CONTENTS

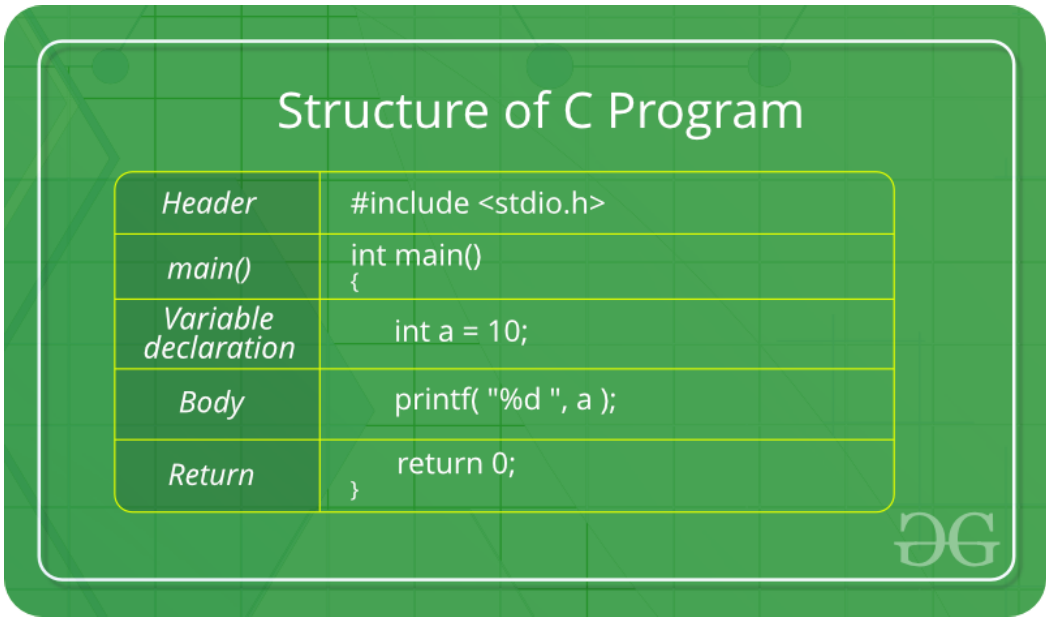
CONTENTS

SIMILARITIES & DIFFERENCES IN PROGRAMMING LANGUAGES

Metrics	C	C++	C#	☕	🐍
Designed by	Dennis Ritchie	Bjarne Stroustrup	James Gosling	Anders Hejlsberg	Guido van Rossum
Created date	1972	1998	2000	23 January 1996	20 February 1991
Total keywords	32 keywords	32 keywords	79 keywords	48 keywords	35 keywords
uses of languages	Database systems, Graphics packages, Word processors, Spread sheets, Operating system development, Compilers and Assemblers, Network drivers, Interpreters.	GUI Applications, Operating Systems, Web Browsers, Database Management System, Libraries, Cloud Computing and Distributed Applications, Job Opportunities.	Desktop applications, Mobile applications, Web applications, Web services, Web sites, Games, VR.	Mobile applications (Especially android apps), Desktop applications, Web applications, Web servers and application servers, Games, Database connection.	web development (server-side), software development, mathematics, system scripting.

C Language Introduction

STRUCTURE OF A C PROGRAM



Structure of C Program

Header	#include <stdio.h>
main()	int main() {
Variable declaration	int a = 10;
Body	printf("%d ", a);

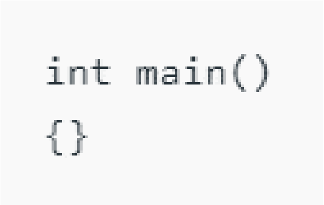
Return	return 0;
	}

1. Header Files Inclusion:

- stddef.h – Defines several useful types and macros.
- stdint.h – Defines exact width integer types.
- stdio.h – Defines core input and output functions
- stdlib.h – Defines numeric conversion functions, pseudo-random network generator, memory allocation
- string.h – Defines string handling functions
- math.h – Defines common mathematical functions

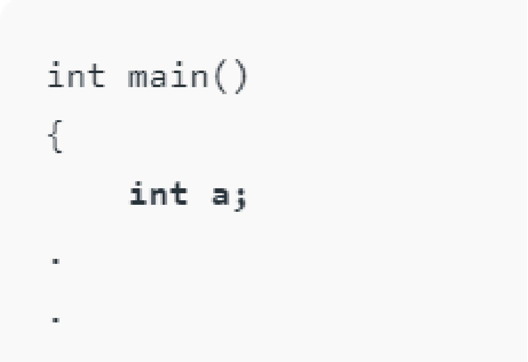
2. Syntax to include a header file in C: #include

3. Main Method Declaration:



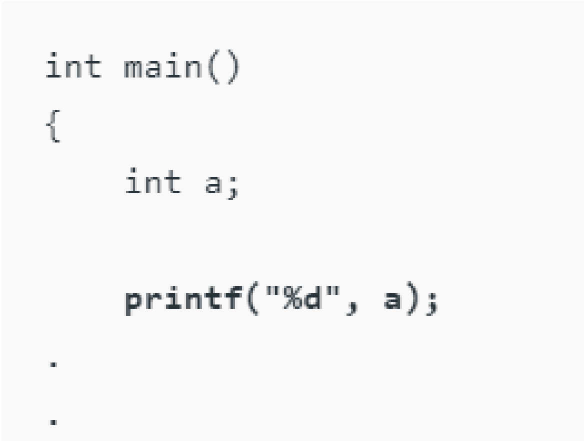
```
int main()
{}
```

4. Variable Declaration:



```
int main()
{
    int a;
.
.
```

5. Body:

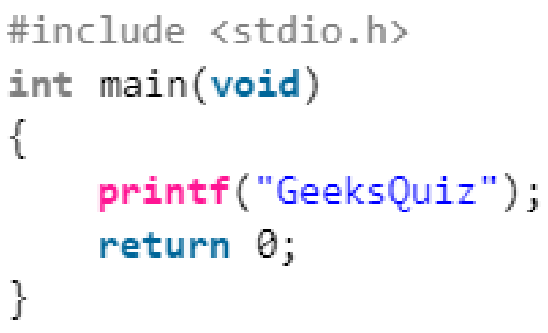


```
int main()
{
    int a;

    printf("%d", a);
.
.
```

.

