Selection Structure

Mirza Mohammad Lutfe Elahi

Outline

- Control Structures
- Conditions, Relational, and Logic Operators
- The **`if`** Statement and Flowchart
- **`if`** with Compound Statements
- Nested **`if`** statements
- The **`switch`** Statement
- Operator Precedence, Complementing a Condition
- Common Programming Errors

Control Structure

- Control structure
  - Control the flow of execution in a program or a function
- Three kinds of control structures
  - **Sequence** (Compound Statement)
  - **Selection** (`if` and `switch` Statements)
  - **Repetition** [Chapter 5]
- **Selection control structure**
  - Chooses among alternative program statements

Compound Statement

- A group of statements bracketed by { and }
- Executed Sequentially
- A function body consists of a compound statement

```
{
    statement1 ;
    statement2 ;
    . . .
    statementn ;
}
```

Compound Statement Specifies Sequential Execution

Conditions

- **Condition**
  - An expression that evaluates to **false (0)** or **true (1)**
- Conditions are used in **`if`** statements, such as:

```
if (a >= b)
  printf("a is greater or equal to b");
else
  printf("a is less than b");
```

- The condition in the above example: **`(a >= b)`**

Relational and Equality Operators

Operator	Meaning	Type
<	less than	relational
>	greater than	relational
<=	less than or equal to	relational
>=	greater than or equal to	relational
==	equal to	equality
!=	not equal to	equality

Evaluate to either **false (0)** or **true (1)**

Relational and Equality Operators

x	i	MAX	y	item	mean	ch	num
-5	1024	1024	7	5.5	7.2	'M'	999

Operator	Condition	Value
<=	x <= 0	true (1)
<	i < MAX	false (0)
>=	x >= y	false (0)
>	item > mean	false (0)
==	ch == 'M'	true (1)
!=	num != MAX	true (1)

Logical Operators

- Three Logical Operators

  && logical AND

  || logical OR

  ! logical NOT

- Truth Table for logical operators

A	B	(A && B)	(A || B)	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Logical Expression

- Logical Expression
  - Condition that uses one or more logical operators

salary	children	temperature	humidity	n
1050	6	38.2	0.85	101

Logical Expression	Value
`salary < 1000 || children > 4`	`true (1)`
`temperature > 35.0 && humidity > 0.90`	`false (0)`
`n >= 0 && n <= 100`	`false (0)`
`!(n >= 0 && n <= 100)`	`true (1)`

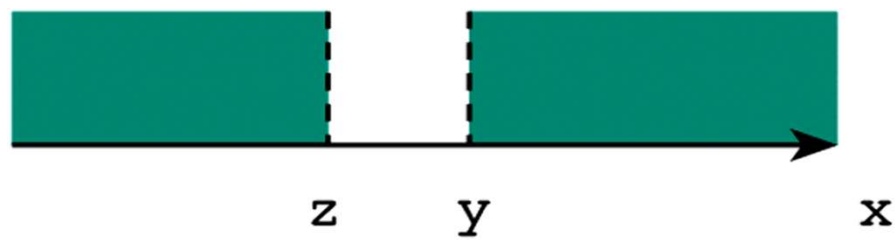
Comparing Characters

- We can also compare characters in C
  - Using the relational and equality operators

Expression	Value
`'9' >= '0'`	`1 (true)`
`'a' < 'e'`	`1 (true)`
`'B' <= 'A'`	`0 (false)`
`'Z' == 'z'`	`0 (false)`
`'A' <= 'a'`	`1 (true)`
`ch >= 'a' && ch <= 'z'`	`ch is lowercase?`

English Conditions as C Expression

English Condition	Logical Expression
x and **y** are greater than **z**	`x > z && y > z`
x is equal to **1** or **3**	`x == 1 || x == 3`
x is in the range **min** to **max**	`x >= min && x <= max`
x is outside the range **z** to **y**	`x < z || x > y`



`if` Statement (One Alternative)

```
if (condition) statement_T ;
```

if **condition** evaluates to **true** then **statement$_T$** is executed; Otherwise, **statement$_T$** is skipped

Example:

```
if (x != 0.0)
  product = product * x ;
```

`if` Statement (Two Alternative)

