UEE1302
Introduction to Computers and Programming

C_Lecture 03:

Control Statements: Part II

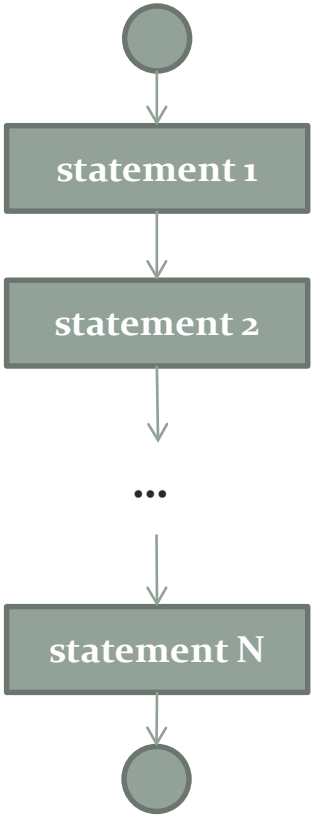
(do..while, for, and switch)

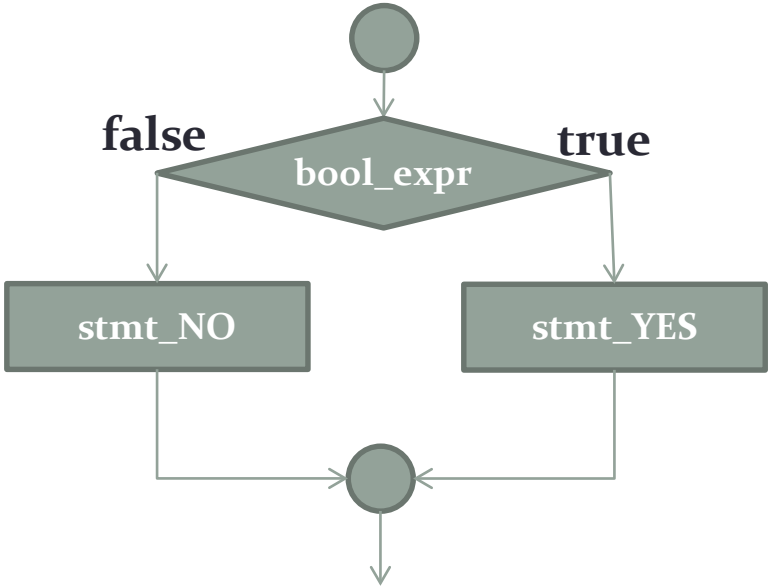
C: How to Program 7th ed.

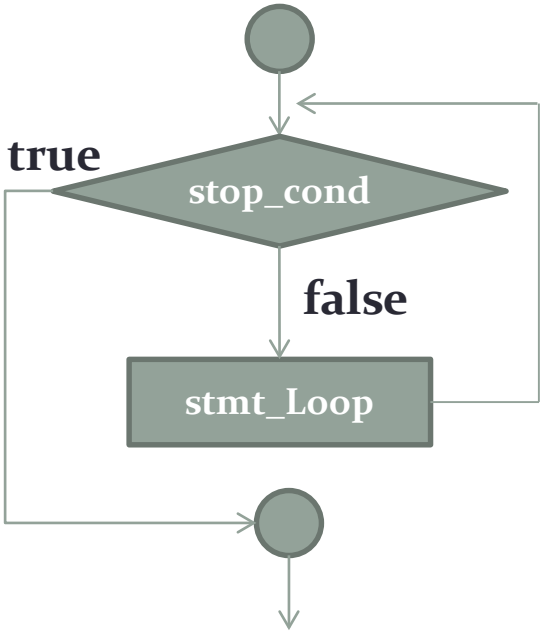
Agenda

- `for` statement
- `do…whlie` statement
- `switch` statement
- Examine `break` and `continue` statements
- Logic Expressions

Flow of Execution







(1) Sequence **(2) Selection** **(3) Repetition**

Flow of Control: Repetition

- `while` loop

```
while ( i < 5 ) {
    printf("good!\n");
    i++;
}
```

- `do/while` loop

```
do {
    printf("good!\n");
    i++;
} while ( i < 5 );
```

Flow of Control: Repetition (cont.)

