Lecture 03

M M Imran

```
/*
* Multiple line
* comment
*/
#include<iostream>

//Single line comment
using namespace std;

//This is where the execution of program begins
int main()
{
  // displays Hello World! on screen
  cout<<"Hello World!";

  return 0;
}
```

#include<iostream> – This statements tells the compiler to include iostream file. This file contains pre defined input/output functions that we can use in our program.

using namespace std; – A namespace is like a region, where defined functions, variables etc are stored. If we don't want to use this line of code, we can use the things in this namespace like this. std::cout, std::endl.

Computer needs to know the definition of the code of the functionalities. It is defined in the header file. So **header file** needs to be included.

Namespace is needed because if a functionalities like cout is used, but not defined in the current scope computer needs to know where to check. So **namespace** needs to be included.

Preprocessor directive #include

- A preprocessor directive is used in C++ to specify commands to be run before code is compiled.
- A preprocessor directive appears on one or more lines and begins with the hash tag (#).
- A library is a collection of code written to perform a related set of tasks, and available to any application.
- Every programming language provides multiple libraries and a way to incorporate them into an application.
- Preprocessor directive #include connects a library to an application.
- Some libraries have two versions, a C version and a C++ version. Generally, the C version has file extension "**.h**" while the C++ version begins with "c" and has no file extension.

Example of includes

```
#include <cstdlib>  // For several general-purpose functions
#include <fstream>  // For file handling
#include <iomanip>  // For formatted output
#include <iostream>  // For cin, cout, and system
#include <string>  // For string data type
```

Variable

- A variable is a named spot in memory used to hold a value.
- A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.
- A variable has at least three characteristics:
  - Name
  - Data type
  - Value

```
data_type variable1_name = value1;
data_type variable2_name = value2, variable3_name = value3;
```

Declaring Variables

```
#include<iostream>

using namespace std;

int main()
{
  int num1=100, num2=200;
  int num3 = 300;
  int num4;
  return 0;
}
```

Rules for naming a variable

- A variable name can only have alphabets, numbers, and the underscore _.
- A variable name cannot begin with a number.
- It is a preferred practice to begin variable names with a lowercase character. For example, name is preferable to Name.
- A variable name cannot be a keyword. For example, int is a keyword that is used to denote integers.
- A variable name can start with an underscore. However, it's not considered a good practice.

Types of variables based on their scope

```
int main {

//Some code

}
```

Any variable declared inside these curly braces have scope limited within these curly braces, if you declare a variable in main() function and try to use that variable outside main() function then you will get compilation error.

Now that we have understood what is scope. Lets move on to the types of variables based on the scope.

1. Global variable
2. Local variable

Global Variable

A variable declared outside of any function (including main as well) is called global variable.

Global variables have their scope throughout the program, they can be accessed anywhere in the program.

```
#include <iostream>
using namespace std;
// This is a global variable
char myVar = 'A';
int main()
{
   cout <<"Value of myVar: "<< myVar<<endl;
   myVar='Z';
   cout <<"Value of myVar: "<< myVar;
   return 0;
}
```

Output:

```
Value of myVar: A
Value of myVar: Z
```

Local Variable

Local variables are declared inside the braces of any user defined function, main function, loops or any control statements (if, if-else etc) and have their scope limited inside those braces.

