مبانی برنامه نویسی
به زبان سی

۲۲ آبان ۱۳۹۹

جلسه چهارم

ملکی مجد

مباحث این جلسه:

- پاسخ تمرین BMI
- الگوریتم
- شبه کد
- ساختار کنتری
- ساختار تصمیم گیری If else

نوشتن برنامه محاسبه BMI و تشخیص محدوده آن

برنامه ای بنویسید که از کاربر به ترتیب وزن به کیلوگرم و قد به متر را بگیرد.
طبق فرمول زیر BMI او را محاسبه کند و نمایش بدهد.

$$BMI = \frac{weightInKilograms}{heightInMeters \times heightInMeters}$$

طبق جدول زیر به کاربر اعلام کند که در کدام دسته قرار می گیرد.

```
BMI VALUES
Underweight: less than 18.5
Normal:      between 18.5 and 24.9
Overweight:  between 25 and 29.9
Obese:       30 or greater
```

3.1 Introduction

- Before writing a program to solve a particular problem, it's essential to have **a thorough understanding** of the problem and a **carefully planned approach** to solving the problem.

- We discuss techniques that facilitate the development of **structured** computer programs.

3.2 Algorithms

- The solution to any computing problem involves executing a series of actions in a specific order.
- A procedure for solving a problem in terms of
  - the actions to be executed, and
  - the order in which these actions are to be executed
- is called an algorithm.
- Correctly specifying the order in which the actions are to be executed is important.

3.2 Algorithms (Cont.)

- Consider the "rise-and-shine algorithm" followed by one junior executive for getting out of bed and going to work:
- (1) Get out of bed, (2) take off pajamas, (3) take a shower, (4) get dressed, (5) eat breakfast, (6) carpool to work.
- This routine gets the executive to work well prepared to make critical decisions.

3.2 Algorithms (Cont.)

- Suppose that the same steps are performed in a **slightly different order**:
  - (1) Get out of bed, (2) take off pajamas, (3) get dressed, (4) take a shower, (5) eat breakfast, (6) carpool to work.
  - In this case, our junior executive shows up for work soaking wet.
- Specifying the order in which statements are to be executed in a computer program is called program control.

3.3 Pseudocode

- Pseudocode is an **artificial and informal language** that helps you develop algorithms.
- Pseudocode is similar to **everyday English**;
  - it's convenient and user friendly although it's **not an actual computer programming** language.
- Pseudocode programs are not executed on computers.
- Rather, they merely help you "***think out***" a program before attempting to write it in a programming language such as C.
- Pseudocode consists purely of characters, so you may conveniently type pseudocode programs into a computer using an editor program.

3.3 Pseudocode (Cont.)

- A carefully prepared pseudocode program may be **converted easily** to a corresponding C program.
- Pseudocode consists **only of action statements**—those that are executed when the program has been converted from pseudocode to C and is run in C.
- **Definitions** are not executable statements.
  - They're messages to the compiler.

3.3 Pseudocode (Cont.)

- For example, the definition
  - `int i;`
  - simply tells the compiler the type of variable `i` and instructs the compiler to reserve space in memory for the variable.
- But this definition does not cause any action—such as input, output, or a calculation—to occur when the program is executed.

- Some programmers choose to list each variable and briefly mention the purpose of each at the beginning of a pseudocode program.

3.4 Control Structures

- Normally, statements in a program are executed one after the other in the order in which they're written.
- This is called sequential execution.

- Various C statements we'll soon discuss enable you to specify that the next statement to be executed may be other than the next one in sequence.
  - This is called transfer of control.

  - During the 1960s, it became clear that the indiscriminate use of transfers of control was the root of a great deal of difficulty experienced by software development groups.

