examples

```
if (income > 30000)
    txtRate = 0.23;
else
    txtRate = 0.15;
```

```
if (income > 30000)
    txtRate = 0.23;
    cout << "The tax rate is " << txtRate << endl;
else
    txtRate = 0.15;
    cout << "The tax rate is " << txtRate << endl;
```
The value of a Relationship

• Relational expressions are also known as Boolean expressions (true or false).

```cpp
int a = 4;
cout << (a == 4);   // 1
```

• == equality operator

• = assignment operator

The value of a Relationship

• Assume x is 25, y is 7, what are the values of the following expression:

```cpp
int x = 25, y = 7;
x < y
x > y
x >= y
x <= y
y != x
```

• false -> 0

• true -> 1 or anything other than 0

The value of a Relationship

```cpp
int val = 10, size = 4;
if (val + size)
if (sizeof(int))
if (-val)
if (val = size)
if (val == size)
```

• What are the values inside the parenthesis:

```cpp
if (val + size)  // true
if (sizeof(int))  // true
if (-val)        // true
if (val = size)  // true
if (val == size) // false
```

Comparing floating-point numbers

• Because of the way that floating-point numbers are represented using binary numbers, some fractional members cannot be exactly stored in memory. Rounding errors sometimes occur.

```cpp
double a = 1.5, b = 1.5;
a += 0.0000000000000000001;
if (a == b)
cout << "a and b are the same.";
else
```

Flags

• Because the condition needs to be true or false, you can use a bool variable.

```cpp
bool highScore = false;
if (average > 95)
highScore = true;
if (highScore)
cout << "Good score!";
```

• Integer flags

```cpp
int highScore = 0;
if (average > 95)
highScore = 1;
if (highScore)
cout << "Good score!";
```
Nested if statements

To test more than one condition, an if statement can be nested inside another if statement.

Example

```cpp
if (gpa_score >= 3.0)
{
    if (gpa_score >= 3.5)
        cout << "Your GPA satisfied the requirements of PhD student!\n";
    else
        cout << "Your GPA satisfied the requirements of master student!\n";
}
else
    cout << "Your GPA is lower than the requirement of graduate school!\n";
```

The inside if statement is entirely inside the if statement portion of the outside if statement.

Nested statements

- The inside if statement is entirely inside the if statement portion of the outside if statement.
- The code can grow complex and difficult to read if a series of nested if/else statements is not indented and formatted well.

Menu-driven program execution controlled

Menu-driven programs execute actions based on user selection of a list of numbered or lettered choices.

Example

Menu:

1. Add two numbers
2. Subtract two numbers
3. Multiply two numbers
4. Divide two numbers
5. Exit

Enter choice: 3

Menu-driven programs can be implemented using if/else if/else statements.

Example

```cpp
if (choice == 1)
    cout << "Result: \n";
else if (choice == 2)
    cout << "Result: \n";
else if (choice == 3)
    cout << "Result: \n";
else if (choice == 4)
    cout << "Result: \n";
else
    cout << "Invalid choice.\n";
```

If/else if/else statement

- Format

```cpp
if (condition_1)
{
    statements
}
else if (condition_2)
{
    statements
}
else
{
    statements
}
```

If/else if/else statements

- Example

```cpp
if (gpa_score >= 3.5)
    cout << "Your GPA satisfied the requirements of PhD student!\n";
else if (gpa_score >= 3.0)
    cout << "Your GPA satisfied the requirements of master student!\n";
else
    cout << "Your GPA is lower than the requirement of graduate school!\n";
```

Menu-driven programs

- Program execution controlled by user selecting from a list of actions.
- Menus can be used to implement if/else if/else statements.

Example

```cpp
if (choice == 1)
    cout << "Result: \n";
else if (choice == 2)
    cout << "Result: \n";
else if (choice == 3)
    cout << "Result: \n";
else if (choice == 4)
    cout << "Result: \n";
else
    cout << "Invalid choice.\n";
```

Nested if statements

- Example

```cpp
if (gpa_score >= 3.5)
    cout << "Your GPA satisfied the requirements of PhD student!\n";
else if (gpa_score >= 3.0)
    cout << "Your GPA satisfied the requirements of master student!\n";
else
    cout << "Your GPA is lower than the requirement of graduate school!\n";
```

- The inside if statement is entirely inside the if statement portion of the outside if statement.
- The code can grow complex and difficult to read if a series of nested if/else statements is not indented and formatted well.
Menus

Many programs use menus:

```c++
int choice;
cout << "Enter the type of problem you would like to solve
1. addition
2. subtraction
3. multiplication
4. division
5. quit"
; cin>>choice;
if(choice==1)
result=num1+num2;
else if(choice==2)
result=num1-num2;
else if(choice==3)
result=num1*num2;
else if(choice==4)
result=num1/num2;
else
cout<<"Good-bye";
```

