LECTURE 06

Repetition

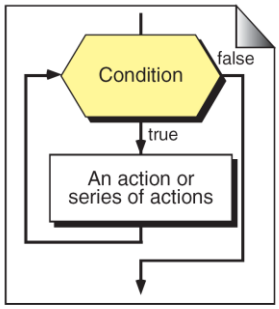
Outcomes

1. To understand basic loop concepts.
2. To create simple C program that use `for`, `while`, `do-while` statements.

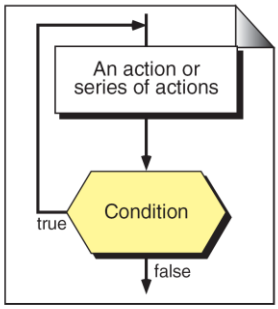
Contents

1. Introduction

C has three loop statements which are the `for`, the `while`, and the `do-while`. The first two are pre-test loops and the last one is a post-test loop. So what is pre-test and post-test loops? In a pre-test loop, in each of iteration, the condition is tested first. If it is true, the loop continues, otherwise the loop is terminated. It is vice versa in post-test loop whereby the loop continues and the condition is tested later. If condition is true, a new iteration is started, otherwise the loop is terminated. The below figures illustrate these two loop types.



(a) Pretest Loop



(b) Post-test Loop

The `while` and `do-while` are most commonly used for **event-controlled loops**, and the `for` is usually used for **counter-controlled loops**. In event-controlled loop, an event changes condition from true to false. For example, `marks` is the condition of the `while` loop, if `marks` is not between 0 to 100, this condition is true and the loop keeps running to ask user to enter user's mark. The event-controlled loop is shown in the below examples.

```
/*
C Program to demonstrate event-controlled loops (pre-test)
Written by: AFAN, FKP, UMP    Date: October 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int marks;
      scanf("%d",&marks);
      while ((marks<0) || (marks>100))
      {
            printf("\n");
            scanf("%d",&marks);
      }
      return 0;
```

```
}
/*
C Program to demonstrate event-controlled loops (post-test)
Written by: AFAN, FKP, UMP    Date: October 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int marks;
      do
      {
            scanf("%d",&marks);
            printf("\n");
      }while ((marks<0) || (marks>100));
      return 0;
}
```

Meanwhile in counter-controlled loops, number of times an action is to be repeated is known. We must initialize, update, and test the counter. The update can be increment (we are counting up) or a decrement (we are counting down). For example, we want to display increment numbers from 0 – 4 (increment mode `i++`) or we want to display decrement numbers from 4 – 0 (decrement mode `i--`). The sample of codes for counter-controlled loops is illustrated in the next paragraph.

```
/*
C Program to demonstrate counter-controlled loops
Written by: AFAN, FKP, UMP    Date: October 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{     int i;
      printf("Increment Numbers\n");
      for (i=0; i<5; i++)
            printf("%d\t", i);
      printf("Decrement Numbers\n");
      for (i=4; i>0; i--)
            printf("%d\t", i);
      return 0;
}
```

```
Increment Numbers
2          3          4
Decrement Numbers
1
```

2. The `for` Loop

The basic syntax for `for` loop is from the initial value of counter, the counter is checked whether it is meet some condition. If the answer of a condition is true, one or more action statements are executed. Then, at the end of loop, counter will be updated. The loop will be repeated until the condition is false. When it is false, it will exit the `for` loop and go to the next line of your program. This syntax is simplified in the right fugure. To further understand the `for` loop, lets use the below example.

for loop syntaxes:

```
for (initial value; condition; update counter)
        statement;
OR
for (initial value; condition; update counter)
{
        statement;
        statement;
}
```

```
/*
C Program to demonstrate for loop
Written by: AFAN, FKP, UMP     Date: October 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int i;
      for (i=0; i<3; i++)
      {
            printf("Computer Programming\n");
      }
      printf("End\n");
      return 0;
}
```

```
Computer Programming
Computer Programming
Computer Programming
End
```

The output of the above program is three times of Computer Programming word and one time End word. Below is the explanations of this program.

At first iteration:

`i=0` is true for condition `i<3`.

Then `printf` Computer Programming.

At the end of first iteration `i` become `1` because `i++` means `i+1 = 0+1`.

At second iteration:

`i=1` is true for condition `i<3`.

Then `printf` Computer Programming.

At the end of second iteration `i` become `2` because `i++` means `i+1 = 1+1`.

At third iteration:

`i=2` is true for condition `i<3`.

Then `printf` Computer Programming.