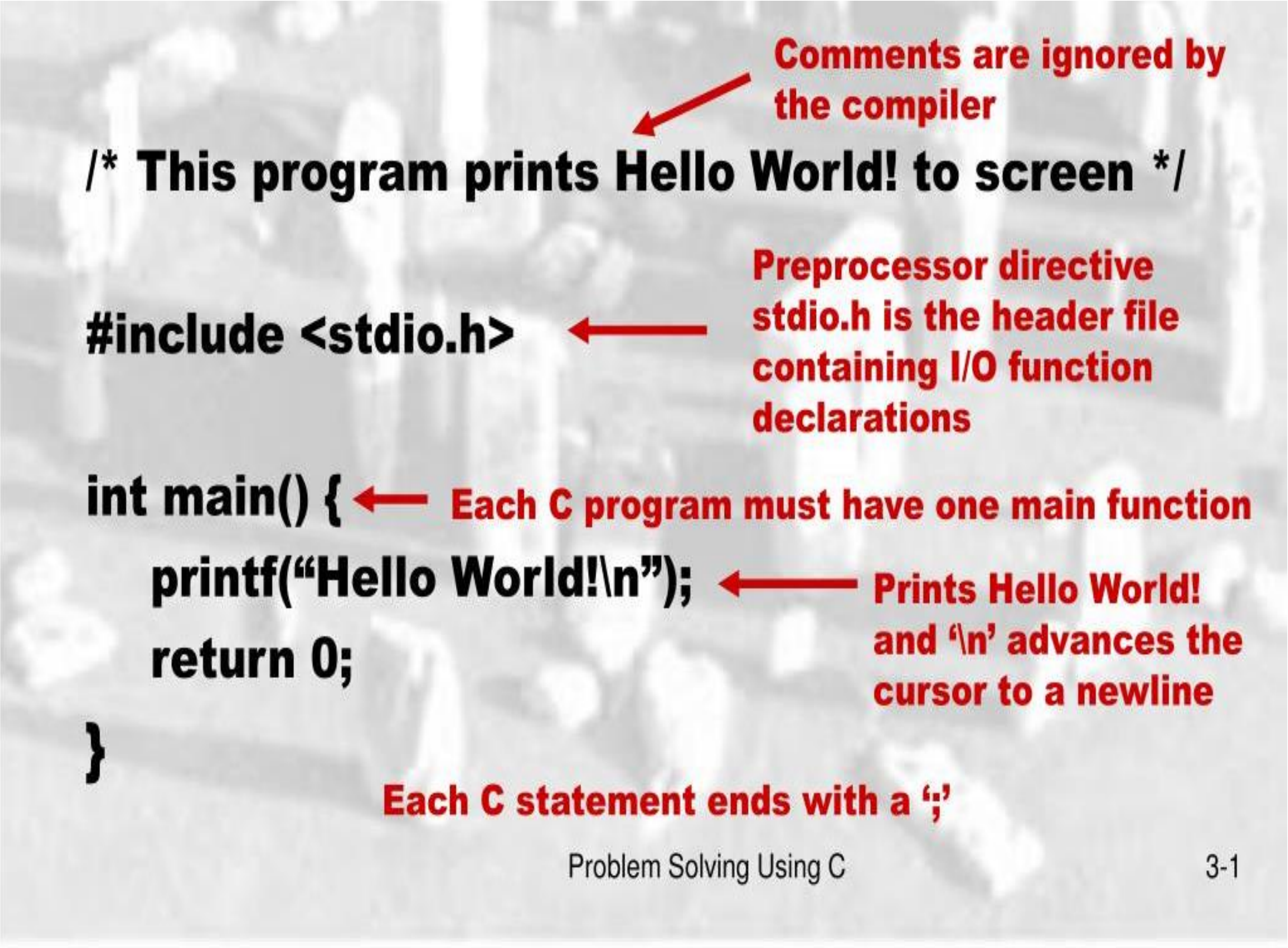
.

6. Writing first program:



```
#include <stdio.h>
int main(void)
{
    printf("GeeksQuiz");
    return 0;
}
```

HELLO WORD PROGRAM IN C



```
/* This program prints Hello World! to screen */

#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

Comments are ignored by the compiler

Preprocessor directive stdio.h is the header file containing I/O function declarations

Each C program must have one main function

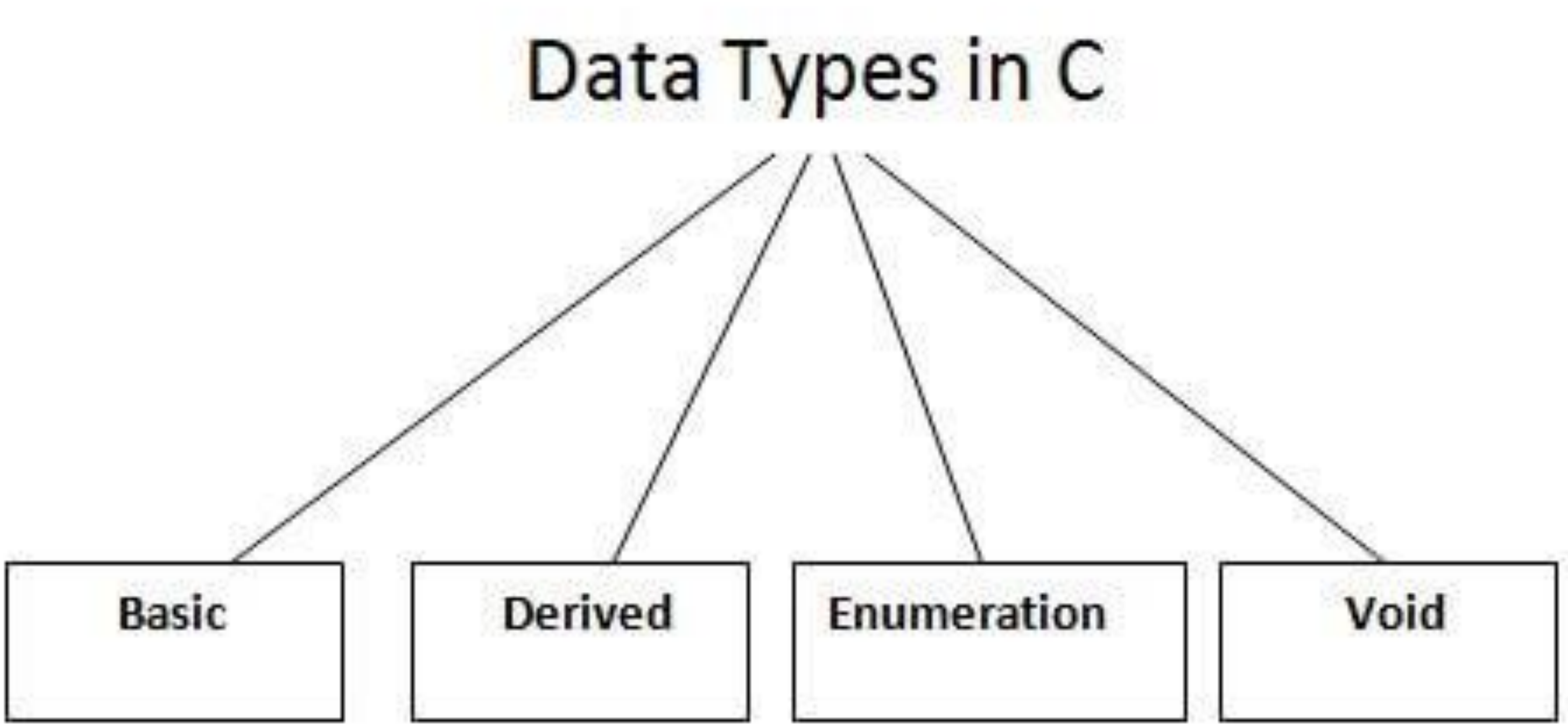
Prints Hello World! and '\n' advances the cursor to a newline

Each C statement ends with a ';'

Problem Solving Using C 3-1

Explanation: In the above code, you use header file <stdio.h> for standard input output to implement commands like printf and getch.

DATA TYPES IN C



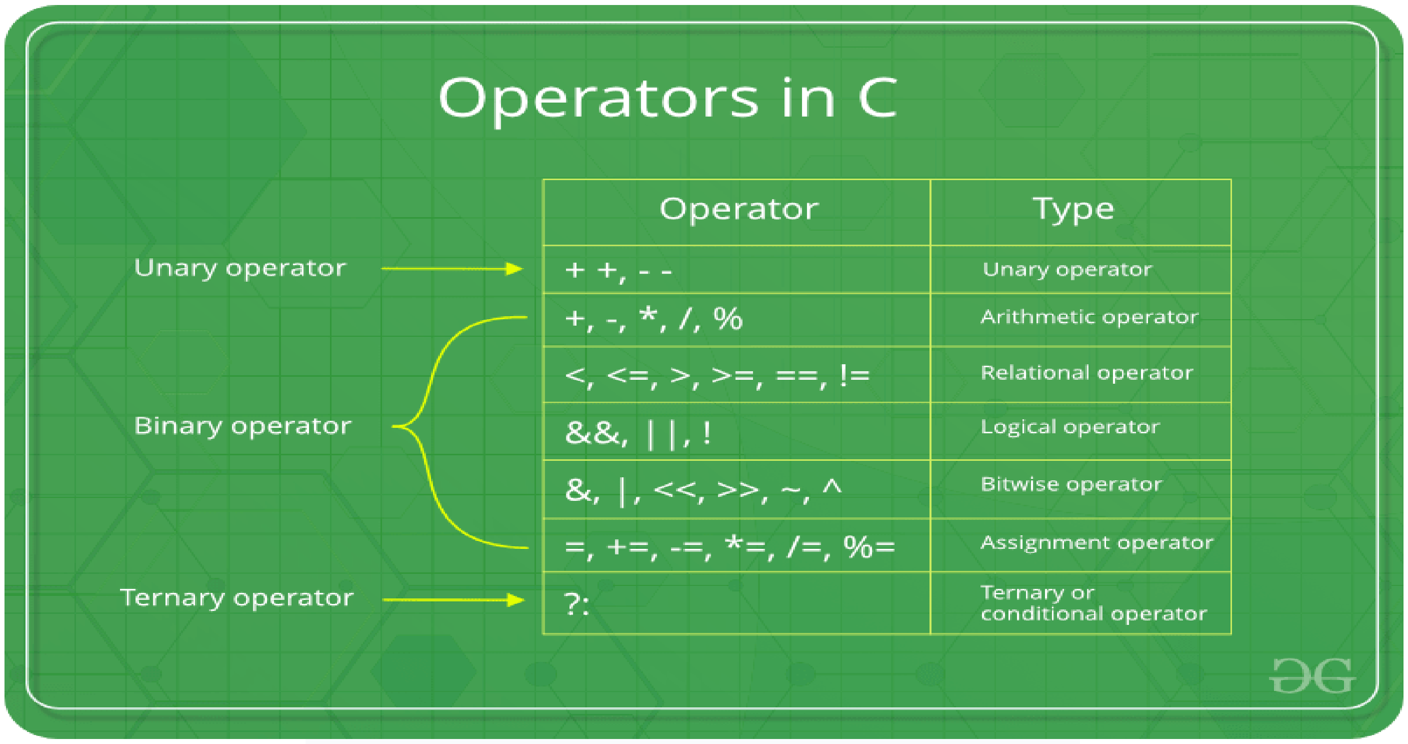
Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

KEYWORDS IN C

- **A list of 32 keywords in the c language is given below:**

Auto	Break	Case	Char	Const	Continue	Default	Do
Double	Else	Enum	Extern	Float	For	Goto	If
Int	Long	Register	Return	Short	Signed	Sizeof	Static
Struct	Switch	Typedef	Union	Unsigned	Void	Volatile	While

C PROGRAMMING OPERATORS



Operators in C

Operator	Type
+ +, - -	Unary operator
+, -, *, /, %	Arithmetic operator
<, <=, >, >=, ==, !=	Relational operator
&&, | |, !	Logical operator
&, |, <<, >>, ~, ^	Bitwise operator
=, +=, -=, *=, /=, %=	Assignment operator
?:	Ternary or conditional operator

Unary operator
Binary operator
Ternary operator

C ARITHMETIC OPERATORS

- An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	Multiplication
/	Division
%	remainder after division (modulo division)

- Example :

