Chapter 6
Control Statements - Selection

Outline

- Algorithm, pseudocode, and flow chart
- Control statements
- The *if* statement
- The *switch* statement
- The *goto* statement

Top-Down Design

- Before writing a program:
  - Have a thorough understanding of the problem.
  - Plan an approach for solving it carefully .

- While writing a program:
  - Know what “building blocks” are available.
  - Use good programming principles.

Algorithms

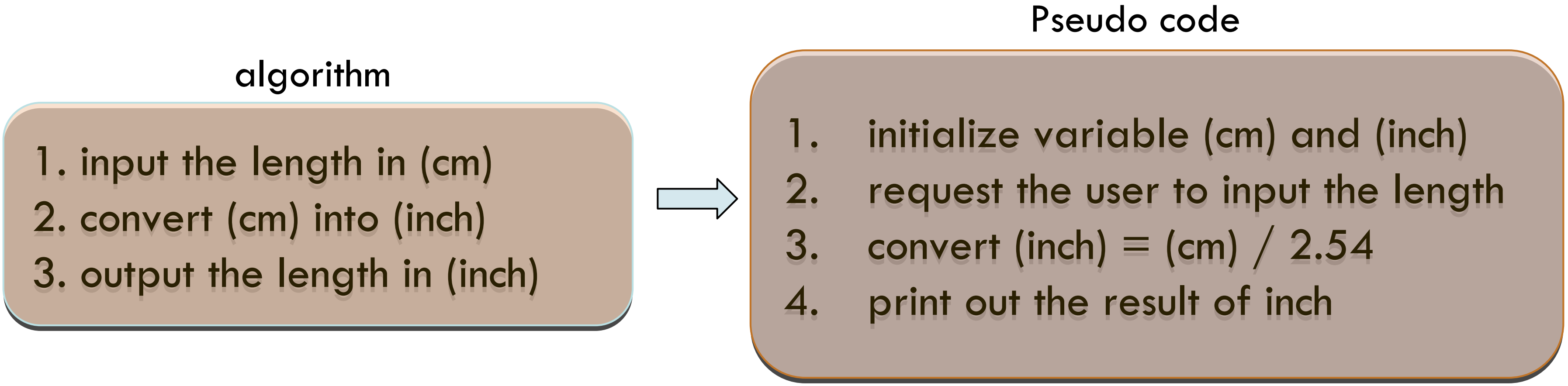
- Computing problems
  - Solved by executing a series of actions in a specific order
- Algorithm: procedure in terms of
  - **Actions** to be executed
  - The **order** in which these actions are to be executed
- **Program control**
  - Specify order in which statements are to be executed

1. input the length in (cm)
2. convert (cm) into (inch)
3. output the length in (inch)

example

Pseudocode

- Informal language that helps us develop algorithms.
- Help us think out a program before writing it.
- Easy to convert into a C program.



algorithm

```
1. input the length in (cm)
2. convert (cm) into (inch)
3. output the length in (inch)
```

Pseudo code

```
1.  initialize variable (cm) and (inch)
2.  request the user to input the length
3.  convert (inch) = (cm) / 2.54
4.  print out the result of inch
```

Control Structure

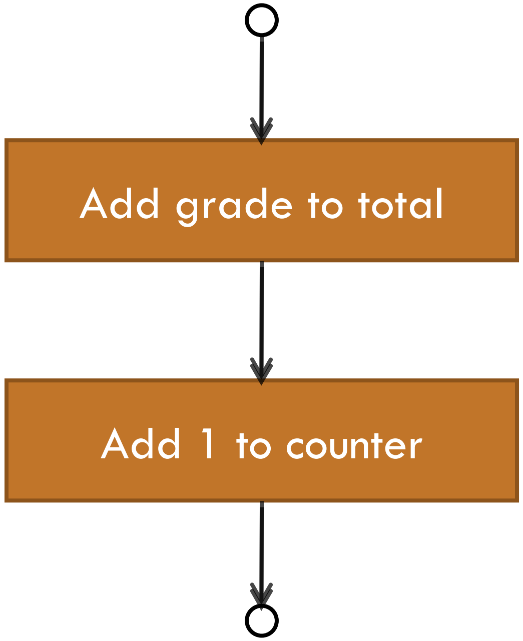
- ***Sequential*** structure
  - Step-by-step execution
  - Represented by ***rectangle*** symbol in the flow chart
- ***Selection*** structure
  - One specific execution from multiple (or single) choices
  - Statements: **if, if-else, switch**
  - Represented by ***diamond*** symbol in the flow chart
- ***Repetition*** structure
  - Repeat a control block several times
  - Statements: **while, do..while, for**

Flow Chart

- Graphical representation of an algorithm
- Composed of symbols connected by flow lines (arrows)
  - ***Action symbol (rectangle)***
    - Indicate any type of action.
  - ***Decision symbol (diamond)***
    - Indicate a decision that redirects the program to different sequences.

Flow Chart (Cont.)

