To:
Tenouk

C LAB WORKSHEET 7
A C & C++ Repetition: The for Loop 1

Items in this page:

1. C program controls or loop - repetition.
2. The for loop.
3. The related tutorial reference for this worksheet are: C & C++ program control 1 and C/C++ program control 2.

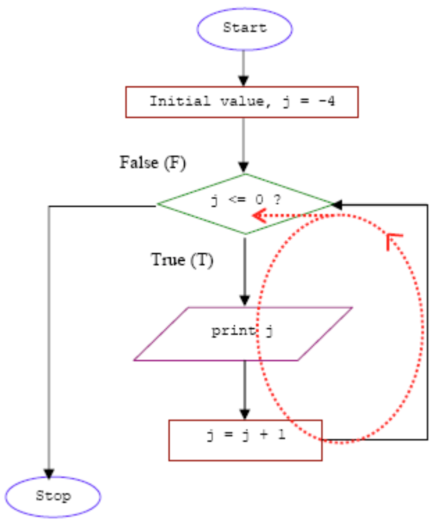
- In this practice we will concentrate on the program controls specifically loop constructs that used in C/C++ programming. We will use flowchart to assist us in understanding the program controls.
- Instead of a sequence of C/C++'s code execution in programs, in the real C/C++ programming there is a lot of conditional and branching. Other than a **sequence** code execution, a common loops (**repetition**) in C/C++ programming is the for, while and do-while loops. For conditional execution (**selection**) we can use if and its variations such as if-else, if-else-if and switch-case-break. Other constructs for conditional compilation includes using goto and exit however in this worksheet we will play around with for loop and its friends.

Flowchart and tracechart

- Flowchart shows the logic of a program graphically and optionally, tracechart shows the sequence of steps taken to execute the flowchart. Let have a C pseudocode for simple integer iteration program example.

    1. Declare an integer variable, j and set an initial value to -4, j = -4.
    2. Check the condition for j, j <= 0 true or false?
    3. Print the j's value.
    4. Increment j by 1.
    5. Repeat the iteration process until j <= 0 is false.
    6. If the condition j<=0 is false, stop exit the loop and continue the execution of the next code if any.

- In a simple word we are going to print an integer number from -4 to 0. From the pseudocode, let translate it to a flowchart, you will see a better C program logic. The **loop** (using for construct in this example) is depicted by the dash red line and it is not part of the flowchart.



- From the Figure:

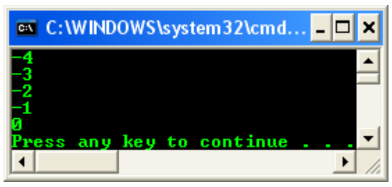
    1. The flowchart starts at the oval **Start** (some use **Begin**) and continues until the oval called **Stop** (some use **End**) is encountered – start or stop of the program execution.
    2. Assignment statements are shown in **rectangles** – assignment or operation statements.
    3. A decision is shown in a **diamond** – decision, yes or no, true or false.
    4. Printing and reading of values use **parallelograms** – printing and reading values (input/output).
    5. The flow directions are represented by **arrows** – flow direction.

- You may find other shapes such as **small circle** used in the flowchart but those used in the previous Figure are the common shapes.
- From the flowchart we can describe the detail of the steps taken for the loop execution in a tracechart. For this example we have 17 steps to complete the loop execution. The C/C++ code optimization for speed and program size can be done by reducing the steps taken and simplify the code used, provided the output still retained.

