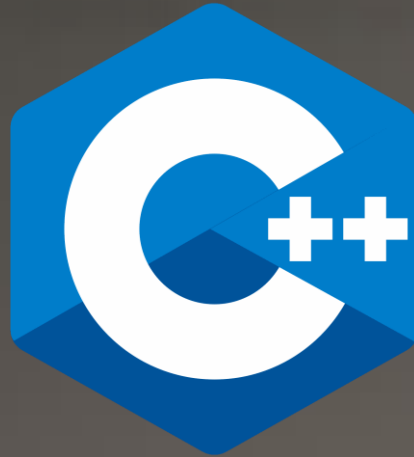


# C++ for Self-Driving Vehicles



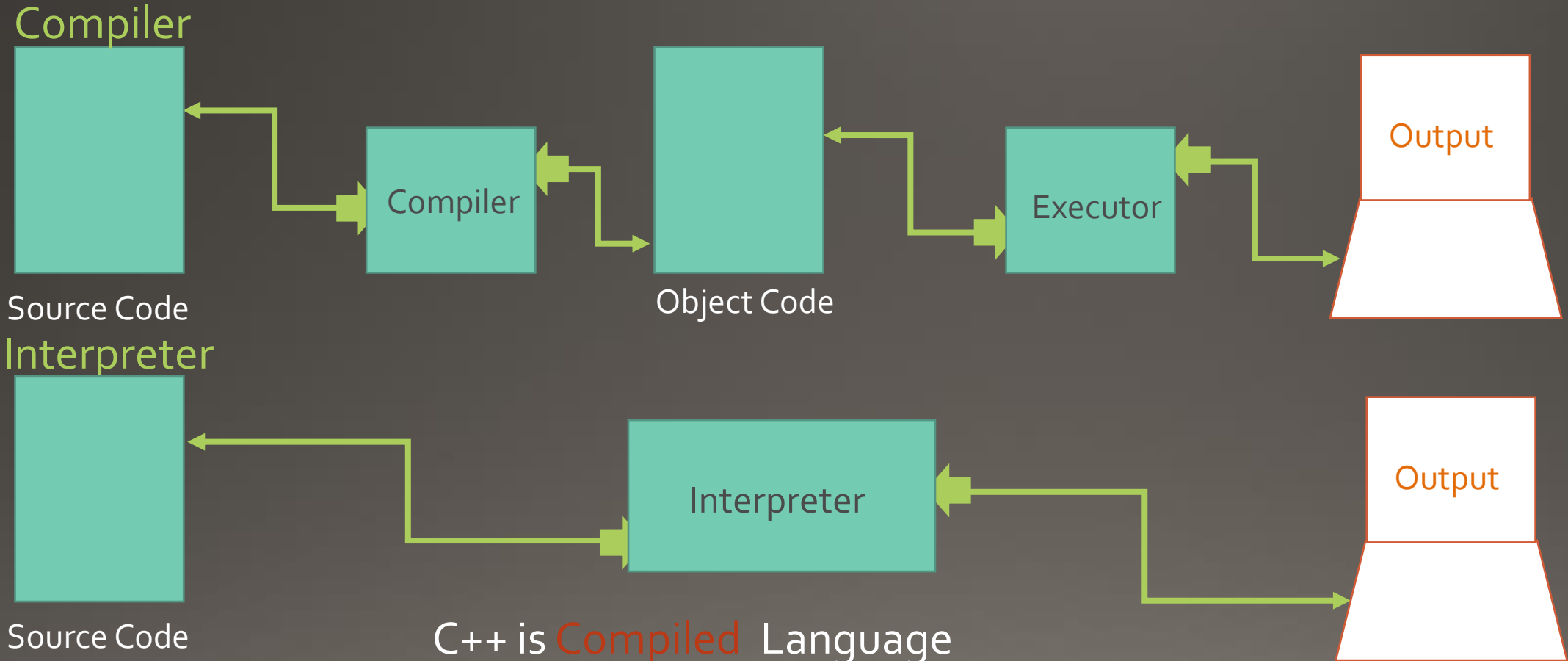
## Introduction To C++

- C++ is a multi-paradigm programming language that supports object-oriented programming (OOP), created by Bjarne Stroustrup in 1983 at Bell Labs, C++ is an extension(superset) of C programming and the programs are written in C language can run in C++ compilers.
- C++ is used by programmers to create computer software. It is used to create general systems software, drivers for various computer devices, software for servers and software for specific applications and also widely used in the creation of video games.
- C++ is used by many programmers of different types and coming from different fields. C++ is mostly used to write device driver programs, system software, and applications that depend on direct hardware manipulation under real-time constraints. It is also used to teach the basics of object-oriented features because it is simple and is also used in the fields of research. Also, many primary user interfaces and system files of Windows and Macintosh are written using C++. So, C++ is a popular, strong and frequently used programming language of this modern programming era.



# How does C++ work?

## Compiler vs Interpreter



## Introduction To C++

- A compiler is a computer program that transforms humanly readable (programming language) source code into another computer language (binary) code.
- In simple terms, Compiler takes the code that you wrote and turned in to the binary code that the computer can understand.
- You can install MinGW for windows

```
C:\Users\Mohamed Mesbah>gcc -v
```

```
gcc version 9.2.0 (MinGW.org GCC Build-2)
```

- For Linux:
  - g++ is a C++ compiler that comes with most Unix distributions.
- GCC will work already in your system for Linux.



