CSE 232

Programming with C++

Lecture 3

C++ Keywords, Variables and Data Types

Prepared by

Md. Mijanur Rahman, Prof. Dr.
Dept. of Computer Science and Engineering
Jatiya Kabi Kazi Nazrul Islam University, Bangladesh
Email: mijan@jkkniu.edu.bd
Web: www.mijanrahman.com

Contents

C++ Keywords, Variables and Data Types

C++ Identifiers

- In C++ programming language, identifiers are the unique names assigned to variables, functions, classes, structs, or other entities within the program.

- For example:

```
int num = 100;
```

- **num** is an identifier.

Rules to Name of an Identifier in C++

- We can use any word as an identifier as long as it follows the following rules:
  1. An identifier can consist of letters (A-Z or a-z), digits (0-9), and underscores (_). Special characters and spaces are not allowed.
  2. An identifier can only begin with a letter or an underscore only.
  3. C++ has reserved keywords that cannot be used as identifiers since they have predefined meanings in the language.
  4. Identifier must be unique in its namespace.

- Additionally, C++ is a case-sensitive language so the identifier such as Num and num are treated as different.

C++ Keywords

- Keywords (reserved words) have special meanings to the C++ compiler and are always written or typed in 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 keywords is 95.

Examples of Keywords:

asm	double	new	switch
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	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

How keywords are different from identifiers?

- There are some main properties of keywords that distinguish keywords from identifiers:

Keywords

- Keywords are predefined/reserved words
- Keywords always start in lowercase
- It defines the type of entity.
- A keyword contains only alphabetical characters,
- It should be lowercase.
- No special symbols or punctuations are used in keywords and identifiers.

Identifiers

- Identifiers are the values used to define different programming items like a variable, integers, structures, and unions.
- Identifiers can start with an uppercase letter as well as a lowercase letter.
- It classifies the name of the entity.
- An identifier can consist of alphabetical characters, digits, and underscores.
- It can be both upper and lowercase.
- No special symbols or punctuations are used in keywords and identifiers. The only underscore can be used in an identifier.

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.

C++ Variables

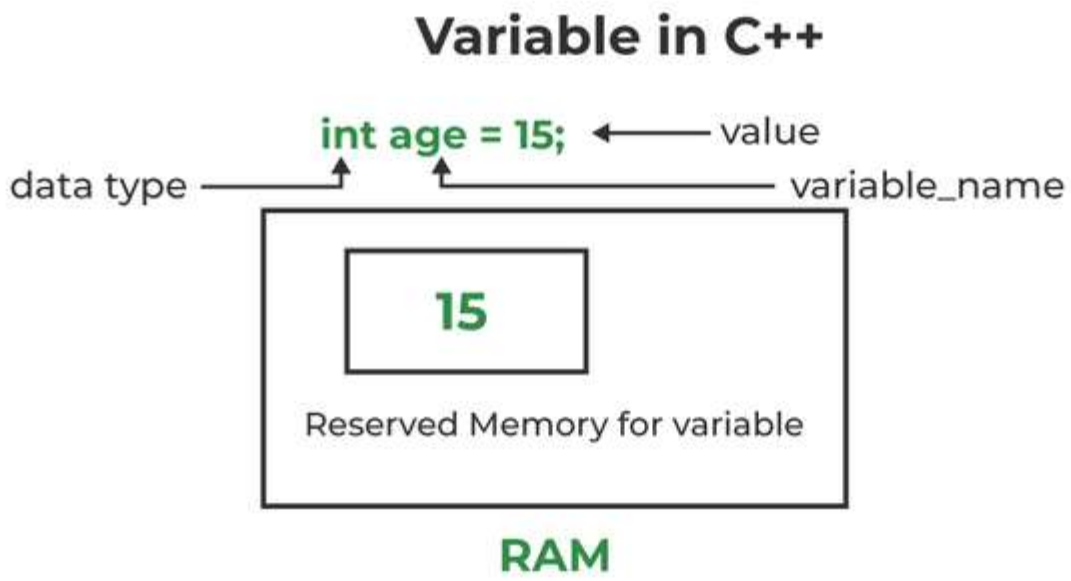
- **How to Declare Variables?**
- A typical variable declaration is of the form:

```
// Declaring a single variable
type variable_name;

// Declaring multiple variables:
type variable1_name, variable2_name, variable3_name;
```

- A variable name can consist of alphabets (both upper and lower case), numbers, and the underscore '_' character. However, the name must not start with a number.

