


<b>Soal Praktikum</b> <i>Practicum Case</i>	
<b>COMP6360004</b> <b>Algorithm and Programming</b>	
<b>Teknik Informatika</b> <i>Computer Science</i>	<b>E222-COMP6360004-EP01-04</b>
<b>Periode Berlaku</b> Semester Ganjil 2022/2023 <i>Valid on Odd Semester Year 2022/2023</i>	<b>Revisi 00</b> <i>Revision 00</i>

### Learning Outcomes

- Apply syntax and functions in C language in problem solving
- Construct a program using C language in problem solving

### Topic

- Session 04 - Program Control Repetition

### Sub Topics

- For
- While
- Do-While
- Break vs Continue
- Create simple program using repetition

### Tutorial/Panduan

Most programs involve repetition, or looping. A loop is a group of instructions the computer executes repeatedly while some loop-continuation condition remains true. There are two types of repetition: **counter-controlled repetition** and **sentinel-controlled repetition**. Counter-controlled repetition is sometimes called definite repetition because we know in advance exactly how many times the loop will execute. Sentinel-controlled repetition is sometimes called indefinite repetition because it's not known in advance how many times the loop will execute. In counter-controlled repetition, a control variable is used to count the number of repetitions. The control variable is incremented (usually by 1) each time the group of instructions is performed. When the correct number of repetitions has been performed, the loop terminates, and the program resumes execution with the statement after the repetition statement. Sentinel values are used to control repetition when the number of repetitions is not known in advance, and the loop includes statements that obtain data each time the loop is performed. The sentinel value indicates "end of data." The sentinel is entered after all regular data items have been supplied to the program. Sentinels must be distinct from regular data items. Counter-controlled repetition requires:

- (1.) The name of a control variable (or loop counter).
- (2.) The initial value of the control variable.
- (3.) The increment (or decrement) by which the control variable is modified each time through the loop.
- (4.) The condition that tests for the final value of the control variable (i.e., whether looping should continue).

```

#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    int counter = 1; /* initialization */

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

    return 0; /* indicate program ended successfully */
} /* end function main */

```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

Consider this simple program,  
which prints the numbers from 1 to 10.

The definition names the control variable (counter), defines it to be an integer, reserves memory space for it, and sets it to an initial value of 1. This definition is not an executable statement.

## for repetition statement

The for repetition statement handles all the details of counter-controlled repetition. When the for statement begins executing, its control variable is initialized. Then, the loop-continuation condition is checked. If the condition is true, the loop's body executes. The control variable is then incremented, and the loop begins again with the loop-continuation condition. This process continues until the loop-continuation condition fails. The general format of the for statement is this where expression1 initializes the loop-control variable, expression2 is the loop-continuation condition, and expression3 increments the control variable.

```
for ( expression1; expression2; expression3 )
    statement
```

```

#include <stdio.h>

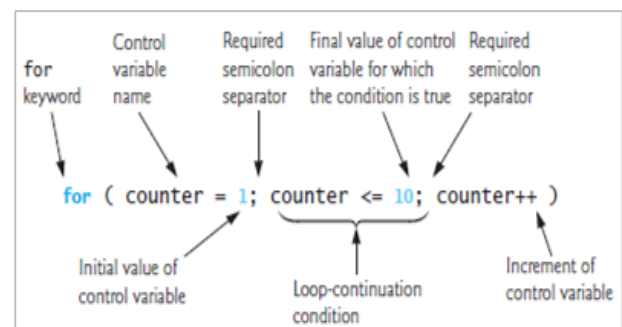
/* function main begins program execution */
int main( void )
{
    int counter; /* define counter */

    /* initialization, repetition condition, and increment
    are all included in the for statement header. */
    for ( counter = 1; counter <= 10; counter++ ) {
        printf( "%d\n", counter );
    } /* end for */

    return 0; /* indicate program ended successfully */
} /* end function main */

```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10



The program operates as follows. When the for statement begins executing, the control variable counter is initialized to 1. Then, the loop-continuation condition counter <= 10 is checked. Because the initial value of counter is 1, the condition is satisfied, so the printf statement (line 13) prints the value of counter, namely 1. The control variable counter is then incremented by the expression counter++, and the loop begins again with the loop-continuation test. Since the control variable is now equal to 2, the final value is not exceeded, so the program performs the printf statement again. This process continues until the control variable counter is incremented to its final value of 11—this causes the loop-continuation test to fail, and repetition terminates. The program continues by performing the first statement after the for statement (in this case, the return statement at the end of the program).

### Syntax :

```

int x, y;
for (x=1; x<=3; x++)
    for (y=5; y>=1; y--)
        printf("%d %d ", x, y);

```

### Output :

15 14 13 12 11 25 24 23 22 21 35 34 33 32 31

For can also be formed in nested for statement (loop in a loop). The repetition operation will start from the inner side loop.

**while statement and do..while statement**

In the while statement, the loop-continuation condition is tested at the beginning of the loop before the body of the loop is performed. The do...while statement tests the loop-continuation condition after the loop body is performed. Therefore, the loop body will be executed at least once. When a do...while terminates, execution continues with the statement after the while clause. It's not necessary to use braces in the do...while statement if there is only one statement in the body. However, the braces are usually included to avoid confusion between the while and do...while statements. Expression in both while statement and do..while statement is Boolean expression. It will result in true (not zero) or false (equal to zero).