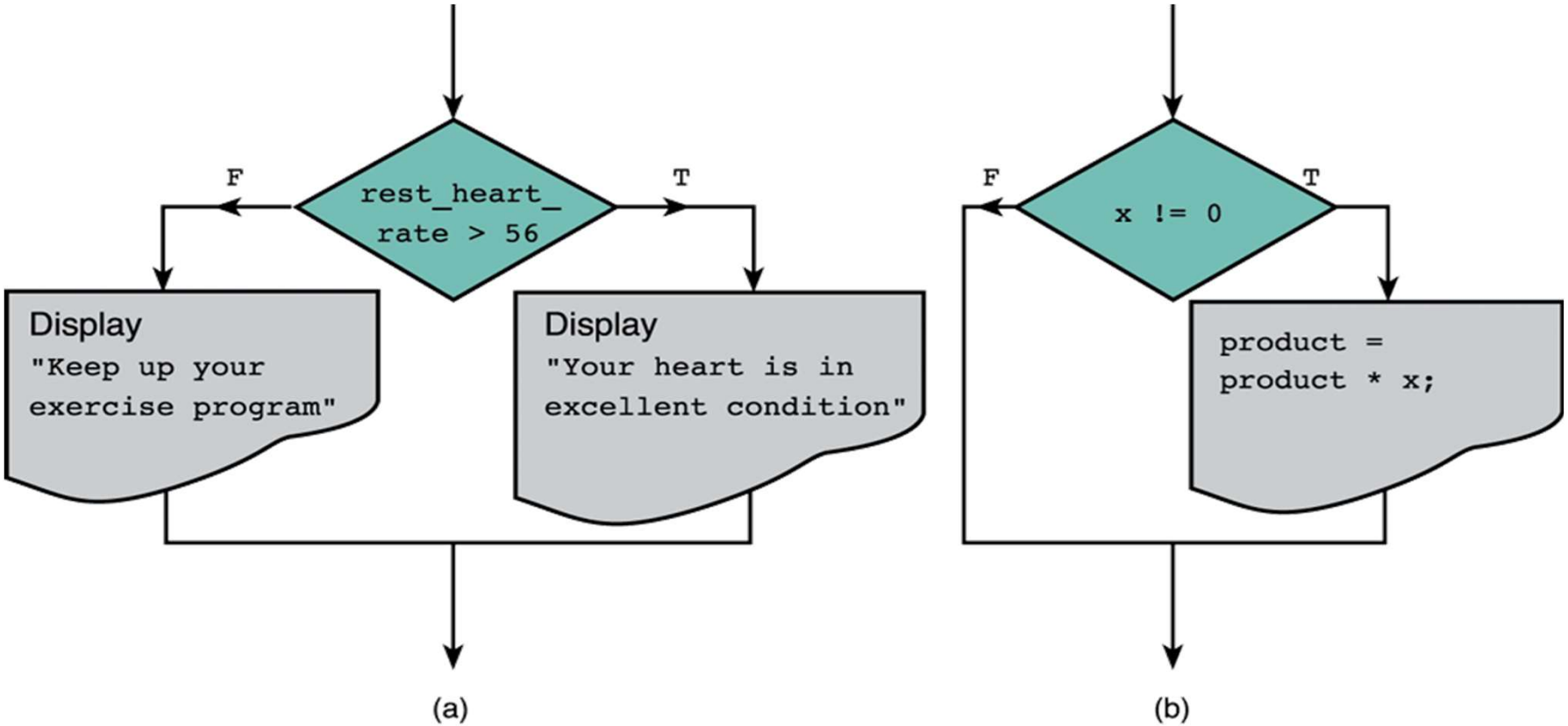
```
if (condition) statement_T ;
else statement_F ;
```

if **condition** evaluates to **true** then **statement$_T$** is executed and **statement$_F$** is skipped; Otherwise, **statement$_T$** is skipped and **statement$_F$** is executed

