Lecture 03

C++ Programming Basics:

- Namespace Fundamentals,
- using,
- The standard Namespace,
- Data types,
- Input with cin, Output Using cout,
- Type Conversion: Automatic Conversions, Casts

Namespace Fundamentals: :

- In C++, a namespace is a collection of related names or identifiers (functions, class, variables) which helps to separate these identifiers from similar identifiers in other namespaces or the global namespace.
- The identifiers of the C++ standard library are defined in a namespace called std.
- In order to use any identifier belonging to the standard library, we need to specify that it belongs to the std namespace. One way to do this is by using the scope resolution operator ::
- e.g.std::cout << "Hello World!";

Here, we have used the code std:: before cout. This tells the C++ compiler that the cout object we are using belongs to the std namespace.

Namespace Fundamentals:

C++ std Identifiers:

- All the standard library identifiers provided by the standard header files like <iostream>, <string>, <vector>, etc. are declared in the std namespace.

- For example, identifiers cin and cout are defined inside the standard header file <iostream> of the namespace std.

Namespace Fundamentals:

Utilizing std Identifiers:

- We can utilize identifiers of the std namespace in our program with:

- the :: operator
- the using declaration
- the using directive

Namespace Fundamentals:

std Namespace Using :: Operator

The first way we access identifiers in the std namespace is by directly qualifying the identifier with the prefix std::. Here,

std is the C++ standard library namespace

:: is the scope resolution operator

Namespace Fundamentals:

```
#include <iostream>

int main() {

  std::string first_name;

  std::cout << "Enter your first name: ";
  std::cin >> first_name;
  std::cout << "Hello " << first_name << "!" << std::endl;
  std::cout << "Welcome!";

  return 0;
}
```

Namespace Fundamentals:

std Namespace With using Declaration:

We can bring selected identifiers to the current scope with the help of the using declaration. To do this, we utilize the using keyword.

By doing this, we won't need to prefix the specified identifiers with std::. For example,

Namespace Fundamentals:

std Namespace With using Declaration:

```
#include <iostream>
using std::cout;
using std::endl;
using std::string;
int main() {
string first_name ;
cout << "Enter your first name: ";
  std::cin >> first_name;
  cout << "Hello " << first_name << "!" << endl;
  cout << "Welcome!";
return 0;
}
```

Namespace Fundamentals:

std Namespace With using Directive

- We can use the using directive to bring all the identifiers of the namespace std as if they were declared globally.
- To do this, we utilize the using keyword.

By doing this, we can:

