Lecture 3
Selection (Conditional) Statements

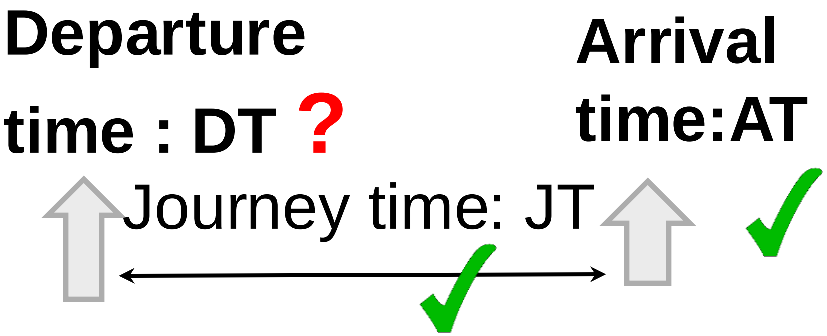
I. Motivation
II.1 The if Selection Statement
II.2 The if…else Selection Statement
II.3 Nested if...else Statements
II.4 Conditional Operator (?:)
II.5 switch Multiple-Selection Statement
Appendix

Ack: slides based on deitel C How to Program, 8/e notes

See the Note sections

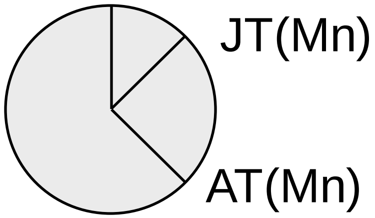
I. Motivation

- Looking back at Q11 of tutorial 2.
- Assume every time variable is given in the format *HrMn*



- To set the values of the *Hr* and *Mn* fields of the *DT:*
  - ❑ If JT(Mn)=0, DT(Hr)=AT(Hr)-JT(Hr)
  - ❑ If JT(Mn) ≠ 0,
    - ▪ If JT(Mn)≤ AT(Mn), then DT(Hr)=AT(Hr)-JT(Hr)

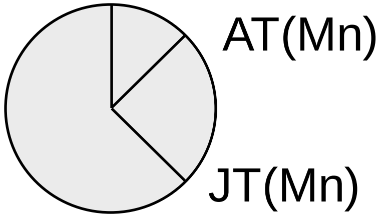
Different actions on different conditions



DT(Mn)=AT(Mn)-JT(Mn)

Else DT(Hr)=AT(Hr)-JT(Hr)-1

DT(Mn)=60+AT(Mn)-JT(Mn)



II. Selection Statements in C

- **Selection (conditional) statements** are used to **choose** among **alternative courses of action**
- C provides **three main types of selection structures** in the form of statements:
  - The **`if`** selection statement: **either** selects (performs) an action if a condition is true or skips the action if the condition is false.
  - The **`if…else`** selection statement: performs an action if a condition is **true** and performs a different action if the condition is **false.**
  - The **`switch`** selection statement: performs **one of many different actions** depending on the value of an **expression**.

…Continued

- The `if` statement is called a single-selection statement because **it selects or ignores a single action.**

- The `if…else` statement is called a double-selection statement **because it selects between two different actions.**

- The `switch` statement is called a multiple-selection statement **because it selects among many different actions.**

II.1 The if Selection Statement

- An example: suppose the passing grade on an exam is 60.
  - The pseudo-code statement

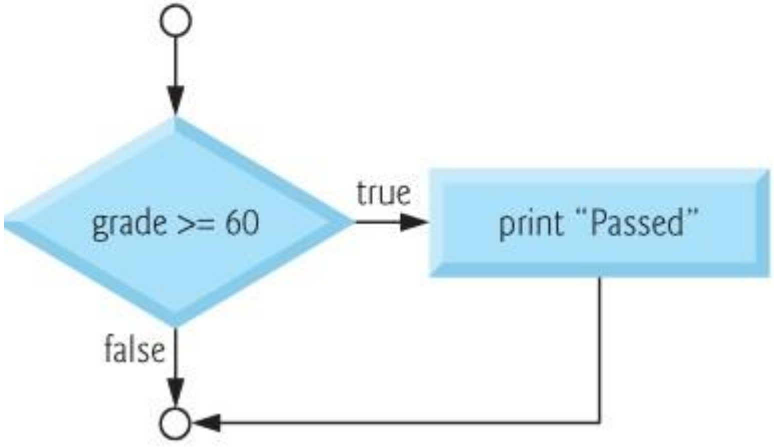
```
– If student's grade is greater than or equal to 60
      Print "Passed"
```

determines whether the condition "student's grade is greater than or equal to 60" is true or false.

