CS12 CH: If - Else If - Else

Chaining Conditional Statements

Mr. Gullo

October 2024

Learning Objectives

By the end of this lesson, you will be able to:

- Understand the structure and flow of `if-else` `if-else` statements

Learning Objectives

By the end of this lesson, you will be able to:

- Understand the structure and flow of `if-else if-else` statements
- Identify when to use `else if` for mutually exclusive conditions

Learning Objectives

By the end of this lesson, you will be able to:

- Understand the structure and flow of `if-else if-else` statements
- Identify when to use `else if` for mutually exclusive conditions
- Recognize and eliminate redundant conditions in conditional chains

Learning Objectives

By the end of this lesson, you will be able to:

- Understand the structure and flow of `if-else if-else` statements
- Identify when to use `else if` for mutually exclusive conditions
- Recognize and eliminate redundant conditions in conditional chains
- Apply `if-else if-else` statements to solve classification problems

Learning Objectives

By the end of this lesson, you will be able to:

- Understand the structure and flow of `if-else if-else` statements
- Identify when to use `else if` for mutually exclusive conditions
- Recognize and eliminate redundant conditions in conditional chains
- Apply `if-else if-else` statements to solve classification problems
- Debug common logic errors in multi-branch conditionals

Demo Programs for This Lesson

Programs to demonstrate live:

- `1elseif_skeleton.cpp` - Basic structure of else if chain (divisibility by 2 and 3)
- `2elseif_posNeg.cpp` - Classifying numbers as positive, negative, or zero
- `3elseif_2_3.cpp` - Same as skeleton (divisibility testing)
- `4elseif_bad.cpp` - Age classification with redundant conditions (teaching moment)

Location: `lessonPrograms/`

Note: These are for instructor demonstration only. Use to show proper else-if structure and common pitfalls.

Why Else If?

Problem: Sometimes we need to check multiple mutually exclusive conditions.

Why Else If?

Problem: Sometimes we need to check multiple mutually exclusive conditions.

Example Scenario:

- Is a number positive, negative, or zero?
- What age category does a person fall into?
- Which commission bracket applies to a sales amount?

Why Else If?

