Relational Operators

- relational operators (result is bool):
  - == Equal to (do not use =)
  - != Not equal to
  - > Greater than
  - < Less than
  - >= Greater than or equal to
  - <= Less than or equal to

```c++
int x=90;
int n=6;
// 7 < 25
// 89 == x
// x % 2 != 0
// 8 + 5 * 10 <=10 * n
```

- operator precedence:

```
Which operation happens first? next? ...
int x, y;
... x < y -10 ...
... x * 5 >= y + 10 ...
bool t1 = x > 7;
bool t2 = x * 5 >= y + 10;
```

if/else

- if and else
  - if expression is true, statement 1 is executed
  - if expression is false, statement2 is executed

```c++
double rate, monthlySales;
if (monthlySales > 3000)
  rate = .025;
else
  rate = .029;
```

- the else is optional:
  - if expression is true, statement is executed, otherwise statement is skipped

```c++
if (expression)
  statement
else
  statement2
```

Block or compound statement

- a set of statements inside braces:

```c++
{
  int x;
  cout << "Enter a value for x: " << endl;
  cin >> x;
}
```

- This allows us to use multiple statements when by rule only one is allowed.

```c++
int number;
cout << "Enter a number" << endl;
cin >> number;
if (number % 2 == 0)
{
  number = number / 2;
cout << "0";
}
else
{
  number = (number + 1) / 2;
cout << "1";
}
Nested if/else

- if-else is a statement. It can occur as a statement inside of another if-else statement.

```java
if (testScore < 60)
    grade = 'F';
else {
    if (testScore < 70)
        grade = 'D';
    else {
        if (testScore < 90)
            grade = 'C';
        else
            grade = 'B';
    }
    grade = 'A';
}
```

This is equivalent to the code on the left. It is just formatted differently.

```java
if (testScore < 60)
    grade = 'F';
else if (testScore < 70)
    grade = 'D';
else if (testScore < 80)
    grade = 'C';
else if (testScore < 90)
    grade = 'B';
else
    grade = 'A';
```

Logical Operators

- logical operators (values and results are bool):

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- operator precedence:

- examples T/F?:

```java
int x=6;
int y=10;
a. x = 5 && y <= 3
b. x > 0 && x < 10
c. x == 10 || y == 10
d. x == 10 || x == 11
e. !(x > 0)
f. !(x > 6 || y == 10)
```

switch statement

- switch stmt:

  ```java
  switch (expression) {
      case constant: statements
          ...
      case constant: statements
      default: statements
  }
  ```

  - execution starts at the case labeled with the value of the expression.
  - if no match, start at default
  - use break to exit switch (usually at end of statements)

- example:

  ```java
  switch (ch) {
      case 'a':
      case 'A': cout << "Option A";
                  break;
      case 'b':
      case 'B': cout << "Option B";
                  break;
      default: cout << "Invalid choice";
  }
  ```

Input Validation

- Input validation: inspecting input data to determine whether it is acceptable
- Invalid input is an error that should be treated as an exceptional case.
  - The program can ask the user to re-enter the data
  - The program can exit with an error message

```java
cout << "Enter a score between 0 and 100: ";
cin >> score;
if (score >= 0 && score <= 100) {
    //do something with score here
} else {
    cout << "That is an invalid score. \n";
}
```
More assignment statements

- **Compound assignment**
  
<table>
<thead>
<tr>
<th>operator</th>
<th>usage</th>
<th>equivalent syntax:</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += e;</td>
<td>x = x + e;</td>
</tr>
<tr>
<td>-=</td>
<td>x -= e;</td>
<td>x = x - e;</td>
</tr>
<tr>
<td>*=</td>
<td>x *= e;</td>
<td>x = x * e;</td>
</tr>
<tr>
<td>/=</td>
<td>x /= e;</td>
<td>x = x / e;</td>
</tr>
</tbody>
</table>

- **increment, decrement**
  
<table>
<thead>
<tr>
<th>operator</th>
<th>usage</th>
<th>equivalent syntax:</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>x++;</td>
<td>++x; x = x + 1;</td>
</tr>
<tr>
<td>--</td>
<td>x--;</td>
<td>--x; x = x - 1;</td>
</tr>
</tbody>
</table>

while loops

- **while**
  
  while (expression)
  
  statement

  - if expression is true, statement is executed, repeat

  **Example:**