- If the condition is **true**, then "Passed" is printed, and the next pseudocode statement in order is "performed"
- If the condition is **false**, the printing is ignored, and the next pseudocode statement in order is performed.

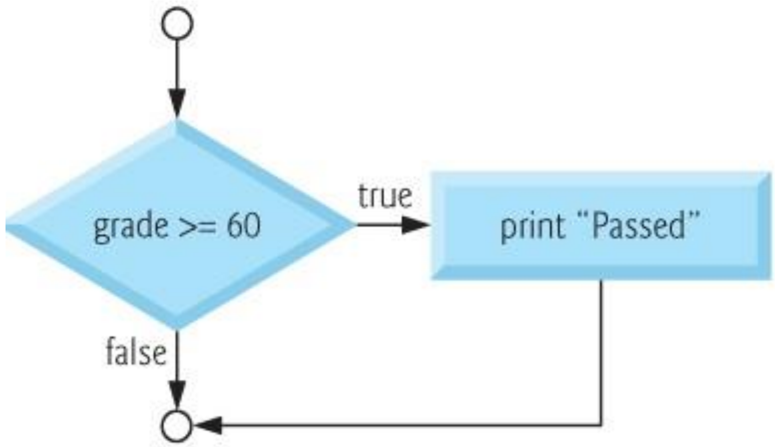
…Continued

- The previous pseudo code can be represented using the flowchart below
  - The **diamond symbol**, also called the **decision symbol**, which indicates that a decision is to be made.
  - The decision symbol **contains an expression**, such as a condition, that can be either **true** or **false**.



  - The decision symbol has *two flowlines* emerging from it. One indicates the direction to take when the expression in the symbol is **true** and the other the direction to take when the expression is **false.**

…Continued



```
int grade;

…..
if ( grade >= 60 ) {
   printf( "Passed\n" );
} // end if
```

C code

❖ In general, a **single selection statement** corresponds in C to

if(*expression*){
  --*statements of your action*
}

- ❑ **If the action consists of one statement, the enclosing curly braces {} can be removed**

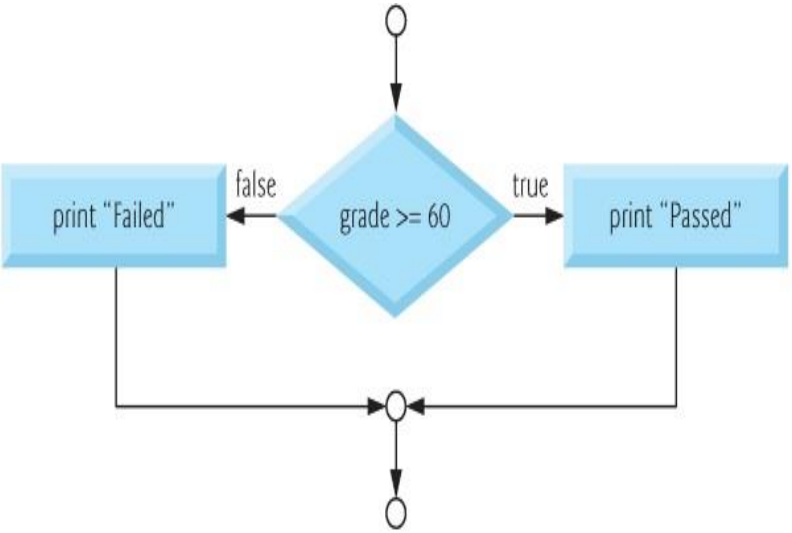
II.2 The `if…else` Selection Statement

- The `if…else` selection statement allows you to specify that *different* actions are to be performed when the condition is **true** and when it's **false.**
  - For example, the pseudocode statement

    ```
    – If student's grade is greater than or equal to 60
            Print "Passed"
      else
            Print "Failed"
    ```

    prints *Passed* if the student's grade is greater than or equal to 60 and *Failed* if the student's grade is less than 60*.*
  - In either case, after printing occurs, the next pseudo-code statement in sequence is "performed."

…Continued



```
int grade;

if ( grade >= 60 ) {
   printf( "Passed\n" );
} // end if
else {
   printf( "Failed\n" );
} // end else
```

C code

- In general, a **double selection statement** corresponds in C to

```
if(expression){
  --statements of your action 1
}
else {
  --statements of your action 2
}
```

II.3 Nested if...else Statements

- Nested **if…else** statements test for **multiple cases** by placing **if…else** statements inside **if…else** statements.

