

Department of Cybersecurity Technologies Engineering

PROGRAMMING ESSENTIALS

Lecture 3: C++ Syntax



Dr. Hayder Kareem Algabri

hayder_kareem_algabri@hilla-unc.edu.iq



2024-2025

Lecture 3: C++ Syntax

Contents

1. C++ Definition:	2
2. What is Syntax?	2
Basic Syntax of a C++ Program:	2
3. Token	4
Keywords:	5
Identifier:	8
Constants:	8
Operators In C++ Language	10
1. Assignment Operator:	10
2. Arithmetic Operators:	10
3. Compound Assignment Operators	11
4. Relational Operators:	11
5. Logical Operators	11
6. Bitwise Operators	13
7. Increment and Decrement Operators	15
8. PRECEDENCE OF OPERATORS	16
The sizeof() operator	17
4. Comments:	19
Single Line Comments:	19
Multi-Line Comments:	19
5. C++ Data Types:	19
1.Primitive Data Types:	19
2. Derived Data Types:	20
3.Abstract or User-Defined Data Types:	20
6. C++ Variables:	21
Rules For Declaring Variable	22
7. Input and Output Statements	23

Lecture 3: C++ Syntax

1. C++ Definition:

C++ is a general-purpose, object-oriented programming language. It was developed in 1979 by Bjarne Stroustrup at AT & T Bell Laboratory. It is a high-level programming language & advanced version of C programming language. As Compared to other programming languages like Java, and Python, it is the fastest programming language.

2. What is Syntax?

Syntax refers to the rules and regulations for writing statements in a programming language. They can also be viewed as the grammatical rules defining the structure of a programming language. The C++ language also has its syntax for the functionalities it provides. Different statements have different syntax specifying their usage but C++ programs also have basic syntax rules that are followed throughout all the programs.

Basic Syntax of a C++ Program:**Structure of C++ Program**

	1	<code>#include <iostream></code>	Header File
	2	<code>using namespace std;</code>	Standard Namespace
	3	<code>int main()</code>	Main Function
FUNCTION BODY	4	<code>{</code>	
	5	<code>int num1 = 24;</code> <code>int num2 = 34;</code>	Declaration of Variable
	6	<code>int result = num1 + num2;</code>	Expressions
	7	<code>cout << result << endl;</code>	Output
	8	<code>return 0;</code>	Return Statement
	9	<code>}</code>	

Lecture 3: C++ Syntax**➤ Header File:**

The header files contain the definition of the functions and macros we are using in our program. They are defined on the top of the C++ program. In line #1, we used the `#include <iostream>` statement to tell the compiler to include an iostream header file library which stores the definition of the cin and cout methods that we have used for input and output. `#include` is a preprocessor directive using which we import header files.

Syntax:

```
#include <Library-name >
```

➤ Namespace:

A namespace in C++ is used to provide a scope or a region where we define identifiers. It is used to avoid name conflicts between two identifiers as only unique names can be used as identifiers. In line #2, we have used the `using namespace std` statement for specifying that we will be the standard namespace where all the standard library functions are defined.

Syntax:

```
Using namespace std;
```

➤ Main Function:

In line #3, we defined the main function as `int main()`. The main function is the most important part of any C++ program. The program execution always starts from the main function. All the other functions are called from the main function. In C++, the main function is required to return some value indicating the execution status.

Lecture 3: C++ Syntax

Syntax:

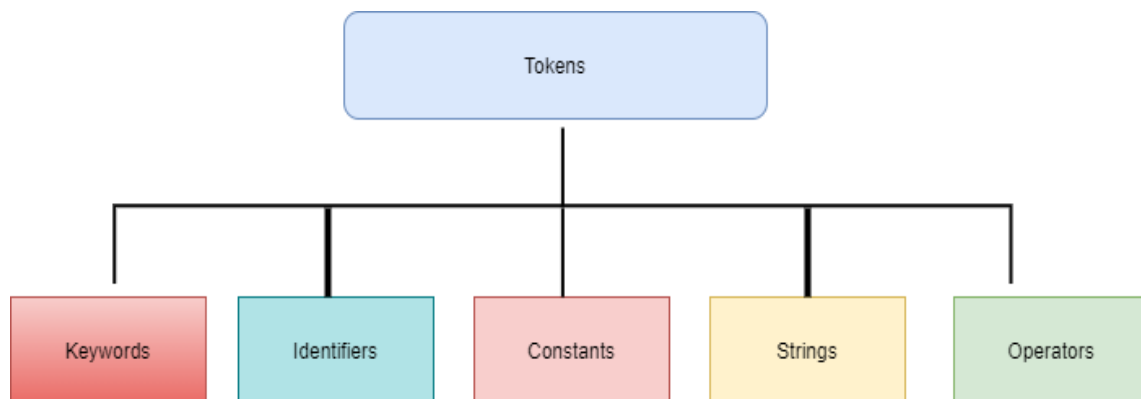
```
int main()  
{ ..... code  
  return 0;  
}
```