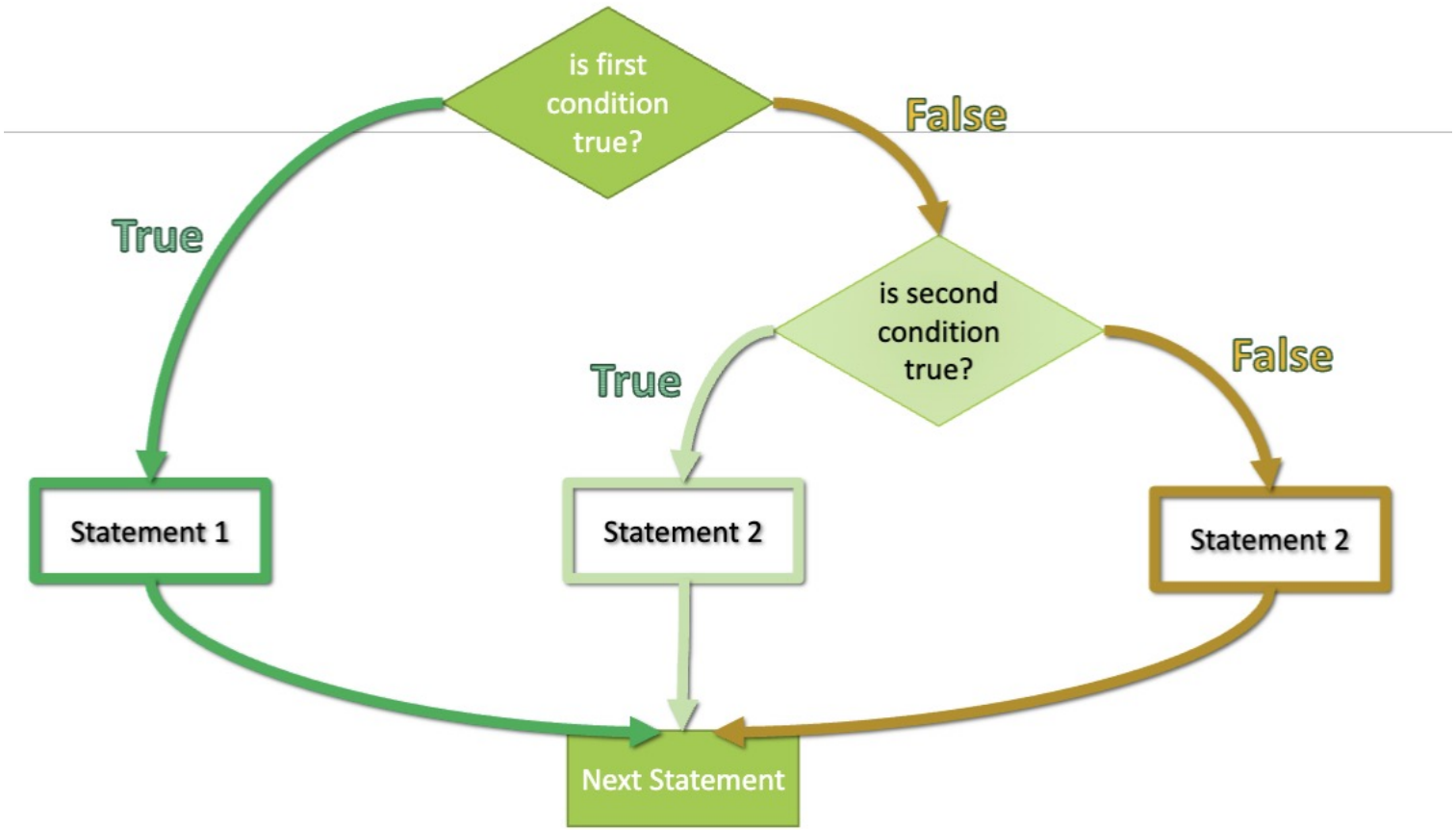
Problem: Sometimes we need to check multiple mutually exclusive conditions.

Example Scenario:

- Is a number positive, negative, or zero?
- What age category does a person fall into?
- Which commission bracket applies to a sales amount?

Key Insight: Only ONE of these conditions should execute, and we should stop checking once we find a match.

The If - Else If - Else Structure



Flow Logic:

1. Check first condition
2. If false, check second condition
3. If false, check third condition

Syntax Structure

General Form:

```
if (condition1) {
    // Statement 1
}
else if (condition2) {
    // Statement 2
}
else if (condition3) {
    // Statement 3
}
else {
    // Default statement
}
// Next statement (continues here)
```

Important: Only ONE block executes, then control moves to next statement.

Example 1: Positive, Negative, or Zero

Demo File: `2elseif_posNeg.cpp`

```
#include <iostream>
using namespace std;

int main() {
    int testNumber;

    cout << "Please enter an integer: ";
    cin >> testNumber;

    if(testNumber < 0)
        cout << testNumber << " is negative.\n";
    else if(testNumber > 0)
        cout << testNumber << " is positive\n";
    else
        cout << testNumber << " is zero.";

    return 0;
}
```

Example 2: Divisibility Testing

Demo File: `1elseif_skeleton.cpp`

```cpp
#include <iostream>
using namespace std;

int main() {
    int testNumber;

    cout << "Please enter an integer: ";
    cin >> testNumber;

    if(testNumber % 6 == 0)
        cout << testNumber << " is divisible by 2 and 3\n";
    else if(testNumber % 2 == 0)
        cout << testNumber << " is divisible by 2 but not 3\n";
    else if(testNumber % 3 == 0)
        cout << testNumber << " is divisible by 3 but not 2\n";
    else
        cout << testNumber << " is neither divisible by 3 nor 2\n";

    return 0;
}
```

Why Check Divisibility by 6 First?

Consider: What happens if we enter the number 12?

Why Check Divisibility by 6 First?

Consider: What happens if we enter the number 12?

