Introduction to Programming and Data Structure

Subject Code: CS1L001

Dr Sudipta Saha

Assistant Professor

Flow Control

Computer Science and Engineering,
School of Electrical Sciences
Indian Institute of Technology Bhubaneswar

Flow Control

- Conditional statements
  - if...else Statement
  - switch Statement
- Loops
  - while Loop
  - For Loop
  - Do-While loop
  - break and continue
- Decision Examples

Conditional statements

C if Statement

General format –

```
if (test expression)
{
   // statements to be executed if the test expression is true
}
```

Conditional statements

C if Statement

General format –

- The if statement evaluates the test expression inside the parenthesis ()

```
if (test expression)
{
  // statements to be executed if the test expression is true
}
```

Conditional statements

C if Statement

General format –

```
if (test expression)
{
   // statements to be executed if the test expression is true
}
```

- The if statement evaluates the test expression inside the parenthesis ()
- *If the test expression is evaluated to true, statements inside the body of if are executed*

Conditional statements

C if Statement

General format –

```
if (test expression)
{
  // statements to be executed if the test expression is true
}
```

- The if statement evaluates the test expression inside the parenthesis ()
- *If the test expression is evaluated to false, statements inside the body of if are not executed*

Conditional statements

Example 1: Display a number if it is negative

```
#include <stdio.h>
int main()
{
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);
    // true if number is less than 0
    if (number < 0)
    {
        printf("You entered –Ve no %d.\n", number);
    } else {
        printf("You entered +Ve no %d.\n", number);
    }

  return 0;
}
```

Conditional statements

Example 1: Display a number if it is negative

```
#include <stdio.h>
int main()
{
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);
    // true if number is less than 0
    if (number < 0)
    {
        printf("You entered –Ve no %d.\n", number);
    }

   if (!(number<0)) {
        printf("You entered +Ve no %d.\n", number);
    }

  return 0;
}
```

Conditional statements

Output

```
Enter an integer: -2
You entered -2.
```

Output

```
Enter an integer: 5
```

Conditional statements

C if –else statement

General format –

```
if (test expression)
{
   // statements to be executed if the test expression is true
} else {
  // statements to be executed if the test expression is false
}
```

- The if statement evaluates the test expression inside the parenthesis ()
- *If the test expression is evaluated to false, statements inside the else body are executed, not if body*

Conditional statements

C if –else statement

General format –

```
if (test expression)
{
   // statements to be execute        st expre
} else if (….){
  // statements to be e        uted if the test expression is false
} else if (…) {
….
} else {
}
```

- The if statement evaluates the test expression inside the parenthesis ()

- *If the test expression is evaluated to false, statements inside the else body are executed, not if body*

Conditional statements

C if –else statement

If else Ladder

```
if (C1)
{
            stmt 1;
} else if (C2){
            stmt 2;
} else if (C3) {
            stmt 3;
} else {
            stmt 4;
}
```

Equivalent if statements -

```
If(C1)
            stmt 1;

If(!C1 && C2)
            stmt 2;

If(!C1 && !C2 && C3)
            stmt 3;

If(!C1 && !C2 && !C3)
            stmt 4;
```

Conditional statements

C if –else statement

If else Ladder

```
if (C1)
{
            stmt 1;
} else if (C2){
            stmt 2;
} else if (C3) {
            stmt 3;
} else {
            stmt 4;
}
```

Equivalent if statements -

```
If(C1)
            stmt 1;

else if(!C1 && C2)
            stmt 2;

else if(!C1 && !C2 && C3)
            stmt 3;

else if(!C1 && !C2 && !C3)
            stmt 4;
```

Write a program to test whether an integer is odd or even

Write a program to test whether an integer is odd or even

```
#include <stdio.h>
int main()
{
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);
```

Write a program to test whether an integer is odd or even

```
// True if the remainder is 0

    if  (number%2 == 0) {
        printf("%d is an even integer.",number);
    } else {
        printf("%d is an odd integer.",number);
    }
    return 0;
}
```

C if...else Ladder

- If-else allows only two different test cases

- But there maybe more than two possibilities

- if...else ladder allows multiple test expressions

Mutually exclusive conditions

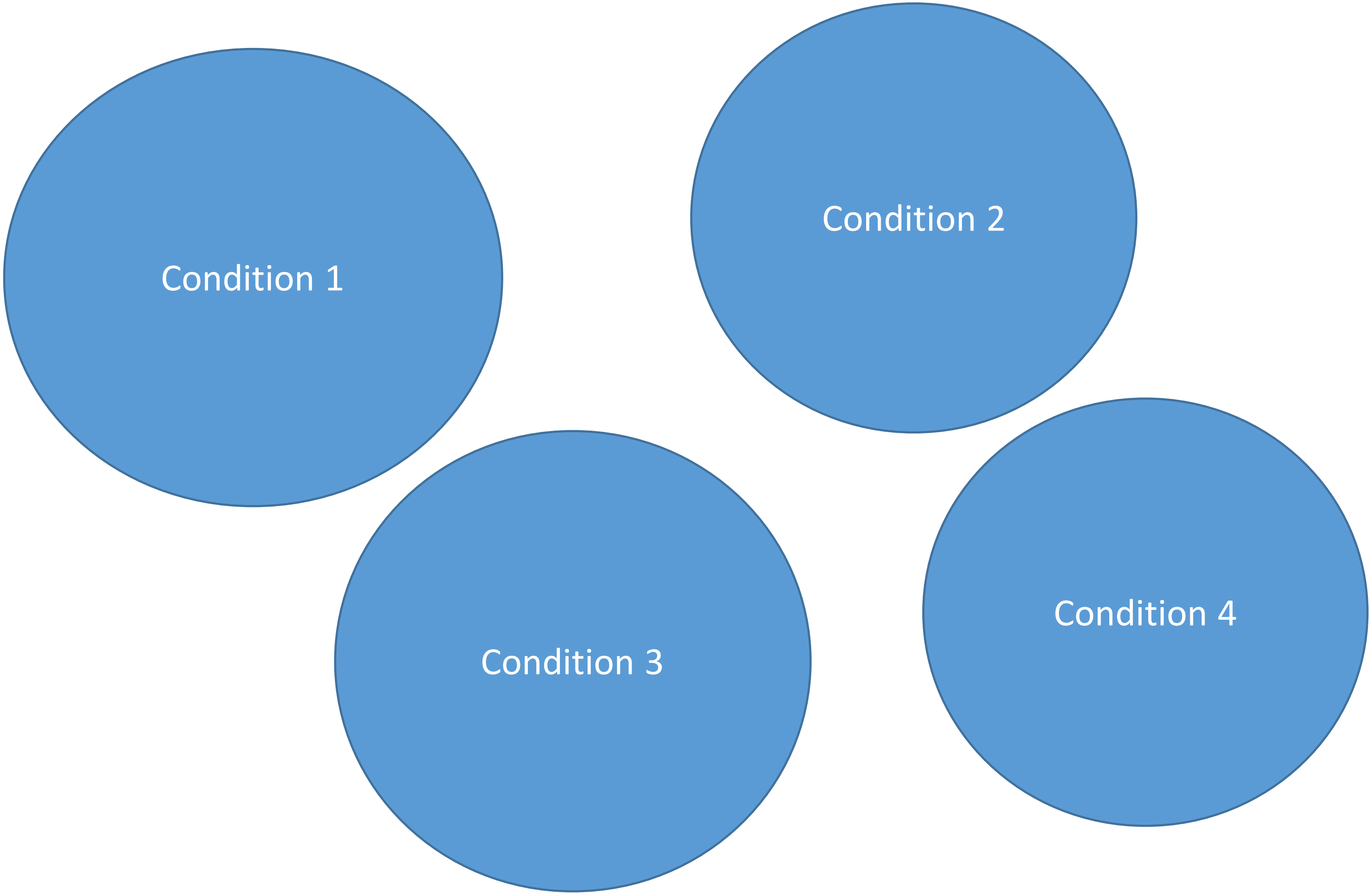
```
number = 4;

if(number==5){          printf(“YES”);              }

if(number==4){          printf(“NO”);               }

if(number==3){          printf(“YES OR NO”);        }

if(number==2){          printf(“YES AND NO”);       }
```

Not mutually exclusive conditions

```
number = 4;

if(number==5 || number ==4){ printf(“YES”);        }

if(number==4){            printf(“NO”);            }

if(number==3){            printf(“YES OR NO”);     }

if(number==2){            printf(“YES AND NO”);    }
```

Not mutually exclusive conditions

```
number = 4;

if(number==5 || number ==4){ printf(“YES”);             }

else if(number==4){          printf(“NO”);              }

else if(number==3){          printf(“YES OR NO”);       }

else if(number==2){          printf(“YES AND NO”);      }
```



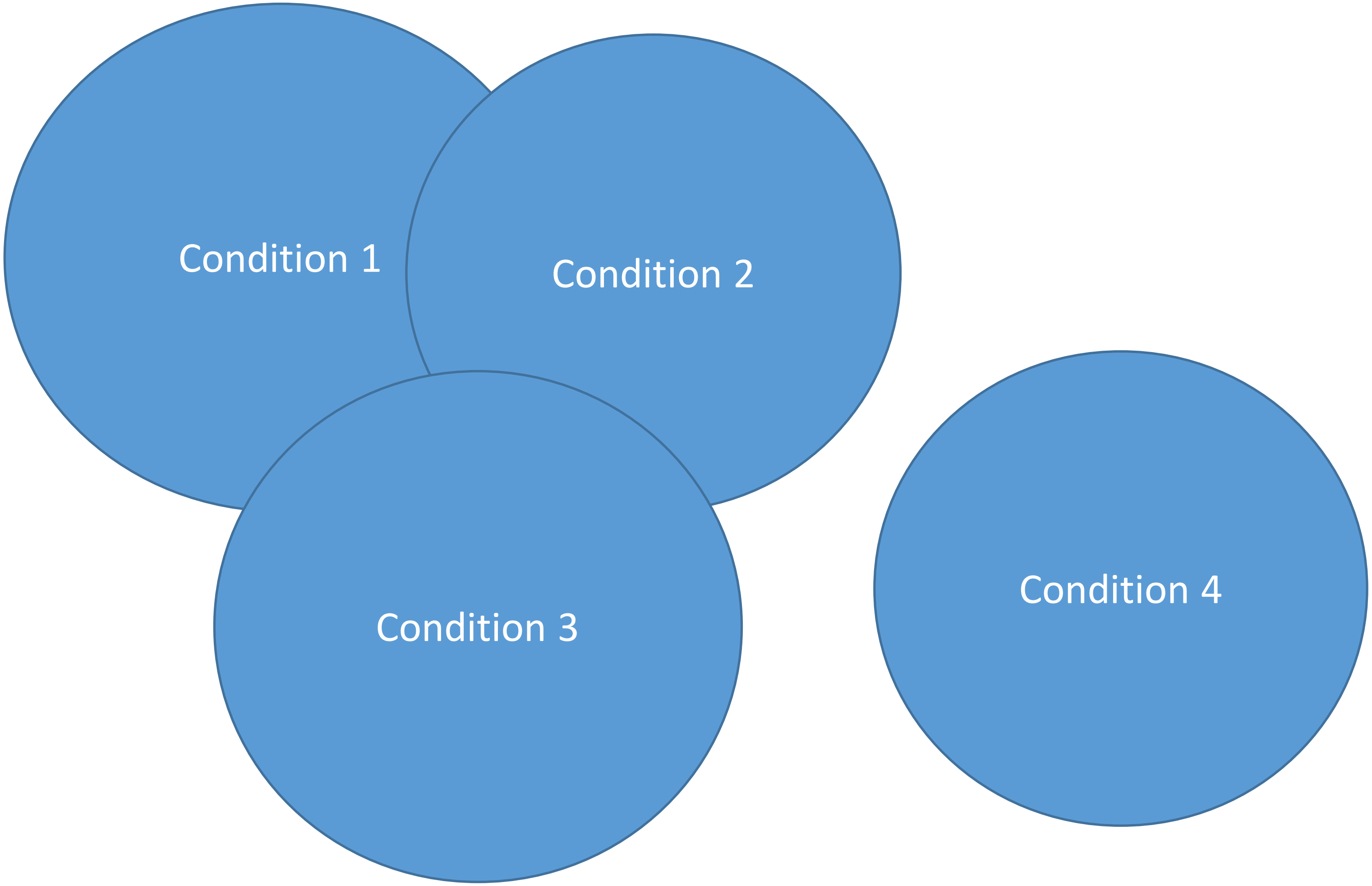
Condition 1

Condition 2

Condition 3

Condition 4

Condition 1 == (number1 == 4 && number2==2 || number1>=6) && (number4<-7)



Condition 1 == (number1 == 4 && number2==2 || number1>=6) && (number4<-7)

1. Safety / Correctness

2. Optimal number of Expression evaluation

Syntax and Semantics

```
if(a==2){
    printf(“YES\n”);
}
```

Syntax and Semantics

```
if(a==2)
    printf(“YES\n”);
```

Syntax and Semantics

```
if(a==2){
    printf(“YES\n”);
    x = 5;
}
```

Syntax and Semantics

```
if(a==2)
      printf(“YES\n”);
      x = 5;
```

Wrong....

```
if(a=2) //.... if(2) ... if(true)
     printf(“YES\n”);
     x = 5;
```

General syntax

1. Syntax of nested if...else statement.
2. if (**test expression1**) {
3. // statement(s)
4. } else if(**test expression2**) {
5. // statement(s)
6. } else if (**test expression3**) { ……..
7. // statement(s)
8. } else {
9. // statement(s)
10. }

General syntax

Syntax of nested if...else statement.

```
if (test expression1) {
   // statement(s)
}
if(test expression2) {
   // statement(s)
}
if (test  expression3) {  ........
   // statement(s)
}
```

Find out the result of comparison between two integer numbers

Find out the result of comparison between two integer numbers

```
 if(number1 == number2) {
        printf("Result: %d = %d",number1,number2);
  } else if (number1 > number2) {
        printf("Result: %d > %d", number1, number2);
  } else {
        printf("Result: %d < %d",number1, number2);
  }
  return 0;
}
```

Nested if-else statement

Include an if...else statement inside the body of another if...else statement.

Nested if-else statement

What is

NESTING

Writing one statement inside another (same)

Nested if-else statement

Relate two integers (comparing) by using nested if – else – NOT using if else ladder.

