CS101: Problem Solving through C Programming

Sachchida Nand Chaurasia

Assistant Professor

Department of Computer Science
Banaras Hindu University
Varanasi
Email id: snchaurasia@bhu.ac.in, sachchidanand.mca07@gmail.com

February 5, 2021

Conditional statements I

```
4 + 2 * 3 > 12 - 2
ANSWER: ??

! (3 > 4)
ANSWER: ??

!((-5 >= -6.2) || (7 != 3) && (6 == (3 + 3)))
 ANSWER: ??


4 % 2 == 0 <= 8 * 2
 ANSWER: ??

10 % 4 * 3 - 8 <= 18 + 30 / 4 - 20
ANSWER: ??

month is 2, year is 1999, (month == 2) && ((year % 4) == 0)
ANSWER: ??

!( 1 || 0 )
ANSWER: ??

!( 1 || 1 && 0 )
 ANSWER: ??

!( ( 1 || 0 ) && 0 )
```

Conditional statements II

```
27 ANSWER: ??
28
29 8 & 1 == 1000 & 0001 == 0000
30 ANSWER: ??
31
32 7 & 1 == 0111 & 0001 == 0001
33 ANSWER: ??
34
35
```

Control statements in C I

Control statements are used to alter the flow of the program. They are used to specify the order in which statements can be executed. They are commonly used to define how control is transferred from one part of the program to another. C language has following control statements:

- if. . . else
- switch
- Loops
  - for
  - while
  - do... while

Compound statement I

A Compound statement is a block of statement grouped together using braces (). In a compound statement, all statements are executed sequentially. A compound statement is also known as a block. It takes the following form:

```
{
        statement1;
        statement2;
        statement3;
        ...
        statementn;
}
Output: ???

```

Compound statement II

```
#include<stdio.h>
int main()
{
	int i = 100;
	printf("A single statement\n");
	{
		// a compound statement
		printf("A statement inside compound statement\n");
		printf("Another statement inside compound statement\n");
	}
	// signal to operating system everything works fine
	return 0;
}
Expected Output:
A single statement
A statement inside compound statement
Another statement inside compound statement
Output: ???

```

if statement I

If statement is used to test a condition and take one of the two possible actions.

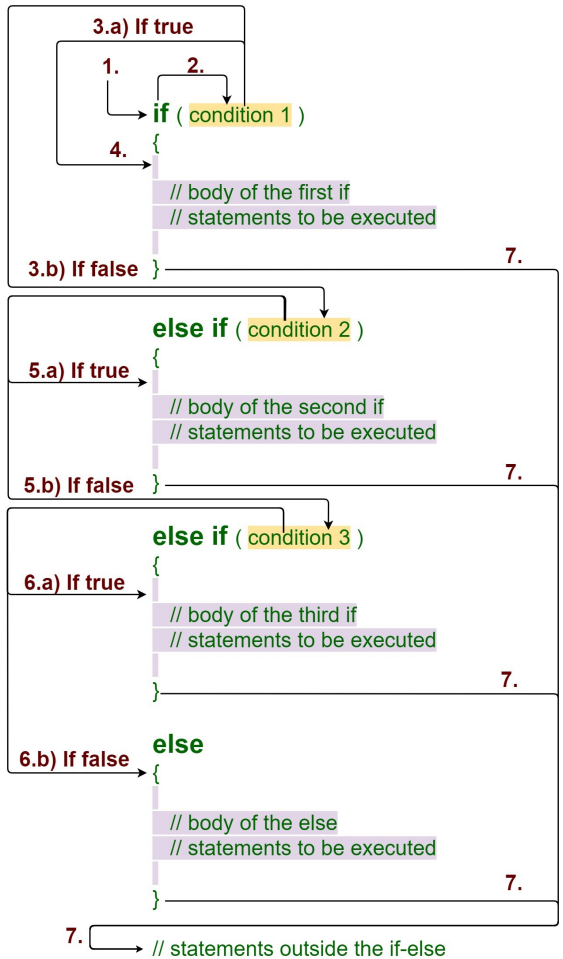
The syntax of the if statement is:

```
if (condition)
{
       // if block
       statement1;
       statement2;
}

```

if statement II

If - else if ladder



if statement III

The *condition* can be any constant, variable, expression, relational expression, logical expression and so on. Just remember that in C, any non-zero value is considered as true while 0 is considered as false.

```
#include<stdio.h>
int main()
{
	int n;
	printf("Enter a number: ");
	scanf("%d", &n);
	if(n % 2 == 0)
	{
		printf("%d is even", n);
	}
	// signal to operating system everything works fine
	return 0;
}
Expected Output:
Enter a number: 46
46 is even
Enter a number: 21
Output: ???

```

if statement IV

Which statement belongs to if?:

```
1 if (condition)
2 statement1;
3 statement2;
4 statement3;
5 Output: ???
6
```

```
1  #include<stdio.h>
2
3  int main()
4  {
5        if(0)
6        printf("statement 1\n");
7        printf("statement 2\n");
8        printf("statement 3\n");
9
10       // signal to operating system everything works fine
11       return 0;
12 }
13 Output: ???
14
```

if statement V

The else clause: The else clause allows us to add an alternative path to the if condition. Statements under the else block are executed only when the if condition is false.

```
if (condition)
{
	// if block
	statement1;
	statement2;
}

else
{
	// else block
	statement3;
	statement4;
}
Output: ???

```

if statement VI

```c
#include<stdio.h>

int main()
{
	int n;
	printf("Enter a number: ");
	scanf("%d", &n);
	if(n % 2 == 0)
	{
		printf("%d is even", n);
	}
	else
	{
		printf("%d is odd", n);
	}
	// signal to operating system everything program ran fine
	return 0;
}
Output: ???

```

if statement VII

Nesting if. . . else: We can add if.. else statement inside if block or else block. This is called nesting of if.. else.

```
if(condition1)
{
	if(condition2)
	{
		statement1;
		statement2;
	}
	else
	{
		statement3;
		statement4;
	}
}
else
{
	if(condition3)
	{
		statement5;
		statement6;
	}
	else
	{
```

if statement VIII

```
23              statement7;
24              statement8;
25      }
26  }
27  Output: ???
28
```

✓ We can nest if.. else statement to any depth.

```
1   #include<stdio.h>
2   int main()
3   {
4       int a, b, c, larger;
5       printf("Enter three numbers: ");
6       scanf("%d %d %d", &a, &b, &c);
7       if(a > b)
8       {
9           if(a > c)
10          {
11              larger = a;
12          }
13          else
14          {
15              larger = c;
16          }
```

if statement IX

```
17        }
18        else
19        {
20              if(b > c)
21              {
22                    larger = b;
23              }
24              else
25              {
26                    larger = c;
27              }
28        }
29        printf("Largest number is %d", larger);
30        // signal to operating system everything works fine
31        return 0;
32  }
33  Output: ???
34
```

if statement X

Matching if.. else parts: Sometimes it becomes confusing to associate an else clause with the if statement. Consider the following example:

```
1 if(a<10)
2   if (a % 2 ==0)
3   printf("a is even and less than 10\n");
4 else
5  printf("a is greater than 10");
6 Output: ???
7
```

✓ We can always avoid such complications using braces ().

