CSE 06131223 ♦ CSE 06131224

Structured Programming

Lecture 9

Decision Making and Branching in C (2)

Prepared by

Md. Mijanur Rahman, Prof. Dr.
Dept. of Computer Science and Engineering
Jatiya Kabi Kazi Nazrul Islam University, Bangladesh
www.mijanrahman.com

Contents

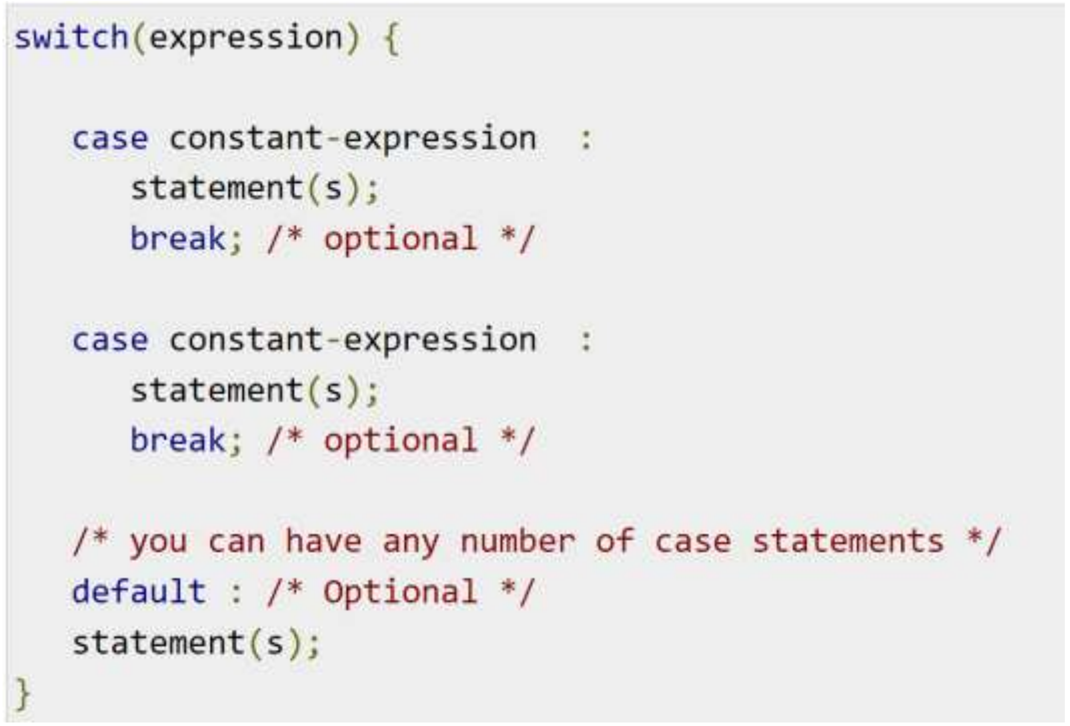
DECISION MAKING AND BRANCHING IN C

Switch Statement

- The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable.
- Here, We can define various statements in the multiple cases for the different values of a single variable.
- Thus, a **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

Switch Statement

- **Syntax:**
- The syntax for a **switch** statement in C programming language is as follows –