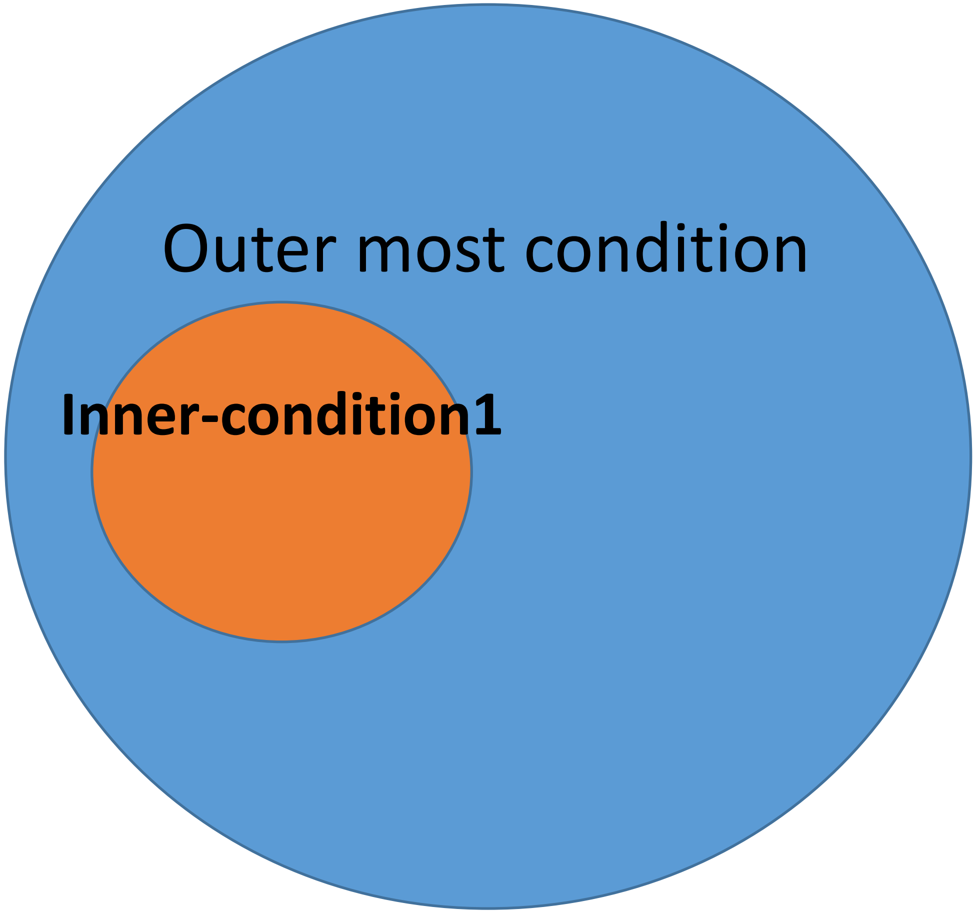
Nested if-else statement

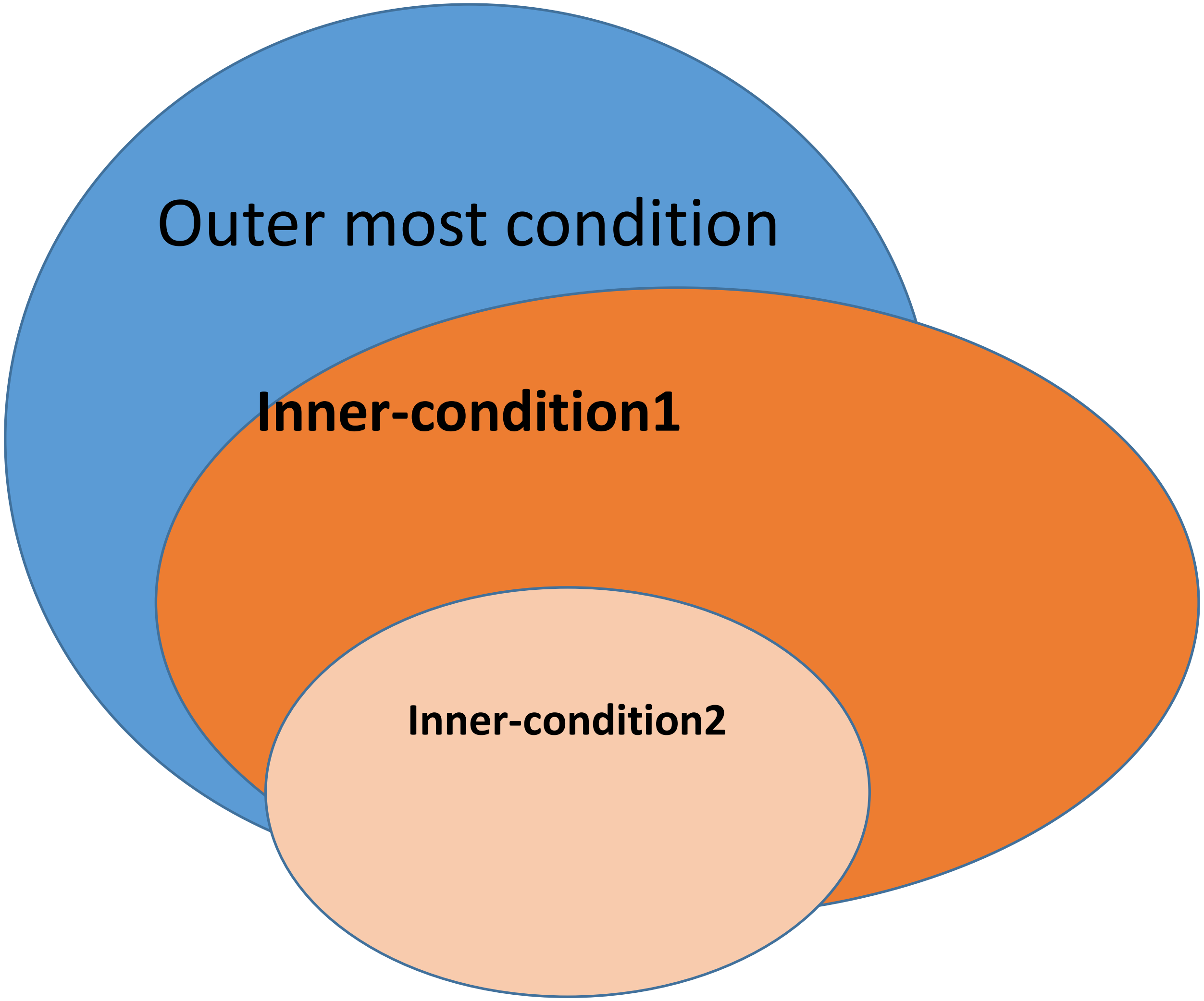
Scan two numbers from user

```
if (number1 >= number2) {
    if (number1 == number2) {
        printf("Result: %d = %d",number1,number2);
    } else {
        printf("Result: %d > %d", number1, number2);
    }
}
```

Nested if-else statement

```
    else {
        printf("Result: %d < %d",number1, number2);
    }
    return 0;
}
```


Outer most condition
Inner-condition1


Outer most condition
Inner-condition1
Inner-condition2

Use of Brackets

If the body of an if...else statement has only one statement, you do not need to use brackets {}.

```
if (a > b) {
    print("Hello World");
}
```

is equivalent to

```
if (a > b)
    print("Hello World");
```

Write a program to test an integer odd or even

Write a program to test an integer odd or even

A simple algorithm can be as follows -

```
If (the number if even)

            print – Number is even

 else
            print – Number is odd
```

Write a program to test an integer odd or even

```
+, -, /, *,%

int a, b;

scanf(…

b = a % 2;

b == 1 OR b == 0
```

Solving a problem

Two parts –

1. Syntax of if else – ladder / nested if else statements
2. Finding the semantics – domain specific formulation

Algorithm

- In mathematics and computer science, an **algorithm** is a finite sequence of well-defined, computer-implementable **instructions**, typically to solve a class of problems or to perform a computation

- Algorithms are always **unambiguous** and are used as specifications for performing **calculations**, data **processing**, automated **reasoning**, and other tasks.

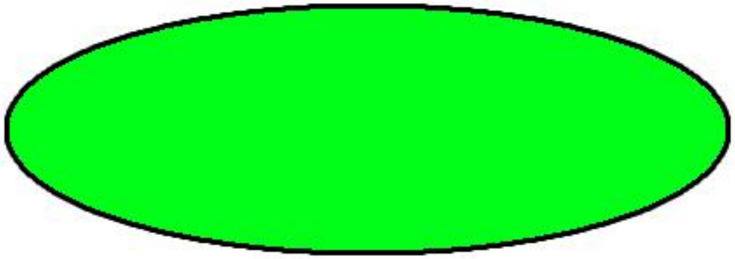
Flowchart

- **What is a Flowchart?**

- Flowchart is a graphical representation of an algorithm.
- Programmers often use it as a program-planning tool to solve a problem.
- It makes use of symbols which are connected among them to indicate the flow of information and processing.
- The process of drawing a flowchart for an algorithm is known as “flowcharting”.

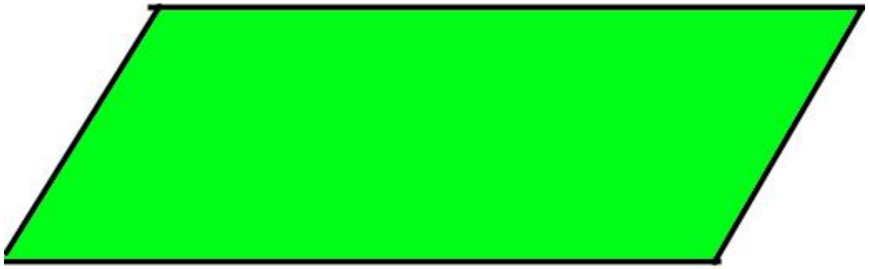
Basic Symbols used in Flowchart Designs

- **Terminal:** The oval symbol indicates **Start, Stop and Halt** in a program’s logic flow. A pause/halt is generally used in a program logic under some error conditions. Terminal is the **first** and **last** symbols in the flowchart.



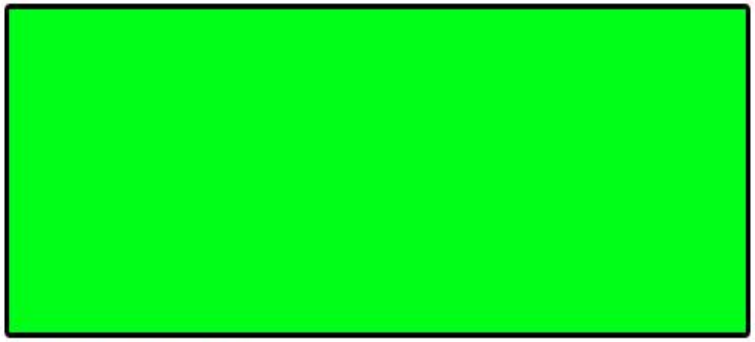
Basic Symbols used in Flowchart Designs

- **Input/Output:** A parallelogram denotes any function of input/output type. Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.



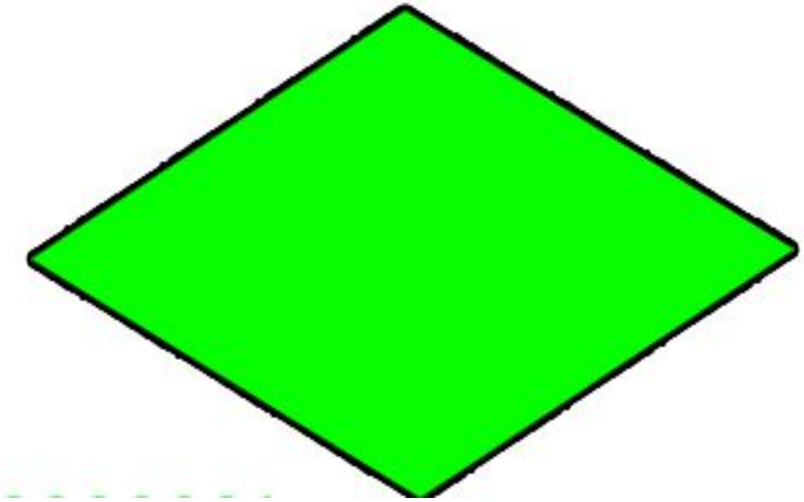
Basic Symbols used in Flowchart Designs

- **Processing:** A box represents arithmetic instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.



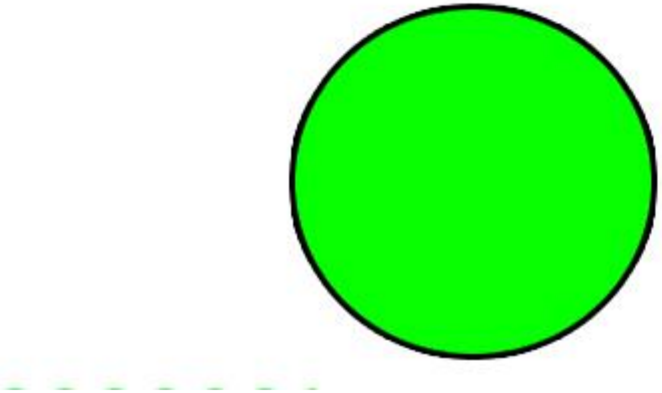
Basic Symbols used in Flowchart Designs

- **Decision:** Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.



Basic Symbols used in Flowchart Designs

- **Connectors:** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.