Loop count	Steps	j	j <= 0 or not?	printed value
Start				
Loop #1	1	-4		
	2		True	
	3			-4
Loop #2	4	-3		
	5		True	
	6			-3
Loop #3	7	-2		
	8		True	
	9			-2
Loop #5	10	-1		
	11		True	
	12			-1
Loop #6	13	0		
	14		True	
	15			0
Stop	16	1		
	17		False (loop stops)	

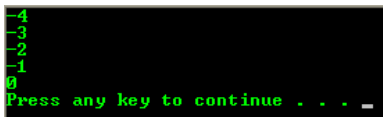
- From the flowchart we can do the program coding and test the output as shown below with a variation of coding styles. Create a project named **progcontrol**. Then add C++ source file named **progcontrolsrc** to the project. Don't forget to set your project to be compiled as C code. Try the following example and see the output. Study the for loop construct and the many faces of coding styles.

```
#include <stdio.h>

void main()
{
      int j;
      j = -4;
      for( ; j <= 0 ; )
      {
            printf("%d\n", j);
            j = j + 1;
      }
}
```



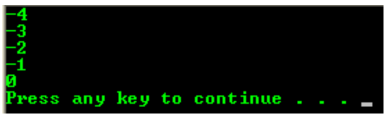
```
-4
-3
-2
-1
0
Press any key to continue . . .
```

```
#include <stdio.h>

void main()
{
      int j = -4;
      for( ; j <= 0 ; )
      {
            printf("%d\n", j);
            j = j + 1;
      }
}
```



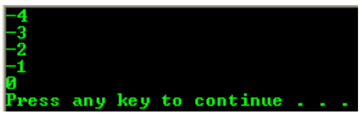
```
-4
-3
-2
-1
0
Press any key to continue . . .
```

```
#include <stdio.h>

void main()
{
      int j;
      for(j = -4; j <= 0 ; )
      {
            printf("%d\n", j);
            j = j + 1;
      }
}
```



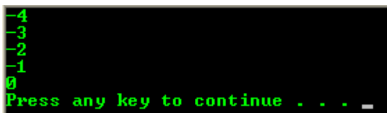
```
-4
-3
-2
-1
0
Press any key to continue . . .
```

```
#include <stdio.h>

void main()
{
      int j;
      for(j = -4; j <= 0 ; j = j + 1)
            printf("%d\n", j);
}
```



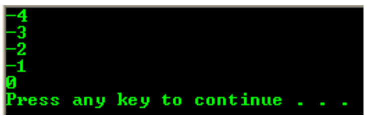
```
-4
-3
-2
-1
0
Press any key to continue . . .
```

```
#include <stdio.h>

void main()
{
      int j;
      for( j = -4; j <= 0 ; )
            printf("%d\n", j++);
}
```



```
-4
-3
-2
-1
0
Press any key to continue . . .
```

```
#include <stdio.h>

void main()
{
    int j;
    for( j = -4; j <= 0 ; j++)
        printf("%d\n", j);
}
```



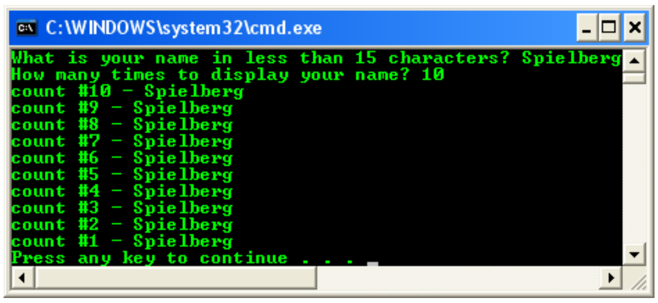
```
-4
-3
-2
-1
0
Press any key to continue . . .
```

- Notice that if only one statement after the for statement, then a set of braces is not required. They are needed only if more than one statements in the loop. Generally, the for statement will have the following form.

```
for(initial value; condition; iteration)
{
    C/C++ statement(s);
}
```

