Table of Contents

Tag-names/Enumeration
Enumeration constants - use
Composites - initialization sequence
Composites - initialization syntax
Static members in a class
C - characters and strings
FString - a real, useful example

link to handout
link to syllabus
Tag-names/Enumeration

⇒ the identifier used by the compiler symbol table
⇒ enumeration constants - improve code readability

Syntax examples:

```c
enum tag-name { enum list };  
class tag-name { member list };  
```

Tag-names are not associated just with enumeration constants

Enumeration list examples:

```c
enum colors(white, black, orange, pink);  
// default values - white = 0, black = 1, etc.
enum colors(white = 1, black, orange = 5, pink);  
// values - white = 1, black = 2, orange = 5, pink = 6
```
Enumeration constants - use

Example: use for readability

```cpp
enum color { red, yellow, green, blue };  
color screenbkgrnd
if (screenbkgrnd != green) { action };  // readable
if (screenbkgrnd != 2) { action };  // a mystery
```

Initializing variables - enum or define

```cpp
#define idSize 7;  
enum {nameSize = 30};  
class Student {
    private:
    char id[idSize+1];  
    char name[nameSize+1];  
};
```

#define or enum can are interchangeable. Adopt either for you programming style
Composites - initialization sequence

- Important - order which constructor parameters are used
- all members (including classes) are constructed before the class is constructed
- the class is destroyed before all members are destroyed

Example:

```cpp
class Point;

class Rectangle {
    public:
    Rectangle(int uplfX = 0, int uplfY = 0, int lwrtX = 0, int lwrtY = 0);

    private:
    Point uplft, lwrgt;  // constructed first
```
Composites - initialization syntax

Alternative Initialization - Examples:

```cpp
class Point {
    public:
        Point(int x, int y); // constructor
        private: int xVal, yVal;
    }

    // intialization code - one way
    Point::Point(int x, int y) { xVal = x; yVal = y;
        // rest of initialization }

    // intialization code - alternative
    Point::Point(int x, int y) : xVal(x), yVal(y)
        { // rest of initialization }
```
Static members in a class

⇒ create class “global” members - unique & singular
⇒ use - when a member must be shared by all instantiations of the class

Example:
```cpp
class Point {
public:
    Point ( ) { numPoints++; }; // default constructor
    ~Point( ) { numPoints--; }; // destructor
private:
    static short numPoints;
    float xVal, yVal;
};
```

Recall: “Static” has many different meanings in C++
Strings - as in C

⇒ Basic structure - char buffer[ML]; // an array of characters
⇒ No operations, no overflow check, etc.
⇒ String - char array term. with a NULL char - ‘\0’
⇒ String utility in library <string.h>
  ⇒ copy; concat.; compare; search; tokens; etc.

Example:

```c
char name[ ] = “Myname is Jane”; // string created
char newName[30]; // just a buffer of chars
char* stgPtr; // pointer to character
newName = name; // error - no copy. newName loc fixed
strcpy(newName, name); // ok
stgPtr = name; // okay assigning a pointer
stgPtr = &newName[1]; // okay now points to 2nd char
```
Strings in C++ - FString a useful example

⇒ Number of string class utilities - adopt one for *your* use
⇒ FString - simple, easy to use, does things you want
  ⇒ constructs from C or FString strings
  ⇒ Assign and Append from C or FString strings
  ⇒ Ops - compare, less than, equal, greater than - add !=
    ⇒ future - add operator functions, +, =, ==, >, <; etc.
⇒ Return C string pointer (const) - use for C string ops.
⇒ I/O operators >>, <<
⇒ Fixed length - 255 (protected) - future extend with pointers

links to FString casestudy [header code](#), [source code](#), [test program](#)