# Hello World Program

- Open an empty file and save it with the extension .cpp
  - Where "helloworld.cpp" is the name of your c++ file.

 helloworld.cpp

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!";
    return 0;
}
```

- Save your file. Open your command line, navigate to the directory where you saved your file, and run:

```
\Desktop\HelloWorld>g++ -Wall -std=c++14 helloworld.cpp -o helloworld
```

- The output should read:

```
C:\Users\Mohamed Mesbah\Desktop\HelloWorld>helloworld
Hello World!
```



# C++ Introduction

- `#include <iostream>` is a header file library that lets us work with input and output objects, such as `cout` (used in line 5). Header files add functionality to C++ programs.
- `using namespace std` means that we can use names for objects and variables from the standard library.
- `int main()` This is called a function. Any code inside its curly brackets `{ }` will be executed.
- `cout` (pronounced "see-out") is an object used together with the insertion operator (`<<`) to output/print text. In our example it will output "Hello World".
- Every C++ statement ends with a semicolon `;`.
- `return 0` ends the main function.
- `endl` and `\n` way to insert a new line.
- `//` uses a single-line comment at the end of a line of code.
- Multi-line comments start with `/*` and ends with `*/`

```
#include <iostream>
using namespace std;
/* The code below will print the words Hello World!
to the screen, and it is amazing */
int main() {
    cout << "Hello World!"<<endl; // print Hello world
    cout << "Hello Mesbah!\n";
    return 0;
}
```



## Variables

- `int` - stores integers (whole numbers), without decimals, such as 123 or -123
- `double` - stores floating point numbers, with decimals, such as 19.99 or -19.99
- `char` - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- `string` - stores text, such as "Hello World". String values are surrounded by double quotes
- `bool` - stores values with two states: true or false

*`type variable = value;`*



## Variables

```
int myNum = 5;           // Integer (whole number without decimals)
double myFloatNum = 5.99; // Floating point number (with decimals)
char myLetter = 'D';     // Character
string myText = "Hello"; // String (text)
bool myBoolean = true;   // Boolean (true or false)
const int myNum = 15;    // myNum will always be 15
```





## User Input

`cout` is pronounced "see-out". Used for **output**, and uses the insertion operator (`<<`)  
`cin` is pronounced "see-in". Used for **input**, and uses the extraction operator (`>>`)

```
int x;  
cout << "Type a number: "; // Type a number and press enter  
cin >> x; // Get user input from the keyboard  
cout << "Your number is: " << x; // Display the input value
```



## Data Types

```
int myNum = 5;           // Integer (whole number)
float myFloatNum = 5.99; // Floating point number
double myDoubleNum = 9.98; // Floating point number
char myLetter = 'D';     // Character
bool myBoolean = true;   // Boolean
string myText = "Hello"; // String
```



## Data Types

Data Type	Meaning	Size (in Bytes)
int	Integer	2 or 4
float	Floating-point	4
double	Double floating-point	8
char	Character	1
bool	Boolean	1
void	Empty	0



## Operators

Operators in C++ can be classified into 6 types:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Other Operators



## Arithmetic Operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation(Remainder after division)

```
// printing the sum of a and b
cout << "a + b = " << (a + b) << endl;

// printing the difference of a and b
cout << "a - b = " << (a - b) << endl;

// printing the product of a and b
cout << "a * b = " << (a * b) << endl;

// printing the division of a by b
cout << "a / b = " << (a / b) << endl;

// printing the modulo of a by b
cout << "a % b = " << (a % b) << endl;
```



## Assignment Operators

Operator	Meaning	Example
=	<code>a = b;</code>	<code>a = b;</code>
+=	<code>a += b;</code>	<code>a = a + b;</code>
-=	<code>a -= b;</code>	<code>a = a - b;</code>
*=	<code>a *= b;</code>	<code>a = a * b;</code>
/=	<code>a /= b;</code>	<code>a = a / b;</code>
%=	<code>a %= b;</code>	<code>a = a % b;</code>



## Relational Operators

Operator	Example	Meaning
==	Is Equal To	3 == 5 gives us <b>false</b>
!=	Not Equal To	3 != 5 gives us <b>true</b>
>	Greater Than	3 > 5 gives us <b>false</b>
<	Less Than	3 < 5 gives us <b>true</b>
>=	Greater Than or Equal To	3 >= 5 give us <b>false</b>
<=	Less Than or Equal To	3 <= 5 gives us <b>true</b>



## Logical Operators

Operator	Example	Equivalent to
&&	expression1 && expression2	Logical AND. True only if all the operands are true.
	expression1    expression2	Logical OR. True if at least one of the operands is true.
!	!expression	Logical NOT. True only if the operand is false.





## Bitwise Operators

Operator	Description
&	Binary AND
	Binary OR
^	Binary XOR
~	Binary One's Complement
<<	Binary Shift Left
>>	Binary Shift Right



## C++ Flow Control

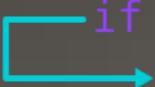
### if, if...else and Nested if...else

#### if Statement

```
if (condition) {  
    // body of if statement  
}
```


#### Condition is true

```
int number = 5;  
  
if (number > 0) {  
    // code  
}  
  
// code after if
```



#### Condition is false

```
int number = 5;  
  
if (number < 0) {  
    // code  
}  
  
// code after if
```



## if...else

```
if (condition) {  
    // block of code if condition is true  
}  
else {  
    // block of code if condition is false  
}
```

### Condition is true

```
int number = 5;  
  
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if...else
```

### Condition is false

```
int number = 5;  
  
if (number < 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if...else
```

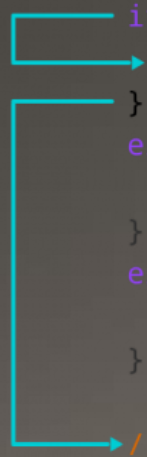


## if...else...else if statement