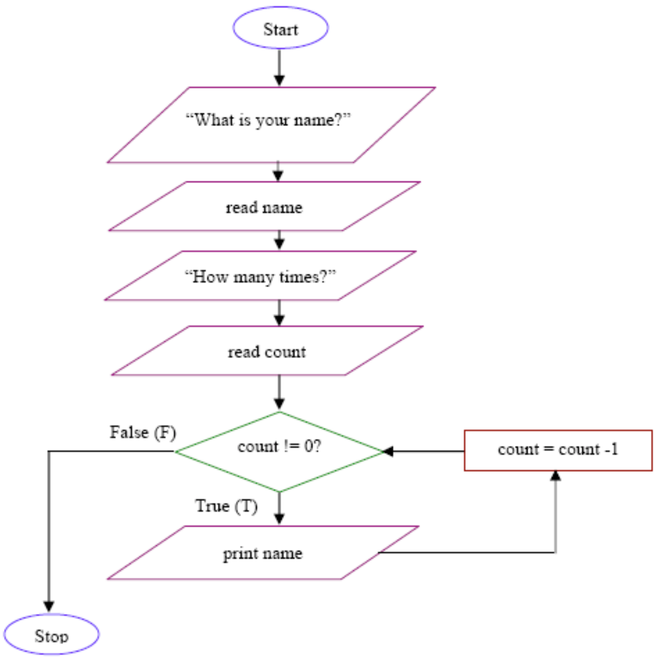
- Remember there is no semicolon at the end of the for statement. Try the following example and show the output.

```
#include <stdio.h>

void main(void)
{
    int count;
    char name[15];
    printf("What is your name in less than 15 characters? ");
    // scanf("%s", &name);  // older version
    scanf_s("%s", &name, 14);
    printf("How many times to display your name? ");
    // scanf("%d", &count); // older version
    scanf_s("%d", &count, 1);
    // iterate descendingly and the two following
    // commented out are the 'for' variations...
    // for(; count != 0; count = --count)
    // for(; count != 0; count = count--)
    for(; count != 0; count = count - 1)
        printf("count #%d - %s\n", count, name);
}
```



```
What is your name in less than 15 characters? Spielberg
How many times to display your name? 10
count #10 - Spielberg
count #9 - Spielberg
count #8 - Spielberg
count #7 - Spielberg
count #6 - Spielberg
count #5 - Spielberg
count #4 - Spielberg
count #3 - Spielberg
count #2 - Spielberg
count #1 - Spielberg
Press any key to continue . . .
```

- In this example, we first obtain the user’s name and save it in name[ ]. Then we obtain the number of times the name to be displayed. In the sample run shown, this number is 10 and it is stored in variable count. Then the loop begins. From the program we can construct the flowchart as shown below.



- The terminal condition is evaluated, count != 0? (is count not equal to 0?) is true, since count is equal to 10, then the body of the loop is executed. Since only one statement is in the body of the loop, no set of braces is required and the name is printed.
- Now we go up to the for statement and assign count to be 1 (10 – 1 = 9) less than what it was. Then evaluate again, count != 0? is still true so the loop is performed, that is the name is printed a second time. The iteration of the loop is perform until count != 0? is false, means count = 0 and the loop is stopped. Take note that the loop is stopped not the program itself if there are more codes after the loop body, the execution will continue.
- From the previous examples you can se that we can write down the C program from the flowchart and vice versa, we also can create a flowchart from C program. Typically we start designing our C program from scratch using pseudocode, and then we build the flowchart and finally write the real C codes. Optionally we can build the tracechart based on the flowchart for the loop part of the

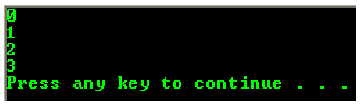
program to see the detail of the loop execution.

Activities And Questions

Run the following program, show the output and answer the question.

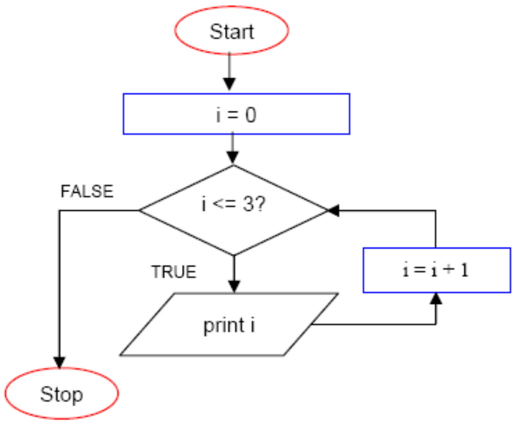
```
#include <stdio.h>

void main()
{
      int i;
      i = 0;
      for( ; i <= 3; )
      {
            printf("%d\n", i);
            i = i + 1;
      }
}
```

a. What is the first value of i?
b. What is the last value of i that is printed?
c. Build a flowchart for the program.
d. Change the i = 0; to i = 1 in the program and rebuild. What is the first value of i? What is the last value of i that is printed?