Sequential structure: composed of multiple sequential actions

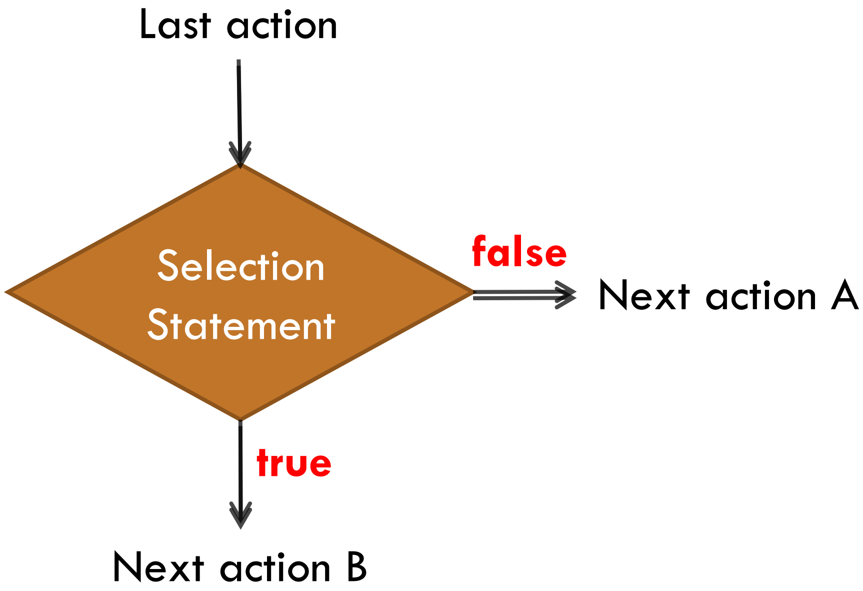


total = total + grade

count = count + 1

Selection Statements

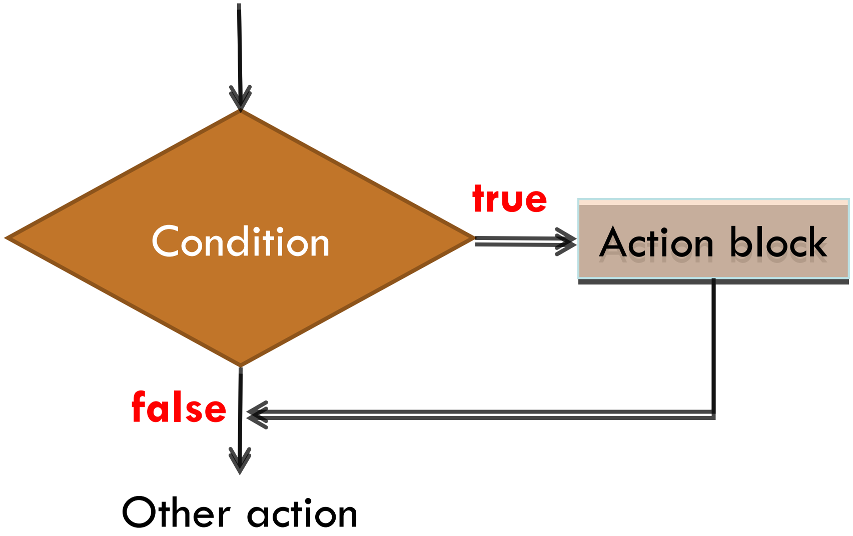
- Select the next action based on several (or one) conditions.
- Statements
  - if
  - if-else
  - switch
  - goto ***(not recommend)***



if

```
Syntax:
        if (condition)
        {
                statement 1;
                statement 2;
                …
                statement n;
        }
```

```
if (condition)
                single statement;
```



Condition

true

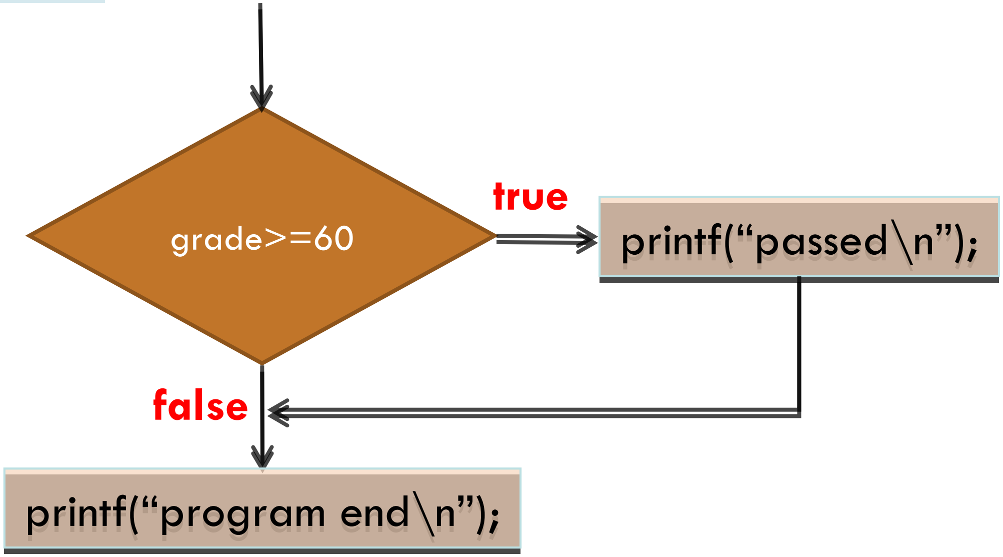
Action block

false

Other action

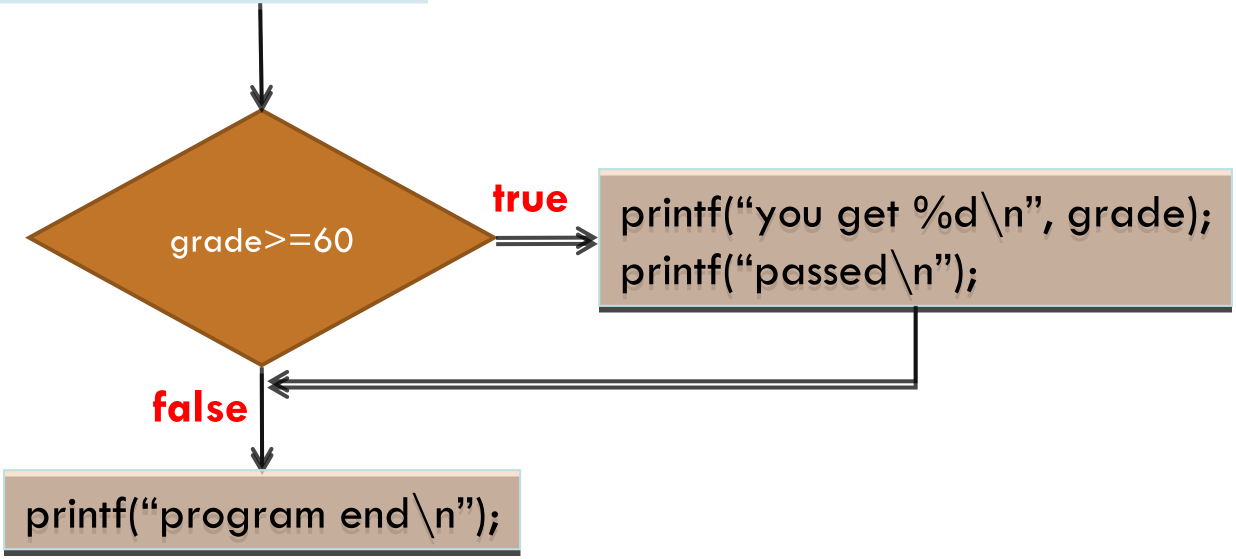
if: Example 1

```
if (grade >= 60)
        printf("passed\n");
printf("program end\n");
```



if: Example 2

```
if (grade >= 60)
{
            printf(“you get %d\n”, grade);
            printf(“passed\n”);
}
printf(“program end\n”);
```



```
printf(“you get %d\n”, grade);
printf(“passed\n”);
```