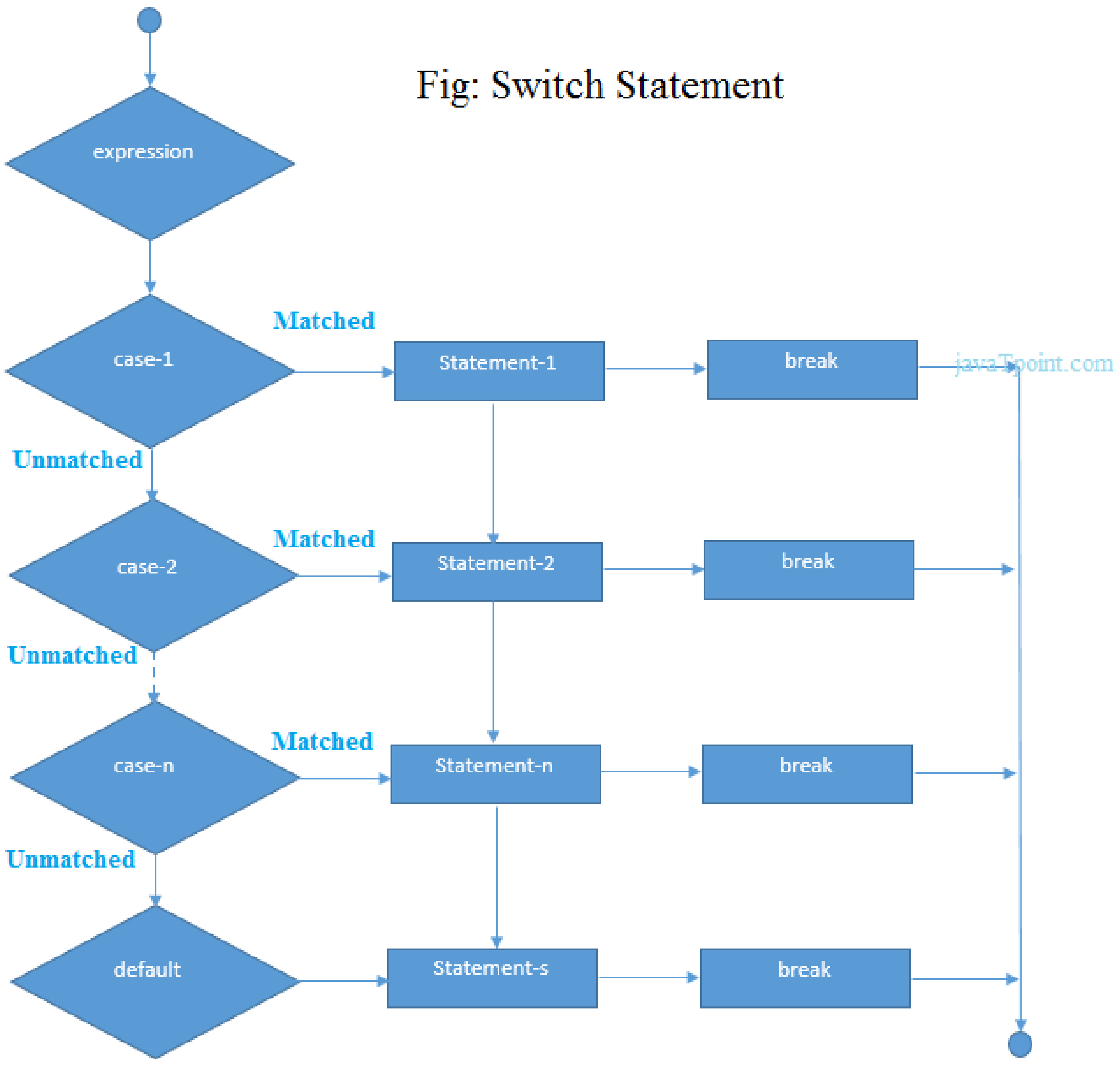
```
switch(expression) {

   case constant-expression  :
      statement(s);
      break; /* optional */

   case constant-expression  :
      statement(s);
      break; /* optional */

   /* you can have any number of case statements */
   default : /* Optional */
   statement(s);
}
```

Switch Statement

- The following rules apply to a **switch** statement −
    - The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
    - You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
    - The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
    - When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
    - When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
    - Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
    - A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

Switch Statement

- **Flow Diagram:**



Switch Statement

- **Example:**

```
int main()
{
  int day;
  printf("Enter the day no (1-7):");
  scanf(“%d”, &day);
  switch(day)
  {
    case 1:
      printf("Saturday");
    break;
    case 2:
      printf("Sunday");
    break;
    case 3:
      printf(“Monday");
    break;
    case 4:
      printf("Tuesday");
    break;
    case 5:
      printf("Wednesday");
    break;
    case 6:
      printf("Thursday");
    break;
    case 7:
      printf(“Friday");
    break;
    default:
     printf(“Invalid input!");
    break;
   }
return 0;
}
```

Output:

```
Enter the day no (1-7): 6
                 Thursday
```

Jump Statement

- These statements are used in C or C++ for the unconditional flow of control throughout the functions in a program.
- They support four types of jump statements:
  - **Break**
  - **Continue**
  - **Goto**
  - **Return**

