Lecture 9.2

Topics

1. **`switch`** Statement – Revisited
2. Introduction to Repetition/Loop Programming Structures – **`while`** Statement

---

1. `switch` Statement – Revisited

Recall that the C **`switch`** statement can handle multiple options beside the extended `if-else if-else` statements.

1.1 `switch` Syntax and Flowchart

Its syntax is given as follows,

```
switch ( testExpression ) {
  case constantValue1 : statement1
                        break;
  case constantValue2 : statement2
                        break;
  …
  case constantValueN : statementN
                        break;
  default :             statementDefault
}
```

where

(1) `testExpression` must produce an integral value. It is commonly given as a unary expression in the form of an identifier.

(2) `constantValue1, constantValue2, …, constantValueN` represent all possible values matching with the above integral value (i.e., testExpression or its result).

The **`switch`** statement will have the following characteristics.

(a) The test expression after the **`switch`** keyword must be an integral type.

(b) The expression after the **`case`** keyword must be a constant expression. The expression together with the **`case`** keyword is called a **`case-label`** statement. Note that a constant expression is an expression that is evaluated at compiled time, not run time.

(c) No two **`case`** labels can have the same value.

(d) Two **`case`** labels can be associated with the same statements.

(e) The **`default`** label is not required. If there is no match, then the control jumps outside of the **`switch`** statement.

(f) There can be at most one **`default`** label. It can be placed anywhere, but it is mostly placed last in the structure.