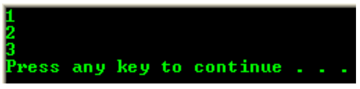


```
0
1
2
3
Press any key to continue . . .
```

a. 0.
b. 3.



c.

d.


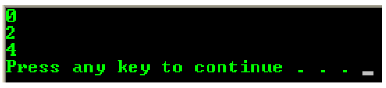
```
1
2
3
Press any key to continue . . .
```

The first value of i is 1 and the last value of i is 3.

Try the following program.

```
#include <stdio.h>

void main()
{
      int i;
      i = 0;                    // Statement 1
      for( ; i <= 4; )        // Statement 2
      {
            printf("%d\n", i);
            i = i + 2;         // Statement 3
      }
}
```

a. What is the first value of i? What is the last value of i that is printed?
b. Which statement determines the initial value? 1, 2 or 3?
c. Which statement determines the final value?
d. Which statement determines how the value of i is increased?



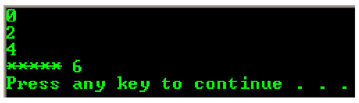
```
0
2
4
Press any key to continue . . . _
```

a. First value of i is 0 and the last value is 4.
b. Statement 1.
c. Statement 2.
d. Statement 3.

Run the following program.

```
#include <stdio.h>

void main()
{
      int i;
      i = 0;                       // Statement 1
      for( ; i <= 4; )           // Statement 2
      {
            printf("%d\n", i);
            i = i + 2;               // Statement 3
      }
      printf("***** %d\n", i);     // Statement 4
}
```

a. What is the first value of i?
b. What is the last value of i that is printed inside the loop?
c. Is statement 3 inside the loop? What about statement 4?
d. Which statement is done before the loop?
e. Which one is done after the loop?
f. How can we tell which statements are inside the loop?

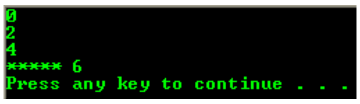


```
0
2
4
***** 6
Press any key to continue . . .
```

a. 0.
b. 4.
c. Statement 3 is inside the loop and Statement 4 is outside the loop.
d. Statement 1.
e. Statement 4.
f. Statements that are inside the curly braces ({ }) immediately after the for loop. These statements construct the for loop body.

Next, let simplify the previous program. Run and show the output.

```
#include <stdio.h>

void main()
{
    int i;
    for(i = 0; i <= 4;  i = i + 2)   // Statement 1
        printf("%d\n", i);           // Statement 2
    printf("***** %d\n", i);         // Statement 3
}
```



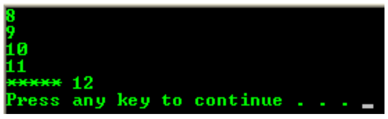
```
0
2
4
***** 6
Press any key to continue . . .
```

The for loop now shows three expressions. We will call the first expression as i = 0, the second expression i <= 4, and the third expression is i = i + 2.

a. Which of these three expressions determines the terminal condition of the loop?
b. Which one determines the incremental value of i?
c. Which one determines the initial value of i?
d. Is statement 2 in the loop or after it?
e. Is statement 3 in the loop or after it?

a. i <= 4.
b. i = i + 2.
c. i = 0.
d. In the loop.
e. After the loop. Without curly braces, only the first statement immediately after the for loop statement will be in the loop.

```
#include <stdio.h>

void main()
{
    int i;
    for(i = 8; i <= 11;  i = i + 1)    // Statement 1
        printf("%d\n", i);             // Statement 2
    printf("***** %d\n", i);           // Statement 3
}
```