```
// Working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

Output :

```
a+b = 13

a-b = 5

a*b = 36

a/b = 2

Remainder when a divided by b=1
```

- **The operators `+`, `-` and `*` computes addition, subtraction, and multiplication respectively as you might have expected.**

- **In normal calculation, `9/4 = 2.25`. However, the output is `2` in the program.**

- **It is because both the variables `a` and b are integers. Hence, the output is also an integer. The compiler neglects the term after the decimal point and shows answer `2` instead of `2.25`.**

- **The modulo operator `%` computes the remainder. When `a=9` is divided by `b=4`, the remainder is `1`. The `%` operator can only be used with integers.**

- **Suppose `a = 5.0`, `b = 2.0`, `c = 5` and `d = 2`. Then in C programming,**

```
// Either one of the operands is a floating-point number
a/b = 2.5
a/d = 2.5
c/b = 2.5


// Both operands are integers
c/d = 2
```

C INCREMENT AND DECREMENT OPERATORS

- **C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.**

- **Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.**

- Example :

```
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
   int a = 10, b = 100;
   float c = 10.5, d = 100.5;

   printf("++a = %d \n", ++a);
   printf("--b = %d \n", --b);
   printf("++c = %f \n", ++c);
   printf("--d = %f \n", --d);

   return 0;
}
```

Output :

```
++a = 11
--b = 99
++c = 11.500000
--d = 99.500000
```

- **Here, the operators ++ and -- are used as prefixes. These two operators can also be used as postfixes like a++ and a--. Visit this page to learn more about how increment and decrement operators work when used as postfix.**

C ASSIGNMENT OPERATORS

- An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

- Example :

```
// Working of assignment operators
#include <stdio.h>
int main()
{
   int a = 5, c;

   c = a;      // c is 5
   printf("c = %d\n", c);
   c += a;     // c is 10
   printf("c = %d\n", c);
   c -= a;     // c is 5
   printf("c = %d\n", c);
   c *= a;     // c is 25
   printf("c = %d\n", c);
   c /= a;     // c is 5
   printf("c = %d\n", c);
   c %= a;     // c = 0
   printf("c = %d\n", c);

   return 0;
}
```

Output :

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

C RELATIONAL OPERATORS

- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

- Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

➢ Example :

```
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

Output :

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

C LOGICAL OPERATORS

- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

➢ Example :

```
// Working of logical operators

#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10, result;

   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) is %d \n", result);
```

```
    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);

    result = !(a != b);
    printf("!(a != b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```

Output :

```
(a == b) && (c > b) is 1
(a == b) && (c < b) is 0
(a == b) || (c < b) is 1
(a != b) || (c < b) is 0
!(a != b) is 1
!(a == b) is 0
```

Explanation of logical operator program

- (a == b) && (c > 5) evaluates to 1 because both operands (a == b) and (c > b) is 1 (true).

- (a == b) && (c < b) evaluates to 0 because operand (c < b) is 0 (false).

- (a == b) || (c < b) evaluates to 1 because (a = b) is 1 (true).

- (a != b) || (c < b) evaluates to 0 because both operand (a != b) and (c < b) are 0 (false).

- !(a != b) evaluates to 1 because operand (a != b) is 0 (false). Hence, !(a != b) is 1 (true).

- !(a == b) evaluates to 0 because (a == b) is 1 (true). Hence, !(a == b) is 0 (false).

C BITWISE OPERATORS

- During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

- Bitwise operators are used in C programming to perform bit-level operations.