```
if (condition1) {
    // code block 1
}
else if (condition2) {
    // code block 2
}
else {
    // code block 3
}
```

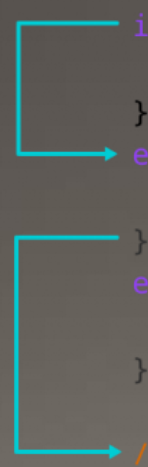
1st Condition is true

```
int number = 2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}
//code after if
```



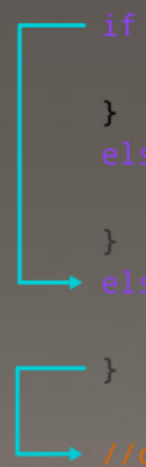
2nd Condition is true

```
int number = 0;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}
//code after if
```



All Conditions are false

```
int number = -2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}
//code after if
```




## Nested if...else

```
// outer if statement
if (condition1) {
    // statements

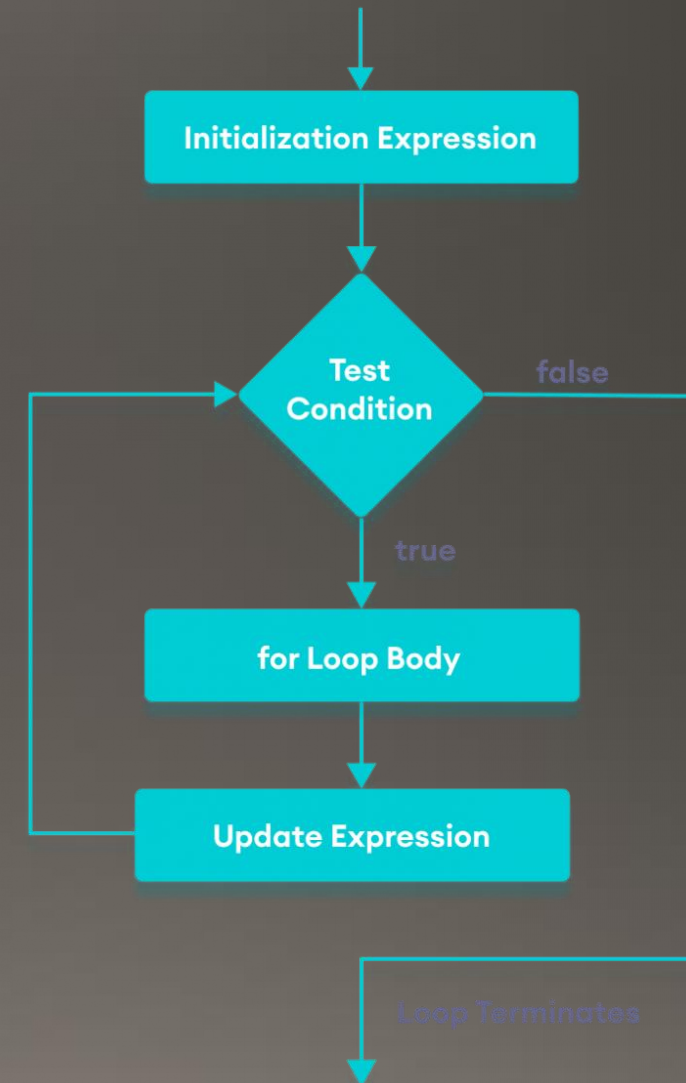
    // inner if statement
    if (condition2) {
        // statements
    }
}
```



# loops in C++

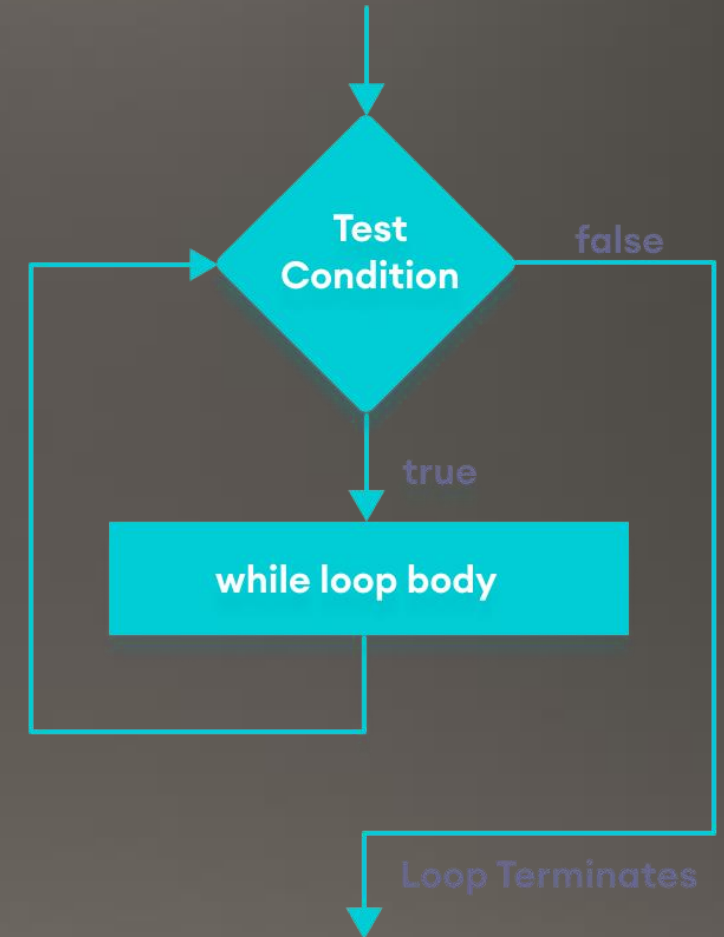
## for Loop