avoid using the std:: prefix

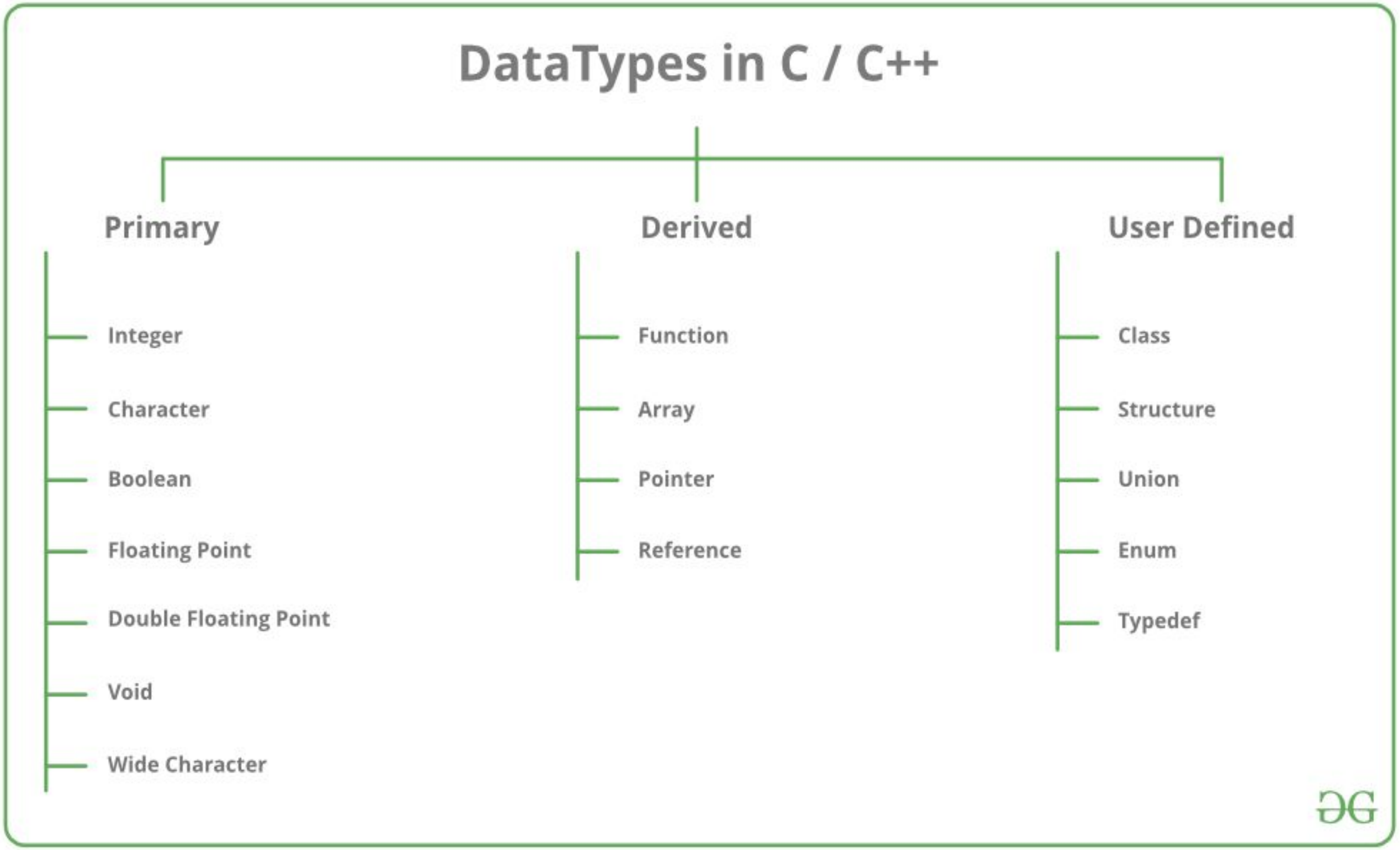
avoid utilizing the using declaration repeatedly

Namespace Fundamentals:

std Namespace With using Directive

```
#include <iostream>
using namespace std;
int main() {
string first_name ;
 cout << "Enter your first name: ";
 cin >> first_name;
 cout << "Hello " << first_name << "!" << endl;
 cout << "Welcome!";
 return 0;
 }
```

Data types:


DataTypes in C / C++
Primary
Integer
Character
Boolean
Floating Point
Double Floating Point
Void
Wide Character
Derived
Function
Array
Pointer
Reference
User Defined
Class
Structure
Union
Enum
Typedef

Data types:

Primitive Data Types

Integer: The keyword used for integer data types is int. **Character:** Character data type is used for storing characters. The keyword used for the character data type is char. Characters typically require 1 byte of memory space and range from -128 to 127 or 0 to 255.

Boolean: Boolean data type is used for storing Boolean or logical values. A Boolean variable can store either true or false. The keyword used for the Boolean data type is bool.

Floating Point: Floating Point data type is used for storing single-precision floating-point values or decimal values. The keyword used for the floating-point data type is float. Float variables typically require 4 bytes of memory space.

Data types:

Double Floating Point: Double Floating Point data type is used for storing double-precision floating-point values or decimal values. The keyword used for the double floating-point data type is double. Double variables typically require 8 bytes of memory space.

void: Void means without any value. void data type represents a valueless entity. A void data type is used for those function which does not return a value.

Wide Character: Wide character data type is also a character data type but this data type has a size greater than the normal 8-bit data type. Represented by wchar_t. It is generally 2 or 4 bytes long.

sizeof() operator: sizeof() operator is used to find the number of bytes occupied by a variable/data type in computer memory.

Data types:

- C++ has support for a Boolean type (called bool) but C does not.
- C/C++ has the char data type:

Similar to Java in that it stores a single character.

Differs from Java in that it is 8-bits big and just stores the ASCII value of the character.

- Although not a primitive type, Java supports the string class.

Input with cin, Output Using cout

- The two instances cout in C++ and cin in C++ of iostream class are used very often for printing outputs and taking inputs respectively.
- These two are the most basic methods of taking input and printing output in C++.
- To use cin and cout in C++ one must include the header file iostream in the program.