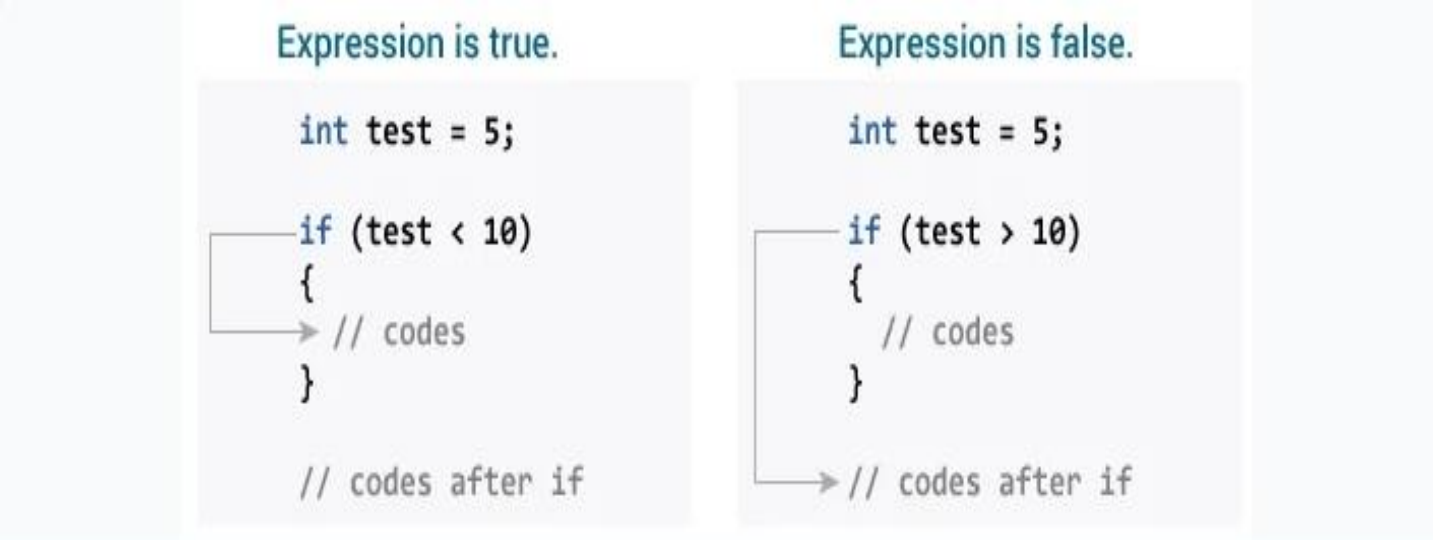
Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

C IF STATEMENT

- The syntax of the if statement in C programming is:

```
if (test expression)
{
   // code
}
```

- How if statement works?
- 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.
- If the test expression is evaluated to false, statements inside the body of if are not executed.



Expression is true.

```
int test = 5;

if (test < 10)
{
   // codes
}

// codes after if
```

Expression is false.

```
int test = 5;

if (test > 10)
{
   // codes
}

// codes after if
```

Working of if Statement

➢ Example :

```
// Program to 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 %d.\n", number);
    }

    printf("The if statement is easy.");

    return 0;
}
```

Output 1 :

```
Enter an integer: -2
You entered -2.
The if statement is easy.
```

- When the user enters -2, the test expression number<0 is evaluated to true. Hence, You entered -2 is displayed on the screen.

Output 2 :

```
Enter an integer: 5
The if statement is easy.
```

- When the user enters 5, the test expression number<0 is evaluated to false and the statement inside the body of if is not executed

C IF...ELSE STATEMENT

- The if statement may have an optional else block. The syntax of the if..else statement is:

```
if (test expression) {
   // run code if test expression is true
}
else {
   // run code if test expression is false
}
```

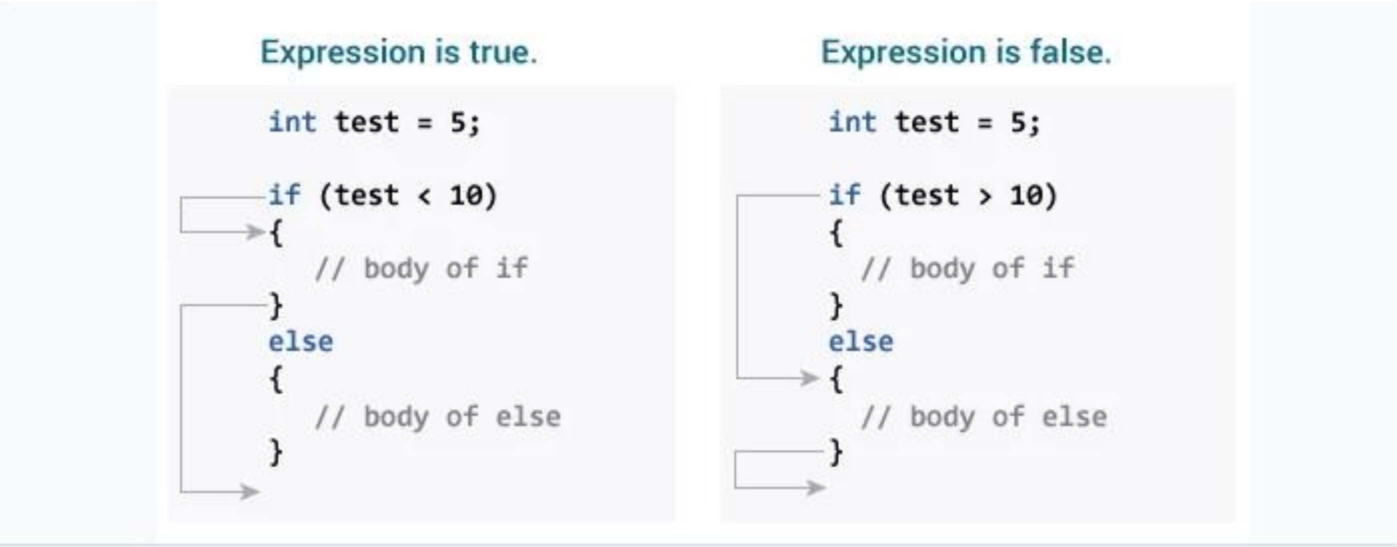
- How if...else statement works?

If the test expression is evaluated to true,

- statements inside the body of if are executed.
- statements inside the body of else are skipped from execution.

If the test expression is evaluated to false,

- statements inside the body of else are executed
- statements inside the body of if are skipped from execution.



Expression is true.

```
int test = 5;

if (test < 10)
{
   // body of if
}
else
{
   // body of else
}
```

Expression is false.

```
int test = 5;

if (test > 10)
{
   // body of if
}
else
{
   // body of else
}
```

Working of if...else Statement

- Example :

```
// Check whether an integer is odd or even

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

    // 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;
}
```

Output :

```
Enter an integer: 7
7 is an odd integer.
```

- When the user enters 7, the test expression `number%2==0` is evaluated to false. Hence, the statement inside the body of `else` is executed.

C IF...ELSE LADDER

- The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.

- The if...else ladder allows you to check between multiple test expressions and execute different statements.

- Syntax of if...else Ladder

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

- Example :

```c
// Program to relate two integers using =, > or < symbol

#include <stdio.h>
int main() {
   int number1, number2;
   printf("Enter two integers: ");
   scanf("%d %d", &number1, &number2);

   //checks if the two integers are equal.
   if(number1 == number2) {
      printf("Result: %d = %d",number1,number2);
   }

   //checks if number1 is greater than number2.
   else if (number1 > number2) {
      printf("Result: %d > %d", number1, number2);
   }

   //checks if both test expressions are false
   else {
      printf("Result: %d < %d",number1, number2);
   }

   return 0;
}
```

Output :

```
Enter two integers: 12
23
Result: 12 < 23
```

C NESTED IF...ELSE

- It is possible to include an if...else statement inside the body of another if...else statement.

- This program given below relates two integers using either <, > and = similar to the if...else ladder's example. However, we will use a nested if...else statement to solve this problem.

