EXPLICIT DEFAULT FUNCTION

Explicitly defaulted function declaration is a new form of function declaration that is introduced into the C++11 standard which allows you to append the **'=default;'** specifier to the end of a function declaration to declare that function as an explicitly defaulted function.

Why do we need?
This makes the compiler generate the default implementations for explicitly defaulted functions, which are more efficient than manually programmed function implementations.

Constraints for making default:
A defaulted function needs to be a special member function (default constructor, copy constructor, destructor etc), or has no default arguments.

Advantages:
Using '= default' can also be used with copy constructor and destructors. An empty copy constructor, for example, will not do the same as a defaulted copy constructor (which will perform member-wise copy of its members). Using the '= default' syntax uniformly for each of these special member functions makes code easier to read.

Program_01

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

class A {
public:

	// parameterized constructor
	A(int x)
	{
		cout << "This is a parameterized constructor";
	}

	// Using the default specifier to instruct
	// the compiler to create the default
	// implementation of the constructor.
	A() = default;
};

int main()
{
	// executes using defaulted constructor
	A a;
	// uses parametrized constructor
	A x(1);
	return 0;
}
```

Output:

```
	This is a parameterized constructor
```

Program_02

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

class A {
public:

	// parameterized constructor
	A(int x) = default;

	// Using the default specifier to instruct
	// the compiler to create the default
	// implementation of the constructor.
	A() = default;
};

int main()
{
	// executes using defaulted constructor
	A a;
	// uses parametrized constructor
	A x(1);
	return 0;
}
```

Output:

```
 ‘A::A(int)’ cannot be defaulted A(int x) = default;/*
```

<u>EXPLICIT DELETE FUNCTION</u>

The C++ 11 standard introduced another use of this operator, which is: **To disable the usage of a member function.** This is done by appending the **=delete;** specifier to the end of that function declaration.

<u>Advantages:</u>
Deleting of special member functions provides a cleaner way of preventing the compiler from generating special member functions that we don’t want.

Prevents call of undesired functions.

<u>Program_01:</u>

```
// C++ program to disable the usage of
// copy-constructor using delete operator
#include <iostream>
using namespace std;
class A {
public:
	A(int x): m(x)
	{
	}
```

```
        // Delete the copy constructor
        A(const A&) = delete;

        // Delete the copy assignment operator
        A& operator=(const A&) = delete;
        int m;
};

int main()
{
        A a1(1), a2(2), a3(3);

        // Error, the usage of the copy
        // assignment operator is disabled
        a1 = a2;

        // Error, the usage of the
        // copy constructor is disabled
        a3 = A(a2);
        return 0;
}
```

Ouput:

```
        Error
```