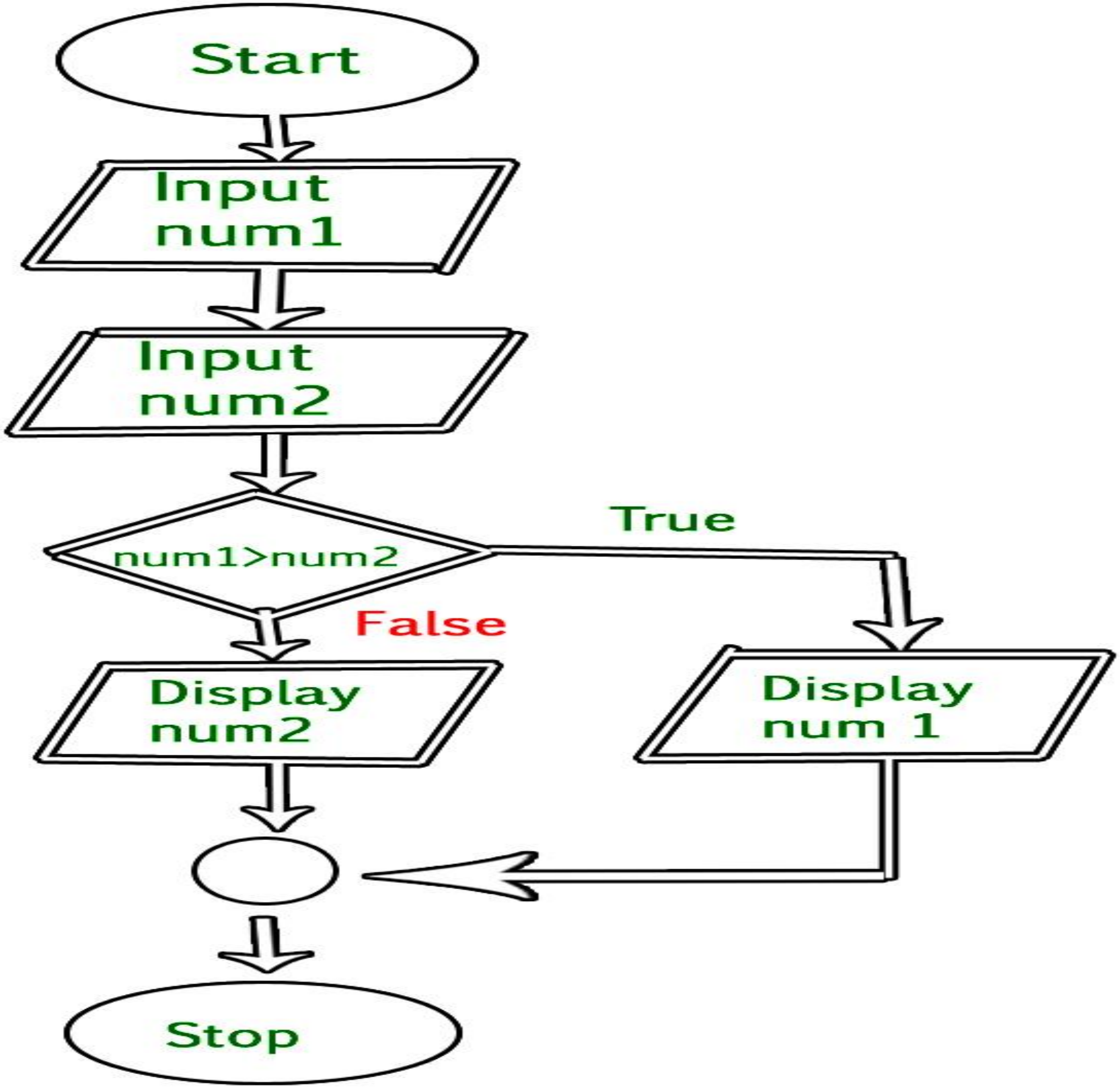


Basic Symbols used in Flowchart Designs

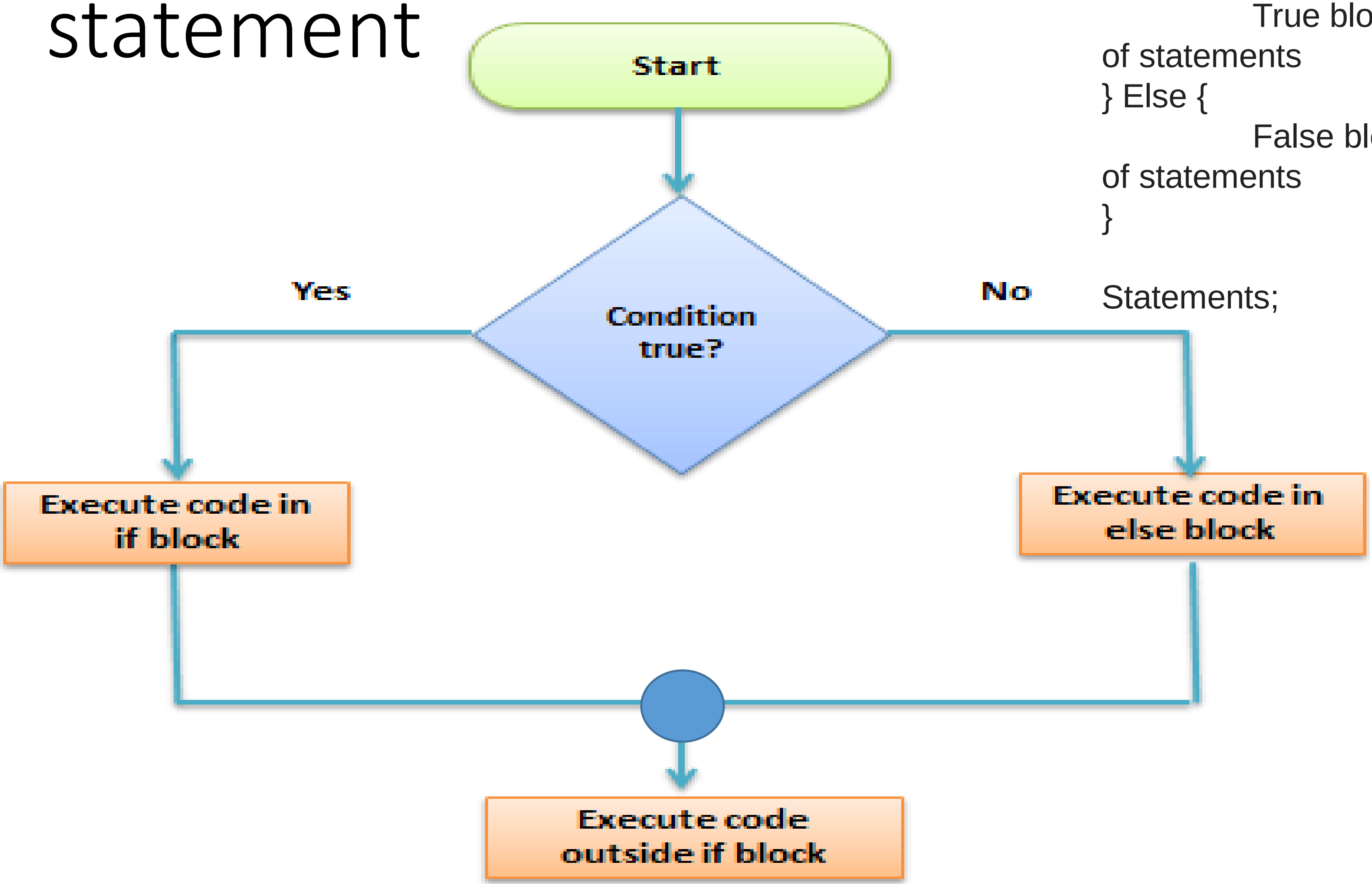
- **Flow lines:** Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.

Example

- **Draw a flowchart to input two numbers from user and display the largest of two numbers**


Start
Input num1
Input num2
num1>num2
True
False
Display num2
Display num 1
Stop

The flow chart for an if-else statement



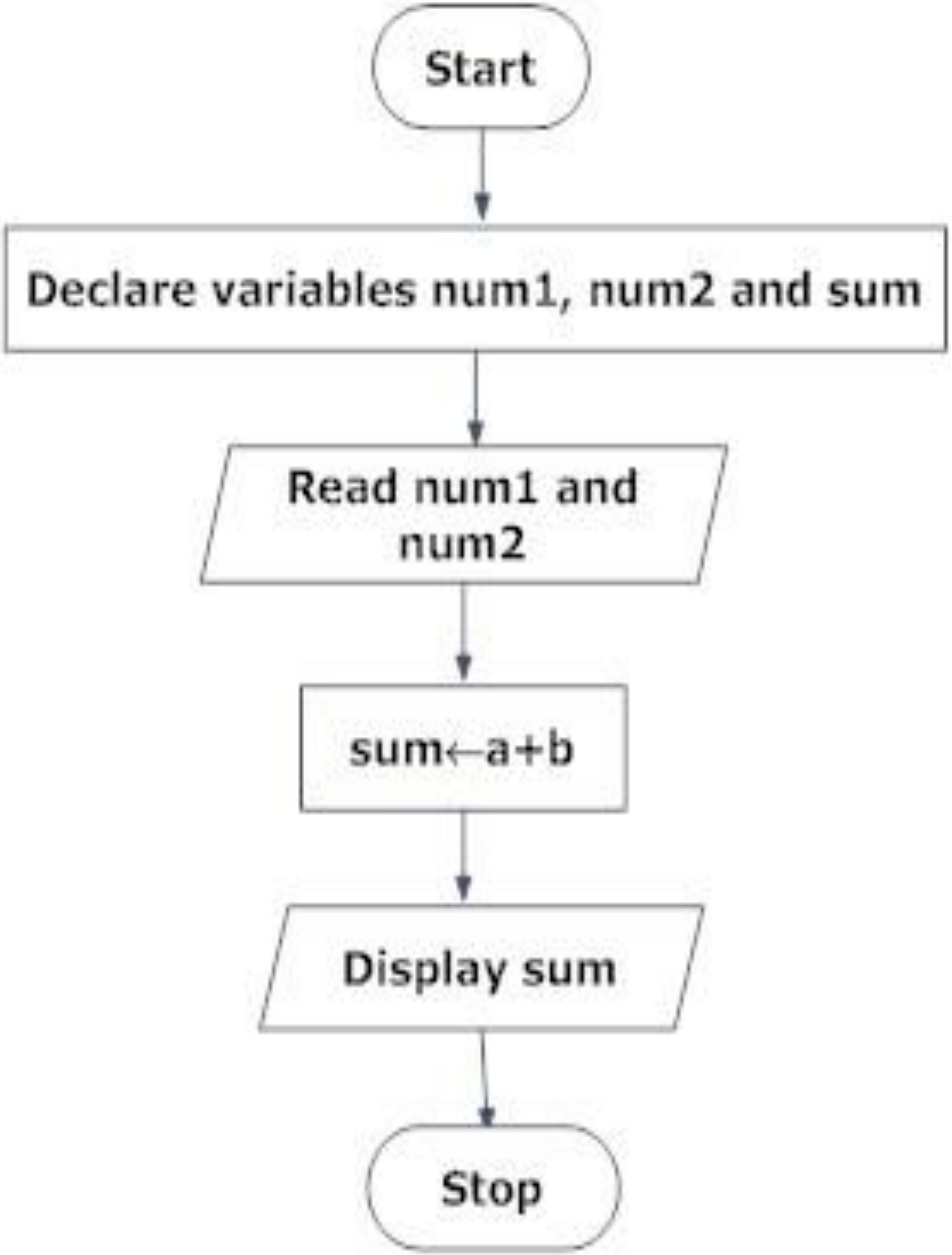
```
if (test-expression) {
            True block of statements
} Else {
            False block of statements
}

Statements;
```

Example

- **Draw a flowchart to add two numbers entered by user.**

Example

- **Draw a flowchart to add two numbers entered by user.**

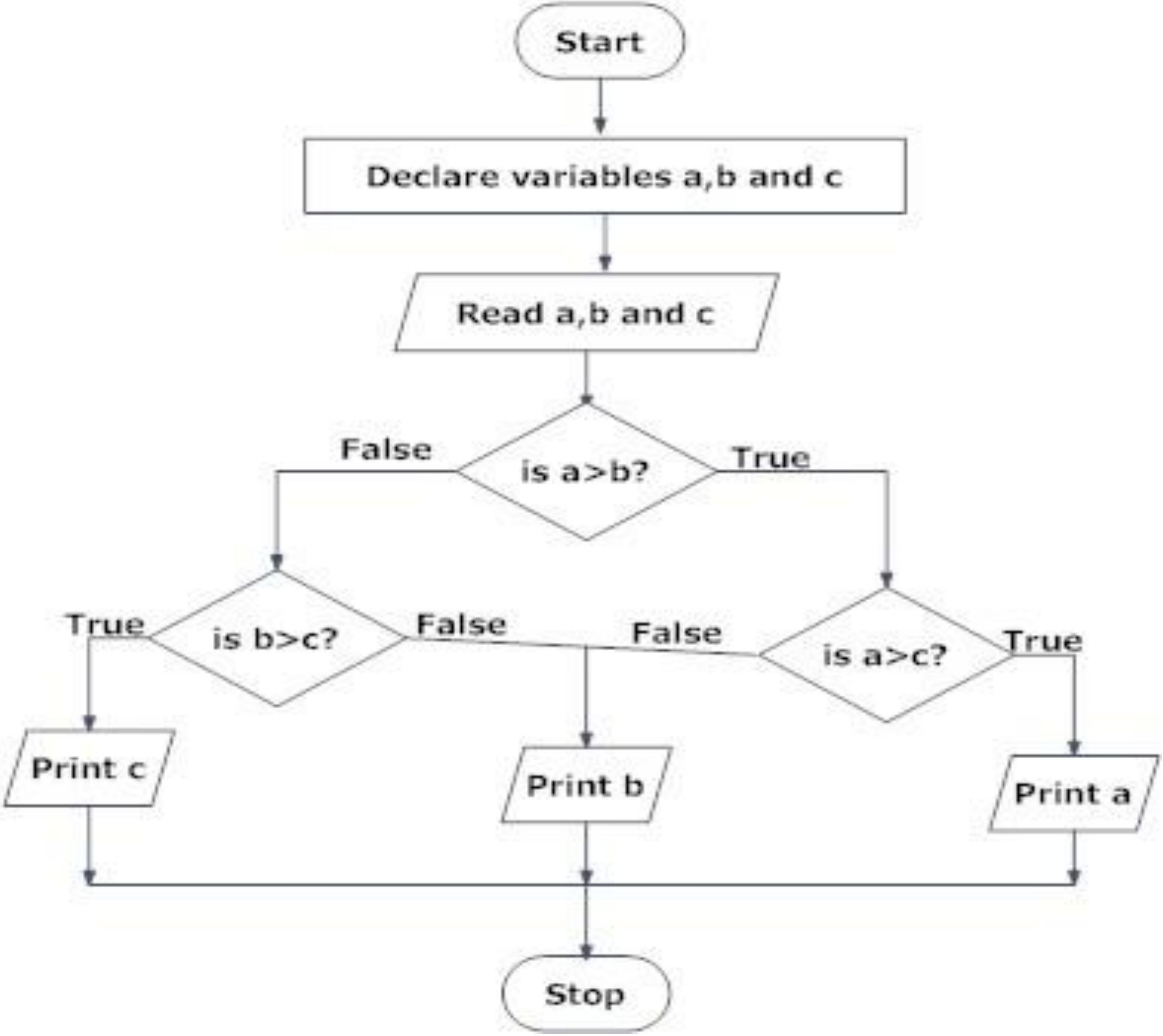


Example

- **Draw flowchart to find the largest among three different numbers entered by user.**

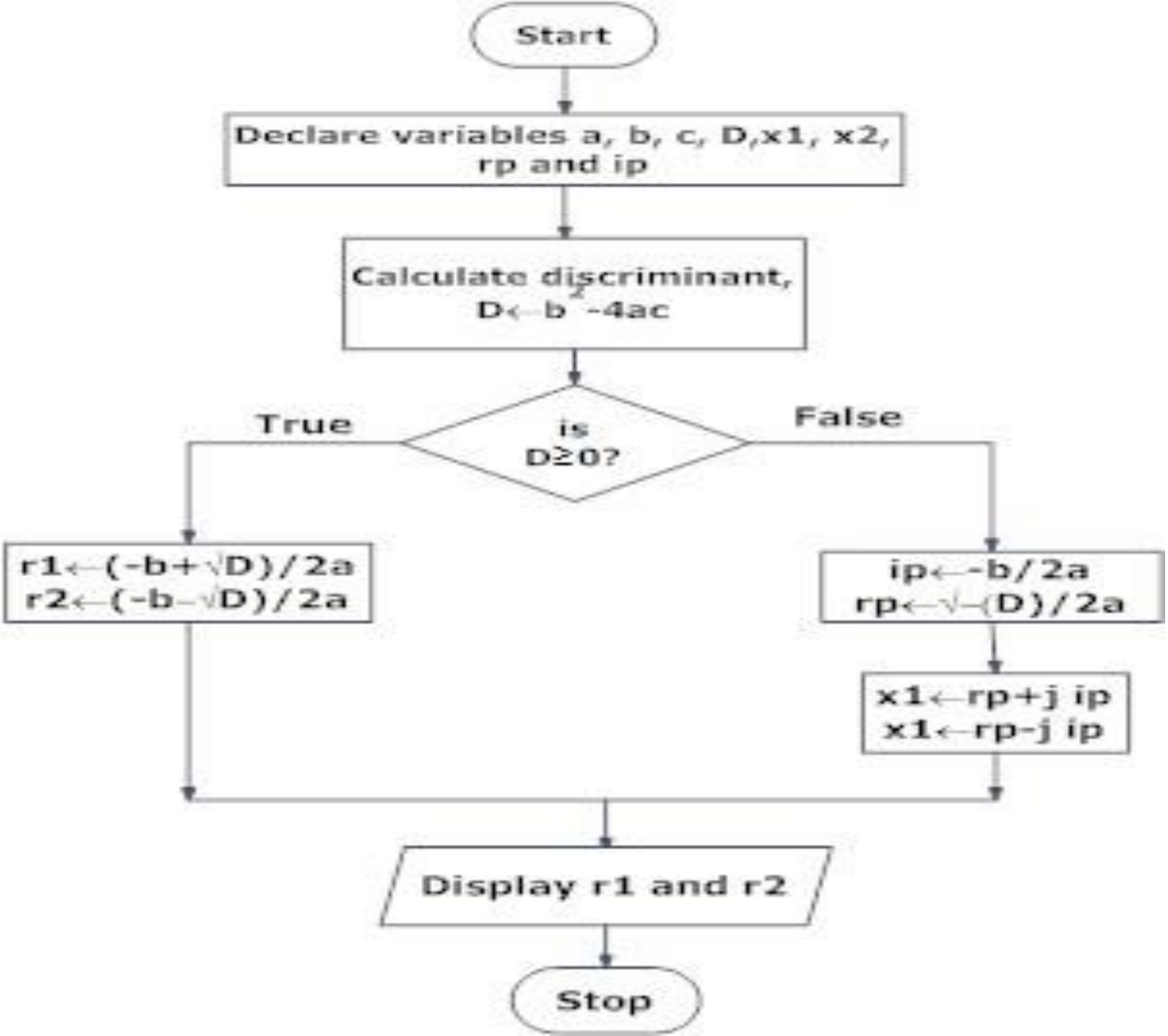
Example

- **Draw flowchart to find the largest among three different numbers entered by user.**


Start
Declare variables a,b and c
Read a,b and c
is a>b?
False
True
is b>c?
True
False
False
is a>c?
True
Print c
Print b
Print a
Stop

Example

- **Draw a flowchart to find all the roots of a quadratic equation $ax^2+bx+c=0$**



Start

Declare variables a, b, c, D, x1, x2, rp and ip

Calculate discriminant, $D \leftarrow b^2 - 4ac$

is $D \geq 0$?