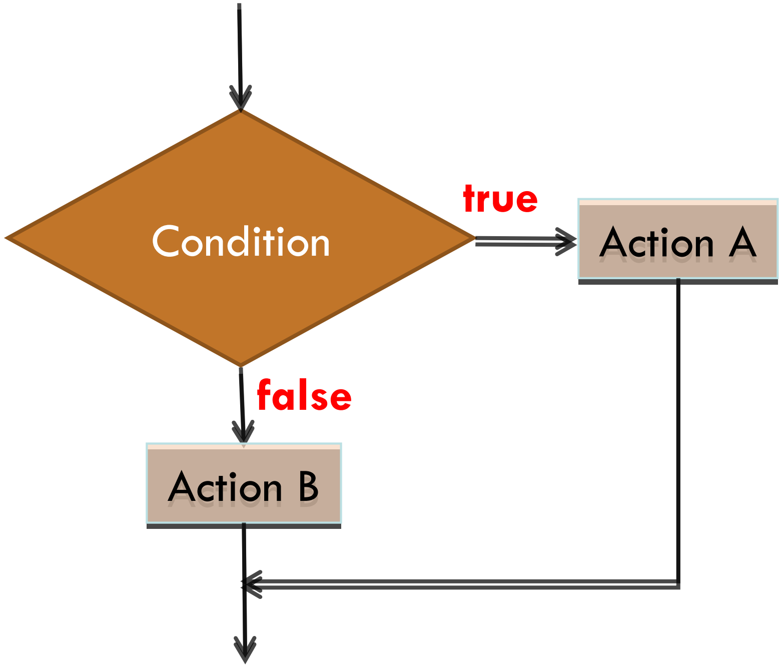
```
printf(“program end\n”);
```

if else

- Specify another action to be performed when the condition is *false.*

```
if(condition)
{
            statement 1;
            statement 2;
            …
}
Else
 {
            statement 1;
            statement 2;
            …..
}
```

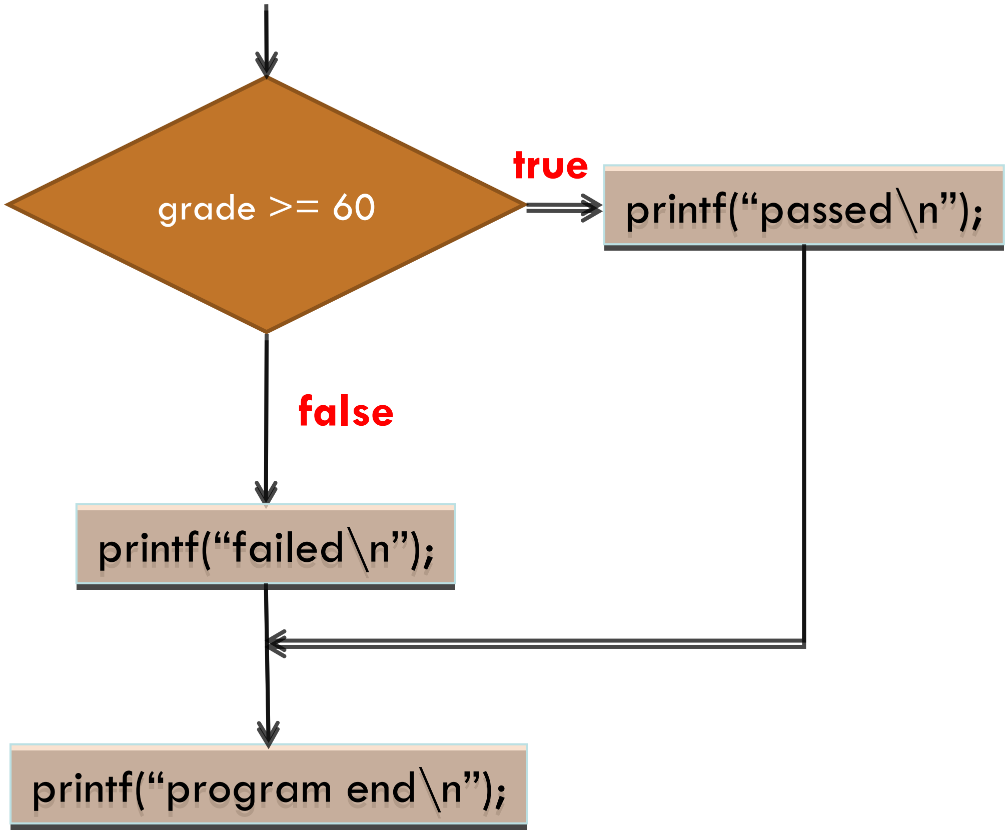
```
if(condition)
            single statement;
else
            single statement;
```



Can remove the braces if there is only one statement

if else: Example

```
if (grade >= 60)
            printf(“passed\n”);
else
            printf(“failed\n”);
printf(“program end\n”);
```



Ternary Conditional Operator (? :)

```
Condition ? Expression1 : Expression2;
```

↕

```
if (condition)
        Expression1;
else
        Expression2;
```

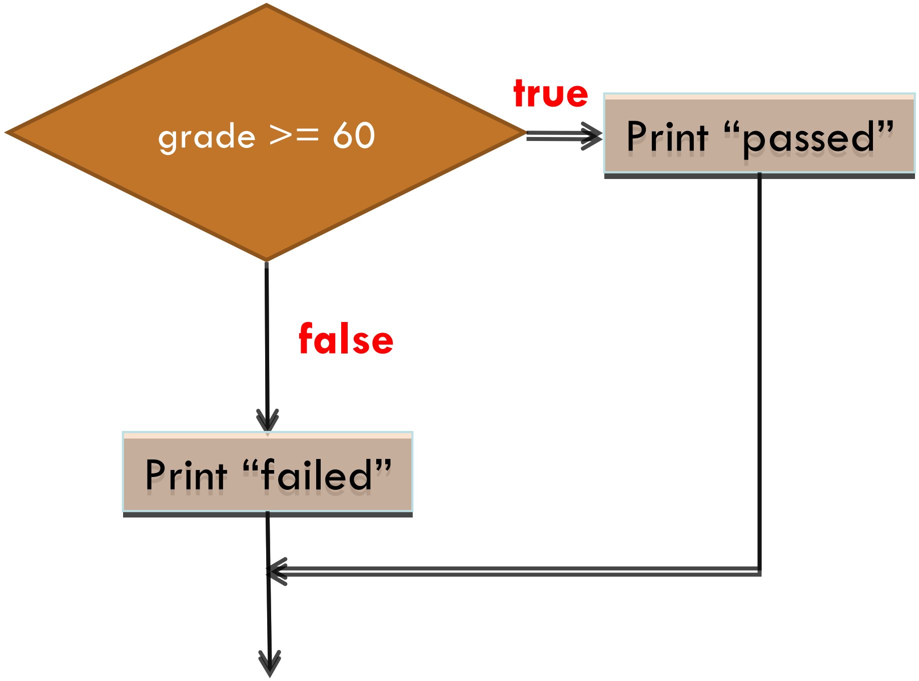
```
VariableName = Condition ? Expression1 : Expression2;
```

↕

```
if (condition)
        VariableName = Expression1;
else
        VariableName = Expression2;
```

