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;
```

```
typedef struct _date {

    _date(DateEnum day = MON) {
        this->day = day;
    }

    void setDay(DateEnum newDay) {
        this->day = newDay;
    }

    DateEnum nextDay(void) {
        this->day = (DateEnum)((this->day + 1) % 7);
        // 7 % 7 = 0, now we have valid #s
        return this->day;
    }

    DateEnum lastDay(void) {
        this->day = (DateEnum)((this->day - 1) % 7);
        // -1 % 7 = 6, now we have valid #s
        return this->day;
    }

    DateEnum getDay(void) {
        return this->day;
    }

private:

    DateEnum day = MON;
} Date;
```

```
typedef struct _date {

        DateEnum day = MON;
} Date;
```

BETTER SOLUTION: **PUT IT IN A STRUCT**

```
typedef enum : int {
    SUN = 0,
    MON = 1,
    TUE = 2,
    WED = 3,
    THU = 4,
    FRI = 5,
    SAT = 6,
} DateEnum;
```

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 * sizeof(int) + sizeof(int *)* bytes
  - When you write int * arr = calloc(sizeof(int), 50), it takes *50 * sizeof(int)* bytes
  - So, in C++, this nicer style takes *sizeof(int *)* more bytes

- *Italics = size* ; **bold = keyword** ; underscore = code

LINKED LISTS

- They are much easier since most of the code is given
- Now, we are simply implementing some features that use new and delete

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

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;
  }
}
```

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 or
        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;
}
```

CASTING IN C++

- (double)(x * y)/10;
- static_cast<double>(x * y)/10

CLASSES VS STRUCTS (IN C++)

- They are identical, except for default access
- Structs
  - **Public** unless otherwise noted
- 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)