



ICS-103

Computer Programming in C

Chapter 4:

Selection Structures

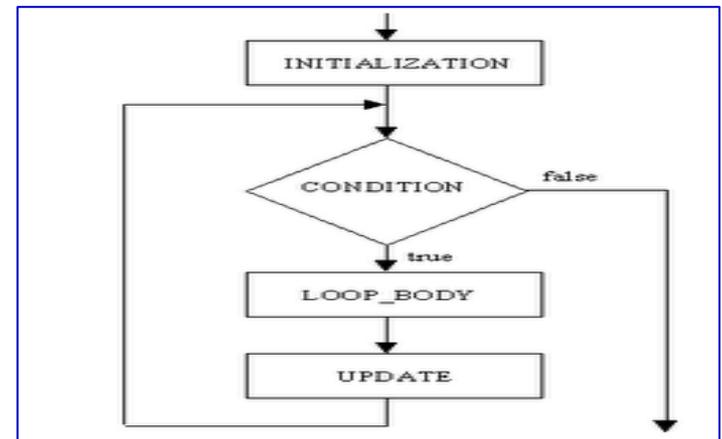
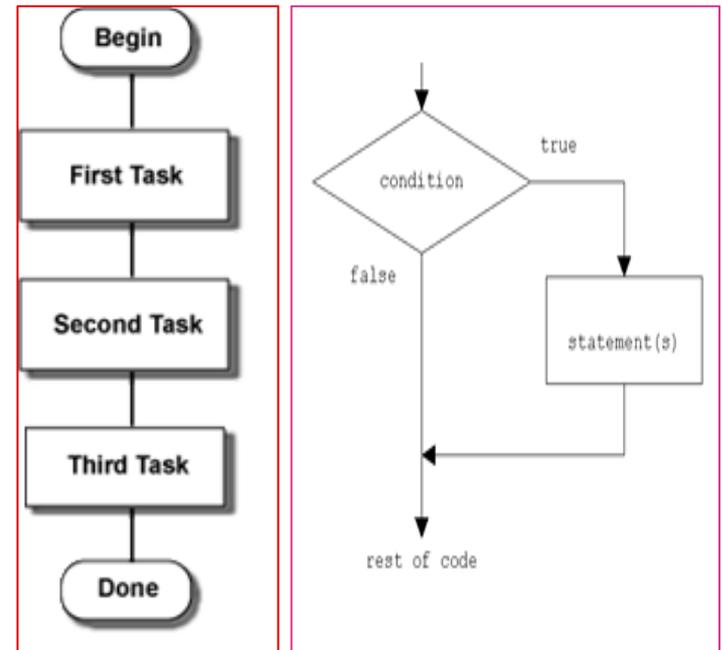
Dr. Tarek Ahmed Helmy El-Basuny

Outline of Ch. 4 Topics

- ❑ Control structure statements in C are either:
 - Sequential statements are executed in the order they are specified in the program.
 - Selection statements cause the program control to be transferred to a specific flow based upon whether a certain condition is true or not.
 - Repetition statements allow you to specify that some statements can be repeated while some condition remains true.
- ❑ Compound statement: we can group multiple C statements into a single statement.
- ❑ Conditional, Relational, and Logical Operators
- ❑ The if statement and its flowchart
- ❑ The if statement with compound statements
- ❑ Nested if statements
- ❑ The switch statement
- ❑ Operator Priority in C
- ❑ Complementing a condition
- ❑ Common Programming Errors

Control Structure Statements

- ❑ In the programs written so far, C statements executed sequentially based on their order within the body of the main function.
- ❑ In fact, we can change the flow of control by using **control structure statements**.
- ❑ **Control structure statement:**
 - A statement that is used to **control** the **flow** of execution in a program or a function.
- ❑ **Three kinds of control structures:**
 - **Sequential:** Sequential execution of statements (i.e. **Compound Statement**)
 - **Selection:** Used for decisions, branching or choosing between 2 or more alternative paths (i.e. **if** and **switch** Statements)
 - **Repetition:** Used for looping; repeating a piece of code multiple times [i.e. **while**, **for loop**, **do...while**, will be discussed in Chapter 5].



Compound Statement

- ❑ A **Compound statement** consists of several individual statements enclosed within a pair of braces { }. The individual statements may be: assignment, arithmetic expression statement, control statements, compound statement, etc.
- ❑ A compound statement does not end with a semicolon.
- ❑ Usually, a function body consists of a compound statement.
- ❑ **Accessibility of variables inside and outside of compound statement:**
- ❑ A variable which is declared **outside** the compound statements, it is accessible both **inside** and **outside** the compound statements.
- ❑ A variable which is declared **inside** the compound statements, it is not accessible **outside** the compound statements.
- ❑ **It is possible to declare a variable with the same name both inside and outside the compound statements, but it is not recommended.**
- ❑ The statements in the compound statement execute sequentially.

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



Example of a compound statement

```
int num = 10 ;  
if(num > 0)  
{  
    printf ("\nNumber is Positive");  
    printf ("\nThis is an Example of Compound Statement");  
}
```

Conditions

- ❑ The condition is an expression that evaluates to either **false (0)** or **true (1)**.
- ❑ **Conditional statements** cause dynamic flow of execution of the same program.
 - That means, each time the program runs, it may have different flow of execution based on certain condition being **true** or **false**.
- ❑ Conditions are used in **if** statements, as following:

```
if (a >= b)  
    printf("a is greater or equal to b");  
  
else  
    printf("a is less than b");
```
- ❑ The condition in the above example is: **(a >= b)**

Relational and Equality Operators

- The C language provides four relational and two equality operators for comparing the values of expressions as shown next.