- For example, the following pseudocode statement will print A for exam grades greater than or equal to 90, B for grades greater than or equal to 80 (**but less than 90**), C for grades greater than or equal to 70 (**but less than 80**), D for grades greater than or equal to 60 (**but less than 70**) and F for all other grades.

…Continued

Pseudo code:

```
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”
```

…Continued

- This pseudocode may be written in C as

```
int grade;

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

- If the variable `grade` is greater than or equal to 90, <u>all four conditions will be true, but only the `puts` statement after the first test will be executed.</u>
  - After that `puts` is executed, the `else` part of the "outer" if…else statement is **skipped.**
  - ☞ **The order of IF statements is IMPORTANT**

…Continued

- You may prefer to write the preceding `if` statement as

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

Compound statement in if block

- To include **several statements** in the body of an `if` or `else`, you **must** enclose the set of statements in braces (`{` and `}`), otherwise the statements after the first one WILL ALWAYS executed regardless on the value of the expression.
  - if you have only one statement in the **if**'s body, you do not need the enclose it in braces.

- A set of statements contained within a pair of braces is called a compound statement or a block.

…Continued

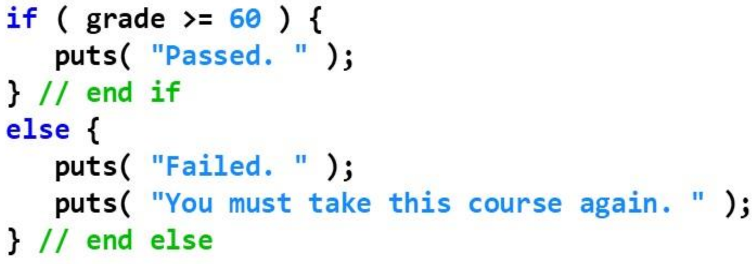
- The following example includes a **compound statement** in the **else** part of an **if…else** statement.

```
if ( grade >= 60 ) {
   puts( "Passed. " );
} // end if
else {
   puts( "Failed. " );
   puts( "You must take this course again. " );
} // end else
```

…Continued

- if grade is less than `60`, the program executes both `puts` statements in the body of the `else` and prints

```
– Failed.
  You must take this course again.
```



```
if ( grade >= 60 ) {
   puts( "Passed. " );
} // end if
else {
   puts( "Failed. " );
   puts( "You must take this course again. " );
} // end else
```

- The braces surrounding the two statements in the `else` are important. Without them, the statement

```
puts( "You must take this course again." );
```

would be outside the body of the `else` part of the `if` and **would execute regardless** of whether the grade was less than 60.

…Continued

- Just as a **compound statement** can be placed anywhere a single statement can be placed, it’s also possible to have **no statement at all**, i.e., the **empty statement.**
  - The **empty statement** is represented by placing a semicolon (**;**) where a statement would normally be.

II.4 Conditional Operator (? :)

- C provides the conditional operator (**?** **:**) which is closely related to the **if…else** statement.
- The conditional operator is C's only **ternary operator**; it takes *three* operands: **cond ? Expr1:expr2**
  - These three operands with the conditional operator (**?** **:**) form a conditional expression.
    - The **first operand** is the **expression condition.**
    - The **second operand** is the **outcome for the entire conditional expression** **if the condition is *true***
    - the **third operand** is **the outcome for the entire conditional expression** **if the condition is *false*.**

…Continued

- For example, the `puts` statement

```
puts( grade >= 60 ? "Passed" : "Failed" );
```

contains as its argument a **conditional expression** that evaluates to the string `"Passed"` if the **condition** `grade >= 60` is **true** and to the string `"Failed"` if the **condition** is **false**.

- The `puts` statement performs in essentially the same way as the preceding **`if…else`** statement.

…Continued

- **The second and third operands** in a **conditional expression** can also **be actions to be executed**.
- For example, the **conditional expression**

