WORKSHOP

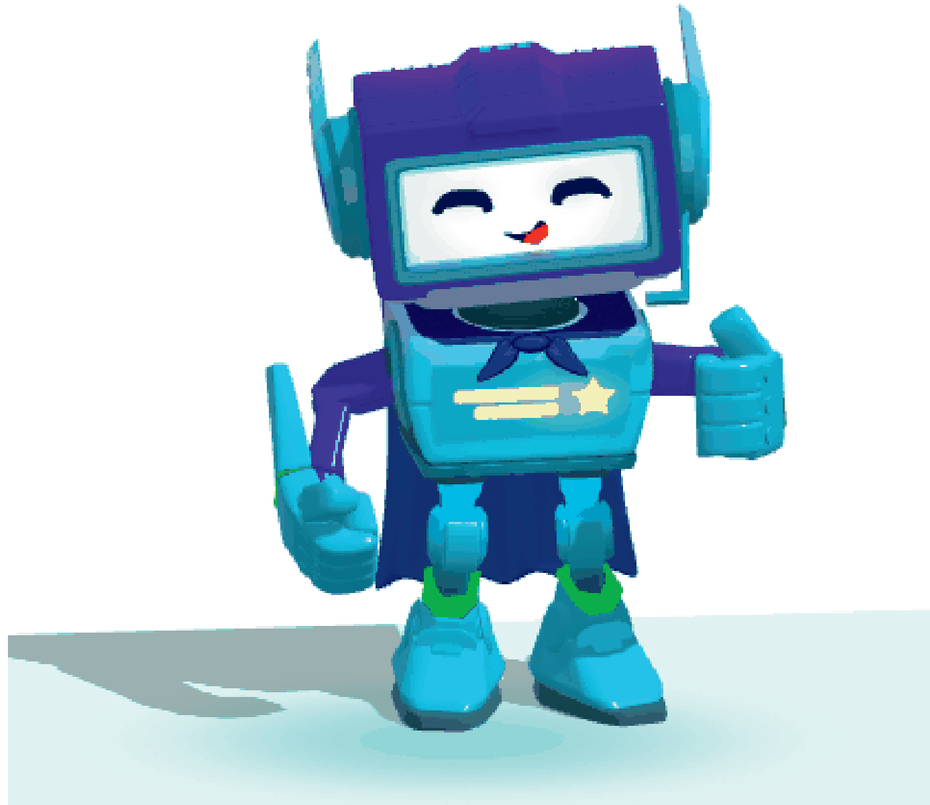
Day-2

Today's Agenda

1. Loops (While and For)
2. Break Statement
3. Continue Statement
4. Arrays (1D and 2D)
5. Strings
6. User Inputs and Memory Address
7. Pointers

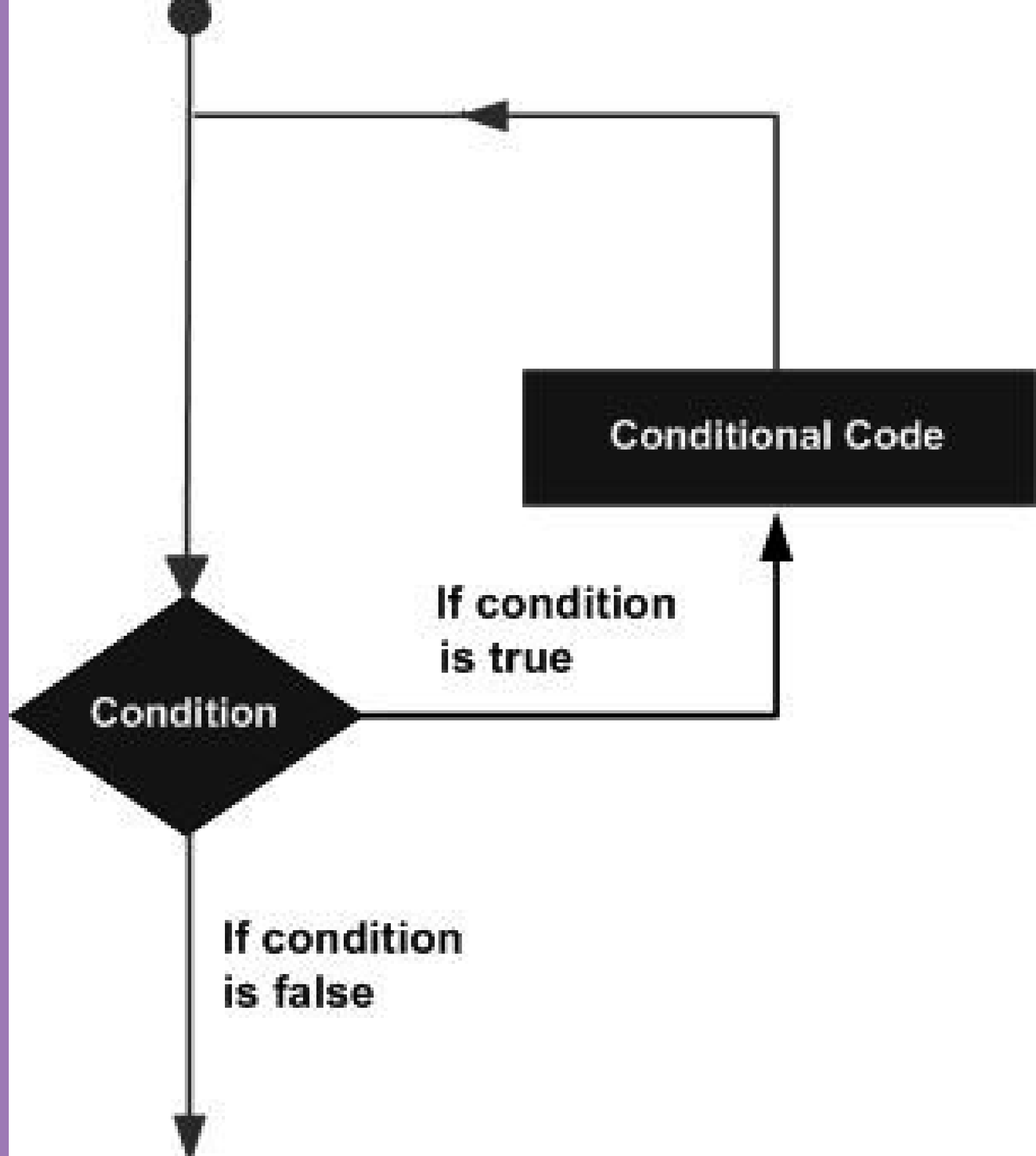
Why do you want to write programs?

What are Loops??



Loops

A loop statement allows us to execute a statement or group of statements multiple times