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

- Relational operators in C evaluate to either false (0) or true (1).
- The equality operator == checks if the values of two operands are equal or not. **If they are equal**, then the condition becomes true.
- The non equality operator != checks if the values of two operands are equal or not. **If they are not equal**, then the condition becomes true.

Examples of 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
\leq	$x \leq 0$	true (1)
$<$	$i < \text{MAX}$	false (0)
\geq	$x \geq y$	false (0)
$>$	$\text{item} > \text{mean}$	false (0)
$==$	$\text{ch} == \text{'M'}$	true (1)
$!=$	$\text{num} != \text{MAX}$	true (1)

Logical Operators

❑ Three Logical Operators are:

&& 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

Example of Logical Expressions

□ Logical Expression

⇒ The 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)

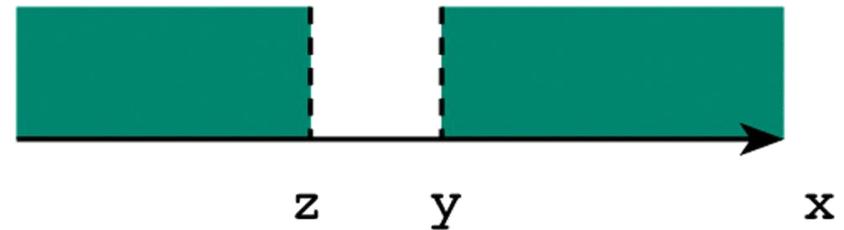
Comparing Characters

- We can also compare characters in **C** based on their **ASCII** values.
 - 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 Logical Expressions

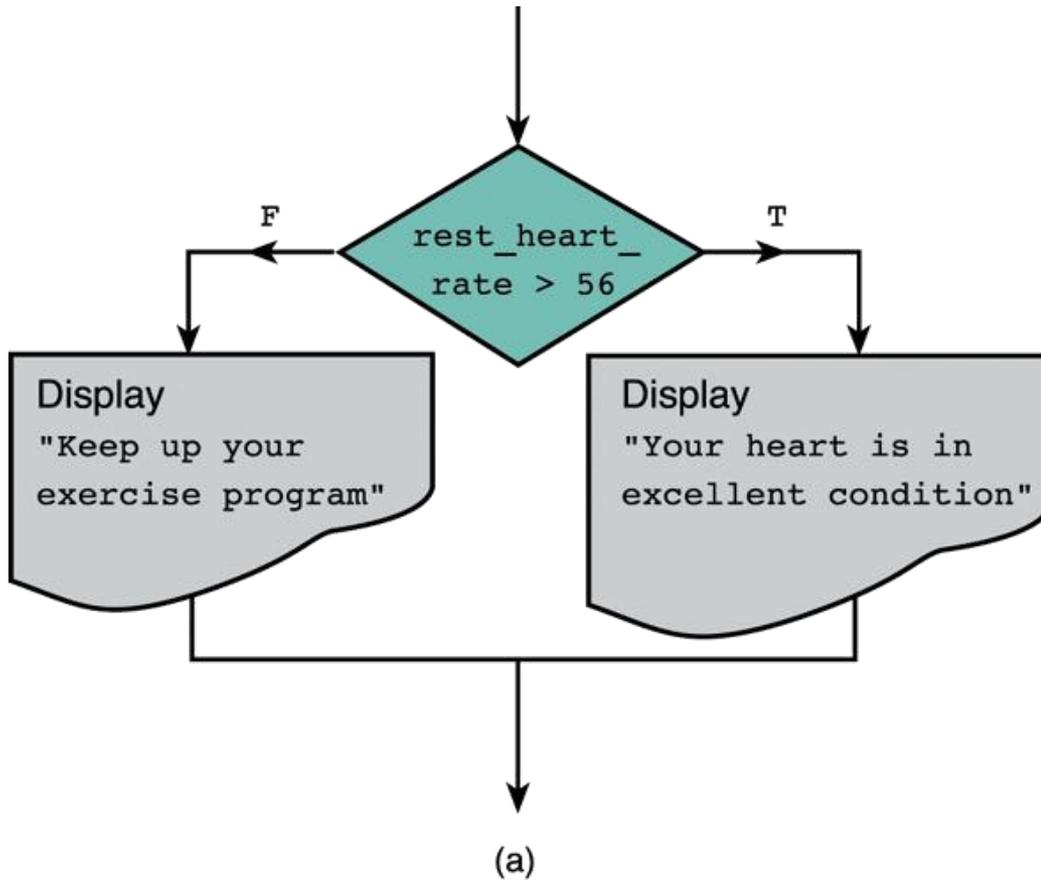
English Condition	Logical Expression
x and y are greater than z	$x > z \ \&\& \ y > z$
x is equal to 1 or 3	$x == 1 \ \ \ \ x == 3$
x is in the range min to max	$x \geq \text{min} \ \&\& \ x \leq \text{max}$
x is outside the range z to y	$x < z \ \ \ \ x > y$