```
grade >= 60 ? puts( "Passed" ) : puts( "Failed" );
```

is read, “If `grade` is greater than or equal to `60` then **`puts`**`("Passed")`, otherwise **`puts`**`( "Failed" )`.”

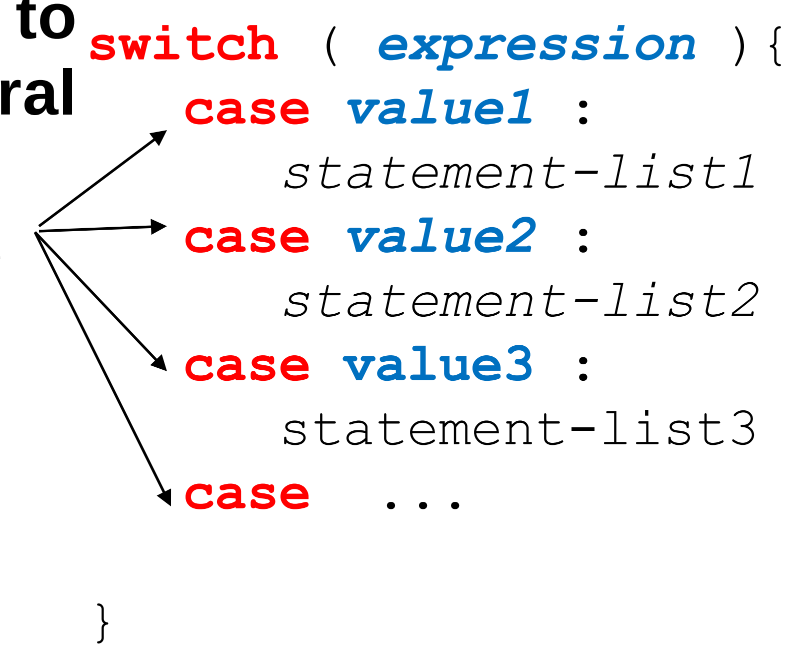
  - ❑ This, too, is comparable to the preceding **`if…else`** statement.

II.5 switch Multiple-Selection Statement

- Occasionally, an algorithm contains a ***series of decisions*** in which a **variable or expression** is tested separately for each of the **constant integral values** it may have, and different actions are to be taken.
  - This is called multiple selection.
- C provides the **`switch`** **multiple-selection** statement to handle such decision making.
- The `switch` statement consists of a series of **`case`** labels, an **optional `default`** case, and statements to execute for each case.

…Continued

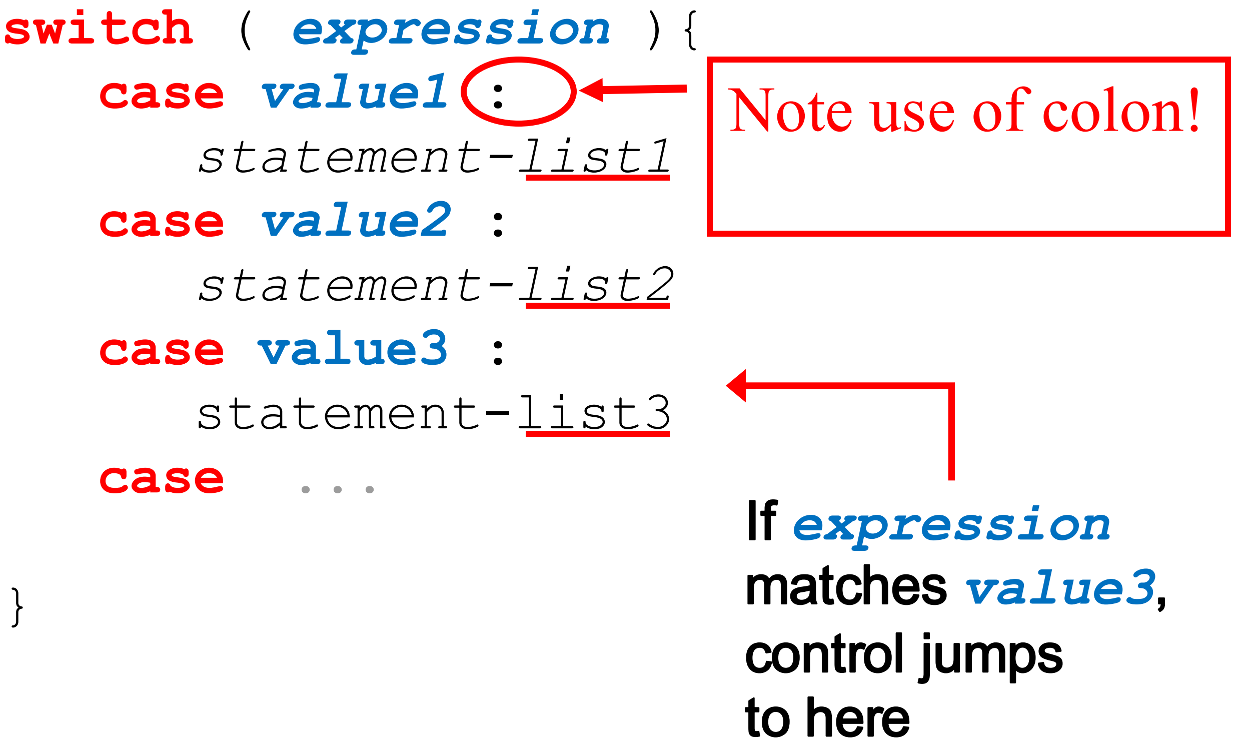
- The `switch` statement evaluates an expression, then **attempts to <u>match its value</u> to one of several possible** case <u>labels/values</u>
- Keyword `switch` is followed by the expression in parentheses.
  - This is called the controlling expression.
- The value of this expression is compared with each of the `case` labels.
- If a match occurs, the statements for that `case` are executed.