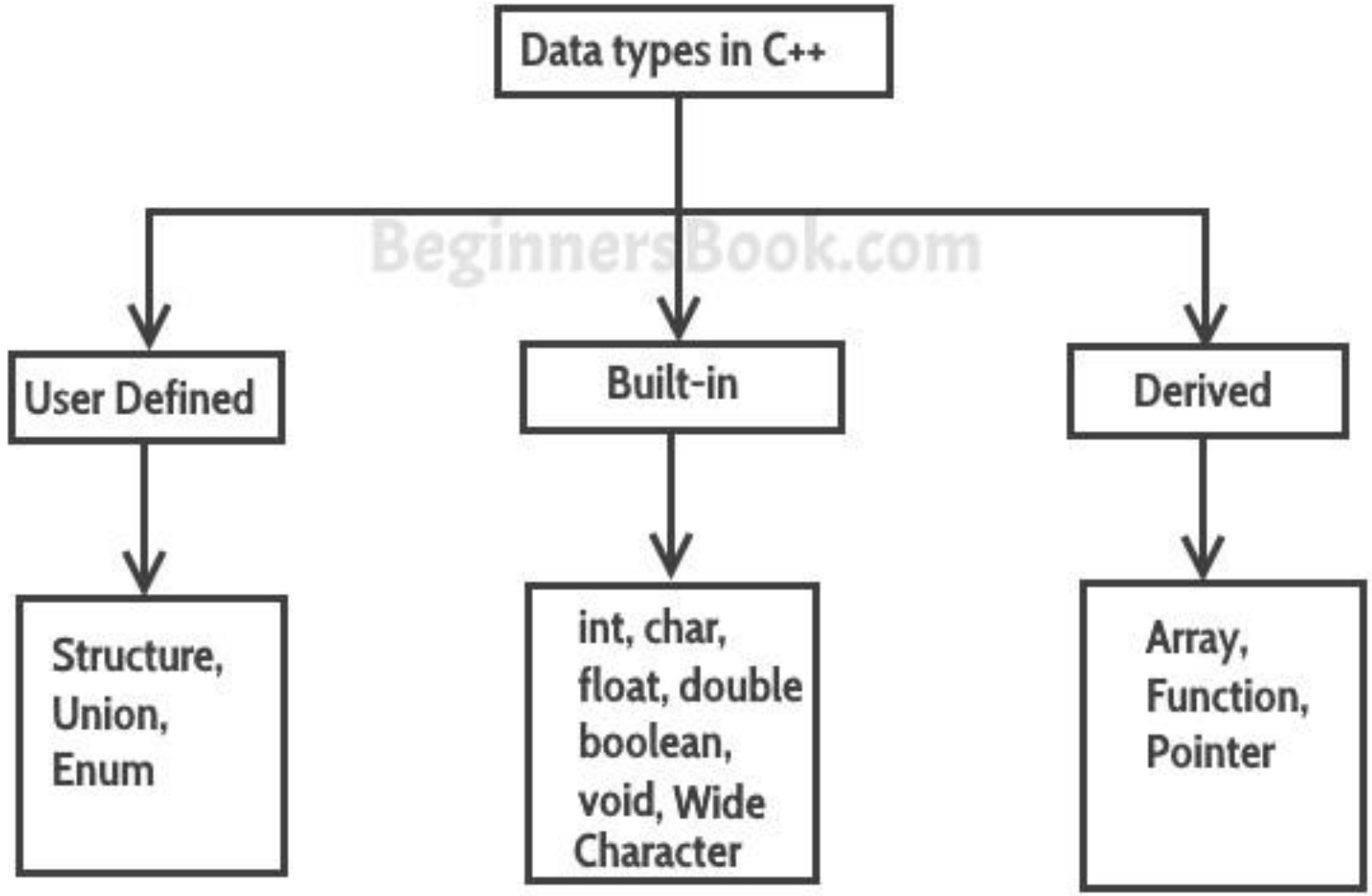
This program will generate compile time error as **myVar** is outside of it’s scope.

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

char myFuncn() {
// This is a local variable
char myVar = 'A';
}
int main()
{
   cout <<"Value of myVar: "<< myVar<<endl;
   myVar='Z';
   cout <<"Value of myVar: "<< myVar;
   return 0;
}
```

Data Types

- Data types define the type of data a variable can hold, for example an integer variable can hold integer data, a character type variable can hold character data etc.
- Every variable has a data type. This is so that humans and the computer know how to handle and interpret it.
- Every programming language provides a set of data types.
- Data types in C++ are categorised in three groups:
  - Built-in
  - User-defined
  - Derived.


Data types in C++
BeginnersBook.com
User Defined
Built-in
Derived
Structure,
Union,
Enum
int, char,
float, double
boolean,
void, Wide
Character
Array,
Function,
Pointer

Built-in 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
wchar_t	Wide Character	2
bool	Boolean	1
void	Empty	0

int

- The int keyword is used to indicate integers.
- Its size is usually 4 bytes. Meaning, it can store values from -2147483648 to 2147483647.

int salary = 85000;

float and double

- **float** and **double** are used to store floating-point numbers (decimals and exponentials).
- The size of float is 4 bytes and the size of double is 8 bytes. Hence, double has two times the precision of float.