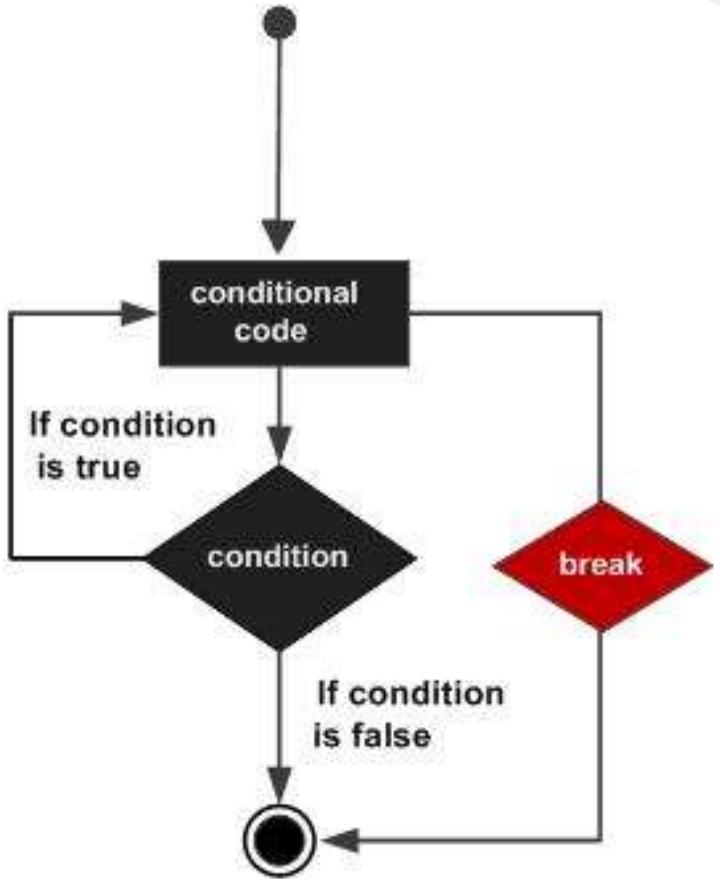
Break Statement

- The **break** statement in C programming has the following two usages –
  - This loop control statement is used to terminate the loop. When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
  - It can be used to terminate a case in the **switch** statement.
- If you are using nested loops, 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;
```

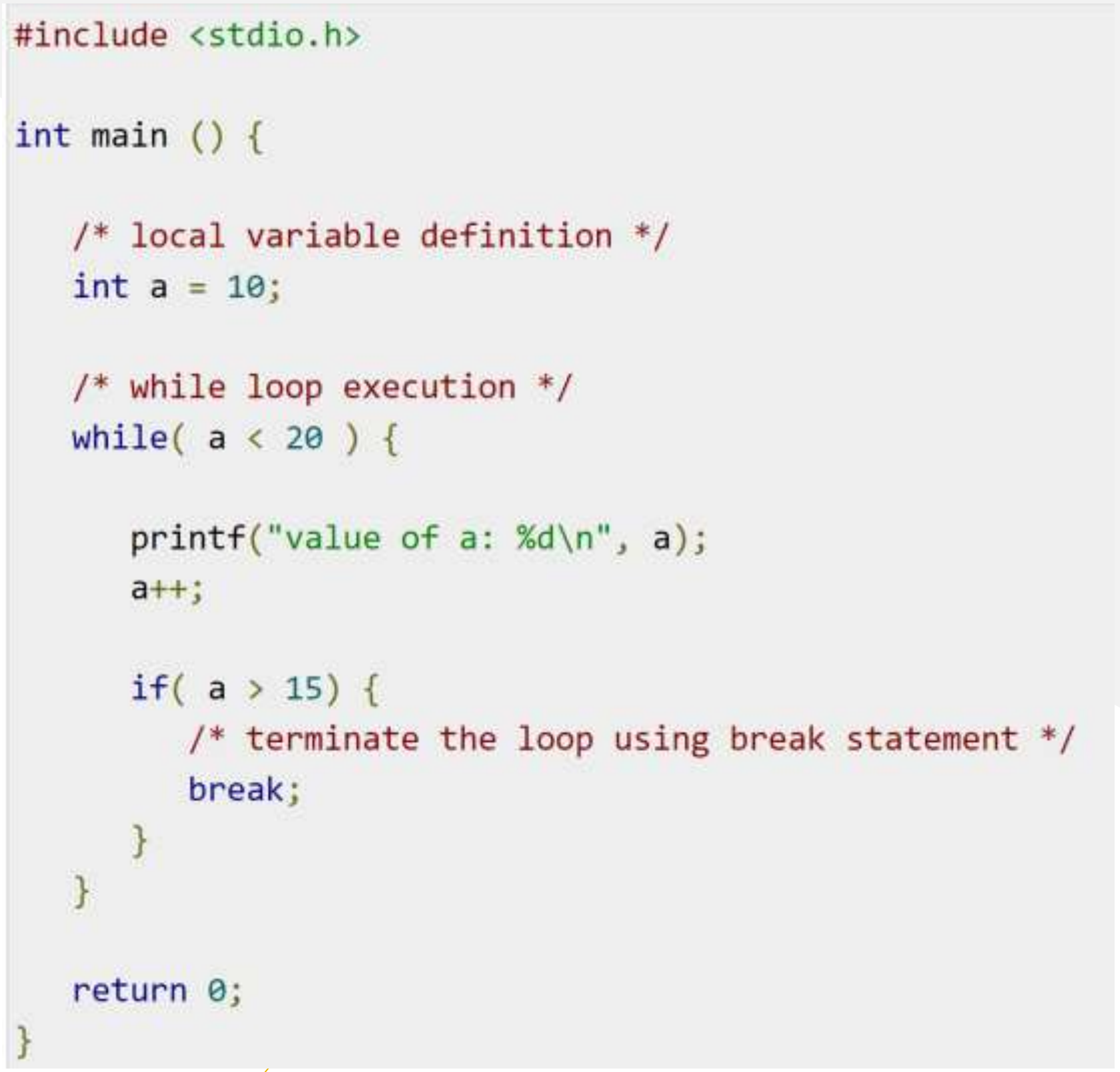
Break Statement

- **Flow Diagram:**



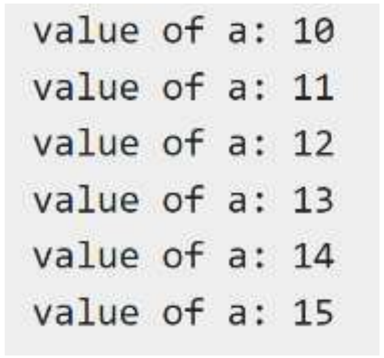
Break Statement

- **Example:**



```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {

      printf("value of a: %d\n", a);
      a++;

      if( a > 15) {