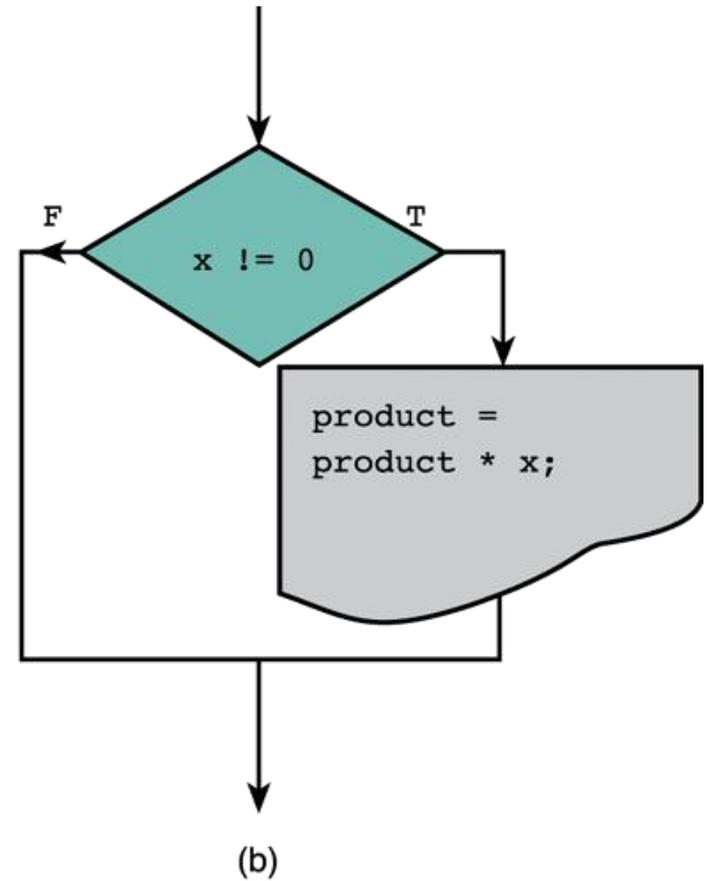
English Conditions as Logical Expressions

English Condition	Logical Expression
Hours worked is over 40	(hours > 40.0)
$0 \leq x \leq 10$	((x >= 0) && (x <= 10))
Value is not negative	(value >= 0)
Sum does not equal 100	(sum != 100)
The character must be between A & F inclusive	((ch >= 'A') && (ch <= 'F'))
Any value over 100.00 receives a discount	(value > 100.00)
Two numbers, a and b, differ by at least 5	(abs(a - b) >= 5)
Value does not equal -1	(value != -1)
Answer is either y or Y	((answer= ='y') (answer=='Y'))
The value is over 50 or under 25	((value > 50) (value < 25))
The number is even	((number % 2) == 0)

Flowcharts of **if** Statements



Two Alternatives
if-else statement



One Alternative
if statement

if Statement (One Alternative)

```
if ( statement is TRUE )
```

Execute this line of code;

Or it can be:

```
if (condition) statementT;
```

⇒ if condition evaluates to **true** then statement_T is executed;
Otherwise, statement_T is skipped.

□ Example 1:

```
if (x != 0.0)
```

```
product = product * x ;
```

□ Example 2:

```
if ( 5 < 10 )
```

```
printf( "Five is now less than ten, that's a big surprise" );
```

if Statement (Two Alternatives)

```
if (condition) statementT;
```

```
else statementF;
```

- ⇒ if condition evaluates to **true** then `statementT` is executed and `statementF` is skipped;
- ⇒ Otherwise, `statementT` is skipped and `statementF` is executed

Example 1:

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

```
else printf("Negative");
```

Example 2:

```
if ( age < 100 ) { /*If the age is less than 100*/
```

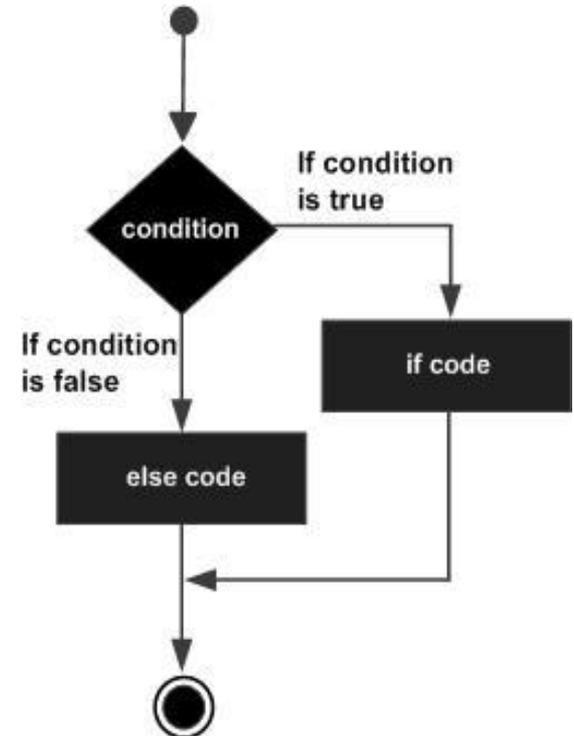
```
printf ("You are pretty young!\n" );
```

```
}
```

```
Else { /* the age is more than or equal*/
```

```
printf( "You are old\n" );
```

```
}
```



The if/else Selection Structure

- We may use **compound statement** (Set of statements within a pair of braces) after if or else part.