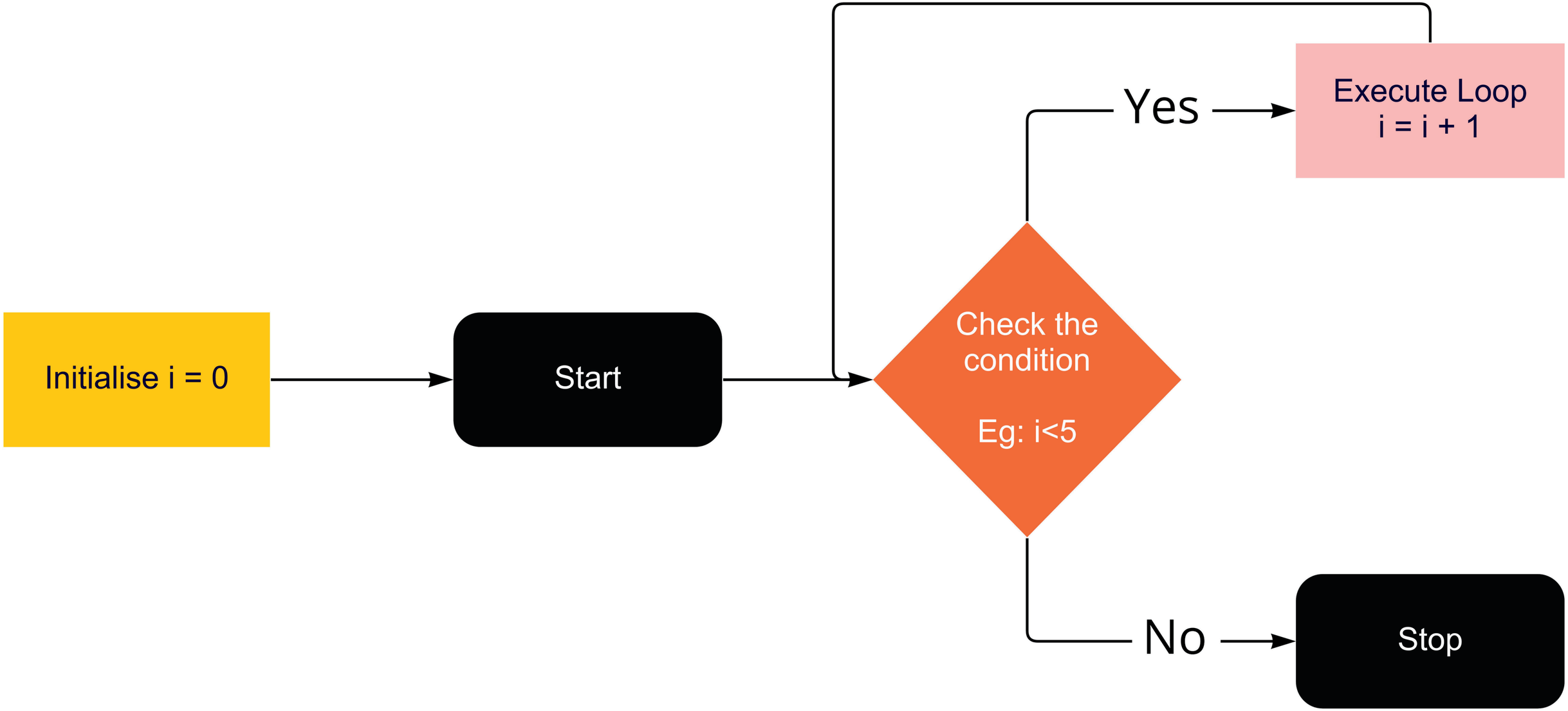


Advantages of Loops

```
printf("1")
printf("2")
printf("3")
    .
    .
    .
printf("10,000")
```

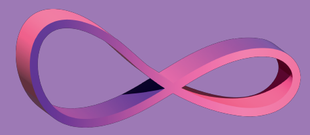
1. Code Reusability
2. Saves Time
3. Easy to Traverse

```
index=0;    (index)
Loop starts
    printf(index);
    increment index
Loop end
    Continue until
    condition fails.
```



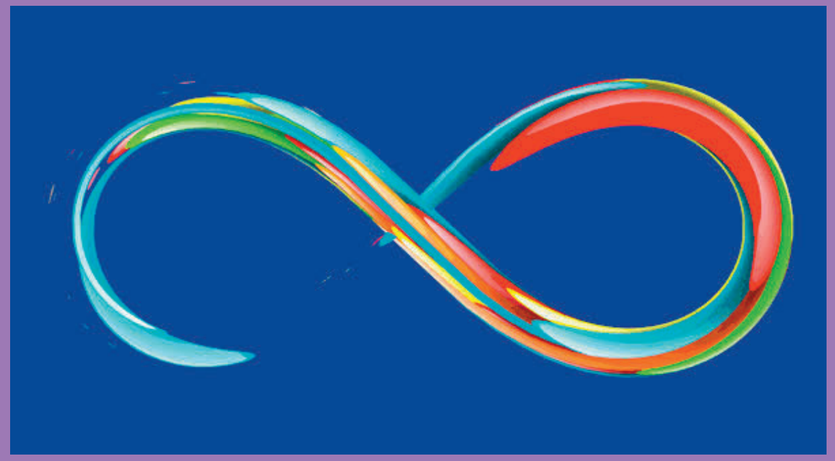
Basic Syntax of Loops

Types of Loops



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

What is a FOR loop?



The For Loop is a loop where the program tells the compiler to run a specific code FOR a specified number of times.

For Loop

This loop allows using three statements, first is the counter initialization, next is the condition to check it and then there is an increment/decrement operation to change the counter variable.

```
"i++" or "i--"
```

The last is the increment/decrement operation again separated by a semicolon.

```
i = 0;
```

If the variable is “i” then it needs to be initialized like the following.

```
"; i < 0; " or "; i >= a; "
```

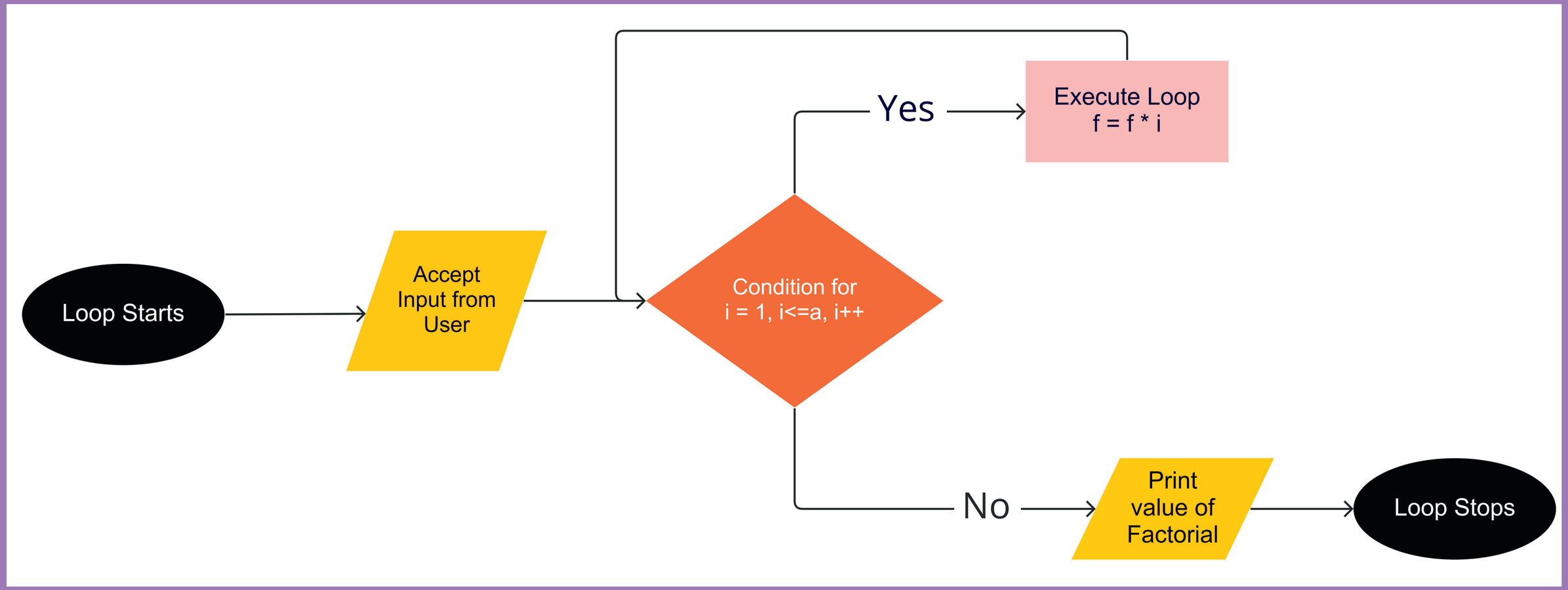
Next, there comes a conditional statement separated by a semicolon.
Example: It can be a specific value, or it can be another variable.