Example1:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    cout << "Hello World!";  
    return 0;  
}
```

3. Token

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:



Lecture 3: C++ Syntax

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. The total count of reserved keywords is 95. Below is the table for some commonly used C++ keywords.

C++ Keyword			
asm	double	new	<u>switch</u>
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	<u>if</u>	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

- ❖ **asm**: To declare that a block of code is to be passed to the assembler.
- ❖ **auto**: A storage class specifier that is used to define objects in a block.
- ❖ **break**: Terminates a switch statement or a loop.
- ❖ **case**: Used specifically within a switch statement to specify a match for the statement's expression.
- ❖ **catch**: Specifies actions taken when an exception occurs.

Lecture 3: C++ Syntax

- ❖ **char**: Fundamental data type that defines character objects.
- ❖ **class**: To declare a user-defined type that encapsulates data members and operations or member functions.
- ❖ **const**: To define objects whose value will not alter throughout the lifetime of program execution.
- ❖ **continue**:- Transfers control to the start of a loop.
- ❖ **default**:- Handles expression values in a switch statement that are not handled by case.
- ❖ **delete**: Memory deallocation operator.
- ❖ **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.
- ❖ **double**: Fundamental data type used to define a floating-point number.
- ❖ **else**: Used specifically in an if-else statement.
- ❖ **enum**: To declare a user-defined enumeration data type.
- ❖ **extern**: An identifier specified as an extern has an external linkage to the block.
- ❖ **float**:- Fundamental data type used to define a floating-point number.
- ❖ **for**: Indicates the start of a statement to achieve repetitive control.
- ❖ **friend**: A class or operation whose implementation can access the private data members of a class.
- ❖ **goto**: Transfer control to a specified label.
- ❖ **if**: Indicate the start of an if statement to achieve selective control.
- ❖ **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.
- ❖ **int**: Fundamental data type used to define integer objects.

Lecture 3: C++ Syntax

- ❖ **long:** A data type modifier that defines a 32-bit int or an extended double.
- ❖ **new:** Memory allocation operator.
- ❖ **operator:** Overloads a c++ operator with a new declaration.
- ❖ **private:** Declares class members which are not visible outside the class.
- ❖ **protected:** Declares class members which are private except to derived classes
- ❖ **public:** Declares class members who are visible outside the class.
- ❖ **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.
- ❖ **return:** Returns an object to a function's caller.
- ❖ **short:** A data type modifier that defines a 16-bit int number.
- ❖ **signed:** A data type modifier that indicates an object's sign is to be stored in the high-order bit.
- ❖ **sizeof:** Returns the size of an object in bytes.
- ❖ **static:** The lifetime of an object-defined static exists throughout the lifetime of program execution.
- ❖ **struct:** To declare new types that encapsulate both data and member functions.
- ❖ **switch:** This keyword is used in the "Switch statement".
- ❖ **template:** parameterized or generic type.
- ❖ **this:** A class pointer points to an object or instance of the class.
- ❖ **throw:** Generate an exception.
- ❖ **try:** Indicates the start of a block of exception handlers.
- ❖ **typedef:** Synonym for another integral or user-defined type.

Lecture 3: C++ Syntax

- ❖ **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.
- ❖ **unsigned:** A data type modifier that indicates the high-order bit is to be used for an object.
- ❖ **virtual:** A function specifier that declares a member function of a class that will be redefined by a derived class.
- ❖ **void:** Absent of a type or function parameter list.
- ❖ **volatile:** Define an object which may vary in value in a way that is undetectable to the compiler.
- ❖ **while:** Start of a while statement and end of a do-while statement.

Identifier:

Identifiers refer to the name of variables, functions, arrays, classes, etc. created by the programmer. They are the fundamental requirement of any language.

Rules for naming identifiers:

- Identifier names cannot start with a digit or any special character.
- A keyword cannot be used as an identifier name.
- Only alphabetic characters, digits, and underscores are permitted.
- The upper case and lower-case letters are distinct. i.e., A and a are different in C++.
- In C++, all the variables must be declared before use.

Constants:

Constants are expressions with a fixed value.

There are two simple ways in C++ to define constants:

- Using **#define** preprocessor.
- Using **const** keyword.

Lecture 3: C++ Syntax

The #define directive can be used to define constants and it is placed after headers files.

Example 1: // defined constants #define: calculate circumference