```
for (initialization; condition; update) {  
    // body of-loop  
}
```



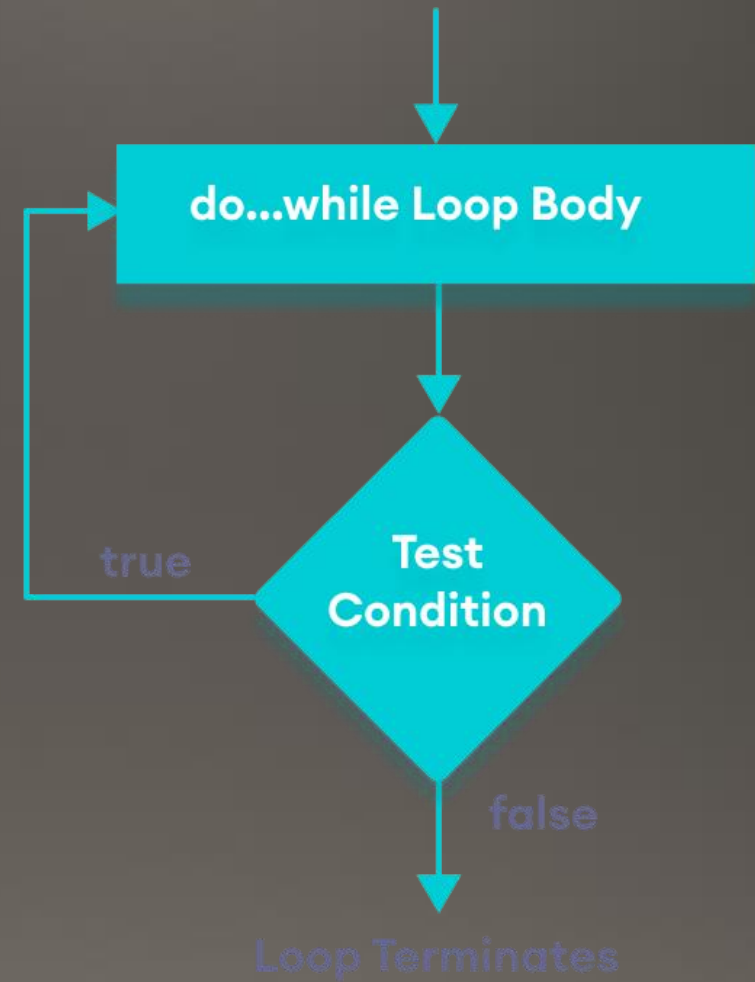
## while Loop

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



## do...while Loop

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





# break & continue Statement

## break Statement

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

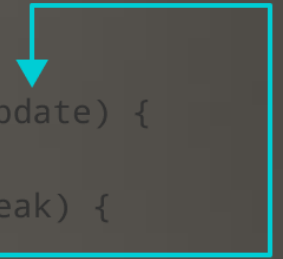


```
while (condition) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

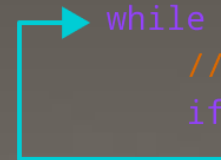


## continue Statement

```
for (init; condition; update) {
    // code
    if (condition to break) {
        continue;
    }
    // code
}
```



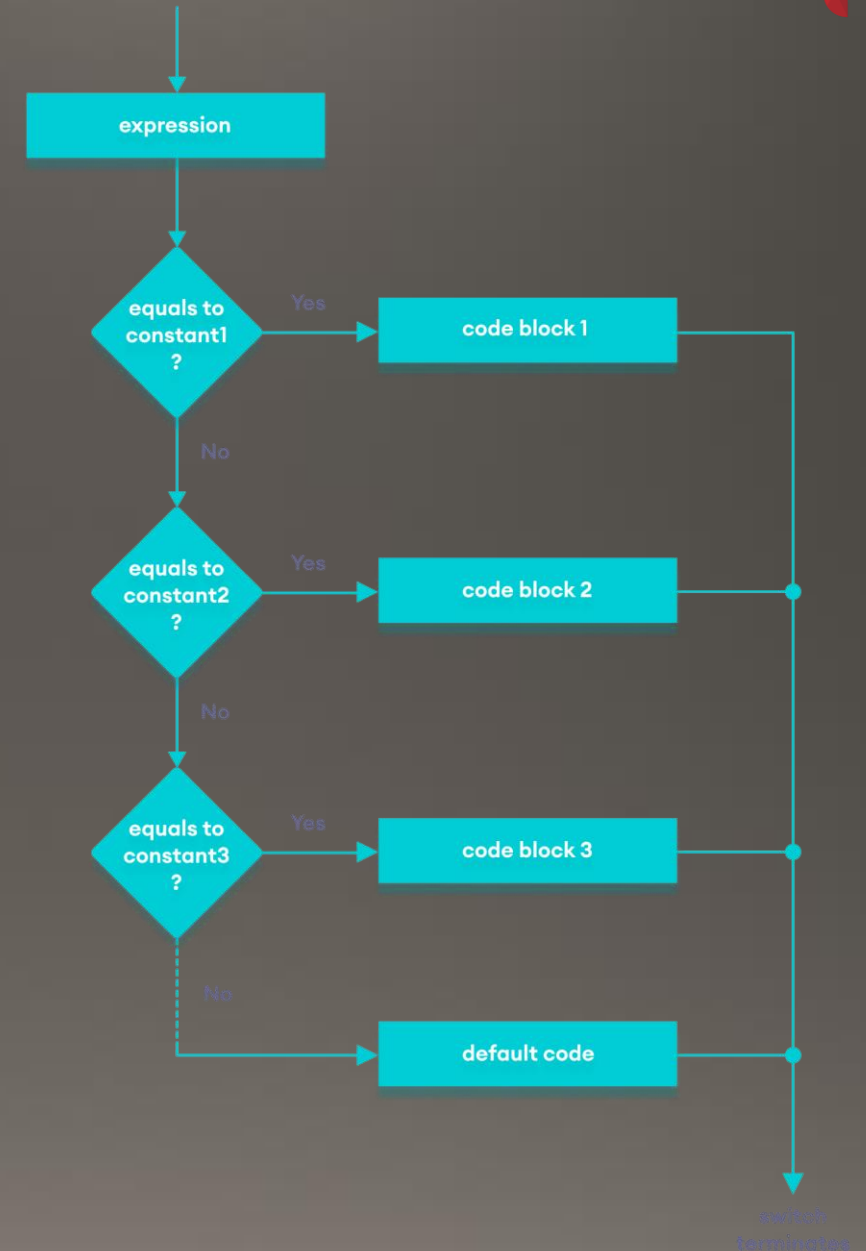
```
while (condition) {
    // code
    if (condition to break) {
        continue;
    }
    // code
}
```