A general flowchart is given in **Figures 1 & 2** as follows,

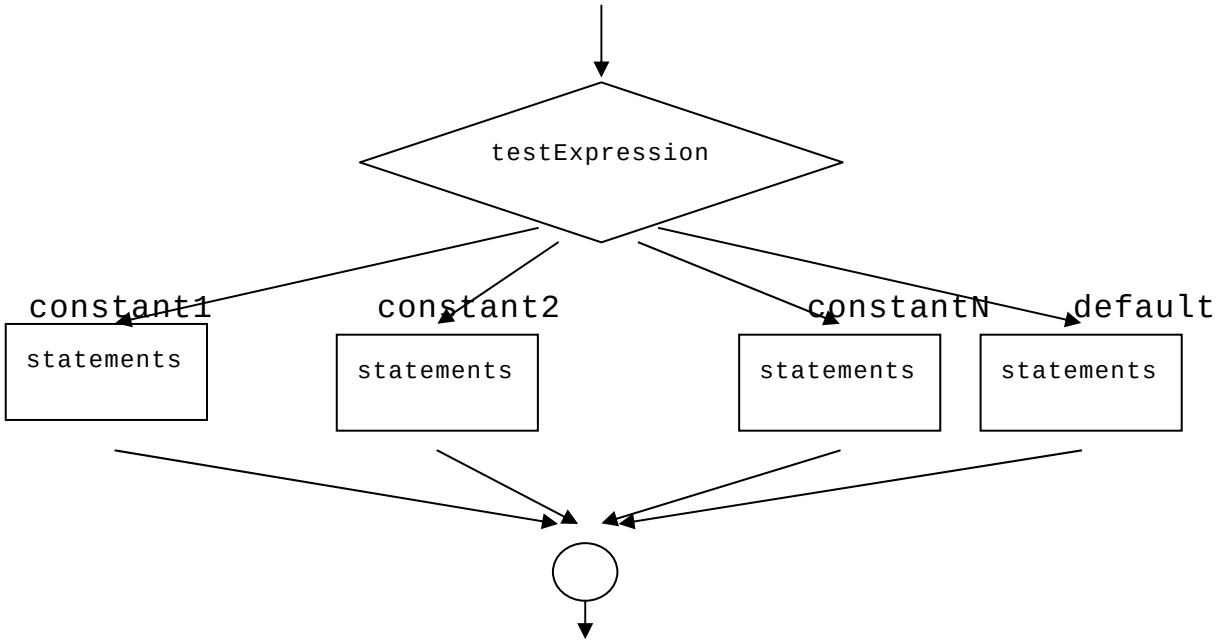


Figure 1 A general `switch` structure

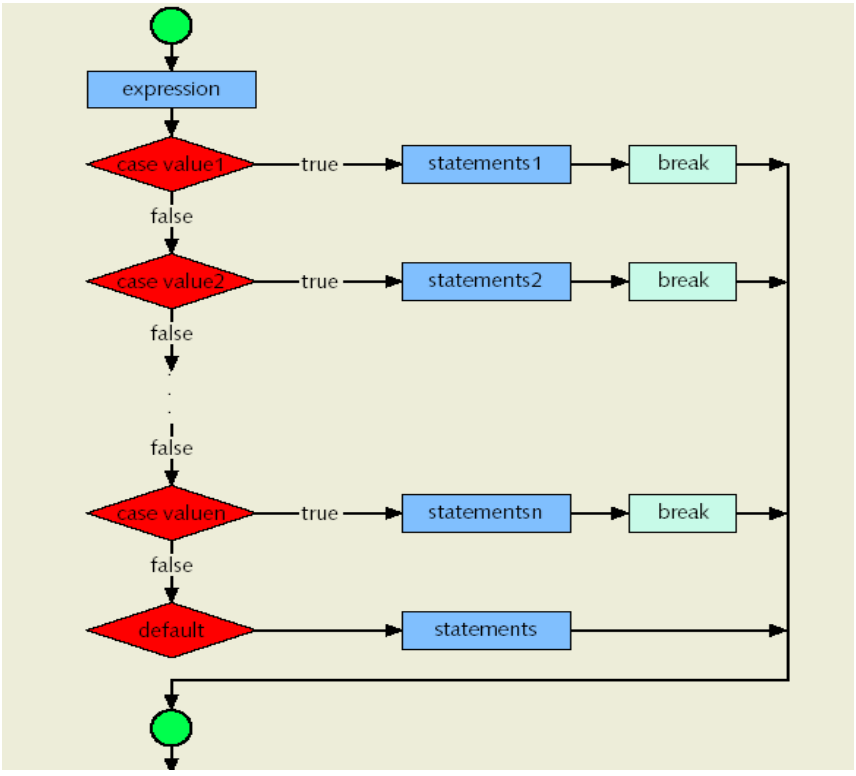


Figure 2 A `switch` structure

1.2 Example – Menu setup

Recall that a menu program will provide the user with options and selections. The execution will continue after an option is selected and entered to the program.

Let's consider a menu of four basic arithmetic operations:

(1) Add

(2), Subtract

(3) Multiply

(4) Divide

As the application (i.e., program) is run, the monitor will show the following output:

- Menu with options will be displayed, and
- The user must select and enter the (appropriate) option, and
- The program, based on the selection, will perform the desired operation.