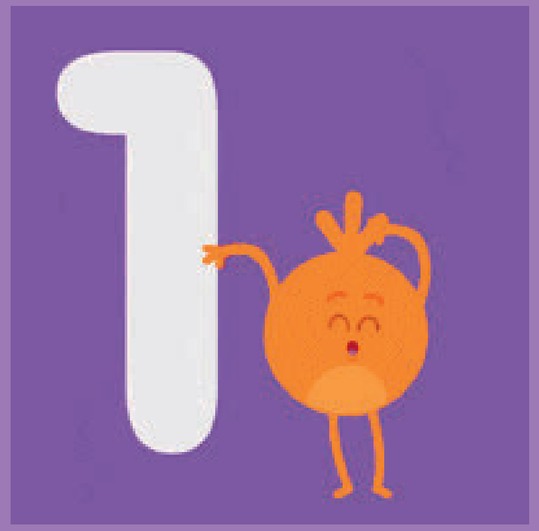
Final Sequence

```
for (i = 0; i <= 10; i++)
{

    //Statement block

}
```

Let's Code Along

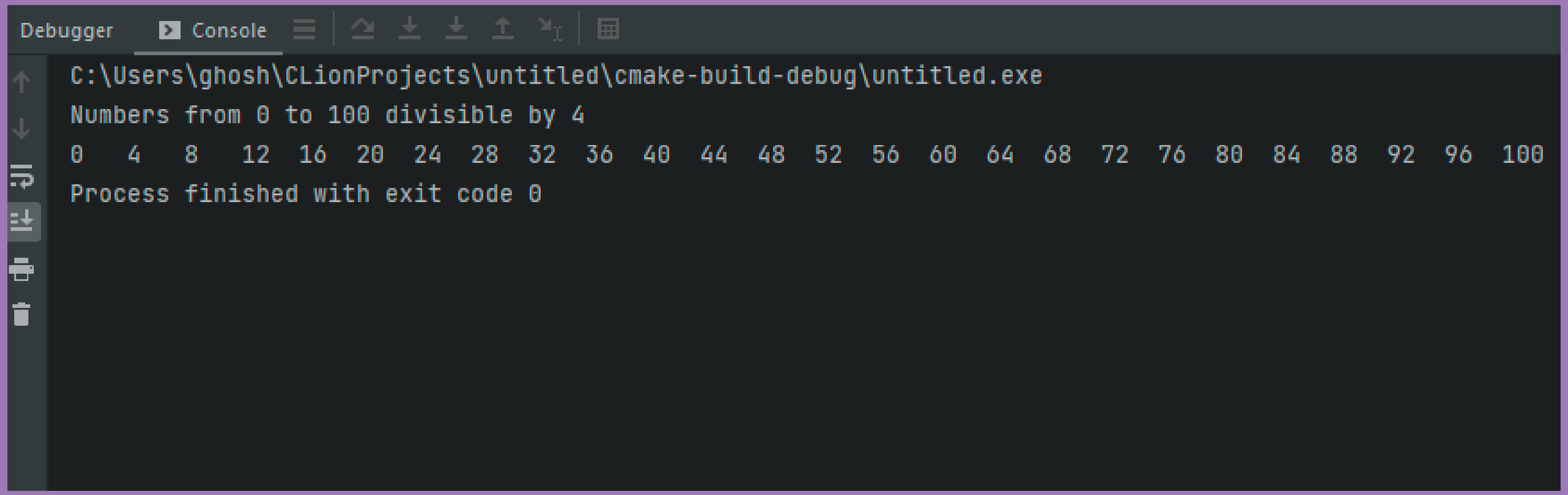
Program to find the factorial of a number





Program to find all numbers between 1 to 100 divisible by 4

TRY THIS YOURSELF


Debugger
Console
C:\Users\ghosh\CLionProjects\untitled\cmake-build-debug\untitled.exe
Numbers from 0 to 100 divisible by 4
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100
Process finished with exit code 0

Output

What is a WHILE loop?

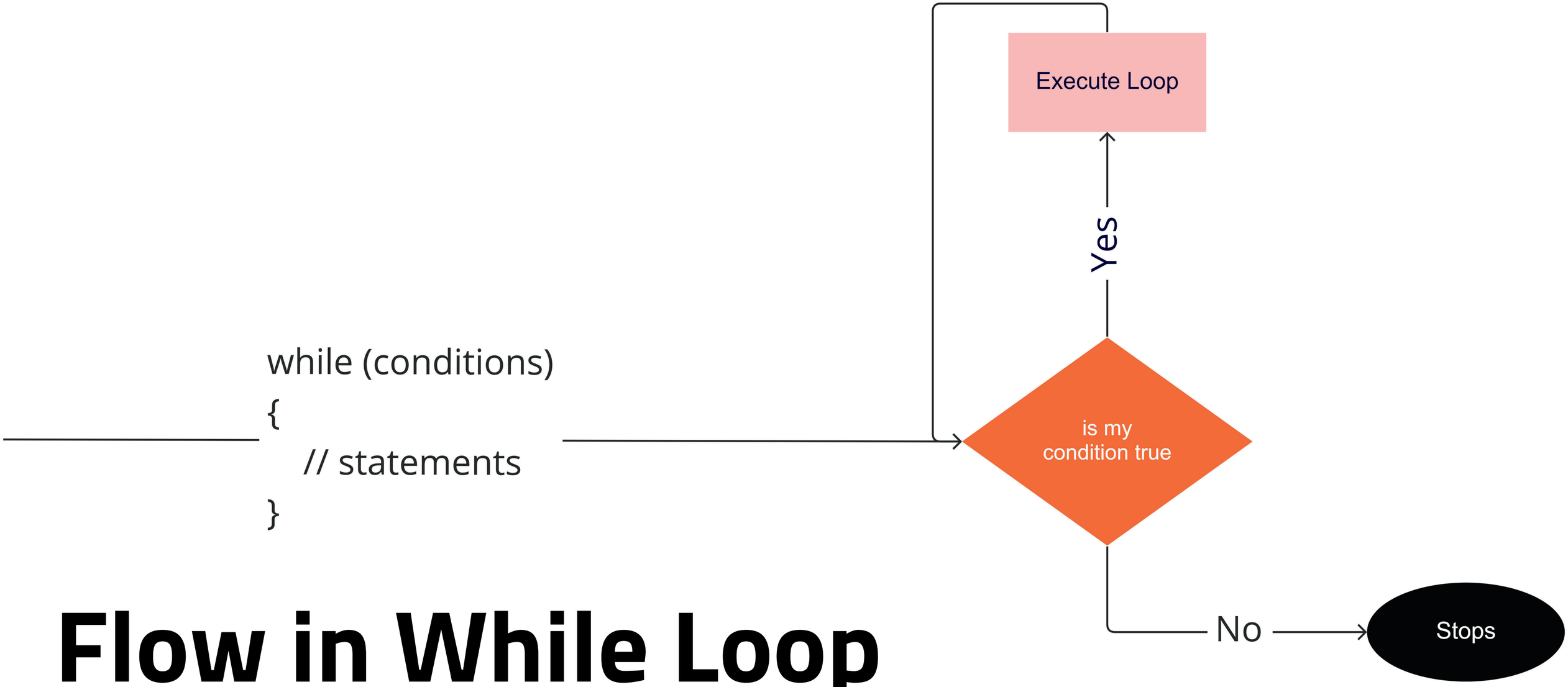
While Loops

The while loop begins by first checking the terminal condition and then it decides whether to enter the loop or not.

If the underlying condition is true, then it goes ahead and executes the block of code in the loop. After the first iteration, it again checks with the changed (increased/decreased) values of the variables (the condition operands) and decides the further course of execution.