  ```
  int number;
  cout << “Enter a number, 0 when finished: ”; cin << number;
  while (number != 0)
  {
    cout << “You entered ” << number << endl;
    cout << “Enter the next number: “; cin >> number;
  }
  cout << “Done” << endl;
  ```

  **output:**

  Enter a number, 0 when finished: 22
  You entered 22
  Enter the next number: 5
  You entered 5
  Enter the next number: 0
  Done

  for loops

  - **for:**
    
    for (expr1; expr2; expr3)
    
    statement

    - equivalent to:
      
      ```
      expr1;
      while (expr2) {
        statement
        expr3;
      }
      ```

    - Good for implementing count-controlled loops:
      
      **pattern:** for (initialize; test; update)

      ```
      for (int number = 1; number <= 3; number++)
      {
        cout << “Student” << number << endl;
      }
      cout << “Done” << endl;
      ```

  two kinds of loops

  - **conditional loop**
    
    * execute as long as a certain condition is true

  - **count-controlled loop:**
    
    * executes a specific number of times
      - initialize counter to zero (or other start value).
      - test counter to make sure it is less than count.
      - update counter during each iteration.

    ```
    int number = 1;
    while (number <= 3)
    {
      cout << “Student” << number << endl;
      number = number + 1; // or use number++
    }
    cout << “Done” << endl;
    ```

    number is a “counter”. It keeps track of the number of times the loop has executed.
do-while loops

- do while:

  ```
  do
  statement
  while (expression);
  ```

  statement may be a compound statement (a block: {statements})

  statement is executed.
  if expression is true, then repeat

- The test is at the end, statement ALWAYS executes at least once.

```
int number;
do {
  cout << "Enter a number, 0 when finished: ";
  cin >> number;
  cout << "You entered " << number << endl;
} while (number != 0);
```

Keeping a running total (summing)

- Example:

```
int days;
float total = 0.0; //Accumulator
cout << "How many days did you run? ";
cin >> days;
for (int i = 1; i <= days; i++)
{
  float miles;
  cout << "Enter the miles for day " << i << " ";
cin >> miles;
  total = total + miles;
}
cout << "Total miles run: " << total << endl;
```

Sentinel controlled loop

- Use a special value to signify end of the data:

```
float total = 0.0; //Accumulator
float miles;
cout << "Enter the miles you ran each day, ";
cout << "one number per line.\n";
cin >> miles;
while (miles != -1)
{
  total = total + miles;
cin >> miles;
}
cout << "Total miles run: " << total << endl;
```

- Sentinel value must NOT be a valid value

Nested loops

- When one loop appears in the body of another

```
for (row=1; row<=3; row++)  //outer
{
  for (col=1; col<=3; col++) //inner
    cout << row * col << " ";
cout << endl;
}
```

Output:

```
1 2 3
2 4 6
3 6 9
```
continue and break Statements

- Use `break` to terminate execution of a loop
- When used in a nested loop, terminates the inner loop only.

- Use `continue` to go to end of current loop and prepare for next repetition
- While, do-while loops: go immediately to the test, repeat loop if test passes
- For loop: immediately perform update step, then test, then repeat loop if test passes

Sample Problem 1

- A software company sells a package that retails for $99. Quantity discounts are given according to the following table.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-19</td>
<td>20%</td>
</tr>
<tr>
<td>20-49</td>
<td>30%</td>
</tr>
<tr>
<td>50-99</td>
<td>40%</td>
</tr>
<tr>
<td>100 or more</td>
<td>50%</td>
</tr>
</tbody>
</table>

Write a program that asks for the number of units sold and computes the total cost of the purchase.
- Input Validation: Make sure the number of units is greater than 0. Otherwise output an error message.

Sample Problem 2

- In Programming Challenge 10 of Chapter 3 you were asked to write a program that converts a Celsius temperature to Fahrenheit. Modify that program so it uses a loop to display a table of the Celsius temperatures 0–20, and their Fahrenheit equivalents.

Sample Problem 3

- Write a program with a loop that lets the user enter a series of integers. The user should enter -99 to signal the end of the series. After all the numbers have been entered, the program should display the largest and smallest numbers entered.
- Modify the program so that it also displays “ALL POSITIVE” if all of the numbers are greater than zero. Otherwise it should output “NOT all positive”.