

# Branching and Iteration

- ❖ The C language provides three types of decision-making constructs: if-else, the *conditional expression* `?:`, and the switch statement.
- ❖ It also provides three looping constructs: while, do-while, and for.

# Boolean Expressions

- Evaluate to true or false
- Forms
  - Relational expression: `<expr> <relational operator> <expr>`
    - Examples:  
`7 < 5`  
`a + b > 6`
  - Logical expression: `<Boolean expr> <logical operator> <Boolean expr>`
    - Examples:  
`(x < 7) && (y > 3)`

# Relational Operators

Standard Algebraic Relational Operator	C Relational Operator	C Condition Example	Meaning of C Condition
Inequality			
$<$	$<$	$x < y$	<b>x</b> is less than <b>y</b>
$\leq$	$\leq$	$x \leq y$	<b>x</b> is less than or equal to <b>y</b>
$>$	$>$	$x > y$	<b>x</b> is greater than <b>y</b>
$\geq$	$\geq$	$x \geq y$	<b>x</b> is greater than or equal to <b>y</b>
Equality			
$=$	$==$	$x == y$	<b>x</b> is equal to <b>y</b>
$\neq$	$!=$	$x != y$	<b>x</b> is not equal to <b>y</b>

4<sup>th</sup>: Ch 4 p. 46

3<sup>rd</sup>: Ch 5 p. 46

# Logical Operators (Compound Relationals)

- **&&** (logical **AND**)
  - Returns **true** if both conditions are **true**
- **||** (logical **OR**)
  - Returns **true** if either of its conditions is **true**
- **!** (logical **NOT**, logical negation)
  - Is a unary operator, only takes one operand following
  - Reverses the truth/falsity of its condition
  - Returns **true** when its condition is **false**

# Logical Operators Truth Table

P	Q	P && Q	P    Q	!P
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

# Precedence of Operators

1. `()`, `[]`
2. Unary `+`, unary `-`, `!`, `++`, `--`
3. Type casting
4. `*`, `/`, `%`
5. `+`, `-`
6. `<`, `<=`, `>`, `>=`
7. `==`, `!=`
8. `&&`
9. `||`
10. `=`