3.4 Control Structures (Cont.)

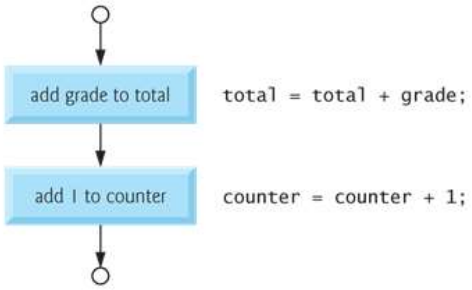
- The finger of blame was pointed at the `goto` statement that allows programmers to specify a transfer of control to one of many possible destinations in a program.
- The notion of so-called structured programming became almost synonymous with "`goto` elimination."
- Research had demonstrated that programs could be written without any `goto` statements.

3.4 Control Structures (Cont.)

- Research had demonstrated that all programs could be written in terms of only three control structures, namely
  - the sequence structure,
  - the selection structure and
  - the repetition structure.

3.4 Control Structures (Cont.)

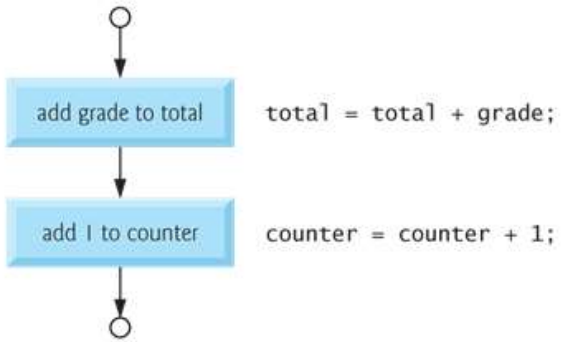
- the computer executes C statements one after the other in the order in which they're written.
- A flowchart is a graphical representation of an algorithm or of a portion of an algorithm.
- Flowcharts are drawn using certain special-purpose symbols such as rectangles, diamonds, ovals, and small circles; these symbols are connected by arrows called flowlines.
  - The flowchart segment of Fig. 3.1 illustrates C's sequence structure.



3.4 Control Structures (Cont.)

- Like pseudocode, flowcharts are useful for developing and representing algorithms, although pseudocode is preferred by most programmers.

- We use the rectangle symbol, also called the action symbol, to indicate any type of action including a calculation or an input/output operation.
  - The flowlines in the figure indicate the order in which the actions are performed—first, `grade` is added to `total`, then `1` is added to `counter`.



3.4 Control Structures (Cont.)

- When drawing a flowchart that represents a complete algorithm, an oval symbol containing the word "Begin" is the first symbol used in the flowchart; an oval symbol containing the word "End" is the last symbol used.

- When drawing only a portion of an algorithm, the oval symbols are omitted in favor of using small circle symbols, also called connector symbols.

- Perhaps the most important flowcharting symbol is the diamond symbol, also called the decision symbol, which indicates that a decision is to be made.

3.4 Control Structures (Cont.)

- C provides three types of selection structures in the form of statements.

1. The `if` selection statement either performs (selects) an action if a condition is true or skips the action if the condition is false.
2. The `if...else` selection statement performs an action if a condition is true and performs a different action if the condition is false.
3. The `switch` selection statement performs one of many different actions depending on the value of an expression.

3.4 Control Structures (Cont.)

- C has only seven control statements: sequence, three types of selection and three types of repetition.

- 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.
- C provides three types of repetition structures in the form of statements, namely `while`, `do…while`, and `for` (discussed later).
- That's all there is.

3.4 Control Structures (Cont.)

- As with the sequence structure of Fig. 3.1, we'll see that the flowchart representation of each control statement has two small circle symbols, one at the entry point to the control statement and one at the exit point.
- These single-entry/single-exit control statements make it easy to build programs.

3.4 Control Structures (Cont.)

- The control-statement flowchart segments can be attached to one another by connecting the exit point of one control statement to the entry point of the next.
- We'll learn that there is only one other way control statements may be connected—a method called control-statement nesting.

- Thus, any C program we'll ever need to build can be constructed from only seven different types of control statements combined in only two ways.
  - This is the essence of simplicity.