⇒ Example:

```
if ( grade >= 60 )
{
    printf( "Passed.\n" );
    printf( "You have done the job well.\n" );
}
else
{
    printf( "Failed.\n" );
    printf( "You must take this course again.\n" );
}
```

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");  
}
```

Swap the values of two variables using if Statement

```
/* to swap the values of two variables x and y*/  
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-else Statement

```
if (testExpression1) {  
    // statements to be executed if testExpression1 is true }  
else if(testExpression2) {  
    // statements to be executed if testExpression1 is false and  
    testExpression2 is true }  
else if (testExpression 3) {  
    // statements to be executed if testExpression1 and  
    testExpression2 are false and testExpression3 is true  
} ..  
else {  
    // statements to be executed if all test expressions are false  
}
```

Nested if Statements

❑ Nested if statement

- if statement inside another if statement
- Program decisions with multiple alternatives

❑ **Example:** The value of x is going to be tested. If it is positive then add 1 to the positive counter or if it is negative then add 1 to the negative counter otherwise it is zero and then add 1 to the zero counter.

```
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;
```

• Note:

- Since only one condition is true.
- That means only one action will be executed.
- The rest of actions will be skipped.

Multiple-Alternative 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** Statements

- The previous example can be written in the following way but.
 - ⇒ All conditions are always tested (none is skipped) and,
 - ⇒ 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****

Example-1: Nested if-else Statement

```
1. // Program to relate two integers using =, > or <
2. #include <stdio.h>
3. int main() {
4.     int number1, number2;
5.     printf("Enter two integers: ");
6.     scanf("%d %d", &number1, &number2);
7.     //checks if two integers are equal.
8.     if(number1 == number2) {
9.         printf("Result: %d = %d",number1,number2); }
10.    //checks if number1 is greater than number2.
11.    else if (number1 > number2) {
12.        printf("Result: %d > %d", number1, number2); }
13.    // if both test expression is false
14.    else {
15.        printf("Result: %d < %d",number1, number2); }
16.    return 0; }
```

Example: if/else Selection Structure

- Write a C program that reads two variables and decides if they are equal or not .

```
int var1, var2;
printf("Input the value of var1:");
scanf("%d", &var1);
printf("Input the value of var2:");
scanf("%d",&var2);
if (var1 !=var2) {
printf("var1 is not equal to var2");
    //Below – if-else is nested inside another if block
    if (var1 >var2) {
        printf("var1 is greater than var2");
    }
else {
printf("var2 is greater than var1");
}}
else { printf("var1 is equal to var2"); }
```

Example-2: Nested if-else Statement

- Write a C program that reads the student mark and decides the letter grade.

- ⇒ Pseudo-code for a nested `if/else` structure

Read the grade of the student

If student's grade is greater than or equal to 90

Print "A"

else

If student's grade is greater than or equal to 80

Print "B"

else

If student's grade is greater than or equal to 70

Print "C"

else

If student's grade is greater than or equal to 60

Print "D"

else

Print "F"

Example-3: Nested if-else Statement

- Given the following **decision table**, use it to calculate the Tax of the employee.

Salary Range (\$)	Base Tax (\$)	% Excess
0.00 to 14,999.99	0.00	15%
15,000.00 to 29,999.99	2,250.00	18%
30,000.00 to 49,999.99	5,400.00	22%
50,000.00 to 79,999.99	11,000.00	27%
80,000.00 to 150,000.00	21,600.00	33%

Function `comp_tax`

- ❑ Function `comp_tax` computes the tax based on the tax table shown in the previous slide.

```
1.  /*
2.   * Computes the tax due based on a tax table.
3.   * Pre : salary is defined.
4.   * Post: Returns the tax due for 0.0 <= salary <= 150,000.00;
5.   *       returns -1.0 if salary is outside the table range.
6.   */
7.  double
8.  comp_tax(double salary)
9.  {
10.     double tax;
11.
12.     if (salary < 0.0)
13.         tax = -1.0;
```