         /* terminate the loop using break statement */
         break;
      }
   }

   return 0;
}
```

- When the above code is compiled and executed, it produces the following result –



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

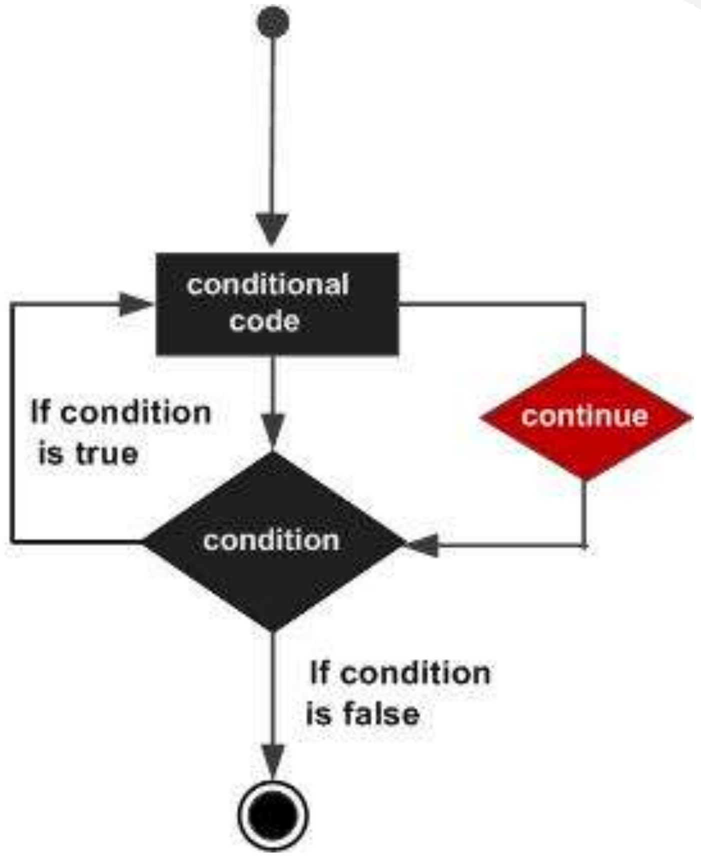
Continue Statement

- The **continue** statement in C programming works somewhat like the **break** statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.
    - For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute.
    - For the **while** and **do...while** loops, **continue** statement causes the program control to pass to the conditional tests.
- **Syntax:**
- The syntax for a **continue** statement in C is as follows –

