

SIES: BASIC C PROGRAMMING

L #09: LOOPS

Seung Beop Lee

School of International Engineering and Science

CHONBUK NATIONAL UNIVERSITY

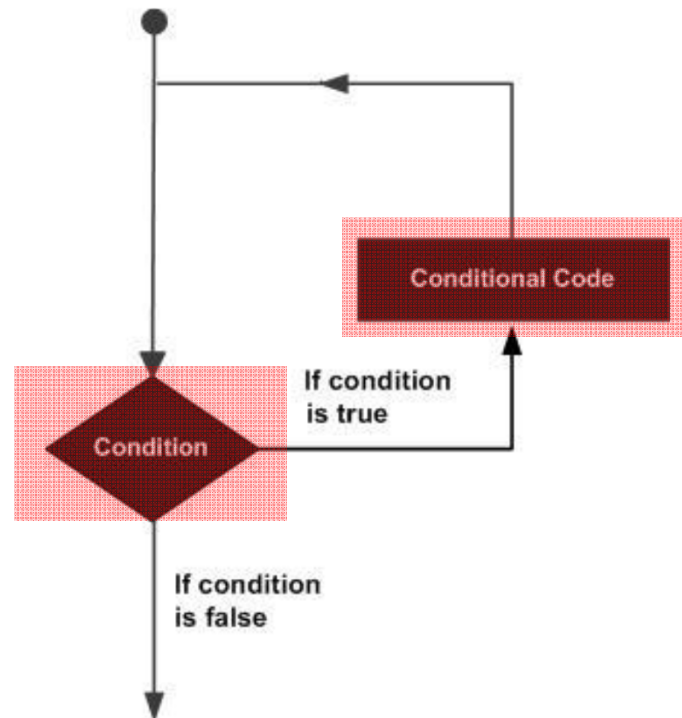
Outline

- **Loops**
- **Loop control statement**

Loops

Loops

- What is the loop in C?
 - A loop statement allows us to execute a statement or group of statements **multiple times**.
 - The following figure is the general form of a loop statement in most of the programming languages.
 - Most of the loop statement have a **condition** and **conditional code**.



Loops

▪ Loops in C

- C programming language provides the **five types** of loops to handle looping requirements.
 - while loop
 - for loop
 - do...while loop
 - nested loop
 - Infinite loop

Loops

■ while loop

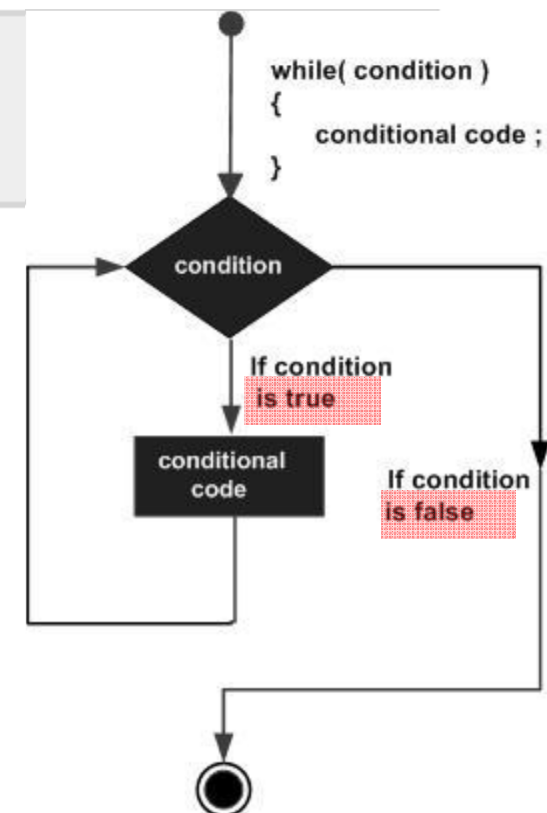
- A **while** loop statement in C programming language repeatedly executes a target statement **as long as a given condition is true**.

■ Syntax

- The syntax of a **while** loop in C programming language is:

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

- The **statement (i.e. conditional code)** can be a **single** statement or a **block** of statements. The condition may be any expression, and true is any nonzero value.
- **Note** that The loop iterates while the condition is **true**.
- When the condition becomes **false**, program control passes to the line immediately following the loop.