At the end of third iteration `i` become `3` because `i++` means `i+1 = 2+1`.

At fourth iteration:

`i=3` is not true for condition `i<3`.

Then the `for` loop is terminated (quit).

When quit the `for` loop:

Then `printf` End.

A `for` loop is used when a loop is to be executed a known number of times. We can do the same thing with a `while` loop, but the `for` loop is easier to read and more nature for counting loops. Other example of using `for` loop is as below.

Question:

Develop a program for the following problem.

Read 5 integers and display the value of their summation.

Input: 5 integers (n1, n2, n3, n4, n5)

Process: sum = n1+n2+n3+n4+n5

Output: the summation of n1, n2, n3, n4, n5 (sum)

Following is the example of solution for this question.

```
/*
C Program to demonstrate for loop
Written by: AFAN, FKP, UMP    Date: October 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int i,n;
      int sum=0;
      for (i=0; i<5; i++)
      {
```

```
            scanf("%d", &n);
            sum=sum+n;
        }
        printf("\nSum of 5 integers are: %d: ", sum);
        return 0;
}
```

```
1 2 3 4 5
Sum of 5 integers are: 15
```

3. The `while` Loop

The basic syntax for `while` loop is if the answer of condition is true, one or more action statements are executed in the body of `while`. When the condition is false, it will exit the `while` loop and go to the next line of your program. The condition will be tested before every iteration since `while` loop is a pre-test loop. This terminology is illustrated in the right fugure. To further understand the `while` loop, lets use the below example.

while loop syntaxes:

```
while (condition)
        statement;
OR
while (condition)
{
        statement;
        statement;
}
```

Question:
Receive a number of positive integers and display the summation of these integers.

Below is the example of solution for this question.

```
/*
C Program to demonstrate while loop
Written by: AFAN, FKP, UMP     Date: October 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
        int x,sum;
        sum=0;
        scanf("%d",&x);
        while (x>0)
        {
                sum=sum+x;
                scanf ("%d",&x);
        }
        printf("Sum = %d",sum);
        return 0;
}
```

```
Example of input: 30 16 42 -9
Sum = 88
```

Below is the explanations of the above program:

Step1: Lets say user put `30` for first input. Now `x` become `30`.

Step2: Then, enter `while` loop. `while (x>0)` is true [`(30>0)`], `sum = sum + x = 0` (intial sum) `+ 30 = 30`. Now `sum` become `30`. Insert a new input. Lets say user insert `16`. Now `x` become `16`.

Step3: End of `while` loop

Step4: Loop back to `while` loop

Step5: Then, enter `while` loop. `while (x>0)` is true [`(46>0)`], `sum = sum + x = 30 + 16 = 46`. Now `sum` become `46`. Insert a new input. Lets say user insert `42`. Now `x` become `42`.

Step6: End of `while` loop

Step7: Loop back to `while` loop

Step8: Then, enter `while` loop. `while (x>0)` is true [`(42>0)`], `sum = sum + x = 46 + 42 = 88`. Now `sum` become `88`. Insert a new input. Lets say user insert `-9`. Now `x` become `-9`.

Step9: End of `while` loop

Step10: Loop back to `while` loop

Step11: Then, enter `while` loop. `while (x>0)` is false [`(-9>0)`], then the `while` loop is terminated (quit).

Step12: End of `while` loop

Step13: The statement after `while` loop is `printf sum`. Then `sum = 88` will displayed.

Other example of using `while` loop is as below.

Question:

Given an exam mark as input, display the appropriate message based on the rules below:

If the mark is greater than 49, display “PASS”, otherwise display “FAIL”

However, for input outside the 0-100 range, display “WRONG INPUT” and prompt the user to input again until a valid input is entered

Following is the example of solution for this question.

```
/*
C Program to demonstrate while loop
Written by: AFAN, FKP, UMP    Date: October 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int marks;
      scanf("%d",&marks);
      while ((marks<0) || (marks>100))
```

```
	{
		printf("WRONG INPUT");
		scanf("%d",&marks);
	}
	if (marks>49)
		printf("PASS");
	else
		printf("FAIL");
	return 0;
}
```

```
101
WRONG INPUT
-10
WRONG INPUT
55
PASS
```

4. The `do-while` Loop

The `do-while` loop is almost the same with `while` loop. However, it is differ in several things. 1) It is post-test loop, therefore, it is started with `do` statement meaning that do some statements, and ended with condition checking, and 2) `while (condition)` statement is ended with semicolon (`;`). This syntax is simplified in the right fugure. To further understand the `do-while` loop, lets use the below example.

do-while loop syntaxes:

```
do
{
      statement;
} while (condition);
OR
do
{
      statement;
      statement;
} while (condition);
```

Question:

Write a coding that ask user to insert start and end input in integer. Display all of numbers in increasing manner (`+1`) from start to end number.