➢ Example :

```
#include <stdio.h>
int main() {
  int number1, number2;
  printf("Enter two integers: ");
  scanf("%d %d", &number1, &number2);

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

  return 0;
}
```

LOOP IN C

- C programming has three types of loops:

1. for loop
2. while loop
3. do...while loop

C FOR LOOP

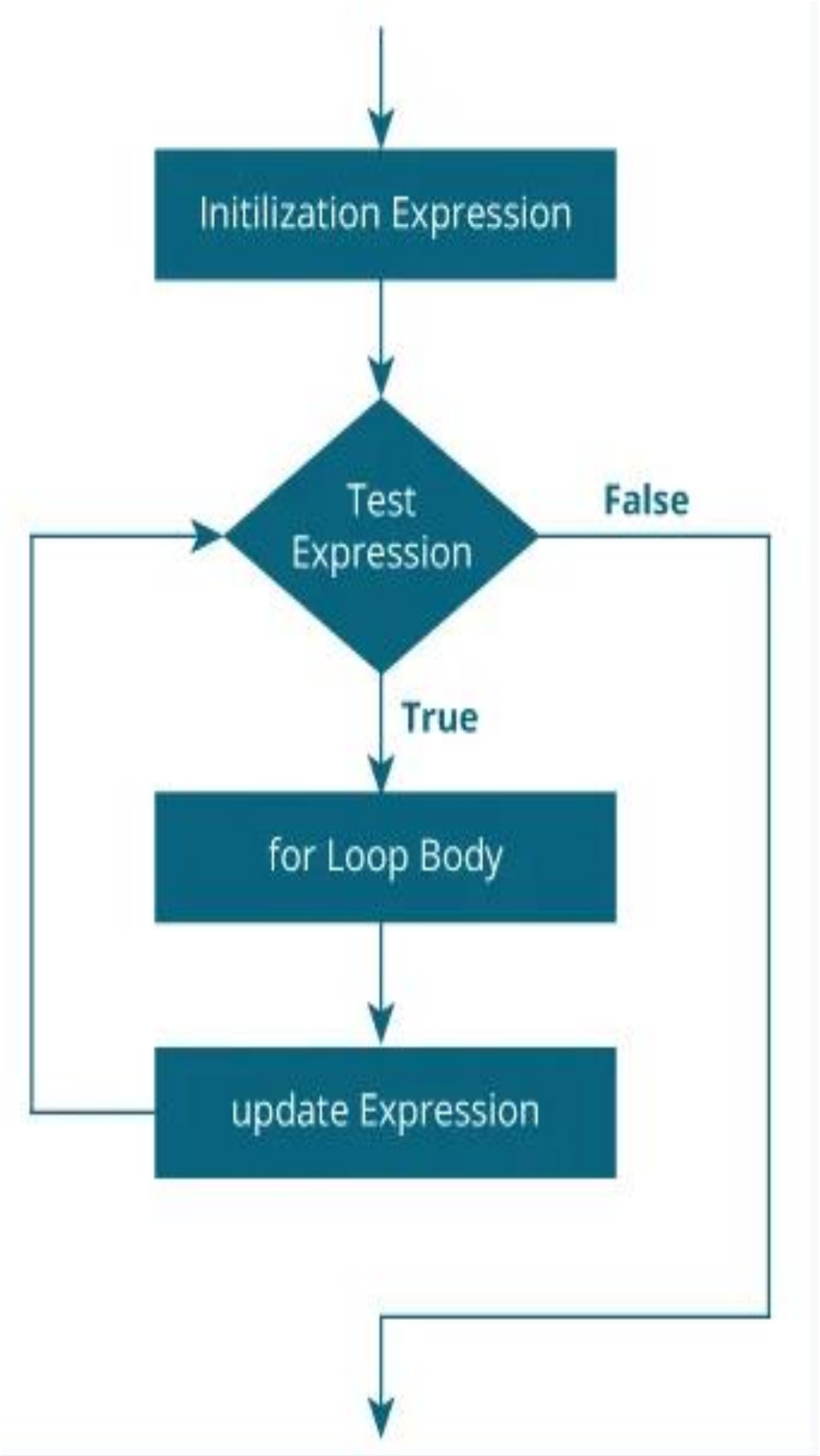
- The syntax of the for loop is:

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

- How for loop works?
- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.

- This process goes on until the test expression is false. When the test expression is false, the loop terminates.

C FOR LOOP FLOWCHART



Working of for loop

- Example :

```
// Print numbers from 1 to 10
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
  return 0;
}
```

Output :

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

1. `i` is initialized to 1.
2. The test expression `i < 11` is evaluated. Since 1 less than 11 is true, the body of `for` loop is executed. This will print the 1 (value of `i`) on the screen.
3. The update statement `++i` is executed. Now, the value of `i` will be 2. Again, the test expression is evaluated to true, and the body of for loop is executed. This will print 2 (value of `i`) on the screen.
4. Again, the update statement `++i` is executed and the test expression `i < 11` is evaluated. This process goes on until `i` becomes 11.
5. When `i` becomes 11, `i < 11` will be false, and the `for` loop terminates.

C WHILE LOOP

- The syntax of the while loop is:

```
while (testExpression)
{
   // the body of the loop
}
```

- How for loop works?

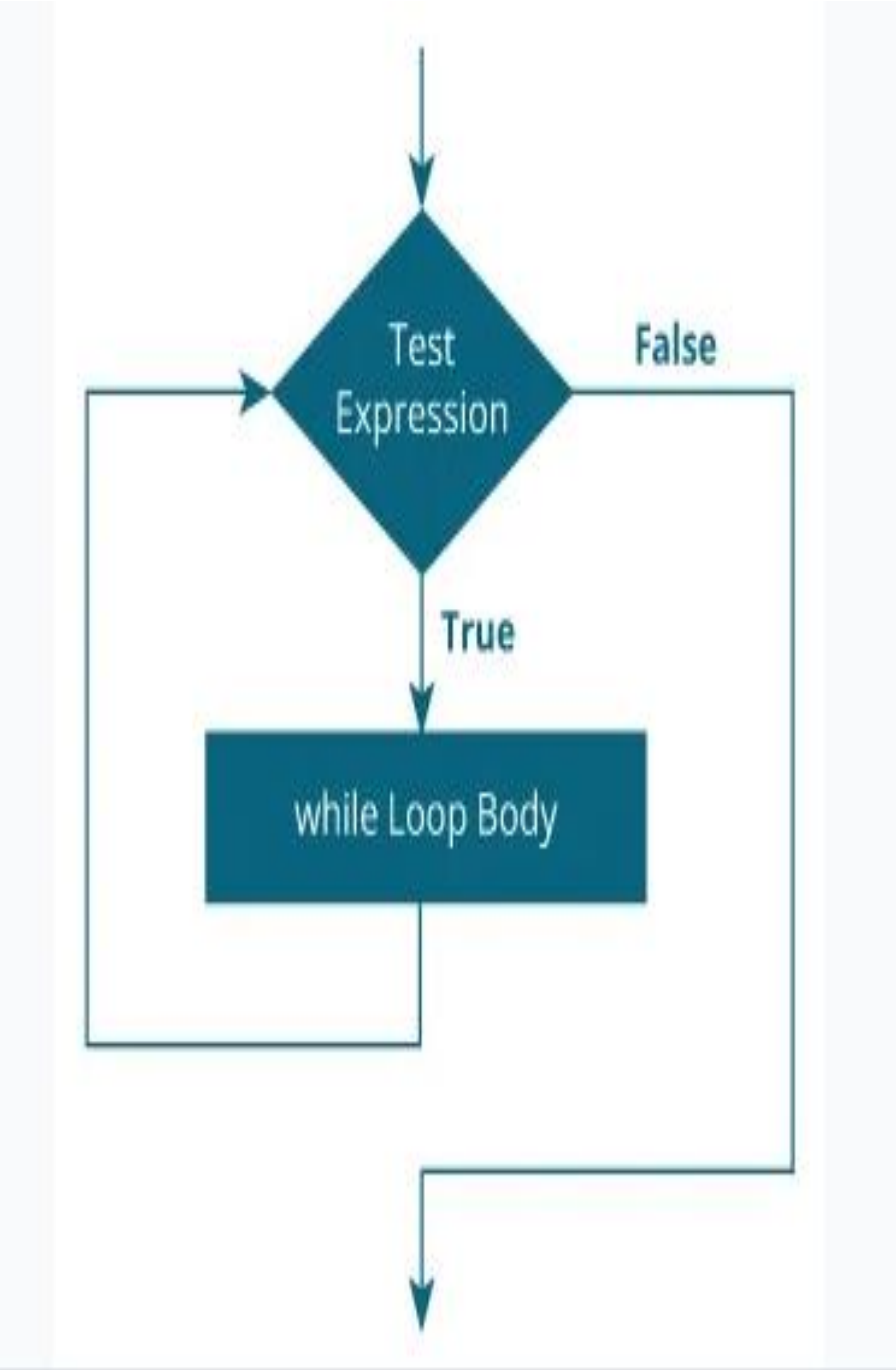
- The while loop evaluates the test expression inside the parenthesis ().