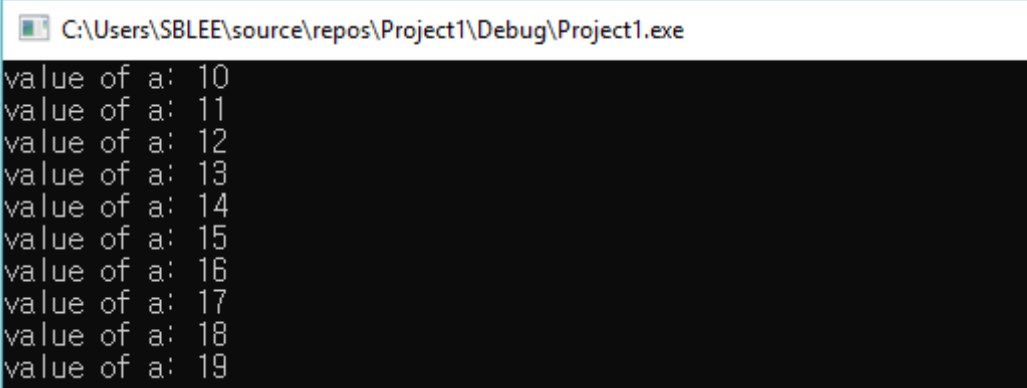


Loops

■ while loop

- One of the key points of the while loop is that the loop might not ever run.
- When the condition is tested and the result is **false**, the loop body will be skipped and the first statement after the while loop will be executed.
- The following example is in order to understand the **while** loop available in C:

```
1  #include <stdio.h>
2  int main()
3  {
4      /* local variable definition */
5      int a = 10;
6
7      /* while loop execution */
8      while (a < 20) // when the condition is true, the following statements will be executed.
9      {
10         printf("value of a: %d\n", a);
11         a++; //Increment operator(++) increases the integer value by one.
12     }
13
14     return 0;
15 }
```



```
C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Loops

- for loop

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to **execute a specific number of times**.

- Syntax

- The syntax of a **for** loop in C programming language is:

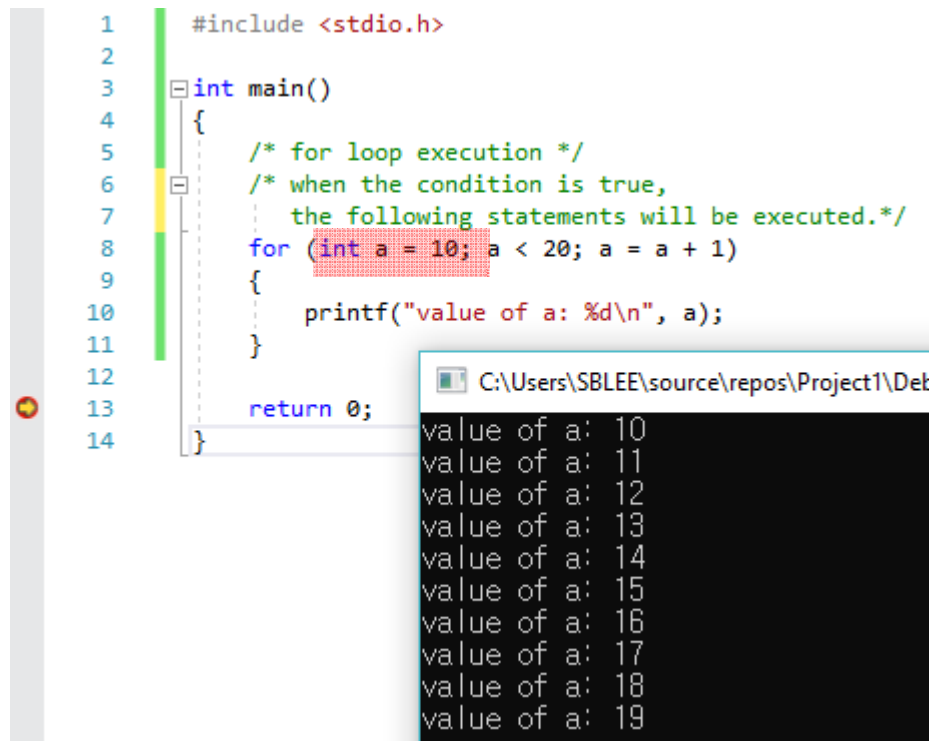
```
for ( init; condition; increment )  
{  
    statement(s);  
}
```

- ✓ The **init** step is executed first, and only once. This step allows you to **declare** and **initialize** any **loop control variables**. You are not required to put a statement here, as long as a semicolon appears.
- ✓ Next, the **condition** is evaluated. If it is **true**, the body (i.e. statement) of the loop is **executed**. If it is **false**, the body of the loop does **not execute** and flow of control **jumps to the next statement** just after the for loop.

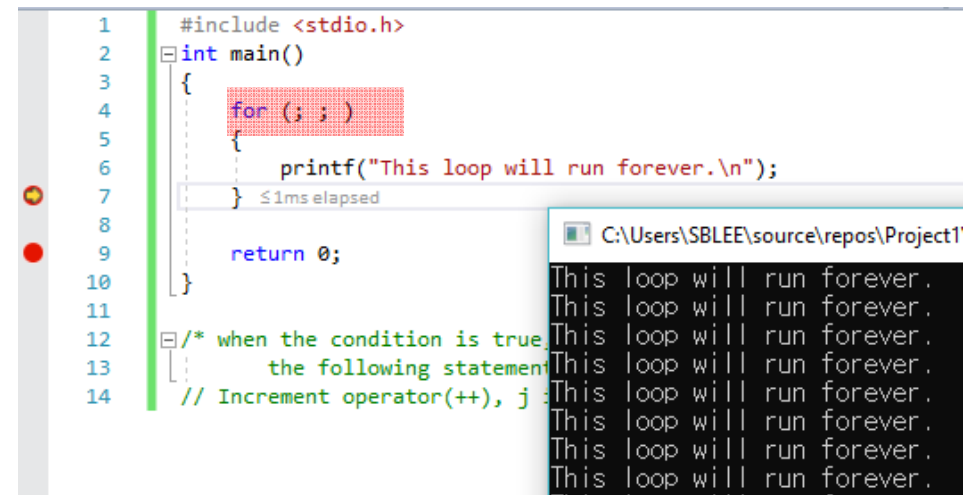
Loops

- Flow Diagram of the for loop