Example:

```
if (x >= 0.0) printf("Positive");
else printf("Negative");
```

Flowcharts of `if` Statements



Two Alternatives
if-else statement

One Alternative
if statement

`if` with Compound Statements

```
if (ch >= 'A' && ch <= 'Z') {
  printf("Letter '%c' is Uppercase\n", ch);
  ch = ch – 'A' + 'a';
  printf("Converted to lowercase '%c'\n", ch);
}
else {
  printf("'%c' is not Uppercase letter\n", ch);
  printf("No conversion is done\n");
}
```

Hand Tracing an `if` Statement

```
if (x > y) {  /* switch x and y  */
  temp = x;   /* save x in temp  */
  x = y;      /* x becomes y     */
  y = temp;   /* y becomes old x */
}
```

if statement	x	y	temp	Effect
	12.5	5.0	?	
if (x>y) {				12.5>5.0 is true
temp = x ;			12.5	Store old x in temp
x = y ;	5.0			Store y in x
y = temp ;		12.5		Store old x in y

Nested `if` Statements

- Nested **`if`** statement
  - **`if`** statement inside another **`if`** statement
  - Program decisions with **multiple alternatives**
- Example

```
if (x > 0)
   num_pos = num_pos + 1;
else
   if (x < 0)
      num_neg = num_neg + 1;
   else /* x equals 0 */
      num_zero = num_zero + 1;
```

Multiple-Alternatives Decision Form

- The conditions are evaluated in sequence until a true condition is reached
- If a condition is true, the statement following it is executed, and the rest is skipped

```
if (x > 0)
   num_pos = num_pos + 1;
else if (x < 0)
   num_neg = num_neg + 1;
else /* x equals 0 */
   num_zero = num_zero + 1;
```

More Readable

Sequence of `if` Statement

- All conditions are always tested (none is skipped)
- Less efficient than nested **`if`** for alternative decisions

```
if (x > 0)
    num_pos = num_pos + 1;
if (x < 0)
    num_neg = num_neg + 1;
if (x == 0)
    num_zero = num_zero + 1;
```

Less Efficient than nested `if`

Implementing a Decision Table

Use a multiple-alternative **`if`** statement to implement a decision table that describes several alternatives

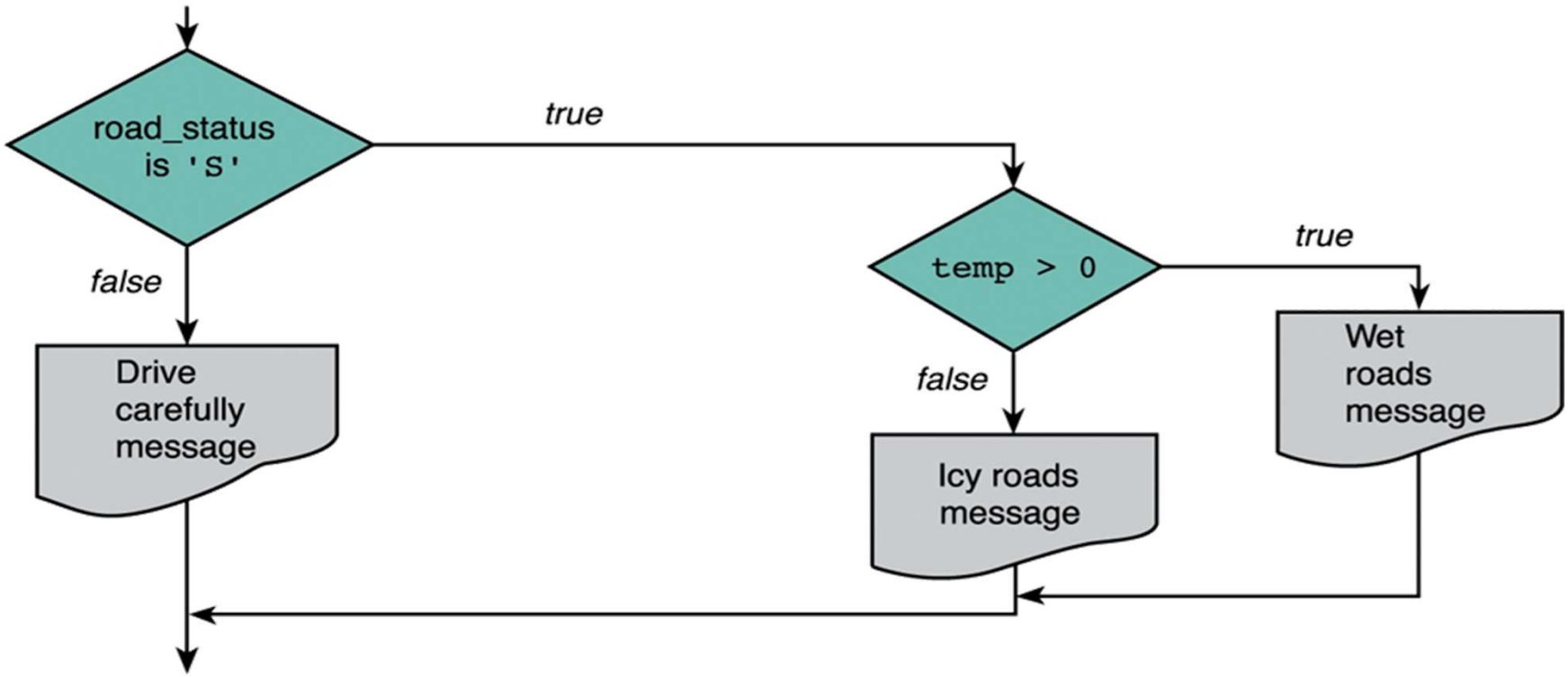
Salary Range ($)	Base Tax	Rate
Salary < $15,000	$0	15%
$15,000 ≤ Salary < $30,000	$2,250	18%
$30,000 ≤ Salary < $50,000	$4,950	22%
$50,000 ≤ Salary < $80,000	$9,350	27%
Salary ≥ $80,000	$17,450	33%