```
1  #include <iostream>
2  using namespace std;
3  #define PI 3.14159
4  #define NEWLINE '\n'
5  int main()
6  {
7      double r=5.0; // radius
8      double circle;
9      circle = 2 * PI * r;
10     cout << circle;
11     cout << NEWLINE;
12     return 0;
13 }
```

Example 2: // defined constants/Const : calculate circumference

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      const int num1=10,num2=20;
6      int sum;
7      sum= num1+num2;
8      cout<<"The summation of two numbers is"<<endl;
9      cout<<"sum="<<sum<<endl;
10     return 0;
11 }
```

Lecture 3: C++ Syntax

Operators In C++ Language**1. Assignment Operator:**

The basic assignment operator is (=) which is often called equal to. Consider the

following assignments.

```
int x = 5, y = 10, z, w; // Declaration and initialization
```

```
z = x; // assignment
```

```
w = x + y; // assignment
```

```
x=(b=3, b+2); //first assign 3 to variable b and then calculate x
```

```
y=(x = 5)+2 ; // assign the value 5 to x, then assign x+2 to y
```

```
a = b = c = 5; // assign the value 5 to a, b, and c
```

2. Arithmetic Operators:

Arithmetic operators are used to perform the basic arithmetic operations. They are explained in the following table:

Operator	Usage	Examples
+	Used for addition	Sum = a + b
-	Used for subtraction	Difference = a - b
*	Used for multiplication	Product = a * b
/	Used for division	Quotient = a / b
%	This operator is called the remainder or the modulus operator. It is used to find the remainder after the division. This operator cannot be used with floating type variables.	Remainder = a % b

Lecture 3: C++ Syntax

3. Compound Assignment Operators

C++ allows combining the arithmetic operators with assignment operator as in the following table.

Operator	C++ expression	Equivalent expression	Explanation and use
+=	<code>B += 5;</code>	<code>B = B + 5;</code>	<code>int B = 4;</code> <code>B += 5 ; // B=9</code>
-=	<code>C -= 6;</code>	<code>C = C - 6;</code>	<code>int C = 10;</code> <code>C -= 6 ; // C=4</code>
*=	<code>D *= 2;</code>	<code>D = D * 2;</code>	<code>int D = 10;</code> <code>D *= 2; // D=20</code>
/=	<code>E /= 3;</code>	<code>E = E / 3;</code>	<code>int E = 21;</code> <code>E /= 3; // E = 7</code>
%=	<code>F %= 4;</code>	<code>F = F % 4;</code>	<code>int F = 10;</code> <code>F %= 4 ; // F=2</code>

4. Relational Operators:

The relational operators are explained in the following table:

Operator	Usage	Example	Explanation
<	Less than	<code>A < B</code>	A is less than B.
>	Greater than	<code>A > B</code>	A is greater than B.
<=	Less than or equal to	<code>A <= B</code>	A is less than or equal to B.
>=	Greater than or equal to	<code>A >= B</code>	A is greater than or equal to B.
==	Equality	<code>A == B</code>	A equal to B.
!=	Not equal to	<code>A != B</code>	A is not equal to B.

5. Logical Operators

The logical operators are used to combine multiple conditions (logical statements). The following table describes the logical operators:

Lecture 3: C++ Syntax

Operator	Usage	Example
&& (logical AND)	The compound condition is true, if both conditions are true.	((a>b) && (a>c))
(logical OR)	The compound statement is true, if any or both conditions are true.	((a>b) (a>c))
! (logical NOT)	It negates the condition.	!(a>b)

Example 3: program illustrates the application of logical operators.

```

1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int p= 1, q= 0, r=1, s,t,x, y, z;
6      s = p||q;
7      t = !q;
8      x = p&&q;
9      y = (p || q && r||s);
10     z = (!p || !q && !r || s);
11     cout << "s = " <<s << ", t = " <<t << ", x = " <<x << endl;
12     cout << "y = " <<y << ", z = " <<z << endl;
13     return 0;
14 }
```

Lecture 3: C++ Syntax

6. Bitwise Operators

Bitwise operation means convert the number into binary and perform the operation on each bit individually. The bitwise operators are listed in the table below:

