Lecture 10.1

Topics

1. Introduction to Repetition/Loop Programming Structures – `for` Loop

---

1. Introduction to Repetition/Loop Programming Structures – `for` Loop

At times, one would like to select and run (or test) a functionality with different sets of data. For example, a program logic can be run; 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 desired 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 for execution. The user will control how many times the required statements to be selected and executed. The process will stop only if the user would choose the option to quit!

To perform the above logic, a combination of conditional structures and loop will be needed. We will look into this combination in the next few discussions.

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

1.1 Programming Structures

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

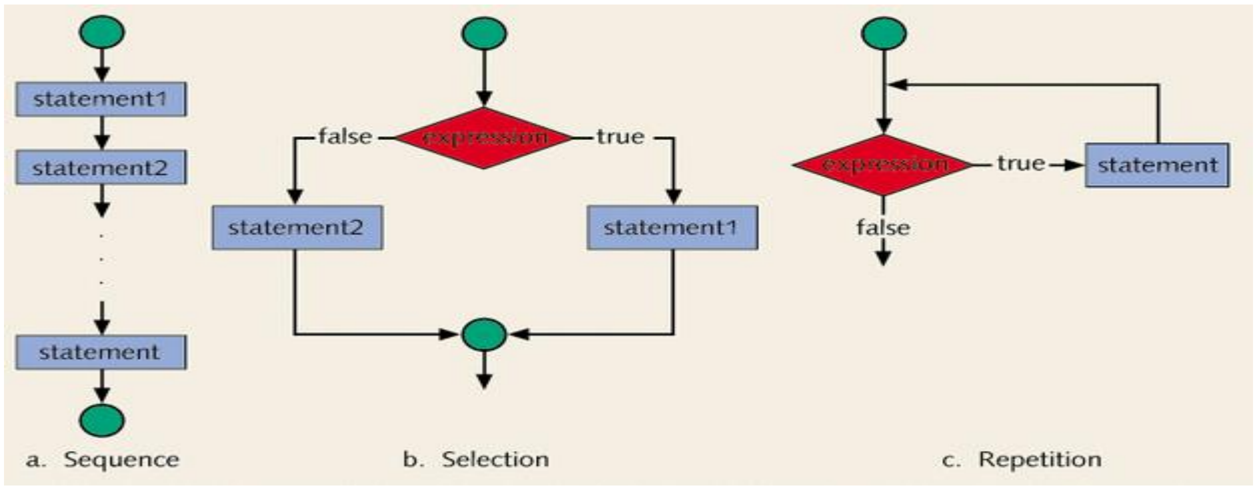


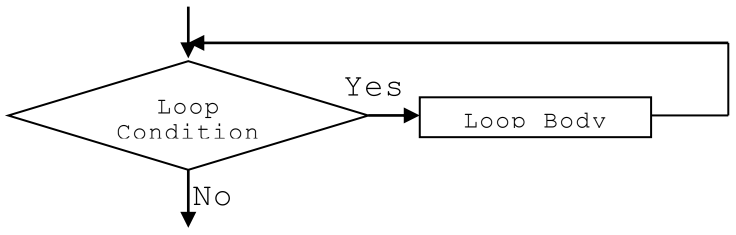
Figure 1 Three basic programming structures

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

1.2 Loop Structures

General looping structures are depicted in **Figures 2&3** 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.



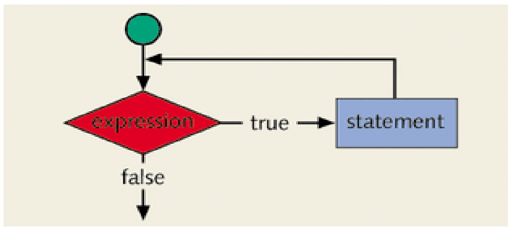
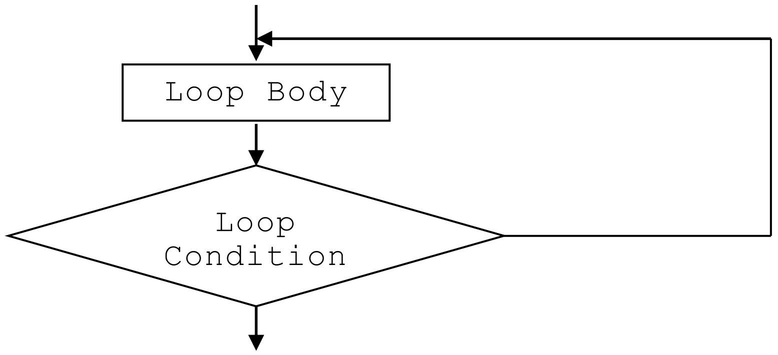