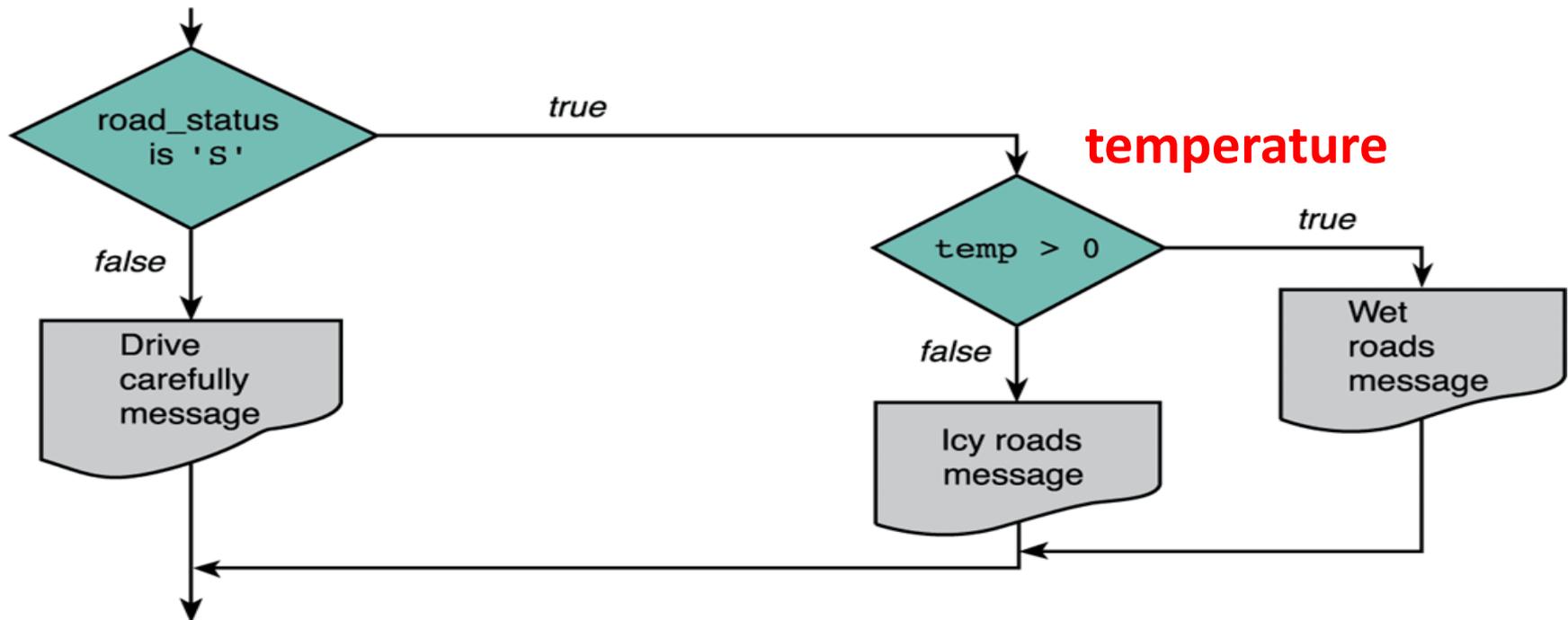
Function `comp_tax` (cont'd)

```
14.     else if (salary < 15000.00)
15.         tax = 0.15 * salary;
16.     else if (salary < 30000.00)
17.         tax = (salary - 15000.00) * 0.18 + 2250.00;
18.     else if (salary < 50000.00)
19.         tax = (salary - 30000.00) * 0.22 + 5400.00;
20.     else if (salary < 80000.00)
21.         tax = (salary - 50000.00) * 0.27 + 11000.00;
22.     else if (salary <= 150000.00)
23.         tax = (salary - 80000.00) * 0.33 + 21600.00;
24.     else
25.         tax = -1.0;
26.
27.     return (tax);
28. }
```

Road Sign Decision

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

'S' means the road is **Slippery/oily**



Road Sign Nested **if** Statement

```
if (road_status == 'S')
```

```
    if (temp > 0) {
```

```
        printf("Wet roads ahead\n");
```

```
        printf("Stopping time doubled\n");
```

```
    }
```

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

```
    else {
```

```
        printf("Icy roads ahead\n");
```

```
        printf("Stopping time quadrupled\n");
```

```
    }
```

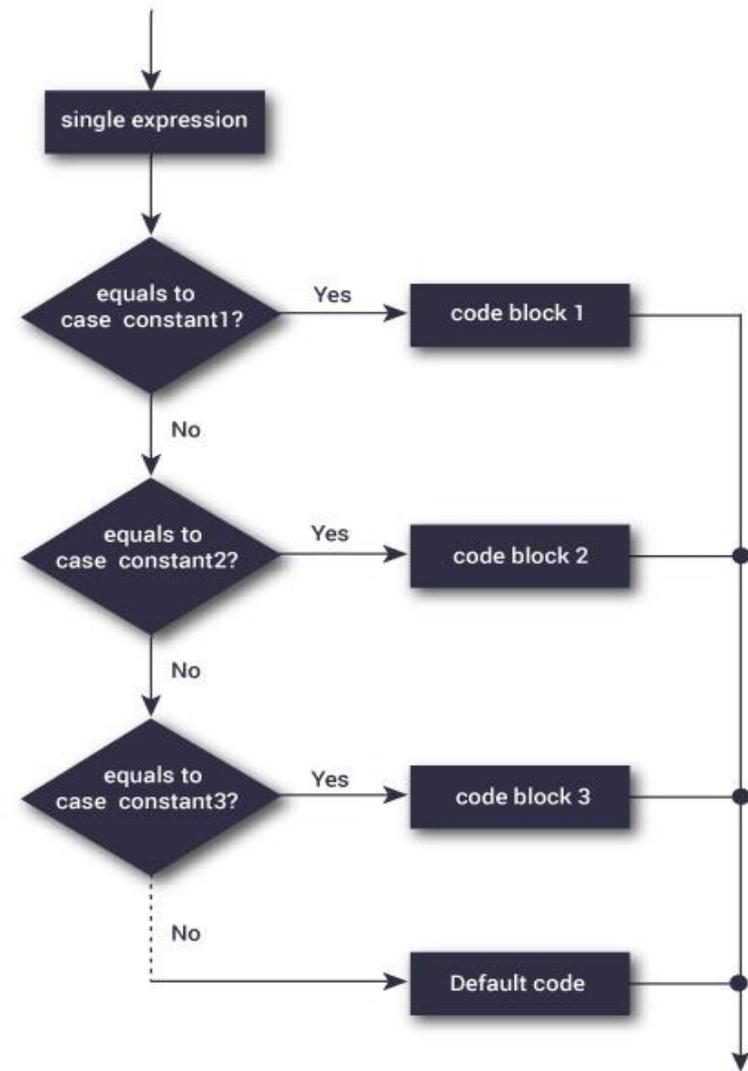
```
else
```

```
    printf("Drive carefully!\n");
```