```
float area = 64.74;
double volume = 134.64534;
double distance = 45E12    // 45E12 is equal to 45*10^12
```

char

- Keyword char is used for characters.
- Its size is 1 byte.
- Characters in C++ are enclosed inside single quotes ' ‘.

```
char test = 'h';
```

wchar_t

- Wide character wchar_t is similar to the char data type, except its size is 2 bytes instead of 1.
- It is used to represent characters that require more memory to represent them than a single char.

```
wchar_t test = L 'ם '    // Storing Hebrew character;
```

bool

- The bool data type has one of two possible values: true or false.
- Booleans are used in conditional statements and loops.

```
bool cond = false;
```

void

- The void keyword indicates an absence of data. It means "nothing" or "no value".
- We will use void when we learn about functions and pointers.
- We cannot declare variables of the void type.

Type Modifiers

We can further modify some of the fundamental data types by using type modifiers. There are 4 type modifiers in C++. They are:

1. signed
2. unsigned
3. short
4. Long

These modifiers are applicable for **int, double, char** data types

Modified Data Types List

Data Type	Size (in Bytes)	Meaning
signed int	4	used for integers (equivalent to int)
unsigned int	4	can only store positive integers
short	2	used for small integers (range -32768 to 32767)
unsigned short	2	used for small positive integers (range 0 to 65,535)
long	at least 4	used for large integers (equivalent to long int)
unsigned long	4	used for large positive integers or 0 (equivalent to unsigned long int)
long long	8	used for very large integers (equivalent to long long int).
unsigned long long	8	used for very large positive integers or 0 (equivalent to unsigned long long int)
long double	12	used for large floating-point numbers
signed char	1	used for characters (guaranteed range -127 to 127)
unsigned char	1	used for characters (range 0 to 255)

Example

```
long b = 4523232;
long int c = 2345342;
long double d = 233434.56343;
short d = 3434233; // Error! out of range
unsigned int a = -5;    // Error! can only store positive numbers or 0
```

User-defined data types

We have three types of user-defined data types in C++

1. struct
2. union
3. enum

We will cover them later.

Derived data types

We have three types of derived-defined data types in C++

1. Array
2. Function
3. Pointer

We will cover them later.

```
//*************************************************************
// Given the length and width of a rectangle, this C++ program
// computes and outputs the perimeter and area of the rectangle.
//*************************************************************

#include <iostream>

using namespace std;

int main()
{
    double length;
    double width;
    double area;
    double perimeter;

    cout << "Program to compute and output the perimeter and "
         << "area of a rectangle." << endl;

    length = 6.0;

    width = 4.0;
    perimeter = 2 * (length + width);

    area = length * width;

    cout << "Length = " << length << endl;
    cout << "Width =  " << width << endl;
    cout << "Perimeter = " << perimeter << endl;
    cout << "Area = " << area << endl;