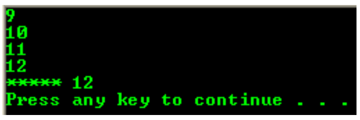


```
8
9
10
11
***** 12
Press any key to continue . . .
```

a. The for loop has three expressions separated by two semicolons. Which of these parts are assignments, that is a statement where a variable is being changed?
b. Which part sets the initial value of i?
c. Which part terminates the loop?
d. Which part increment i?

a. i <= 11 and i = i + 1.
b. i = 8.
c. i <=11.
d. i = i + 1.

```
#include <stdio.h>

void main()
{
    int i;
    for(i = 8; i <= 11;  printf("%d\n", i))   // Statement 1
        i = i + 1;                            // Statement 2
    printf("***** %d\n", i);                  // Statement 3
}
```

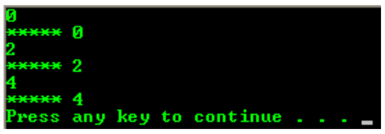


```
9
10
11
12
***** 12
Press any key to continue . . .
```

a. Why didn't the 8 get printed this time?
b. Why did 12 get printed twice, once inside the loop and once after the loop?

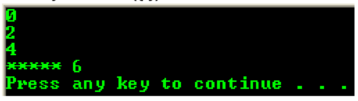
a. It is because when the initial value of i is 8, the statement 2, i = i + 1 makes i = 9 and then printed by the printf("%d\n", i) statement. Compared to the previous example, the code in the for loop will be **executed** first whereas the expressions in the for statement are just **evaluated** at the beginning, executed on the second and later iteration.

```
#include <stdio.h>

void main()
{
    int i;
    for(i = 0; i <= 4;  i = i + 2)      // Statement 1
    {
        printf("%d\n", i);              // Statement 2
        printf("***** %d\n", i);        // Statement 3
    }
}
```



```
0
***** 0
2
***** 2
4
***** 4
Press any key to continue . . .
```

a. Why is statement 3 being executed in the loop?
b. We must have a pair of what items if more than one statement is to be executed in the loop?
c. Remove the opening and closing curly braces of the for loop. Rebuild the program. Why statement 3 was not executed in the loop?

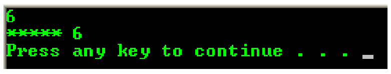
a. Because statement 3 is in the for loop body.
b. A curly braces ({ }).
c. 

```
0
2
4
***** 6
Press any key to continue . . .
```

Statement 3 was not executed because it is outside of the for loop body.

Add a semicolon at the end of the for and watch what happens.

```
#include <stdio.h>

void main()
{
    int i;
    for(i = 0; i <= 4;  i = i + 2);     // Statement 1
    {
        printf("%d\n", i);              // Statement 2
        printf("***** %d\n", i);        // Statement 3
    }
}
```



```
6
***** 6
Press any key to continue . . .
```

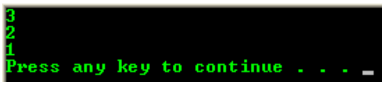
a. Was statement 2 executed inside the loop?
b. Was statement 3 executed inside the loop?
c. What was the only difference between this experiment and the previous one?

a. No.
b. No.
c. A semicolon was added at the end of the for statement. In this case the for loop becomes a single, isolated C statement. Only expressions in the for loop, that is (i = 0; i <= 4; i = i + 2) have been evaluated and executed then the for loop terminates, leaving the final value of i equal to 6 = 4 + 2. The statements in the curly braces will print the final value of i, that is 6, once for each printf() statement.

Next, let try making i go backward (decrement) by changing <= to >= operator.

```
#include <stdio.h>

void main()
{
     int i = 4;
     for(; i >= 2;)
     {
          i = i - 1;
          printf("%d\n", i);
     }
}
```

a. Initially, the condition i >= 2? True or false?
b. With every iteration of the loop, this condition is closer and closer to becoming true or false?