Logical Operators

• Used to create relational expressions from other relational expressions

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>Operators, meaning, and explanation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp; &amp;</td>
<td></td>
</tr>
<tr>
<td>! T =&gt; F</td>
<td>Logical Operators</td>
</tr>
<tr>
<td>T &amp;&amp; T =&gt; T</td>
<td>– ! is unary–high precedence</td>
</tr>
<tr>
<td>T &amp;&amp; F =&gt; F</td>
<td>–</td>
</tr>
<tr>
<td>F &amp;&amp; F =&gt; F</td>
<td>– &amp;&amp; just before</td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>! T =&gt; F</td>
<td>– &amp;&amp; is just before</td>
</tr>
<tr>
<td>!F =&gt; T</td>
<td>–</td>
</tr>
</tbody>
</table>

Checking numeric ranges

• If you want to identify if a number is inside a certain range use the && operator
  ```c++
  if (age>=12 && age<20)
  cout << "You are a teenager."
  ```

• If you want to check to see if a number is outside a certain range use the || operator
  ```c++
  if (age<12 || age>=20)
  cout << "You are not a teenager."
  ```

Precedence and Associativity

Manipulating inequality:

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>Operators, meaning, and explanation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;= &lt;=</td>
<td>Lessthan or Equal To</td>
</tr>
<tr>
<td>==</td>
<td>Equal To</td>
</tr>
<tr>
<td>!=</td>
<td>Not Equal To</td>
</tr>
<tr>
<td>&gt; &lt;</td>
<td>Greater Than or Equal To</td>
</tr>
<tr>
<td>&gt; &lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
</tr>
</tbody>
</table>

Manipulating logic:

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>Operators, meaning, and explanation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>! T =&gt; F</td>
<td>Logical Operators</td>
</tr>
<tr>
<td>T &amp;&amp; T =&gt; T</td>
<td>– ! is unary–high precedence</td>
</tr>
<tr>
<td>T &amp;&amp; F =&gt; F</td>
<td>–</td>
</tr>
<tr>
<td>F &amp;&amp; F =&gt; F</td>
<td>– &amp;&amp; just before</td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>! T =&gt; F</td>
<td>– &amp;&amp; is just before</td>
</tr>
<tr>
<td>!F =&gt; T</td>
<td>–</td>
</tr>
</tbody>
</table>

Manipulating inequality:

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>Operators, meaning, and explanation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;= &lt;=</td>
<td>Lessthan or Equal To</td>
</tr>
<tr>
<td>==</td>
<td>Equal To</td>
</tr>
<tr>
<td>!=</td>
<td>Not Equal To</td>
</tr>
<tr>
<td>&gt; &lt;</td>
<td>Greater Than or Equal To</td>
</tr>
<tr>
<td>&gt; &lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
</tr>
</tbody>
</table>

Manipulating logic:

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>Operators, meaning, and explanation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>! T =&gt; F</td>
<td>Logical Operators</td>
</tr>
<tr>
<td>T &amp;&amp; T =&gt; T</td>
<td>– ! is unary–high precedence</td>
</tr>
<tr>
<td>T &amp;&amp; F =&gt; F</td>
<td>–</td>
</tr>
<tr>
<td>F &amp;&amp; F =&gt; F</td>
<td>– &amp;&amp; just before</td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>! T =&gt; F</td>
<td>– &amp;&amp; is just before</td>
</tr>
<tr>
<td>!F =&gt; T</td>
<td>–</td>
</tr>
</tbody>
</table>

Manipulating inequality:

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>Operators, meaning, and explanation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;= &lt;=</td>
<td>Lessthan or Equal To</td>
</tr>
<tr>
<td>==</td>
<td>Equal To</td>
</tr>
<tr>
<td>!=</td>
<td>Not Equal To</td>
</tr>
<tr>
<td>&gt; &lt;</td>
<td>Greater Than or Equal To</td>
</tr>
<tr>
<td>&gt; &lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
</tr>
</tbody>
</table>

Manipulating logic:

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>Operators, meaning, and explanation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>! T =&gt; F</td>
<td>Logical Operators</td>
</tr>
<tr>
<td>T &amp;&amp; T =&gt; T</td>
<td>– ! is unary–high precedence</td>
</tr>
<tr>
<td>T &amp;&amp; F =&gt; F</td>
<td>–</td>
</tr>
<tr>
<td>F &amp;&amp; F =&gt; F</td>
<td>– &amp;&amp; just before</td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>! T =&gt; F</td>
<td>– &amp;&amp; is just before</td>
</tr>
<tr>
<td>!F =&gt; T</td>
<td>–</td>
</tr>
</tbody>
</table>
Checking numeric ranges

• Identify the edges first, then decide if you want && or ||. Use several numbers to test the correctness of the condition.

• Beware not to create a condition that will always be true or always be false:
  
```java
if (age>=12 || age<20)   //always true
```

```java
if (age<12 && age>=20) //always false
```

Validating User Input

• Many times when we get input from a user we want to make sure they entered a correct number. Then we use several numbers to test the correctness of the condition that will always be true or always be false. The correctness of the condition will determine if you want && or ||. Use several numbers to test the correctness of the condition that will always be true or always be false.

  ```java
  //always true
  if (age>=12 || age<20)
  ```

Validating User Input

Style Rule 11

• Whenever there is a potential for a divide by zero problem, your code is to divide by zero.