Example 1

```
/**
 *Program Name:  cis26L0921.c
 *Discussion:    Arithmetic menu with switch-structure
 */
#include <stdio.h>

/*Function prototypes*/

void displayMenu( void );

double add( double, double );

double subtract( double, double );

double multiply( double, double );

double divide( double, double );

int main() {
  int iOption;
  double dNum1;
  double dNum2;
  double dResult;

  displayMenu();

  printf( "\n\nSelect and enter an integer for option + ENTER: " );
  scanf( "%d", &iOption );

  printf( "\nEnter first operand: " );
  scanf( "%lf", &dNum1 );

  printf( "\nEnter second operand: " );
  scanf( "%lf", &dNum2 );
```

```
  switch( iOption ) {
    case 1: dResult = add( dNum1, dNum2 );
            printf( "\n%f + %f --> %f", dNum1, dNum2, dResult );
            break;
    case 2: dResult = subtract( dNum1, dNum2 );
            printf( "\n%f - %f --> %f", dNum1, dNum2, dResult );
            break;
    case 3: dResult = multiply( dNum1, dNum2 );
            printf( "\n%f * %f --> %f", dNum1, dNum2, dResult );
            break;
    case 4: dResult = divide( dNum1, dNum2 );
            printf( "\n%f / %f --> %f", dNum1, dNum2, dResult );
            break;
    default: printf( "\nInvalid Option!" );
  }

  printf( "\n" );

  return 0;
}

/**
 *Function Name: displayMenu()
 *Description  : Displaying operation menu
 *Pre          : None
 *Post         : None
 */
void displayMenu() {
  printf( "\n\t(1) Add\n\t(2) Subtract\n\t(3) Multiply"
          "\n\t(4) Divide" );
  return;
}

/**
 *Function Name: add()
 *Description  : Adding two numbers
 *Pre          : Two numbers
 *Post         : Sum of two numbers
 */
double add( double dOld1, double dOld2 ) {
  return ( dOld1 + dOld2 );
}

/**
 *Function Name: subtract()
 *Description  : Subtracting two numbers
 *Pre          : Two numbers
 *Post         : Difference of two numbers
 */
double subtract( double dOld1, double dOld2 ) {
  return ( dOld1 - dOld2 );
}

/**
 *Function Name: multiply()
 *Description  : Multiplying two numbers
 *Pre          : Two numbers
 *Post         : Product of two numbers
```

```
 */
double multiply( double dOld1, double dOld2 ) {
  return ( dOld1 * dOld2 );
}

/**
 *Function Name: divide()
 *Description  : Dividing two numbers
 *Pre          : Two numbers
 *Post         : Result of the division of two numbers
 */
double divide( double dOld1, double dOld2 ) {
  return ( dOld1 / dOld2 );
}
```

OUTPUT – Sample Run #1

```
        (1) Add
        (2) Subtract
        (3) Multiply
        (4) Divide

Select and enter an integer for option + ENTER: 1

Enter first operand: 4

Enter second operand: 5

4.000000 + 5.000000 --> 9.000000
```

OUTPUT – Sample Run #2

```
        (1) Add
        (2) Subtract
        (3) Multiply
        (4) Divide

Select and enter an integer for option + ENTER: 2

Enter first operand: 4

Enter second operand: 5

4.000000 - 5.000000 --> -1.000000
```

OUTPUT – Sample Run #3

```
        (1) Add
        (2) Subtract
        (3) Multiply
        (4) Divide

Select and enter an integer for option + ENTER: 3

Enter first operand: 4
```

```
Enter second operand: 5

4.000000 * 5.000000 --> 20.000000
```

2. Introduction to Repetition/Loop Programming Structures – `while` Statement

The above example shows that the program can be run many times as needed; and each time, there is only one of the four (4) operations being selected and performed, and the program ends.

What if we want to have a program that will allow us to select and run the options until we decide to stop? A sample output may look as follows,

```
        MENU --

        (1) Add
        (2) Subtract
        (3) Multiply
        (4) Divide
        (5) Quit

Select and enter an integer for option + ENTER: 1

Enter first operand: 2

Enter second operand: 3

2.000000 + 3.000000 --> 5.000000

        MENU --

        (1) Add
        (2) Subtract
        (3) Multiply
        (4) Divide
        (5) Quit

Select and enter an integer for option + ENTER: 9

Invalid Option!

        MENU --

        (1) Add
        (2) Subtract
        (3) Multiply
        (4) Divide
        (5) Quit

Select and enter an integer for option + ENTER: 3

Enter first operand: 5

Enter second operand: 6

5.000000 * 6.000000 --> 30.000000
```

```
        MENU --

        (1) Add
        (2) Subtract
        (3) Multiply
        (4) Divide
        (5) Quit

Select and enter an integer for option + ENTER: 5

It is fun! Bye ...
```

A closer look would point to a repeated call to display the menu and then continue to the selection of option. This process eliminates the same group of statements to be rewritten over and over in the program.

More than that, one may not know how many times the same statements to be repeated. Thus, new structure and syntax must be used to resolve these issues.

Let's begin with the introduction of repetition or loop structures in C before revisiting the above exercise.

2.1 Programming Structures

Recall that there are 3 programming structures as illustrated in the figure below.