C++ Variables

- How to Declare Variables?

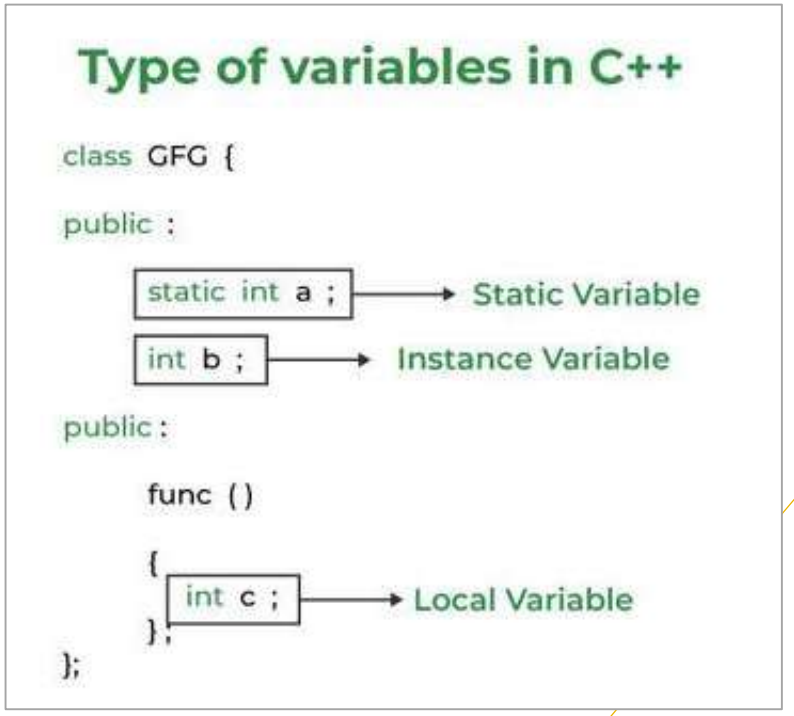


C++ Variables

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

Types of C++ Variables

- There are three types of variables based on the scope of variables in C++
  - Local Variables
  - Instance Variables
  - Static Variables



Types of C++ Variables

- **Local Variables:** A variable defined within a block or method or constructor is called a local variable.
  - These variables are created when entered into the block or the function is called and destroyed after exiting from the block or when the call returns from the function.
  - The scope of these variables exists only within the block in which the variable is declared. i.e. we can access this variable only within that block.
  - Initialization of local variables is not mandatory, but it is highly recommended to ensure they have a defined value before use.

Types of C++ Variables

- **Instance Variables:** Instance variables are non-static variables and are declared in a class, outside any method, constructor, or block.
  - As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
  - Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.
  - Initialization of Instance Variable is not Mandatory.
  - Instance Variable can be accessed only by creating objects.

Types of C++ Variables

- **Static Variables:** Static variables are also known as Class variables.
  - These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
  - Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
  - Static variables are created at the start of program execution and destroyed automatically when execution ends.
  - Initialization of Static Variable is not Mandatory. Its default value is 0
  - If we access the static variable like the Instance variable (through an object), the compiler will show the warning message and it won't halt the program. The compiler will replace the object name with the class name automatically.
  - If we access the static variable without the class name, the Compiler will automatically append the class name.

Types of C++ Variables

- The syntax for static and instance variables:

```
class Example
{
   static int a; // static variable
   int b;       // instance variable
}
```

Scope of Variables in C++

- In general, the scope is defined as the extent up to which something can be worked with.
- In programming, the scope of a variable is defined as the extent of the program code within which the variable can be accessed or declared or worked with.
- There are mainly two types of variable scopes:
  - Local Variables
  - Global Variables

Scope of Variables in C++

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

Global Variable

```
// global variable
int global = 5;

// main function
int main()
{
    // local variable with same
    // name as that of global variable
    int global = 2;

    cout << global << endl;
}
```

Local variable

Constants in C++

- In C++, constants are values that remain fixed throughout the execution of a program.