- `for` loop

```
for ( i = 0; i < 5; i++ ) {
    printf("good!\n");
}
```

while (1): Counter-Controlled

- If you know exactly how many pieces of data need to be read, the `while` loop becomes a counter-controlled loop
- Example of counter-controlled loop:

```
counter = 0; // initialize loop control
while (counter < Limit)// test loop control
{
    statements;
    …
    counter++; // update loop control
}
```

while (2): Sentinel-Controlled

- Sentinel variable is tested in the condition and loop ends when sentinel is encountered
- Example of sentinel-controlled loop:

```
scanf("%d", &target); // initialize loop control
while (target != sentinel) //test loop control
{
    statements;
    // update loop control
    scanf("%d", &target);
}
```

Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires
  - the name of a control variable (or loop counter)
    - with integer values
  - the initial value of the control variable
  - the increment (or decrement) by which the control variable is modified each time through the loop
  - the condition that tests for the final value of the control variable (i.e., whether looping should continue)

Example of Counter-Controlled Loop

```c
// Fig. 4.1: fig04_01.c
// Counter-controlled repetition
#include <stdio.h>

int main( void )
{
    int counter = 1; // initialization

    while ( counter <= 10 ) // repetition condition
    {
        printf( "%d\n", counter); // display counter
        ++counter; // increment
    }

    return 0;
}
```

Example of Counter-Controlled Loop (cont.)

- screen output

```
1
2
3
4
5
6
7
8
9
10
```

`for` Repetition Statement

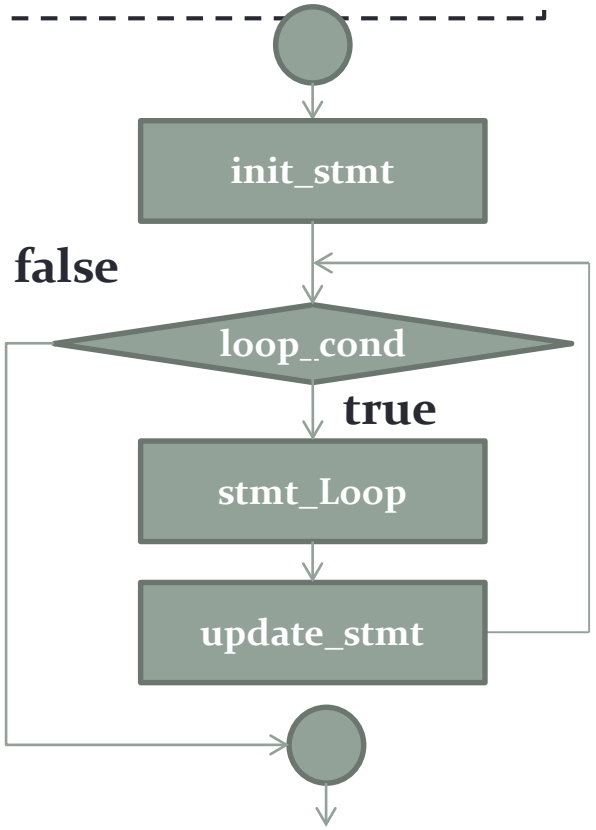
- The `for` repetition statement specifies the counter-controlled repetition details in a single line of code.
  - The initialization occurs once when the loop is encountered.
  - The condition is tested next and each time the body completes.
  - The body executes if the condition is true.
  - The increment occurs after the body executes.
  - Then, the condition is tested again.
- If there is more than one statement in the body of the for, braces are required to enclose the body of the loop.

`for` Repetition Statement (cont.)