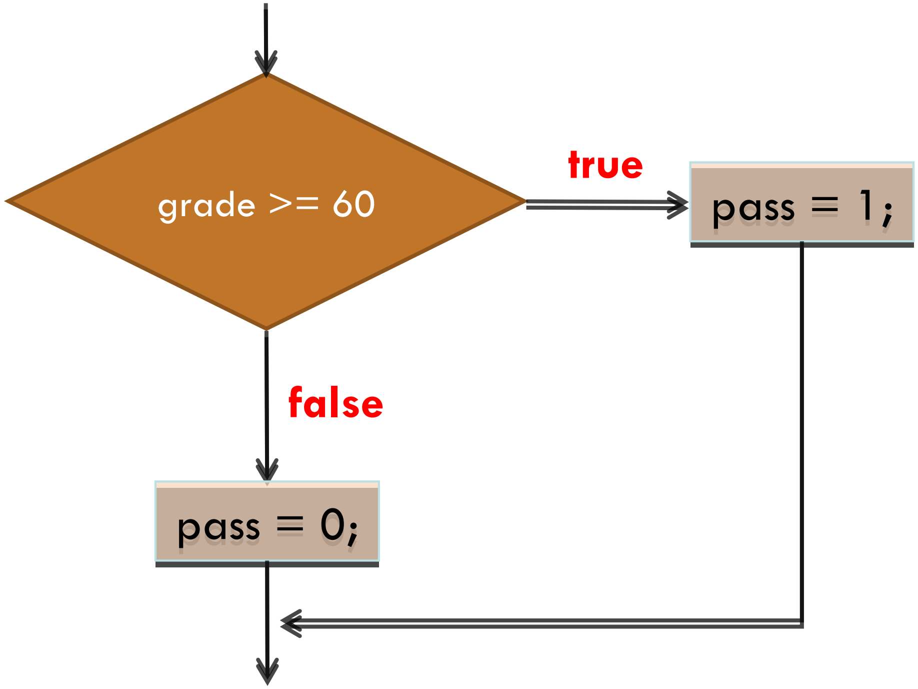
?: Operator: Example 1

```
printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );
grade >= 60 ? printf( “Passed\n” ) : printf( “Failed\n” );
```



?: Operator: Example 2

```
pass = grade >= 60 ? 1 : 0;
```



Comparison

- Program 1

```
–if (grade >= 60)
     printf(“passed\n”);
  else
     printf(“failed\n”);
```

1. Check if grade >= 60
2. Print “passed” or “failed”

- Program 2

```
–if (grade >= 60)
     printf(“passed\n”);
  if (grade < 60)
     printf(“failed\n”);
```

1. Check if grade >= 60
2. Print “passed” (if necessary)
3. **Check if grade < 60**
4. Print “failed” (if necessary)

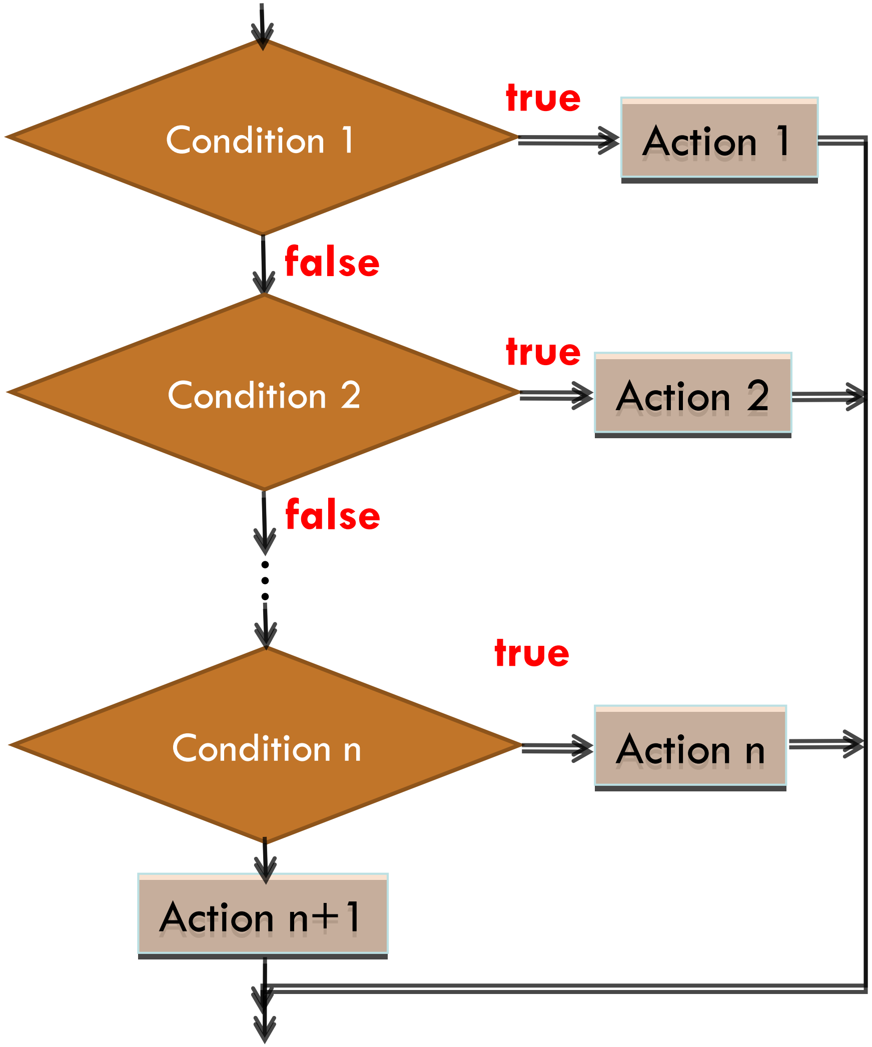
Redundant, unnecessary complexity

if-else if-else

- Used when there are multiple conditions
- Syntax

```
if (condition 1) {
        Action 1;
} else if (condition 2) {
        Action 2;
} else if (condition 3) {
        Action 3;
} else {
        Action 4;
}
```