True

False

$r1 \leftarrow (-b+\sqrt{D})/2a$

$r2 \leftarrow (-b-\sqrt{D})/2a$

$ip \leftarrow -b/2a$

$rp \leftarrow \sqrt{-(D)}/2a$

$x1 \leftarrow rp+j\ ip$

$x1 \leftarrow rp-j\ ip$

Display r1 and r2

Stop

Switch case

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

Switch case

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

- 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.

Syntax

```
switch (expression)
{
    case constant1:
      // statements
      break;
    case constant2:
      // statements
      break;
    .
    .
    default:
      // default statements
}
```

Syntax

```
switch (expression)
{
   case constant1:
     // statements
     break;
   case constant2:
     // statements
     break;
   .
   .
   .
   default:
     // default statements
}
```

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.

For example, if the value of the expression is equal to constant2, statements after case constant2: are executed until break is encountered.

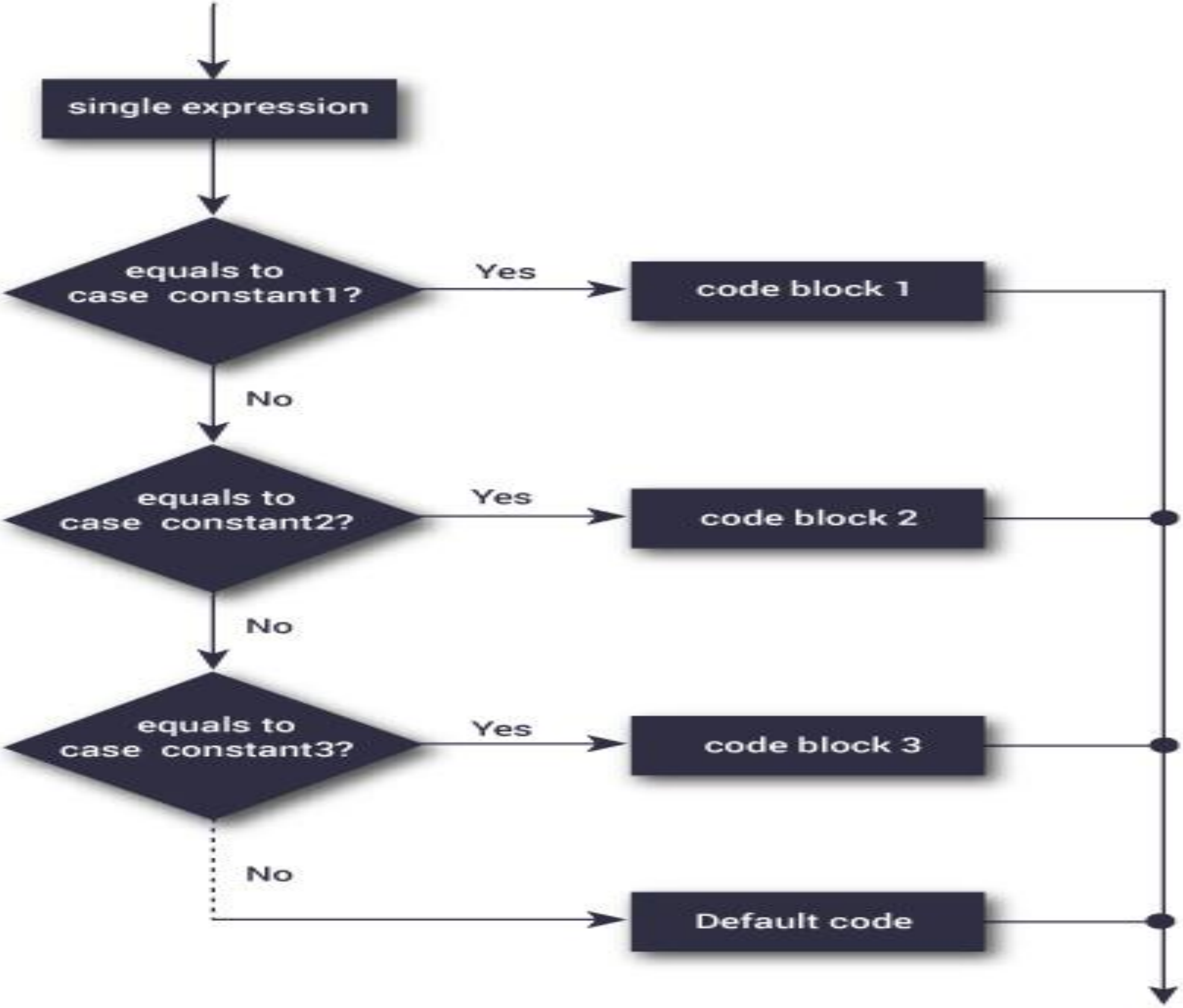
If there is no match, the default statements are executed.

Syntax

```
switch (expression)
{
   case constant1:
     // statements
     break;
   case constant2:
     // statements
     break;
   .
   .
   .
   default:
     // default statements
}
```

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.



single expression
equals to case constant1?
Yes
code block 1
No
equals to case constant2?
Yes
code block 2
No
equals to case constant3?
Yes
code block 3
No
Default code

Switch-case: Example

```
#include <stdio.h>

    int main() {

        int num = 8;
        switch (num) {

            case 7:
                printf("Value is 7");
                break;
            case 8:
                printf("Value is 8");
                break;
            case 9:
                printf("Value is 9");
                break;
            default:
                printf("Out of range");
                break;
        }
        return 0;
    }
```

```
#include <stdio.h>

    int main() {

        int num = 8;
        switch (num) {

            case 7:
                printf("Value is 7");
                break;
            case 8:
                printf("Value is 8");
                break;
            case 9:
                printf("Value is 9");
                break;
            default:
                printf("Out of range");
                break;
        }
        return 0;
    }
```

Output:

```
Value is 8
```

Switch-case: Example

```
#include <stdio.h>
int main() {
int language = 10;
  switch (language) {
  case 1:
    printf("C#\n");
    break;
  case 2:
    printf("C\n");
    break;
  case 3:
    printf("C++\n");
    break;
  default:
    printf("Other programming language\n");
}
}
```

Switch-case: Example

```
#include <stdio.h>
int main() {
int language = 10;
  switch (language) {
  case 1:
    printf("C#\n");
    break;
  case 2:
    printf("C\n");
    break;
  case 3:
    printf("C++\n");
    break;
  default:
    printf("Other programming language\n");
}
}
```

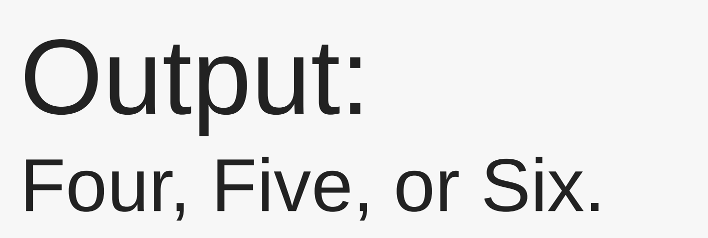
```
Other
programming
language
```

Switch-case: Example

```
#include <stdio.h>
int main() {
int number=5;
switch (number) {
  case 1:
  case 2:
  case 3:
    printf("One, Two, or Three.\n");
    break;
  case 4:
  case 5:
  case 6:
    printf("Four, Five, or Six.\n");
    break;
  default:
    printf("Greater than Six.\n");
}
}
```

Switch-case: Example

```
#include <stdio.h>
int main() {
int number=5;
switch (number) {
  case 1:
  case 2:
  case 3:
    printf("One, Two, or Three.\n");
    break;
  case 4:
  case 5:
  case 6:
    printf("Four, Five, or Six.\n");
    break;
  default:
    printf("Greater than Six.\n");
}

Adsbhwekjf

}
```



```
Output:
Four, Five, or Six.
```

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.

Write a nested switch case base solution for –

1. ***User will have to type his own ID***
2. ***if the ID is valid it will ask him to enter his password***
3. ***if the password is correct the program will print the name of the user, otherwise ,the program will print Incorrect Password***
4. ***if the ID does not exist , the program will print Incorrect ID***

Nested Switch

```
#include <stdio.h>
int main() {
    int ID = 500;
    int password = 000;
    printf("Please 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:
            printf("incorrect password");
            break;
        }
        break;
      default:
        printf("incorrect ID");
        break;
    }
}
```

Nested Switch

```
OUTPUT:

Please Enter Your ID:
 500
Enter your password:
 000
Welcome Dear Programmer
```

Loops

Purpose –

To repeat a block of code until a specified condition is met

- for loop
- while loop
- do...while loop

- Write / print your name 10 times using a program.

```
#include<…>
main (){

        // One unit of the task set
        printf(“<Your name>\n”);
        printf(“<Your name>\n”);
        printf(“<Your name>\n”);
        printf(“<Your name>\n”);
        printf(“<Your name>\n”);
        ….

}
```

- Write / print your name 1000 times using a program.

```
#include<…>
main (){

        // One unit of the task set
        printf(“<Your name>\n”);
        printf(“<Your name>\n”);
        printf(“<Your name>\n”);
        printf(“<Your name>\n”);
        printf(“<Your name>\n”);
        ….

}
```

- Syntax rules
- Semantics

For loop

The syntax of the for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
   // statements inside the body of loop
}
…
```

For loop

The **initialization statement** is executed only once.

Test expression is evaluated.

If false, loop terminates.

if true, **body of for loop** is executed.

For loop

And the **update expression** is updated.

Again the **test expression** is evaluated.

Goes on until **test expression** is false. When the **test expression** is false, the loop terminates.

```c
// C program to illustrate need of loops
#include <stdio.h>

int main()
{
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");

    return 0;
}
```

Printing Hello World using loop

```
// C program to illustrate need of loops
#include <stdio.h>

int main()                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
{
   int i=0;
        for(i=1 ;    i<=10 ;    ++i)
           printf( "Hello World\n");

   return 0;
}
```

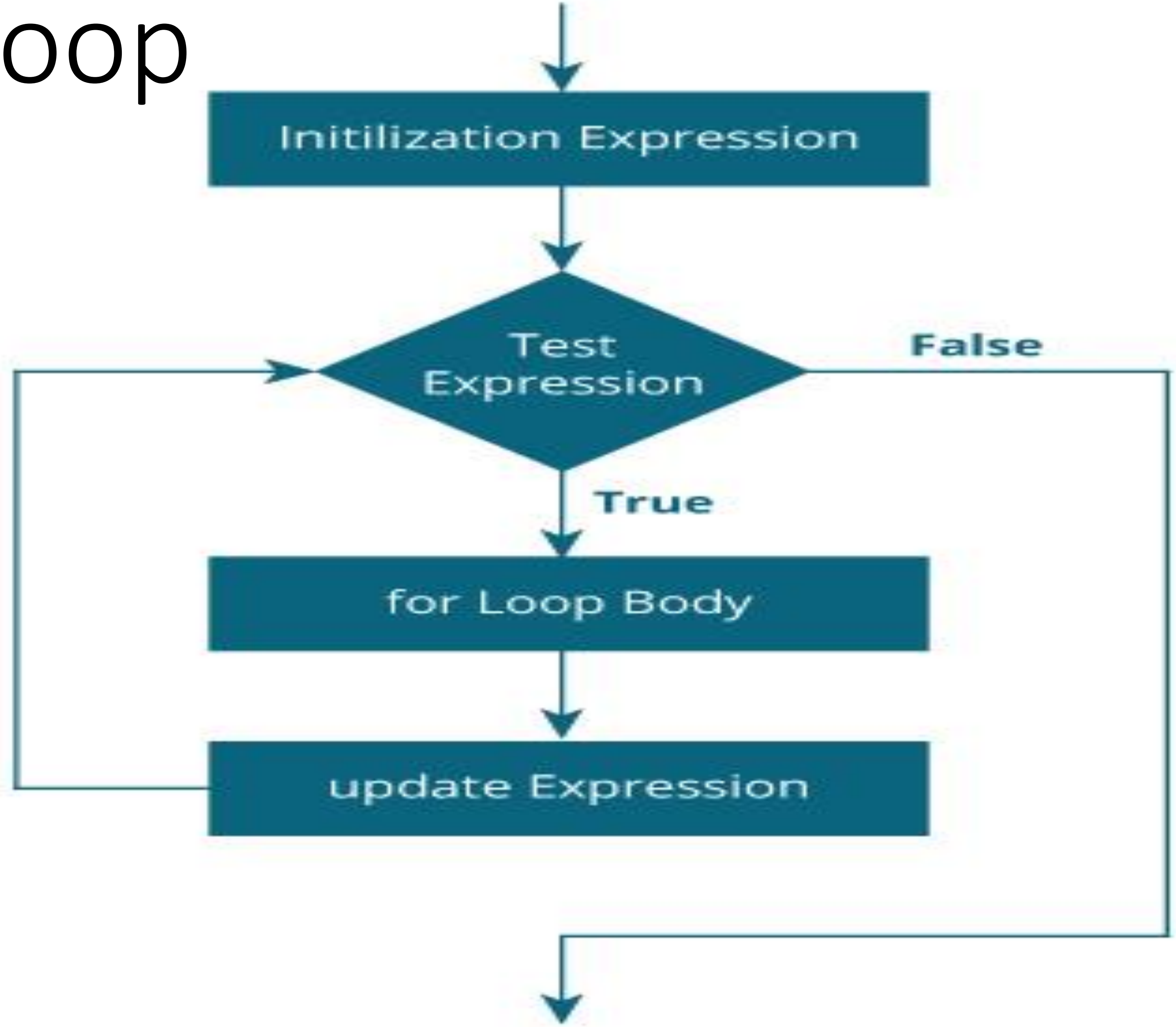
Printing Hello World using loop

```
// C program to illustrate need of loops
#include <stdio.h>

int main()               0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
{
   int i=0;
        for(;;i++)
           printf( "Hello World\n");

   return 0;
}
```

Printing Hello World using loop

```
// C program to illustrate need of loops
#include <stdio.h>

int main()
{
    int i=0;
           for(  ;    ;   )
               printf( "Hello World\n");

    return 0;
}
```

For loop


Initilization Expression
Test Expression
False
True
for Loop Body
update Expression

Print numbers from 1 to 100

```
#include <stdio.h>
int main() {
  int i;
  for (i = 1; i < 101; ++i)
  {
    printf("%d ", i);
  }
  return 0;
}
```

Print numbers from 1 to 100

Output:

1 2 3 …. 100

Program to calculate the sum of first n natural numbers (Positive integers 1,2,3...n are known as natural numbers)

```
#include <stdio.h>
int main()
{
    int num, count=0, sum = 0;
    printf("Enter a positive integer: ");
    scanf("%d", &num);
    for(count = 1; count <= num; ++count) {
        sum += count;
    }
    printf("Sum = %d", sum);
    return 0;
}
```

Output