```
switch ( expression ){
     case value1 :
          statement-list1
     case value2 :
          statement-list2
     case value3 :
          statement-list3
     case   ...

}
```

…Continued

- Each case contains a **constant integral value** and **a list of statements**
  - The flow of control transfers to the list of statements associated with the case value that matches the expression value



```
switch ( expression ){
    case value1 :
        statement-list1
    case value2 :
        statement-list2
    case value3 :
        statement-list3
    case  ...

}
```

Note use of colon!

If *expression* matches *value3*, control jumps to here

Break in Switch Statement

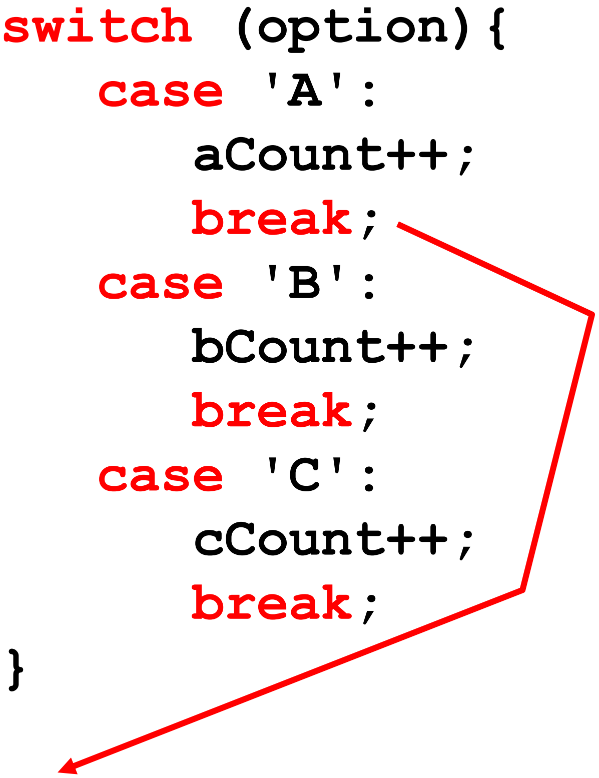
- The **break** *statement* can be used as the last statement in each case's **statement list**

- A `break` statement causes control to transfer to the end of the `switch` statement

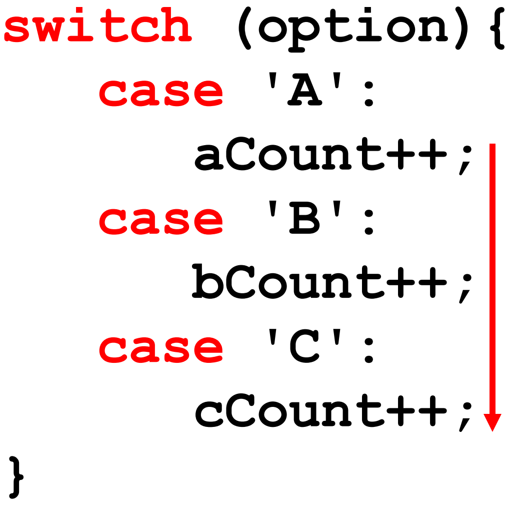
- If a `break` statement is **NOT** used, **the flow of control will continue into the next case**

```
switch ( expression ){
   case value1 :
      statement-list1
      break;
   case value2 :
      statement-list2
      break;
   case value3 :
      statement-list3
      break;
   case  ...