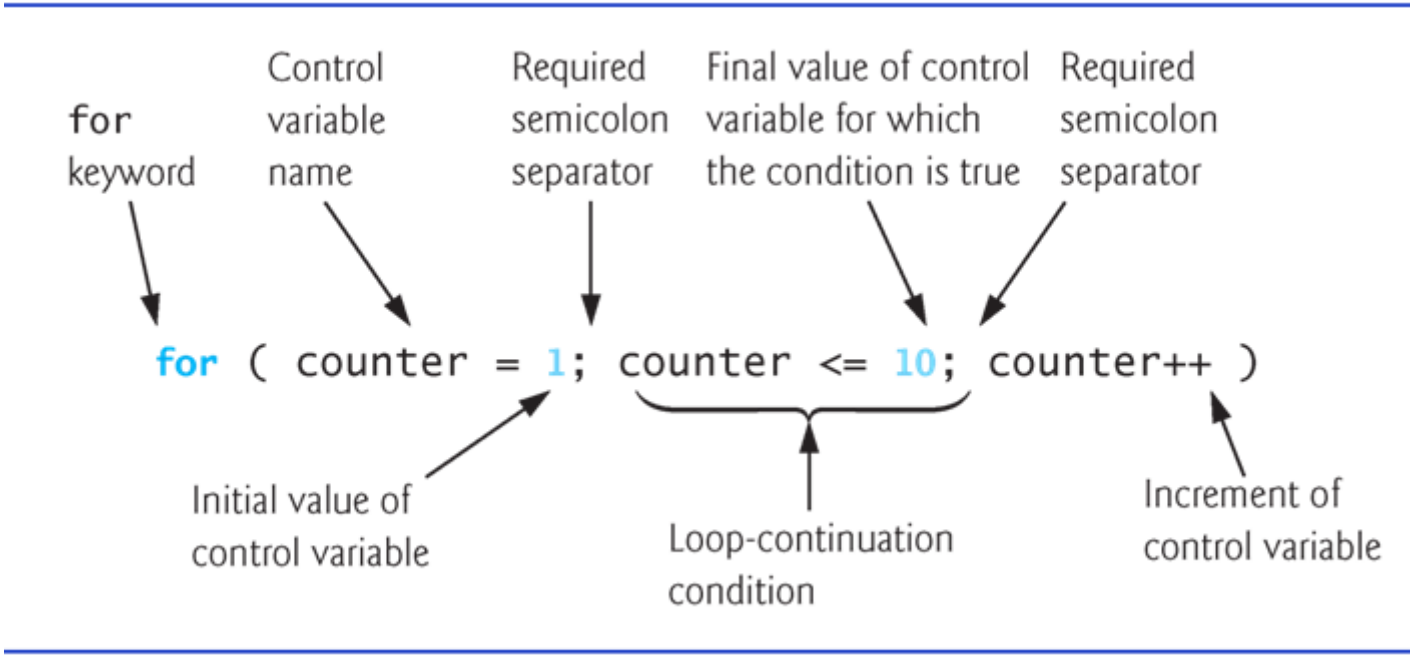
- Syntax of `for` statement:

```
for( init_stmt; loop_cond; update_stmt )
    stmt_for;
```

  - `init_stmt`: initialize ctrl variable
  - `loop_cond`: compare ctrl variable
  - `update_stmt` : update ctrl variable



`for` Statement Header Components



for vs. while

- Rewrite `for` statement in `while` loop

```
init_stmt;
while ( loop_cond )
{
    stmt_for;
    update_stmt;
}
```

- Rewrite `while` statement in `for` loop

```
for ( ; entry_cond; )
{
    stmt_while;
}
```

`for` vs. `while` (cont.)

- Components of `for` statement are optional but semicolons "`;`" must always be present
  - omissions => doing nothing in init_stmt and update_stmt or being true in loop_cond
  - Ex: `for ( ; ; ) { statements;}` is valid
- Differentiate `for` from `while`:
  - initialization statements grouped as first set of items init_stmt within the `for`'s ()
  - loop condition and loop statements are fixed: no change in `for`
  - update statements as the last set of items update_stmt within the `for`'s ()

for Loop Example 1

- Example 1:

```
for ( count = 2; count <= 20; count += 2 ) {
    printf( "%d ", count);
}
```

- Modified Example 1a:

```
count = 2;
for ( ;  count <= 20 ;  count += 2 ) {
    printf( "%d ", count);
}
```

for Loop Example 1 (cont.)

- Example 1:

```
for ( count = 2; count <= 20; count += 2 ) {
    printf( "%d ", count);
}
```

- Modified Example 1b:

```
count = 2;
for ( ; count <= 20; ) {
    printf( "%d ", count);
    counter += 2;
}
```

for Loop Example 1 (cont.)

- Example 1:

```
for ( count = 2; count <= 20; count += 2 ) {
    printf( "%d ", count);
}
```

- Modified Example 1c:

```
for ( count = 2; count <= 20; printf( "%d ",
count), counter += 2 ) {
        ;
}
```

for Loop Example 2

- Example 2:

```
for ( i = 1;  i <= 5;  i++ ) {
    printf( "Happy!\n");
    printf( "*\n");
}
```

- Modified Example 2a:

```
for ( i = 1;  i <= 5;  i++ )
    printf( "Happy!\n");
    printf( "*\n");
```

for Loop Example 2 (cont.)