- 12 is divisible by 2 ✓
- 12 is divisible by 3 ✓
- 12 is divisible by 6 ✓

Why Check Divisibility by 6 First?

Consider: What happens if we enter the number 12?

- 12 is divisible by 2 ✓
- 12 is divisible by 3 ✓
- 12 is divisible by 6 ✓

Order Matters!

- If we check `testNumber % 2 == 0` first, we'd say "divisible by 2 but not 3"
- This would be WRONG for 6, 12, 18, etc.
- Must check the MOST SPECIFIC condition first

Common Mistake: Redundant Conditions

Demo File: `4elseif_bad.cpp` (Why is this called "bad"?)

Common Mistake: Redundant Conditions

Demo File: `4elseif_bad.cpp` (Why is this called "bad"?)

```
if(userAge <= 1)
    cout << "The user is an infant\n";
else if(userAge >= 2 && userAge <= 12)
    cout << "The user is a child\n";
else if(userAge >= 13 && userAge <= 19)
    cout << "The user is an teenager\n";
else
    cout << "The user is an adult\n";
```

Common Mistake: Redundant Conditions

Demo File: `4elseif_bad.cpp` (Why is this called "bad"?)

```
if(userAge <= 1)
    cout << "The user is an infant\n";
else if(userAge >= 2 && userAge <= 12)
    cout << "The user is a child\n";
else if(userAge >= 13 && userAge <= 19)
    cout << "The user is an teenager\n";
else
    cout << "The user is an adult\n";
```

Question: Is `userAge >= 2` necessary on line 3?

Understanding Redundant Conditions

Analysis:

- First condition: `userAge <= 1`

Understanding Redundant Conditions

Analysis:

- First condition: `userAge <= 1`
- If this is FALSE, what do we know?

Understanding Redundant Conditions

Analysis:

- First condition: `userAge <= 1`
- If this is FALSE, what do we know?
- We know: `userAge > 1`, which means `userAge >= 2`

Understanding Redundant Conditions

Analysis:

- First condition: `userAge <= 1`
- If this is FALSE, what do we know?
- We know: `userAge > 1`, which means `userAge >= 2`
- So checking `userAge >= 2` is REDUNDANT

Understanding Redundant Conditions

Analysis:

- First condition: `userAge <= 1`
- If this is FALSE, what do we know?
- We know: `userAge > 1`, which means `userAge >= 2`
- So checking `userAge >= 2` is REDUNDANT

Key Principle: When an `else if` executes, ALL previous conditions are already known to be false. Use this information to simplify your conditions!

Improved Version

Better Code (No Redundancy):

```
if(userAge <= 1)
    cout << "The user is an infant\n";
else if(userAge <= 12)
    cout << "The user is a child\n";
else if(userAge <= 19)
    cout << "The user is an teenager\n";
else
    cout << "The user is an adult\n";
```

Improved Version

Better Code (No Redundancy):

```
if(userAge <= 1)
    cout << "The user is an infant\n";
else if(userAge <= 12)
    cout << "The user is a child\n";
else if(userAge <= 19)
    cout << "The user is an teenager\n";
else
    cout << "The user is an adult\n";
```

Why this works:

- Line 3: We know `userAge > 1`, so just check upper bound
- Line 5: We know `userAge > 12`, so just check upper bound
- Line 7: We know `userAge > 19`, so must be adult

Exercise 1: Sales Commission Calculator

Problem: Calculate commission based on total sales.

Sales Amount	**Commission Rate**
\$0 - \$10,000	5%
\$10,001 - \$50,000	8%
\$50,001 - \$100,000	10%
Over \$100,000	12%

Task: Write a program that:

- Prompts user for total sales
- Calculates income (commission)
- Outputs the result