```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* for loop execution */
6      /* when the condition is true,
7       the following statements will be executed.*/
8      for (int a = 10; a < 20; a = a + 1)
9      {
10         printf("value of a: %d\n", a);
11     }
12
13     return 0;
14 }
```



```
1  #include <stdio.h>
2
3  int main()
4  {
5      for (; ; )
6      {
7          printf("This loop will run forever.\n");
8      }
9
10     return 0;
11 }
12
13 /* when the condition is true,
14 the following statement
15 // Increment operator(++), j :
```



Loops

- for loop

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to **execute a specific number of times**.

- Syntax

- The syntax of a **for** loop in C programming language is:

```
for ( init; condition; increment )  
{  
    statement(s);  
}
```

- ✓ The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- ✓ Next, the **condition** is evaluated. If it is **true**, the body of the loop is **executed**. If it is **false**, the body of the loop does **not execute** and flow of control **jumps to the next statement** just after the for loop.

Loops

- for loop

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to **execute a specific number of times**.

- Syntax

- The syntax of a **for** loop in C programming language is:

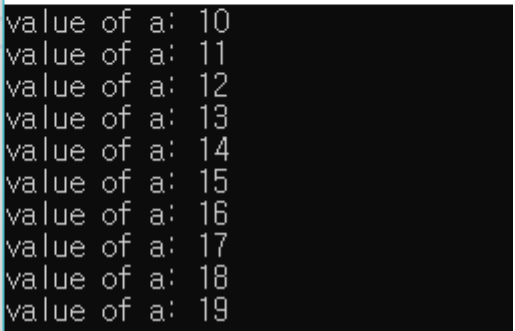
```
for ( init; condition; increment )  
{  
    statement(s);  
}
```

- ✓ **After** the body of the for loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to **update any loop control variables**. This statement also can be left blank, as long as a semicolon appears after the condition.
- ✓ The condition is **now evaluated again**. If it is **true**, the loop **executes** and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes **false**, the for loop **terminates**.

Loops

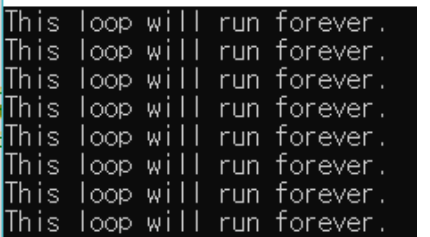
- Flow Diagram of the for loop