Operator	Description	Example of code
&	Bitwise AND	<code>A & B;</code>
	Bitwise OR	<code>A B;</code>
^	Bitwise Exclusive OR	<code>A ^ B;</code>
~	complement	<code>~A;</code>
<<	Shift Left	<code>A<<2; //shift left by 2 places</code>
>>	Shift Right	<code>A>>2; //shift right by 2 places</code>
&=	Bitwise AND assign	<code>A &= B; // A=A&B</code>
=	Bitwise OR assign	<code>A = B; // A=A B</code>
^=	Bitwise XOR assign	<code>A ^= B; // A=A^B</code>
<<=	Bitwise shift left assign	<code>A <<= 2; //Shift left by 2 places and assign to A</code>
>>=	Bitwise shift right assign	<code>A >>= 1; //Shift right by 1 place and assign to A</code>

Lecture 3: C++ Syntax

Example 4: program1 illustrates the application of Bitwise operators.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int A=42, B=12, C=24, D;
6      D = A^B;
7      C <<= 1;
8      A <<=2;
9      B >>=2 ;
10     cout<<"A="<<A<<endl;
11     cout<<"B="<<B<<endl;
12     cout<<"C="<<C<<endl;
13     cout<<"D="<<D;
14 }
```

Example 5: program 2 illustrates the application of Bitwise operators.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int A =20, D , E, F;
6      int B = 18, C = 30;
7      D = C^B;
8      E = A &B ;
9      F = C |A ;
10     cout <<"E="<<E<<endl;
11     cout <<"F="<<F<<endl;
12     cout <<"D="<<D<<endl;
13 }
```

Lecture 3: C++ Syntax

7. Increment and Decrement Operators

They are used for increasing and decreasing a variable by unity. These operators may be placed before or after the variable name as shown in the following table.

Operator	Pre or post	Description
<code>++k</code>	Pre-increment	First increase the value of k by one then evaluate the current statement by taking incremented value.
<code>k++</code>	Post-increment	First use the current value of k to evaluate the current statements then increase k by unity.
<code>--k</code>	Pre-decrement	First decrease the value of k by unity then evaluate the statement.
<code>k--</code>	Post-decrement	First use the current value of k to evaluate the current statements then decrease k by unity.

Pre-increment	Post-increment
<pre>int a= 10, b= 11, c; c=a+ ++b; //c=10+12=22, b=12</pre>	<pre>int a= 10, b= 11, c; c=a+ b++; //c=10+11=21 , b=12</pre>

Example 6: program illustrates the increment and decrement operators.

```

1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a = 6, p = 4, r=3, n =5, A,B,C,K ;
6      A = 6 * ++n ;
7      cout << "A = " << A << "\t n = " << n << endl;
8      K = 5 * a-- ;
9      cout << "K = " << K << "\t a = " << a << endl;
10     B = r++ * r++ ;
11     cout << "B = " << B << "\t r = " << r << endl;
12     C = p-- * p-- ;
13     cout << "C = " << C << "\t p = " << p << endl;
14 }
```

Lecture 3: C++ Syntax**8. PRECEDENCE OF OPERATORS**

Precedence is an important aspect of operators. A list of operators and their precedence are given in the following table:

Operator	Description
()	Highest precedence given to parentheses. In case of several pairs of parentheses without nesting, they are evaluated from left to right.
++ and --	Increment and decrement operators have the precedence over the other arithmetic operators. In case of several such operators, they are evaluated from left to right.
*, /, %	Multiplication, division and remainder operators are evaluated after increment and decrement operators. In case of several such operators, they are evaluated from left to right.
+, -	Plus and minus are evaluated last. If there are several such operators then they are evaluated from left to right.

