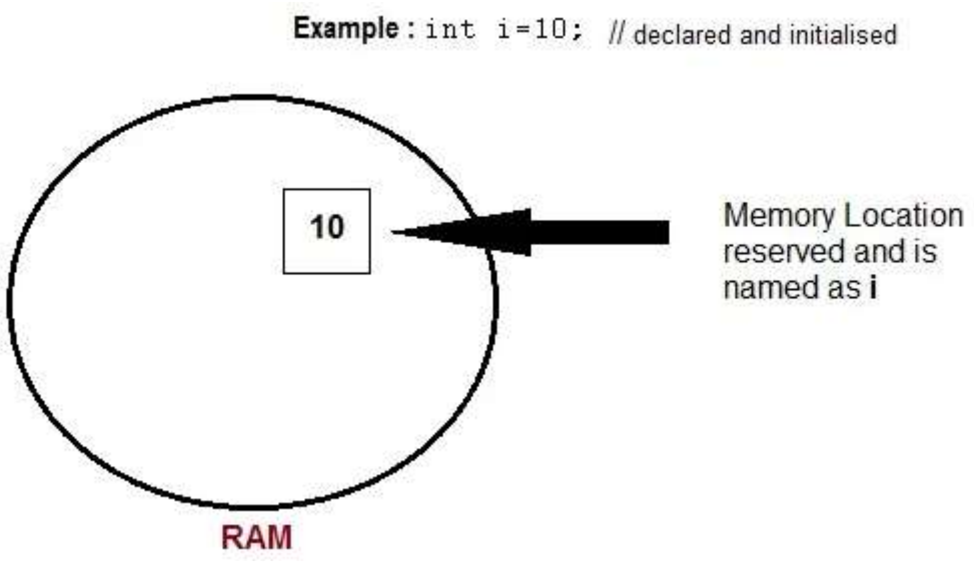
Variables in C++

Variable are used in C++, where we need storage for any value, which will change in program. Variable can be declared in multiple ways each with different memory requirements and functioning. Variable is the name of memory location allocated by the compiler depending upon the datatype of the variable.



Example : `int i=10;` // declared and initialised
10
Memory Location reserved and is named as **i**
RAM

Basic types of Variables

bool	For variable to store boolean values(True or False)
char	For variables to store character types.
Int	for variable with integral values

float and double are also types for variables with large and floating point values

Each variable while declaration must be given a datatype, on which the memory assigned to the variable depends. Following are the basic

Declaration and Initialization

Variable must be declared before they are used. Usually it is preferred to declare them at the starting of the program, but in C++ they can be declared in the middle of program too, but must be done before using them.

For example:

```
int i;      // declared but not initialised

char c;

int i, j, k;  // Multiple declaration
```

Initialization means assigning value to an already declared variable,

```
int i;   // declaration

i = 10;  // initialization
```

Initialization and declaration can be done in one single step also,

```
int i=10;        //initialization and declaration in same step

int i=10, j=11;
```

If a variable is declared and not initialized by default it will hold a garbage value. Also, if a variable is once declared and if try to declare it again, we will get a compile time error.

```
int i,j;

i=10;

j=20;

int j=i+j;   //compile time error, cannot redeclare a variable in same scope
```

}

Scope of Variables

All the variables have their area of functioning, and out of that boundary they don't hold their value, this boundary is called scope of the variable. For most of the cases its between the curly braces,in which variable is declared that a variable exists, not outside it. We will study the storage classes later, but as of now, we can broadly divide variables into two main types,

- Global Variables
- Local variables

Global variables

Global variables are those, which ar once declared and can be used throughout the lifetime of the program by any class or any function. They must be declared outside the main() function. If only declared, they can be assigned different values at different time in program lifetime. But even if they are declared and initialized at the same time outside the main() function, then also they can be assigned any value at any point in the program.

For example: Only declared, not initialized

```
include <iostream>

using namespace std;

int x;              // Global variable declared

int main()

{

   x=10;              // Initialized once

   cout <<"first value of x = "<< x;

   x=20;              // Initialized again

   cout <<"Initialized again with value = "<< x;

}
```

Local Variables

Local variables are the variables which exist only between the curly braces, in which its declared. Outside that they are unavailable and leads to compile time error.

```
include <iostream>

using namespace std;

int main() {

   int i=10;

   if(i<20)      // if condition scope starts

   {

      int n=100;   // Local variable declared and initialized

   }           // if condition scope ends

   cout << n;     // Compile time error, n not available here

}
```

Some special types of variable

There are also some special keywords, to impart unique characteristics to the variables in the program. Following two are mostly used.

1. Final - Once initialized, its value cant be changed.
2. Static - These variables holds their value between function calls.