Flow in While Loop

```
while (conditions)
{
    // statements
}
```

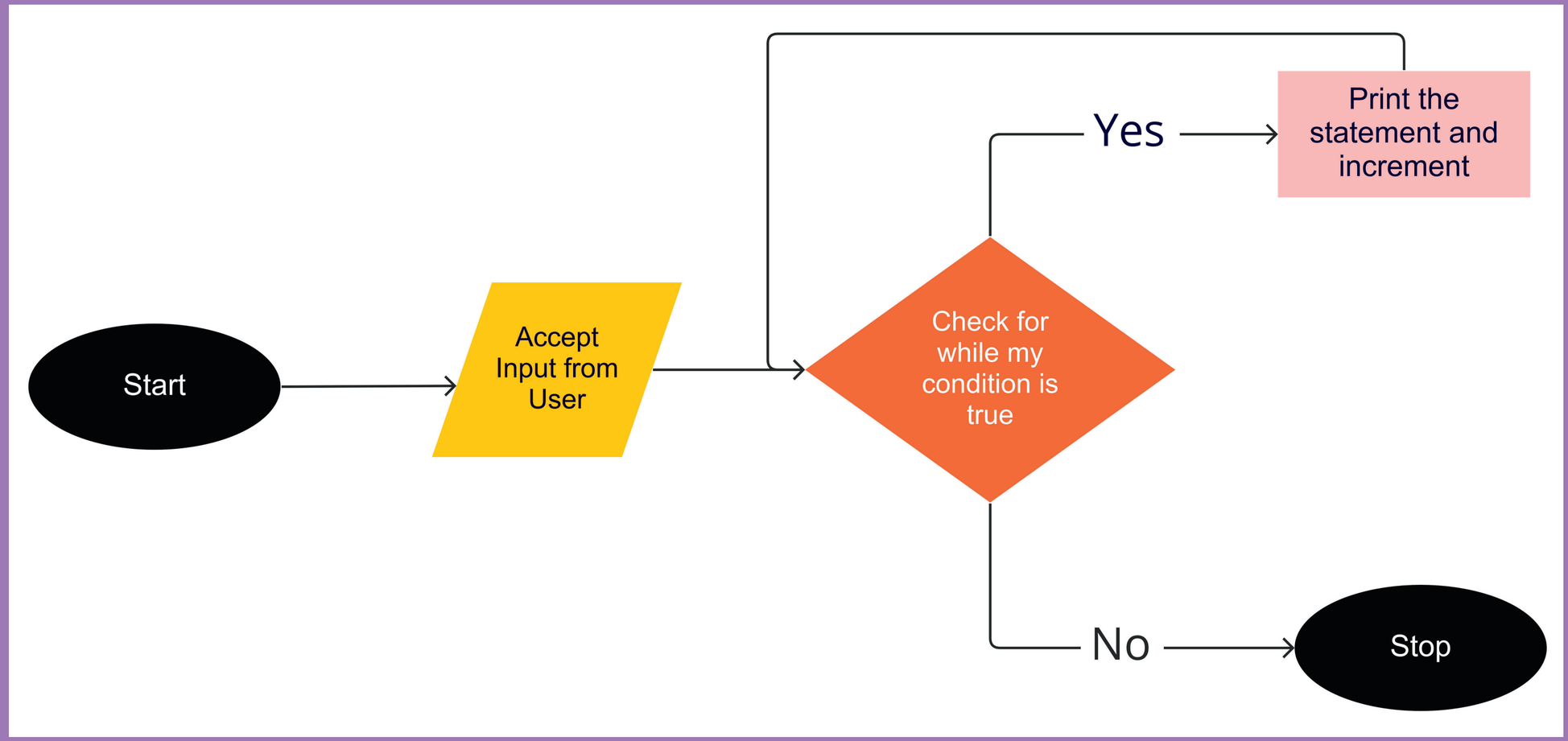


Syntax

```
while (conditions){

    // statements

}
```

Let's Code Along

Help me in Writing "Sorry, I won't disturb the class" 100 times



Write a C program to find the sum of all numbers from 0 to n using while loop.

TRY THIS YOURSELF

Output

```
C:\Users\ghosh\CLionProjects\Untitled\cmake-build-debug\Untitled.exe
Enter the last number:
4
The Number is 0
Print sum value till now 0


The Number is 1
Print sum value till now 1


The Number is 2
Print sum value till now 3


The Number is 3
Print sum value till now 6


The Number is 4
Print sum value till now 10


The value of SUM is 10

Process finished with exit code 0
```

Loop Control Statements

1. Break Statement
2. Continue Statement

Used to help a user to specify a program control's flow.

These make it possible for the program to make certain decisions, perform various tasks repeatedly, or even jump from any one section of the code to a different section.

What is a BREAK statement?

The break statement is mostly used with if statements in loops and switch cases.

We can change how the break statement works in loops to break the loop control.

```
for (initialization; condition test; increment or decrement)
{
        //loop body contains statements to be executed
         if (break condition)
        {
---------break;
        }
}


-->//statement just below loop
```

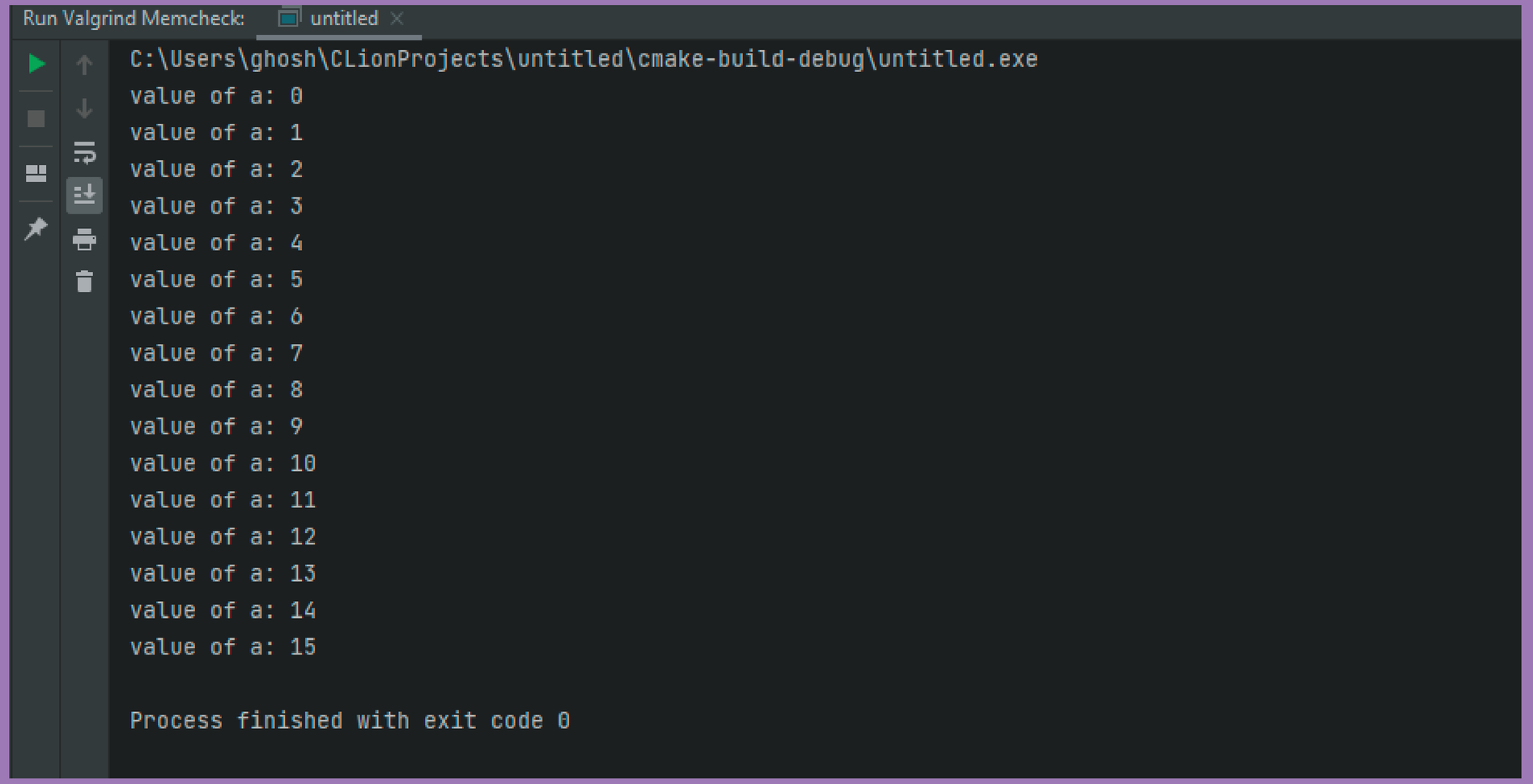
```
while(condition)
  {
      //loop body
      if (break condition)
     {
---------break;
     }
  }


-->//statement just below loop
```

Let's Code Along

Let us work on a Break statement with FOR loop.

Let's try and Print all numbers from 0 to 20, but, I don't want the program to run after I reach 15.

Output


Run Valgrind Memcheck:
untitled
C:\Users\ghosh\CLionProjects\untitled\cmake-build-debug\untitled.exe
value of a: 0
value of a: 1
value of a: 2
value of a: 3
value of a: 4
value of a: 5
value of a: 6
value of a: 7
value of a: 8
value of a: 9
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
Process finished with exit code 0