Figure 2 A general flowchart depicts a loop — Pre-Test Loop



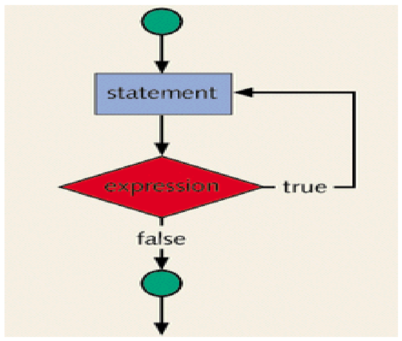


Figure 3 Loop with body executed **at least once** — Post-Test Loop

Let's look at these structures next.

1.3 Loop Structures

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

(A) Loop body must be executed at least once — Post-Test Loop, and

(B) Loop body may not be executed at all — Pre-Test Loop.

Figure 2 depicts a loop with the loop body may not be executed at all. **Figure 3** depicts a loop that must be executed at least once

In programming languages such as C, C++, Java, JavaScript, C#, Go (and many other computer languages with some variations), 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);
```

In particular, the **`for`** -loop has a very clear and unique syntax to indicate the above steps or components.

1.1 `for` Loop – Syntax

The **`for`**-loop is also called the **counting loop** because the loop has a very clear starting and ending steps that would imply some number of times the loop must go through.

The syntax of a **`for`**-loop is given as follows,

```
for (initialization; conditioning; update) {

  // Loop Body/Processing

}
```

It would be best to explain the above through examples.

1.2 Examples

Example 1 – Finding the total of all integers within some fixed interval (e.g., `[1, 100]`)

```
/**
 * Program Name: cis6L1011.c
 * Disscussion:  for-loop in summing integers
 * Written By:
 * Date:
 */
#include <stdio.h>

int main() {
  int total;
  int i;

  for (total = 0, i = 1; i <= 100; i++) {
    total += i;
  }

  printf("\nSum of all integers from 1 through 100 (inclusive): %d\n",
    total);

  return 0;
}

/** PROGRAM OUTPUT

Sum of all integers from 1 through 100 (inclusive): 5050
*/
```

Example 2 – Finding the total of 5 integers

```
/**
 * Program Name: cis6L1012.c
 * Disscussion:  for-loop in summing
 * Written By:
 * Date:
 */
#include <stdio.h>

int sum5Int(void);

int main() {
  int sum;

  sum = sum5Int();

  printf("\nThe sum is %d\n", sum);

  return 0;
}

// Function Definitions

int sum5Int() {
  int value;
  int total = 0;
  int i;

  for (i = 0; i < 5; i++) {
    printf("\nEnter an int: ");
    scanf("%d", &value);

    total += value;
  }

  return total;
}

/** PROGRAM OUTPUT
```

```
Enter an int: 2

Enter an int: 45

Enter an int: 11

Enter an int: 36

Enter an int: 9

The sum is 103
*/
```

Example 3 – Finding the sum of all integers in some given interval.

```
/**
 * Program Name: cis6L1013.c
 * Disscussion:  for-loop in summing
 * Written By:
 * Date:
 */
#include <stdio.h>

int sum5Int(void);
int sumAllInt(void);

int main() {
  int sum;

  sum = sumAllInt();

  printf("\nThe sum is %d\n", sum);

  return 0;
}

// Function Definitions

int sumAllInt() {
  int total = 0;
  int starting;
  int ending;
  int i;

  printf("\nEnter the starting value of the interval: ");
  scanf("%d", &starting);

  printf("\nEnter the ending value of the interval: ");
  scanf("%d", &ending);

  for (i = starting; i <= ending; i++) {
    total += i;
  }

  return total;
```

```c
}

int sum5Int() {
  int value;
  int total = 0;
  int i;

  for (i = 0; i < 5; i++) {
    printf("\nEnter an int: ");
    scanf("%d", &value);

    total += value;
  }

  return total;
}

/** PROGRAM OUTPUT

Enter the starting value of the interval: 1

Enter the ending value of the interval: 100

The sum is 5050
*/
```