C++
Flow of Control

DOYEL PAL

Outline

- **Boolean Expressions**
- **Branching Mechanisms**
  - if-else
  - Nesting if-else
- **Loops**
  - While, do-while, for
  - Nesting loops

Boolean Expressions

- **Comparison Operators**
  - ==
  - !=
  - >, >=
  - <, <=
- **Logical Operators**
  - Logical AND (&&)
  - Logical OR (||)

Evaluating Boolean Expressions

- **Data type bool**
  - Returns true or false
  - true, false are predefined library consts

Evaluating Boolean Expressions: Truth Tables

NOT

Exp1	!(Exp1)
true	false
false	true

OR

(Exp 1)	(Exp 2)	(Exp 1) || (Exp 2)
true	true	true
true	false	true
false	true	true
false	false	false

AND

(Exp 1)	(Exp 2)	(Exp 1)&& (Exp 2)
true	true	true
true	false	false
false	true	false
false	false	false

Evaluate

1. The value of `count` is 0; `limit` is 10.

```
(count == 0)&&(limit < 20)
```

Answer: true

2. The value of count is 0; limit is 10.

Evaluate:

```
(count != 0)||(limit < 20)
```

Answer: true.

Branching Mechanisms

- **if-else statements**
  - Choice of two alternate statements based on condition expression
- **Formal syntax**

```
if (<boolean_expression>)
    <yes_statement>
else
    <no_statement>
```

if-else Statement Example

```
if (hrs > 40)
    grossPay = rate*40 + 1.5*rate*(hrs-40);
else
    grossPay = rate*hrs;
```

Compound/Block Statement

- Must use compound statement { }
  for multiples
  - Also called a "block" stmt
- Each block should have block statement
  - Even if just one statement
  - Enhances readability

Compound Statement Example

- Note indenting in this example

```
if (myScore > yourScore)
{
 cout << "I win!\n";
 wager = wager + 100;
}
else
{
 cout << "I wish these were golf
scores.\n";
 wager = 0;
}
```

Common Pitfalls

- Operator "=" vs. operator "=="
- One means "assignment" (=)
- One means "equality" (==)
  - VERY different in C++!
  - Example:
    if (x = 12) ←Note operator used! ×
        Do_Something
    else
        Do_Something_Else

The Optional else

- else clause is optional
  - If, in the false branch (else), you want "nothing" to happen, leave it out
  - Example:

```
if (sales >= minimum)
    salary = salary + bonus;
cout << "Salary = %" << salary;
```

  - Note: nothing to do for false condition, so there is no else clause!
  - Execution continues with cout statement

Nested Statements

- Can also contain another if-else stmt!
  - Example:

```
if (speed > 55)
    if (speed > 80)
        cout << "You're really
speeding!";
    else
        cout << "You're speeding.";
```

    - Note proper indenting!

Multiway if-else

- Not new, just different indenting
  - Syntax:

Multiway if–else Statement

SYNTAX

```
if (Boolean_Expression_1)
     Statement_1
else if (Boolean_Expression_2)
     Statement_2
              .
              .
              .
else if (Boolean_Expression_n)
     Statement_n
else
     Statement_For_All_Other_Possibilities
```

Multiway if-else Example

EXAMPLE

```
if ((temperature < -10) && (day == SUNDAY))
    cout << "Stay home.";
else if (temperature < -10) //and day != SUNDAY
    cout << "Stay home, but call work.";
else if (temperature <= 0) //and temperature >= -10
    cout << "Dress warm.";
else //temperature > 0
    cout << "Work hard and play hard.";
```

The Boolean expressions are checked in order until the first `true` Boolean expression is encountered, and then the corresponding statement is executed. If none of the Boolean expressions is *true*, then the *Statement_For_All_Other_Possibilities* is executed.