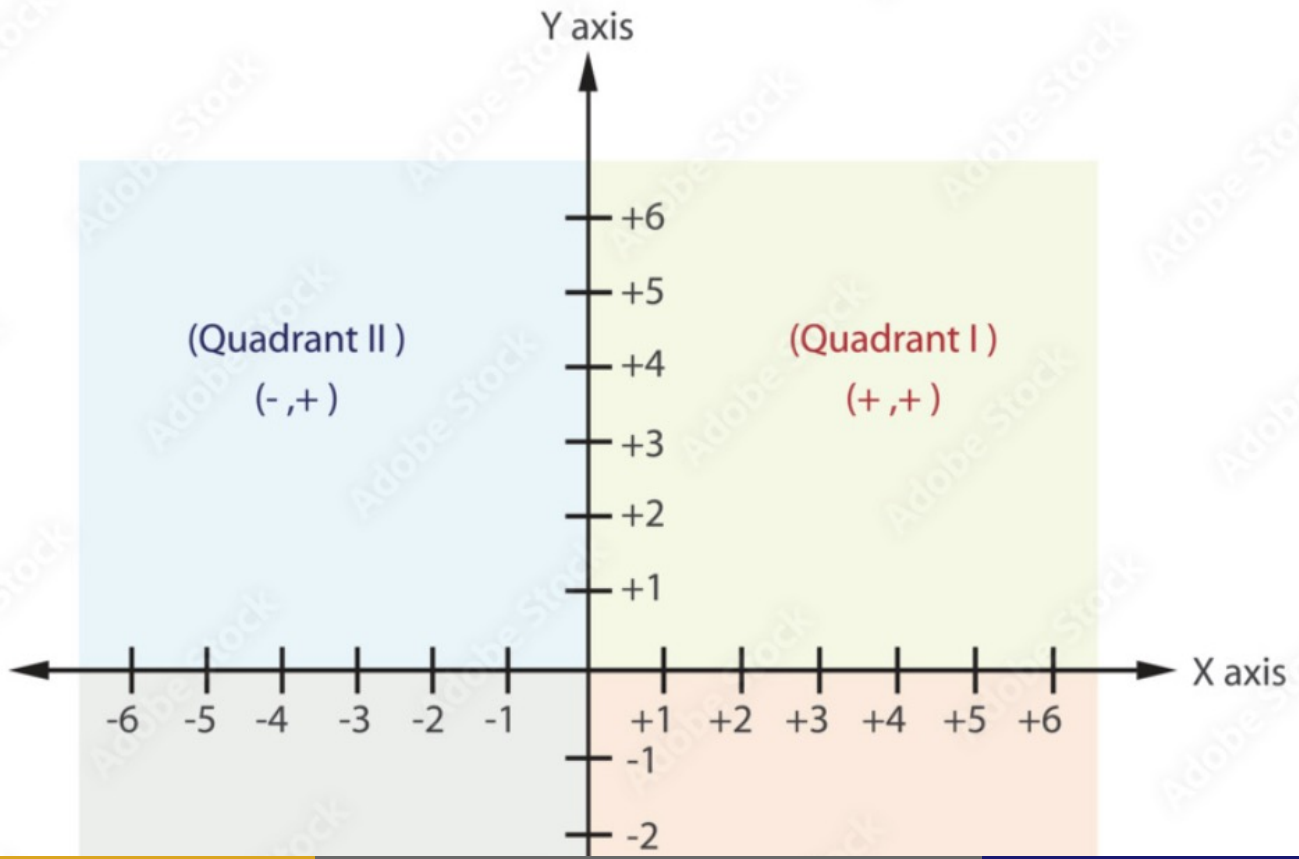
Exercise 1: Template with TODOs

```cpp
#include <iostream>
using namespace std;

int main() {
    double sales, commission;

    cout << "Enter total sales: $";
    cin >> sales;

    // TODO 1: Check if sales <= 10000, calculate 5% commission

    // TODO 2: Check if sales <= 50000, calculate 8% commission

    // TODO 3: Check if sales <= 100000, calculate 10% commission

    // TODO 4: For sales > 100000, calculate 12% commission

    // TODO 5: Output the commission with appropriate formatting

    return 0;
}
```

Exercise 2: Coordinate Classification

Problem: Given a point (x, y), determine its location:

- At the origin
- On the x-axis (not origin)
- On the y-axis (not origin)
- In quadrant 1, 2, 3, or 4



