STRONG TYPED ENUM:

Why should i go for strong typed enum?

- It is scoped.
- It is allows forward declaration.
- Comparison of different enum type is not possible

Program_01:
To c++98 enum is unscoped to make it scoped, we used namespace

```
#include<iostream>
/*
When two enum of same value name are used it creates a error in C99 std
To overcome this we use namespace and keeping the enum inside it

*/
        namespace cat1{
                enum Select
                {
                        Zero,
                        One,
                        Two = 100
                };
        }

        namespace cat2{
                enum Choose{
                        Hello,
                        World,
                        Zero
                };
        }

int main()
{
        cat1::Select var = cat1::Two;
        cat2::Choose var1 = cat2::Zero;
        std::cout << var << "\n" << var1 << std::endl;
        return 0;

}
```

Output:

```
jayakumar@ARI-IIFL-Ubuntu-19:~/work_space/C++/enum/enum_code$ g++ enum_01.cpp
jayakumar@ARI-IIFL-Ubuntu-19:~/work_space/C++/enum/enum_code$ ./a.out
100
2
```

Program_02:

When we dont specify the data correctly to enum. It will result in error and compilation error is obtained.

```cpp
#include<iostream>
/*
This will not WORK
Because
The compiler does not know underlying size and thus size of the enumerator
As a result it cannot compile this piece of code.
*/
enum Selection;

void make(Selection s)
{
        std::cout << "selection::one " << One << std::endl;
}

enum Selection
{
        None,
        One,
        Two
};

int main()
{
        Selection num;
        make(num);
        return 0;
}
```

Program_03:

Implicit conversion in c++98

```cpp
#include<iostream>
/*
Here process function was supposed to run with int parameter, not selection.
But our program run with selection as parameter and no warning was shown.
So implicit conversion is taking place
*/
enum Selection: char
{
        None = 98,
        Single = 99,
        Multiple = 100
};

void process(int value)
{
        std::cout << "The value " << value << std::endl;
}

int main()
{
        process(None);
        process(Single);
        process(Multiple);
        return 0;
}
```

Output:

```
The value 98
The value 99
The value 100
```

Program_04:

```
#include<iostream>
/*
enum1 data must be of enum1 type same applies to enum2

*/
enum class enum1
{
        None = 100,
        Single,
        Multiple
};

enum class enum2
{
        None = 1,
        Single,
        Multiple
};

int main()
{
        enum1 var1 = enum1::Multiple;
        enum2 var2 = enum2::Multiple;
        std::cout << "enum1 Multiple " << static_cast<char>(var1) << std::endl;
        std::cout << "enum2 Multiple " << static_cast<bool>(var2) << std::endl;
        return 0;
}
```

Output:

```
enum1 Multiple f
enum2 Multiple 1
```

Program_05:

```
#include<iostream>
/*
C++11
Here forward declaration is being used which didnt work in C++98
*/

//Prototype needs the type of enum
enum class Selection;

void make(Selection s)
{
        std::cout << "selection::one " << static_cast<int>(s) << std::endl;
}

enum class Selection
{
        None=100,
        One,
        Two
};

int main()
{
        Selection num = Selection::One;
        make(num);
        return 0;
}
```

Output:

Selection::one 101

Program_06:

```
#include<iostream>

int main()
{
        enum data
        {
                one, two, three
        };
        //Implicit data conversion
        //      Here the enum of data type is implicitly converted to integer
        int var1 = three;
        std::cout << "The value of var1 " << var1 << std::endl;
        return 0;
}
~
```

Output:

The value of var1 2