# switch..case Statement

```
switch (expression) {
  case constant1:
    // code to be executed if
    // expression is equal to constant1;
    break;

  case constant2:
    // code to be executed if
    // expression is equal to constant2;
    break;
  .
  .
  .
  default:
    // code to be executed if
    // expression doesn't match any constant
}
```



## goto Statement

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

```
goto label;
... ..
... ..
→ label:
... ..
... ..
```

```
→ label;
... ..
... ..
goto label:
... ..
... ..
```



## C++ Functions

### Function Declaration

```
returnType functionName (parameter1, parameter2,...) {  
    // function body  
}
```

```
// function declaration  
void greet() {  
    cout << "Hello World";  
}
```




## Calling a Function

```
#include<iostream>

void greet() {
    // code
}

int main() {
    ... ..
    greet();
    ... ..
}
```

function call



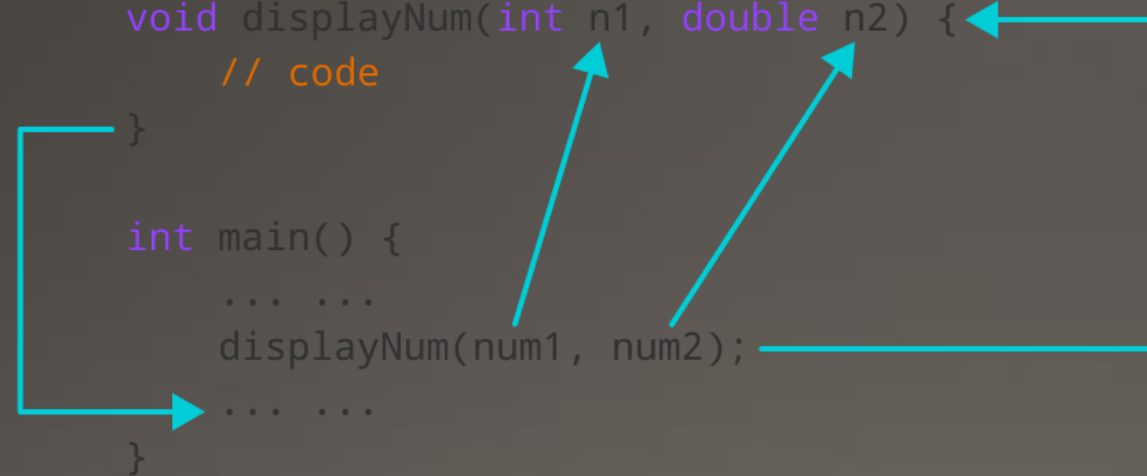
# Function Parameters

```
#include<iostream>

void displayNum(int n1, double n2) {
    // code
}

int main() {
    ... ..
    displayNum(num1, num2);
    ... ..
}
```

function call



The diagram illustrates the function call and parameter passing. A red arrow points from the function call `displayNum(num1, num2);` in the `main` function to the opening curly brace of the `displayNum` function definition. Two red arrows point from the arguments `num1` and `num2` to the parameters `n1` and `n2` respectively, showing the transfer of values.



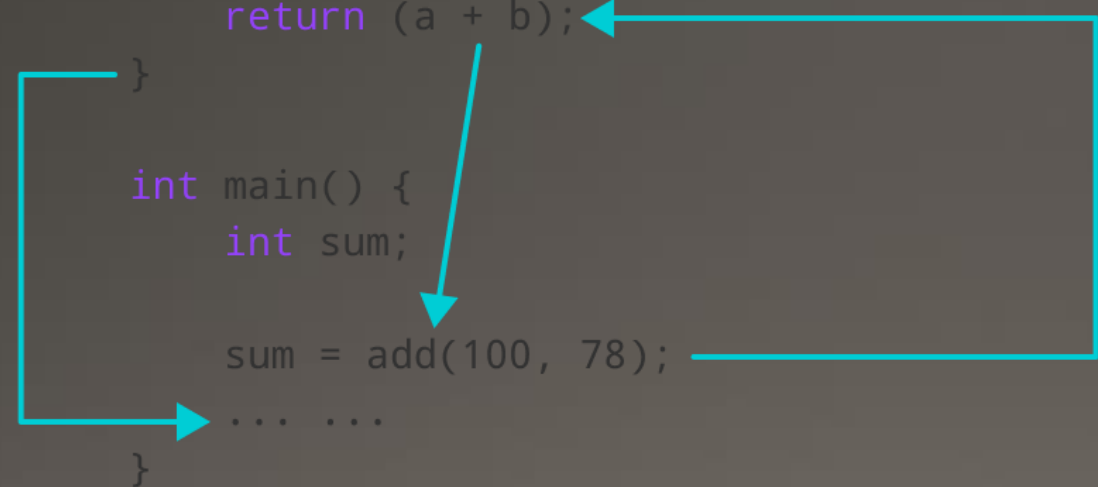
## Return Statement

```
#include<iostream>

int add(int a, int b) {
    return (a + b);
}

int main() {
    int sum;
    sum = add(100, 78);
    ... ..
}
```

function call



## Function Prototype

returnType functionName(dataType1, dataType2, ...);

```
// function prototype
void add(int, int);

int main() {
    // calling the function before declaration.
    add(5, 3);
    return 0;
}

// function definition
void add(int a, int b) {
    cout << (a + b);
}
```