```
continue;
```

Continue Statement

- **Flow Diagram:**



Continue Statement

- **Example:**

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         continue;
      }

      printf("value of a: %d\n", a);
      a++;

   } while( a < 20 );

   return 0;
}
```

- When the above code is compiled and executed, it produces the following result –

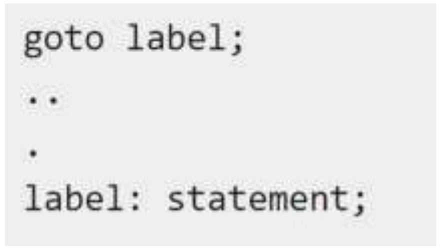
```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Goto Statement

- The **goto** statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statement can be used to repeat some part of the code for a particular condition.
- It can also be used to break the multiple loops which can't be done by using a single break statement.
- However, using goto is avoided these days since it makes the program less readable and complicated.

Syntax:

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

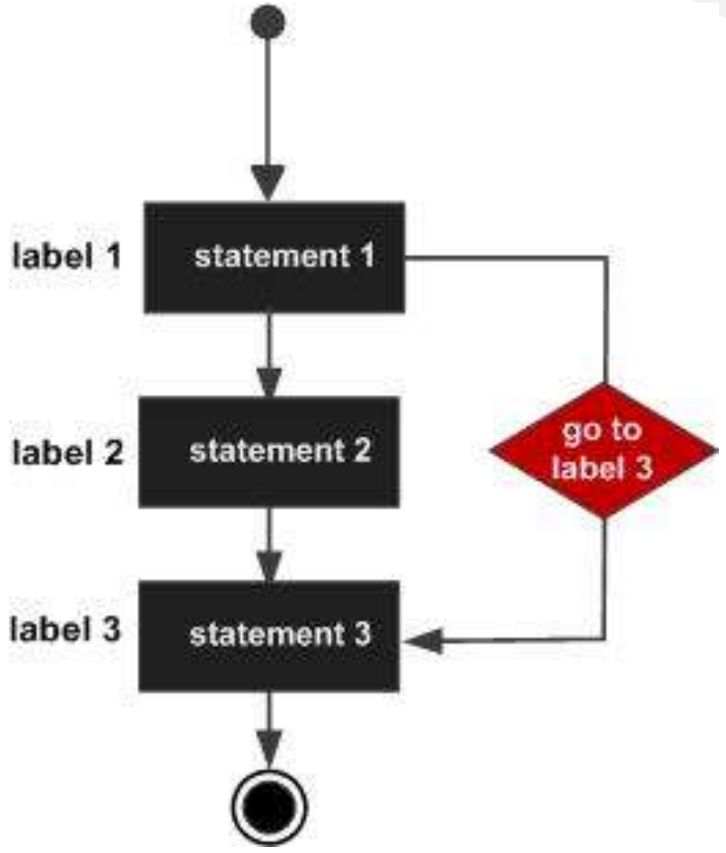