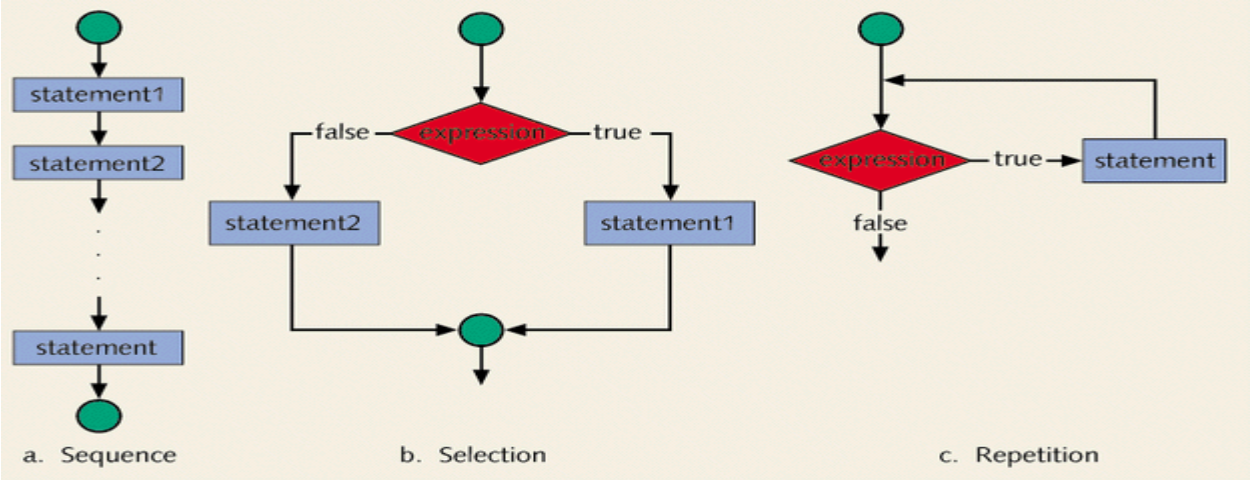


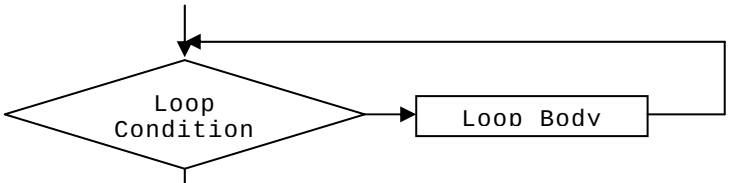
Figure 3 Three basic programming structures

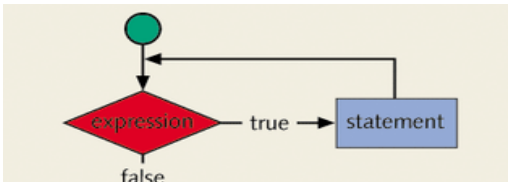
The sequential and conditional structures have been introduced. The loop structure will be introduced next.

2.2 Loop Structures

General looping structures are depicted in **Figures 4&5** below.

Looking at the flowchart for a loop, there is at least one conditional block and other processing blocks. A loop must end or exit through the result of this conditional block. There are several different structures of loop.

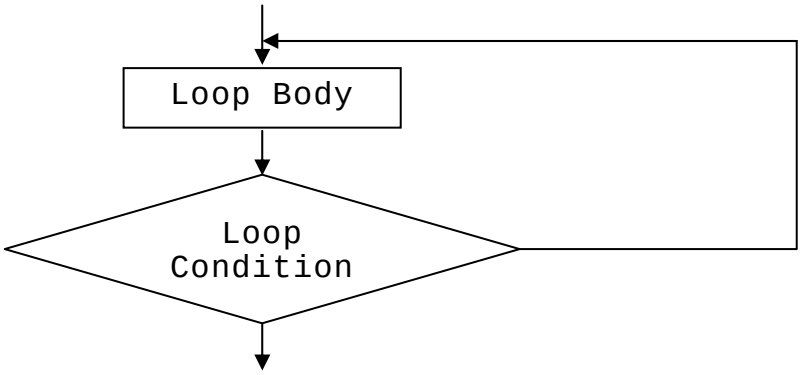




Yes

No

Figure 4 A general flowchart depicts a loop



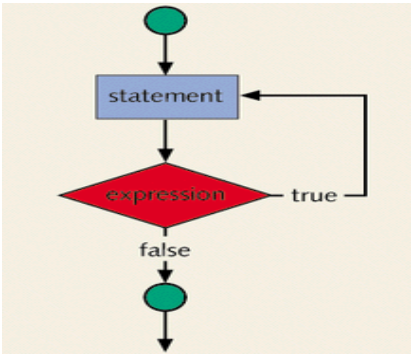


Figure 5 Loop with body executed **at least once**

Let's look at these structures next.

