

LAB 6

INCREMENTING ENUM VALUES

- Suppose we have an enum of type int (default) with values 0-6 (default)
- Suppose we have a var named **day**
- We cannot simply use `day++`
- Instead, we need:
 - **`day = (DateEnum)(day + 1);`**
- Problem with the above:
 - **`day = (DateEnum)7;`**
 - Is undefined. Now, we do not know the value
- Solution:
 - Use mod 7 (`%7`) => **`day = (DateEnum)((day + 1) % 7);`**

```
10  typedef enum : int {  
11      SUN = 0,  
12      MON = 1,  
13      TUE = 2,  
14      WED = 3,  
15      THU = 4,  
16      FRI = 5,  
17      SAT = 6,  
18  } DateEnum;
```

DECREMENTING ENUM VALUES

- Suppose we have an enum of type int (default) with values 0-6 (default)
- Suppose we have a var named **day**
- We cannot simply use `day++`
- Instead, we need:
 - **`day = (DateEnum)(day - 1);`**
- Problem with the above:
 - **`day = (DateEnum)-1;`**
 - Is undefined. Now, we do not know the value
- Solution:
 - Use mod 7 (`% 7`) => **`day = (DateEnum)((day - 1) % 7);`**

```
10  typedef enum : int {  
11      SUN = 0,  
12      MON = 1,  
13      TUE = 2,  
14      WED = 3,  
15      THU = 4,  
16      FRI = 5,  
17      SAT = 6,  
18  } DateEnum;
```



```

20 typedef struct _date {
21
22     _date(DateEnum day = MON) {
23         this->day = day;
24     }
25
26     void setDay(DateEnum newDay) {
27         this->day = newDay;
28     }
29
30     DateEnum nextDay(void) {
31         this->day = (DateEnum)((this->day + 1) % 7);
32         // 7 % 7 = 0, now we have valid #s
33         return this->day;
34     }
35
36     DateEnum lastDay(void) {
37         this->day = (DateEnum)((this->day - 1) % 7);
38         // -1 % 7 = 6, now we have valid #s
39         return this->day;
40     }
41
42     DateEnum getDay(void) {
43         return this->day;
44     }
45
46 private:
47
48     DateEnum day = MON;
49 } Date;

```

```

20 typedef struct _date {
21

```

BETTER SOLUTION:
PUT IT IN A STRUCT

```

10 typedef enum : int {
11     SUN = 0,
12     MON = 1,
13     TUE = 2,
14     WED = 3,
15     THU = 4,
16     FRI = 5,
17     SAT = 6,

```

```

48 DateEnum day = MON; DateEnum;
49 } Date;

```

TESTING FUNCTIONS

- Functions should perform a single task
 - Input
 - Output
 - Computation
- Of course, if we have functions that only do one of those, then programs would be useless
 - We need to have end functions which do not call any other functions
 - **Note: end functions is not an official term**
 - A function should call a function to handle input, output, and computation

TESTING FUNCTIONS

- This separation allows for us to test the computation functions
- We always assume the user does not know or understand how to use the program
 - That means we need to handle poor/improper input
- When we separate the computation, we will perform the tests only on the computation
- We can test input/output files by verifying the data is correct, but we cannot test stdin (cin) or stdout (cout)
- Testing can only be performed on functions which accept an input and produce an output

HOW TO INCLUDE C HEADERS

- `#include <cstdlib>`
 - Instead of `#include <stdlib.h>`
- Use ``c`{LIB_NAME}` without the ``.h`` as seen above
- If you need a library only in C, then we will use the format so developers can clearly see it is a standard C library rather than C++
- Note that C++ still used many C components, but the header names are different

DELETE, NEW

- As mentioned in lab 5:
- **delete** is used to deallocate memory from **new** allocations
- **new** is used to allocate memory; you do not need to pass in explicit sizes
- Important note:
 - When you write int * arr = new int[50], it takes $50 * \text{sizeof}(\text{int}) + \text{sizeof}(\text{int} *)$ bytes
 - When you write int * arr = calloc(sizeof(int), 50), it takes $50 * \text{sizeof}(\text{int})$ bytes
 - So, in C++, this nicer style takes $\text{sizeof}(\text{int} *)$ more bytes
- *Italics* = size ; **bold** = keyword ; underscore = code

CWE 481 - REVIEW

- Examples from mitre.org CWE 481
- “In many languages the compare statement is very close in appearance to the assignment statement and are often confused. This bug is generally the result of a typo and usually causes obvious problems with program execution. If the comparison is in an if statement, the if statement will usually evaluate the value of the right-hand side of the predicate.”