if statement XI

```
1  if(a < 10)
2  {
3         if (a % 2 ==0)
4         {
5                printf("a is even and less than 10\n");
6         }
7
8         else
9         {
10               printf("a is greater than 10");
11        }
12 }
13 Output: ???
14
```

if statement XII

else if clause: if-else is a bi-directional statement that is used to test a condition and take one of the possible two actions.

```
1  if(condition1)
2  {
3        statement1;
4  }
5  else if(condition2)
6  {
7        statement2;
8  }
9  else if(condition3)
10 {
11       statement3;
12 }
13 ...
14 else
15 {
16       statement4;
17 }
18 Output: ???
19
```

if statement XIII

✓ Using else-if clause we can write nested if-else statement in a more compact form.

```c
#include<stdio.h>
int main()
{
	int a, b, c, larger;
	printf("Enter three numbers: ");
	scanf("%d %d %d", &a, &b, &c);
	if(a > b && a > c)
	{
		larger = a;
	}
	else if(b > a && b > c)
	{
		larger = b;
	}
	else
	{
		larger = c;
	}
	printf("Largest number is %d", larger);
	// signal to operating system everything works fine
	return 0;
}
```

if statement XIV

```
23 Output: ???
24
```

if statement XV

Ask question:

```
#include<stdio.h>

int main()
{
        int a=12;
        if(0 && ++a)
        printf("statement 1\n");
        printf("statement 2\n");
        printf("statement 3\n");

        printf("a=%d\n",a);

        // signal to operating system everything works fine
        return 0;
}
Output: ???

```

if statement XVI

```c
// C program to demonstrate working of logical operators
#include <stdio.h>

int main()
{
	int a = 10, b = 4, c = 10, d = 20;

	// logical operators

	// logical AND example
	if (a > b && c == d)
	printf("a is greater than b AND c is equal to d\n");
	else
	printf("AND condition not satisfied\n");

	// logical AND example
	if (a > b || c == d)
	printf("a is greater than b OR c is equal to d\n");
	else
	printf("Neither a is greater than b nor c is equal "
	" to d\n");

	// logical NOT example
	if (!a)
	printf("a is zero\n");
```

if statement XVII

```
26        else
27        printf("a is not zero");
28
29        return 0;
30  }
31  Output: ???
32
```

```
1   #include <stdbool.h>
2   #include <stdio.h>
3   int main()
4   {
5         int a = 10, b = 4;
6         bool res = ((a == b) && printf("GeeksQuiz"));
7         return 0;
8   }
9   Output: ???
10
```

if statement XVIII

```
#include <stdbool.h>
#include <stdio.h>
int main()
{
       int a = 10, b = 4;
       bool res = ((a != b) && printf("GeeksQuiz"));
       return 0;
}
Output: ???

```

```
#include <stdbool.h>
#include <stdio.h>
int main()
{
       int a = 10, b = 4;
       bool res = ((a != b) || printf("GeeksQuiz"));
       return 0;
}
Output: ???

```

if statement XIX

```c
#include <stdbool.h>
#include <stdio.h>
int main()
{
       int a = 10, b = 4;
       bool res = ((a == b) || printf("GeeksQuiz"));
       return 0;
}
Output: ???

```

```c
 /* C Relational Operations on integers */
#include <stdio.h>
int main()
{
         int a = 9;
         int b = 4;
         printf(" a >  b: %d \n", a > b);
         printf("a >= b: %d \n", a >= b);
         printf("a <= b: %d \n", a <= b);
         printf("a <  b: %d \n", a < b);
         printf("a == b: %d \n", a == b);
         printf("a != b: %d \n", a != b);
}
```

if statement XX

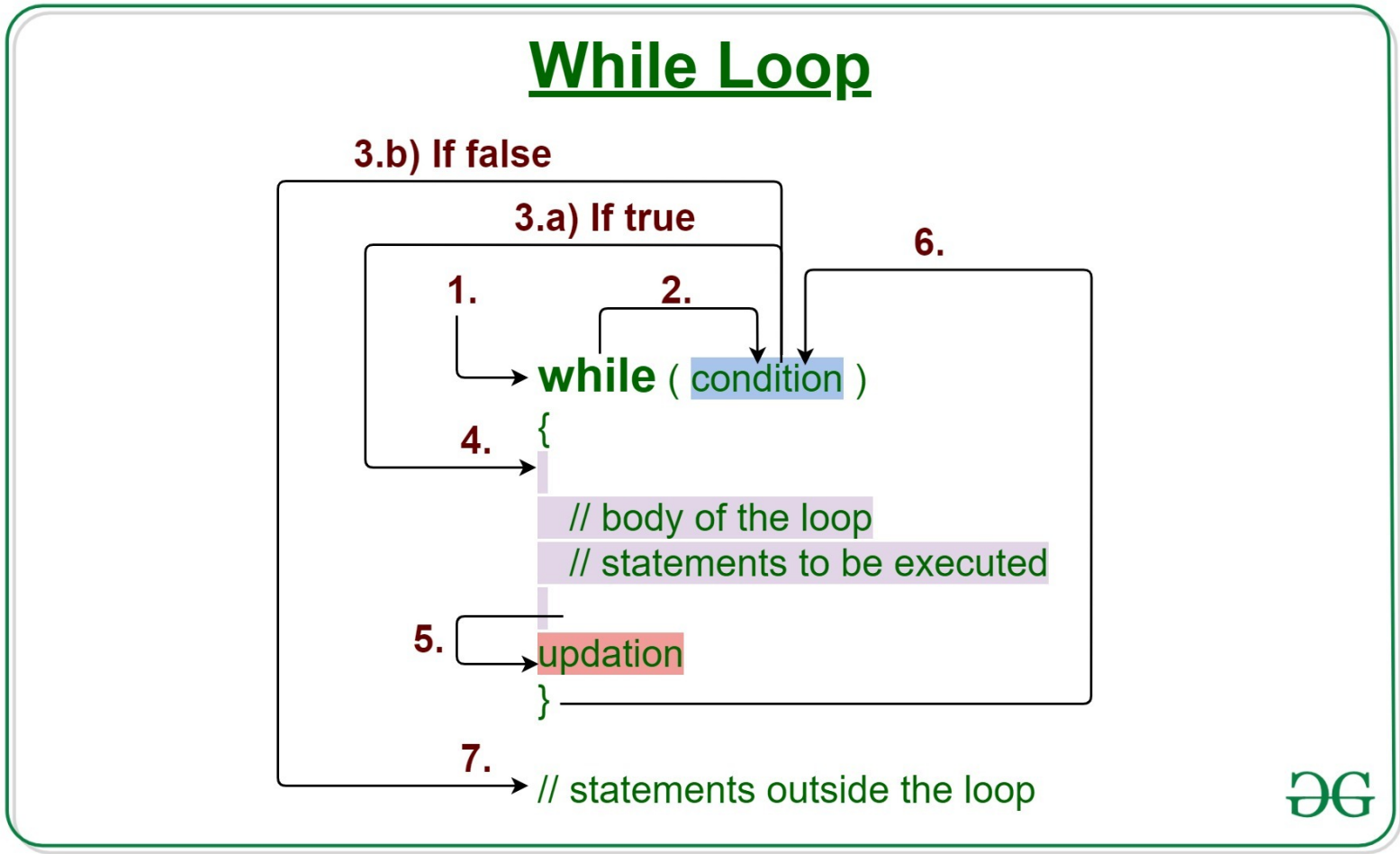
```c
/* Using C Relational Operators in If Condition */
#include <stdio.h>
void main()
{
        int x = 10;
        int y = 25;
        if (x = y)
      {
            printf(" x is equal to y \n" );
      }
        else
      {
            printf(" x is not equal to y \n" );
      }
}
```

The while loop I

Loops are used to execute statements or block of statements repeatedly. For example, suppose we want to write a program to print "Hello" 10 times. One way to achieve this is to write the following statement 10 times.

```
while(condition)
{
	// body of while loop
	statement 1;
	statement 2;
}
```

The while loop II



The while loop III

✓ Just like the if-else statement, the while loop starts with a condition.