# Function Overloading

```
float absolute(float var) {  
    // code  
}
```

```
int absolute(int var) {  
    // code  
}
```

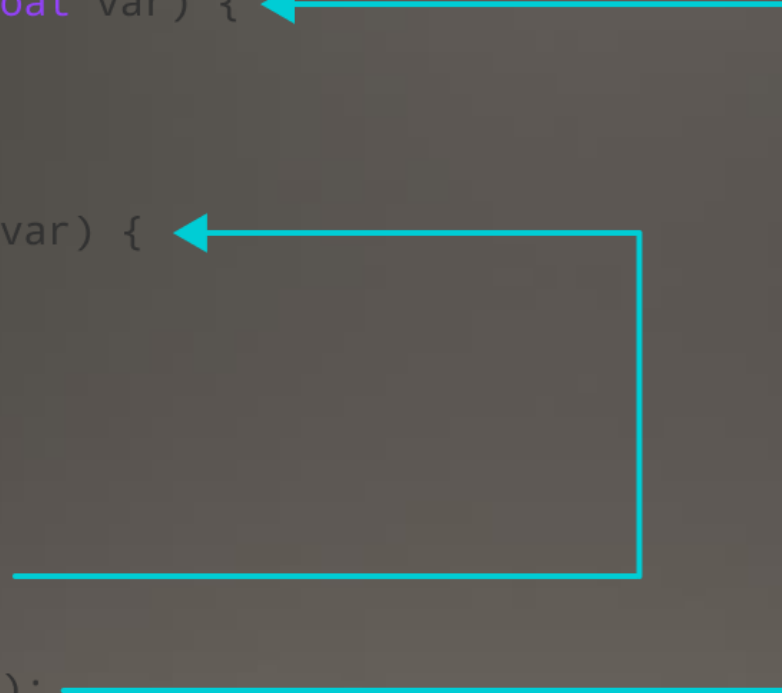
```
int main() {
```

```
    absolute(-5);
```

```
    absolute(5.5f);
```

```
    ... ..
```

```
}
```



## Local variable

A variable defined inside a function (defined inside function body between braces) is called a local variable or automatic variable.

Its scope is only limited to the function where it is defined. In simple terms, local variable exists and can be accessed only inside a function.



```
using namespace std;

void test();

int main()
{
    // local variable to main()
    int var = 5;

    test();

    // illegal: var1 not declared inside main()
    var1 = 9;
}

void test()
{
    // local variable to test()
    int var1;
    var1 = 6;

    // illegal: var not declared inside test()
    cout << var;
}
```



## Global Variable

If a variable is defined outside all functions, then it is called a global variable.

```
#include <iostream>
using namespace std;

// Global variable declaration
int c = 12;

void test();

int main()
{
    ++c;

    // Outputs 13
    cout << c << endl;
    test();

    return 0;
}

void test()
{
    ++c;

    // Outputs 14
    cout << c;
}
```



## Static Local variable

Keyword **static** is used for specifying a static variable. For example:

```
int main()
{
    static float a;
    ... ..
}
```

```
#include <iostream>
using namespace std;

void test()
{
    // var is a static variable
    static int var = 0;
    ++var;

    cout << var << endl;
}

int main()
{
    test();
    test();

    return 0;
}
```



## Recursion

A function that calls itself is known as a recursive function. And, this technique is known as recursion.

```

void recurse() {
    ... ..
    recurse();
    ... ..
}

int main() {
    ... ..
    recurse();
    ... ..
}
    
```

Diagram illustrating recursive calls:

- A blue arrow labeled "recursive call" points from the `recurse();` line inside the `recurse()` function to the opening curly brace of the `recurse()` function.
- A blue arrow labeled "function call" points from the `recurse();` line inside the `main()` function to the opening curly brace of the `recurse()` function.



```
// Factorial of n = 1*2*3*...*n

#include <iostream>
using namespace std;

int factorial(int);

int main() {
    int n, result;

    cout << "Enter a non-negative number: ";
    cin >> n;

    result = factorial(n);
    cout << "Factorial of " << n << " = " << result;
    return 0;
}

int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```



```

int main() {
    ... ..
    result = factorial(n);
    ... ..
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

```

Diagram illustrating the recursive calls for calculating the factorial of 4 (n=4):

- n = 4:** The function calls `factorial(3)`. The result of `factorial(3)` is  $3 * 2 = 6$ , which is returned to the `n = 4` call. The final result for `n = 4` is  $4 * 6 = 24$ , which is returned to the `main` function.
- n = 3:** The function calls `factorial(2)`. The result of `factorial(2)` is  $2 * 1 = 2$ , which is returned to the `n = 3` call. The final result for `n = 3` is  $3 * 2 = 6$ , which is returned to the `n = 4` call.
- n = 2:** The function calls `factorial(1)`. The result of `factorial(1)` is 1, which is returned to the `n = 2` call. The final result for `n = 2` is  $2 * 1 = 2$ , which is returned to the `n = 3` call.
- n = 1:** The function returns 1, which is returned to the `n = 2` call.





## C++ Arrays

array is a variable that can store multiple values of the same type.

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