Example :

```
#include <iostream.h>

using namespace std;

int main() {

   final int i=10;

   static int y=20;

}
```

Tokens

When the compiler is processing the source code of a C++ program, each group of characters separated by white space is called a token. Tokens are the smallest individual units in a program. A C++ program is written using tokens. It has the following tokens:

- Keywords
- Identifiers
- Constants
- Strings
- Operators

Keywords

Keywords(also known as reserved words) have special meanings to the C++ compiler and are always written or typed in short(lower) cases. Keywords are words that the language uses for a special purpose, such as void, int, public, etc. It can't be used for a variable name or function name or any other identifiers.

C++ Keyword							
asm	class	double	Friend	new	return	switch	union
auto	const	else	Goto	operator	short	template	unsigned
break	continue	enum	if	private	signed	this	virtual
case	default	extern	Inline	protected	sizeof	throw	void
catch	delete	float	Int	public	static	try	volatile
char	do	for	Long	register	struct	typedef	while

1. **asm**: To declare that a block of code is to be passed to the assembler.
2. **auto**: A storage class specifier that is used to define objects in a block.
3. **break**: Terminates a switch statement or a loop.

4. **case**: Used specifically within a switch statement to specify a match for the statement’s expression.
5. **catch**: Specifies actions taken when an exception occurs.
6. **char**: Fundamental data type that defines character objects.
7. **class**: To declare a user-defined type that encapsulates data members and operations or member functions.
8. **const**: To define objects whose value will not alter throughout the lifetime of program execution.
9. **continue**:- Transfers control to the start of a loop.
10. **default**:- Handles expression values in a switch statement that are not handled by case.
11. **delete**: Memory deallocation operator.
12. **do**: indicate the start of a do-while statement in which the sub-statement is executed repeatedly until the value of the expression is logical-false.
13. **double:** Fundamental data type used to define a floating-point number.
14. **else:** Used specifically in an if-else statement.
15. **enum:** To declare a user-defined enumeration data type.
16. **extern**: An identifier specified as an extern has an external linkage to the block.
17. **float**:- Fundamental data type used to define a floating-point number.
18. **for**: Indicates the start of a statement to achieve repetitive control.
19. **friend:** A class or operation whose implementation can access the private data members of a class.
20. **goto**: Transfer control to a specified label.
21. **if**: Indicate the start of an if statement to achieve selective control.
22. **inline:** A function specifier that indicates to the compiler that inline substitution of the function body is to be preferred to the usual function call implementation.
23. **int:** Fundamental data type used to define integer objects.
24. **long:** A data type modifier that defines a 32-bit int or an extended double.
25. **new**: Memory allocation operator.
26. **operator:** Overloads a c++ operator with a new declaration.
27. **private**: Declares class members which are not visible outside the class.

28. **protected:** Declares class members which are private except to derived classes
29. **public:** Declares class members who are visible outside the class.
30. **register:** A storage class specifier that is an auto specifier, but which also indicates to the compiler that an object will be frequently used and should therefore be kept in a register.
31. **return**: Returns an object to a function's caller.
32. **short:** A data type modifier that defines a 16-bit int number.
33. **signed:** A data type modifier that indicates an object's sign is to be stored in the high-order bit.
34. **sizeof:** Returns the size of an object in bytes.
35. **static:** The lifetime of an object-defined static exists throughout the lifetime of program execution.
36. **struct:** To declare new types that encapsulate both data and member functions.
37. **switch**: This keyword is used in the "Switch statement".
38. **template**: parameterized or generic type.
39. **this**: A class pointer points to an object or instance of the class.
40. **throw:** Generate an exception.
41. **try**: Indicates the start of a block of exception handlers.
42. **typedef**: Synonym for another integral or user-defined type.
43. **union:** Similar to a structure, struct, in that it can hold different types of data, but a union can hold only one of its members at a given time.
44. **unsigned:** A data type modifier that indicates the high-order bit is to be used for an object.
45. **virtual**: A function specifier that declares a member function of a class that will be redefined by a derived class.
46. **void:** Absent of a type or function parameter list.
47. **volatile**: Define an object which may vary in value in a way that is undetectable to the compiler.
48. **while**: Start of a while statement and end of a do-while statement.

MyAnatomy

Identifiers in C++

All C++ variables must be identified with unique names.

These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume)

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

Example :

```
#include <iostream.h>

using namespace std;

int main() {

        int Number_of_Alpabets = 26;   // Good

        int n = 26; // OK, but not so easy to understand what m actually is

}
```

The general rules for naming variables are:

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case sensitive (myVar and myvar are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Reserved words (like C++ keywords, such as int) cannot be used as names

Constants in C++

When you do not want others (or yourself) to change existing variable values, use the const keyword (this will declare the variable as "constant", which means unchangeable and read-only):

Example 1 :

```
#include <iostream.h>

using namespace std;

int main() {

        int minutesPerHour = 60;

        int m = 60;

}
```

Example 2 :

```
#include <iostream.h>

using namespace std;

int main() {

        const int minutesPerHour = 60;  // myNum will always be 60

        minutesPerHour = 10;  // error: assignment of read-only variable
'minutesPerHour'

}
```