✓ First, the condition is evaluated, if it is true then the statements in the body of the while are executed.

✓ After executing the body of the while loop, the condition is checked again, if it is still true then once again statements in the body of the while are executed.

✓ This process keeps repeating until the condition becomes false.

Properties of while loop:

- A conditional expression is used to check the condition.
- The statements defined inside the while loop will repeatedly execute until the given condition fails.

The while loop IV

- The condition will be true if it returns 0. The condition will be false if it returns any non-zero number.
- In while loop, the condition expression is compulsory.
- Running a while loop without a body is possible.
- We can have more than one conditional expression in while loop.
- If the loop body contains only one statement, then the braces are optional.

The while loop V

```c
// Print numbers from 1 to 5

#include <stdio.h>
int main()
{
	int i = 1;

	while (i <= 5)
	{
		printf("%d\n", i);
		++i;
	}
	return 0;
}
```

The while loop VI

```c
#include<stdio.h>
int main()
{
	int i = 1;
	// keep looping while i < 100
	while(i < 100)
	{
		// if i is even
		if(i % 2 == 0)
		{
			printf("%d ", i);
		}
		i++; // increment i by 1
	}
	return 0;
}
```

The while loop VII

✓ The following program calculates the sum of digits of a number entered by the user.

```
1  int main()
2  {
3        int n, num, sum = 0, remainder;
4        printf("Enter a number: ");
5        scanf("%d", &n);
6        num = n;
7        // keep looping while n > 0
8        while( n > 0 )
9        {
10             remainder = n % 10; // get the last digit of n
11             sum += remainder; // add the remainder to the sum
12             n /= 10;  // remove the last digit from n
13       }
14       printf("Sum of digits of %d is %d", num, sum);
15       // signal to operating system everything works fine
16       return 0;
17 }
```

The while loop VIII

```c
#include<stdio.h>
void main ()
{
	int j = 1;
	while(j+=2,j<=10)
	{
		printf("%d ",j);
	}
	printf("%d",j);
}
```

```c
#include<stdio.h>
void main ()
{
	while()
	{
		printf("hello Javatpoint");
	}
}
```

The while loop IX

✓ Infinitive while loop in C.

```c
while(1)
{
	//statement
}
```

```c
#include<stdio.h>
void main ()
{
	int x = 10, y = 2;
	while(x+y-1)
	{
		printf("%d %d",x--,y--);
	}
}
```

The while loop X

```c
#include <stdio.h>
int main()
{
	int i=1, j=1;
	while (i <= 4 || j <= 3)
	{
		printf("%d %d\n",i, j);
		i++;
		j++;
	}
	return 0;
}
```

The while loop XI

```c
#include <stdio.h>
int main()
{
	int i=1;
	short j=32766;
	while (i <= 4 || j >= 3)
	{
		printf("%d %d\n",i, j);
		i++;
		j++;
	}
	return 0;
}
Output: ???
```

```c
int i = 1;
while(i<10)
{
	printf("%d\n", i);
}
```

The while loop XII

```
int i = 1;
while(i=10)
{
	printf("%d", i);
}
```

```
float f = 2;
while(f != 31.0)
{
	printf("%f\n", f);
	f += 0.1;
}
```

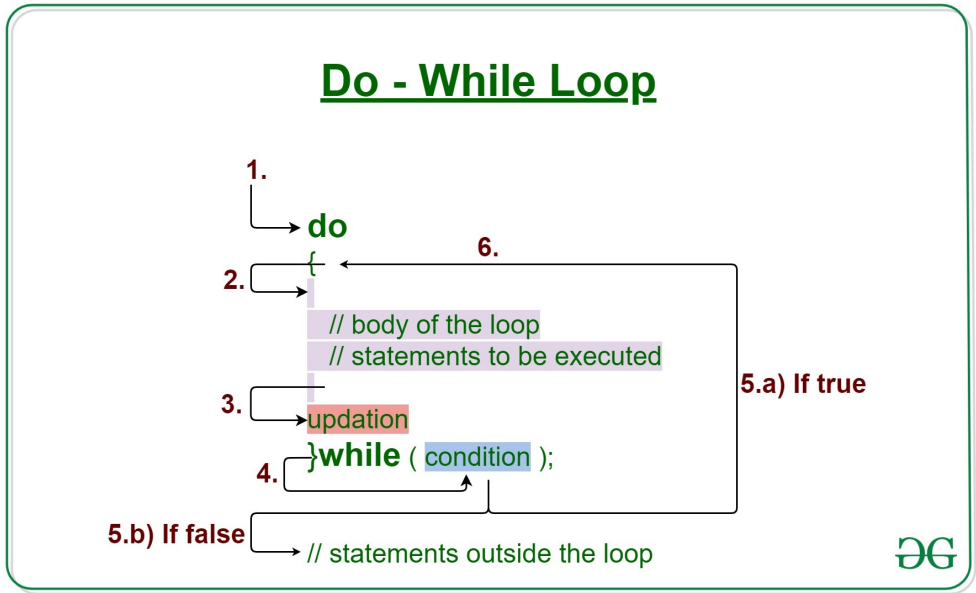
✓ This loop is infinite because computers represent floating point numbers as approximate numbers, so 3.0 may be stored as 2.999999 or 3.00001. So the condition (f != 31.0) never becomes false. To fix this problem write the condition as f <= 31.0.

The while loop XIII

```
1 int i = 0;
2
3 while(i<=5);
4 printf("%d\n", i);
```

```
1 int i = 0;
2 while(i<=5)
3 {
4       ; // a null statement
5 }
6
7 printf("%d\n", i);
```

do..while loop I




```
do
{
      // body of do while loop
      statement 1;
      statement 2;
}
while(condition);
```

do..while loop II

✓ In do while loop first the statements in the body are executed then the condition is checked.

✓ If the condition is true then once again statements in the body are executed.

✓ This process keeps repeating until the condition becomes false.

✓ As usual, if the body of do while loop contains only one statement, then braces () can be omitted.

Notice that unlike the while loop, in do while a semicolon(;) is placed after the condition.

✓ The do while loop differs significantly from the while loop because in do while loop statements in the body are executed at least once even if the condition is false.

do..while loop III

✓ In the case of while loop the condition is checked first and if it true only then the statements in the body of the loop are executed.

```c
//The following program print numbers between 1 and 100 which are multiple of 3 using the do while loop:
#include<stdio.h> // include the stdio.h
int main()
{
	int i = 1; // declare and initialize i to 1
	do
	{
		// check whether i is multiple of 3 not or not
		if(i % 3 == 0)
		{
			printf("%d ", i); // print the value of i
		}
		i++; // increment i by 1
	}
	while(i < 100);  // stop the loop when i becomes greater than 100
	return 0;
}
```

do..while loop IV

Where should I use do while loop?

✓ Let's say you want to create a program to find the factorial of a number. As you probably know that factorial is only valid for 0 and positive numbers.