- Example 2:

```
for ( i = 1;  i <= 5;  i++ ) {
    printf( "Happy!\n");
    printf( "*\n");
}
```

- Modified Example 2b:

```
for ( i = 1;  i <= 5;  i++ );
    printf( "Happy!\n");
    printf( "*\n");
```

Examples using the `for` Statement

- Vary the control variable from 1 to 100 in increments of 1.

```
for ( i = 1; i <= 100; i++ )
```

- Vary the control variable from 100 down to 1 in decrements of 1.

```
for ( i = 100; i >= 1; i-- )
```

- Vary the control variable from 7 to 77 in steps of 7.

```
for ( i = 7; i <= 77; i += 7 )
```

- Vary the control variable from 20 down to 2 in steps of -2.

```
for ( i = 20; i >= 2; i -= 2 )
```

- Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17.

```
for ( i = 2; i <= 17; i += 3 )
```

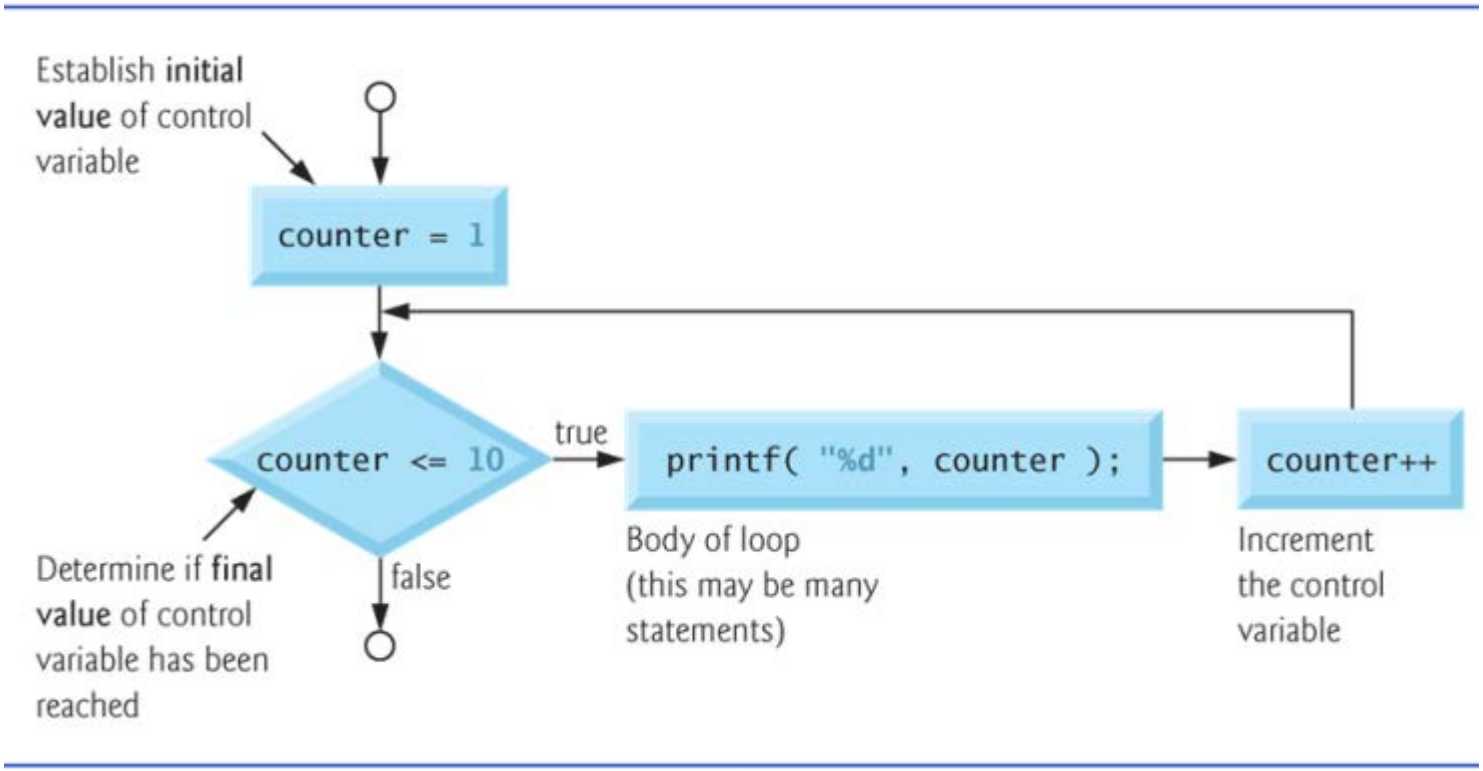
- Vary the control variable over the following sequence of values: 99, 88, 77, 66, 55.

```
for ( i = 99; i >= 55; i -= 11 )
```

Flowchart of a `for` repetition statement

- Example:

```
for ( count = 1; count <= 10; count ++ )
    printf( "%d ", count);
```



Applications – Compound Interest Calculates

- Consider the following problem statement:
  - A person invests $1000.00 in a savings account yielding 5 percent interest. Assuming that all interest is left on deposit in the account, calculate and print the amount of money in the account at the end of each year for 10 years. Use the following formula for determining these amounts:

    $a = p (1 + r)^n$

    where

    p is the original amount invested (i.e., the principal),
    r is the annual interest rate,
    n is the number of years and
    a is the amount on deposit at the end of the nth year.
- This problem involves a loop that performs the indicated calculation for each of the 10 years the money remains on deposit.