Input with cin, Output Using cout

Standard output stream (cout):

- Usually the standard output device is the display screen.
- The C++ cout statement is the instance of the ostream class.
- It is used to produce output on the standard output device which is usually the display screen.
- The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).

Input with cin, Output Using cout

Standard output stream (cout):

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

Input with cin, Output Using cout

standard input stream (cin):

- Usually the input device in a computer is the keyboard.
- C++ cin statement is the instance of the class istream and is used to read input from the standard input device which is usually a keyboard.
- The extraction operator(>>) is used along with the object cin for reading inputs.
- The extraction operator extracts the data from the object cin which is entered using the keyboard.

Input with cin, Output Using cout

standard input stream (cin):

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

int main()
{
    int age;
    cout << "Enter your age:";
    cin >> age;
    cout << "\nYour age is: " << age;
    return 0;
}
```

Input with cin, Output Using cout

standard input stream (cin):

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

int main()
{
    int age;
    cout << "Enter your age:";
    cin >> age;
    cout << "\nYou are " << age<<”years old”;
    return 0;
}
```

Type Conversion: Automatic Conversions, Casts

Type Casting: In typing casting, a data type is converted into another data type by the programmer using the casting operator during the program design. In typing casting, the destination data type may be smaller than the source data type when converting the data type to another data type, that's why it is also called narrowing conversion.

Syntax/Declaration:-

destination_datatype = (target_datatype)variable;

(): is a casting operator.

target_datatype: is a data type in which we want to convert the source data type.

Type Conversion: Automatic Conversions, Casts

```
float x;
byte y;
...
...
y=(byte)x;
```

Type Conversion: Automatic Conversions, Casts

Type conversion : In type conversion, a data type is automatically converted into another data type by a compiler at the compiler time. In type conversion, the destination data type cannot be smaller than the source data type, that’s why it is also called widening conversion. One more important thing is that it can only be applied to compatible data types.

Type Conversion example –

```
int x=30;
float y;
y=x;
```

Function

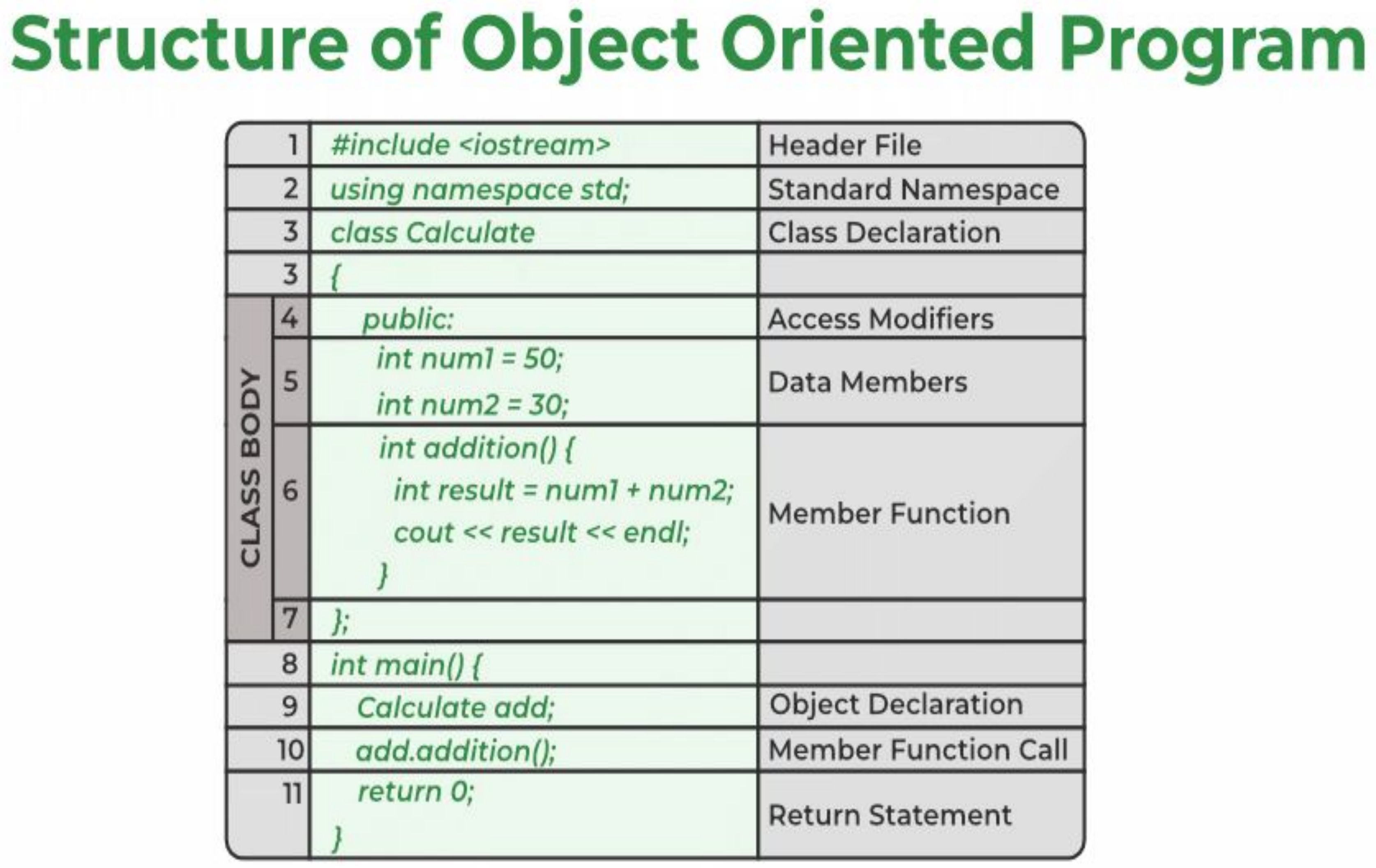
```
void myFunction() {
  cout << "I just got executed!";
}

int main() {
  myFunction(); // call the function
  return 0;
}
```