✓ do..while is one way you can approach this problem.

✓ Let's say the user entered a negative number, so instead of displaying an error message and quitting the program, a better approach would be to ask the user again to enter a number.

✓ You have to keep asking until the user enters a positive number or 0.

Once a positive number or 0 is entered, calculate factorial and display the result.

Let's see how we can implement it using while and do while loop.

do..while loop V

```c
//Using while loop
#include<stdio.h> // include the stdio.h
int main()
{
	int num;
	char num_ok = 0;
	// keep asking for numbers until num_ok == 0
	while(num_ok==0)
	{
		printf("Enter a number: ");
		scanf("%d", &num);
		// if num >= 0 set num_ok = 1 and stop asking for input
		if(num>=0)
		{
			num_ok = 1;
		}
	}
}
```

do..while loop VI

```c
//Using do while loop
#include<stdio.h> // include the stdio.h

int main()
{
	int num;

	do
	{
		printf("Enter a number: ");
		scanf("%d", &num);
	}
	while(num<0); // keep asking for numbers until num < 0

}
```

do..while loop VII

```c
// Program to add numbers until the user enters zero

#include <stdio.h>
int main()
{
	double number, sum = 0;

	// the body of the loop is executed at least once
	do
	{
		printf("Enter a number: ");
		scanf("%lf", &number);
		sum += number;
	}
	while(number != 0.0);

	printf("Sum = %.2lf",sum);

	return 0;
}
```

✓ Infinitive do while loop: The do-while loop will run infinite times if we pass any non-zero value as the conditional expression.

do..while loop VIII

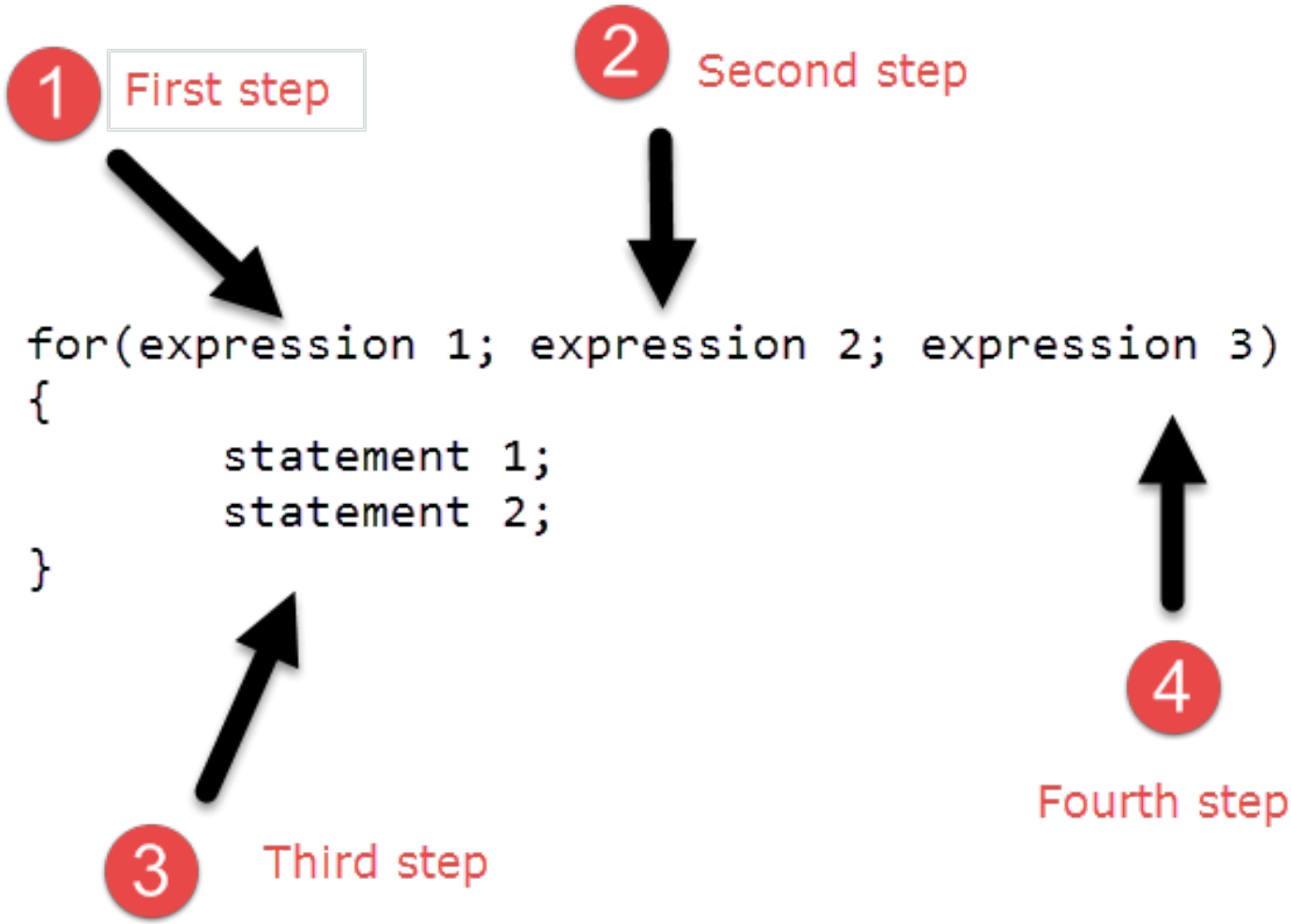
```
1 do
2 {
3       //statement
4 }
5 while(1);
```

The for loop in C I

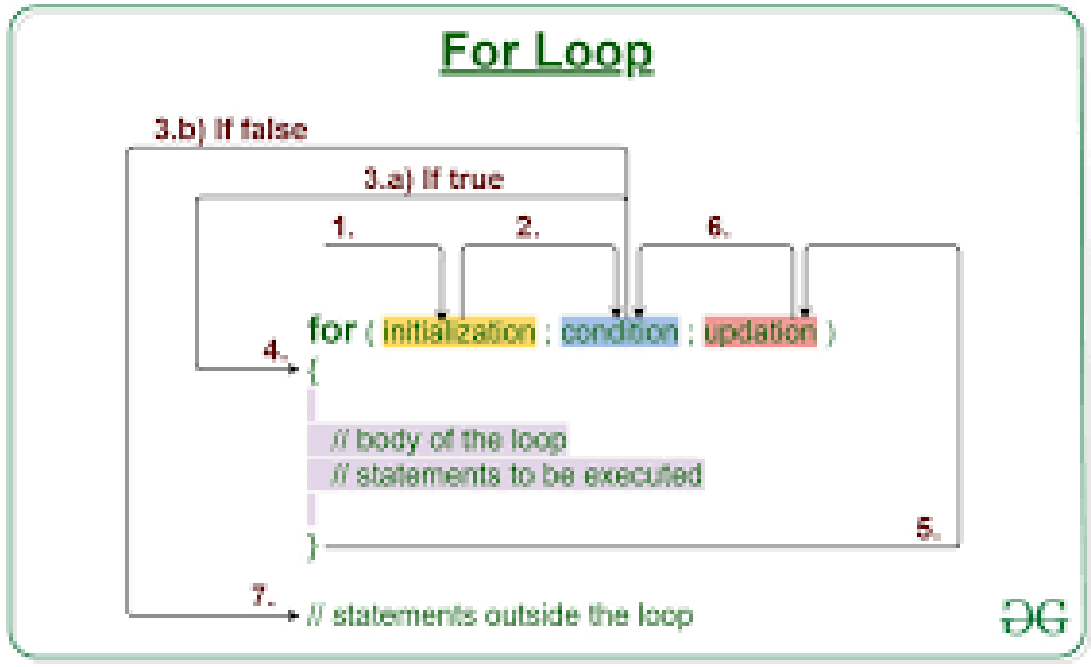
```
for(expression1; expression2; expression3)
{
       // body of for loop
       statement1;
       statement2;
}
```

- ✓ The $expression1$ is the initialization expression.
- ✓ The $expression2$ is the test expression or condition.
- ✓ The $expression3$ is the update expression.