Let's Code Along

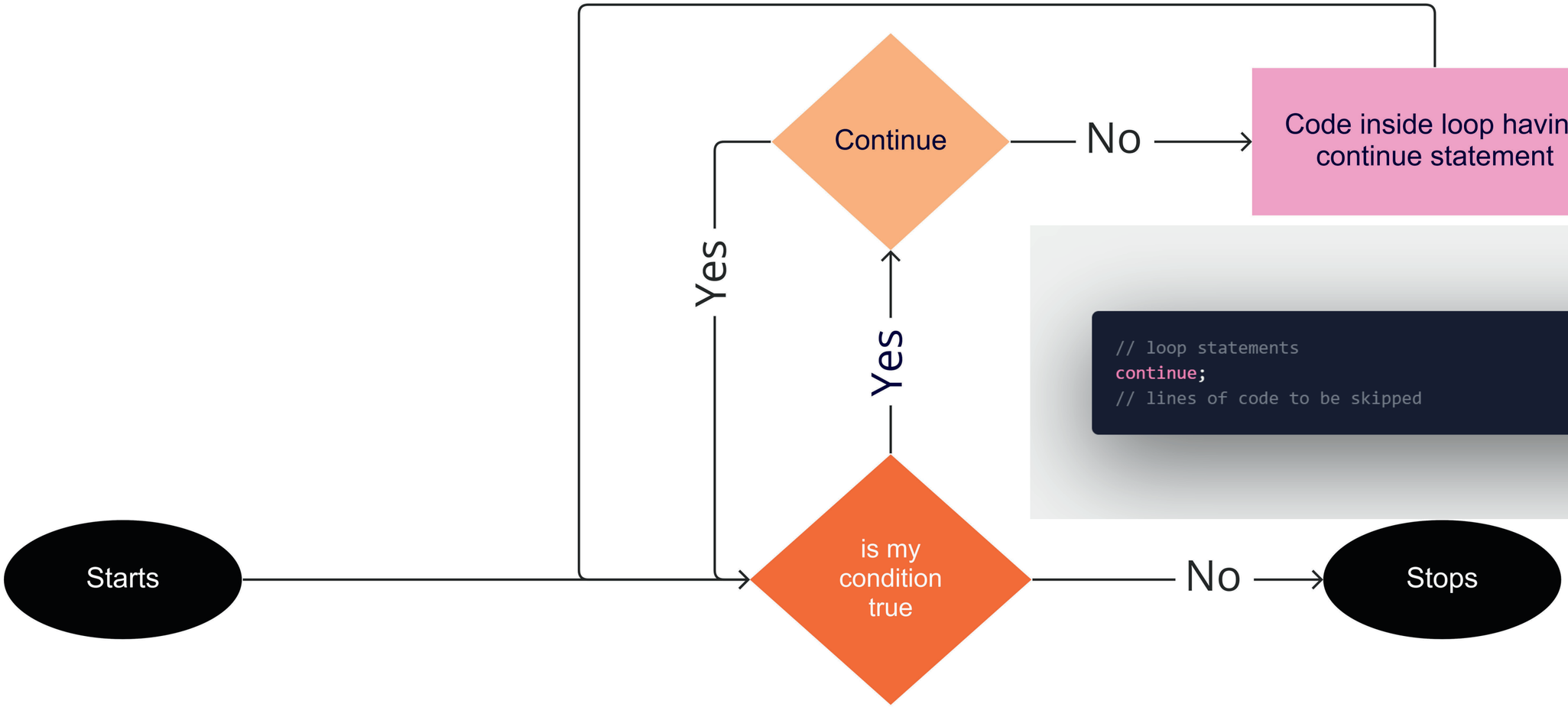
Let us work on a Break statement with WHILE loop.

Program to calculate the sum of numbers (10 numbers max) and
If the user enters a number less than 5, the loop terminates and
the number is not added to the sum

What is a CONTINUE statement?

Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

It is mainly used for a condition so that we can skip some code for a particular condition.




```
// loop statements
continue;
// lines of code to be skipped
```

Flow in Continue Statement

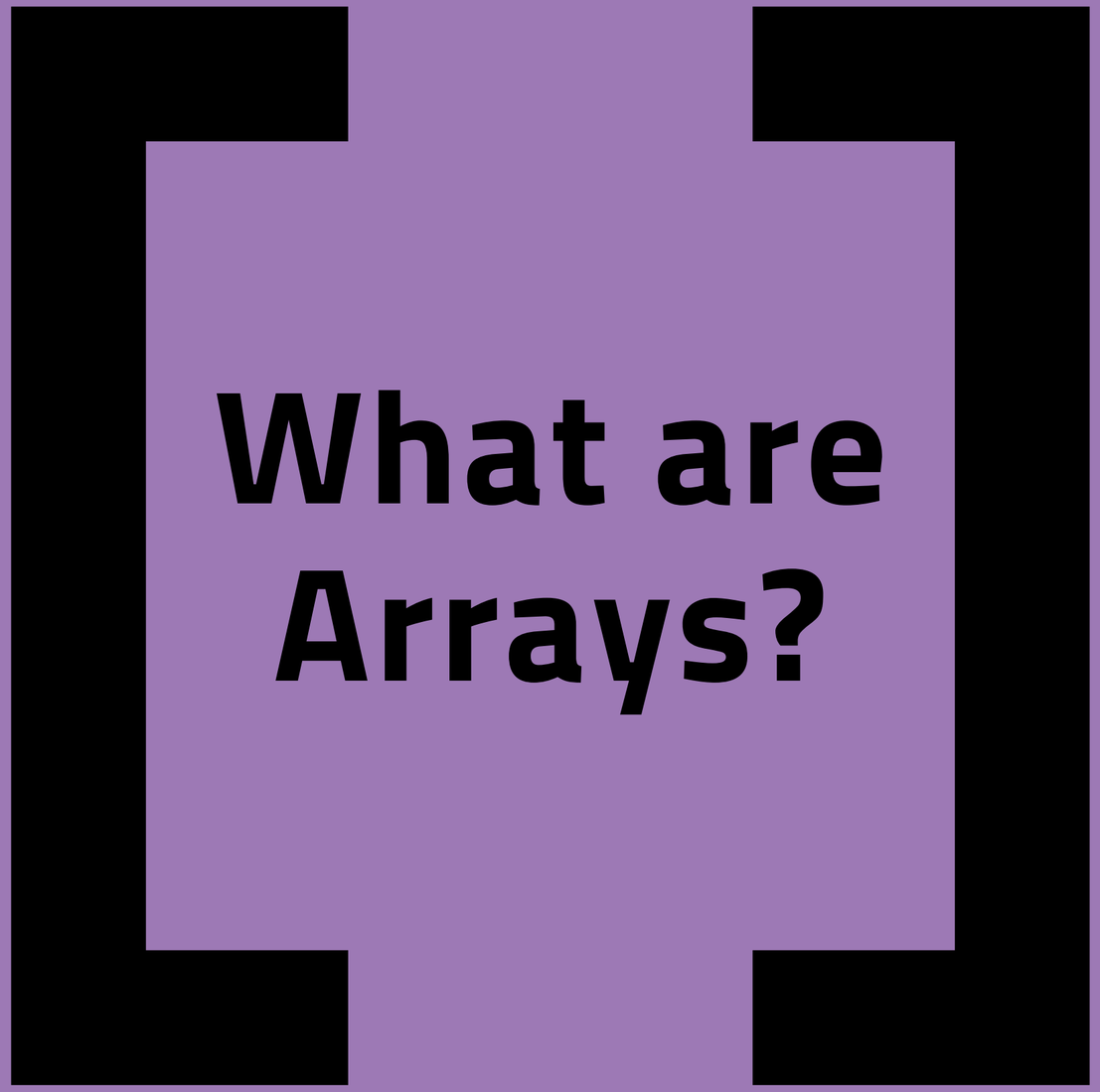
Let's Code Along

Finding odd numbers from 0 to 20.

Now, what if we try to write some code or some statements after a continue statement? Will those be executed?

Let a movie theatre has 30 seats, and 5 seats from the 15th seat are booked, so how can we show the remaining seats to people?

TRY THIS YOURSELF


What are
Arrays?

What are Arrays

An array is a group of related data items stored at adjacent memory locations, and its elements can be accessed at random using an array's indices.

- It is a collection of variables with similar data types that are referred to by a single element.
- Its components are kept in a single, continuous memory location.
- The array's size should be specified when it is declared and are always counted beginning with zero (0)
- The position of an array element in the array can be used to access it.

Types of Arrays

1. 1D Array
2. 2D Array

How to declare a 1D array?

The general syntax of defining array is like-

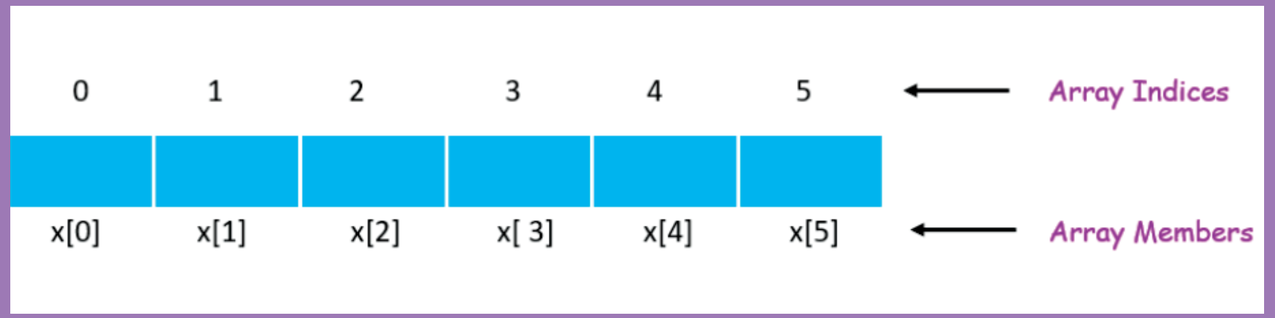
```
dataType arrayName[arraySize];
```

```
int x[6];
```

How to access a 1D Array?