Computing the Tax from a Table

```
if (salary < 15000)
  tax = 0.15*salary;
else if (salary < 30000)
  tax = 2250 + (salary – 15000)*0.18;
else if (salary < 50000)
  tax = 4950 + (salary – 30000)*0.22;
else if (salary < 80000)
  tax = 9350 + (salary – 50000)*0.27;
else
  tax = 17450 + (salary – 80000)*0.33;
```

Road Sign Decision

- You are writing a program to control the warning signs at the exists of major tunnels.



Road Sign Nested `if` Statement

```
if (road_status == 'S')
   if (temp > 0) {
      printf("Wet roads ahead\n");
      printf("Stopping time = 10 minutes\n");
   }
   else {
      printf("Icy roads ahead\n");
   printf("Stopping time = 20 minutes\n");
   }
else
   printf("Drive carefully!\n");
```

C associates **else** with the most recent incomplete **if**

The `switch` Statement

- Can be used to select one of several alternatives
- Based on the value of a variable or simple expression
- Variable or expression may be of type **`int`** or **`char`**
- But not of type **`double`**
- **Example:** Simple Calculator

User Input	Operation
`'+'`	`result = a + b;`
`'-'`	`result = a - b;`
`'*'`	`result = a * b;`
`'/'`	`result = a / b;`

Example of `switch` Statement

```
switch (op) {          // op must be of type char
  case '+':
    result = a + b;
    break;
  case '-':
    result = a - b;
    break;
  case '*':
    result = a * b;
    break;
  case '/':
    result = a / b;
    break;
  default:
    printf("Error: unknown operation %c\n", op);
    return;            // to terminate the function
}
```

Explanation of `switch` Statement

- It takes the value of the character `op` and compares it to each of the cases in a top down approach.
- It stops after it finds the first `case` that is equal to the value of the variable `op`.
- It then starts to execute each line following the matching case till it finds a `break` statement.
- If no case is equal to the value of `op`, then the `default` case is executed.
- `default` is **optional**. If no other case is equal to the value of the *controlling expression* and there is no default case, the entire switch body is skipped.

More About The `switch` Statement

- One or more C statements may follow a **`case`** label.
- You do not need to enclose multiple statements in curly brackets after a **`case`** label.
- You cannot use a string as a **`case`** label.

  `case "Add":` is not allowed
- Do not forget **`break`** at the end of each alternative.
  - If the **`break`** statement is omitted then execution falls through into the next alternative.
- Do not forget the { } of the **`switch`** statement.