The for loop in C II



The for loop in C III




```
1 for(expression1; expression2; expression3)
2 statement1;
3
4 // The above for loop is equivalent to:
5
6 for(expression1; expression2; expression3)
7 {
8       statement1;
9 }
```

The for loop in C IV

```c
#include<stdio.h>

int main()
{
	int i; // loop variable
	int sum = 0; // variable to accumulate sum

	for(i = 1; i <= 100; i++)
	{
		sum += i;
	}

	printf("Sum = %d", sum);

	// return 0 to operating system
	return 0;
}
```

Expressions in the for loop are Optional

✓ All the three expressions inside the for loop are optional, but the two semicolons must always be present.

The for loop in C V

- We can omit the $expression1$ if the initialization is done outside the for loop.
- If the $expression2$ is omitted then the condition is always true, leading to the creation of an infinite loop - a loop which never stops executing. To avoid infinite loops you should include a break or return statement within in the loop body. We discuss the break and return statements in detail in upcoming chapters.
- We can omit the $expression3$ if the update expression is present inside the body of the for loop.

Properties of $expression1$

- The expression represents the initialization of the loop variable.
- We can initialize more than one variable in $expression1$.

The for loop in C VI

- $expression1$ is optional.
- In C, we can not declare the variables in $expression1$. However, It can be an exception in some compilers.

Properties of $expression2$

- $expression2$ is a conditional expression. It checks for a specific condition to be satisfied. If it is not, the loop is terminated.
- $expression2$ can have more than one condition. However, the loop will iterate until the last condition becomes false. Other conditions will be treated as statements.
- $expression2$ is optional.

The for loop in C VII

- $expression2$ can perform the task of expression 1 and expression 3. That is, we can initialize the variable as well as update the loop variable in $expression2$ itself.
- We can pass zero or non-zero value in $expression2$. However, in C, any non-zero value is true, and zero is false by default.

Properties of $expression3$

- $expression3$ is used to update the loop variable.
- We can update more than one variable at the same time.
- $expression3$ is optional.

The for loop in C VIII

```c
/* 1st variation - expression1 is omitted */

#include<stdio.h>

int main()
{
	int i = 1, sum = 0;

	//expression 1 is omitted

	for( ; i <= 100; i++)
	{
		sum += i;
	}

	printf("Sum = %d", sum);

	// return 0 to operating system
	return 0;
}
```

The for loop in C IX

```c
/*
  2nd variaton - expression2 is omitted
*/

#include<stdio.h>

int main()
{
	int i, sum = 0;

	for(i = 1 ; ; i++)  // $expression2$ is omitted
	{
		if(i > 100)
		{
			/* the break statement causes the loop to terminate.
			We will discuss the break statement in detail
			in later chapters.
			 */
			break;
		}
		sum += i;
	}
	printf("Sum = %d", sum);

	// return 0 to operating system
```

The for loop in C X

```
26      return 0;
27  }
```

```
1   /* 3rd variation - expression3 is omitted */
2   #include<stdio.h>
3   int main()
4   {
5       int i, sum = 0;
6       // expression3 is omitted
7       for(i = 1 ; i <= 100 ; )
8       {
9           sum += i;
10          i++; // update expression
11      }
12      printf("Sum = %d", sum);
13      // return 0 to operating system
14      return 0;
15  }
```

The for loop in C XI

```c
/* 4th variation - all the expressions are omitted */
#include<stdio.h>
int main()
{
	int i = 0; // initialization expression
	int sum = 0;
	for( ; ; )
	{
		if(i > 100) // condition
		{
			break; // break out of the for loop
		}
		sum += i;
		i++; // update expression
	}
	printf("Sum = %d", sum);
	// return 0 to operating system
	return 0;
}
```

The for loop in C XII

```c
#include <stdio.h>
int main()
{
	int i,j,k;
	for(i=0,j=0,k=0; i<4,k<8,j<10; i++)
	{
		printf("%d %d %d\n",i,j,k);
		j+=2;
		k+=3;
	}
}
```

The for loop in C XIII

Nesting of Loops:

✓ Just as if-else statement can be nested inside another if-else statement, we can nest any type of loop inside any other type.

✓ For example, a for loop can be nested inside another for loop or inside while or do while loop. Similarly, while and do while can also be nested.

```c
#include<stdio.h>
int main()
{
	int row = 0, col = 0;
	for(row = 0; row < 10; row++)  // number of lines
	{
		for(col = 0; col < row; col++)  // num of * in each lines
		{
			printf(" * ");
		}
		printf("\n");
	}
	// return 0 to operating system
	return 0;
}
```

The for loop in C XIV

```c
#include<stdio.h>
void main ()
{
	int i;
	for(i=0; i<10; i++)
	{
		int i = 20;
		printf("%d ",i);
	}
}
Output:???
```

```c
// ++count and count++ ????
#include <stdio.h>
int main()
{
	int num, count, sum = 0;

	printf("Enter a positive integer: ");
	scanf("%d", &num);

	// for loop terminates when num is less than count
	for(count = 1; count <= num; ++count)
	{
```

The for loop in C XV

```
13                sum += count;
14        }
15
16        printf("Sum = %d", sum);
17
18        return 0;
19  }
20  Output:???
```

The for loop in C XVI

Infinitive for loop in C

✓ To make a for loop infinite, we need not give any expression in the syntax.

✓ Instead of that, we need to provide two semicolons to validate the syntax of the for loop. This will work as an infinite for loop.

```
#include<stdio.h>
void main ()
{
	for(; ; )
	{
		printf("welcome to javatpoint");
	}
}
```

```
short int i;
for (i = 1; i < 32768; i++)
{
	printf("%d\n", i);
}
Output: ???
```

The for loop in C XVII

```
/* FIRST */
for(i=0; i<10; i++)
  for(j=0; j<100; j++)
//do something
filter_none
brightness_4
/* SECOND */
for(i=0; i<100; i++)
  for(j=0; j<10; j++)
//do something
```

```
int main()
{
        int c1 = 0, c2 = 0;

        /* FIRST */
        for(int i=0; i<10; i++,c1++)
      for(int j=0; j<100; j++, c1++);
        //do something


        /* SECOND */
        for(int i=0; i<100; i++, c2++)
        for(int j=0; j<10; j++, c2++);
```