The general syntax of accessing an array is like-

```
array[index];
```



- The array indexes begin with 0. x[0] denotes the first element stored at index 0.
- If an array has n elements, the last element is stored at index (n-1). The last element in this example is x[5].
- An array's elements have consecutive addresses in the memory.

1D Array Initialization?

```
int x[6] = {10, 20, 30, 40, 50, 60};
```

```
int x[] = {10, 20, 30, 40, 50, 60};
```

```
int x[3];
x[0] = 10;
x[1] = 20;
x[2] = 30;
```

How to Input 1D Array Elements?

```c
// take input and store it in the 3rd element
scanf("%d", &mark[2]);

// take input and store it in the ith element
scanf("%d", &mark[i-1]);
```

How to Output 1D Array Elements?

```c
// print the first element of the array
printf("%d", mark[0]);

// print the third element of the array
printf("%d", mark[2]);

// print ith element of the array
printf("%d", mark[i-1]);
```

Let's Code Along 1D Arrays

Program to take 5 values from the user and store them in an array

Print the elements stored in the array

Find the average of n numbers using arrays

TRY THIS YOURSELF

What are 2D Arrays

```
int x[3][4];
```

Here, x is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

	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]

2D Array Initialization?

```
int c[][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
int c[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
int c[2][3] = {1, 2, 3, 4, 5, 6};
```

```
int c[2][3];
  c[0][0] = 1;
  c[0][1] = 2;
  c[0][2] = 3;
  c[1][0] = 4;
  c[1][1] = 5;
  c[1][2] = 6;
```

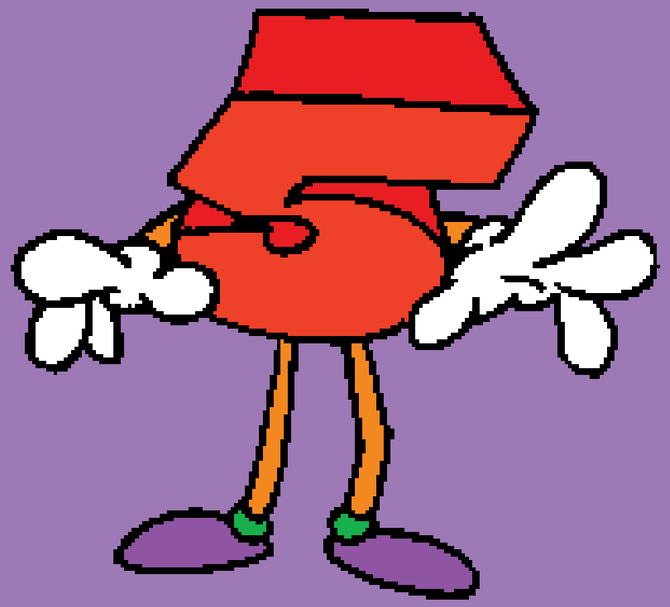
How to Output 2D Array Elements?

```c
// print the first element of the two arrays
printf("%d", mark[0][0]);

// print the third element of the second array
printf("%d", mark[2][1]);

// print ith element of the jth array
printf("%d", mark[i-1][j-1]);
```

Let's Code Along 2D Arrays

Write a C program to store the temperature of two cities for a week and display it.

Display the data that you have stored.



C program to find the sum of two matrices of order 2*2

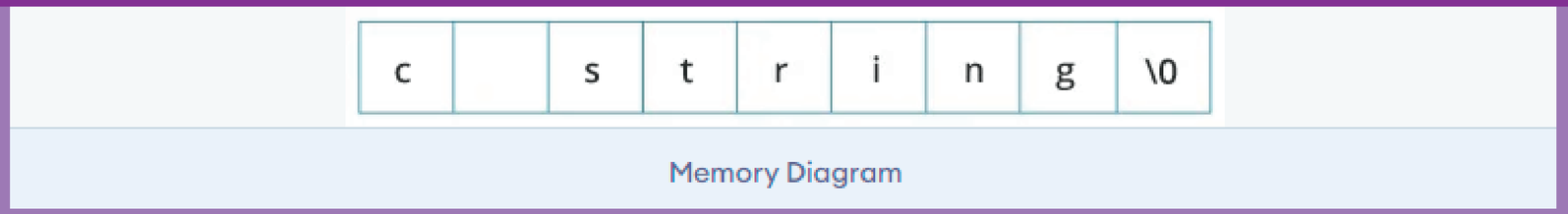
TRY THIS YOURSELF

What are Strings?

A string is a sequence of characters terminated with a null character \0

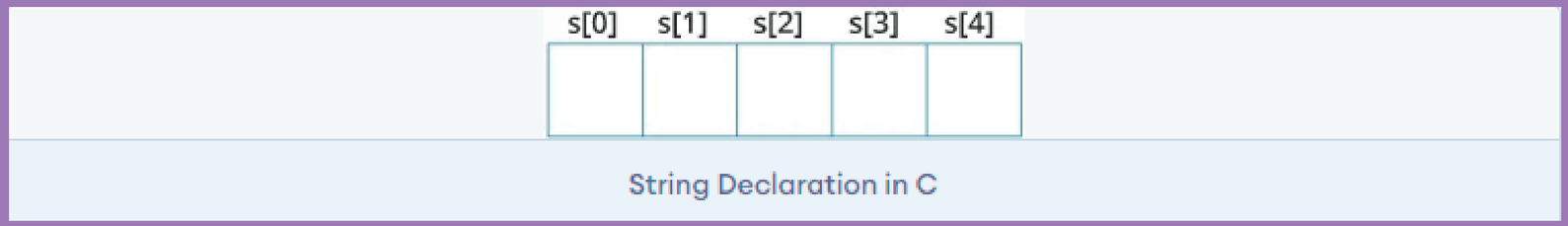
```
char c = ["c string"];
```

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.



String Declaration

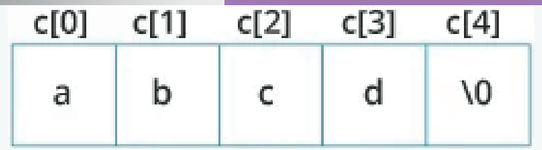
```
char s[5];
```



String Declaration in C

String Initialization

```c
//  Type 1
char s[] = "hello";

// Type 2
char s[50] = "hello";

// Type 3
char s[] =  {'h','e','l','l','o'};

// Type 4
char s[6] = {'h','e','l','l','o','\0'};
```



String Initialization in C

Which one is Correct?

❶

```
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.

❷

```
char c[100];
c = "C programming";
```

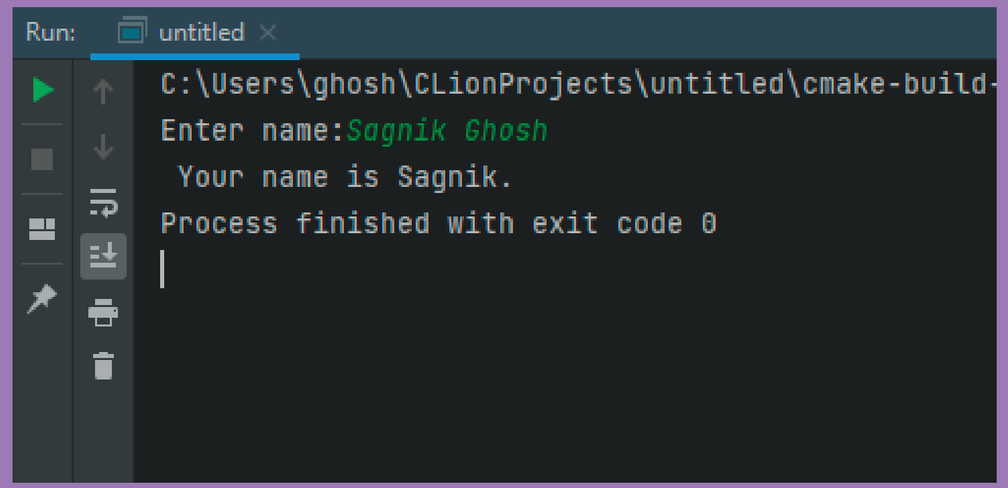
Arrays and strings do not support the assignment operator once it is declared

Read String from the user

You can use the scanf() function to read a string.

The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab, etc.).