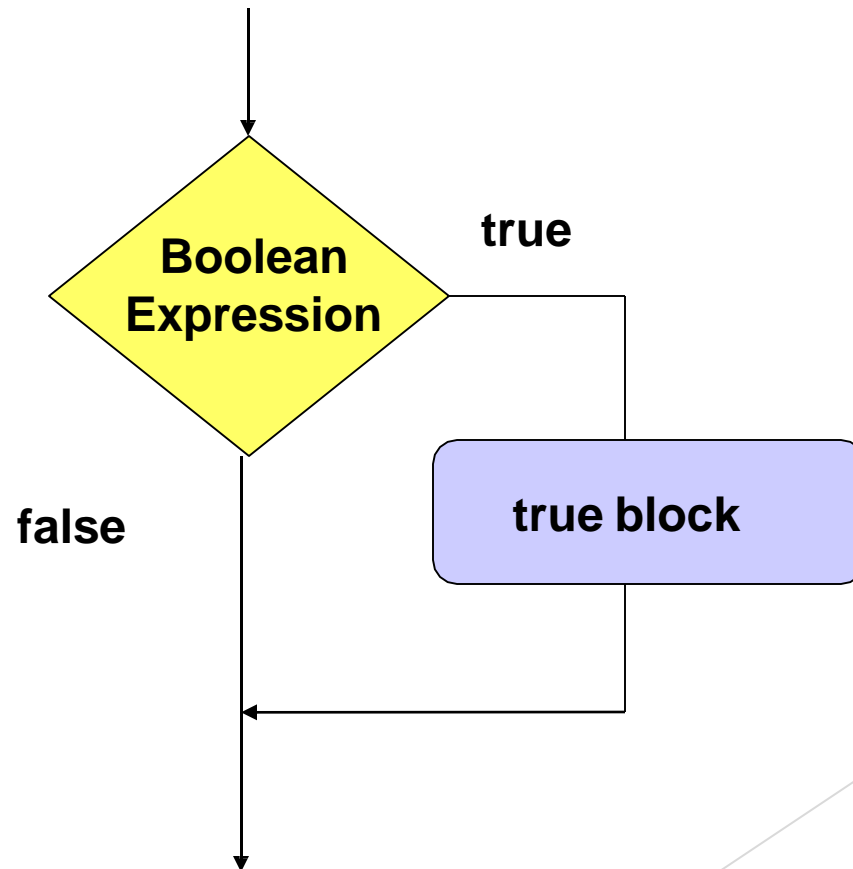
# The *if* Selection Structure

- Selection structure
  - used when we want the computer to choose between two alternative courses of action



# The *if* Selection Structure

- *if* Statement





# The *if* Selection Structure

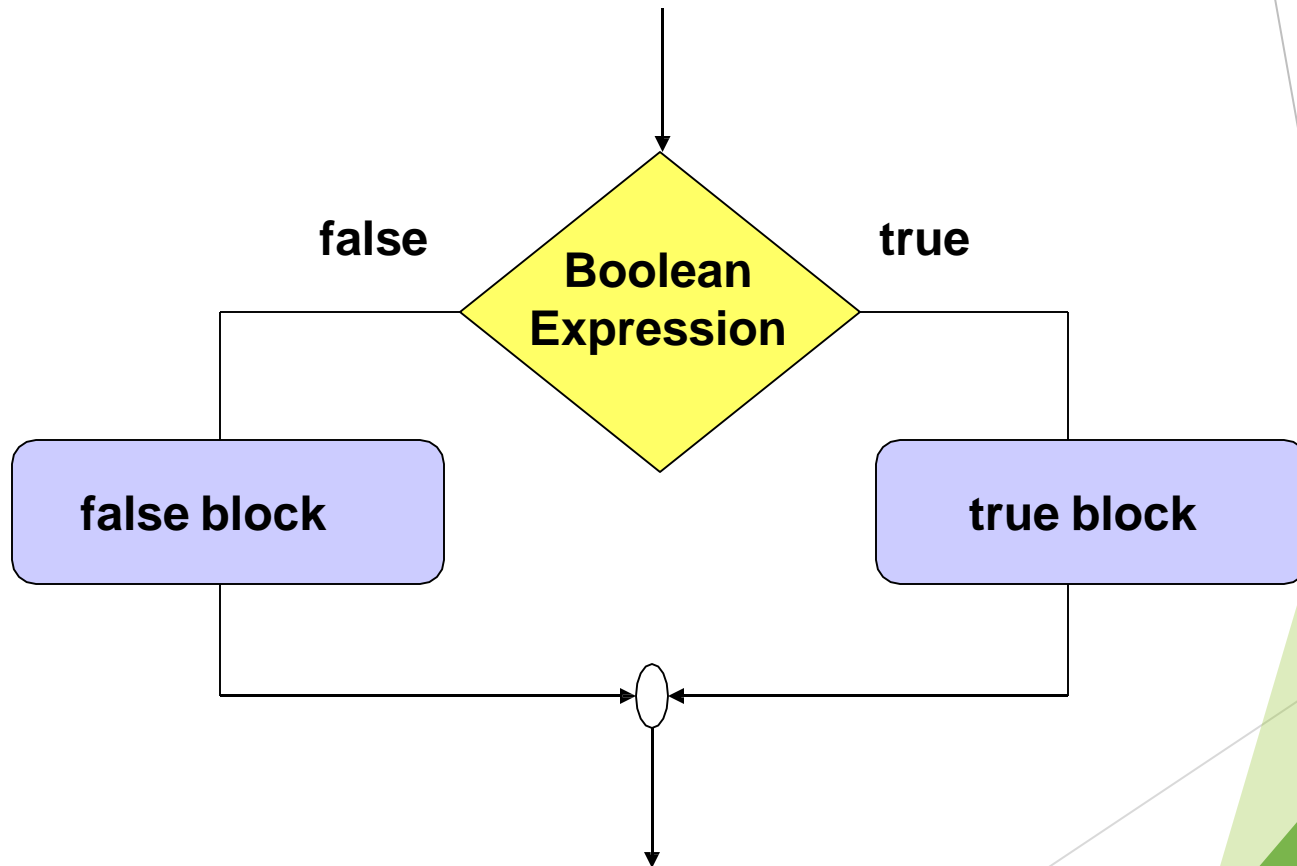
- ▶ General form of *if*:
- ▶ `if (Boolean Expression)`
  - ▶ `{`  
    `statement2;`  
    `} statement1;`