```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* for loop execution */
6      /* when the condition is true,
7       the following statements will be executed.*/
8      for (int a = 10; a < 20; a = a + 1)
9      {
10         printf("value of a: %d\n", a);
11     }
12
13     return 0;
14 }
```



value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19

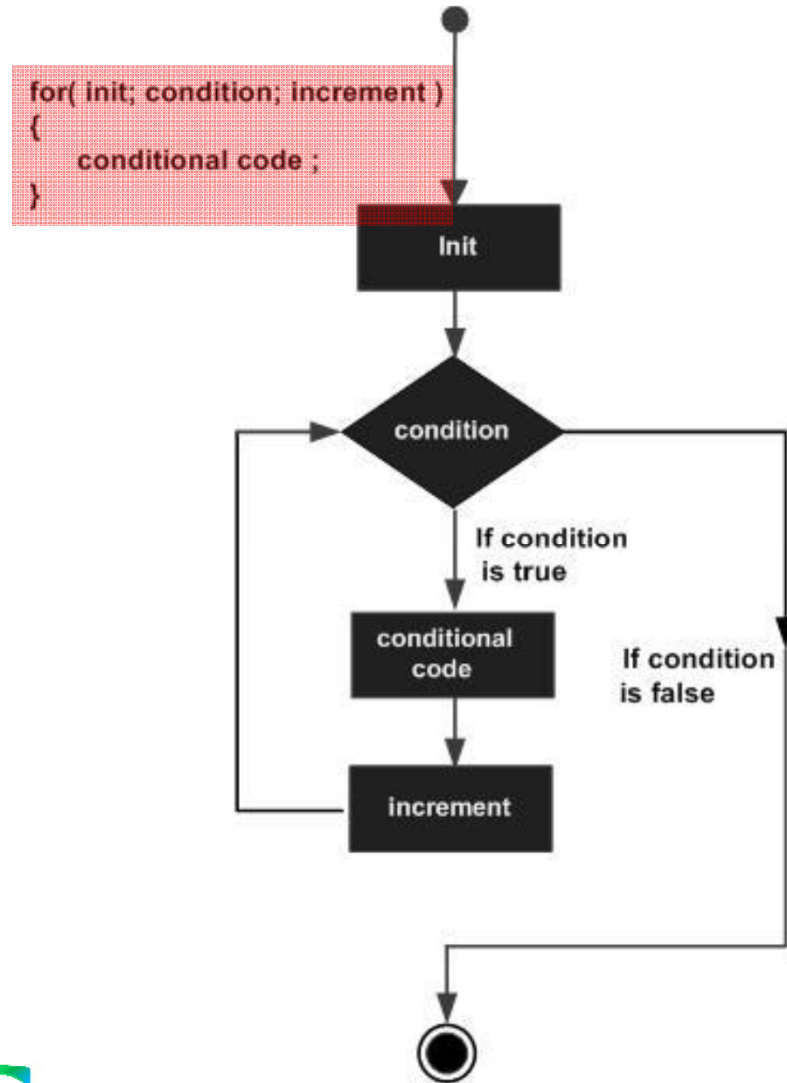
```
1  #include <stdio.h>
2  int main()
3  {
4      for (; ; )
5      {
6          printf("This loop will run forever.\n");
7      }
8
9      return 0;
10 }
```



This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.

Loops

- Flow Diagram of the for loop