- If the test expression is true, statements inside the body of while loop are executed. Then, the test expression is evaluated again.

- The process goes on until the test expression is evaluated to false.

- If the test expression is false, the loop terminates (ends).

C WHILE LOOP FLOWCHART



Working of while loop

- Example :

```
// Print numbers from 1 to 5

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

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

   return 0;
}
```

Output :

```
1
2
3
4
5
```

Here, we have initialized i to 1.

1. When i is 1, the test expression i <= 5 is true. Hence, the body of the while loop is executed. This prints 1 on the screen and the value of i is increased to 2.

2. Now, i is 2, the test expression i <= 5 is again true. The body of the while loop is executed again. This prints 2 on the screen and the value of i is increased to 3.

3. This process goes on until i becomes 6. When i is 6, the test expression i <= 5 will be false and the loop terminates.

C DO...WHILE LOOP

- The do...while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

- The syntax of the do...while loop is:

```
do
{
  // the body of the loop
}
while (testExpression);
```

- How do...while loop works?

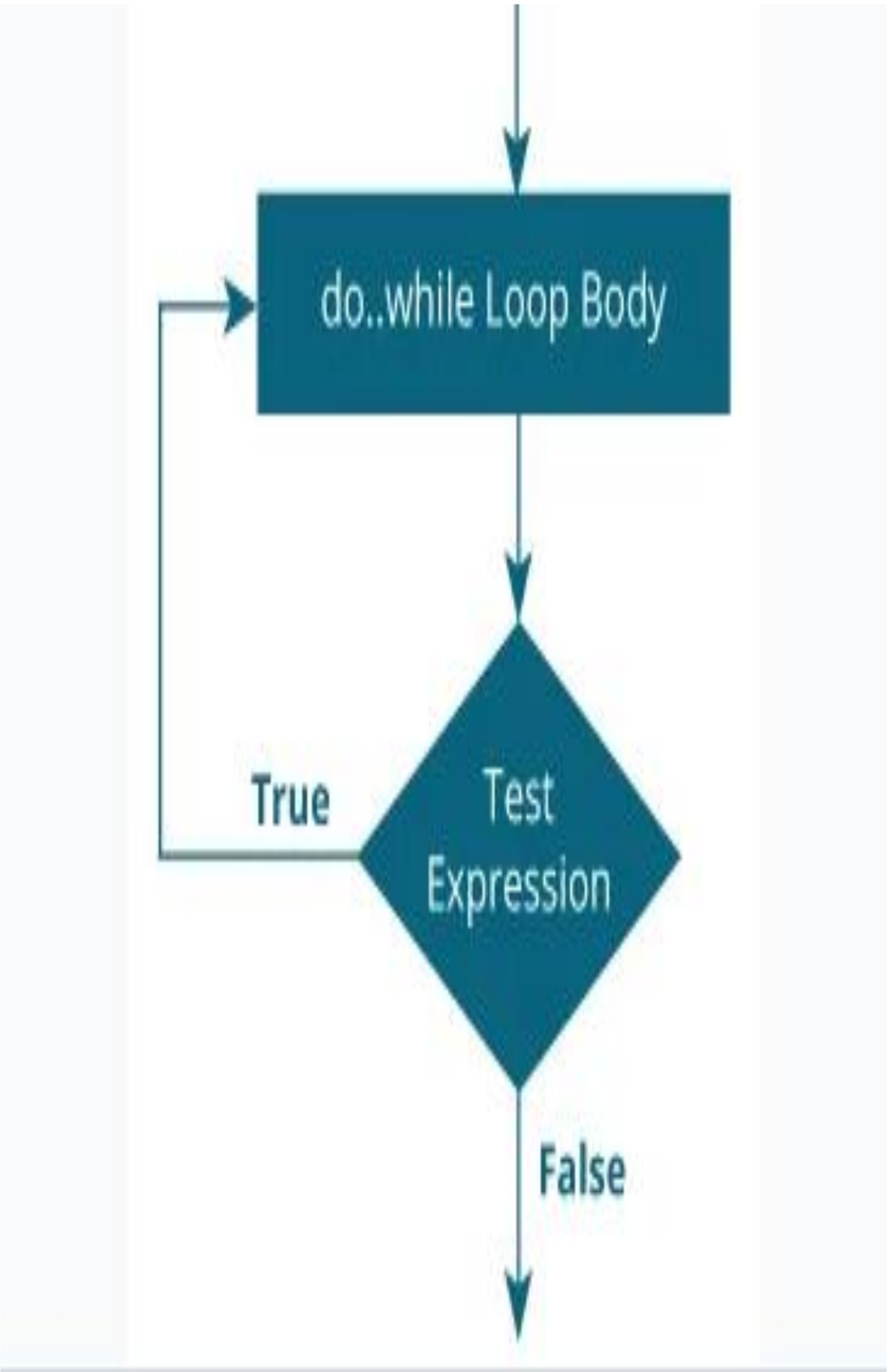
- The body of do...while loop is executed once. Only then, the test expression is evaluated.

- If the test expression is true, the body of the loop is executed again and the test expression is evaluated.

- This process goes on until the test expression becomes false.

- If the test expression is false, the loop ends.

C DO…WHILE LOOP FLOWCHART



Working of do...while loop

➢ Example :

```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;
}
```

Output :

```
Enter a number: 1.5
Enter a number: 2.4
Enter a number: -3.4
Enter a number: 4.2
Enter a number: 0
Sum = 4.70
```

C Switch Statement

- 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 of the switch statement :