    return 0;
}
```

Comments

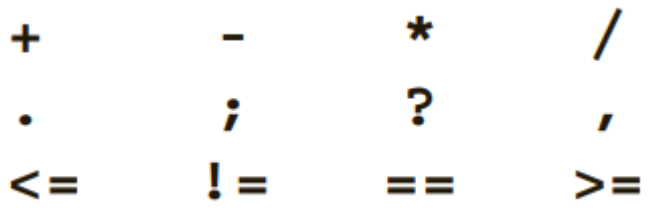
Variable declarations. A statement such as
`double length;`
instructs the system to allocate memory space and name it **length**.

Assignment statement. This statement instructs the system to store **6.0** in the memory space **length**.

Assignment statement.
This statement instructs the system to evaluate the expression **length * width** and store the result in the memory space **area**.

Output statements. An output statement instructs the system to display results.

Special Symbols



- The smallest individual unit of a program written in any language is called a token.
- Tokens are divided into special symbols, word symbols, and identifiers
- The first row includes mathematical symbols for addition, subtraction, multiplication, and division.
- The second row consists of punctuation marks taken from English grammar. Note that the comma is also a special symbol.
- Note that a blank, which is not shown above, is also a special symbol.
- The third row consists of tokens made up of two characters that are regarded as a single symbol.
- No character can come between the two characters in the token, not even a blank

Reserved Words (Keywords)

- A reserved word is a word that cannot be used as an identifier, such as the name of a variable, function, or label – it is "reserved from use".
- There are a total of 95 reserved words in C++
- Sample keywords

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Operators

- An operator is simply a symbol that is used to perform operations.
- Operators are symbols that perform operations on variables and values. For example, + is an operator used for addition, while - is an operator used for subtraction.
- An operator works on two or more operands and produce an output. For example: 3+4+5 here + operator works on three operands and produce 12 as output.

Types of 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

Arithmetic operators are used to perform arithmetic operations on variables and data.

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

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

int main() {
    int a, b;
    a = 7;
    b = 2;

    // 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;

    return 0;
}
```

Assignment Operators

In C++, assignment operators are used to assign values to variables.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

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

int main() {
    int a, b;

    // 2 is assigned to a
    a = 2;

    // 7 is assigned to b
    b = 7;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "\nAfter a += b;" << endl;

    // assigning the sum of a and b to a
    a += b;  // a = a +b
    cout << "a = " << a << endl;

    return 0;
}
```

Relational Operators

A relational operator is used to check the relationship between two operands.

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

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

int main() {
    int a, b;
    a = 3;
    b = 5;
    bool result;

    result = (a == b);   // false
    cout << "3 == 5 is " << result << endl;

    result = (a != b);  // true
    cout << "3 != 5 is " << result << endl;

    result = a > b;   // false
    cout << "3 > 5 is " << result << endl;

    result = a < b;   // true
    cout << "3 < 5 is " << result << endl;

    result = a >= b;  // false
    cout << "3 >= 5 is " << result << endl;

    result = a <= b;  // true
    cout << "3 <= 5 is " << result << endl;

    return 0;
}
```

Logical Operators

Logical operators are used to check whether an expression is true or false. If the expression is true, it returns 1 whereas if the expression is false, it returns 0.

Operator	Example	Meaning
&&	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.

```cpp
int main() {
    bool result;

    result = (3 != 5) && (3 < 5);     // true
    cout << "(3 != 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 < 5);    // false
    cout << "(3 == 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 > 5);    // false
    cout << "(3 == 5) && (3 > 5) is " << result << endl;

    result = (3 != 5) || (3 < 5);    // true
    cout << "(3 != 5) || (3 < 5) is " << result << endl;

    result = (3 != 5) || (3 > 5);    // true
    cout << "(3 != 5) || (3 > 5) is " << result << endl;

    result = (3 == 5) || (3 > 5);    // false
    cout << "(3 == 5) || (3 > 5) is " << result << endl;

    result = !(5 == 2);    // true
    cout << "!(5 == 2) is " << result << endl;

    result = !(5 == 5);    // false
    cout << "!(5 == 5) is " << result << endl;

    return 0;
}
```

Bitwise Operators

In C++, bitwise operators are used to perform operations on individual bits. They can only be used alongside char and int data types.

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

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

int main() {
    // declare variables
    int a = 12, b = 25;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "a & b = " << (a & b) << endl;

    return 0;
}
```

```
12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

//Bitwise AND Operation of 12 and 25

      00001100
&     00011001
      ________
      00001000  = 8 (In decimal)
```

Other C++ Operators

Operator	Description	Example
sizeof	returns the size of data type	sizeof(int); // 4
?:	returns value based on the condition	string result = (5 > 0) ? "even" : "odd"; // "even"
&	represents memory address of the operand	# // address of num
.	accesses members of struct variables or class objects	s1.marks = 92;
->	used with pointers to access the class or struct variables	ptr->marks = 92;
<<	prints the output value	cout << 5;
>>	gets the input value	cin >> num;

Thanks !!