}
```

Example: Break in a Switch-Case



```
switch (option){
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
}
```



```
switch (option){
    case 'A':
        aCount++;
    case 'B':
        bCount++;
    case 'C':
        cCount++;
}
```

Fall through.

default case

- A **switch** statement can have an **optional** **default** *case*

- The default case has **NO associated value** and simply uses the reserved word **default**

- If the **default** case is present, control will transfer to it if NO other case value matches the expression value

- If there is no default case, and no other value matches, control falls through to the statement after the switch

…Continued

```
switch ( expression ){
   case value1 :
      statement-list1
      break;
   case value2 :
      statement-list2
      break;
   case value3 :
      statement-list3
      break;
   default: ...
      statement-default
}
```

- Each case can have one or more actions.
- The switch statement is different from all other control statements in that **braces are not required around the action(s) in a case of a switch.**
- **A switch case action can use any of the C language construct including switch case itself!**

Constraints and Recommendations

- The expression of a **`switch`** statement must result in an ***integral type***, meaning an integer (**`byte, short, int`**) or a **`char (or a Boolean value`** `although it will generate a` **`warning`**`)`
  - It **cannot** be a **floating point** value (`float` or `double`)
- The implicit boolean condition in a `switch` statement is **equality**
  - If needed, put the relational check in the `switch` expression
- The values should be a **constant integral value**

…Continued

- No two `case` labels may **have the same value.**
- Two `case` labels may be associated with the same statements.
- The `default` label is not required but sometimes preferable
  - values not explicitly tested in a switch would normally be ignored. The default case helps prevent this by focussing you on the need to process exceptional conditions.
- There can be only one `default` label, and it is usually **last.**
  - Although the **case** clauses and the **default** case clause in a **switch** statement **can occur in any order**, it's common to place the default clause last
- In a **switch** statement, when the **default clause** is last, the **break** statement **is not required**. You may prefer to include this break for clarity and symmetry with other cases.
- **Listing several** **case labels** **together** simply means that the **same set of actions is to occur <u>for either of these cases.</u>**

Appendix: Example 1

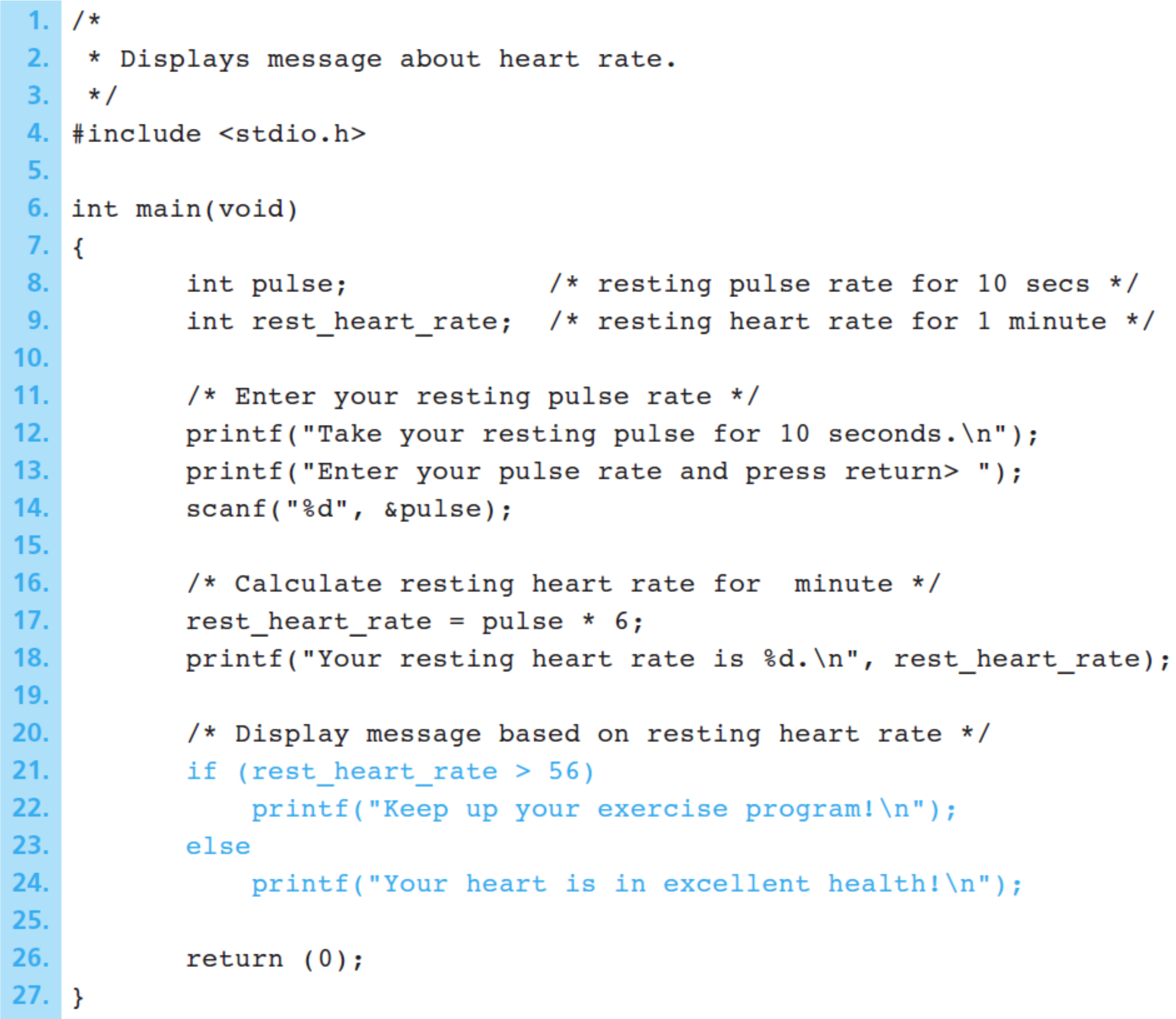
- Write a program that asks a user to enter his resting pulse rate. If his resting heart rate (per minute) is above 56, then display a message “Keep up your exercise program”. Otherwise, the program should display “Your heart is in excellent condition”.

A sample run of the program is shown below:

```
Sample Run 1
Take your resting pulse for 10 seconds.
Enter your pulse rate and press return> 12
Your resting heart rate is 72.
Keep up your exercise program!
```

```
Sample Run 2
Take your resting pulse for 10 seconds.
Enter your pulse rate and press return> 9
Your resting heart rate is 54.
Your heart is in excellent health!
```

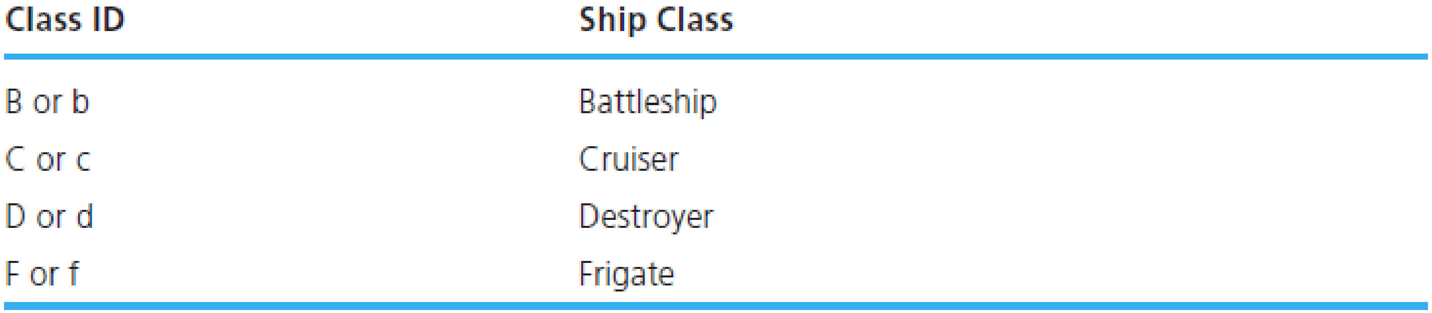
Solution



```
/*
 * Displays message about heart rate.
 */
#include <stdio.h>

int main(void)
{
       int pulse;            /* resting pulse rate for 10 secs */
       int rest_heart_rate;  /* resting heart rate for 1 minute */

       /* Enter your resting pulse rate */
       printf("Take your resting pulse for 10 seconds.\n");
       printf("Enter your pulse rate and press return> ");
       scanf("%d", &pulse);

       /* Calculate resting heart rate for  minute */
       rest_heart_rate = pulse * 6;
       printf("Your resting heart rate is %d.\n", rest_heart_rate);

       /* Display message based on resting heart rate */
       if (rest_heart_rate > 56)
           printf("Keep up your exercise program!\n");
       else
           printf("Your heart is in excellent health!\n");