```
Enter a positive integer: 10
Sum = 55
```

Explanation of the code -

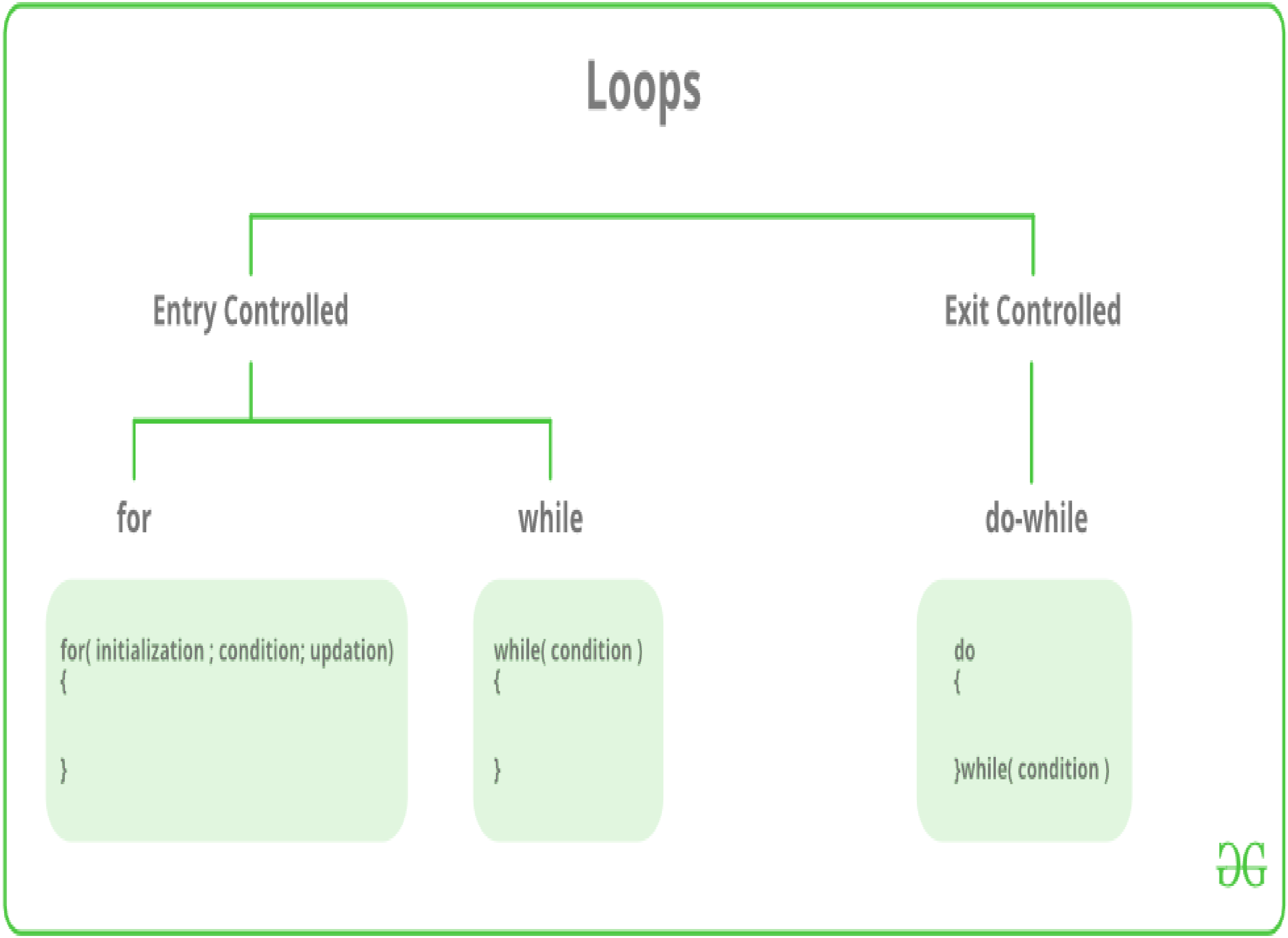
Output

```
Enter a positive integer: 10
Sum = 55
```

Explanation of the code -

Types of loops

- There are mainly two types of loops:
- **Entry Controlled loops**: In this type of loops the test condition is tested before entering the loop body. **For Loop** and **While Loop** are entry controlled loops.
- **Exit Controlled Loops**: In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. **do – while loop** is exit controlled loop.



Loops

Entry Controlled

Exit Controlled

for

while

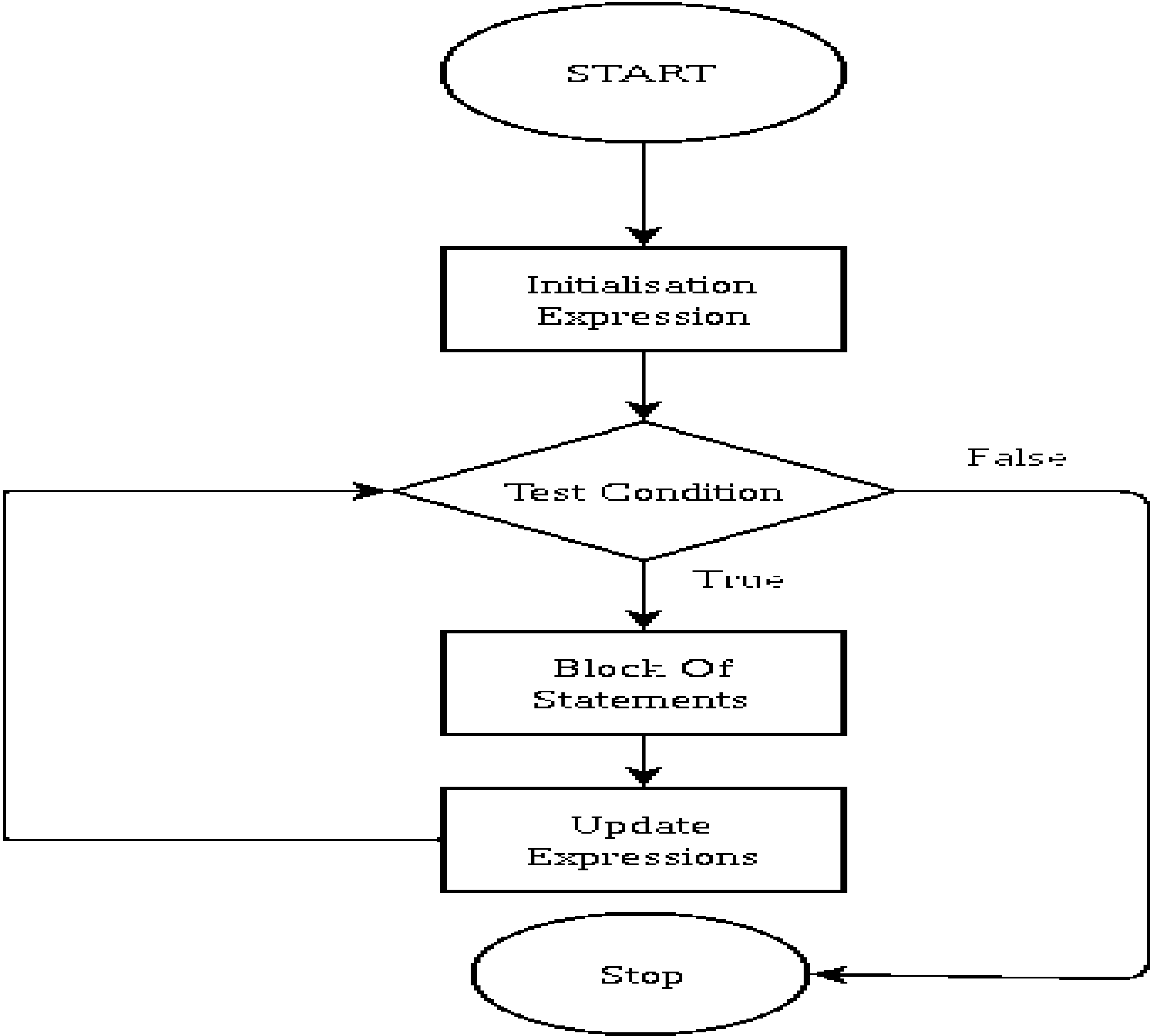
do-while

```
for( initialization ; condition; updation)
{


}
```

```
while( condition )
{


}
```

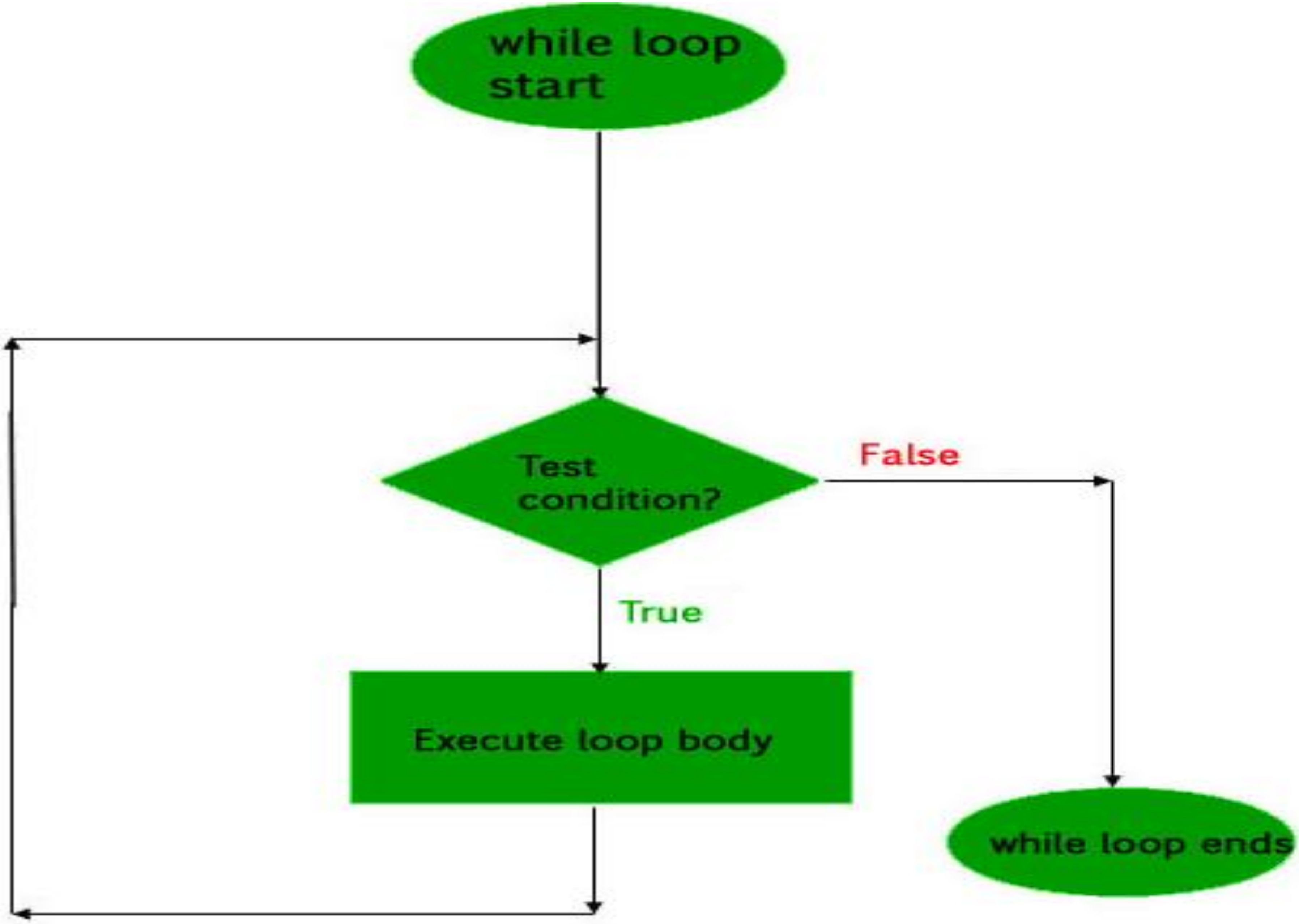
```
do
{


}while( condition )
```

Draw the flow chart / diagram of a for loop


START
Initialisation Expression
Test Condition
False
True
Block Of Statements
Update Expressions
Stop

While loop

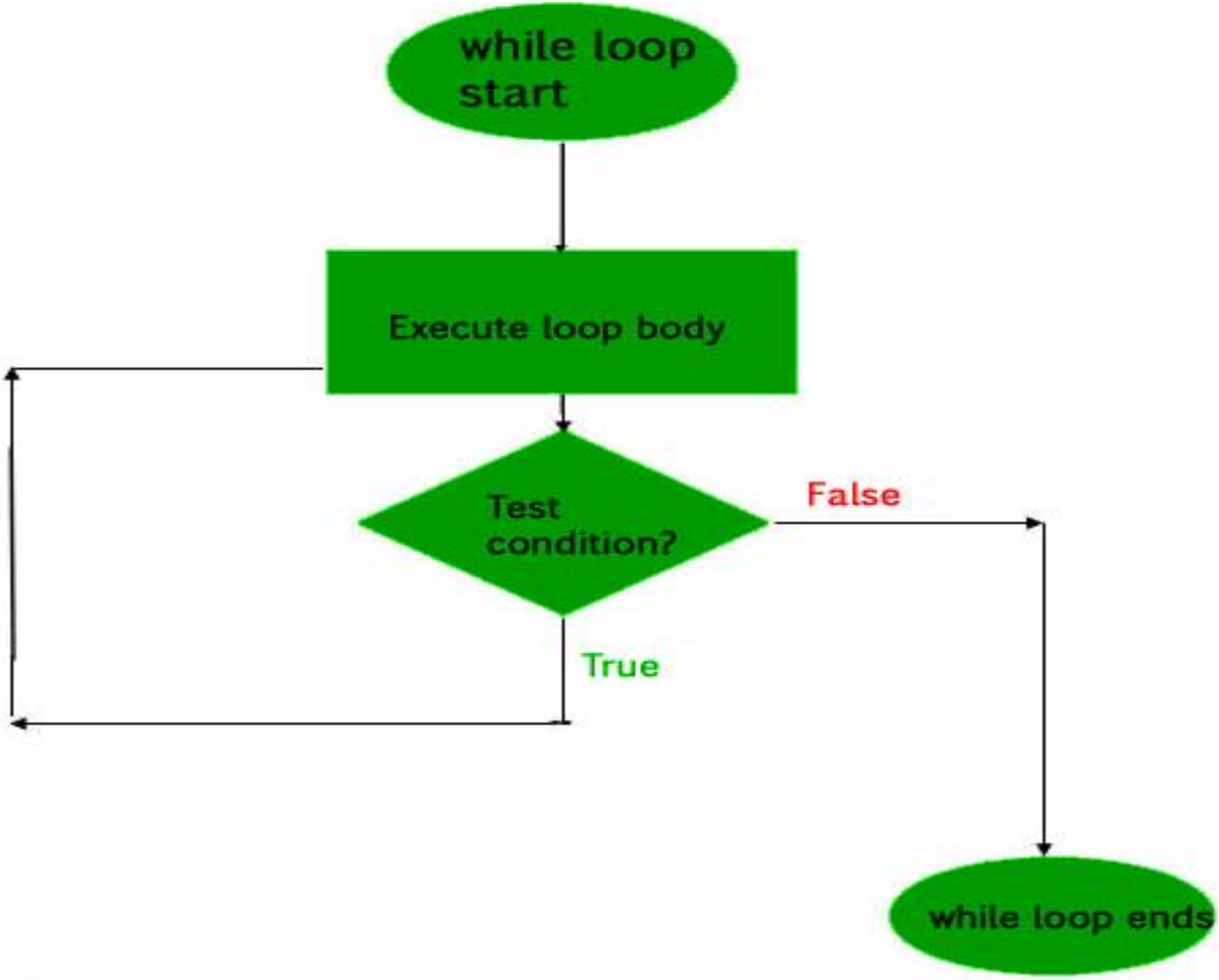
```
initialization expression;
while (test_expression)
{
   // statements

  update_expression;
}
```


while loop start
Test condition?
False
True
Execute loop body
while loop ends

```c
// C program to illustrate while loop
#include <stdio.h>

int main()
{
    // initialization expression
    int i = 1;

    // test expression
    while (i < 6)
    {
        printf( "Hello World\n");

        // update expression
        i++;
    }

    return 0;
}
```

Do while loop

```
initialization expression;
do
{
  // statements

  update_expression;
} while (test_expression);
```


while loop start
Execute loop body
Test condition?
False
True
while loop ends

```
// C program to illustrate do-while loop
#include <stdio.h>

int main()
{
   int i = 2; // Initialization expression

   do
   {
      // loop body
      printf( "Hello World\n");

      // update expression
      i++;

   }  while (i < 1);   // test expression

   return 0;
}
```

Infinite loop

An infinite loop (sometimes called an endless loop) is a piece of coding that lacks a functional exit

It repeats indefinitely.

An infinite loop occurs when a condition always evaluates to true. Usually, this is an error.

