Enumerable types (enums)

- Introduces in C89/C90 standard
- Represent one value from the set of values. Values are C symbols.
- Who does it work:
  - Every symbolic value from enum is mapped to a unique integer (whole) number.
  - One every place where enum symbol is used, it is replaced with that corresponding integer number. This is done by compiler.

Enum example

- Value 0 is attached to the first symbol in the set, unless some other value is explicitly attached.

```
enum Days
{
   Sunday = 1,
   Monday,
   Tuesday,
   Wednesday,
   Thursday,
   Friday,
   Saturday
};
```

- For every subsequent symbol attached value is calculated as value of the previous symbol, plus 1.

- Sometimes (usually in some older code) #define directives are used to make a set of symbols.

```
#define   Sunday      1
#define   Monday      2
#define   Tuesday     3
#define   Wednesday   4
#define   Thursday    5
#define   Friday      6
#define   Saturday    7
```

Scope of enum symbols

- Defined symbols fall into the current scope:

```
enum PerasFriends { SIMA, DJURA, STEVA };

SIMA // this is how we refer to that symbol
```

- Which can be a problem simetimes:

```
enum PerasFriends { SIMA, DJURA, STEVA };
enum MikasFriends { DJOLE, SIMA, MILE };
SIMA // which SIMA do you mean?
```

- Advice is to always do something like this:

```
enum PerasFriends { PD_SIMA, PD_DJURA, PD_STEVA };
enum MikasFriends { MD_DJOLE, MD_SIMA, MD_MILE };
PD_SIMA // Now it is clear.
```

Size of the enum type (underlying integer type)

- Enum type reduces to some integer type
- Compiler can decide to which integer type it will reduce it to, but the C standard imposes two conditions:
  - It must be big enough to contain all values from the enum
  - but, it is not required to support the enum (compile it) if it needs type bigger than int.
- Some compiler support enums that require integer type bigger than int.

Signedness of enum types

- Just as the size, the signedness is also on compiler to decide.
- It doesn’t need to be the same for all enums.
- Usually, if there are enum symbols with attached negative number, the underlying integer will be signed, and unsigned otherwise. **But**, it doesn’t have to be like that!!!
- Do not rely on enum signedness! There is really very little reason to do that.

A glimpse into C++

- **enum class** exists in C++:
- 1) scope

```
enum class PerasFriends { SIMA, DJURA, STEVA };
enum struct MikasFriends { DJOLE, SIMA, MILE };
PerasFriends ::SIMA
```

- 2) Underlying integer type

```
enum class Example1 { A, B, V }; // int is always underlying
enum class Example2 : long { X, Y, Z }; // now it is long
enum class Example3;
...
enum class Example3 { P, Q, R };
```

- 3) Conversion

```
   // Now this is not allowed (without casting)
   Day x = 75;
   Month y = MONDAY;
```