Scope Rule #2

• Scope of a variable is the block in which it is defined, from the point of definition to the end of the block.

• Usually defined at beginning of function.

• May be defined close to first use if speed is an issue.
Scope Rule #2

- Variables defined inside `{ }` have local or block 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.
- See page-211 Program 4-22

Comparing Characters and C-strings

- We can compare characters and C-strings.
- Comparing Characters
  - Which is greater?
    - A
    - W
    - a
    - W
  - You can compare characters. They are in the ASCII table.
  - A-Z are 65-90
  - a-z are 97-122

- Comparing C-strings
  - You cannot use relational operators with C-strings.
  - Using `=` with C-strings will always return false because the compiler is really storing the addresses of where the string begins.
  - (name1 == name2) is comparing the addresses.
  - Since each c-string has its own memory location, the result is always false.
- Must use the `strcmp` function to compare C-strings.
  - `strcmp` compares the ASCII codes of the strings character-by-character.
  - It returns a negative number if `title1` < `title2`
  - It returns a positive number if `title1` > `title2`
  - It returns 0 if they are the same
  - The address of `title1`, `title2`
- You can compare characters. They are in the ASCII table.
  - A-Z are 65-90
  - a-z are 97-122
- Which is greater?
  - A
  - W
  - a
  - W
- Comparing C-strings
  - Must use the `strcmp` function to compare C-strings.
  - `strcmp` compares the ASCII codes of the characters in the C-strings.
  - Comparison is character-by-character.
  - `strcmp` returns 0 if they are the same.
  - It returns a positive number if `title1` > `title2`.
  - It returns a negative number if `title1` < `title2`.
- See page-211 Program 4-22

Variables declared inside `{ }` have { } local or block scope.
- Vartiables defined inside another block.
- When inside a block within another block, can define variables with the same name.
- Block scope.
Comparing C-strings

• greater or smaller is determined using the ASCII table
• ONLY needs to find first non-matching character to determine outcome
• In the ASCII table A-Z are 65-90
  lower case are 97-122
  • lowercase > uppercase
  • which is greater? Bill bill

Conditional Operator

• if you are only testing to see if they are the same
• returns 0 if they are the same
  if (strcmp(title1, title2) == 0)
  or
  if (!strcmp(title1, title2))

• if you are only testing to see if they are the same

Conditional Operator

• Can use to create short if/else statements
• Format: expr ? expr : expr;

• x<0 ? y=10 : z=20;

First Expression:
Expression to be tested

2nd Expression:
Executes if first expression is true

3rd Expression:
Executes if the first expression is false

Conditional Operator

• The value of a conditional expression is:
  – The value of the second expression if the first expression is true
  – The value of the third expression if the first expression is false

• Parentheses ( ) may be needed in an expression due to precedence of conditional operator

Comparing C-strings

- A-Z
  • Which is greater? Bill
  • Lowercase > uppercase
  • Are 97-122
  • In the ASCII table A-Z are 65-90

  Character to determine outcome
  • Only needs to find first non-matching character or smaller is determined using

Comparing C-strings
The switch Statement

1. Used to select among statements from several alternatives
2. In some cases, can be used instead of if/else if statements

```
switch (expression) //integer
{
    case exp1:
        statement1;
    case exp2:
        statement2;
    ...          // default
    case expn:
        statementn;
    default:
        statementn+1;
}
```

• expression must be an integer variable or an expression that evaluates to an integer value
• exp1 through expn must be constant integer expressions or literals, and must be unique in the switch statement
• default is optional but recommended
• break statement - how it works
  1) expression is evaluated
  2) The value of expression is compared against exp1 through expn
  3) If no matching value is found, the program branches to the statement after the default: break statement
  4) If a matching value is found, the program branches to the statement following the expi case and continues to the end of the switch statement
The switch statement

Using switch with a menu

Testing for File Open Errors

```cpp
switch (switch_expression) {
  case switch_choice:
    // Code to handle the case
    break;
  case another_switch_choice:
    // Code to handle another case
    break;
  default:
    // Code to handle default case
    break;
}
```

- Can test a file stream object to detect if an open operation failed:
  ```cpp
  infile.open("test.txt");
  if (!infile)
    cout << "File open failure!";
  ```

- Can also use the `fail` member function:
  ```cpp
  infile.open("test.txt");
  if (infile.fail())
    cout << "File open failure!";
  ```

The switch statement is a natural choice for

- Using menu choices as `expr` in case
- Using user input as `expression` in `switch`
- Then get the users menu selection
- Display the menu
- Can test menu choices as `expr` in case
- Menu-driven program:
  ```cpp
  switch (menu_choice) {
    case menu_choice1:
      // Code to handle the menu choice
      break;
    case menu_choice2:
      // Code to handle another menu choice
      break;
    default:
      // Code to handle default menu choice
      break;
  }
  ```

Testing for File Open Errors

• The file stream member function `open` is used to open a file.
• The file stream member function `open` is used to open a file.
• The file stream member function `open` is used to open a file.
• The file stream member function `open` is used to open a file.
• The file stream member function `open` is used to open a file.

The switch statement