```
goto label;
..
.
label: statement;
```

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

Goto Statement

- **Flow Diagram:**



Goto Statement

- **Example:**

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP:do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }

      printf("value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

- When the above code is compiled and executed, it produces the following result –

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

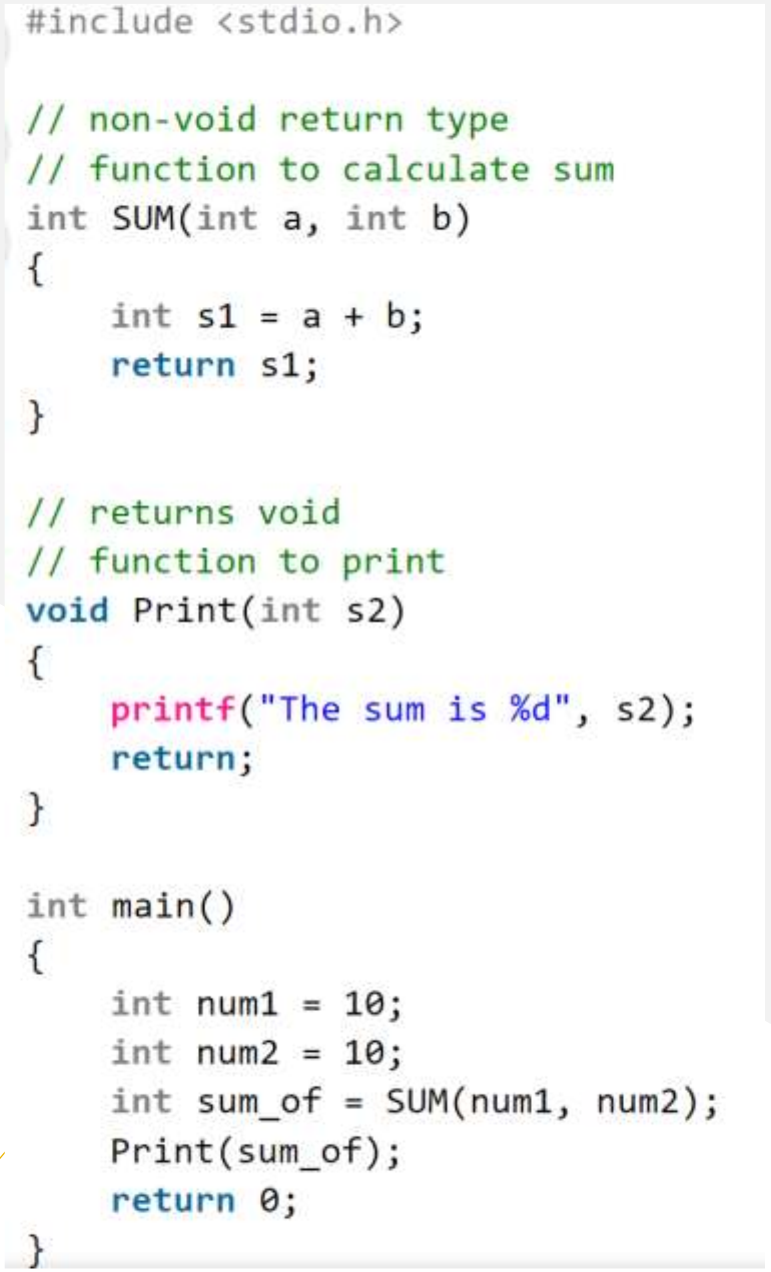
Return Statement

- The **return** in C or C++ returns the flow of the execution to the function from where it is called. This statement does not mandatorily need any conditional statements.
- As soon as the statement is executed, the flow of the program stops immediately and return the control from where it was called.
- The return statement may or may not return anything for a void function, but for a non-void function, a return value is must be returned.
- **Syntax:**
- The syntax for a **return** statement in C is as follows –

```
return [expression];
```

Return Statement

- **Example:**



```c
#include <stdio.h>

// non-void return type
// function to calculate sum
int SUM(int a, int b)
{
    int s1 = a + b;
    return s1;
}

// returns void
// function to print
void Print(int s2)
{
    printf("The sum is %d", s2);
    return;
}

int main()
{
    int num1 = 10;
    int num2 = 10;
    int sum_of = SUM(num1, num2);
    Print(sum_of);
    return 0;
}
```

- When the above code is compiled and executed, it produces the following result −

```
The sum is 20
```

?

THE END