- Constants in C++ refer to variables with fixed values that cannot be changed.
- Once they are defined in the program, they remain constant throughout the execution of the program.
- They are generally stored as read-only tokens in the code segment of the memory of the program.
- They can be of any available data type in C++ such as int, char, string, etc.

Constants in C++

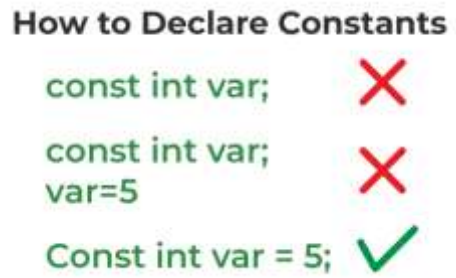
- **How to Define Constants in C++?**
- We can define the constants in C++ using three ways:
  1. Using const Keyword
  2. Using constexpr Keyword
  3. Using #define Preprocessor

Constants in C++

- **Constant using const Keyword**
- Defining constants using const keyword is one of the older methods that C++ inherited from C language. In this method, we add the const keyword in the variable definition as shown below:

- Syntax for const Keyword

  const DATATYPE variable_name = value;



Constants in C++

- **Example of using const Keyword:**

```
ERROR!
/tmp/KYRs0OViJZ.cpp: In function 'int main()':
/tmp/KYRs0OViJZ.cpp:19:14: error: assignment of read-only variable 'cons'
   19 |         cons = 0;
      |         ~~~~~^~~


=== Code Exited With Errors ===
```

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

int main()
{
    // declaring a variable
    int var = 10;

    // declaring a constant variable
    const int cons = 24;

    cout << "Initial Value:" << endl;
    cout << "var: " << var << endl;
    cout << "cons: " << cons << endl;

    // changing the value of both the constants
    var = 24;
    cons = 0;

    cout << "Final Value:" << endl;
    cout << "var: " << var << endl;
    cout << "cons: " << cons << endl;

    return 0;
}
```

Constants in C++

- **Constants using constexpr Keyword**
- The **constexpr** keyword is similar to const keyword and is also used to declare constants in C++.
- But the major difference is that the constexpr constants are initialized at compiler time, which is why their value must be known at the compile time.
- On the other hand, const keyword constants can be initialized at runtime and compile time.

- Syntax:

```
constexpr DATATYPE variable_name = value ;
```

Constants in C++

- **Example of using constexpr Keyword:**

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

int main()
{
    // defining constant
    int constexpr hoursIn_day = 24;

    // printing value
    cout << "Total hours in a day: " << hoursIn_day;
}
```

Constants in C++

- **Constants using #define Preprocessor**
- We can also define constants using the #define.
- Constants created using the #define preprocessor are called “macro constants”.
- Unlike the other methods, constants defined using this method simply work as an alias for their value which is substituted during preprocessing.
- Syntax:

```
#define MACRO_NAME replacement_value
```

- We use the #define directive to create a macro. First, we start with the #define directive, followed by the name we want to give to our macro. Then, we provide a replacement value for the macro.

Constants in C++

- **Example of using #define Preprocessor:**

```
1  #include <iostream>
2  using namespace std;
3
4  // using #define to create a macro
5  #define Side 6
6
7  int main()
8  {
9      // using constant
10     const double area = Side * Side;
11
12     cout << "The area of square with side " <<Side<<" is ";
13     cout << area;
14
15     return 0;
16 }
```