```
#include <iostream>
using namespace std;

int main() {
    int numbers[5];

    cout << "Enter 5 numbers: " << endl;

    // store input from user to array
    for (int i = 0; i < 5; ++i) {
        cin >> numbers[i];
    }

    cout << "The numbers are: ";

    // print array elements
    for (int n = 0; n < 5; ++n) {
        cout << numbers[n] << " ";
    }

    return 0;
}
```



## Multidimensional Arrays

we can create an array of an array, known as a multidimensional array.

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

	Col 1	Col 2	Col 3	Col 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]



## Initialization of two-dimensional array

```
int test[2][3] = { {2, 4, 5}, {9, 0, 19}};
```

	Col 1	Col 2	Col 3
Row 1	2	4	5
Row 2	9	0	19



## Initialization of three-dimensional array

```
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}, {5, 1, 4, 9} }  
};
```

Element 1 = { {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} }

Element 2 = { {13, 4, 56, 3}, {5, 9, 3, 5}, {5, 1, 4, 9} }



## C++ Structures

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types.

The **struct** keyword defines a structure type followed by an identifier (name of the structure).



```
#include <iostream>
using namespace std;

struct Person
{
    char name[50];
    int age;
    float salary;
};

int main()
{
    Person p1;

    cout << "Enter Full name: ";
    cin.get(p1.name, 50);
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;

    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
    cout << "Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;

    return 0;
}
```



# Passing structure to function



```
#include <iostream>
using namespace std;

struct Person
{
    char name[50];
    int age;
    float salary;
};

void displayData(Person); // Function declaration

int main()
{
    Person p;

    cout << "Enter Full name: ";
    cin.get(p.name, 50);
    cout << "Enter age: ";
    cin >> p.age;
    cout << "Enter salary: ";
    cin >> p.salary;

    // Function call with structure variable as an argument
    displayData(p);

    return 0;
}

void displayData(Person p)
{
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout << "Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```



## Enumeration

An enumeration is a user-defined data type that consists of integral constants. To define an enumeration, keyword **enum** is used.

```
#include <iostream>
using namespace std;

enum week { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

int main()
{
    week today;
    today = Wednesday;
    cout << "Day " << today+1;
    return 0;
}
```





## C++ Pointers

pointers are variables that store the memory addresses of other variables.

### Printing Variable Addresses in C++

```
#include <iostream>
using namespace std;

int main()
{
    // declare variables
    int var1 = 3;
    int var2 = 24;
    int var3 = 17;

    // print address of var1
    cout << "Address of var1: " << &var1 << endl;

    // print address of var2
    cout << "Address of var2: " << &var2 << endl;

    // print address of var3
    cout << "Address of var3: " << &var3 << endl;
}
```



```
#include <iostream>
using namespace std;
int main() {
    int var = 5;

    // declare pointer variable
    int* pointVar;

    // store address of var
    pointVar = &var;

    // print value of var
    cout << "var = " << var << endl;

    // print address of var
    cout << "Address of var (&var) = " << &var << endl;
    // print pointer pointVar
    cout << "pointVar = " << pointVar << endl;

    // print the content of the address pointVar points to
    cout << "Content of the address pointed to by pointVar (*pointVar) = " << *pointVar << endl;

    return 0;
}
```

pointVar

0x61ff08

var

5

0x61ff08

points to address of var (&var)



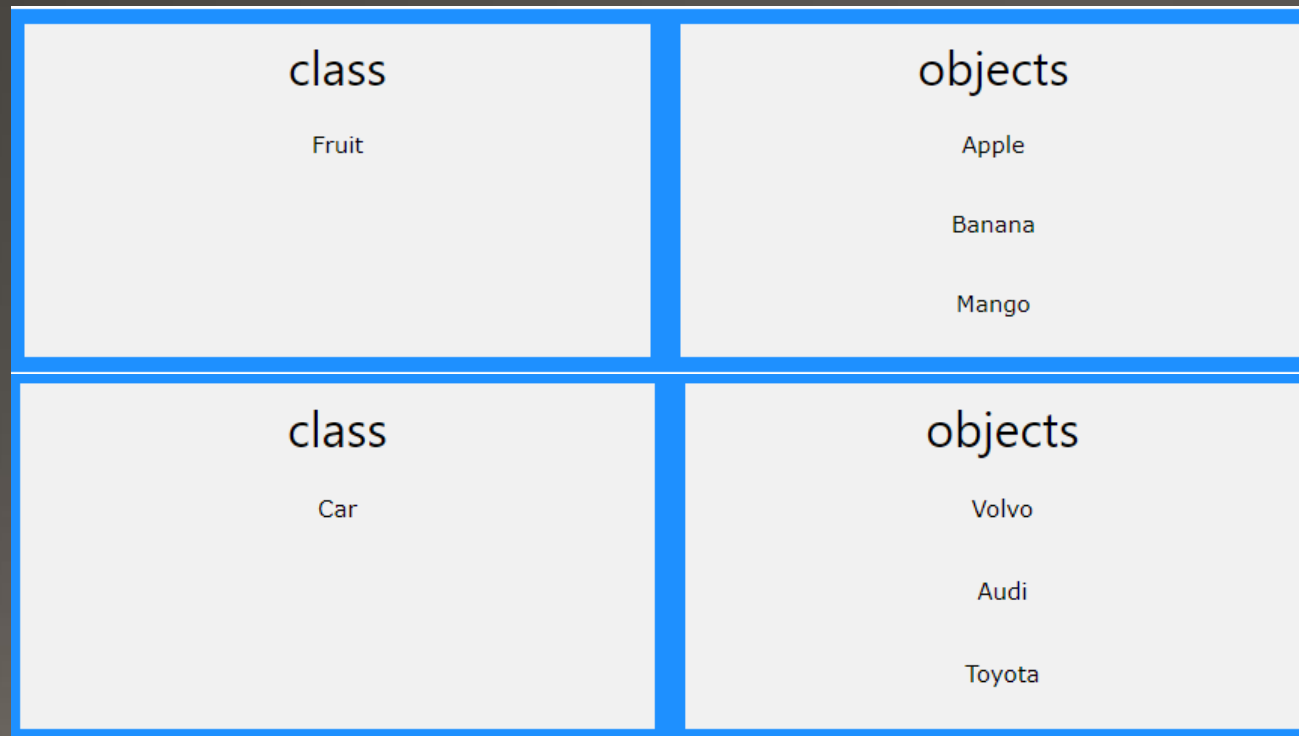
```
#include <iostream>
using namespace std;
int main() {
    int var = 5;
    int* pointVar;

    // store address of var
    pointVar = &var;
    // print var
    cout << "var = " << var << endl;
    // print *pointVar
    cout << "*pointVar = " << *pointVar << endl
    << endl;
    cout << "Changing value of var to 7:" << endl;
    // change value of var to 7
    var = 7;
    // print var
    cout << "var = " << var << endl;
    // print *pointVar
    cout << "*pointVar = " << *pointVar << endl
    << endl;
    cout << "Changing value of *pointVar to 16:" << endl;
    // change value of var to 16
    *pointVar = 16;
    // print var
    cout << "var = " << var << endl;
    // print *pointVar
    cout << "*pointVar = " << *pointVar << endl;
    return 0;
}
```