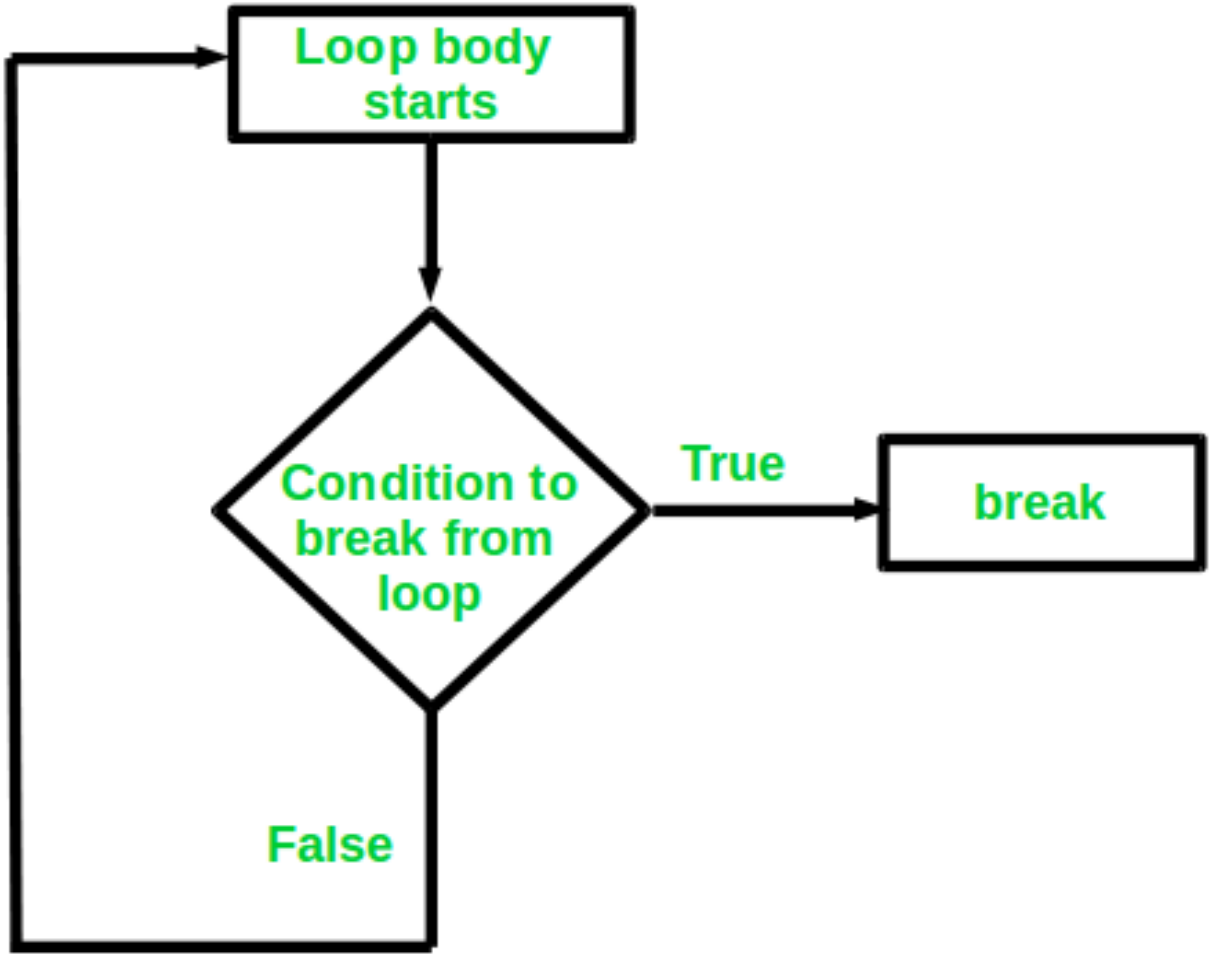
The for loop in C XVIII

```
14        //do something
15
16          printf("Count in FIRST = %d",c1);
17          printf("Count in SECOND  = %d",c2);
18          return 0;
19  }
```

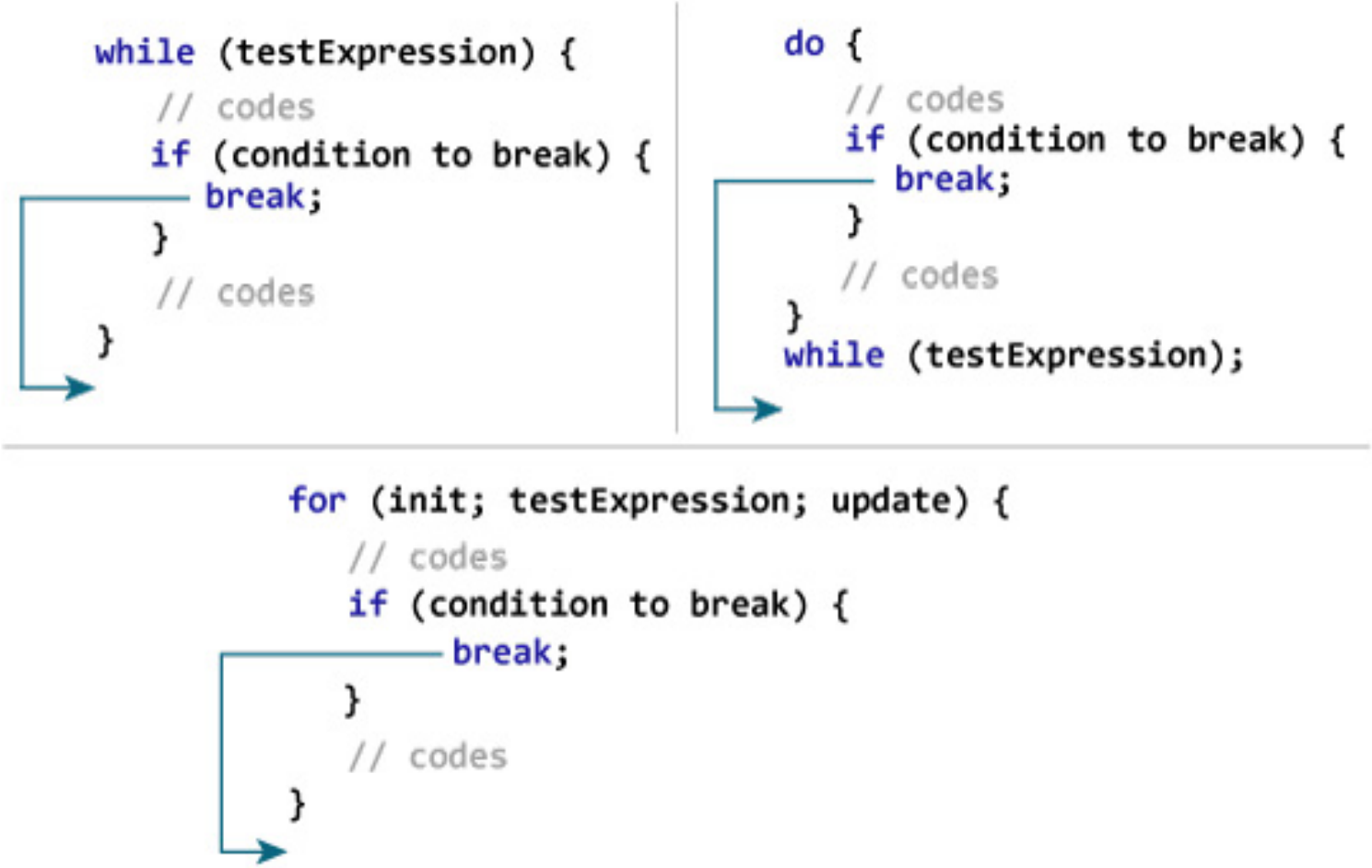
```
1   int main()
2   {
3         int c1 = 0, c2 = 0;
4
5         /* FIRST */
6         for(int i=0; ++c1&&i<10; i++)
7           for(int j=0; ++c1&&j<100; j++);
8          //do something
9
10        /* SECOND */
11        for(int i=0; ++c2&&i<100; i++)
12          for(int j=0; ++c2&&j<10; j++);
13          //do something
14
15        printf("Count in FIRST = %d",c1);
16          printf("Count in SECOND  = %d",c2);
17        return 0;
18  }
```

The for loop in C XIX

The break and continue statement in C I



The break and continue statement in C II



We will see here the usage of break statement with three different types of loops:

The break and continue statement in C III

- Simple loops
- Nested loops
- Infinite loops

1. Simple loops: Consider the situation where we want to search an element in an array. To do this, use a loop to traverse the array starting from the first index and compare the array elements with the given key.