Conditional Operator

- Also called "ternary operator"
  - Allows embedded conditional in expression
  - Essentially "shorthand if-else" operator
  - Example:

```
if (n1 > n2)
    max = n1;
else
    max = n2;
```

  - Can be written:

```
max = (n1 > n2) ? n1 : n2;
```

    - "?" and ":" form this "ternary" operator

Loops

- 3 Types of loops in C++
  - while
    - Most flexible
    - No "restrictions"
  - do-while
    - Least flexible
    - Always executes loop body at least once
  - for
    - Natural "counting" loop

while Loops Syntax

Syntax for `while` and `do-while` Statements

A `while` STATEMENT WITH A SINGLE STATEMENT BODY

```
while (Boolean_Expression)
    Statement
```

A `while` STATEMENT WITH A MULTISTATEMENT BODY

```
while (Boolean_Expression)
{
    Statement_1
    Statement_2
      .
      .
      .
    Statement_Last
}
```

while Loop Example

- Consider:

```
count = 0;              // Initialization
while (count < 3)  // Loop Condition
{
 cout << "Hi ";       // Loop Body
 count++;             // Update
expression
}
```

do-while Loop Syntax

A do-while STATEMENT WITH A SINGLE-STATEMENT BODY

```
do
    Statement
while (Boolean_Expression);
```

A do-while STATEMENT WITH A MULTISTATEMENT BODY

```
do
{
    Statement_1
    Statement_2
      .
      .
      .
    Statement_Last
} while (Boolean_Expression);
```

Do not forget the final semicolon.

do-while Loop Example

```
count = 0;        // Initialization
do
{
  cout << "Hi ";  // Loop Body
  count++;        // Update expression
} while (count < 3);     // Loop Condition
```

- Loop body executes how many times?
- do-while loops always execute body at least once!

while vs. do-while

- Very similar, but…
  - One important difference
    - Issue is "WHEN" boolean expression is checked
    - while: checks BEFORE body is executed
    - do-while: checked AFTER body is executed
- After this difference, they're essentially identical!
- while is more common, due to it's ultimate "flexibility"

for Loop Syntax

```
for (Init_Action; Bool_Exp; Update_Action)
    Body_Statement
```

- Like if-else, Body_Statement can be a block statement
- Much more typical

for Loop Example

- for (count=0;count<3;count++)
  {
  cout << "Hi "; // Loop Body
  }
- How many times does loop body execute?
- Initialization, loop condition and update all
  "built into" the for-loop structure!

Loop Pitfalls: Misplaced ;

- Watch the misplaced ; (semicolon)
  - Example:

```
while (response != 0) ;←
{
    cout << "Enter val: ";
    cin >> response;
}
```

  - Notice the ";" after the while condition!
- Result here: INFINITE LOOP!

Loop Pitfalls: Infinite Loops

- Loop condition must evaluate to false at some iteration through loop
- If not → infinite loop.
- Example:

```
while (1)
{
    cout << "Hello ";
}
```

Each of the following has at least one error, either intent, or an error that may be caught by the compiler or both). What is the error? Assume all the variables you see are defined and initialized.

1.
```
for(int i = 0; i <10; i++);
    sum = sum +x;
```
2.
```
if (x > 0)
        x = 3
    else
        x =4;
```
3.
```
if(x = 0) x = MAXINT;
```
4.
```
if x > 0 x = 3;
```
5.
```
if (x>0) then x =3;
```

Nested Loops

- Any valid C++ statements can be inside body of loop
- This includes additional loop statements!
  Called "nested loops"
- Requires careful indenting:

```
for (outer=0; outer<5; outer++)
    for (inner=7; inner>2; inner--)
        cout << outer << inner;
```

Summary 1

- Boolean expressions
  - Similar to arithmetic → results in true or false
- C++ branching statements
  - if-else, switch
  - switch statement great for menus
- C++ loop statements
  - while
  - do-while
  - for

Summary 2

- do-while loops
  - Always execute their loop body at least once
- for-loop
  - A natural "counting" loop
- Loops can be exited early
  - break statement
  - continue statement
  - Usage restricted for style purposes

Program

Write a program with multiway if-else statements in which letter grades are assigned based a numeric grade based on this "ten point" scheme:

Give a user input numeric grade.

- if the numeric grade is not less than 90, the letter grade is an A,
- if the numeric grade is not less than 80, the letter grade is a B,
- if the numeric grade is not less than 70, the letter grade is C,
- if the numeric grade is not less than 60, the letter grade is D,
- otherwise the letter grade is F.