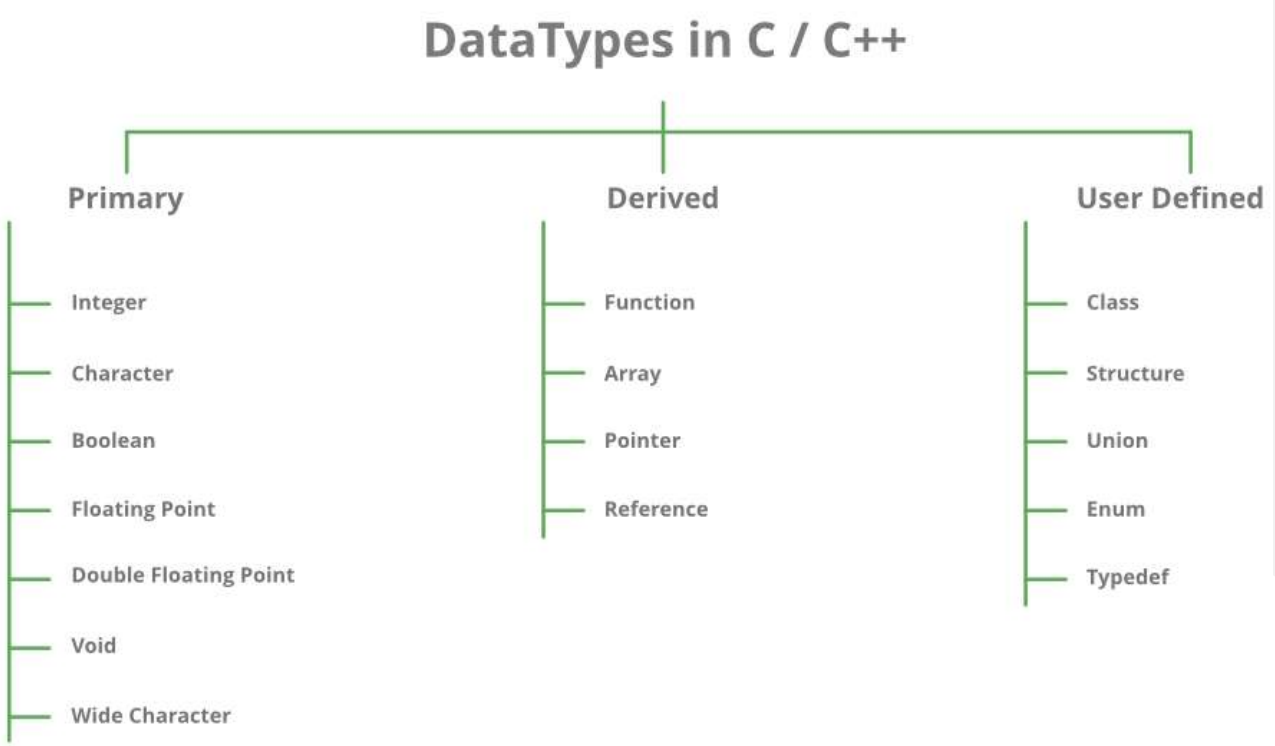
C++ Data Types

- All variables use data type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data they can store.
- Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data type with which it is declared. Every data type requires a different amount of memory.
- C++ supports a wide variety of data types and the programmer can select the data type appropriate to the needs of the application. Data types specify the size and types of values to be stored.

- C++ supports the following data types:
    1. Primary or Built-in or Fundamental data type
    2. Derived data types
    3. User-defined data types

C++ Data Types

- C++ supports the following data types:
  1. Primary or Built-in or Fundamental data type
  2. Derived data types
  3. User-defined data types



C++ Data Types

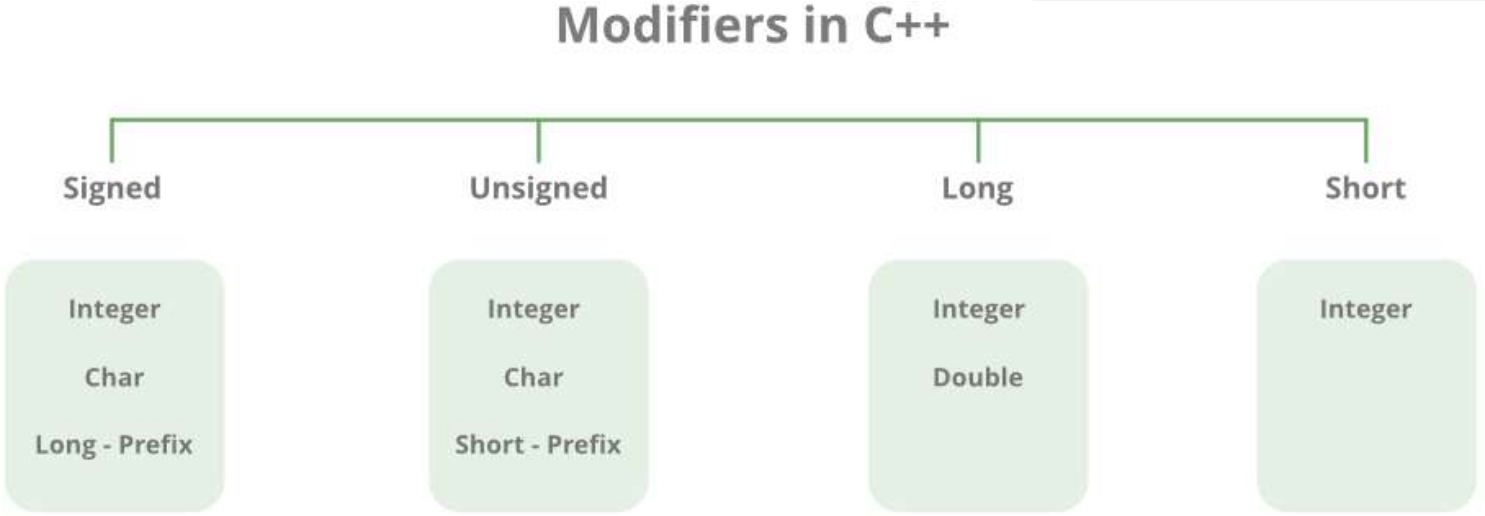
- The size of variables:

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

int main()
{
    cout << "Size of char : " << sizeof(char) << endl;
    cout << "Size of int : " << sizeof(int) << endl;

    cout << "Size of long : " << sizeof(long) << endl;
    cout << "Size of float : " << sizeof(float) << endl;

    cout << "Size of double : " << sizeof(double) << endl;

    return 0;
}
```

```
Size of char : 1
Size of int : 4
Size of long : 8
Size of float : 4
Size of double : 8
```

C++ Data Type Modifiers

- As the name suggests, datatype modifiers are used with built-in data types to modify the length of data that a particular data type can hold.
- Data type modifiers available in C++ are:
  - Signed
  - Unsigned
  - Short
  - Long



C++ Data Type Modifiers

Data Type	Size (in bytes)	Range
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	-3.4×10^38 to 3.4×10^38
double	8	-1.7×10^308 to1.7×10^308
long double	12	-1.1×10^4932 to1.1×10^4932
wchar_t	2 or 4	1 wide character

Type Conversion in C++

- A type cast is basically a conversion from one type to another. There are two types of type conversion:
- **Implicit Type Conversion** Also known as ‘automatic type conversion’.
  - Done by the compiler on its own, without any external trigger from the user.
  - Generally, takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
  - All the data types of the variables are upgraded to the data type of the variable with largest data type.

    bool -> char -> short int -> int ->
    unsigned int -> long -> unsigned ->
    long long -> float -> double -> long double
  - It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Type Conversion in C++

- **Example of Implicit Type Conversion:**

```cpp
// An example of implicit conversion
#include <iostream>
using namespace std;

int main()
{
    int x = 10; // integer x
    char y = 'a'; // character c
    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;
    // x is implicitly converted to float
    float z = x + 1.0;
    cout << "x = " << x << endl
        << "y = " << y << endl
        << "z = " << z << endl;
    return 0;
}
```

```
x = 107
y = a
z = 108
```

Type Conversion in C++

- **Explicit Type Conversion:** This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.
- This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

  (type) expression
- where type indicates the data type to which the final result is converted.

Type Conversion in C++

- **Example of Explicit Type Conversion:**

```cpp
// Explicit type casting

#include <iostream>
using namespace std;

int main()
{
    double x = 10.25;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    cout << "Sum = " << sum;

    return 0;
}
```

```
Sum = 11

=== Code Execution Successful ===
```

Literals In C++

- In C++ programming language, we use literals to represent fixed values. C++ supports various types of literals including integer literals, floating-point literals, character literals, and string literals.
- Literals are fundamental elements used to represent constant values used in C++ programming language. These constants can include numbers, characters, strings, and more. Understanding and using the literals is essential in C++ for data assignment, calculations, and data representation. They are generally present as the right operand in the assignment operation.
- There are five types of literals in C++ which are:
  - Integer Literals
  - Floating Point Literals
  - Character Literals
  - String Literals
  - Boolean Literals

Literals In C++

- There are five types of literals in C++ which are:
  - Integer Literals
  - Floating Point Literals
  - Character Literals
  - String Literals
  - Boolean Literals

Lecture 3

C++ Keywords, Variables and Data Types

THE END