       return (0);
}
```

Example 2

- Write a program that reads a ship’s serial number and displays the class of the ship. Each ship serial number begins with a letter indicating the class of the ship. The program first reads the first letter of a ship’s serial number into the char variable *class* and then displays that character. The switch statement displays a message indicating the class of the ship. It implements the following decision table.



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

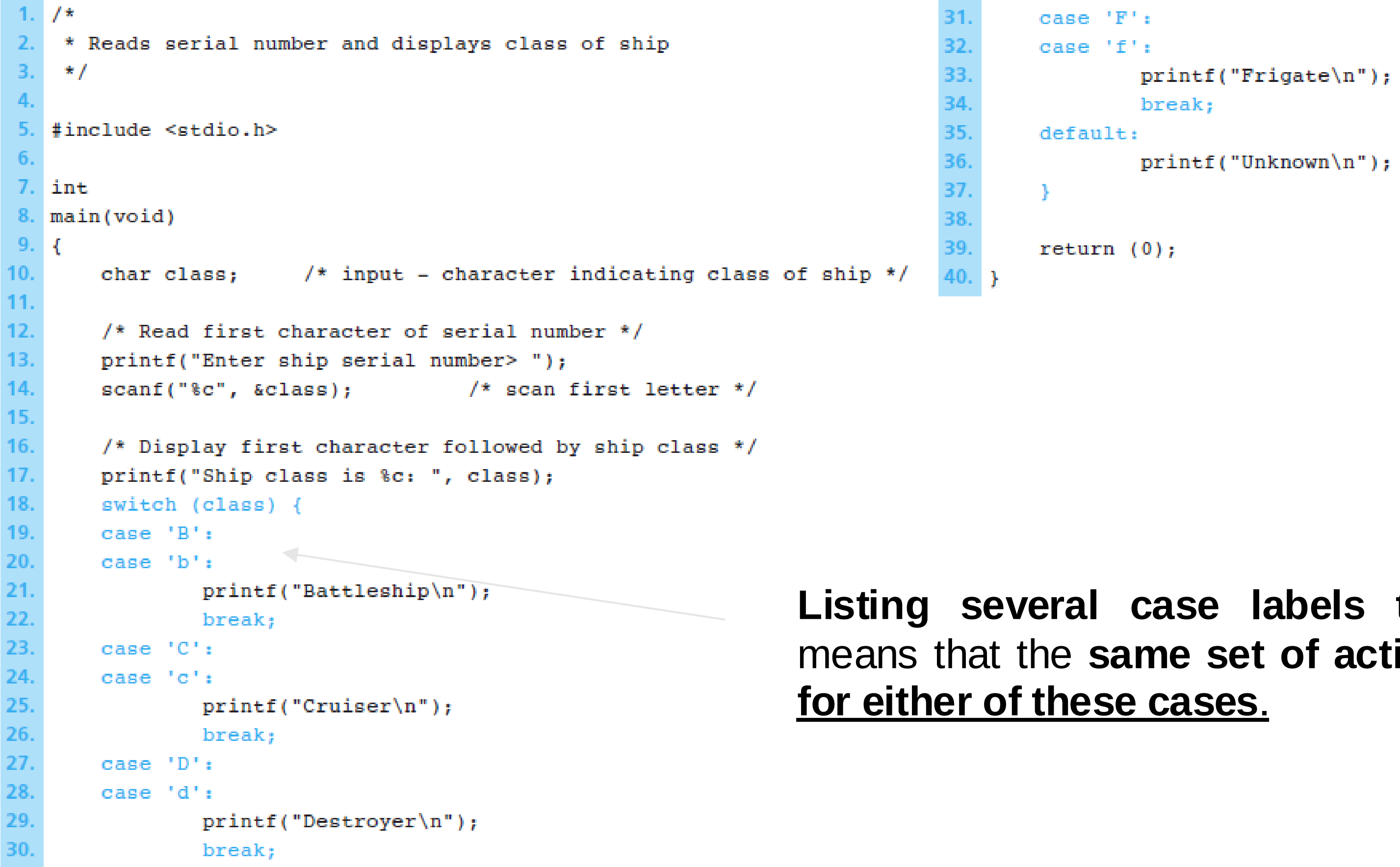
Sample Run 1

```
Enter ship serial
number> f3456
Ship class is f: Frigate
```

Sample Run 2

```
Enter ship serial number>
P210
Ship class is P: Unknown
```

Solution



```c
/*
 * Reads serial number and displays class of ship
 */

#include <stdio.h>

int
main(void)
{
    char class;    /* input - character indicating class of ship */

    /* Read first character of serial number */
    printf("Enter ship serial number> ");
    scanf("%c", &class);        /* scan first letter */

    /* Display first character followed by ship class */
    printf("Ship class is %c: ", class);
    switch (class) {
    case 'B':
    case 'b':
            printf("Battleship\n");
            break;
    case 'C':
    case 'c':
            printf("Cruiser\n");
            break;
    case 'D':
    case 'd':
            printf("Destroyer\n");
            break;
    case 'F':
    case 'f':
            printf("Frigate\n");
            break;
    default:
            printf("Unknown\n");
    }

    return (0);
}
```

Listing several case labels together simply means that the **same set of actions is to occur <u>for either of these cases.</u>**