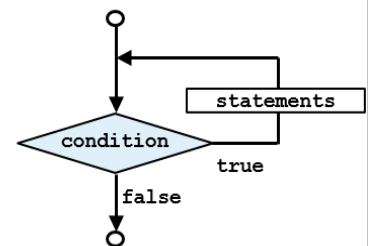
In **while statement**,  
the statement will be executed while  
the expression is not equal to zero.  
Expression evaluation is done  
before the statements executed.

**Syntax :**

*while (expression) statements;*

**or**

```
while (expression) {
    statement1;
    statement2;
    .....
}
```

**Flow Chart of WHILE Statement**

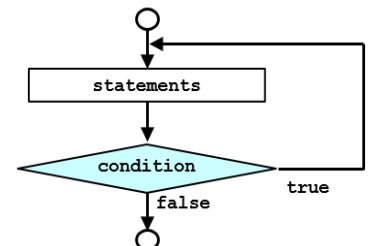
<p>For example, print the value 1 to 10.</p>	<p><b>Syntax:</b></p> <pre>#include&lt;stdio.h&gt; int main() {     int counter = 1;     while ( counter &lt;= 10 ) {         printf( "%d\n", counter );         counter++;     } }</pre>	<p><b>Output:</b></p> <pre>1 2 3 4 5 6 7 8 9 10</pre> <p><b>Flow Chart of WHILE Statement</b></p> <pre> graph TD     Start(( )) --&gt; Condition{counter &lt;= 10}     Condition -- true --&gt; Statements[printf("%d\n", counter); counter++;]     Statements --&gt; Condition     Condition -- false --&gt; End(( ))   </pre>
--	---	---

In **do..while statement**,  
it will keep executing  
while expression is true.

Expression evaluation is done  
after executing the statement(s)

**Syntax :**

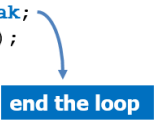
```
do {
    < statements >;
} while (expression);
```

**Flow Chart of DO-WHILE Statement**

<p>For example, print the value 1 to 10.</p>	<p><b>Syntax:</b></p> <pre>#include&lt;stdio.h&gt; int main() {     int counter=0;     do {         printf( "%d\n", counter );         counter++;     } while (counter &lt;= 10); }</pre>	<p><b>Output:</b></p> <pre>1 2 3 4 5 6 7 8 9 10</pre> <p><b>Flow Chart of DO-WHILE Statement</b></p> <pre> graph TD     Start(( )) --&gt; Statements[printf( "%d\n", counter ); counter++;]     Statements --&gt; Condition{counter &lt;= 10}     Condition -- true --&gt; Statements     Condition -- false --&gt; End(( ))   </pre>
--	---	---

## break and continue statements

- The **break** statement, when executed in a while, for, do...while or switch statement, causes immediate exit from that statement. Program execution continues with the next statement.
- The **continue** statement, when executed in a while, for or do..while statement, skips the remaining statements in the body of that control statement and performs the next iteration of the loop. In while and do..while statements, the loop-continuation test is evaluated immediately after the continue statement is executed. In the for statement, the increment expression is executed, then the loop-continuation test is evaluated.

Example using <b>break</b>	Example using <b>continue</b>
<pre>#include &lt;stdio.h&gt; int main() {     int x;     for(x=1; x&lt;=10; x++) {         if (x == 5) <b>break</b>;         printf("%d ", x);     }     return 0; }</pre> 	<pre>#include &lt;stdio.h&gt; int main() {     int x;     for(x=1; x&lt;=10; x++) {         if (x == 5) <b>continue</b>;         printf("%d ", x);     }     return 0; }</pre>
<b>Output : 1 2 3 4</b>	<b>Output : 1 2 3 4 6 7 8 9 10</b>

## Soal

### Case

- One large chemical company pays its salespeople on a commission basis. The salespeople receive \$200 per week plus 9% of their gross sales for that week. For example, a salesperson who sells \$5000 worth of chemicals in a week receives \$200 plus 9% of \$5000, or a total of \$650. Using **while** and **do..while statement**, develop a program that will input each salesperson's gross sales for last week and will calculate and display that salesperson's earnings. Process one salesperson's figures at a time. Here is a sample input/output dialog:

```
Enter sales in dollars (-1 to end): 5000.00
Salary is: $650.00

Enter sales in dollars (-1 to end): 1234.56
Salary is: $311.11

Enter sales in dollars (-1 to end): 1088.89
Salary is: $298.00

Enter sales in dollars (-1 to end): -1
```

- Create a C program to print :

```
*
* *
* * *
* * * *
* * * * *
```

- Solve all of this week's exercises that are available on <https://socs1.binus.ac.id/quiz/>

## References:

Paul Deitel & Harvey Deitel. (2016). C how to program:with an introduction to C++. 08. Pearson Education. Hoboken.ISBN: 9780133976892.