optional

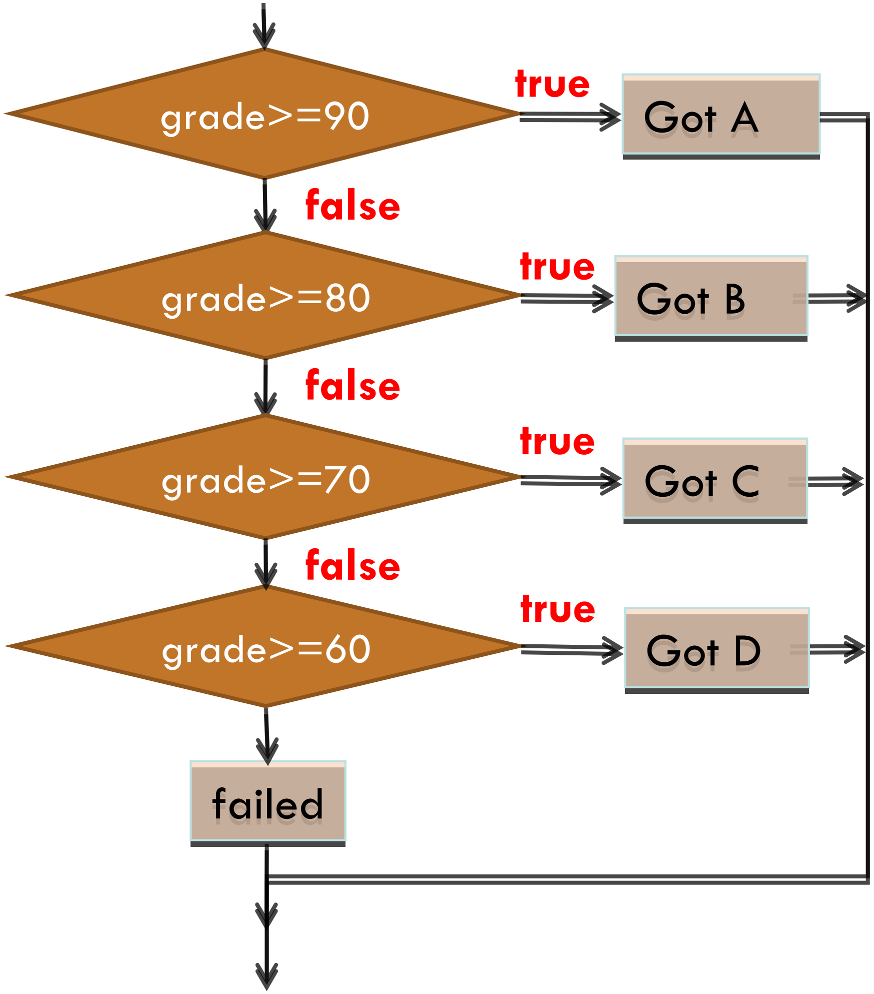


if-else if-else

- If k-th condition is true, skip checking the following conditions
- If one of the conditions is true, skip the selection structure

if-else if-else: Example

```
if (grade >= 90)
            printf(“you got A.\n”);
else if (grade >= 80)
            printf(“you got B.\n”);
else if (grade >= 70)
            printf(“you got C.\n”);
else if (grade >= 60)
            printf(“you got D.\n”);
else
            printf(“failed.\n”);
```



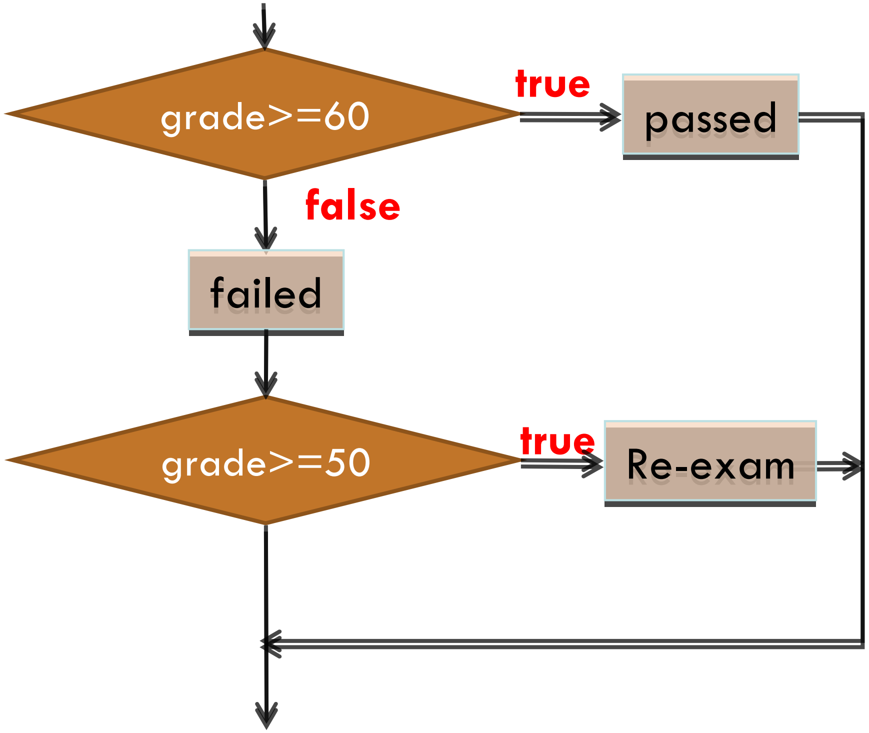
Nested if-else

- Test for multiple cases by placing if-else selection statement inside if-else selection statement.
- Syntax:

```
if(condition 1)
{
            statements;
            if (condition 2)
            {
                        statements;
            }
             else
            {
                        statements;
            }
}
else
{
            statements;
}
```

Nested if-else: Example

```
if (grade >= 60)
        printf(“passed\n”);
else
{
        printf(“failed\n”);
        if(grade >= 50)
                printf(“re-exam\n”);
}
```



grade>=60

true

passed

false

failed

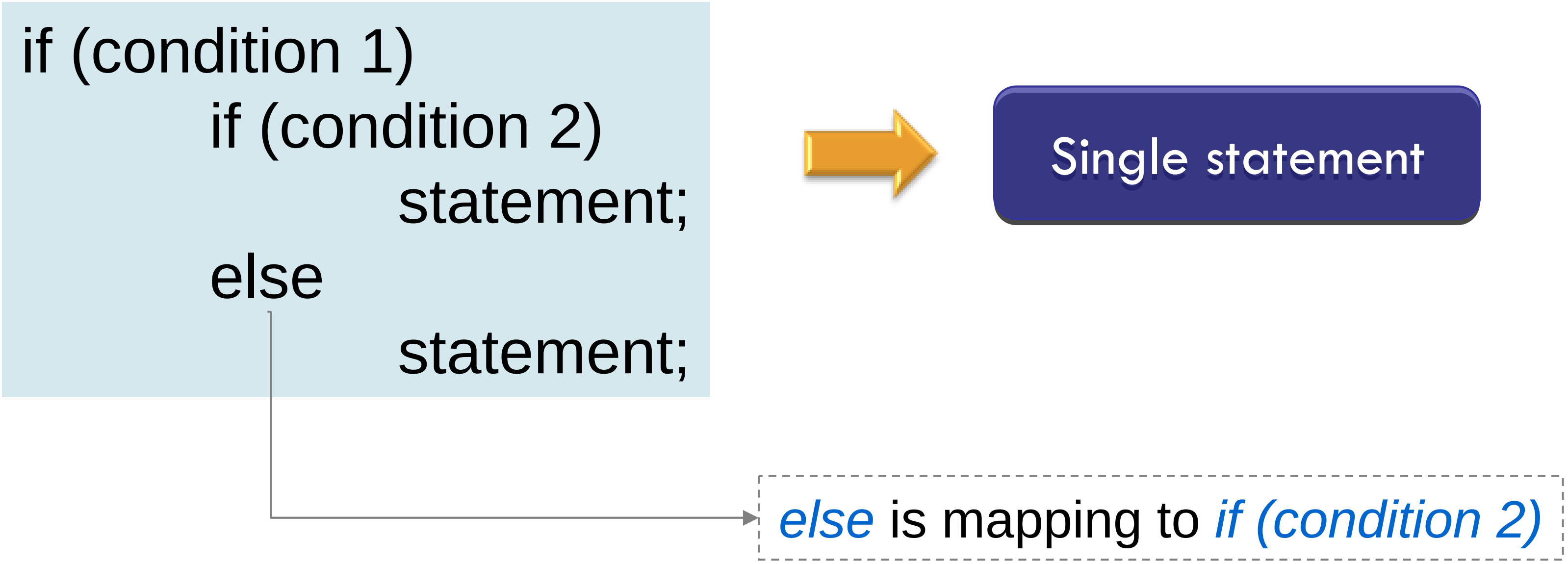
grade>=50

true

Re-exam

Mapping *else* to *if*

- *else* is mapped to the closest *if*



```
if (condition 1)
        if (condition 2)
                statement;
        else
                statement;
```

Single statement

else is mapping to *if (condition 2)*

Mapping *else* to *if* (Cont.)