Applications (cont.)

```
// Fig. 4.6: fig04_06.c
// calculating compound interest
#include <stdio.h>
#include <math.h>

int main ( void )
{
    double amount; // amount on deposit
    double principal = 1000.0; // starting amount
    double rate = .05; // interest rate
    unsigned int year; // year counter

    // output table column head
    printf( "%4s%21s\n", "Year", "Amount on deposit");
```

Applications (cont.)

```
    // calculate amount on deposit for each of 10 year
    for ( year = 1; year <= 10; year++ ) {

        // calculate
        amount = principal * pow( 1.0 + rate, year );
        // display
        printf( "%4d%21.2f\n", year, amount );
    }

    return 0;
}
```

Applications (cont.)

- screen output:

```
Year      Amount on deposit
   1              1050.00
   2              1102.50
   3              1157.63
   4              1215.51
   5              1276.28
   6              1340.10
   7              1407.10
   8              1477.46
   9              1551.33
  10              1628.89
```

<math.h> header file

- C does not include an exponentiation operator, so we use the standard library function `pow`.
  - `pow(x, y)` calculates the value of x raised to the y^{th} power.
- Takes two arguments of type `double` and returns a `double` value.
- This program will not compile without including header file `<math.h>`.
  - Includes information that tells the compiler to convert the value of year to a temporary double representation before calling the function.

http://www.cplusplus.com/reference/cmath/

Comments on `for` loop

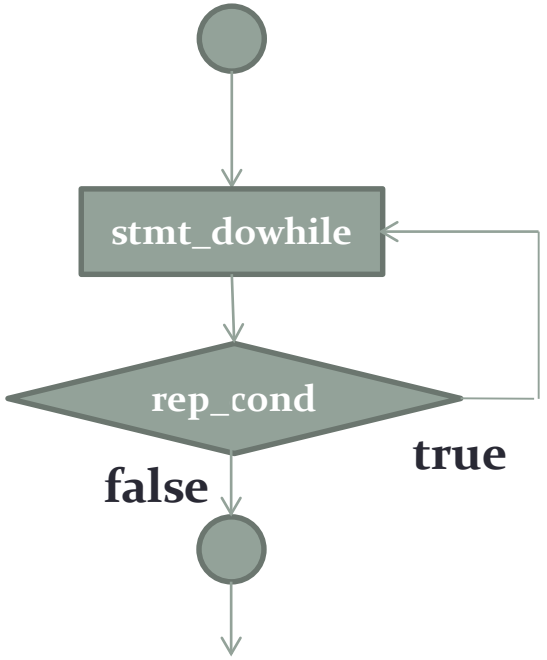
- If loop condition loop_cond is initially false =>do nothing
  - Ex: `for ( i = 0; i > 0; i++ )`
- Update statement update_stmt eventually sets the value of loop_cond to false => otherwise, this `for` loop will run forever.
  - Ex: `for ( i = 0; i < 100; i-- )`
- If ctrl. variable is `float/double`, then different computers may yield different results on ctrl variable => should avoid using such variable.
  - Ex: `for ( i = 0.1; i < 1.2; i += 0.05 )`

The do...while Loop

- Syntax of the `do ... while` statement is:

```
do
   stmt_dowhile;
while (rep_cond);
```

  - Statement `stmt_dowhile` executes first, and then `rep_cond` is evaluated
  - If `rep_cond` evaluates to true, the statement `stmt_dowhile` runs again



do...while Loop Example

- Example 1:

```
i = 1;
do
{
    printf( "%d ", i );
    i *= 2;
} while (i <= 20)
```

- screen output:

```
1 2 4 8 16
```

do…while vs. while

- Example 2a:

```
i = 15;
do {
    i %= 8;
} while ( i > 15 );
printf("%d\n", i);
```

- Example 2b:

```
i = 15;
while ( i > 15 ) {
    i %= 8;
}
printf("%d\n", i);
```

Question: What will be displayed on screen??