Object-Oriented Programming in C++

C++ programming language supports both procedural-oriented and object-oriented programming.

The above examples are based on the procedural-oriented programming paradigm.

Object-Oriented Programming in C++



Structure of Object Oriented Program

	1	#include <iostream>	Header File
	2	using namespace std;	Standard Namespace
	3	class Calculate	Class Declaration
	3	{	
CLASS BODY	4	public:	Access Modifiers
	5	int num1 = 50; int num2 = 30;	Data Members
	6	int addition() { int result = num1 + num2; cout << result << endl; }	Member Function
	7	};	
	8	int main() {	
	9	Calculate add;	Object Declaration
	10	add.addition();	Member Function Call
	11	return 0; }	Return Statement

Object-Oriented Programming in C++

1. Class

- A class is a template of an object. For example, the animal is a class & dog is the object of the animal class.
- It is a user-defined data type.
- A class has its own attributes (data members) and behavior (member functions). The first letter of the class name is always capitalized & use the class keyword for creating the class.

Syntax:

```
class class_name{
// class body
};
```

Object-Oriented Programming in C++

2. Data Members & Member Functions

The attributes or data in the class are defined by the data members & the functions that work on these data members are called the member functions.

Object-Oriented Programming in C++

3. Object

The object is an instance of a class. The class itself is just a template that is not allocated any memory. To use the data and methods defined in the class, we have to create an object of that class.

They are declared in the similar way we declare variables in C++.

Syntax:

```
class_name object_name;
```

We use dot operator (.) for accessing data and methods of an object.

Object-Oriented Programming in C++

3. Object

The object is an instance of a class. The class itself is just a template that is not allocated any memory. To use the data and methods defined in the class, we have to create an object of that class.

They are declared in the similar way we declare variables in C++.

Syntax:

class_name object_name;

Function overloading in C++

- Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters.
- When a function name is overloaded with different jobs it is called Function Overloading.
- In Function Overloading “Function” name should be the same and the arguments should be different.
- Function overloading can be considered as an example of a polymorphism feature in C++.

Function overloading in C++

The parameters should follow any one or more than one of the following conditions for Function overloading:

- Parameters should have a different type
- Parameters should have a different number

Function overloading in C++

```
#include <iostream>
using namespace std;
void add(int a, int b)
{  cout << "sum = " << (a + b);
}
void add(double a, double b)
{cout << endl << "sum = " << (a + b);
}
int main()
{
    add(10, 2);
    add(5.3, 6.2);
return 0;
}
```

Function overloading in C++

```
#include <iostream>
using namespace std;
void add(int a, int b)
{
cout << "sum = " << (a + b);}
void add(int a, int b, int c)
{
cout << endl << "sum = " << (a + b + c);}
int main()
{
   add(10, 2);
   add(5, 6, 4);
return 0;
}
```