The break and continue statement in C IV

```c
#include<stdio.h>
int main()
{
	int i;

	for(i = 1; i < 10 ; i++)
	{
		if(i==5)
		{
			break; // breaks out of the for loop
		}
		printf("Value of i = %d\n", i);
	}

	// signal to operating system everything works fine
	return 0;
}
```

2. Nested Loops: We can also use break statement while working with nested loops. If the break statement is used in the innermost loop. The control will come out only from the innermost loop.

The break and continue statement in C V

```c
// C program to illustrate  using break statement  in Nested loops
#include <stdio.h>
int main()
{
	// nested for loops with break statement
	// at inner loop
	for (int i = 0; i < 5; i++)
	{
		for (int j = 1; j <= 10; j++)
		{
			if (j > 3)
			break;
			else
			printf("*");
		}
		printf("\n");
	}
	return 0;
}
```

The break and continue statement in C VI

```c
// Program to calculate the sum of numbers (10 numbers max) If the user enters a negative number, the
      loop terminates
#include <stdio.h>
int main()
{
      int i;
      double number, sum = 0.0;
      for (i = 1; i <= 10; ++i)
      {
            printf("Enter a n%d: ", i);
            scanf("%lf", &number);
            // if the user enters a negative number, break the loop
            if (number < 0.0)
            {
                  break;
            }
            sum += number; // sum = sum + number;
      }
      printf("Sum = %.2lf", sum);
      return 0;
}
```

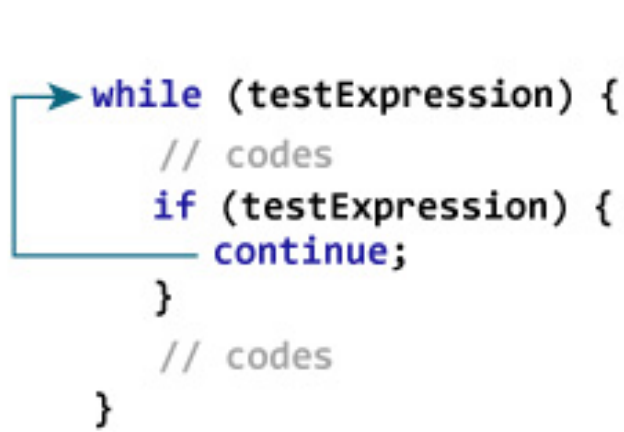
The break and continue statement in C VII

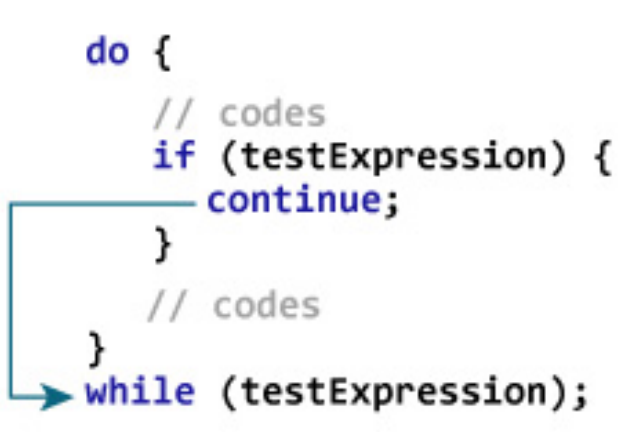
3. Infinite Loops: break statement can be included in an infinite loop with a condition in order to terminate the execution of the infinite loop.

```c
// C program to illustrate using break statement  in Infinite loops
#include <stdio.h>
int main()
{
	// loop initialization expression
	int i = 1;
	// infinite while loop
	while (1)
	{
		if (i > 10)
		break;
		printf("%d ", i);
		i++;
	}
	return 0;
}
```

The break and continue statement in C VIII

Continue statement in C





The break and continue statement in C IX

✓ The continue statement is used to prematurely end the current iteration and move on the to the next iteration.

✓ When the continue statement is encountered in a loop, all the statements after the continue statement are omitted and the loop continues with the next iteration.

✓ The continue statement is used in conjunction with a condition.

The break and continue statement in C X

✓ Sometimes people get confused with between the break and and continue statement.

✓ Always remember that the break statement when encountered breaks out of the loop, but when the continue statement is encountered, the loop is not terminated instead the control is passed to the beginning of the loop.

✓ The continue statement is almost always used with the $if...else$ statement.

The break and continue statement in C XI

```c
#include<stdio.h>
int main()
{
	int i;

	for(i = 0; i < 10; i++)
	{
		if( i % 4 == 0 )
		{
			/*
			when i is divisible by 4
			continue to the next iteration
			*/
			continue;
		}
		printf("%d\n", i);
	}
	// signal to operating system everything works fine
	return 0;
}
```

The switch statement in C I

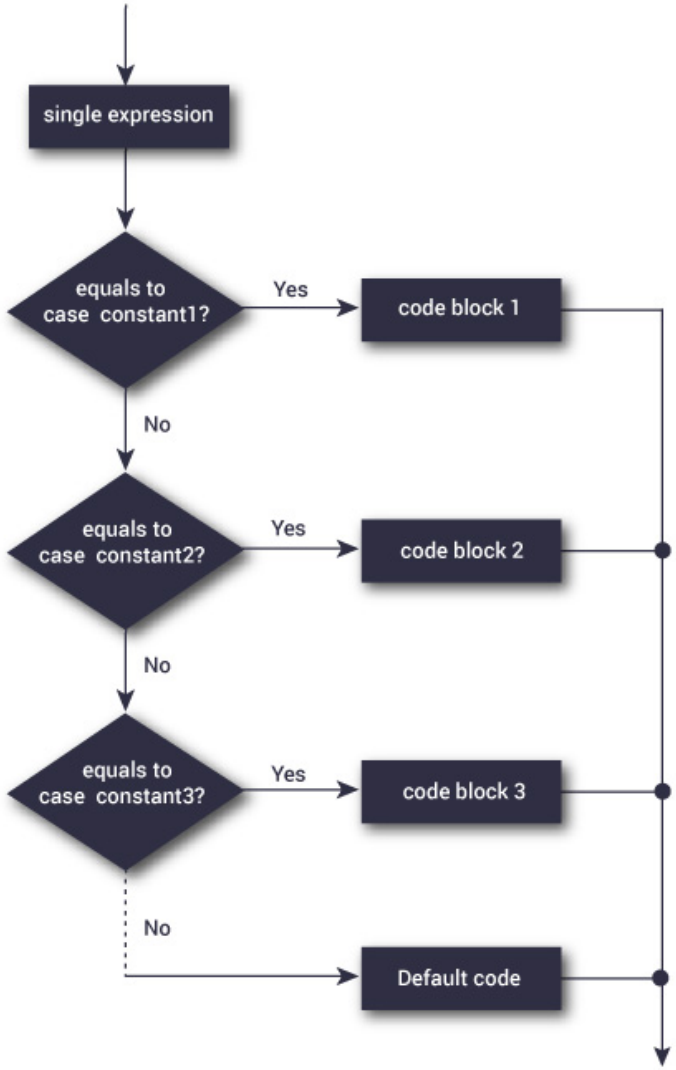
✓ The switch statement allows us to execute one code block among many alternatives.