```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* for loop execution */
6      /* when the condition is true,
7       the following statements will be executed.*/
8      for (int a = 10; a < 20; a = a + 1)
9      {
10         printf("value of a: %d\n", a);
11     }
12
13     return 0;
14 }
```

C:\Users\SBLEE\source\repos\Project1\Det

value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19

Loops

▪ do...while loop

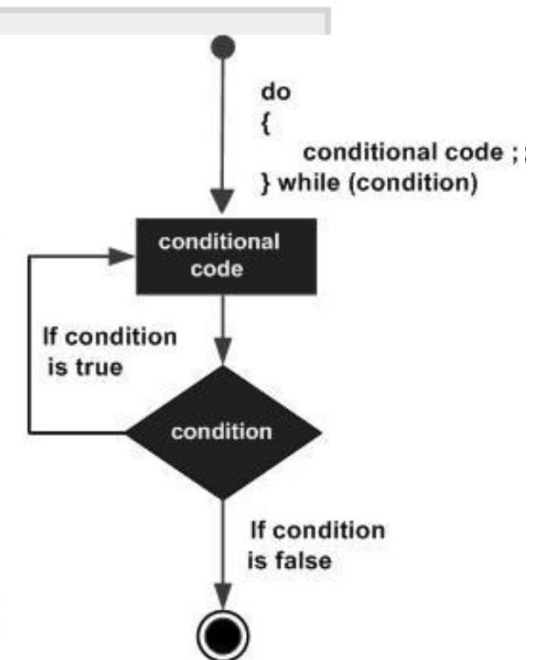
- Unlike **for** and **while** loops, which test the loop condition at the **top** of the loop, the **do...while** loop in C programming language checks its condition at the **bottom** of the loop.
- A **do...while** loop is similar to a while loop, except that a **do...while** loop is guaranteed to **execute at least one time**.

▪ Syntax

- The syntax of a **do...while** loop in C programming language is:

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

- Notice that the conditional expression appears at the **end** of the loop, so the statement(s) in the loop **execute once** before the condition is tested.
- If the condition is **true**, the flow of control jumps back up to **do**, and the statement(s) in the loop execute again. This process repeats until the given condition becomes **false**.

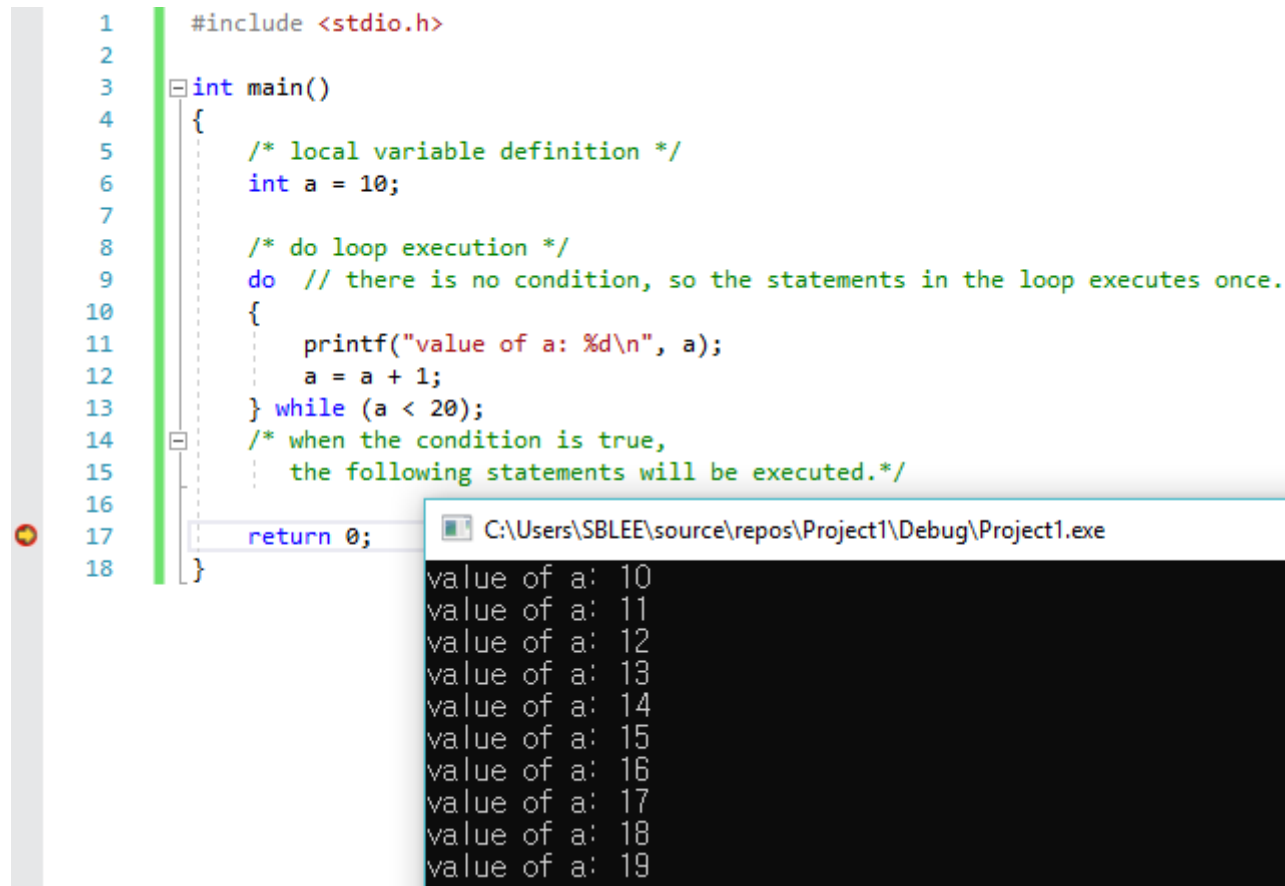


Loops

- do...while loop

- The following example is in order to understand the **do...while** loop available in C:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* local variable definition */
6      int a = 10;
7
8      /* do loop execution */
9      do // there is no condition, so the statements in the loop executes once.
10     {
11         printf("value of a: %d\n", a);
12         a = a + 1;
13     } while (a < 20);
14     /* when the condition is true,
15        the following statements will be executed.*/
16
17     return 0;
18 }
```