# The `if-else` Selection Structure

- `if`
  - Only performs an action if the condition is true
- `if-else`
  - A different action is performed when condition is true and when condition is false

# *if-else* Selection Structure

*if-else* statement



# The *if-else* Selection Structure

General form of *if-else*:

```
if (expression)
{
    statement1A;
    statement2A;
    ...
}
else
{
    statement1B;
    statement2B;
    ...
}
```

# The *if-else* Selection Structure

- Nested *if-else* structures
  - Test for multiple cases by placing **if-else** selection structures inside **if-else** selection structures.

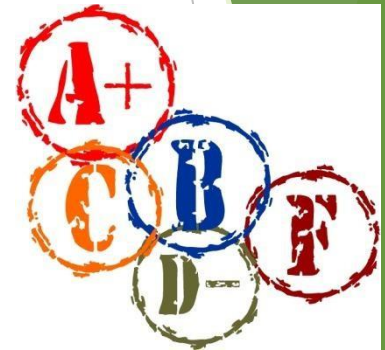


# Nested if-else Structures

```
if (score >= 70)
{
    if (age < 13)
    {
        printf("Great job\n");
    }
    else
    {
        printf("You passed\n");
    }
}
else
{
    printf("You did not pass\n");
}
```

# The *if-else-if* Construct

```
if (grade >= 90)
    printf("A\n");
else
    if (grade >= 80)
        printf("B\n");
    else
        if (grade >= 70)
            printf("C\n");
        else
            if (grade >= 60)
                printf("D\n");
            else
                printf("F\n");
```



- Once a condition is met, the rest of the statements are skipped

# The *if-else-if* Construct

The standard way to indent the previous code is

```
if (grade >= 90)
    printf("A\n");
else if (grade >= 80)
    printf("B\n");
else if (grade >= 70)
    printf("C\n");
else if (grade >= 60)
    printf("D\n");
else
    printf("F\n");
```



Great  
Job!  
A+



# The *if-else* Selection Structure

- Compound statement:
  - Set of statements within a pair of braces
  - Example:

```
if (grade >= 90) {  
    printf("Congratulations!\n");  
    printf("You made an A this course\n");  
}
```



# The *if-else* Selection Structure

–Without the braces, only one statement is executed.  
e.g. given the following code:

```
if (grade >= 90)
    printf("Congratulations!\n");
    printf("You made an A this course\n");
```



- The statement,

```
printf("You made an A this course\n");
```

will be executed independent of the value of grade.

- The statement,

```
printf("Congratulations!\n");
```

will execute only if grade is greater than or equal to 90.

# The *dangling* else

```
if (x < y)
    if (x < z)
        printf("Hello\n");
else
    printf("Goodbye\n");
```

**Note:** the compiler matches an else with the closest unmatched if  
The above will be treated as

```
if (x < y)
    if (x < z)
        printf("Hello\n");
else
    printf("Goodbye\n");
```

# The *dangling* else

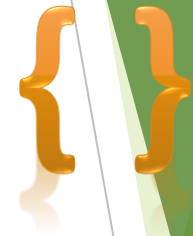
If the else is to match the outer if, use braces.

```
if (x < y)
{
    if (x < z)
        printf("Hello\n");
}
else
    printf("Goodbye\n");
```