- Use brace “{}” to map *else* to further *if*

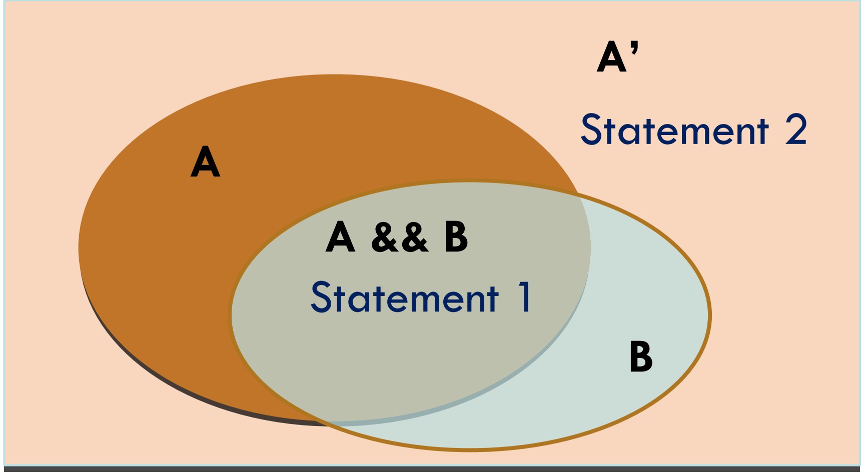
```
if(condition 1)
{
        if(condition 2)
                statement;
}
else
        statement;
```

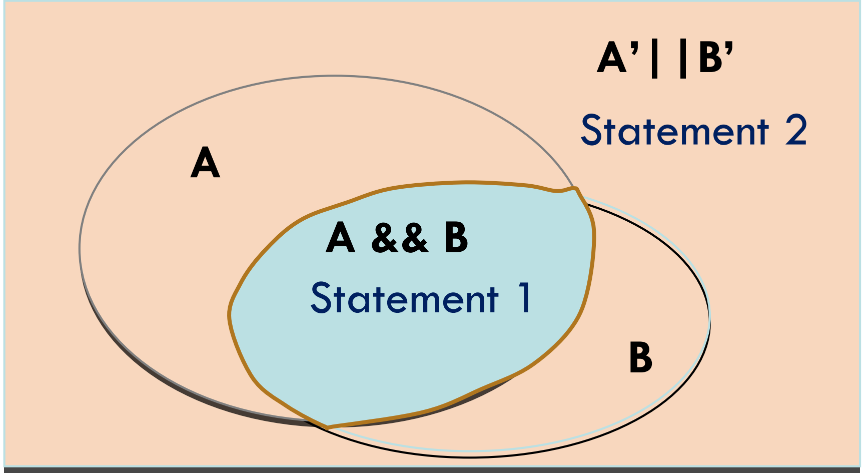
else is mapping to *if (condition 1)*

Comparison

```
• if (condition A)
  {
        if (condition B)
              statement 1;
  }
  else
        statement 2;
```

```
• if (condition A && B)
  {
        statement 1;
  }
  else
        statement 2;
```





Switch

- Useful when a variable or expression is tested such that different actions are taken for each value
- Syntax
  - switch(variable)

```
switch(variable)
{
    case ‘value 1’:
                actions;        // executed if variable=‘value 1’
                break;          // exit from the switch statement
    case ‘value 2’:
                actions;
                 break;
    …
    default:
                actions;
}
```

Can only be int or char

switch: Example

```
int year;
scanf(“%d”, &year);
switch(year) {
        case 1:
                printf(“You are a freshman\n”);
                break;
        case 2:
                printf(“You are a sophomore\n”);
                break;
        case 3:
                printf(“You are a junior\n”);
                break;
        case 4:
                printf(“You are a senior\n”);
                break;
        default:
                printf(“You typed a wrong number\n”);
}
```

Switch without break

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
   char grade;
   printf("Input grade:");
   scanf("%c",&grade);

   switch(grade)
   {
      case 'a':  /* 輸入 a 或 A 時印出 Excellent! */
      case 'A':
         printf("Excellent!\n");
      case 'b':  /* 輸入 b 或 B 時印出 Good! */
      case 'B':
         printf("Good!\n");
      case 'c':  /* 輸入 c 或 C 時印出 Be study hard! */
      case 'C':
         printf("Be study hard!\n");
      default:   /* 輸入其他字元時印出 Failed! */
         printf("Failed!\n");
   }
   system("pause");
   return 0;
}
```

```
/* OUTPUT--

Input grade:b
Good!
Be study hard!
Failed!
-------------------------*/
```

break

- Exit from the switch statement
- How to map multiple values to the same action?

```
char grade;
scanf(“%c”, &grade);
switch(grade)
{
        case ‘A’:
        case ‘a’:
                printf(“excellent!!\n”);      // print “excellent” if a or A
                break;
        case ‘B’:
        case ‘b’:
                printf(“Good.\n”);            // print “good” if b or B
                break;
        default:
                printf(“Please work harder…\n”);
}
```

goto

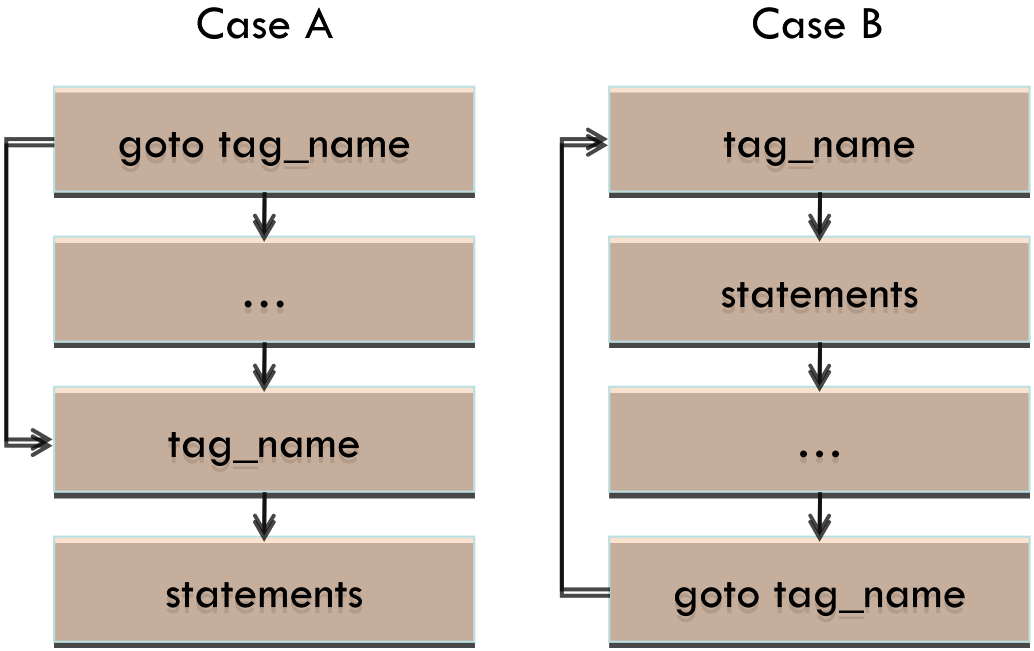
- Force the program to jump to the specified line
- Syntax