2.3 Loop Structures

There are two basic looping requirements for the execution of a loop:

(A) Loop body must be executed at least once, and

(B) Loop body may not be executed at all.

Figure 4 depicts a loop with the loop body may not be executed at all. **Figure 5** depicts a loop that must be executed at least once

In the current C, C++, C#, Java and many other computer languages, there are several loop statements/structures that one can use.

Each loop can be thought of a process where three main steps are preformed after initializing (one may also want to include an **initialization step** as part of a loop structure):

(1) Testing/Conditioning,
(2) Loop Body, and
(3) Update.

In general, they are described as below.

*A. Loops that may **not** have body executed at all:*

Loop #1

```
/*Initial Condition*/
while ( loopCondition ) {
  /*Loop Body*/

  /*Update*/
}
```

Loop #2

```
/*Initial Condition*/
while ( loopCondition ) {
  /*Update*/
```

```
        /*Loop Body*/
      }
Loop #3
      for ( initialization; testExpression; update ) {
        /*Loop Body*/
      }
```

B. Loops that must be executed ***at least once*** **`do-while`** ***loop****:*

```
Loop #1
      /*Initial Condition*/
      do {
        /*Loop Body*/

        /*Update*/
      } while ( loopCondition );
Loop #2
      /*Initial Condition*/
      do {
        /*Update*/

        /*Loop Body*/
      } while ( loopCondition );
```

2.4 Example – `while` loop

Example

```
/**
 *Program Name:  cis26L0922.c
 *Discussion:    while-Loop
 */
#include <stdio.h>

/*Function prototype*/
void printDaySwitch( int );

int main() {
  int iSelection;

  printf( "\nEnter an integer from 0 through 6 "
          "or any other to quit: " );
  scanf( "%d", &iSelection);

  while ( iSelection >= 0 && iSelection <= 6 ) {
    printDaySwitch( iSelection );

    printf( "\nEnter an integer from 0 through 6 "
            "or any other to quit: " );
    scanf( "%d", &iSelection);
  }

  printf( "\n" );
  return 0;
}

/**
 *Function Name: printDaySwitch()
 *Description  : Displaying day of the week
```

```
 *Pre           : Selected day
 *Post          : None
 */
void printDaySwitch( int iDay ) {
  switch( iDay ) {
    case 0: printf( "\nIt is Sunday!" );
            break;
    case 1: printf( "\nIt is Monday!" );
            break;
    case 2: printf( "\nIt is Tuesday!" );
            break;
    case 3: printf( "\nIt is Wednesday!" );
            break;
    case 4: printf( "\nIt is Thursday!" );
            break;
    case 5: printf( "\nIt is Friday!" );
            break;
    case 6: printf( "\nIt is Saturday!" );
            break;
    default: printf( "\nIt is an INVALID selection!" );
  }

  return;
}
```

OUTPUT

```
Enter an integer from 0 through 6 or any other to quit: 1

It is Monday!
Enter an integer from 0 through 6 or any other to quit: 2

It is Tuesday!
Enter an integer from 0 through 6 or any other to quit: 0

It is Sunday!
Enter an integer from 0 through 6 or any other to quit: 3

It is Wednesday!
Enter an integer from 0 through 6 or any other to quit: 4

It is Thursday!
Enter an integer from 0 through 6 or any other to quit: 6

It is Saturday!
Enter an integer from 0 through 6 or any other to quit: 5

It is Friday!
Enter an integer from 0 through 6 or any other to quit: 7
```