The **switch** Statement

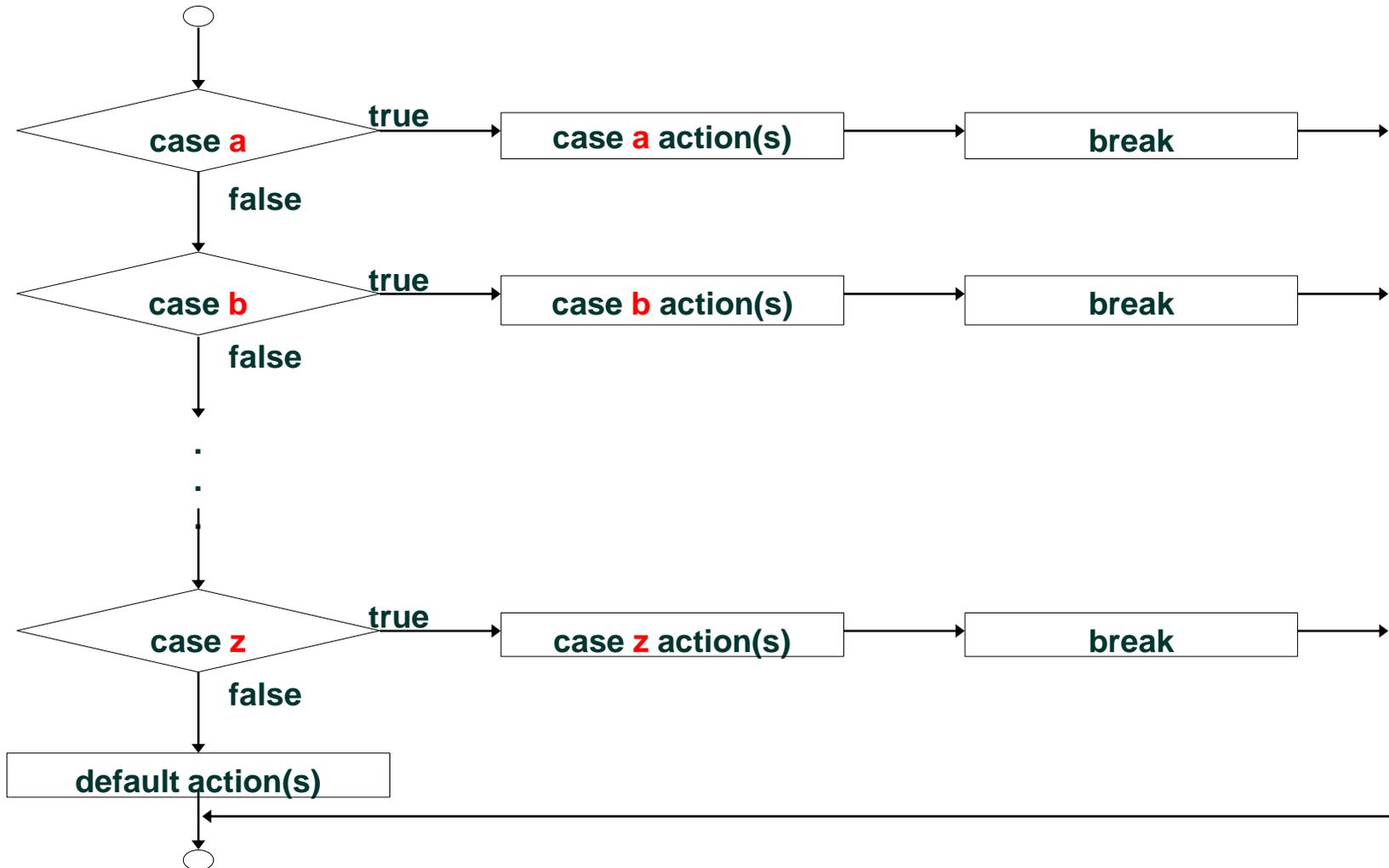
- ❑ It can be used to **select one of several alternatives.**
- ❑ Based on the **value of a variable or simple expression.**
- ❑ Variable or expression may be of type **int** or **char** **but** not of type **double**.
- ❑ The syntax for a **switch** statement in C is as follows:

```
switch(expression) {  
  case constant-expression :  
    statement(s);  
    break; /* optional */  
  case constant-expression :  
    statement(s);  
    break; /* optional */  
  /*you can have any number of case  
    statements*/  
  default : /* Optional */  
    statement(s); }  
}
```



The switch Multiple-Selection Structure

□ Flowchart of the `switch` structure



Explanation of **switch** Statement

- ❑ It takes the value of the variable **class** 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 **class**.
- ❑ 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 **class**, then the **default** case is executed.
- ❑ **Default** case 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 **braces** after a **case** label.
- ❑ You **cannot use a string** as a **case** label.
 - ➔ *i.e.* `case "Cruiser":` 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 braces of the **switch** statement.

Example of switch Statement

- **Example:** Display a message indicating the ship class based on this table.

