Today

- Conditional Branching
  - Logical Expressions
  - if and if-else statements
  - switch statement
  - goto statement

- Iteration Control
  - Loop Statements
    - for, while, do-while structures
    - break and continue

Relational Operators

- Take 2 expressions as operands
  - e.g., "a < 3" reads as "a is less than 3"
- Yield either the int value 0 (false) or the int value 1 (true).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal</td>
</tr>
</tbody>
</table>

Relational Operators (Examples)

- Valid Examples (assume a = 1 and b = 2)
  - a < b       → 1 (true)
  - a := b - 1  → 0 (false)
  - a + 1 <= b  → 1 (true)

- Invalid Examples
  - a <= b
  - a => b

- The Most Confused Case ("=" vs. "==")
  - "a = b" is an assignment expression
  - "a == b" is a test for equality
  - One of the most common problems the C programmer faces is mixing them up
Logical Operators

- The precedence of `&&` is higher than `||`
- Both operators are of lower precedence than all unary, arithmetic and relational operators.
  - i.e., `! > && >> ||`
- Expressions connected by `&&` or `||` are evaluated from left to right.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;&amp;</code></td>
<td>Logical AND</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>!</code></td>
<td>Logical Negation</td>
</tr>
</tbody>
</table>

Truth Table

| a     | b     | a && b | a || b | l
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>zero</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>zero</td>
<td>nonzero</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>nonzero</td>
<td>zero</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>nonzero</td>
<td>nonzero</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

We had covered this before while studying Boolean Algebra in Lecture 2.

Logical Operators (Examples)

- Negation Examples
  - `!5` → 0
  - `!!5` → 1
  - `!(6 < 7)` → 0
  - `!6 < 7` → 1
  - `!(3-4)` → 0

- Given declarations
  ```cpp
  int a = 3, b = 3, c = 3;
  double x = 0.0, y = 2.5;
  char ch = 'g';
  ```

- Expressions
  - `!(a < b) && c` → 1
  - `ch >= 'a' && ch <= 'z'` → 1
  - `x || a && b - 3` → 0
  - `a < b && x < y` → 0
  - `a < b || x < y` → 1
Short Circuit Evaluation

- For the expressions that contain the && or || operands, the expression process stops as soon as the outcome is already known.

- Suppose expr1 is zero.
  - expr1 && expr2 = 0 (expr2 will not be evaluated.)

- Suppose expr1 is nonzero.
  - expr1 || expr2 = 1 (expr2 will not be evaluated.)

The if Statement

- Syntax
  ```
  if (condition)
  statement;
  ```

- If the condition is true (nonzero)
  - the statement will be executed.

- If the condition is false (0)
  - the statement will not be executed.

The if Statement (Example)

- Suppose a program which writes a message if the student passes the class
  ```
  if (grade >= 60)
      printf("Passed!");
  ```

- Multiple statements may be grouped by putting them inside curly braces "{}".
  ```
  if (grade >= 60) {
      printf("Passed!");
      totalPassed++;
  }
  ```

Example: Write a program that prints the maximum of two numbers entered by the user

```c
#include <stdio.h>

int main () {
    int value1, value2, max = 0;
    printf("Enter two values:\n");
    scanf("%d %d", &value1, &value2);
    if (value1 > value2)
        max = value1;
    if (value1 <= value2)
        max = value2;
    printf("%d\n", max);
    return 0;
}
```
The if-else Statement

■ Syntax

```c
if (condition)
    statement 1;
else
    statement 2;
```

■ If the condition is true (nonzero)
  ▪ statement 1 will be executed.

■ If the condition is false (zero)
  ▪ statement 2 will be executed.