Example Language: C

```
void processString (char *str) {  
    int i;  
  
    for(i=0; i<strlen(str); i++) {  
        if (isalnum(str[i])){  
            processChar(str[i]);  
        }  
        else if (str[i] = ':') {  
            movingToNewInput();  
        }  
    }  
}
```

Example Language: C#

```
bool isValid(int value) {  
    if (value=100) {  
        Console.WriteLine("Value is valid.");  
        return true;  
    }  
    Console.WriteLine("Value is not valid.");  
    return false;  
}
```

Example Language: Java

```
public void checkValid(boolean isValid) {  
    if (isValid = true) {  
        System.out.println("Performing processing");  
        doSomethingImportant();  
    }  
    else {  
        System.out.println("Not Valid, do not perform processing");  
        return;  
    }  
}
```

CWE 1341 - 1 PT

- “Code typically requires “opening” handles or references to resources such as memory, files, devices, socket connections, services, etc. When the code is finished with using the resource, it is typically expected to “close” or “release” the resource, which indicates to the environment (such as the OS) that the resource can be re-assigned or reused by unrelated processes or actors - or in some cases, within the same process. API functions or other abstractions are often used to perform this release, such as free() or delete() within C/C++, or file-handle close() operations that are used in many languages.”

Example Language: C

```
char b[2000];
FILE *f = fopen("dbl_cls.c", "r");
if (f)
{
    b[0] = 0;
    fread(b, 1, sizeof(b) - 1, f);
    printf("%s\n", b);
    int r1 = fclose(f);
    printf("\n-----\n1 close done '%d'\n", r1);

    int r2 = fclose(f); // Double close
    printf("2 close done '%d'\n", r2);
}
```

CWE 500 – 1 PT

- "Public static variables can be read without an accessor and changed without a mutator by any classes in the application."

Example Language: C++

```
class SomeAppClass {  
  
    public:  
        static const string appPropertiesConfigFile = "app/p  
  
    ...  
}
```

Example Language: C++

```
class SomeAppClass {  
  
    public:  
        static string appPropertiesConfigFile = "app/properties.config";  
  
    ...  
}
```

Example Language: Java

```
public class SomeAppClass {  
  
    public static final String appPropertiesFile = "app/App  
    ...  
}
```

Example Language: Java

```
public class SomeAppClass {  
  
    public static String appPropertiesFile = "app/Application.properties";  
    ...  
}
```

CWE 495 – 1 PT

- "The product has a method that is declared public, but returns a reference to a private data structure, which could then be modified in unexpected ways."

Example Language: C++

```
class Color
{
    private:
        int[2] colorArray;
        int colorValue;
    public:
        Color () : colorArray { 1, 2 }, colorValue (3) { };
        int[2] & fa () { return colorArray; }
        int & fv () { return colorValue; }
};

int main ()
{
    Color c;

    c.fa () [1] = 42;
    c.fv () = 42;

    return 0;
}
```


WHAT SHOULD WE CHANGE?

- We use `!input.eof()`. This is not the best
- Instead, we want to use `!input.good()`
- `good()` is more reliable than `eof()`

```
ifstream& operator>> (ifstream& input, List& rhs)
{
    char line[100] = "";
    input.getline(line, 100); // read in the line

    while (!input.eof()) // read all lines from th
    {
        // example format: "Smith,John",99
        input.getline(line, 100, ','); // split li
        input.getline(line, 100, ','); // still on
        input.getline(line, 100);      // read the
        // convert char * line to int score type
        int score = atoi(line); // atoi() converts
        rhs.insertAtFront(score); // no need to re
            efficient!
    }

    return input;
}
```

SHALLOW VS DEEP COPYING

- Shallow
 - Use the same references
 - Ints and such as copied over as normal
- Deep
 - Copy each value over, which will use more memory when working with pointers

CASTING IN C++

- Used to be:
 - `(double)(x * y)/10;`
- Now, we have:
 - `static_cast<double>(x * y)/10`

CLASSES VS STRUCTS (IN C++)

- They are identical, except for default access. What is the difference?
- Structs
 - **Public** unless otherwise noted
 - *Note: in C, we cannot put functions inside of structs. This is a feature of C++*
- Classes
 - **Private** unless otherwise noted

USER INPUT EXAMPLE

- You can use this on your PAs!
- Start making a namespace or class of functions that you define!
- Make these PAs easier on yourself, don't rewrite functions
- Examples: (We can do these if you want)
 - `void getUserInput(string &str);`
 - `template <typename T> ; bool validateInput(const string &str, T min, T max) ; bool validateInput(const string &str, string contains);`
 - `bool getYesNo(void)`
 - `bool isnumber(const string &str)`
 - `inline void clearScreen(void)`



OBJECTIVES FOR THIS LAB

- They are much easier since most of the code is given
- Now, we are simply implementing some features that use new and delete
- We also have to implement a deep copy constructor
- Follow the comments

OPERATOR= VS COPY CONSTRUCTOR

```
List l1, l2, l3;  
///  
// l1-l3 are used and filled here  
l1 = l2; // Line 1  
List l4 = l3; // Line 2
```

- Line 1 will use the overridden function operator=
 - L1 has been initialized already, so it will call a function
- Line 2 will use the copy constructor
 - L4 has not been initialized yet, so it will copy the code over
- How can we use the same code? Call the copy constructor from operator=?
 - We cannot call a constructor! This is handled by the compiler!
 - We can, however call the same function from both places
 - That is, we create a *private* copy function where all logic is handled