Exercise 2: Template with TODOs

```cpp
#include <iostream>
using namespace std;

int main() {
    double x, y;

    cout << "Enter x coordinate: ";
    cin >> x;
    cout << "Enter y coordinate: ";
    cin >> y;

    // TODO 1: Check if at origin (x==0 AND y==0)

    // TODO 2: Check if on x-axis (y==0, but not origin)

    // TODO 3: Check if on y-axis (x==0, but not origin)

    // TODO 4-7: Check for each quadrant

    return 0;
}
```

Common Mistakes to Avoid

1. **Wrong Order of Conditions**
   - Always check MORE SPECIFIC conditions first
   - Example: Check "divisible by 6" before "divisible by 2"

Common Mistakes to Avoid

1. **Wrong Order of Conditions**
   - Always check MORE SPECIFIC conditions first
   - Example: Check "divisible by 6" before "divisible by 2"
2. **Redundant Conditions**
   - Remember: `else if` means all previous conditions were false
   - Use this knowledge to simplify your conditions

Common Mistakes to Avoid

1. **Wrong Order of Conditions**
   - Always check MORE SPECIFIC conditions first
   - Example: Check "divisible by 6" before "divisible by 2"

2. **Redundant Conditions**
   - Remember: `else if` means all previous conditions were false
   - Use this knowledge to simplify your conditions

3. **Forgetting the Final Else**
   - Final `else` catches all remaining cases
   - Prevents unexpected behavior for edge cases

Debugging Strategy

When your else-if chain doesn't work:

1. Test each condition boundary
2. Print which condition matched
3. Check for overlapping conditions
4. Verify order is from specific to general

Debugging Strategy

When your else-if chain doesn't work:

1. Test each condition boundary
2. Print which condition matched
3. Check for overlapping conditions
4. Verify order is from specific to general

Example Debug Output:

```
if(condition1) {
    cout << "Branch 1 executed\n";
    // rest of code
}
else if(condition2) {
    cout << "Branch 2 executed\n";
    // rest of code
}
```

Homework Assignment

Complete the following exercises:

1. Sales Commission Calculator (Exercise 1)
2. Coordinate Classification (Exercise 2)
3. Grade Letter Assignment:
   - A: 90-100
   - B: 80-89
   - C: 70-79
   - D: 60-69
   - F: Below 60

Submission Format:

- `firstnameLastname_elseif.cpp`
- Include all three exercises in separate functions
- Test with multiple inputs
- Submit via Schoology by due date

Assessment Resources

Practice Quiz Available:

- 12 multiple-choice questions on if-else if-else
- Located in `Relational Expression THW` folder
- Focus areas:
  - Understanding flow control
  - Identifying redundant conditions
  - Determining correct output
  - Order of condition checking

Reminder: Practice makes perfect! Try writing your own classification problems.

Key Takeaways

- `if-else if-else` chains handle mutually exclusive conditions

Key Takeaways

- `if-else if-else` chains handle mutually exclusive conditions
- Only ONE branch executes, then control skips to next statement

Key Takeaways

- `if-else if-else` chains handle mutually exclusive conditions
- Only ONE branch executes, then control skips to next statement
- Order matters: check MORE SPECIFIC conditions first

Key Takeaways

- `if-else if-else` chains handle mutually exclusive conditions
- Only ONE branch executes, then control skips to next statement
- Order matters: check MORE SPECIFIC conditions first
- Eliminate redundant conditions using logical implications

Key Takeaways

- `if-else if-else` chains handle mutually exclusive conditions
- Only ONE branch executes, then control skips to next statement
- Order matters: check MORE SPECIFIC conditions first
- Eliminate redundant conditions using logical implications
- Always include final `else` to catch remaining cases

Key Takeaways

- `if-else if-else` chains handle mutually exclusive conditions
- Only ONE branch executes, then control skips to next statement
- Order matters: check MORE SPECIFIC conditions first
- Eliminate redundant conditions using logical implications
- Always include final `else` to catch remaining cases
- Test boundary values and edge cases thoroughly

Next Lesson: Switch statements and loops for more complex control flow!