3.5 The `if` Selection Statement

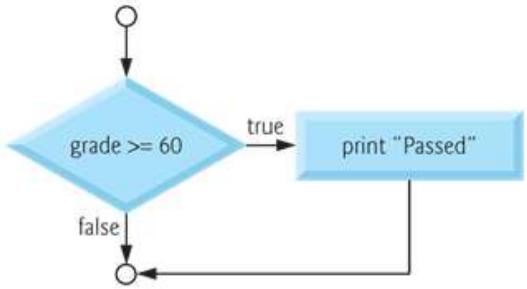
Good Programming Practice 3.2

Pseudocode is often used to "think out" a program during the program design process. Then the pseudocode program is converted to C.

- The pseudocode statement
  - If student's grade is greater than or equal to 60
    Print "Passed"

- The preceding pseudocode *If statement may be written in C as*

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



3.5 The `if` Selection Statement (Cont.)

- In C:
- A decision can be based on any expression—if the expression evaluates to zero, it's treated as false, and if it evaluates to nonzero, it's treated as true.

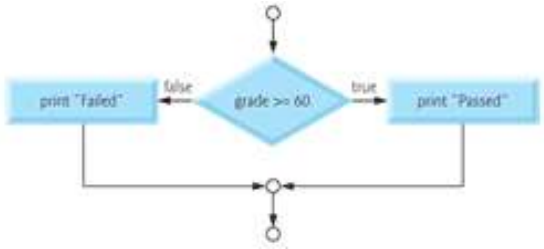
3.5 The `if` Selection Statement (Cont.)

- `The if statement, too, is a single-entry/single-exit structure.`
- We can envision seven bins, each containing only control-statement flowcharts of one of the seven types.
- These flowchart segments are empty—nothing is written in the rectangles and nothing is written in the diamonds.
- Your task, then, is assembling a program from as many of each type of control statement as the algorithm demands, combining those control statements in only two possible ways (stacking or nesting), and then filling in the actions and decisions in a manner appropriate for the algorithm.

3.6 The `if...else` Selection Statement

- For example, the pseudocode statement
  - If student's grade is greater than or equal to 60
    Print "Passed"
    else
    Print "Failed"
- The preceding pseudocode *If...else statement may be written in C as*

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



3.6 The `if...else` Selection Statement (Cont.)

- 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.
  - The operands together with the conditional operator form a conditional expression.
- The first operand is a condition.
- The second operand is the value for the entire conditional expression if the condition is true and
- the third operand is the value for the entire conditional expression if the condition is false.

3.6 The `if...else` Selection Statement (Cont.)

- For example, the `printf` statement
  - `printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );`

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

- The format control string of the `printf` contains the conversion specification `%s` for printing a character string.

3.6 The `if...else` Selection Statement (Cont.)

- The second and third operands in a conditional expression can also be actions to be executed.
- For example, the conditional expression
  - `grade >= 60 ? printf( "Passed\n" ) : printf( "Failed\n" );`

  - is read, "If `grade` is greater than or equal to `60` then `printf("Passed\n")`, otherwise `printf( "Failed\n" )`."

3.6 The `if...else` Selection Statement (Cont.)

- 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`, `C` for grades greater than or equal to `70`, `D` for grades greater than or equal to `60`, and `F` for all other grades.
  - 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"

3.6 The `if...else` Selection Statement (Cont.)

- This pseudocode may be written in C as

```
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" );
```

- Many C programmers prefer to write the preceding `if` statement as

```
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" );
```

3.6 The `if…else` Selection Statement (Cont.)

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

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

اگر {} ها را حذف کنیم چه اتفاقی می افتد؟

3.6 The `if...else` Selection Statement (Cont.)

- A syntax error is caught by the compiler.
- A logic error has its effect at execution time.
- A fatal logic error causes a program to fail and terminate prematurely.
- A nonfatal logic error allows a program to continue executing but to produce incorrect results.