```c
#include <stdio.h>

int main() {

    char name[20];
    printf("Enter name: ");
    scanf("%s", name);
    printf("Your name is %s.", name);
    return 0;

}
```

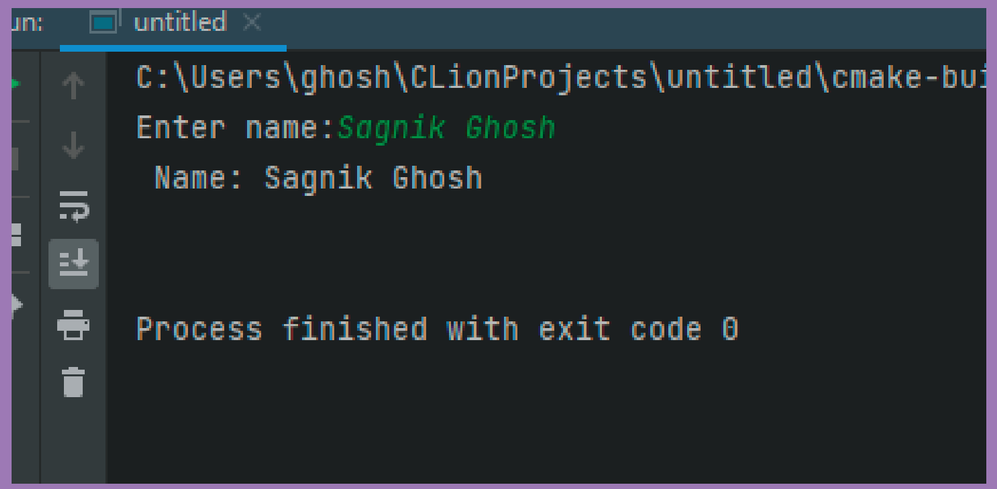


Read a Line

You can use the fgets() function to read a line of string. And, you can use puts() to display the string.

```
fgets(pString, size, stdin);
```

```c
#include <stdio.h>

int main()
{
    char name[30];
    printf("Enter name: ");
    fgets(name, sizeof(name), stdin);  // read string
    printf("Name: ");
    puts(name);    // display string
    return 0;
}
```



Commonly Used String Functions

- strlen()
- strcpy()
- strcmp()
- strcat()

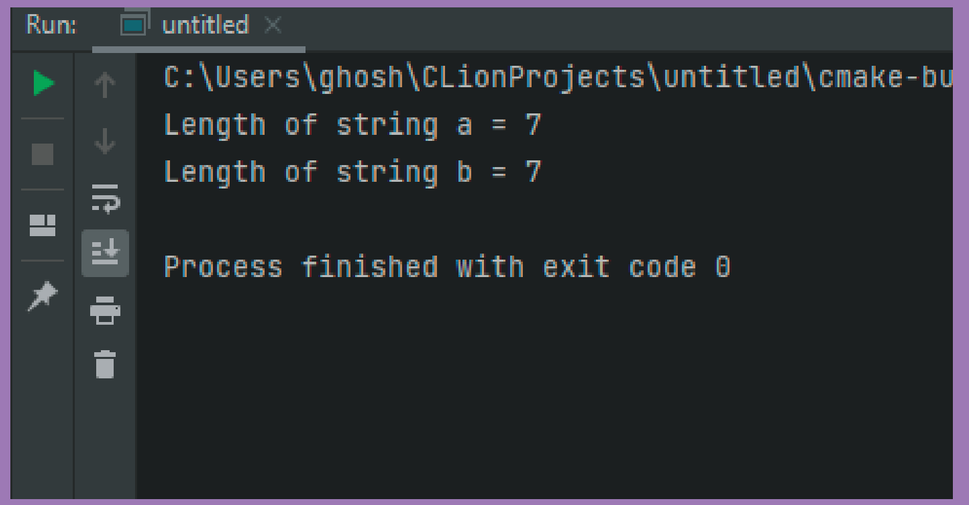
strlen()

The strlen() function calculates the length of a given string.

The strlen() function takes a string as an argument and returns its length.

The returned value is of type size_t (an unsigned integer type).

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char a[20]="Program";
    char b[20]={'P','r','o','g','r','a','m','\0'};

    // using the %zu format specifier to print size_t
    printf("Length of string a = %d \n",strlen(a));
    printf("Length of string b = %d \n",strlen(b));

    return 0;
}
```



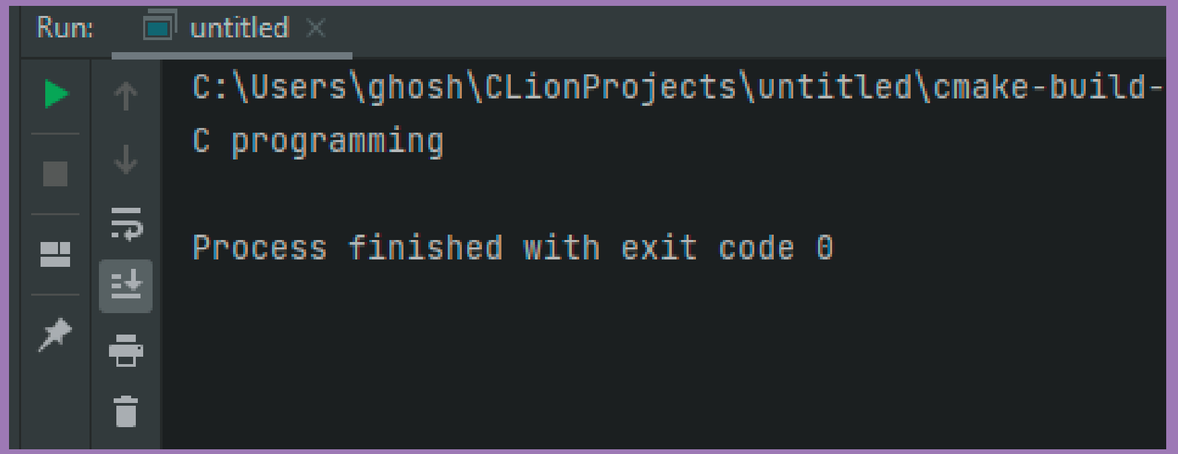
strcpy()

- The strcpy() function copies the string pointed by the source (including the null character) to the destination.

- The strcpy() function also returns the copied string.

```c
char *strcpy(char *destination, const char *source);
```

```c
#include <stdio.h>
#include <string.h>

int main() {
  char str1[20] = "C programming";
  char str2[20];

  // copying str1 to str2
  strcpy(str2, str1);

  puts(str2); // C programming

  return 0;
}
```



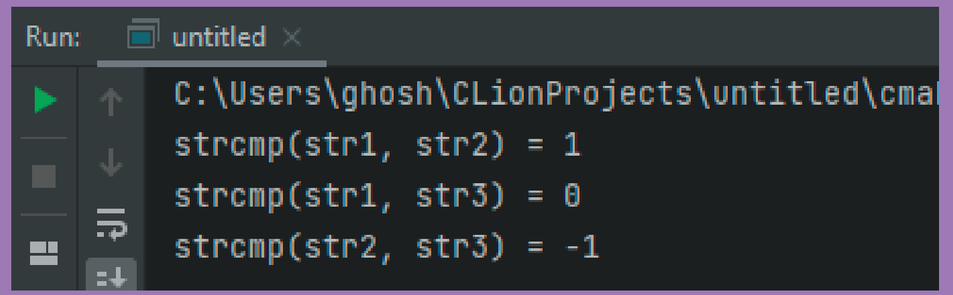
strcmp()

The strcmp() compares two strings character by character. If the strings are equal, the function returns 0.

```
int strcmp (const char* str1, const char* str2);
```

Return Value	Remarks
0	if strings are equal
>0	if the first non-matching character in str1 is greater (in ASCII) than that of str2.
<0	if the first non-matching character in str1 is lower (in ASCII) than that of str2.

```c
#include <stdio.h>
#include <string.h>

int main() {
    char str1[] = "abcd", str2[] = "abCd", str3[] =
"abcd";
    int result;

    // comparing strings str1 and str2
    result = strcmp(str1, str2);
    printf("strcmp(str1, str2) = %d\n", result);

    // comparing strings str1 and str3
    result = strcmp(str1, str3);
    printf("strcmp(str1, str3) = %d\n", result);

    // comparing strings str2 and str3
    result = strcmp(str2, str3);
    printf("strcmp(str2, str3) = %d\n", result);

    return 0;
}
```