# C++ Classes and Objects

- A class is an abstract data type similar to '**C structure**'.
- The Class representation of objects and the sets of operations that can be applied to such objects.
- The class consists of Data members and methods.



- A class is defined in C++ using keyword **class** followed by the name of the class.

```
class className {  
    // some data  
    // some functions  
};
```

```
class Room {  
    public:  
        double length;  
        double breadth;  
        double height;  
  
        double calculateArea(){  
            return length * breadth;  
        }  
  
        double calculateVolume(){  
            return length * breadth * height;  
        }  
};
```



```
#include <iostream>
using namespace std;

// create a class
class Room {

public:
    double length;
    double breadth;
    double height;

    double calculateArea() {
        return length * breadth;
    }

    double calculateVolume() {
        return length * breadth * height;
    }
};

int main() {

    // create object of Room class
    Room room1;

    // assign values to data members
    room1.length = 42.5;
    room1.breadth = 30.8;
    room1.height = 19.2;

    // calculate and display the area and volume of the room
    cout << "Area of Room = " << room1.calculateArea() << endl;
    cout << "Volume of Room = " << room1.calculateVolume() << endl;

    return 0;
}
```



```
#include <iostream>
using namespace std;

class Room {

private:
    double length;
    double breadth;
    double height;

public:

    // function to initialise private variables
    void getData(double len, double brth, double hgt) {
        .....
        length = len;
        breadth = brth;
        height = hgt;
    }

    double calculateArea() {
        .....
        return length * breadth;
    }

    double calculateVolume() {
        .....
        return length * breadth * height;
    }
};

int main() {

    // create object of Room class
    Room room1;

    // pass the values of private variables as arguments
    room1.getData(42.5, 30.8, 19.2);

    cout << "Area of Room = " << room1.calculateArea() << endl;
    cout << "Volume of Room = " << room1.calculateVolume() << endl;

    return 0;
}
```



```
#include <iostream>
using namespace std;

class Car {          // The class
public:              // Access specifier
    string brand;    // Attribute
    string model;    // Attribute
    int year;        // Attribute
    Car(string x, string y, int z) { // Constructor with parameters
        brand = x;
        model = y;
        year = z;
    }
};

int main() {
    // Create Car objects and call the constructor with different values
    Car carObj1("BMW", "X5", 1999);
    Car carObj2("Ford", "Mustang", 1969);

    // Print values
    cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
    cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
    return 0;
}
```





```
#include <iostream>
using namespace std;

class Car {          // The class
public:              // Access specifier
    string brand;    // Attribute
    string model;    // Attribute
    int year;        // Attribute
    Car(string x, string y, int z); // Constructor declaration
};

// Constructor definition outside the class
Car::Car(string x, string y, int z) {
    brand = x;
    model = y;
    year = z;
}

int main() {
    // Create Car objects and call the constructor with different values
    Car carObj1("BMW", "X5", 1999);
    Car carObj2("Ford", "Mustang", 1969);

    // Print values
    cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
    cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
    return 0;
}
```



## Files

Class	Description
<code>ofstream</code>	Creates and writes to files
<code>ifstream</code>	Reads from files
<code>fstream</code>	A combination of <code>ofstream</code> and <code>ifstream</code> : creates, reads, and writes to files



```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
    // Create a text file
    ofstream MyWriteFile("filename.txt");

    // Write to the file
    MyWriteFile << "Files can be tricky, but it is fun enough!";

    // Close the file
    MyWriteFile.close();

    // Create a text string, which is used to output the text file
    string myText;

    // Read from the text file
    ifstream MyReadFile("filename.txt");

    // Use a while loop together with the getline() function to read the file line by line
    while (getline (MyReadFile, myText)) {
        // Output the text from the file
        cout << myText;
    }

    // Close the file
    MyReadFile.close();
}
```



## **courses to be finished**

<https://app.pluralsight.com/library/courses/cplusplus-fundamentals-c17/table-of-contents>

<https://www.udacity.com/course/c-for-programmers--ud210>



# Thank You

