Relational Operators

- relational operators (result is bool):
  
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

- 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)
    statement1
  else
    statement2
  ```

  - if expression is true, statement1 is executed
  - if expression is false, statement2 is executed

- the else is optional:

  ```
  if (expression)
    statement
  ```

  - if expression is true, statement is executed, otherwise statement is skipped

Block or compound statement

- a set of statements inside braces:

  ```
  {
    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.
Nested if/else

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

```
if (score >= 90)
  grade = 'A';
else {
  if (score >= 80)
    grade = 'B';
  else if (score >= 70)
    grade = 'C';
  else if (score >= 60)
    grade = 'D';
  else
    grade = 'F';
}
```

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

```
if (score >= 90)
  grade = 'A';
else if (score >= 80)
  grade = 'B';
else if (score >= 70)
  grade = 'C';
else if (score >= 60)
  grade = 'D';
else
  grade = 'F';
```

- The braces are optional on this side.

Nested if/else

- Here is a flowchart indicating the flow of control during execution of the nested if on the previous slide:

Logical Operators

- logical operators (values and results are bool):
  ```
  !a is true when a is false
  a && b is true when both a and b are true
  a || b is true when either a or b is true
  ```

- operator precedence:

```
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:**
  ```cpp
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:**
  ```cpp
  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

  ```cpp
  cout << "Enter a score between 0 and 100: ";
  cin >> score;
  if (score < 0 || score > 100) {
      cout << "That is an invalid score." << endl;
  } else {
      //do something with score here
  }
  ```

### More assignment statements

- **Compound assignment**
  ```cpp
  operator | usage | equivalent syntax:
  --- | --- | ---
  += | x += e; | x = x + e;
  -= | x -= e; | x = x - e;
  *= | x *= e; | x = x * e;
  /= | x /= e; | x = x / e;
  ```

- **increment, decrement**
  ```cpp
  operator | usage | equivalent syntax:
  --- | --- | ---
  ++ | x++; | ++x; x = x + 1;
  -- | x--; | --x; x = x - 1;
  ```

### while loops

- **while**
  ```cpp
  while (expression) {
      statement
  }
  ```
  statement may be a compound statement (a block: {statement(s)})

- if expression is true, statement is executed, repeat

- **Example:**
  ```cpp
  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
  You entered 5
  Enter the next number: 0
  Done
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.

```cpp
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.

for loops

- for:
  ```cpp
  for (expr1; expr2; expr3)
  {
    statement
  }
  ```
  - equivalent to:
  ```cpp
  while (expr2) {
    statement
  }
  ```
  - Good for implementing count-controlled loops:
    pattern: for (initialize; test; update)

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

Keeping a running total (summing)

- Example:

```cpp
int days;
float total = 0.0; //Accumulator
cout << "How many days did you ride your bike? ";
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 ridden: " << total << endl;
```
Sentinel controlled loop

- A sentinel controlled loop continues to process data until reaching a special value (called the sentinel) that signals the end.
- The first item is retrieved before the loop starts. This is called the priming read, since it gets the process started.
- If the first item is the sentinel, the loop terminates and no data is processed.

```
get the first data item
while item is not the sentinel
    process the item
get the next data item
```

Example: summing using a sentinel

```cpp
float total = 0.0; //Accumulator
float miles;
while (miles != -1)
{
    total = total + miles;
    cout << "Enter the miles you rode (-1 to quit): ";
    cin >> miles;
}
cout << "Total miles ridden: " << total << endl;
```

Nested loops

- When one loop appears in the body of another
- For every iteration of the outer loop, we do all the iterations of the inner loop

```
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
Example problem: Future Value

• Money deposited in a bank account earns interest annually. How much will the account be worth 10 years from now?
• Inputs: the principal, annual interest rate
• Output: value of the investment in 10 years
• Relationship between Inputs and Outputs: Value after one year is given by this formula: principal * (1 + apr). This needs to be done 10 times.

Example problem: Future Value

• Design:

  Print an introduction
  Input the amount of the principal (principal)
  Input the annual percentage rate (apr)
  Repeat 10 times:
    principal = principal * (1 + apr)
  Output the value of principal

Example problem: Future Value

• Code:

```cpp
#include <iostream>
#include <iomanip>

int main() {
    std::cout << std::fixed << std::setprecision(2);
    double principal, apr;
    // Print an introduction
    std::cout << "This program calculates the future ".
    std::cout << "value of a 10-year investment." << std::endl;
    // Input the amount of the principal (principal)
    std::cout << "Enter the initial principal: ");
    std::cin >> principal;
    // Input the annual percentage rate (apr)
    std::cout << "Enter the annual interest rate: ");
    std::cin >> apr;
    // Repeat 10 times:
    for (int i=1; i<=10; i++)
        // principal = principal * (1 + apr)
        principal = principal * (1 + apr);
    // Output the value of principal
    std::cout << "The value in 10 years is: ") << principal << std::endl;
} 
```