✓ The switch statement is a multi-directional statement used to handle decisions.

You can do the same thing with the $if...else$..if ladder. However, the syntax of the switch statement is much easier to read and write.

Also, sometimes it executes faster than the $if - else$ counterpart.

The switch statement in C II



The switch statement in C III

```
switch(expression)
{
       case constant1:
       statement1;
       statement2;
       ...
       case constant2:
       statement3;
       statement4;
       ...
       case constant3:
       statement5;
       statement6;
       ...
       default:
       statement7;
       ...
}
```

The switch statement in C IV

Valid switch expression:

```
switch(a)
switch(a + b + c)
switch(ch1 + a)
switch(a < b)
switch(my_func(12))
switch('a'
```

Invalid switch expression:

```
switch(a + 12.2) // expression must yield an integer value not double or float.
switch(f1)// same reason as above.
switch("string") // string is not allowed.
```

Valid case constant:

```
case 1
case 1 + 2
case 'a'
case 'a' < 'b'
```

The switch statement in C V

Invalid case constant:

```
1 case "string"  // string constants are not allowed
2 case 1.2// floating point constants are not allowed
3 case a // variables are not allowed
4 case a + b // variables are not allowed
5 case 1,2,3 // each case must contain only one constant
```

How does the switch statement work?

✓ The expression is evaluated once and compared with the values of each case label.

✓ If there is a match, the corresponding statements after the matching label are executed.

✓ If there is no match, the default statements are executed.

✓ If we do not use break, all statements after the matching label are executed.

✓ By the way, the default clause inside the switch statement is optional.

The switch statement in C VI

The default block can be placed anywhere. The position of default doesn't matter, it is still executed if no match found.

```c
// The default block is placed above other cases.
#include <stdio.h>
int main()
{
	int x = 4;
	switch (x)
	{
		default: printf("Choice other than 1 and 2");
		break;
		case 1: printf("Choice is 1");
		break;
		case 2: printf("Choice is 2");
		break;
	}
	return 0;
}
```

The switch statement in C VII

✓ The statements written above cases are never executed After the switch statement, the control transfers to the matching case, the statements written before case are not executed.

```c
// Statements before all cases are never executed
#include <stdio.h>
int main()
{
	int x = 1;
	switch (x)
	{
		x = x + 1;  // This statement is not executed
		case 1: printf("Choice is 1");
		break;
		case 2: printf("Choice is 2");
		break;
		default: printf("Choice other than 1 and 2");
		break;
	}
	return 0;
}
```

✓ Two case labels cannot have same value

The switch statement in C VIII

```c
// Program where two case labels have same value
#include <stdio.h>
int main()
{
	int x = 1;
	switch (x)
	{
		case 2: printf("Choice is 1");
		break;
		case 1+1: printf("Choice is 2");
		break;
	}
	return 0;
}
```

✓ The integral expressions used in labels must be a constant expressions.

The switch statement in C IX

```c
// A program with variable expressions in labels
#include <stdio.h>
int main()
{
	int x = 2;
	int arr[] =
	{
		1, 2, 3
	}
	;
	switch (x)
	{
		case arr[0]: printf("Choice 1\n");
		case arr[1]: printf("Choice 2\n");
		case arr[2]: printf("Choice 3\n");
	}
	return 0;
}
```

The switch statement in C X

```c
#include<stdio.h>

int main()
{
	int i, sum;

	printf("Enter a number: \n");
	scanf("%d", &i);

	switch(i)
	{
		case 1:
		printf("Number is one\n");
		break;
		case 2:
		printf("Number is two\n");
		break;
		case 3:
		printf("Number is three\n");
		break;
		case 4:
		printf("Number is four\n");
		break;
		default:
		printf("something else\n");
```

The switch statement in C XI

```
26      }
27
28      // return 0 to operating system
29      return 0;
30  }
```

```
1   // Program to create a simple calculator
2   #include <stdio.h>
3   int main()
4   {
5       char operator;
6       double n1, n2;
7
8       printf("Enter an operator (+, -, *, /): ");
9       scanf("%c", &operator);
10      printf("Enter two operands: ");
11      scanf("%lf %lf",&n1, &n2);
12
13      switch(operator)
14      {
15          case '+':
16          printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
17          break;
18
19          case '-':
```

The switch statement in C XII

```
20              printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
21              break;
22
23              case '*':
24              printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
25              break;
26
27              case '/':
28              printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
29              break;
30
31              // operator doesn't match any case constant +, -, *, /
32              default:
33              printf("Error! operator is not correct");
34      }
35
36      return 0;
37  }
```

The switch statement in C XIII

Nested Switch:

In C, we can have an inner switch embedded in an outer switch. Also, the case constants of the inner and outer switch may have common values and without any conflicts.

```
#include <stdio.h>
int main()
{
       int ID = 500;
       int password = 000;
       printf("Plese Enter Your ID:\n ");
       scanf("%d", &ID);
       switch (ID)
       {
              case 500:
              printf("Enter your password:\n ");
              scanf("%d", &password);
              switch (password)
              {
                     case 000:
                     printf("Welcome Dear Programmer\n");
                     break;
                     default:
```

The switch statement in C XIV

```
19                      printf("incorrect password");
20                      break;
21                }
22                break;
23                default:
24                printf("incorrect ID");
25                break;
26          }
27  }
```

The switch statement in C XV

Using range in switch case in C

you can use range of numbers instead of a single number or character in case statement.

- That is the case range extension of the GNU C compiler and not standard C
- You can specify a range of consecutive values in a single case label, like this: **case low ... high:**
- It can be used for ranges of ASCII character codes like this: **case 'A' ... 'Z':**
- You need to Write spaces around the ellipses . . . . For example, write this: **// Correct -case 1 ... 5:**
  // Wrong - case 1...5:

The switch statement in C XVI

```c
// C program to illustrate
// using range in switch case
#include <stdio.h>
int main()
{
        int arr[] =
      {
              1, 5, 15, 20
      }
      ;

       for (int i = 0; i < 4; i++)

      {
              switch (arr[i])

            {
                    case 1 ... 6:
                printf("%d in range 1 to 6\n", arr[i]);
                break;
                  case 19 ... 20:
                printf("%d in range 19 to 20\n", arr[i]);
                break;
                  default:
                printf("%d not in range\n", arr[i]);
```

The switch statement in C XVII

```
26                      break;
27
28                  }
29
30          }
31           return 0;
32  }
33  Output:
34  1 in range 1 to 6
35  5 in range 1 to 6
36  15 not in range
37  20 in range 19 to 20
```

Error conditions:

✓ low > high : The compiler gives with an error message.

✓ Overlapping case values : If the value of a case label is within a case range that has already been used in the switch statement, the compiler gives an error message.