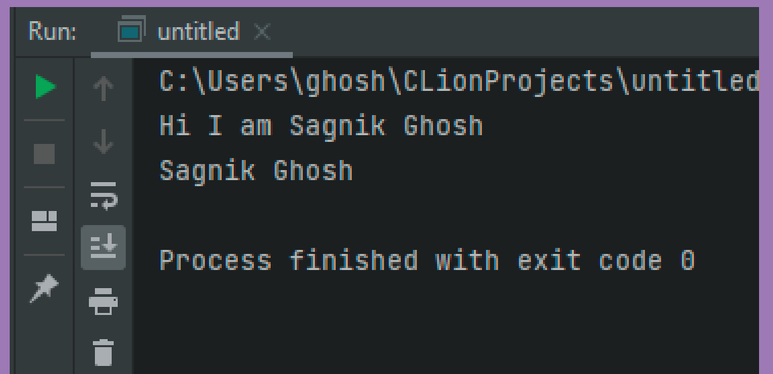
strcat()

the strcat() function contcatenates (joins) two strings.

```
char *strcat(char *destination, const char *source)
```

The strcat() function concatenates the destination string and the source string, and the result is stored in the destination string.

```c
#include <stdio.h>
#include <string.h>
int main() {
    char str1[100] = "Hi I am ", str2[] = "Sagnik Ghosh";

    // concatenates str1 and str2
    // the resultant string is stored in str1.
    strcat(str1, str2);

    puts(str1);
    puts(str2);

    return 0;
}
```



What are Address Operators?

Address Operators

An address of the operator is used within C that is returned to the memory address of a variable. These addresses returned by the address of the operator are known as pointers because they “point” to the variable in memory.

While scanning the user input we used ampersand operator.

While displaying the address of the variable we used ampersand operator.

Establish pointer values and address of operator’s point to the memory location because the value of the pointer is the memory location or memory address. The data item saved in memory.

The address operator is working to return the memory address of a variable. These addresses are returned by the address of the operator are known as pointers because they point to the variable in memory.

What are Pointers?

Pointers (pointer variables) are special variables that are used to store addresses rather than values.

We can declare pointers in three ways

```
// Type 1
int* p

// Type 2
int * p

// Type 3
int *p
```

Assigning Address to Pointers

```
int* pc, c;
c = 5;
pc = &c;
```

To get the value of the thing pointed by the pointers, we use the * operator. For example:

```
int* pc, c;
c = 5;
pc = &c;
printf("%d", *pc);   // Output: 5
```

Changing Value Pointed by Pointers

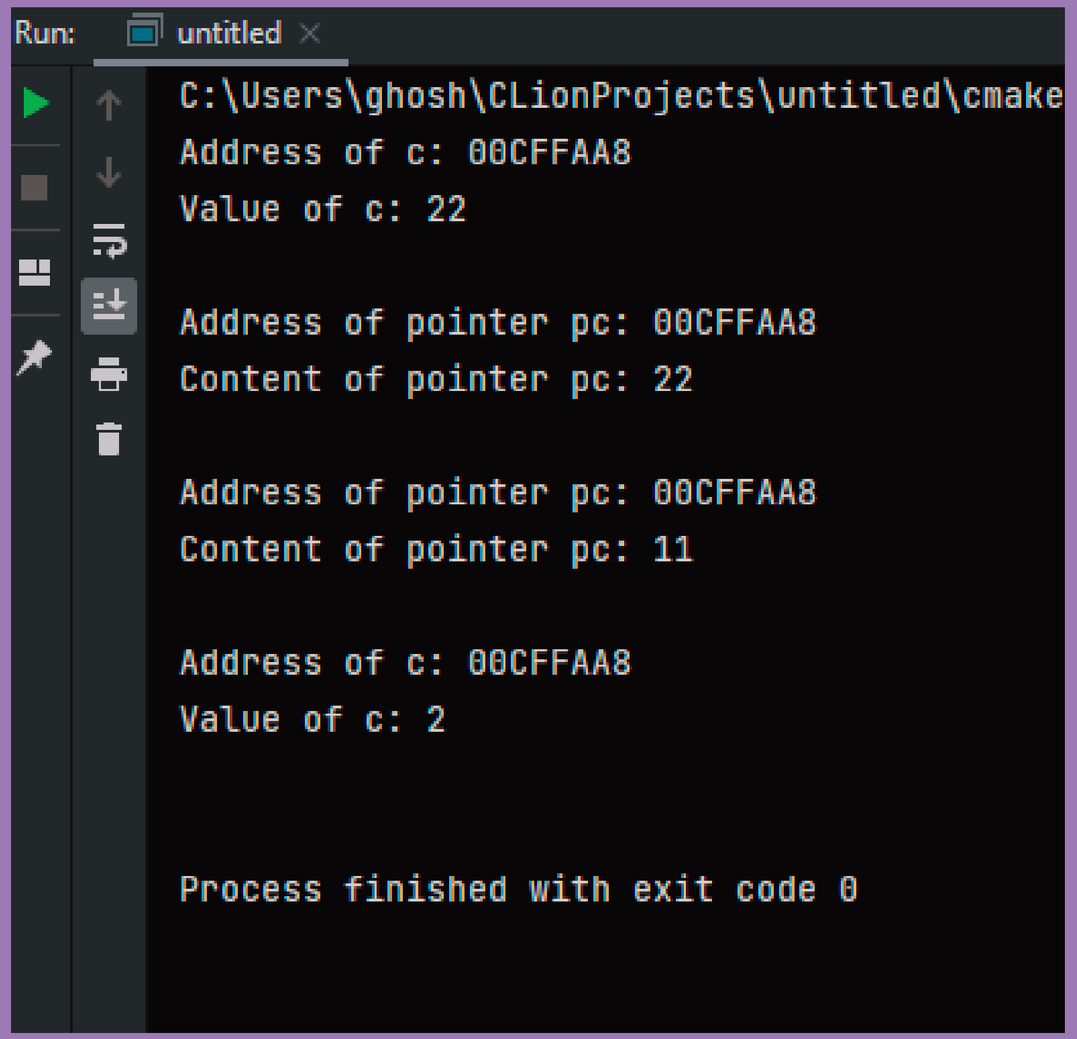
```c
int* pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);    // Output: 1
printf("%d", *pc);  // Ouptut: 1
```

```c
// 2
int* pc, c;
c = 5;
pc = &c;
*pc = 1;
printf("%d", *pc);  // Ouptut: 1
printf("%d", c);    // Output: 1
```

```c
// 3
int* pc, c, d;
c = 5;
d = -15;

pc = &c; printf("%d", *pc); // Output: 5
pc = &d; printf("%d", *pc); // Ouptut: -15
```

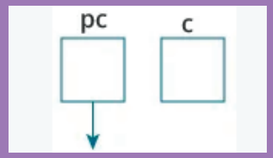
Example

```c
#include <stdio.h>
int main()
{
    int* pc, c;

    c = 22;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c);  // 22

    pc = &c;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc); // 22

    c = 11;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc); // 11

    *pc = 2;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c); // 2
    return 0;
}
```

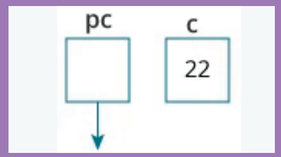


Explaination

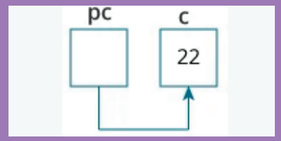
1. `int* pc, c;`



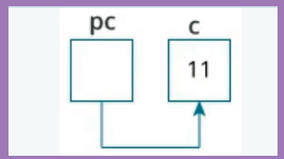
2. `c = 22;`



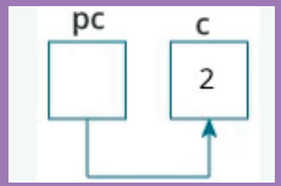
3. `pc = &c;`



4. `c = 11;`



5. `*pc = 2;`



Valid Pointer Declarations

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

SAGNIK GHOSH

twitter.com/sagnik_ghosh_11

linkedin.com/in/sagnikghosh1111

instagram.com/_.sagnik._.ghosh._

Email: sagnik@worqhat.com

Ph# 9960806748