Example: (Re)Write a program that prints the maximum of two numbers entered by the user

```c
#include <stdio.h>

int main ( ) {
    int value1, value2, max = 0;
    printf("Enter two values:\n");
    scanf("%d %d", &value1, &value2);
    if (value1 > value2)
        max = value1;
    else
        max = value2;
    printf("%d
", max);
    return 0;
}
```

The if-else Statement (Example)

■ Suppose a program which writes a different message if the student passes or fails the class

```c
if (grade >= 60)
    printf("Passed!\n");
else
    printf("Failed!\n");
```

■ Multiple statements may be grouped by putting them inside curly braces "{}".

```c
if (grade >= 60) {
    printf("Passed!\n");
    totalPassed++;
} else {
    printf("Failed!\n");
    totalFailed++;
}
```

The Dangling else Problem

■ Consider the code below. Which if does the else belong to?

```c
if (grade < 60) {
    if (attendance == 100)
        printf("Passed!\n");
    else
        printf("Failed!\n");
```

■ Dangling else attaches to the nearest if. Always use curly braces to avoid ambiguous situations.

```c
if (grade < 60) {
    if (attendance == 100)
        printf("Passed!\n");
    else
        printf("Failed!\n");  // Do NOT forget to get the message here, too
}
```
The “?:” Construct

- **Syntax**
  
  \[(expression) \, ? \, value1 \, : \, value2\]

- The statement returns value1 if the expression evaluates to true. Returns value2 otherwise.

- Revisiting if-else statement example:
  
  ```c
  printf("%s\n", grade >= 60 ? "Passed!" : "Failed");
  ```

  Or, it could be written as:

  ```c
  grade >= 60 ? printf("Passed!") : printf("Failed");
  ```

Nested if-else Statements

- Tests for multiple cases by placing if...else selection statements inside if...else selection statement

- **Syntax**
  
  ```c
  if (condition 1)
    statement 1;
  else if (condition 2)
    statement 2;
  ... else if (condition n)
    statement n;
  else
    default statement;
  ```

  - Once condition i is met, rest of statements skipped
  - If no condition is met, default statement is executed

Nested if-else Statements (Example)

- Code segment for a simple calculator

  ```c
  if (operator == '+')
    result += value;
  else if (operator == '-')
    result -= value;
  else if (operator == '*')
    result *= value;
  else if (operator == '/')
    result /= value;
  else
    printf("Unknown operator!");
  ```

The switch Statement

- The **switch** statement evaluates the value of an expression and branches to one of the case labels.

- **Syntax**
  
  ```c
  switch (expression) {
    case constant 1:
      statement;
      break;
    case constant n:
      statement;
      break;
    default:
      statement;
      break;
  }
  ```

  - Duplicate labels are not allowed. The expression must evaluate an integer, character, or enumeration.
The switch Statement (Example)

- Revisiting the code segment for a simple calculator

```c
switch (operator) {
    case '+':
        result += value;
        break;
    case '-':
        result -= value;
        break;
    case '*':
        result *= value;
        break;
    case '/':
        result /= value;
        break;
    default:
        printf("Unknown operator!");
        break;
}
```

Dustier Corner of the switch Statement

- **break** statement exits the switch structure.
- If a **break** statement is not there, execution will continue with the next statement.

```c
switch (control) {
    case 0: printf("Reset
");
    case 1: printf("Initializing
");
        break;
    case 2:
        printf("Working
");
    }
```

- Because, it is not possible to determine if the program is supposed to fall through from case 0 to case 1, or if the programmer forgot to put in a **break** statement.
- **case 2** does not need a **break** as it is the last statement. But, put a **break** there anyways.

The switch Statement

```c
if (month==1) {
    printf("Jan.");
} else if (month==2) {
    printf("Feb.");
} else if (month==3) {
    printf("Mar.");
} else if (month==4) {
    printf("Apr.");
} else if (month==5) {
    printf("May");
} else {
    printf("Summer");
}
```

```c
switch(month) {
    case 1:
        printf("Jan.");
        break;
    case 2:
        printf("Feb.");
        break;
    ...
    case 5:
        printf("May");
        break;
    default:
        printf("Summer");
        break;
}
```

The goto Statement

- **Syntax**
  ```c
goto label;
```

- **Program flow** jumps to the **statement** right after the **label**

- **The goto** statement is covered here only for the sake of completeness.

- **Do NOT use it!**
  - It makes the logic of the program complex.
  - In modern programming, **goto** statement is considered a harmful construct and a bad programming practice.
  - any program can be perfectly written without the use of **goto** statement.
Next Week

- Conditional Branching
  - Logical Expressions
  - if and if-else statements
  - switch statement
  - goto statement

- Iteration Control
  - Loop Statements
    - for, while, do-while structures
  - break and continue