```
goto tag_name;
…
tag_name:
         statements;
```

Case A

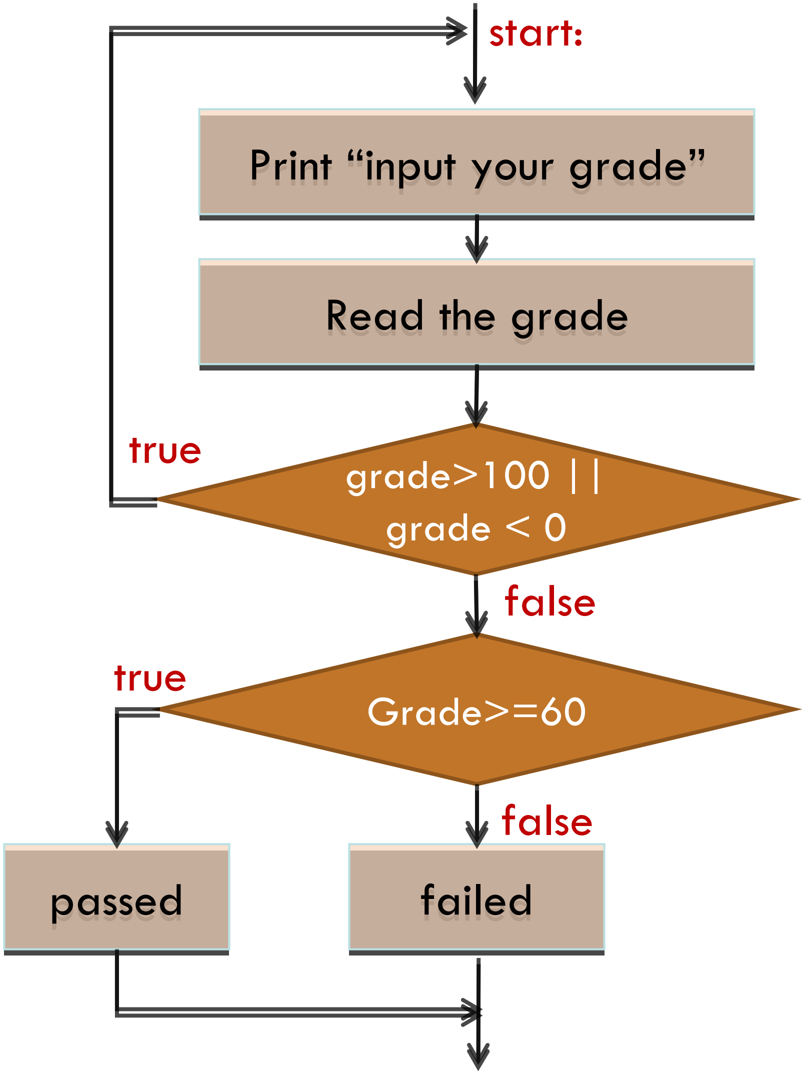
```
tag_name:
         statements;
…
goto tag_name;
```

Case B



goto: Example