```
for ( ; ; )
   {
      printf("This loop will run forever.\n");
   }
```

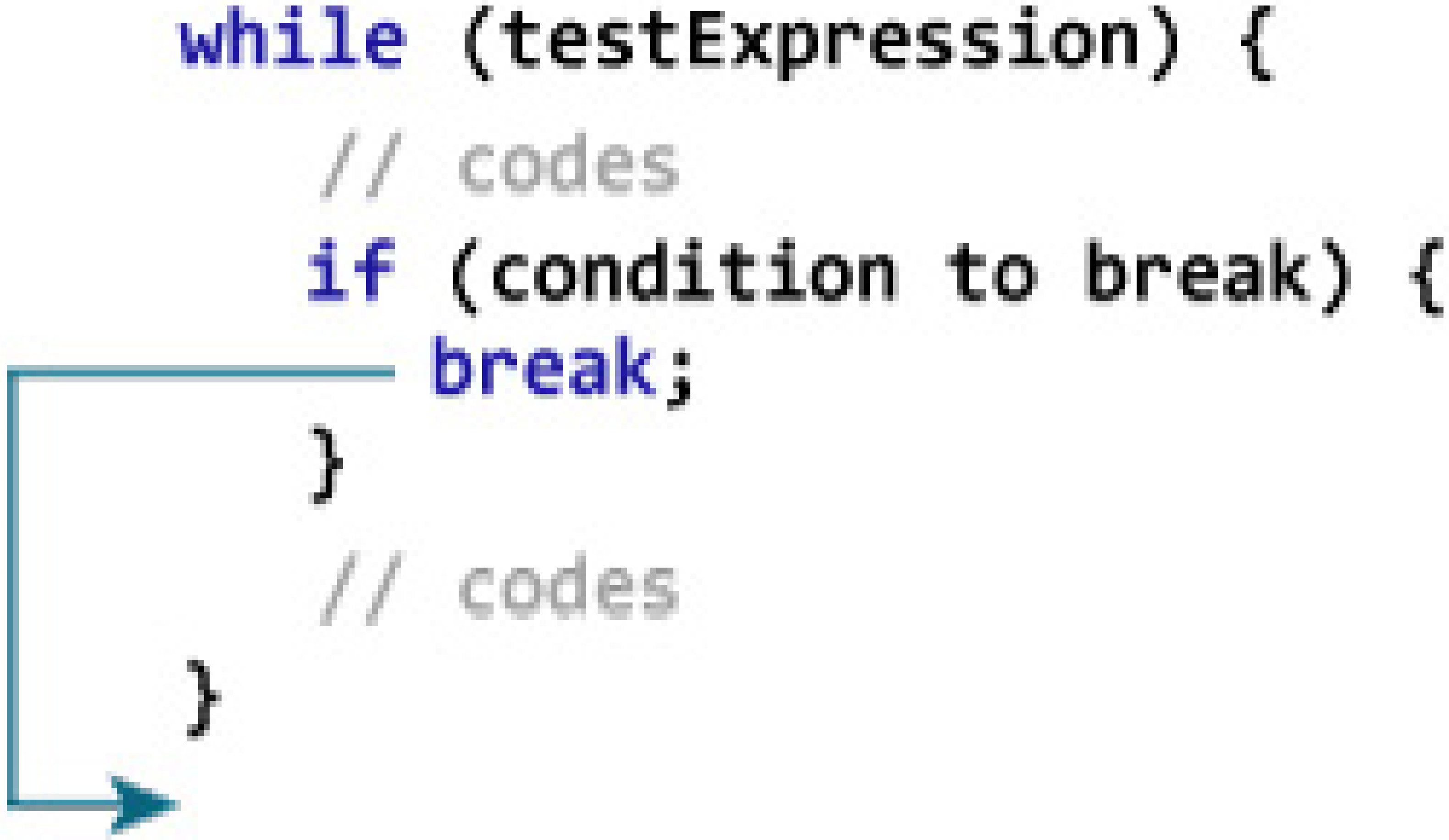
```
while (i != 0)
   {
      i-- ;
      printf( "This loop will run forever.\n");
   }
```

```
while (true)
   {
      printf( "This loop will run forever.\n");
   }
```

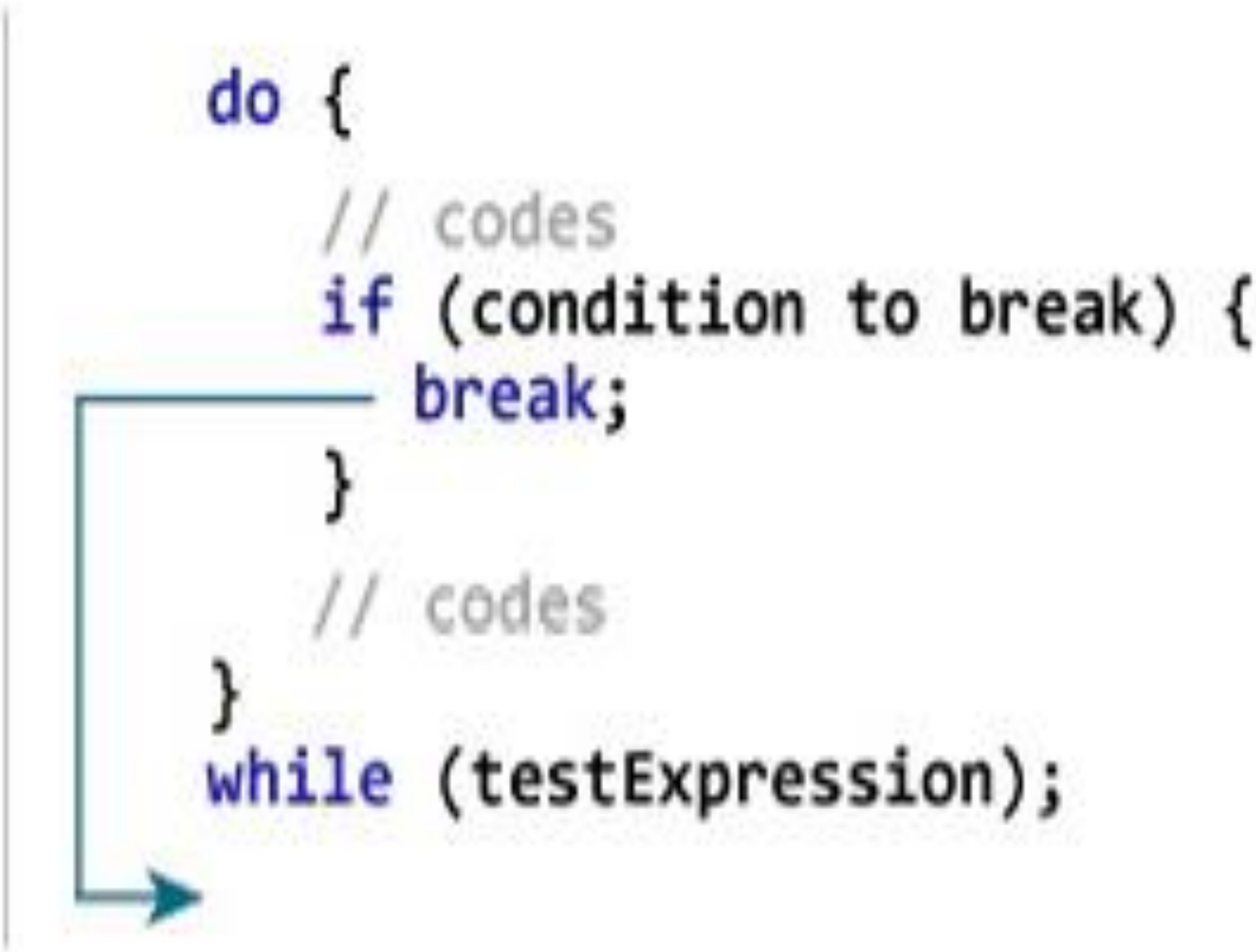
Break and Continue

The break statement ends the loop immediately when it is encountered. Its syntax is:

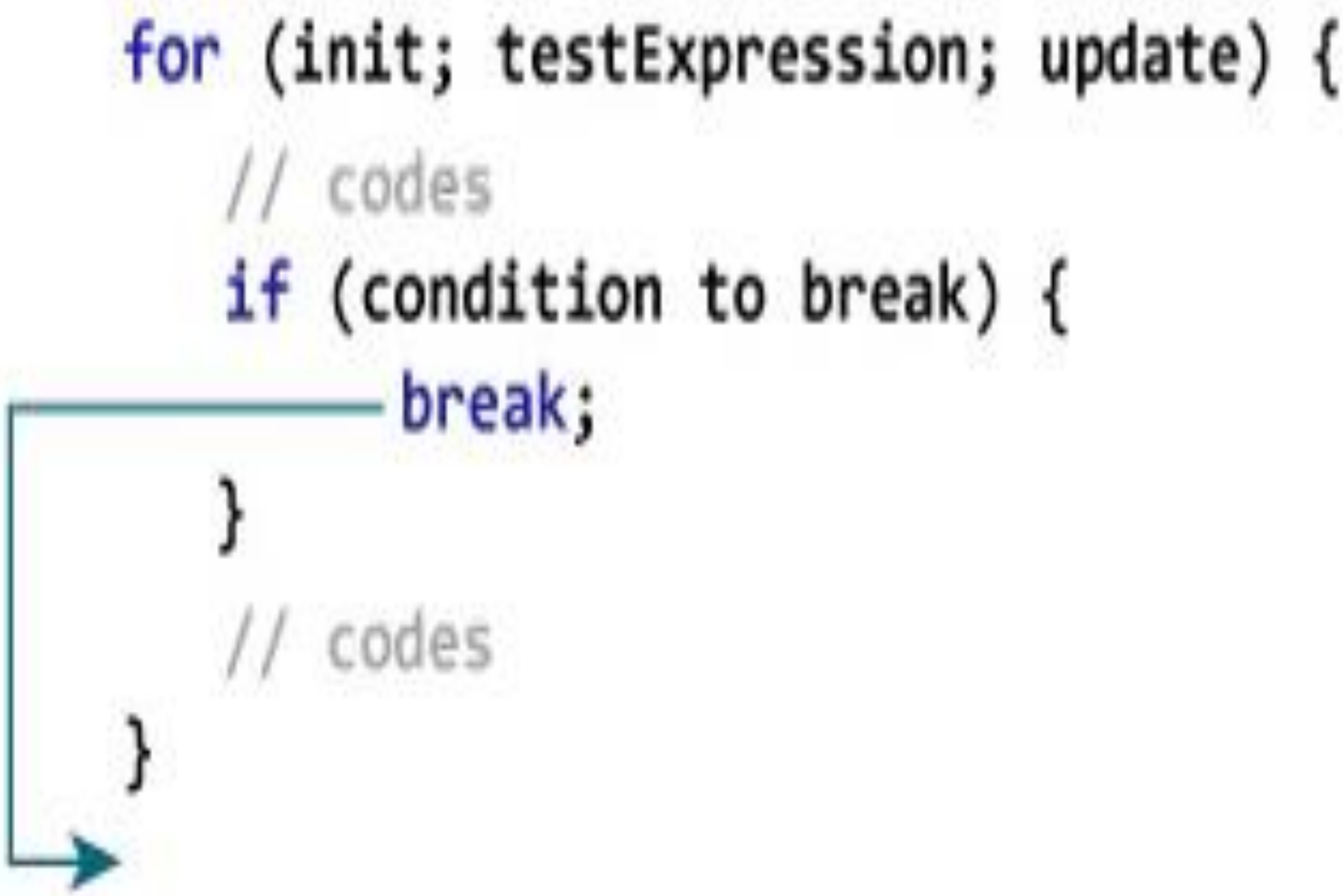
```
    break;
```



```
while (testExpression) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```



```
do {
    // codes
    if (condition to break) {
        break;
    }

    // codes
}
while (testExpression);
```



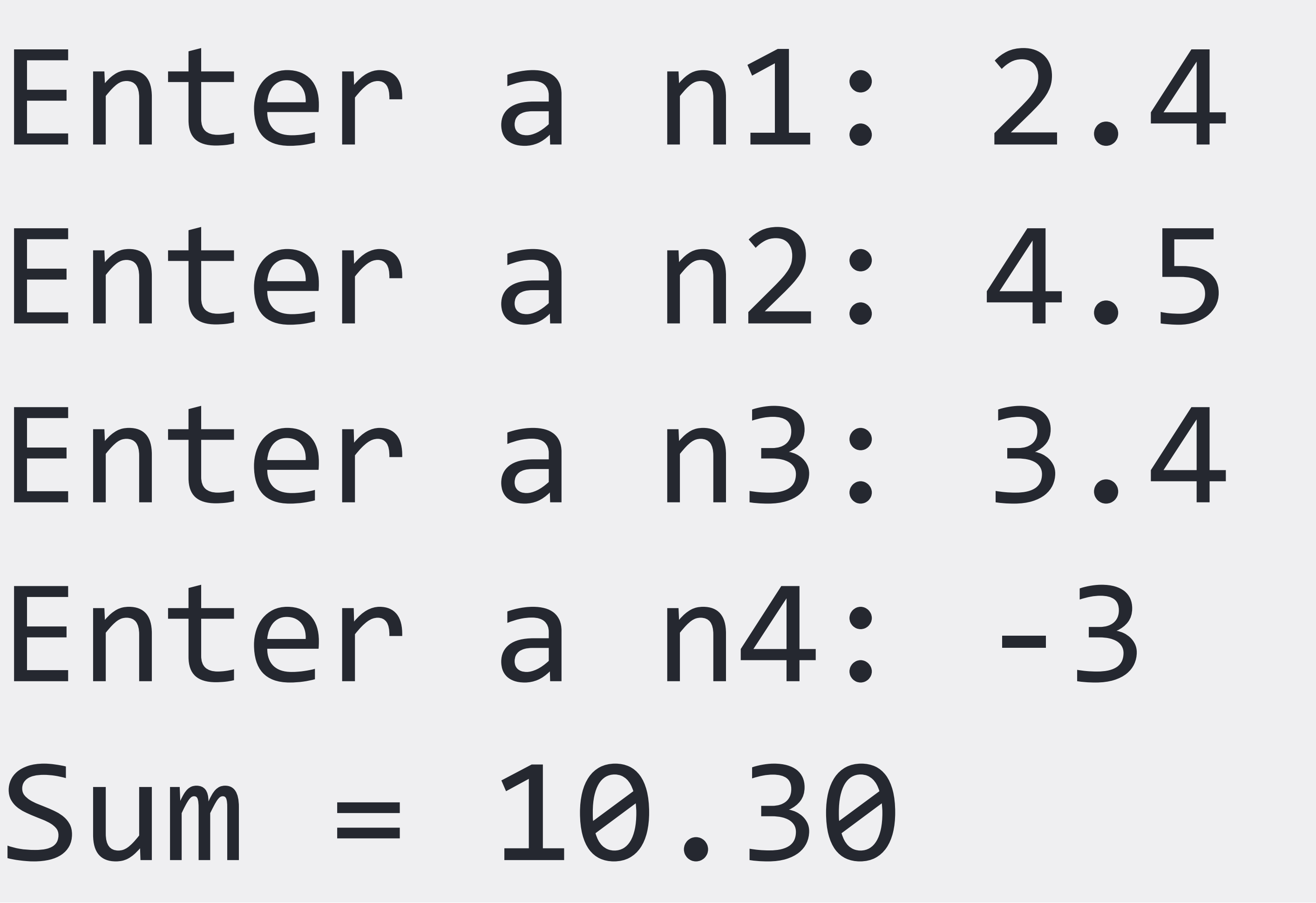
```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

Write a program

To calculate the sum of a maximum of 10 numbers

If a negative number is entered, the loop terminates