Following is the example of solution for this question.

```
/*
C Program to demonstrate do-while loop
Written by: AFAN, FKP, UMP    Date: October 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int start, end;
      printf("Input start and end value: ");
      scanf("%d %d", &start, &end);
      do
      {
            printf("%d\n", start);
            start++;
      } while (start <= end);
      return 0;
}
```

```
Input start and end value: 1 11
1    2    3    4    5    6    7    8    9    10    11
```

Below is the explainations of the above program:

Step1: Lets say user put `1` and `11` for input. Now `start` become `1` and `end` become `11`.

Step2: Then, enter `do-while` loop. `printf start` which is `1`. `start++ = start + 1 = 1 + 1 = 2`. Now `start` become `2`. Check `while (start <= end)` is true [`(2 <= 11)`].

Step3: End of `do-while` loop

Step4: Loop back to `do-while` loop because `while (start <= end)` is true

Step5: Then, enter `do-while` loop. `printf start` which is `2. start++ = start + 1 = 2 + 1 = 3.` Now `start` become `3`. Check `while (start <= end)` is true [`(3 <= 11)`].

Step6: End of `do-while` loop

Step7: Loop back to `do-while` loop because `while (start <= end)` is true

Step8: Then, enter `do-while` loop. `printf start` which is `3. start++ = start + 1 = 3 + 1 = 4.` Now `start` become `4`. Check `while (start <= end)` is true [`(4 <= 11)`].

Step9: End of `do-while` loop

Step10: Loop back to `do-while` loop because `while (start <= end)` is true

Step11: Then, enter `do-while` loop. `printf start` which is `4. start++ = start + 1 = 4 + 1 = 5.` Now `start` become `5`. Check `while (start <= end)` is true [`(5 <= 11)`].

Step12: End of `do-while` loop

Step13: Loop back to `do-while` loop because `while (start <= end)` is true

Step14: Then, enter `do-while` loop. `printf start` which is `5. start++ = start + 1 = 5 + 1 = 6.` Now `start` become `6`. Check `while (start <= end)` is true [`(6 <= 11)`].

Step15: End of `do-while` loop

Step16: Loop back to `do-while` loop because `while (start <= end)` is true

Step17: Then, enter `do-while` loop. `printf start` which is `6. start++ = start + 1 = 6 + 1 = 7.` Now `start` become `7`. Check `while (start <= end)` is true [`(7 <= 11)`].

Step18: End of `do-while` loop

Step19: Loop back to `do-while` loop because `while (start <= end)` is true

Step20: Then, enter `do-while` loop. `printf start` which is `7. start++ = start + 1 = 7 + 1 = 8.` Now `start` become 8. Check `while (start <= end)` is true [`(8 <= 11)`].

Step21: End of `do-while` loop

Step22: Loop back to `do-while` loop because `while (start <= end)` is true

Step23: Then, enter `do-while` loop. `printf start` which is `8. start++ = start + 1 = 8 + 1 = 9.` Now `start` become `9`. Check `while (start <= end)` is true [`(9 <= 11)`].

Step24: End of `do-while` loop

Step25: Loop back to `do-while` loop because `while (start <= end)` is true

Step26: Then, enter `do-while` loop. `printf start` which is `9. start++ = start + 1 = 9 + 1 = 10.` Now `start` become `3`. Check `while (start <= end)` is true [`(10 <= 11)`].

Step27: End of `do-while` loop

Step28: Loop back to `do-while` loop because `while (start <= end)` is true

Step29: Then, enter `do-while` loop. `printf start` which is `10. start++ = start + 1 = 10 + 1 = 11.` Now `start` become `3`. Check `while (start <= end)` is true [`(11 <= 11)`].

Step30: End of `do-while` loop

Step31: Loop back to `do-while` loop because `while (start <= end)` is true

Step32: Then, enter `do-while` loop. `printf start` which is `11`. `start++ = start + 1 = 11 + 1 = 12`. Now `start` become `12`. Check `while (start <= end)` is false [`(12 <= 11)`].

Step33: End of `do-while` loop

Step34: Quit `do-while` loop because `while (start <= end)` is false. There is no other statement after `do-while` loop, finally the program terminated when reached `return 0;`