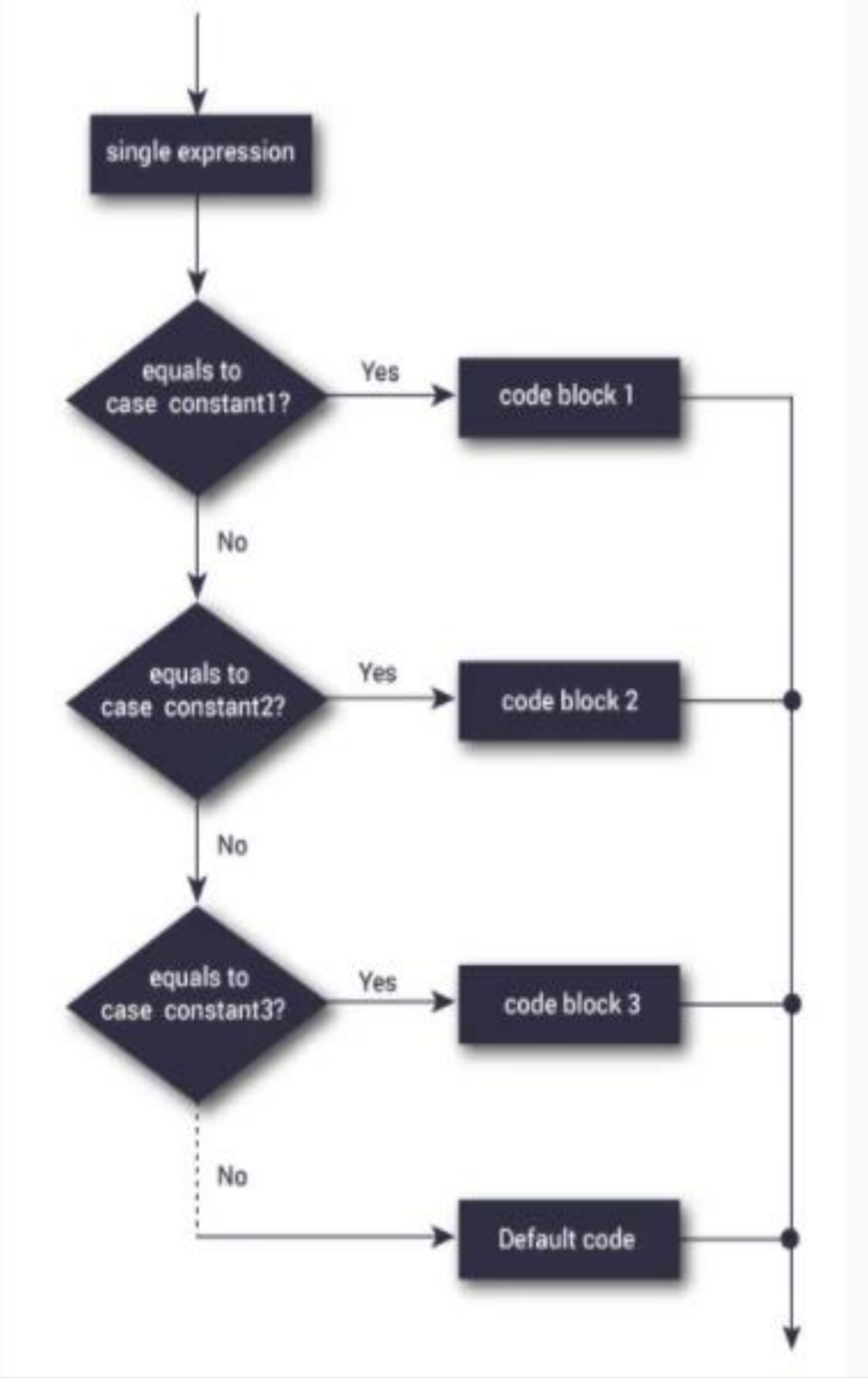
```
switch (expression)
{
    case constant1:
      // statements
      break;

    case constant2:
      // statements
      break;
    .
    .
    .
    default:
      // default statements
}
```

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. 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.

C SWITCH STATEMENT FLOWCHART



switch Statement Flowchart

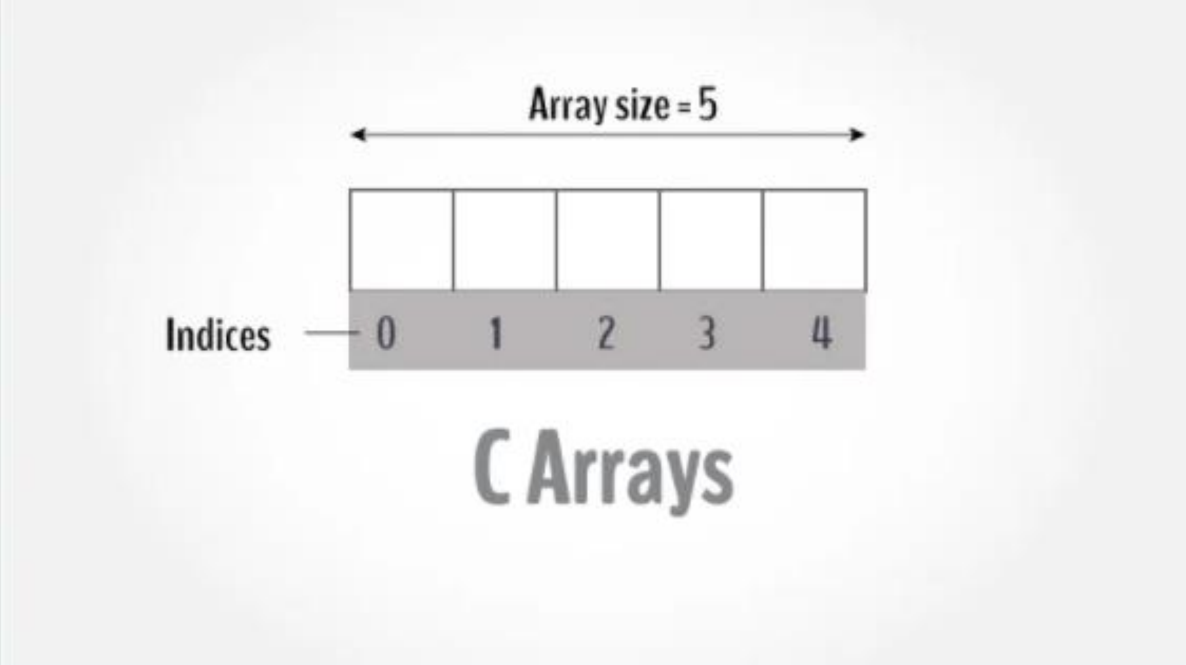
Example :

```c
// Program to create a simple calculator
#include <stdio.h>

int main() {
    char operator;
    double n1, n2;

    printf("Enter an operator (+, -, *, /): ");
    scanf("%c", &operator);
    printf("Enter two operands: ");
    scanf("%lf %lf",&n1, &n2);

    switch(operator)
    {
        case '+':
            printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
            break;

        case '-':
            printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
            break;

        case '*':
            printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
            break;

        case '/':
            printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
            break;

        // operator doesn't match any case constant +, -, *, /
        default:
            printf("Error! operator is not correct");
    }

    return 0;
}
```

Output :

```
Enter an operator (+, -, *,): -
Enter two operands: 32.5
12.4
32.5 - 12.4 = 20.1
```

ARRAYS IN C



Arrays in C

- An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it.

```
int data[100];
```

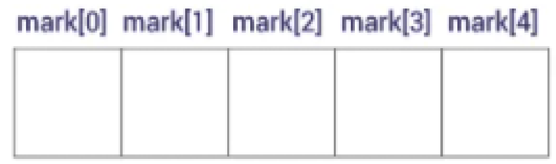
- How to declare an array?

```
dataType arrayName[arraySize];
```

- Access Array Elements

You can access elements of an array by indices.

Suppose you declared an array mark as above. The first element is mark[0], the second element is mark[1] and so on.



Declare an Array

Few keynotes:

- Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.
- If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4]
- Suppose the starting address of mark[0] is 2120d. Then, the address of the mark[1] will be 2124d. Similarly, the address of mark[2] will be 2128d and so on.
  This is because the size of a float is 4 bytes.

How to initialize an array?

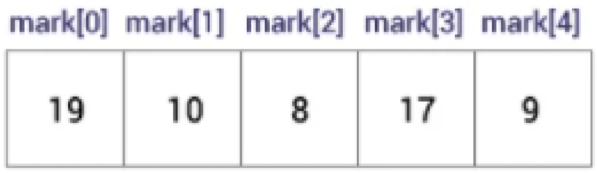
- It is possible to initialize an array during declaration. For example,

```
int mark[5] = {19, 10, 8, 17, 9};
```

- You can also initialize an array like this.

```
int mark[] = {19, 10, 8, 17, 9};
```

- Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.