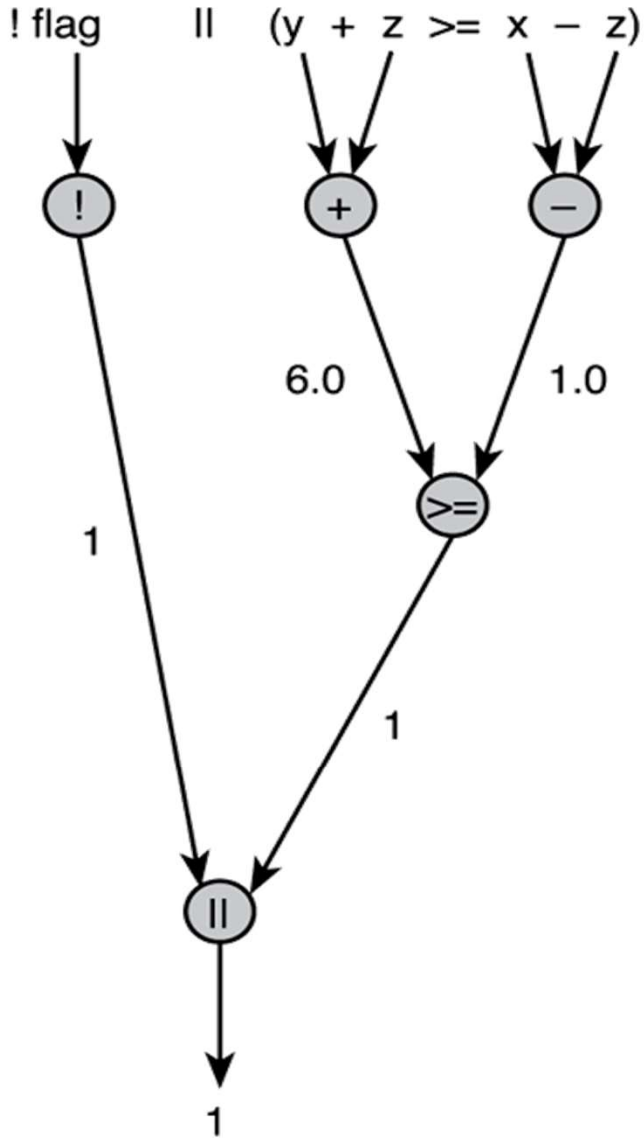
Nested `if` vs `switch`

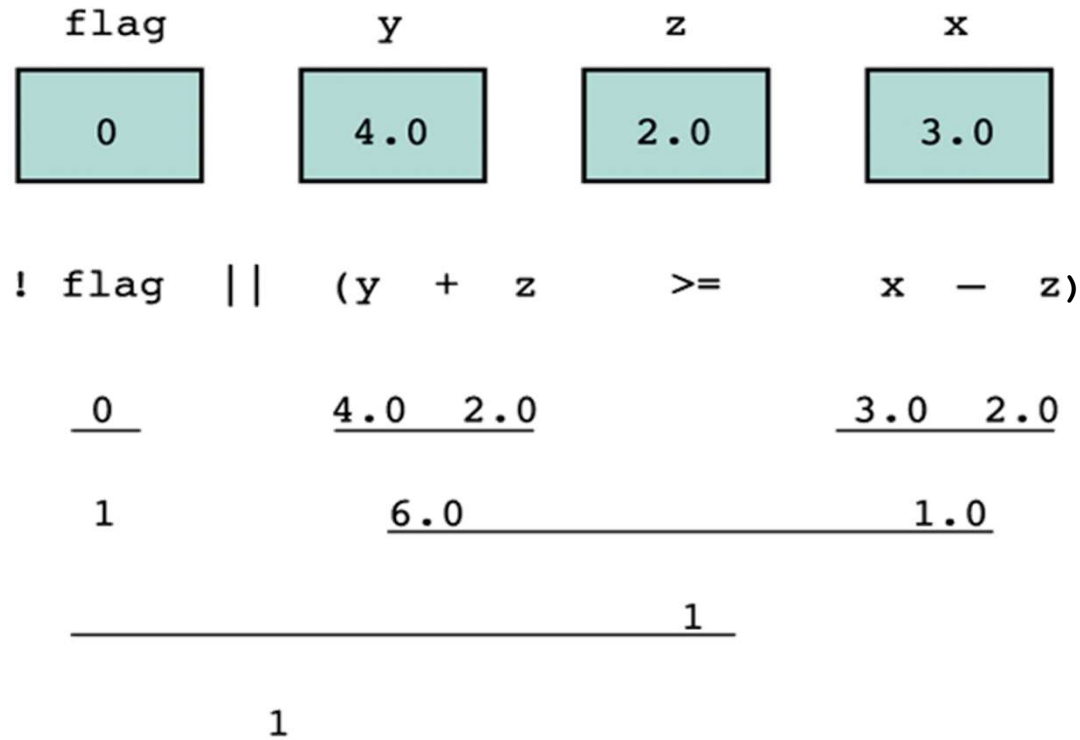
- Nested `if` statements
  - More general than a `switch` statement
  - Can implement any multiple-alternative decision
  - Can be used to check ranges of values
  - Can be used to compare double values
- `switch` statement
  - Syntax is more readable
  - Implemented more efficiently in machine language
  - Use `switch` whenever there are few `case` labels
  - Use `default` for values outside the set of case labels