Loops

▪ Nested loops

- C programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

▪ Syntax

- The syntax for a **nested for loop** in C programming language is as follows:

```
for ( init; condition; increment )
{
    for ( init; condition; increment )
    {
        statement(s);
    }
    statement(s);
}
```

- The syntax for a **nested while** and **nested do...while** loop in C programming

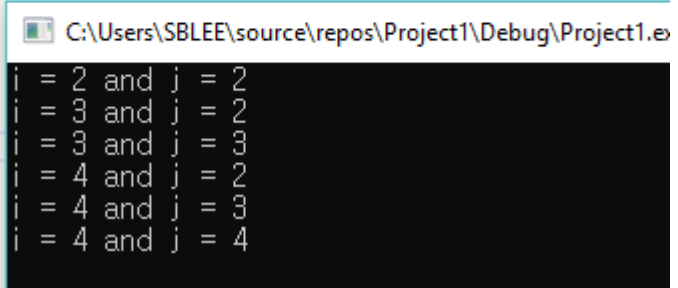
```
while(condition)
{
    while(condition)
    {
        statement(s);
    }
    statement(s);
}
```

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


Decision Making in C

- Example of a nested loop
 - The following example is in order to understand the **nested for loop** available in C:

```
1  #include <stdio.h>
2  int main()
3  {
4      /* local variable definition */
5      int i, j;
6
7      // nested for loop
8
9      for (i = 2; i < 5; i++) // Increment operator(++), i increases the integer value by one.
10     {
11         for (j = 2; j <= i; j++) // Increment operator(++), j increases the integer value by one.
12         {
13             printf("i = %d and j = %d \n", i, j);
14         }
15     }
16
17     return 0;
18 }
19
20 /* when the condition is true,
21    the following statements will be executed.*/
```

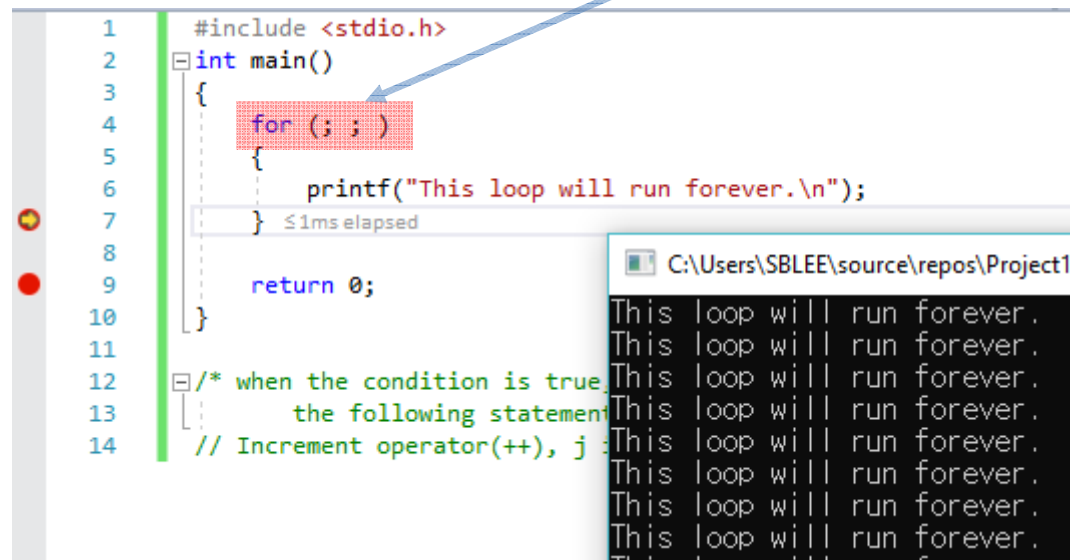


```
= 2 and j = 2
= 3 and j = 2
= 3 and j = 3
= 4 and j = 2
= 4 and j = 3
= 4 and j = 4
```

Loops

▪ Infinite loop

- A loop becomes **infinite** loop if a condition **never** becomes **false**. The **for loop** is traditionally used for this purpose. Since **none** of the three expressions that form the **for loop** are required, you can make an **endless loop** by leaving the conditional expression empty.



```
1  #include <stdio.h>
2  int main()
3  {
4      for (; ; )
5      {
6          printf("This loop will run forever.\n");
7      }
8
9      return 0;
10 }
11
12 /* when the condition is true,
13    the following statement
14    // Increment operator(++), j
```

C:\Users\SBLEE\source\repos\Project1'

This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.