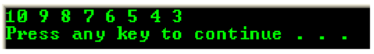


```
3
2
1
Press any key to continue . . .
```

a. True.
b. False.

What about the not equal to operator, != as shown below.

```
#include <stdio.h>

void main()
{
     int i;
     for(i = 10; i != 2; i = i - 1)
          printf("%d ", i);
     printf("\n");
}
```

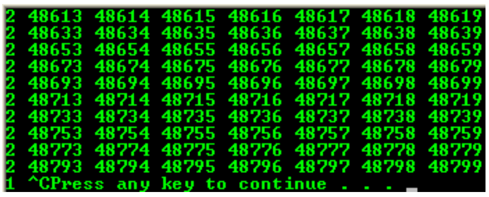


```
10 9 8 7 6 5 4 3
Press any key to continue . . .
```

The printf("%d ", i); statement will be executed except when i = = 2.

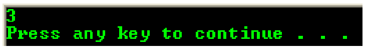
What happen to the following program output? How to stop it? How would you correct it?

```
#include <stdio.h>

void main()
{
     int i;
     // here we set the initial value to 3, but the condition
     // is != 2, print and increment the i by 1...
     // the i != 2 is true forever!
     for(i = 3; i != 2; i = i + 1)
          printf("%d ", i);
}
```



```
2 48613 48614 48615 48616 48617 48618 48619
2 48633 48634 48635 48636 48637 48638 48639
2 48653 48654 48655 48656 48657 48658 48659
2 48673 48674 48675 48676 48677 48678 48679
2 48693 48694 48695 48696 48697 48698 48699
2 48713 48714 48715 48716 48717 48718 48719
2 48733 48734 48735 48736 48737 48738 48739
2 48753 48754 48755 48756 48757 48758 48759
2 48773 48774 48775 48776 48777 48778 48779
2 48793 48794 48795 48796 48797 48798 48799
1 ^CPress any key to continue . . .
```

The output of this program will be infinite number because the for loop doesn't have a terminal condition. The output is 'overflow'. The i != 2 expression is always true in this case. To stop it press Ctrl + C.
We need to provide a terminal condition for the loop. In this case we can change the i = i + 1 to i = i - 1 to provide a terminal condition. So we need to provide a terminal condition in the for statement else the for loop won't terminate.



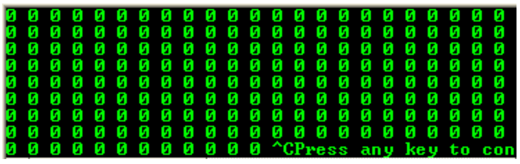
```
3
Press any key to continue . . .
```

Compress or simplify the following code as much as possible by retaining the output.

```
#include <stdio.h>

void main()
{
     int i = 0;
     for(; i < 5;)
          printf("%d ", i);
     i = i + 1;
}
```

a. Which part is the terminal condition? Which statement brings the loop closer to the terminal condition with every iteration? Which statement initializes the loop?
b. What is the operator for the assignment statement? What is the conditional operator for equality? Which statement changes a value? Which statement checks only a value?
c. If a loop doesn't approach the terminal condition with every iteration but diverges from it, what kind of loop is it?

The program seems can't be simplified anymore. In this program i has been initialized to 0. Then the for statement only provide i < 5 condition expression. The for loop statement that has been executed indefinitely is printf("%d ", i);. In this program, the i < 5 is always true because there is no iteration (decrement or increment) to make it closer to the terminal value, i < 5. The for loop run continuously while printing the initial value of i = 0.



```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 ^CPress any key to con
```

a. There is no terminal condition. No statement will bring the loop closer to the terminal condition with every iteration. Statement int i = 0; initializes the loop.
b. An equal sign, = is an assignment operator. Statement i = i + 1 changes a value. Statement i < 5 only check a value.
c. An infinite loop.