Flow of Control: Selection

- `if/else` statement

```
if (score >= 90) {
    grade = 'A';
} else if (score >= 80) {
    grade = 'B';
} else {
    grade = 'C';
}
```

- Ternary operator

```
(grade > 60)? printf("pass"): printf("fail");
```

Flow of Control: Selection (cont.)

- `switch` statement

```
switch (menu) {
    case item1:
        //do something
        break;
    case item2:
    case item3:
        //do something
        break;
    ...
    default:
        //do something
        break;
}
```

Selection (III): `switch`

- A new statement for controlling multiple branches
- Syntax format:

```
switch ( control_expr )
{
    case literal_1: // terminated with a colon
        statement_1;
        statement_1;
        break;
    case literal_2: // terminated with a colon
        statement_3;
        break;
    default:
        statement_n;// terminated with a colon
}
```

Selection (III): `switch` (cont.)

- Four new keywords used:
  - **`switch`**, **`case`**, **`default`**, and **`break`**
- Function:
  - `control_expr` following switch is evaluated => must compare to an literal
  - Result compared sequentially to alternative `case` values until a match is found
  - Statements following matched `case` are executed
  - When `break` reached, `switch` terminates
  - If no match found, run `default` statement block

Rewrite Month Example

- `if` Chain

```
if ( month == 1 )
    printf("Jan.\n");
else if ( month == 2 )
    printf("Feb.\n");
else if ( month == 3 )
    printf("Mar.\n");
…
else if ( month == 11 )
    printf("Nov.\n");
else
    printf("Dec.\n");
```

- `switch`

```
switch (month)
{
    case 1:
        printf("Jan.\n");
        break;
    case 2:
        printf("Feb.\n");
        break;
…
    case 11:
        printf("Nov.\n");
        break;
    default:
        printf("Dec.\n");
}
```

Common Pitfalls on `switch`

- Forgetting `break;`
  - No compiler error
  - Execution simply falls through other cases until `break;`

- Best usage: menus
  - provides clearer big-picture view
  - shows menu structure effectively
  - Each branch is one menu choice

switch Menu Example

```
switch (response)
{
    case '1':
        // Execute menu option 1
        break;
    case '2':
        // Execute menu option 2
        break;
    case '3':
        // Execute menu option 3
        break;
    default:
        printf("Please enter a valid response.");
}
```

- Good habit to enumerate all known cases and prompt by an error message if unknown case occurs

switch Example

```
// Modified from Fig. 4.7: fig04_07.c
// Counting letter grades
#include <stdio.h>

int main ( void )
{
    int grade;  // letter grade entered by user
    unsigned int aCount = 0; // number of As
    unsigned int bCount = 0; // number of Bs
    unsigned int cCount = 0; // number of Cs
    unsigned int dCount = 0; // number of Ds
    unsigned int fCount = 0; // number of Fs

    printf( "Enter the letter grade.\n");
    printf( "Enter the EOF character to end input.\n");
```

switch Example (cont.)

```
// loop until user types end-of-file key sequence
while (( grade = getchar() ) != EOF )
{
    switch ( grade )
    {
        case 'A': // grade was uppercase A
        case 'a': //  or lowercase a
            aCount++;
            break;
        case 'B': // grade was uppercase B
        case 'b': //  or lowercase b
            bCount++;
            break;
        case 'C': // grade was uppercase C
        case 'c': //  or lowercase c
            cCount++;
            break;
```

switch Example (cont.)

```
            case 'D': // grade was uppercase D
            case 'd': //  or lowercase d
                dCount++;
                break;
            case 'F': // grade was uppercase F
            case 'f': //  or lowercase f
                fCount++;
                break;
            case '\n': // ignore newlines
            case '\t': // tabs,
            case ' ':  // and spaces in input
                break;
            default:  // catch all other characters
                printf( "Incorrect letter grade enter.");
                printf( " Enter a new grade.\n");
        } // end of swicth
    } // end of while
```

`switch` Example (cont.)

```
    printf( "\nTotals for each letter grade are:\n" );
    printf( "A: %d\n", aCount); // display number of A grades
    printf( "B: %d\n", bCount); // display number of B grades
    printf( "C: %d\n", cCount); // display number of C grades
    printf( "D: %d\n", dCount); // display number of D grades
    printf( "E: %d\n", eCount); // display number of E grades

    return 0;
}
```