- When the conditional expression is **absent**, it is assumed to be **true**. You may have an initialization and increment expression, but C programmers more commonly use the **for(;;)** construct to signify an **infinite loop**.

- NOTE:** You can terminate an infinite loop by pressing **Ctrl + C** keys.

Loop control statement

Loop control statement

- Loop control statement

- Loop control statements **change execution** from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- C supports the following control statements.

Control Statement	Description
break statement	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
goto statement	Transfers control to the labeled statement.

Loop control statement

▪ break statement

- The **break** statement in C programming has the following two usages:
 1. When the **break** statement is encountered inside a loop, the loop is **immediately terminated** and program control **resumes** at the next statement following the loop.
 2. It can be used to **terminate** a case in the **switch** statement (covered in the next chapter).
- If you are using **nested loops** (i.e., one loop inside another loop), the **break** statement will **stop** the execution of the **innermost loop** and **start** executing the next line of code after the block.

▪ Syntax

- The syntax for a **break** statement in C is as follows:

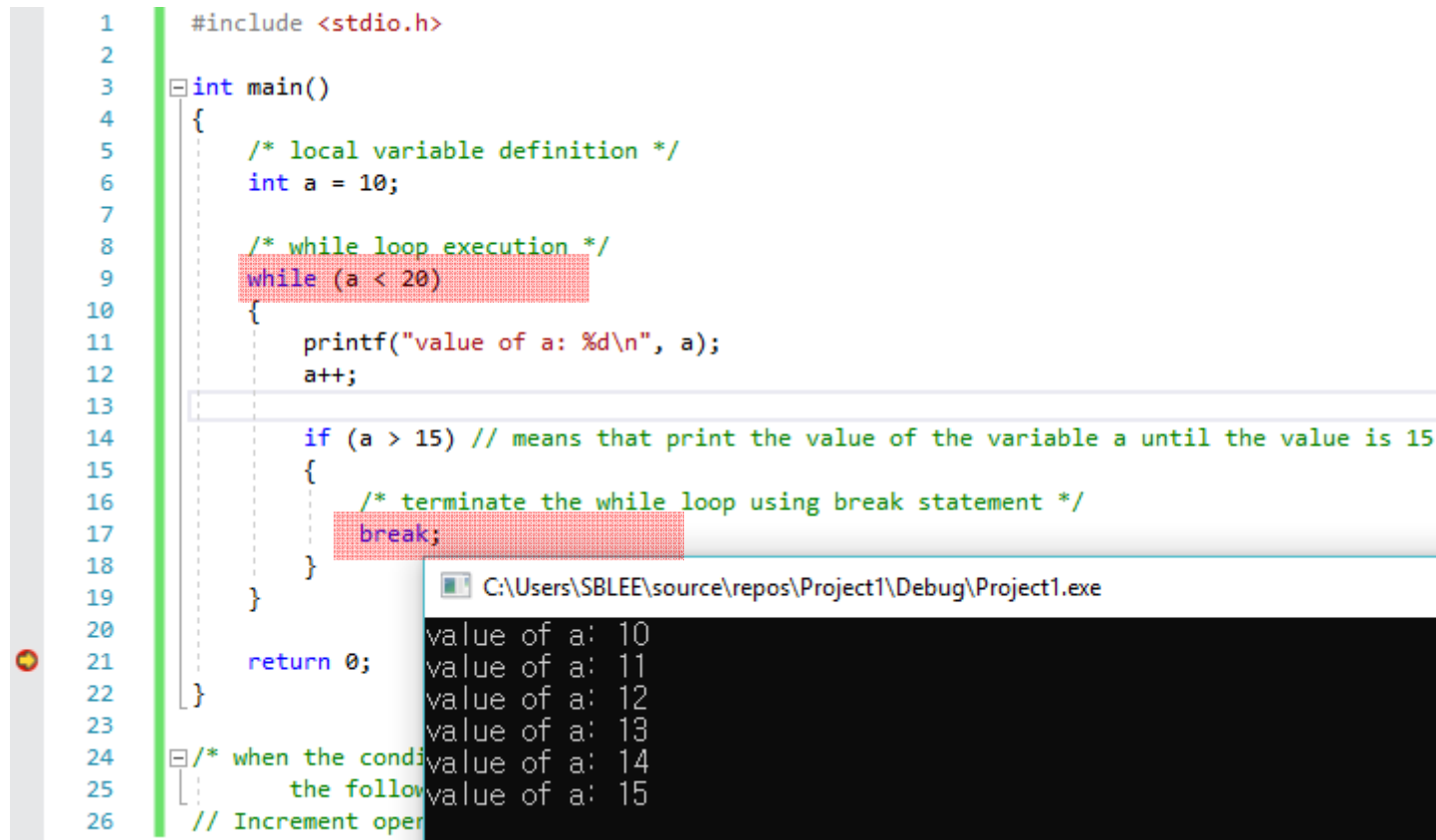
```
break;
```

Loop control statement

▪ Example of a break statement

- The following example is in order to understand the **break** statement available in C:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* local variable definition */
6      int a = 10;
7
8      /* while loop execution */
9      while (a < 20)
10     {
11         printf("value of a: %d\n", a);
12         a++;
13
14         if (a > 15) // means that print the value of the variable a until the value is 15.
15         {
16             /* terminate the while loop using break statement */
17             break;
18         }
19     }
20
21     return 0;
22 }
23
24 /* when the condition is true, the following code will be executed
25 // Increment operation
```



C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe

value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15

Loop control statement

- continue statement