# *if-else* Construct

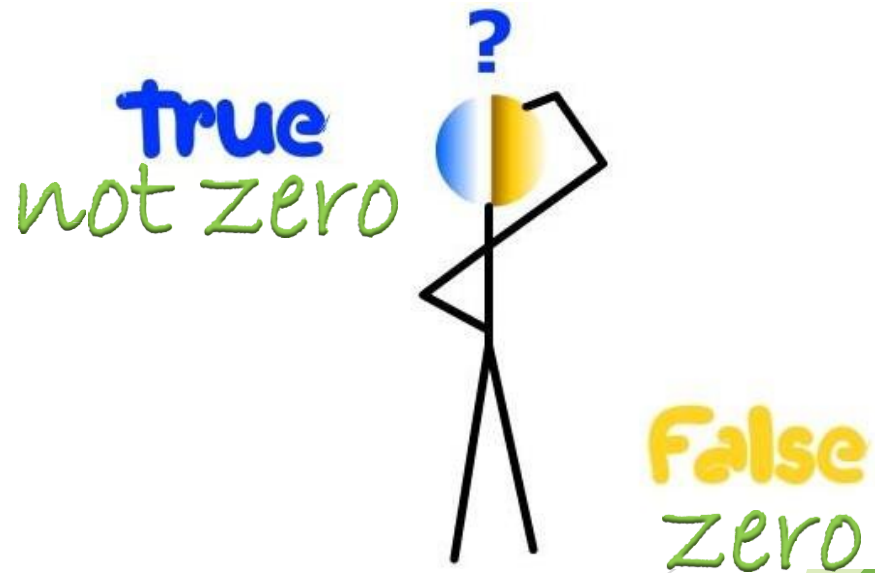
- To avoid confusion, and possible errors, it is best to use braces even for single statements.
  - However, code will be longer



```
if (x < y)
{
    if (x < z)
    {
        printf("Hello\n");
    }
}
else
{
    printf("Goodbye\n");
}
```

# Conditionals

- C uses an integer to represent Boolean values
  - Zero is interpreted as false
  - Any other integer value is interpreted as true



# Conditionals

- `if (n = 0)` is not a syntax error in C.
  - The expression,  $n = 0$ , assigns zero to  $n$  and the value of the expression is 0. Zero is interpreted as false, and the false branch of the if statement will be taken.
- `if (n = 5)` is not a syntax error in C.
  - The expression assigns 5 to  $n$ . 5 is interpreted as true, and the true branch of the if statement will be taken.

warning: suggest parentheses around assignment used as truth value

# Conditionals



- Remember to use the == operator to test for equality.
- To help catch the error when the equality check involves a constant, put the constant on the left hand side of the ==.

● For example, use `if (0 == n)`  
instead of `if (n == 0)`

Since `0 = n` is not a valid assignment in C, the compiler will detect this error when == is intended.

```
error: invalid lvalue in assignment
```



# Conditionals

Program for analysis of people of certain age groups with salary who are eligible for getting a suitable loan in banking system if their condition and norms get satisfied using nested if statement.