```c
# 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, the loop ends
if(number < 0.0)
{
break;
}
sum += number; // sum = sum + number;
}
printf("Sum = %.2lf",sum);
return 0;
}
```

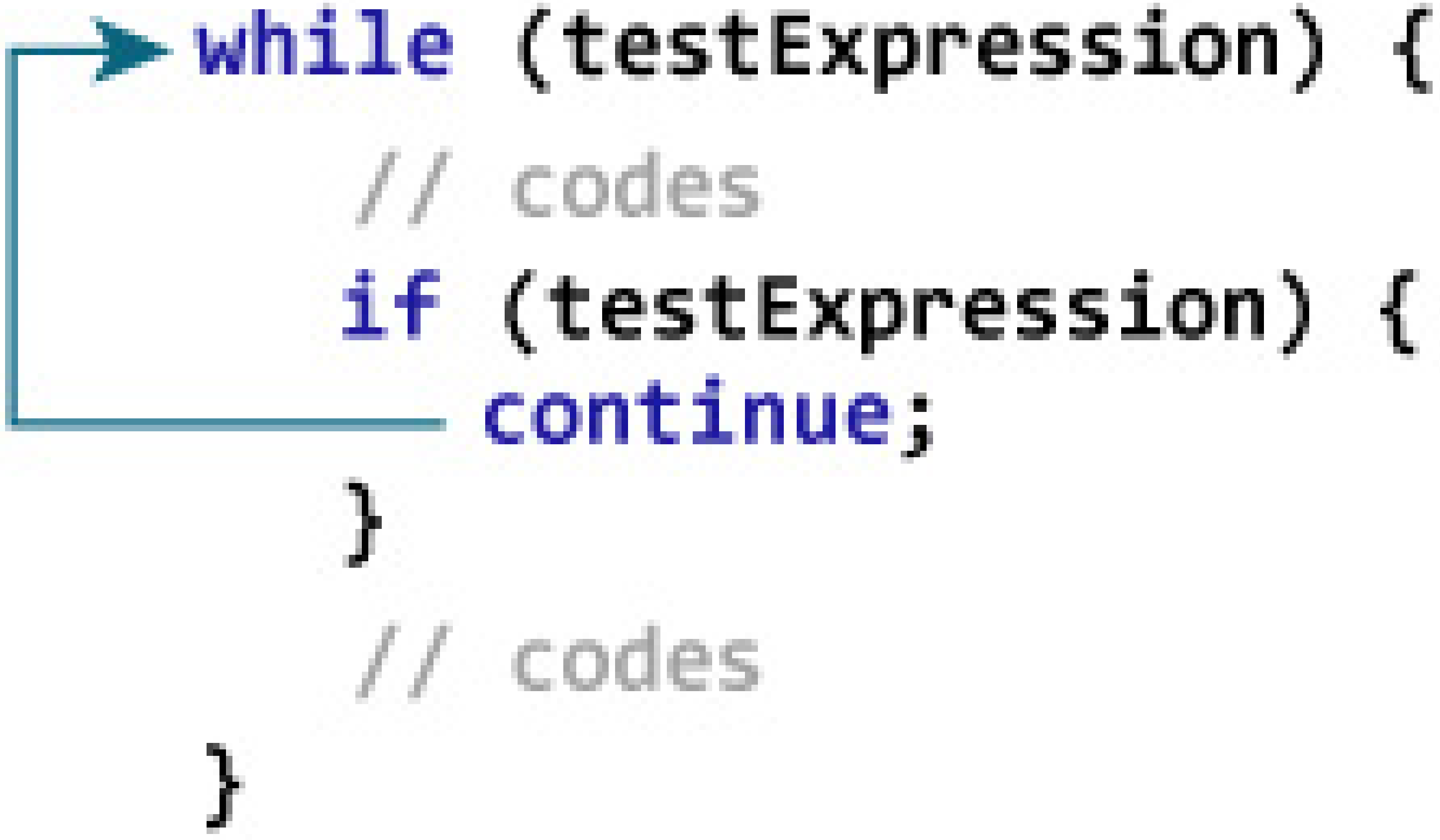


```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

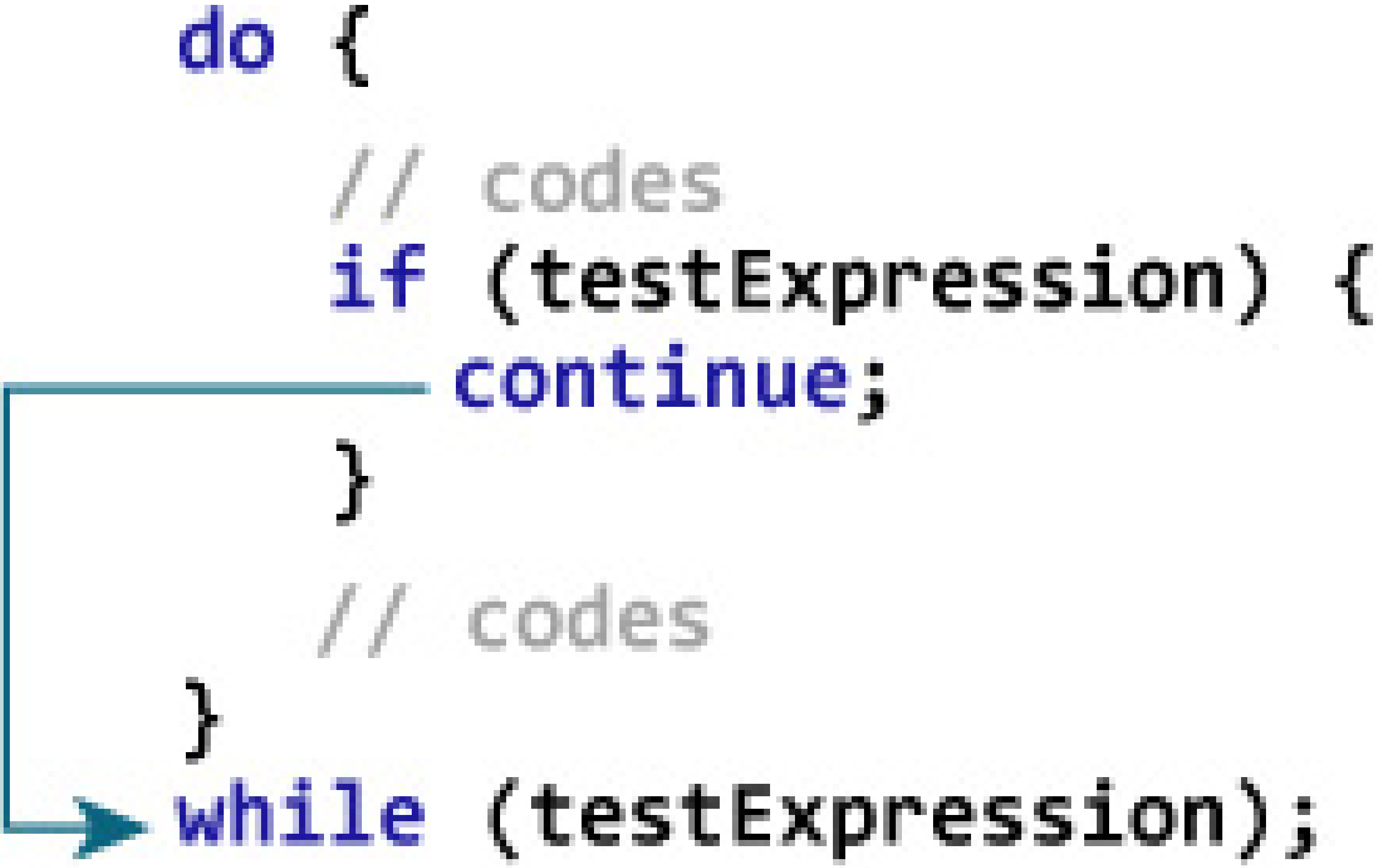
C **Continue** statement

The continue statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

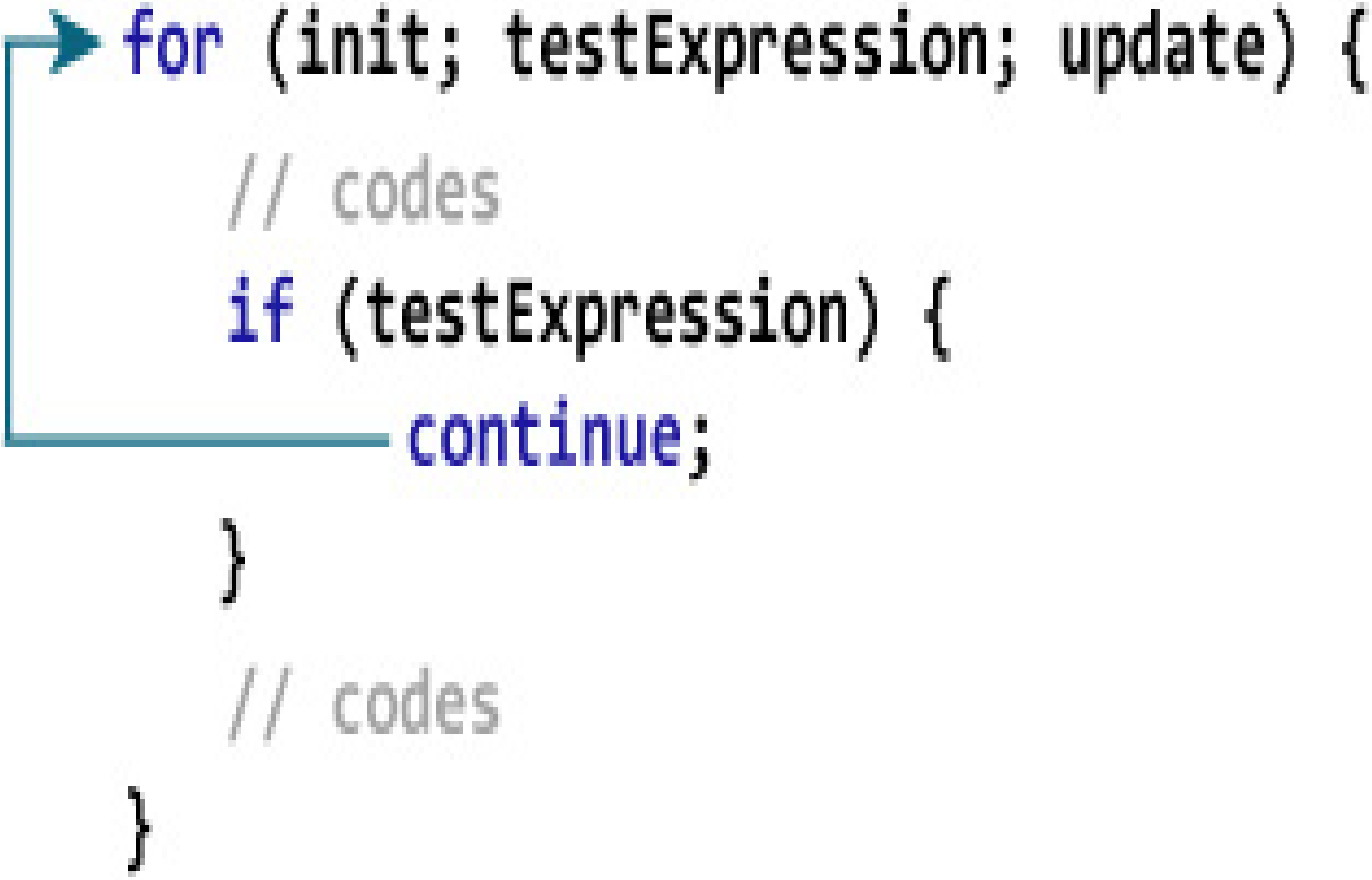
```
continue;
```



```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```



```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```



```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Write a program

To calculate the sum of a maximum of 10 numbers

Negative numbers are skipped from the calculation

```c
# 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(number < 0.0)
{
continue;
}
sum += number; // sum = sum + number;
}
printf("Sum = %.2lf",sum);
return 0;
}
```

```
Enter a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

Examples

```
#include <stdio.h>
int main() {
int num = 5;
while (num > 0) {
  if (num == 3)
    break;
  printf("%d\n", num);
  num--;
}
}
```

Output ?

Examples

5

4

Examples

```
#include <stdio.h>
int main() {
int nb = 7;
while (nb > 0) {
  nb--;
  if (nb == 5)
    continue;
 printf("%d\n", nb);
}
}
```

Output ?

Output

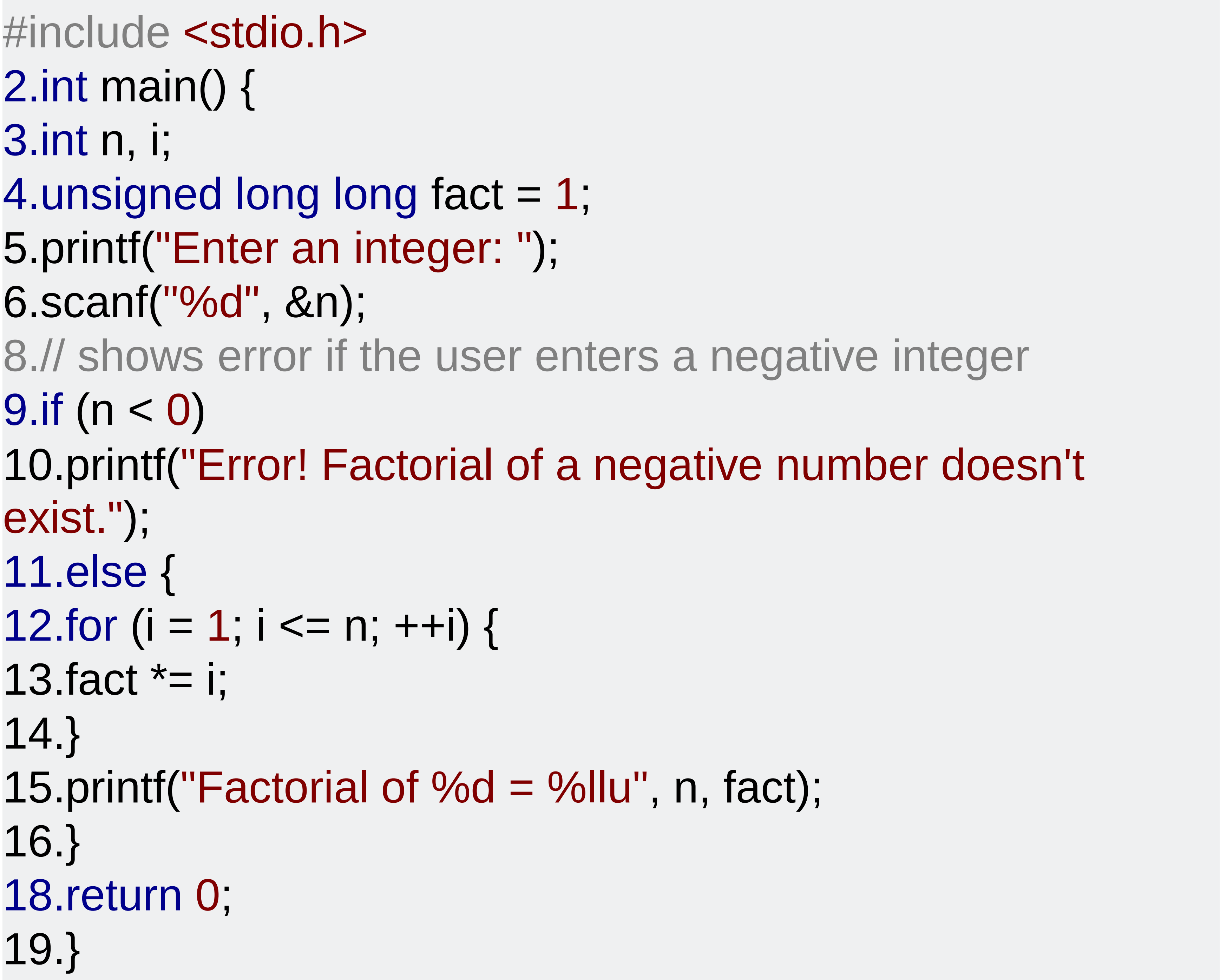
```
6
4
3
2
1
0
```

Some examples

The factorial of a positive number n is given by:

factorial of n (n!) = 1 * 2 * 3 * 4....n

Write a program to derive the factorial of a number -



```
#include <stdio.h>
int main() {
int n, i;
unsigned long long fact = 1;
printf("Enter an integer: ");
scanf("%d", &n);
// shows error if the user enters a negative integer
if (n < 0)
printf("Error! Factorial of a negative number doesn't exist.");
else {
for (i = 1; i <= n; ++i) {
fact *= i;
}
printf("Factorial of %d = %llu", n, fact);
}
return 0;
}
```

```
Enter an integer: 10
Factorial of 10 = 3628800
```

