enum

`enum` – a new type

`enum` is a set of constant `int` values,

that defines a type:

```
enum Season
{
    WINTER,   // = 0 by default
    SPRING,   // = WINTER + 1
    SUMMER,   // = WINTER + 2
    AUTUMN    // = WINTER + 3
};
```

`enum` – a new type

`enum` is a set of constant `int` values,

that defines a type:

```
typedef enum Season
{
    WINTER,   // = 0 by default
    SPRING,   // = WINTER + 1
    SUMMER,   // = WINTER + 2
    AUTUMN    // = WINTER + 3
} Season;
```

enum – a new type, usage

```
typedef enum Season {WINTER, SPRING, SUMMER,
AUTUMN} Season;

Season curr_seasons= SPRING;
curr_season= 19; // legal, but UGLY!
int prev_season= WINTER; // legal, but UGLY!
```

enum – numbering

By default enums start with 0 and then +1 for each one.

This can be changed:

```
typedef enum Color
{
   RED=2,
   GREEN, //==3
   BLUE=8
} Color;
```

Use enum to eliminate magic numbers – alternative to #define

```
#include <stdio.h>
#include <stdlib.h>

enum {INPUT_FILE_NAME=1, OUTPUT_FILE_NAME, ARGS_NUM};

int main(int argc, char* argv[])
{
    if(argc != ARGS_NUM)
    {
        printf("usage: wc <input file name> <output file
name>\n");
        exit(1);
    }

    FILE* inFile, outFile;

    inFile = fopen(argv[INPUT_FILE_NAME],"r");
```

`enums` – why?

- More **readable** code
- Code **less error prone**
- Accessible for **debugger**
- Use of the **numerical values** is not disabled, but **bad programming usually**!

switch

switch

```
if (integer value == 3)
//statement or block
else if (integer value == 5)

//statement or block


switch (integer value)
{
   case 3:
      //statement or block
      break; //optional. Otherwise fall-through!
   case 5:
      //statement or block
   case default:
      //statement or block
}
```

switch & enum

```
typedef enum Season
    {WINTER, SPRING, SUMMER, AUTUMN} Season;
Season s;
...
switch (s)
{
   case WINTER:
     printf("Cold and raining\n");
     break; //optional. Otherwise fall-through!
   case SPRING:
     printf("Nice weather\n");
     break; //optional. Otherwise fall-through!
```

goto

goto keyword

```
goto bla;
whee:
  //code here
bla:
  //code there
```

DANGER: SPAGHETTI CODE!

Avoid whenever possible.

Q: When do use?

A: There are cases. e.g. :

- Break from a lot of nested loops
- Forward jump to error handler