Age<18 illegal

>18, <50 for salary >3000 accept

>50 illegal

Show details for user in weakness point

# Conditionals

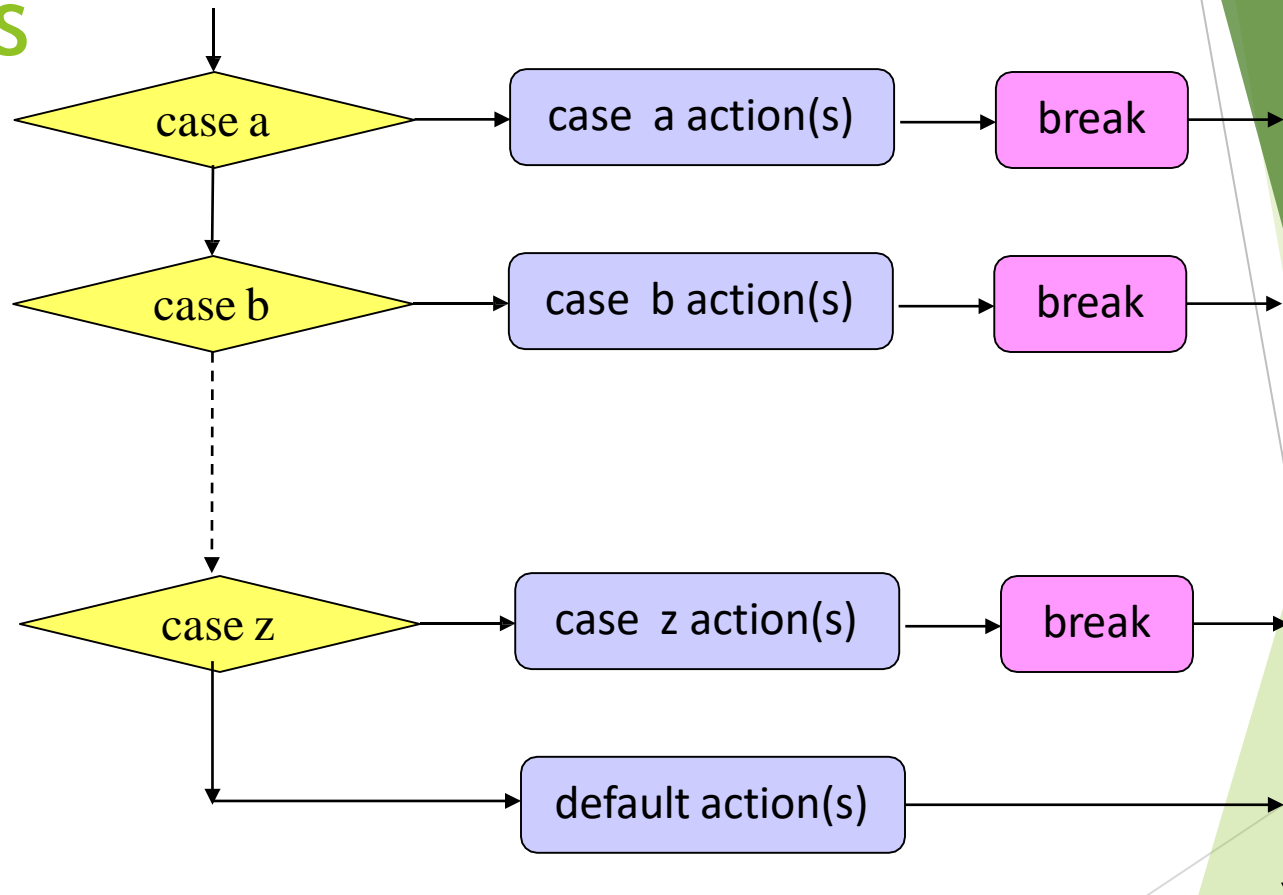
Program to find which number is greater among the considered number and then how the execution happens with the help of nested if statement if the flow gets successful then it is counted as normal flow. 3 numbers using nested if