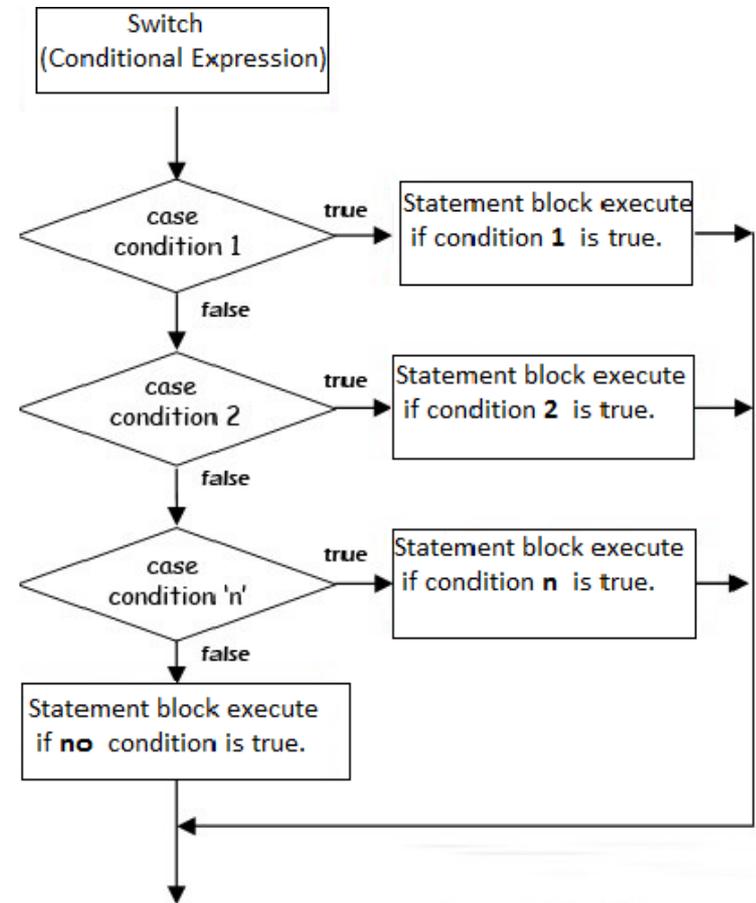
```
1.  switch (class) {
2.  case 'B':
3.  case 'b':
4.      printf("Battleship\n");
5.      break;
6.
7.  case 'C':
8.  case 'c':
9.      printf("Cruiser\n");
10.     break;
11.
12. case 'D':
13. case 'd':
14.     printf("Destroyer\n");
15.     break;
16.
17. case 'F':
18. case 'f':
19.     printf("Frigate\n");
20.     break;
21.
22. default:
23.     printf("Unknown ship class %c\n", class);
24. }
```

Class ID	Ship Class
'B' or 'b'	Battleship
'C' or 'c'	Cruiser
'D' or 'd'	Destroyer
'F' or 'f'	Frigate

Example: Switch statement in C

- What is the output of the following program?

```
1. #include <stdio.h>
2. int main() {
3.   int num=2;
4.   switch(num+2) {
5.     case 1: printf("Case1: Value is: %d", num);
6.     case 2: printf("Case1: Value is: %d", num);
7.     case 3: printf("Case1: Value is: %d", num);
8.     default: printf("Default: Value is: %d", num);
9.   }
10.  return 0;
11. }
```



Example: Switch statement in C

```
1. #include <stdio.h>
2. int main() {
3.     int num=2;
4.     switch(num+2) {
5.         case 1: printf("Case1: Value is: %d", num);
6.         case 2: printf("Case1: Value is: %d", num);
7.         case 3: printf("Case1: Value is: %d", num);
8.         default: printf("Default: Value is: %d", num);
9.     }
10.    return 0;
11. }
```

Output: Default: value is: 2

- Since num value is 2 and after addition the expression resulted 4.
- Since there is no case defined with value 4 the default case is executed.

Example: Switch statement in C

- What is the output of the following program?

```
1. #include <stdio.h>
2. int main() {
3.     int i=2;
4.     switch (i) {
5.         case 1: printf("Case1 ");
6.         case 2: printf("Case2 ");
7.         case 3: printf("Case3 ");
8.         case 4: printf("Case4 ");
9.         default: printf("Default ");
10.    }
11.    return 0;
12. }
```

Example: Switch statement in C

```
1. #include <stdio.h>
2. int main() {
3.     int i=2;
4.     switch (i) {
5.         case 1: printf("Case1 ");
6.         case 2: printf("Case2 ");
7.         case 3: printf("Case3 ");
8.         case 4: printf("Case4 ");
9.         default: printf("Default ");
10.    }
11.    return 0;
12. }
```

Output: Case2 Case3 Case4 Default

- The value of the variable is 2 so the control jumped to the case 2,
- However there are no such statements in the above program which could break the flow after the execution of case 2.
- That's the reason after case 2, all the subsequent cases and default statements got executed.