Operator Precedence

Operator	Precedence
function calls	highest
! + - & (unary operators)	
* / %	
+ -	
< <= >= >	
== !=	
&& (logical AND)	
|| (logical OR)	
= (assignment operator)	lowest

Example Tree, Step-by-Step Evaluation





Short-Circuit Evaluation

- Stopping the evaluation of a logical expression as soon as its value can be determined
- Logical-OR expression of the form `(a || b)`
  - If `a` is **true** then `(a || b)` must be **true**, regardless of `b`
  - No need to evaluate `b`
  - However, if `a` is **false** then we should evaluate `b`
- Logical-AND expression of the form `(a && b)`
  - If `a` is **false** then `(a && b)` must be **false**, regardless of `b`
  - No need to evaluate `b`
  - However, if `a` is **true** then we should evaluate `b`
- Can be used to prevent division by zero

```
(divisor != 0  &&  x / divisor > 5)
```

Logical Assignment

- Use assignment to set `int` variables to **false** or **true**
- The **false** value is **zero**
- C accepts any **non-zero value** **as** **true**

Examples of Logical Assignment

```
senior_citizen = (age >= 65);
even = (n%2 == 0);
uppercase = (ch >= 'A' && ch <= 'Z');
lowercase = (ch >= 'a' && ch <= 'z');
is_letter = (uppercase || lowercase);
```

Complementing a Condition

- **DeMorgan's Theorem**

```
!(expr1 && expr2) == (!expr1 || !expr2)
!(expr1 || expr2) == (!expr1 && !expr2)
```

Example	Equivalent Expression
`!(item == 5)`	`item != 5`
`!(age >= 65)`	`age < 65`
`!(n > 0 && n < 10)`	`n <= 0 || n >= 10`
`!(x == 1 || x == 3)`	`x != 1 && x != 3`
`!(x>y && (c=='Y' || c=='y'))`	`(x<=y) || (c!='Y' && c!='y')`

Common Programming Errors

- **Do Not write:** `if (0 <= x <= 4)`
  - `0 <= x` is either **false (0)** or **true (1)**
  - Then, **false(0) or true(1) are always `<= 4`**
  - Therefore, `(0 <= x <= 4)` **is always true**
- **Instead, write:** `if (0 <= x && x <= 4)`
- **Do Not write:** `if (x = 10)`
  - `=` is the **assignment operator**
  - **x becomes 10** which is **non-zero (true)**
  - `if (x = 10)` **is always true**
- **Instead, write:** `if (x == 10)`

More Common Errors

- In `if` statements:
  - Don't forget to parenthesize the `if (condition)`
  - Don't forget `{` and `}` in `if` with compound statements
- Correct pairings of `if` and `else` statements:
  - **C matches `else` with the closest unmatched `if`**
- In `switch` statements:
  - Make sure the controlling expression and case labels are of the same permitted type (`int` or `char`)
  - Remember to include the `default` case
  - Don't forget `{` and `}` for the `switch` statement
  - Don't forget the `break` at the end of each case