Lecture 3: C++ Syntax

Example 7: Evaluate the value stored in the variable „result“ in the following expressions using precedence of operators.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a=5, b=20, c=10, d=4, e=7, res1,res2,res3,res4;
6      res1 = a * b + c - d % e;
7      cout << "Result 1 = "<<res1<<endl;
8      res2 = a * ++b + c - d % e;
9      cout << "Result 2 = "<<res2<<endl;
10     res3 = ++a * b + (c - d) % --e;
11     cout << "Result 3 = "<<res3<<endl;
12     res4= a * ( b + c ) - ++d % e;
13     cout << "Result 4 = "<<res4<<endl;
14 }
```

9. The sizeof() operator

This operator accepts one parameter, which can be either a type or a variable itself and returns the size in bytes of that type or object :

int x;

int y=sizeof(x);

a = sizeof (char);

Most of the mathematical functions are declared in the **<math.h>** header file, as shown in the table below.

Lecture 3: C++ Syntax

Function	Description	Example
sin(x)	sine of x (x in radians)	sin(2) returns 0.909297
cos(x)	cosine of x (x in radians)	cos (2) returns -0.416147
tan(x)	tangent of x (x in radians)	tan(2) returns -2.18504
asin(x)	inverse sine of x (x in radians)	asin(0.2) returns 0.201358
acos(x)	inverse cosine of x (x in radians)	acos(0.2) returns 1.36944
atan(x)	inverse tangent of x (x in radians)	atan(0.2) returns 0.197396
exp(x)	exponential of x (base e)	exp(2) returns 7.38906
fabs(x)	absolute value of x	fabs(-2) returns 2.0
log(x)	natural logarithm of x (base e)[Ln(x)]	log(2) returns 0.693147
log10(x)	common logarithm of x (base 10)	Log10(2) returns 0.30103
sqrt(x)	square root of x	sqrt(2) returns 1.41421
pow(x,p)	x to the power p	pow(2,3) returns 8.0
ceil(x)	ceiling of x (rounds up)	ceil(3.141593) returns 4.0
floor(x)	floor of x (rounds down)	floor(3.141593) returns 3.0

Lecture 3: C++ Syntax**4. Comments:**

Comments can be used to explain C++ code, and to make it more readable. It can also be used to prevent execution when testing alternative code. Comments can be singled-lined or multi-lined.

Single Line Comments:

Single-line comments start with two forward slashes (//).

```
// This is a comment  
cout << "Hello World!";
```

Multi-Line Comments:

Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by the compiler:

```
/* The code below will print the words Hello World!  
to the screen, and it is amazing */  
cout << "Hello World!";
```

5. C++ Data Types:

Data Types in C++ are Mainly Divided into 3 Types:

1. Primitive Data Types:

These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char, float, bool, etc. Primitive data types available in C++ are:

- Integer.
- Character
- Boolean
- Floating Point
- Double Floating Point

Lecture 3: C++ Syntax

- Valueless or Void
- Wide Character

2. Derived Data Types:

Derived data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:

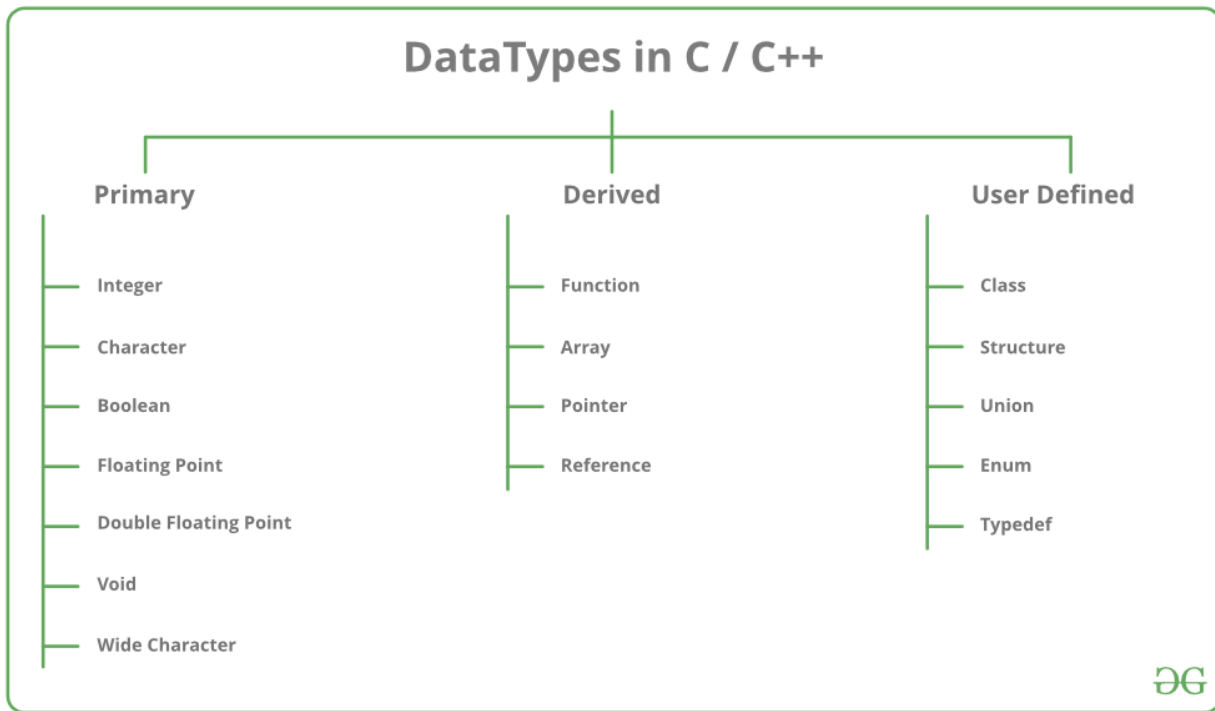
- Function
- Array
- Pointer
- Reference

3. Abstract or User-Defined Data Types:

Abstract or User-Defined data types are defined by the user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:

- Class
- Structure
- Union
- Enumeration
- Typedef defined Datatype

Lecture 3: C++ Syntax



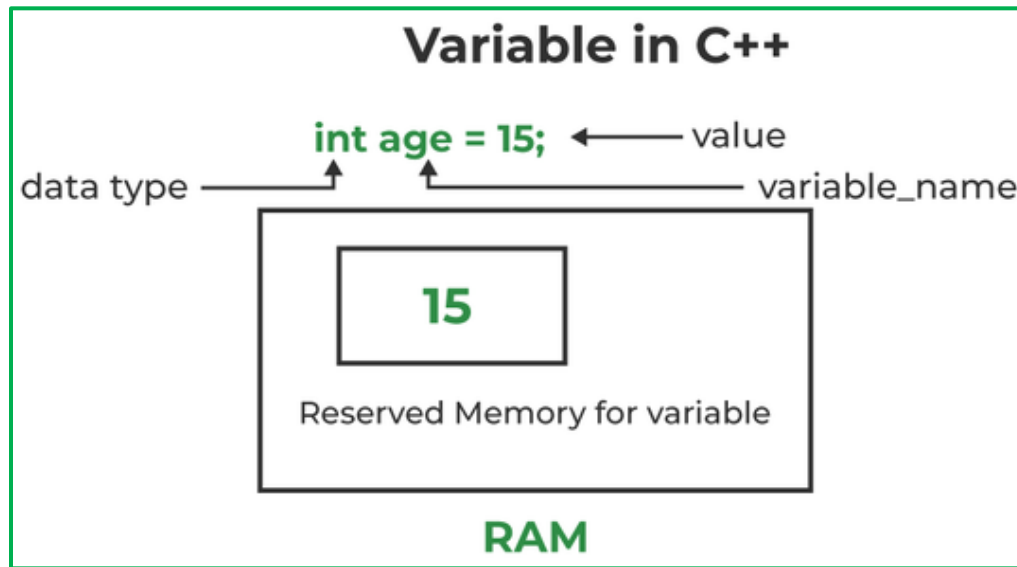
Data Type	Size	Description
<code>boolean</code>	1 byte	Stores true or false values
<code>char</code>	1 byte	Stores a single character/letter/number, or ASCII values
<code>int</code>	2 or 4 bytes	Stores whole numbers, without decimals
<code>float</code>	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits
<code>double</code>	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits

6. C++ Variables:

Variables in C++ is a name given to a memory location. It is the basic unit of storage in a program .

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In C++, all the variables must be declared before use.

Lecture 3: C++ Syntax

**Rules For Declaring Variable**

- The name of the variable contains letters, digits, and underscores.
- The name of the variable is case sensitive (ex Arr and arr both are different variables).
- The name of the variable does not contain any whitespace and special characters (ex #,\$,%,*, etc).
- All the variable names must begin with a letter of the alphabet or an underscore (_).
- We cannot used C++ keyword (ex:float, double, class) as a variable name.

Example

```
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)
```

Lecture 3: C++ Syntax

7. Input and Output Statements

- The input statement has the following syntax:

cin>> variable_name;

cin>> var1>> var2>> var3;

- The output statement has the following syntax:

cout<<variable_name;

cout<<var1<<" "<<var2<<" "<<var3;

cout<<var1<<endl<<var2<<endl<<var3;

Example 8: input first name, last name and print it on screen

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     string fname,lname;
6     cout << "enter the First Name"<<endl;
7     cin>> fname;
8     cout << "enter the Last Name"<<endl;
9     cin>> lname;
10    cout<<"full Name "<<fname <<" "<< lname<<endl;
11    return 0;
12 }
```