Example: Switch statement in C

- What is the output of the following program?

```
1. #include <stdio.h>
2. int main() {
3.     int i=2;
4.     switch (i) {
5.         case 1: printf("Case1 ");
6.         break;
7.         case 2: printf("Case2 ");
8.         break;
9.         case 3: printf("Case3 ");
10.        break;
11.        case 4: printf("Case4 ");
12.        break;
13.        default: printf("Default ");
14.    }
15.    return 0;
16. }
```

Example: Switch statement in C

```
1. #include <stdio.h>
2. int main() {
3.     int i=2;
4.     switch (i) {
5.     case 1: printf("Case1 ");
6.     break;
7.     case 2: printf("Case2 ");
8.     break;
9.     case 3: printf("Case3 ");
10.    break;
11.    case 4: printf("Case4 ");
12.    break;
13.    default: printf("Default ");
14.    }
15.    return 0;
16. }
```

Output: Case2

- The value of the variable `i` is 2, the control goes to case 2 and then break.

□ 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**.

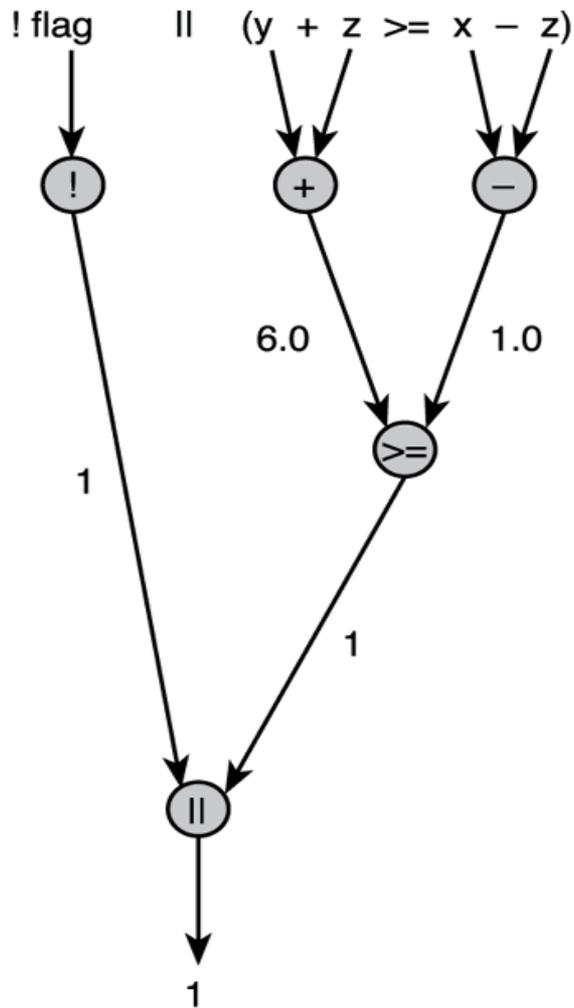
□ **switch** statement

- ⇒ 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 Priority/Precedence/الأُسْبُقِيَّة

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

Evaluation Tree, Step-by-Step Evaluation



flag	y	z	x
0	4.0	2.0	3.0

! flag		(y + z)	>=	(x - z)
<u>0</u>		<u>4.0 2.0</u>		<u>3.0 2.0</u>
1		<u>6.0</u>		<u>1.0</u>
			<u>1</u>	
	<u>1</u>			

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
 $(\text{divisor} \ != \ 0 \ \&\& \ x \ / \ \text{divisor} \ > \ 5)$

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)`

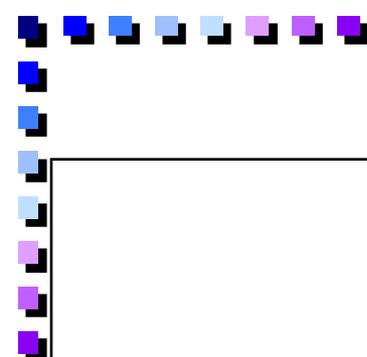
Example	Equivalent Expression
<code>!(item == 5)</code>	<code>item != 5</code>
<code>!(age >= 65)</code>	<code>age < 65</code>
<code>!(n > 0 && n < 10)</code>	<code>n <= 0 n >= 10</code>
<code>!(x == 1 x == 3)</code>	<code>x != 1 && x != 3</code>
<code>!(x>y && (c=='Y' c=='y'))</code>	<code>(x<=y) (c!='Y'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

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 nest 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.
 - ⇒ You do not need to enclose multiple statements in braces after a **case** label.
 - ⇒ You cannot use a string as a **case** label.
 - i.e. **case "Cruiser":** is not allowed
 - ⇒ Do not forget **break** at the end of each alternative.
 - ⇒ Do not forget the braces of the **switch** statement.



The End!!

Thank you

Any Questions?



Outline of Ch. 4 Topics

- In the last class, we discussed:
 - ⇒ Control Structure Statements
 - Sequential, Selection, and Repetition statements
 - ⇒ Compound statements with examples
 - ⇒ Conditions Expression
 - ⇒ Relational, and Logic Operators
 - Examples of Relational and Equality Operators in Expressions
 - Example of Logical Operators in Expressions
 - ⇒ Conditions, Relational, and Logic Operators
 - ⇒ The if statement and its Flowchart
 - ⇒ if Statement with Compound Statements
 - ⇒ Nested **if** statements
- In today's class, we are going to discuss:
 - ⇒ More Program Examples of using Nested **if** statements
 - ⇒ The **switch-case** Statement
 - ⇒ Program Examples of using **switch-case** statements
 - ⇒ Operators Priority,
 - ⇒ Complementing a Condition
 - ⇒ Common Programming Errors