Initialize an Array

Here,

mark[0] is equal to 19

mark[1] is equal to 10

mark[2] is equal to 8

mark[3] is equal to 17

mark[4] is equal to 9

Change Value of Array elements

```
int mark[5] = {19, 10, 8, 17, 9}

// make the value of the third element to -1
mark[2] = -1;

// make the value of the fifth element to 0
mark[4] = 0;
```

➢ EXAMPLE :

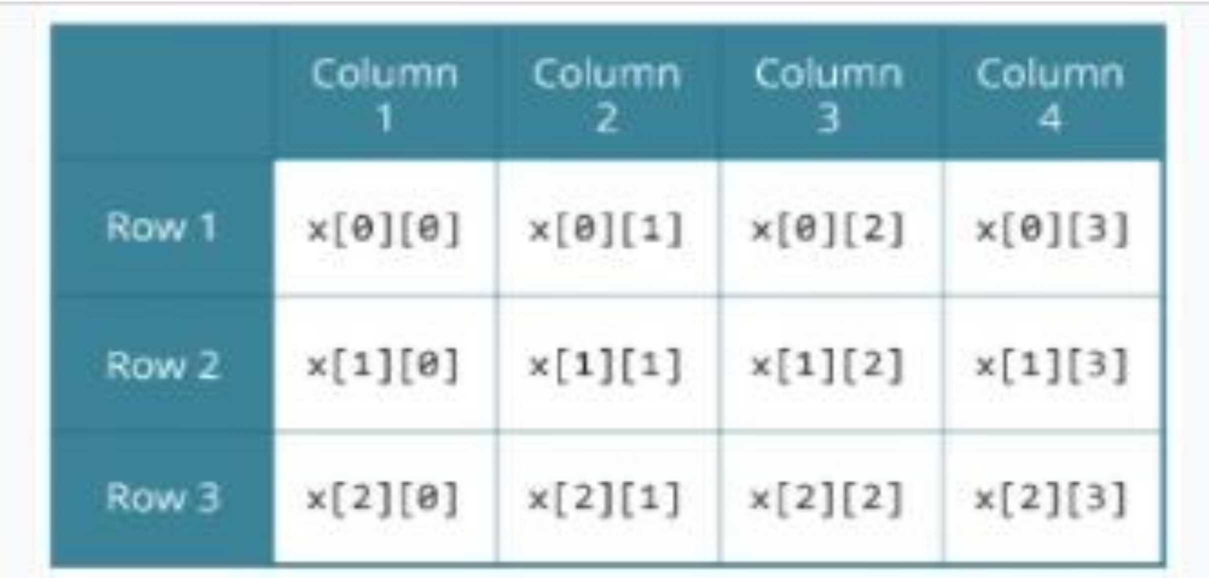
```
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>

int main() {
  int values[5];

  printf("Enter 5 integers: ");

  // taking input and storing it in an array
  for(int i = 0; i < 5; ++i) {
     scanf("%d", &values[i]);
  }

  printf("Displaying integers: ");

  // printing elements of an array
  for(int i = 0; i < 5; ++i) {
     printf("%d\n", values[i]);
  }
  return 0;
}
```

Output :

```
Enter 5 integers: 1
-3
34
0
3
Displaying integers: 1
-3
34
0
3
```

- Here, we have used a `for` loop to take 5 inputs from the user and store them in an array. Then, using another `for` loop, these elements are displayed on the screen.

MULTIDIMENSIONAL ARRAYS IN C



	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Example : Three-dimensional array

```
// C Program to store and print 12 values entered by the user

#include <stdio.h>
int main()
{
  int test[2][3][2];

  printf("Enter 12 values: \n");

  for (int i = 0; i < 2; ++i)
  {
    for (int j = 0; j < 3; ++j)
    {
      for (int k = 0; k < 2; ++k)
      {
        scanf("%d", &test[i][j][k]);
      }
    }
  }

  // Printing values with proper index.

  printf("\nDisplaying values:\n");
  for (int i = 0; i < 2; ++i)
  {
    for (int j = 0; j < 3; ++j)
    {
      for (int k = 0; k < 2; ++k)
      {
        printf("test[%d][%d][%d] = %d\n", i, j, k, test[i][j][k]);
      }
    }
  }

  return 0;
}
```

Output :

```
Enter 12 values:
1
2
3
4
5
6
7
8
9
10
11
12

Displaying Values:
test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 11
test[1][2][1] = 12
```

Initialization of a 3d array

- You can initialize a three-dimensional array in a similar way like a two-dimensional array. Here's an example,

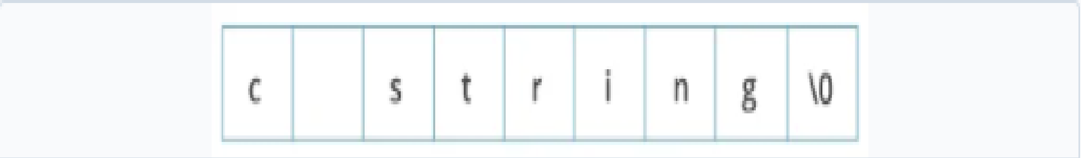
```
int test[2][3][4] = {
    {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},
    {{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```

STRINGS IN C

- In C programming, a string is a sequence of characters terminated with a null character \0. For example:

```
char c[] = "c string";
```

- When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.

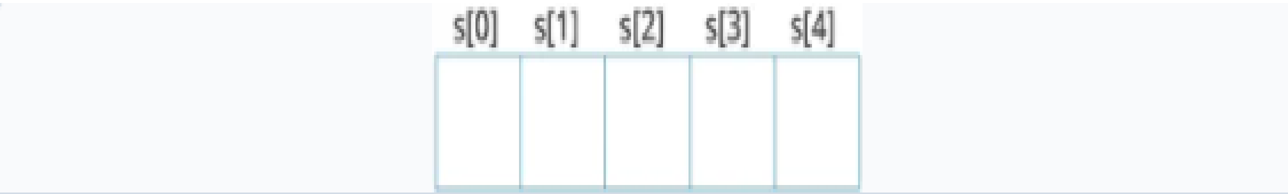


Memory Diagram

HOW TO DECLARE A STRING?

- Here's how you can declare strings:

```
char s[5];
```

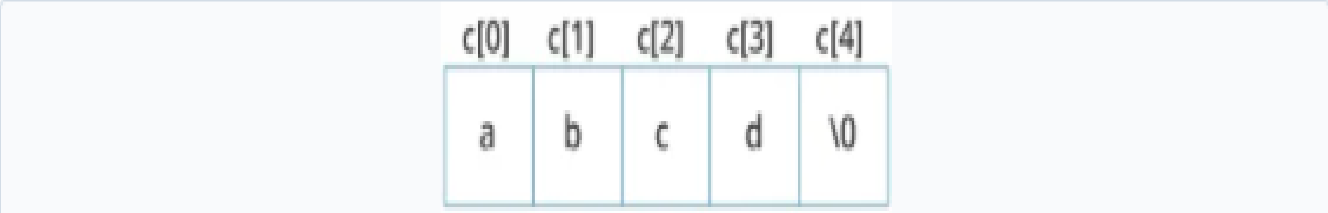


String Declaration in C

Here, we have declared a string of 5 characters.

HOW TO INITIALIZE STRINGS?

- You can initialize strings in a number of ways.



String Initialization in C

```
char c[] = "abcd";

char c[50] = "abcd";

char c[] = {'a', 'b', 'c', 'd', '\0'};

char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

- Let's take another example:

```
char c[5] = "abcde";
```

- Here, we are trying to assign 6 characters (the last character is '\0') to a char array having 5 characters. This is bad and you should never do this.