- The continue statement in C programming works somewhat like the break statement.
- Instead of forcing termination of the break statement, it forces the **next iteration of the loop** to take place, **skipping any code in between**.

- Syntax

- The syntax for a **continue** statement in C is as follows:

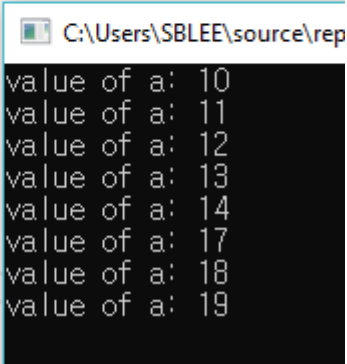
```
continue;
```

Loop control statement

▪ Example of a continue statement

- The following example is in order to understand the **continue** statement available in C:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* local variable definition */
6      int a = 10;
7
8      /* do loop execution */
9      do
10     {
11         if (a == 15)
12         {
13             /* skip the iteration */
14             a = a + 2;
15             continue;
16         }
17
18         printf("value of a: %d\n", a);
19         a++;
20
21     } while (a < 20);
22
23     return 0;
24 }
25
```



Loop control statement

- goto statement

- A **goto** statement in C programming language provides an unconditional jump from the **goto** to a **labeled statement** in the same function.
- NOTE: Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a **goto** can be rewritten so that it doesn't need the **goto**.

- Syntax

- The syntax for a **goto** statement in C is as follows:

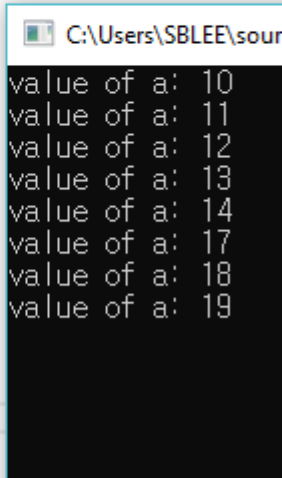
```
goto label;  
..  
.  
label: statement;
```

Loop control statement

- Example of a break statement

- The following example is in order to understand the **goto** statement available in C:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* local variable definition */
6      int a = 10;
7
8      /* do loop execution */
9      LOOP:do
10     {
11         if (a == 15)
12         {
13             /* skip the iteration */
14             a = a + 2;
15             goto LOOP;
16         }
17
18         printf("value of a: %d\n", a);
19         a++;
20
21     } while (a < 20);
22
23     return 0;
24 }
```



Summary

Summary

- ✓ A **loops** in C programming such as **while** loop, **for** loop, **do...while** loop, **nested** loops, and **infinite** loop were covered.
- ✓ Next, the **loop control statements** such as the **break** statement, **continue** statement, and **goto** statement were considered in order to control (leave) a scope.

Thank You

(seungbeop.lee@gmail.com)