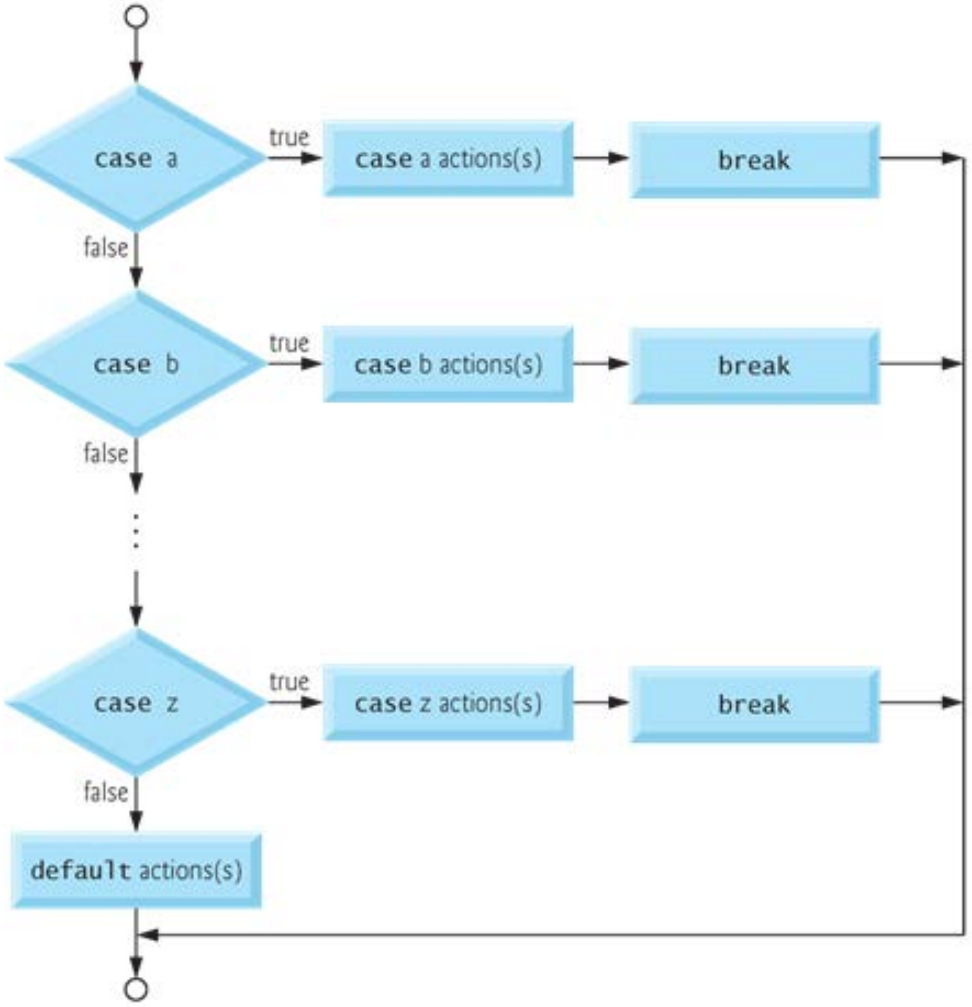
switch Example - Output

- screen output:

```
Enter the letter grade.
Enter the EOF character to end input.
a
c
A
d
E
Incorrect letter grade enter. Enter a new grade.
A
^Z

Number of students who received each letter grade:
A: 3
B: 0
C: 1
D: 1
F: 0
```

`switch` Example - Flowchart



getchar() Function

- The `getchar()` function reads one character from the keyboard.
- Normally, characters are stored in variables of type `char`; however, characters can be stored in any integer data type, because types `short`, `int` and `long` are guaranteed to be at least as big as type `char`.
- We can treat a character either as an integer or as a character, depending on its use.

```
printf( "The character (%c) has the value
%d\n", 'a', 'a');
```

prints the character a and its integer value as follows:

- The character (a) has the value 97

EOF

- EOF stands for "end-of-file".
  - Commonly used as a sentinel value.
  - However, you do not type the value –1, nor do you type the letters EOF as the sentinel value.
- EOF is a symbolic integer constant defined in the `<stdio.h>` header file.
- You type a system-dependent keystroke combination that means "end-of-file" to indicate that you have no more data to enter.
  - Windows: ctrl+z
  - Linux: ctrl+d.

`break` & `continue` in Repetition

- Flow of Control
  - Recall how loops provide "graceful" and clear flow of control in and out
  - In RARE instances, can alter natural flow
- `break`
  - force the loop to exit immediately.
- `continue`
  - skip the rest of loop body
- These statements violate natural flow
  - only used when absolutely necessary!