# The *switch* Multiple-Selection Structure

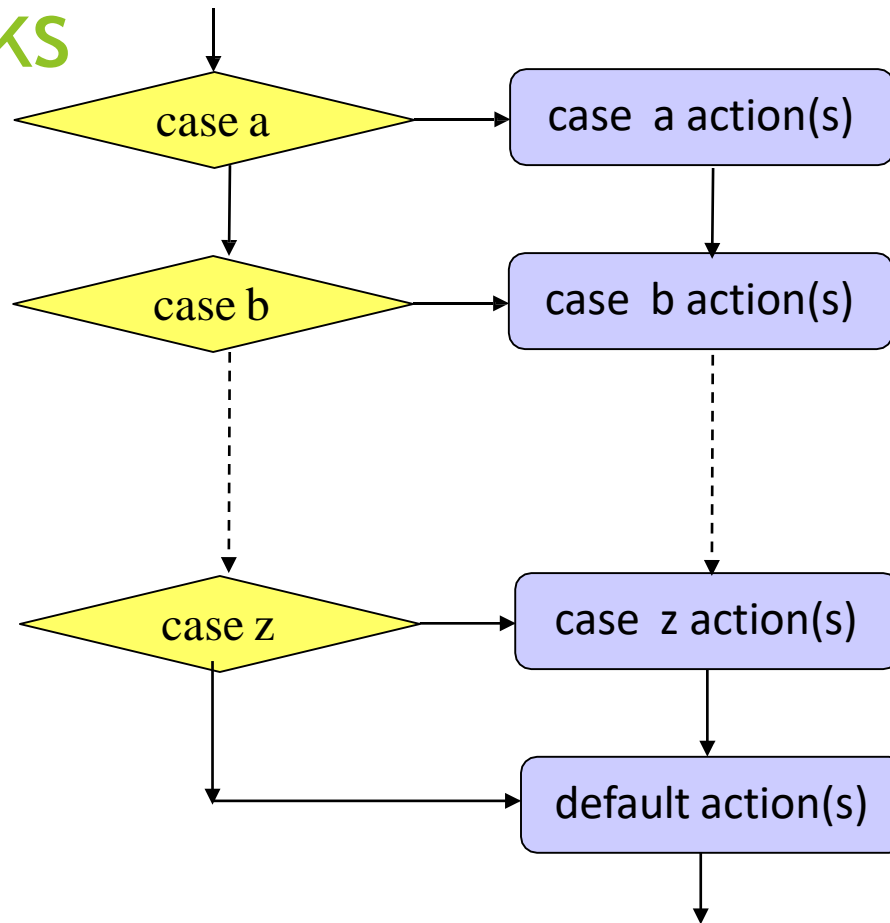
- *switch*
  - Useful when variable or expression is tested for multiple values
  - Consists of a series of **case** labels and an optional **default** case



# The *switch* Multiple-Selection Structure With Breaks

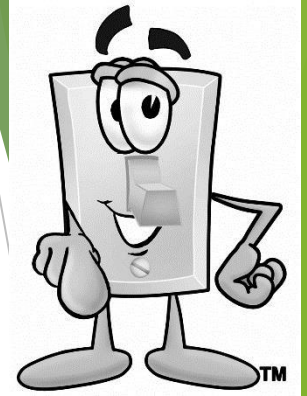


# The *switch* Multiple-Selection Structure Without Breaks



# *switch* Statement Syntax

```
switch (switch_expression)
{
    case constant1:
        statementSequence1
        break;
    case constant2:
        statementSequence2
        break;
    ...
    case constantN:
        statementSequenceN
        break;
    default:
        defaultStmtSequence
}
```





# *switch* Statement

- The `switch_expression` is compared against the values *constant1, constant2, ..., constantN*
  - *constant1, constant2, ..., constantN* must be simple constants or constant expressions.
    - Can be a char or an int
    - Best to use the same type constant as the switch expression
      - If not, a type conversion will be done.

# *switch* Statement

## Reminder

- The *switch* statement ends
  - break statement
  - end of the switch statement
- When executing the statements after a case label, it continues to execute until it reaches a break statement or the end of the switch.
- If you omit the break statements, then after executing the code for one case, the computer will continue to execute the code for the next case.





# Example of *switch*

```
// Accept letter grade and print corresponding points
printf("Enter letter grade: ");
scanf("%c", &letter_grade);
switch (letter_grade) {
    case 'A':
    case 'a':
        points = 4.0;
        break;
    case 'B':
    case 'b':
        points = 3.0;
        break;
    case 'C':
    case 'c':
        points = 2.0;
        break;
    case 'D':
    case 'd':
        points = 1.0;
        break;
    case 'F':
    case 'f':
        points = 0.0;
        break;
    default:
        points = 0.0;
        printf("Invalid letter grade\n");
}
```

# Quiz

- Write C Program to accept degree of student and print his grade according to :
- Lower than 50 fail
- from 50 to 65 pass
- 65 to 75 good
- 75 to 85 very good
- Greater than 85 excellent

# Quiz

- Write C Program to design calculator that accept two operands and operator than print the output of arithmetic operation using switch statement.

# *Quiz*

- Write C Program to accept three numbers and print the largest one on the screen.

# Quiz

- Write C Program to accept two numbers and swap between them. Print values of two numbers before and after swap process.

# *Quiz*

- Write C Program to accept character and check if it small or capital letter

# *Quiz*

- Write C Program to accept number of day on week then print the day name using switch statement.