Some notes

- Since the factorial of a number may be very large, the type of factorial variable is declared as

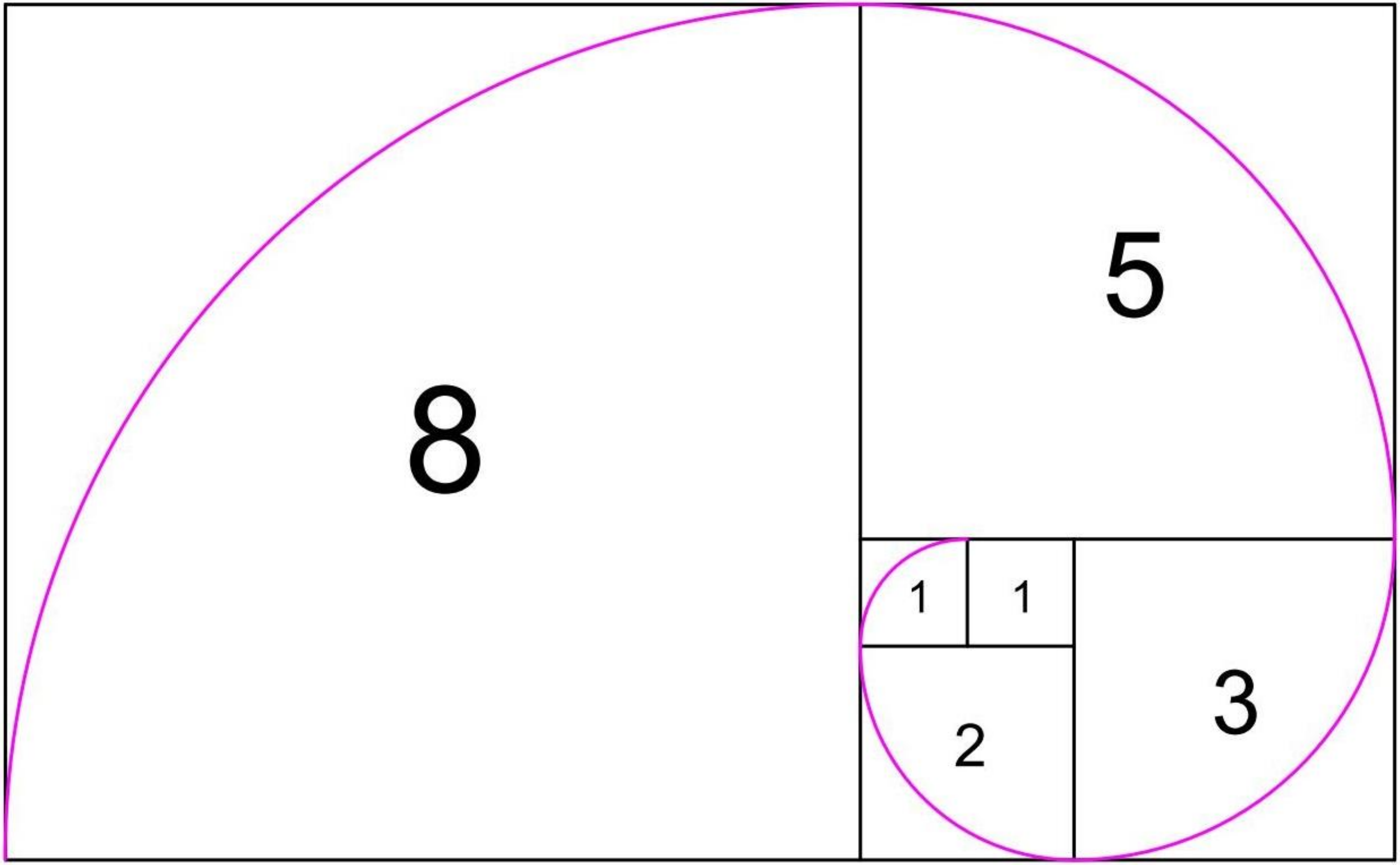
  unsigned long long.

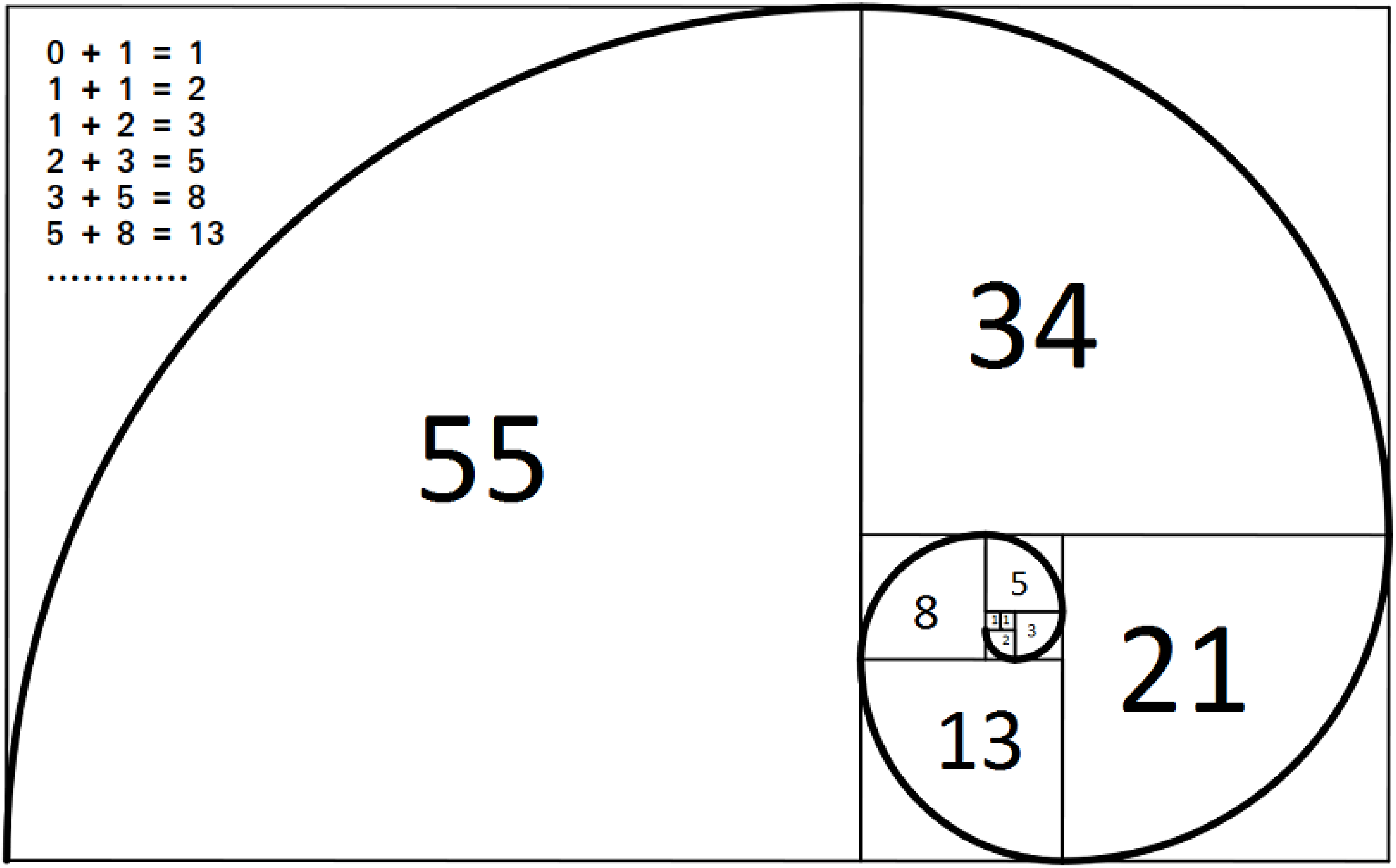
- If the user enters a negative number, the program displays a custom error message.

The Fibonacci sequence is a sequence where the next term is the sum of the previous two terms. The first two terms of the Fibonacci sequence are 0 followed by 1.

```
The Fibonacci
sequence: 0, 1, 1,
2, 3, 5, 8, 13, 21
```

Fibonacci Series up to n terms


8
5
1
1
2
3



0 + 1 = 1
1 + 1 = 2
1 + 2 = 3
2 + 3 = 5
3 + 5 = 8
5 + 8 = 13
....
55
34
21
13
8
5
3
2
1
1

```c
#include <stdio.h>
int main() {
int i, n, t1 = 0, t2 = 1, nextTerm;
printf("Enter the number of terms: ");
scanf("%d", &n);
printf("Fibonacci Series: ");
for (i = 1; i <= n; ++i) {
printf("%d, ", t1);
nextTerm = t1 + t2;
t1 = t2;
t2 = nextTerm;
}
return 0;
}
```

```
Enter a positive integer: 100
Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13,
21, 34, 55, 89,
```

Nested loops

```
for ( initialization; condition; increment ) {

  for ( initialization; condition; increment ) {

    // statement of inside loop
  }

  // statement of outer loop
}
```

Nested loops

```
while(condition) {

  while(condition) {

    // statement of inside loop
  }

  // statement of outer loop
}
```

```
do{

   do{

      // statement of inside loop
   }while(condition);

   // statement of outer loop
}while(condition);
```

There is no rule that a loop must be nested inside its own type.

In fact, there can be any type of loop nested inside -

Any type and to Any level.

```
do{

   while(condition) {

      for ( initialization; condition; increment ) {

         // statement of inside for loop
      }

      // statement of inside while loop
   }

   // statement of outer do-while loop
}while(condition);
```

Some examples to demonstrate the use of Nested Loops

A prime number is a positive integer that is divisible only by 1 and itself.

For example: 2, 3, 5, 7, 11, 13, 17

WAP - Program to Check Prime Number

```
#include <stdio.h>
int main() {
    int n, i, flag = 0;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    for (i = 2; i <= n / 2; ++i) {
        // condition for non-prime
        if (n % i == 0) {
            flag = 1;
            break;
        }
    }
if (n == 1) {
        printf("1 is neither prime nor composite.");
    }
else {
        if (flag == 0)
            printf("%d is a prime number.", n);
        else
            printf("%d is not a prime number.", n);
    }
    return 0;
}
```

```
Enter a positive integer: 29
29 is a prime number.
```

In the program, a for loop is iterated from i = 2 to i < n/2.

In each iteration, whether n is perfectly divisible by i is checked using:

In the program, a for loop is iterated from i = 2 to i < n/2.

In each iteration, whether n is perfectly divisible by i is checked using:

```
if (n % i == 0) {

}
```

If n is perfectly divisible by i, n is not a prime number.

In this case, flag is set to 1, and the loop is terminated using the break statement.

After the loop, if n is a prime number, flag will still be 0.

However, if n is a non-prime number, flag will be 1.

Using a nested for loop find the prime numbers from 2 to 100

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

   /* local variable definition */
   int i, j;

   for(i = 2; i<100; i++) {

      for(j = 2; j <= (i/j); j++)
      if(!(i%j)) break; // if factor found, not prime
      if(j > (i/j)) printf("%d is prime\n", i);
   }

   return 0;
}
```

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

```
#include <stdio.h>
int main() {
    int n1, n2, min;
    printf("Enter two positive integers: ");
    scanf("%d %d", &n1, &n2);
    // maximum number between n1 and n2 is stored in min
    min = (n1 > n2) ? n1 : n2;
    while (1) {
        if (min % n1 == 0 && min % n2 == 0) {
            printf("The LCM of %d and %d is %d.", n1, n2, min);
            break;
        }
        ++min;
    }
    return 0;
}
return 0;
}
```

Programs to print triangles using *, numbers and characters

Program to print half pyramid using *

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

```
#include<stdio.h>
int main() {
   int i, j, rows;
   printf("Enter number of rows: ");
   scanf("%d", &rows);
   for (i=1; i<=rows; ++i) {
      for (j=1; j<=i; ++j)
      { printf(“ "); }
      printf("\n");
   }
   return 0;
}
```

Programs to print triangles using *, numbers and characters

Program to print half pyramid using *

```
*                        1, 1
* **                     2, 3, 4-1
* * * * *                3, 5, 6 - 1
* * * * * * *            4, 7, 8 -1
* * * * * * * * *        5, 9, 10-1
```

Programs to print triangles using *, numbers and characters

Program to print half pyramid using *

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

```
#include<stdio.h>
int main() {
    int i,j,rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; ++i) {
        for (j=1; j<=i; ++j)
        { printf("%d ",j); }
        printf("\n");
    }
    return 0;
}
```

Program to print half pyramid using alphabets, user provides the last char.

```
A
B B
C C C
D D D D
E E E E E
```

```
#include<stdio.h>
int main() {
   int i, j;
   char input, alphabet='A';
   printf("Enter the uppercase character you want to print
   in last row: ");
   scanf("%c", &input);
   for (i=1; i<=(input-'A'+1); ++i) {
      for (j=1; j<=i; ++j)
      { printf("%c", alphabet); }
      ++alphabet;
      printf("\n");
   }
   return 0;
}
```

Programs to print inverted half pyramid using * and numbers

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

```
#include<stdio.h>
int main() {
    int i, j, rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=rows; i>=1; --i) {
        for (j=1; j<=i; ++j)
        { printf("* "); }
        printf("\n");
    }
    return 0;
}
```

Inverted half pyramid using numbers

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```c
#include<stdio.h>
int main() {
    int i ,j, rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=rows; i>=1; --i) {
        for (j=1; j<=i; ++j)
        { printf("%d ",j); }
        printf("\n");
    }
    return 0;
}
```

Programs to display pyramid and inverted pyramid using * and digits

```
        *            5, 1 (i,j)
      * * *          4, 3
    * * * * *         3, 5
  * * * * * * *       2, 7
* * * * * * * * *     1, 9
```

```
#include<stdio.h>
int main() {
    int i, space, rows, k=0;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; ++i,k=0) {

        for (space=1; space<=rows-i; ++space)
        { printf("  "); }

        while (k!=2*i-1) {
            printf("* ");
            ++k;
        }

        printf("\n");
    }
    return 0;
}
```

Program to print pyramid using numbers

```
        1
      2 3 2
    3 4 5 4 3
  4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
```

```
#include<stdio.h>
int main() {
    int i, space, rows, k=0, count=0, count1=0;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; ++i) {
        for (space=1; space<=rows-i; ++space) {
          printf("  ");
          ++count;
        }
```

```
while (k!=2*i-1) {
      if (count <= rows-1)
      { printf("%d ", i+k);
        ++count;
      }
      else {
        ++count1;
        printf("%d ", (i+k-2*count1));
      }
      ++k;
    }
```

```
 count1=count=k=0;
        printf("\n");
    }
    return 0;
}
```

Print Pascal's triangle

```
         1
        1    1
      1    2    1
    1    3    3     1
  1   4     6    4    1
1   5    10     10    5     1
```

```
#include<stdio.h>
int main() {
    int rows, coef=1, space, i, j;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=0; i<rows; i++) {
        for (space=1; space <= rows-i; space++)
            printf("  ");
        for (j=0; j<=i; j++) {
            if (j==0 || i==0)
                coef = 1;
            else
                coef=coef*(i-j+1)/j;
            printf("%4d", coef);
        }
        printf("\n");
    }
    return 0;
}
```

Print Floyd's Triangle.

```
1
2 3
4 5 6
7 8 9 10
```

```
#include<stdio.h>
int main() {
    int rows, i, j, number= 1;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; i++) {
        for (j=1; j<=i; ++j)
        { printf("%d ", number);
          ++number;
        }
        printf("\n");
    }
    return 0;
}
```