`break` in Loop

- `break`: forces immediate exits from structures:
  - in `switch` statements:
    - the desired case is detected/processed
  - in `while`, `for` and `do...while` statements:
    - an unusual condition is detected
- Example:

```
for ( i = 10; i <= 50; i += 2 ) {
    if ( i%9 == 0)
        break;
    printf( "%d ", i );
}
```

`continue` in Loop

- `continue`: cause the next iteration of the loop to begin immediately
  - execution transferred to the top of the loop
  - apply only to `while`, `for` and `do…while` statements
- Example:

```
i = 0;
while ( i < 100 ) {
    i++;
    if ( i == 50)
        continue;
    printf( "%d", i );
}
```

Nested Loop

- A loop contained within another loop
- Example: Print 9x9 multiples

```
for ( i = 1; i <= 9; i++ ) { //outer loop
    printf( "%d-multiples: \n", i );
    for ( j = 1; j <= 9; j++ ) {//inner loop
        printf( "%d ", i*j );
    } // end of inner loop
    printf( "\n");
} // end of outer loop
```

Nested Loop (cont.)

- Outer (first) Loop:
  - controlled by value of i
- Inner (second) Loop:
  - controlled by value of j
- Rules:
  - For each single trip through outer loop, inner loop runs through its entire sequence
  - Different variables to control each loop
  - Inner loop statements contained within outer loop

Logical Operators

- C provides logical operators that are used to form more complex conditions by combining simple conditions.
- The logical operators are
  - && (logical AND)
  - || (logical OR)
  - ! (logical NOT, also called logical negation).

Logical Operators (cont.)

- AND operator, &&:
  - Used with 2 simple expressions
- Example: `(age > 40) && (term < 10)`
  - Compound condition is true (has value of 1) only if age > 40 and term < 10
- OR operator, ||:
  - Used with two simple expressions
- Example: `(age > 40) || (term < 10)`
  - Compound condition is true if age > 40 or if term < 10 or if both conditions are true

Logical Operators (cont.)

- NOT operator, ! :
  - Changes an expression to its opposite state
  - If expr_A is true, then !expr_A is false

Operator Precedence and Associativity

Operators	Associativity	Type
++ *(postfix)* -- *(postfix)*	right to left	postfix
+ - ! ++ *(prefix)* -- *(prefix)* (*type*)	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

- `x + 1 > 2 || x + 1 < -3` means

  `(x + 1) > 2 || (x + 1) < -3`

Equality (==) vs. Assignment (=)

- Any expression in C that produces a value can be used in the decision portion of control statement.
- Integers as Boolean values
  - All non-zero values => true
  - Zero value => false

```
if ( i == -1 )
    printf( " Nagative!" );
```

```
if ( i = -1 )
    printf( " Nagative!" );
```

Summary

- C has three repetition structures: `while`, `for`, and `do…while`
  - `while` and `for` loops are called pre-test loops
  - `do…while` is called a post-test loop
    - `do...while` always executes at least once
- `for` simplifies the writing of a counter-controlled `while` loop
- `switch` statement: multi-way selection
  - the value of an integer expression is compared to a sequence of integer or character constants or constant expressions
  - program execution transferred to first matching case
  - Execution continues until optional `break` statement is encountered

Summary (cont.)

- Executing a `break` statement in the body of a loop immediately terminates the loop
- Executing a `continue` statement in the body of a `loop` skips to the next iteration
- After a `continue` statement executes in a `for` loop, the update statement is the next statement executed
- More complex conditions can be constructed from relational expressions using logical operators, `&&` (AND), `||` (OR), and `!` (NOT)

Exercise (1)

- Find the error in each of following code segments and explain how to correct it.

a)
```
x = 1;
while ( x <= 10 ) ;
    ++x;
}
```

b)
```
for ( y = .1; y !=1.0; y+= .1 )
    printf( "%f\n", y );
```

c) The following code should print the values 1 to 10.

```
n = 1;
while ( n < 10 )
    printf( "%d ", n++);
```

Exercise (1) (cont.)

- Find the error in each of following code segments and explain how to correct it.

d)
```
switch ( n ) {
    case 1:
        puts ( " The number is 1" );
    case 2:
        puts ( " The number is 2" );
        break;
    default:
        puts ( " The number is not 1 or
2" );
        break;
}
```

Exercise (2)

- Print the multiple of 3 between 1 to 45 using a `while` loop and the value `p`. Assume that the variable `p` has been defined, but no initialized. Print only three integers perline.

Exercise (3)

- Repeat Exercise (2) using a `for` statement.