```
int grade;
start:
        printf(“input your grade: ”);
        scanf(“%d”, &grade);
        if(grade > 100 || grade < 0)
                goto start;
        if(grade >= 60)
                printf(“passed\n”);
        else
                printf(“failed\n”);
```



Disadvantages of goto

- Destroy the sequential structure of a program.
- Recommendation
  - Can be replaced by other statements.
  - Avoid using *goto.*

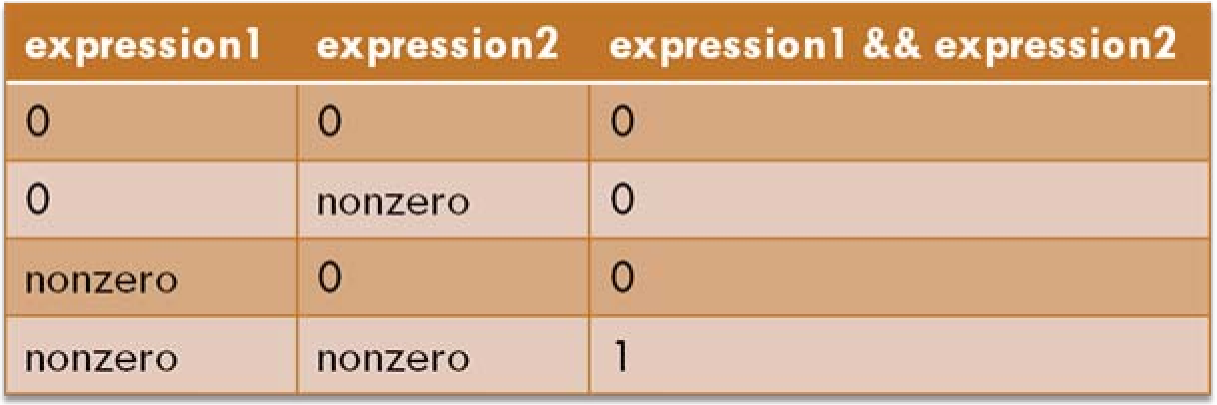
Logical Control (Review)

Logical Operators

- && (and)
  - A && B: return true of both of conditions A and B are true
- || (or)
  - A || B: return true of either of conditions A and B is true
- ! (not)
  - !A: return false when condition A is true

&& (and)

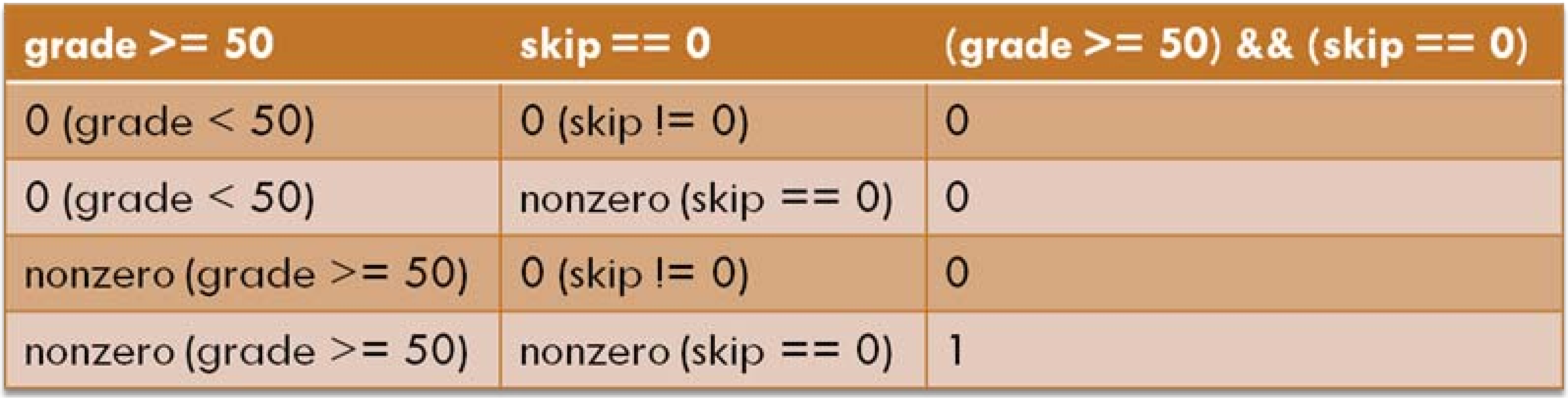
Truth table for &&



expression1	expression2	expression1 && expression2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

&& (and): Example

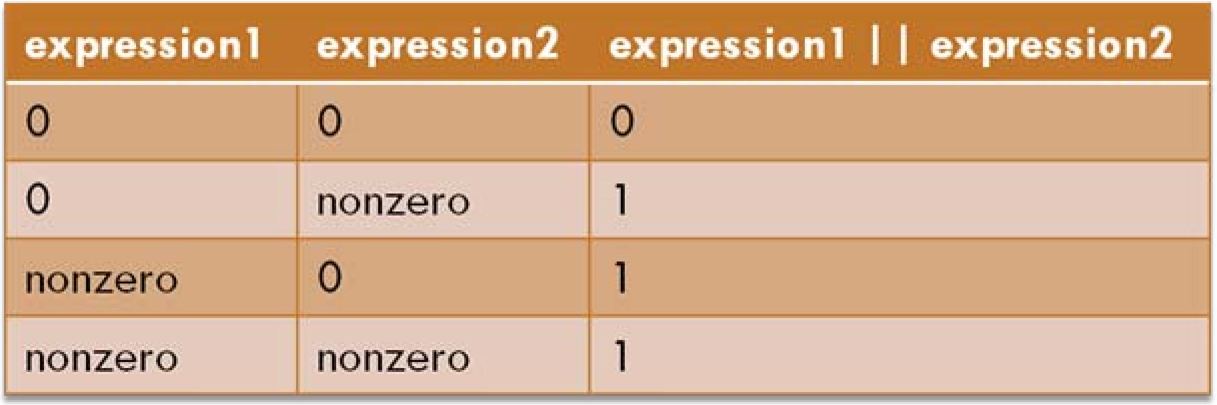
Truth table for &&



grade >= 50	skip == 0	(grade >= 50) && (skip == 0)
0 (grade < 50)	0 (skip != 0)	0
0 (grade < 50)	nonzero (skip == 0)	0
nonzero (grade >= 50)	0 (skip != 0)	0
nonzero (grade >= 50)	nonzero (skip == 0)	1

|| (or)

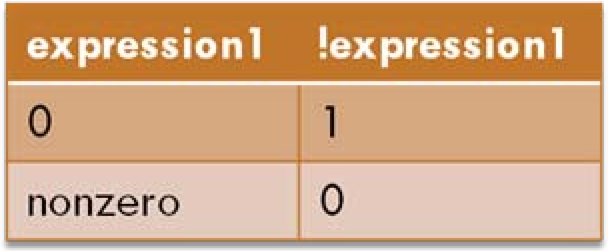
Truth table for | |



expression1	expression2	expression1 | | expression2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

! (not)

Truth table for !



expression1	!expression1
0	1
nonzero	0

Performance Tips

- ((A && B) && C…)
  - Skip checking if it finds one of the expression is false
- ((A || B) || C…)
  - Skip checking if it finds one of the expression is true
- In expressions using operator &&, make the condition that is most likely to be false the leftmost condition.
- In expressions using operator ||, make the condition that is most likely to be true the leftmost condition. This can reduce a program’s execution time.

Comparison Between == and =

- A == B
  - Logical control
  - Check whether A equals B
  - Return true (1) if A is equal to B
  - Return false (0) if A is not equal to B
- A = B
  - Value assignment
  - Assign value B to variable A

Dangerous error

```
if ( a = 4)
  printf("a = 4\n");
```

Lab 06-1: Using if-else

- Request the user to input a grade.
- Request the user to input the number of skips.
- If the grade is higher than or equal to 60, the student passes the exam
  - Print “You passed.”
- If the grade is between 45 and 59, the student can re-exam
  - Print “You can have a makeup exam.”
- If the grade is between 50 and 59 and the number of skip is 0, the student can choose to have a makeup exam or write a report.
  - Print “You can have a makeup exam or write a report.”
- Otherwise
  - Print “You failed.”

Lab 06-2: Using switch

- Four arithmetic operations:
  - Let users input one of the following arithmetical expressions (only allow integer operands). Then use *switch* to detect the operator and print the result on the screen.
    Hint: scanf(“%d %c %d”, &a, &ch, &b);
  - a + b
  - a – b
  - a * b
  - a / b (convert the result to floating point)
  - a % b

Lab 06-3: Salary

- 假設某便利商店的工讀生的月薪資，可以依照下列方式計算：
  - 60個小時之內，每小時75元
  - 61~75個小時，以1.25倍計算
  - 76個小時以後以1.75倍計算
  - 例如，如果工作時數為80小時，則薪資為60*75+15*75*1.25+5*75*1.75=6562.5元。
- 試撰寫一程式，於程式中設定某工讀生該月的工作時數（為一整數），然後計算實領的薪資。

Lab 06-4: Leap Year

- 試撰寫一程式，可由鍵盤讀入一個4個位數的整數，代表西洋的年份，然後判別這個年份是否為閏年 (leap year)。
- ***Hint:*** 每四年一閏，每百年不閏，每四百年一閏，每四千年不閏，例如西元1900雖為4的倍數，但可被100整除，所以不是閏年，同理，2000年是閏年，因可被400整數，而2004當然也是閏年，因可以被4整除。