PIC 10A 2B

TA: Bumsu Kim

UCLA

Today…

- Announcements
- Visual Studio Shortcuts
- Review: Arithmetic Operations on `int` and `double`
- Homework Tips
- String Manipulations

Announcement

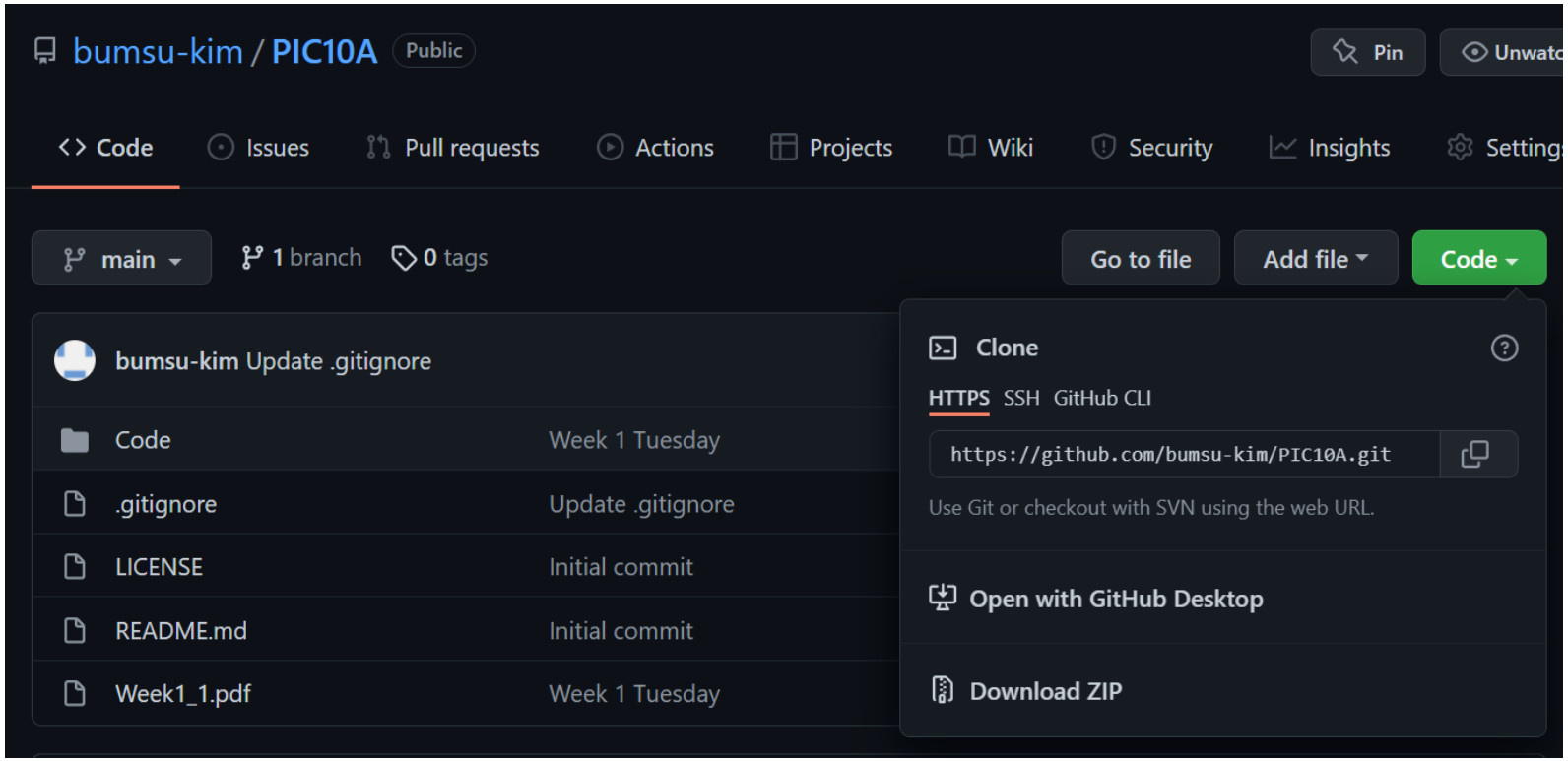
- TA email: bumsu@ucla.edu
- Office Hours: Thursdays 1pm – 2:30pm, on the RIGHT side of the PIC Lab
  - If you can't make it, you can also join the other TAs' OH
    - Th 3pm – 4:30pm @PIC Lab (Dominic Yang)
    - Th 3pm – 5pm @PIC Lab (Kye Shi)
    - Fri 4:15pm – 5:45pm @MS 6118 (Balaji)
- Github Repo: https://github.com/bumsu-kim/PIC10A
- HW 1 is out. The due date is Monday, Oct 10th (5pm)

UCLA

Git

- Go to: https://github.com/bumsu-kim/PIC10A

- Discussion slides and supplementary materials (e.g. code) will be uploaded there



3 options to download files

1. Use Git (may be difficult)
2. Use GitHub Desktop (easy to sync, once set up)
3. Download ZIP (easy but requires downloading the whole repo every time when it is updated)

UCLA

Visual Studio Shortcuts

- After selecting a part of your code,
- [Ctrl] + K + F : Auto indentation/Spacing
- [Ctrl] + K + C : Make Comment
- [Ctrl] + K + U : Uncomment

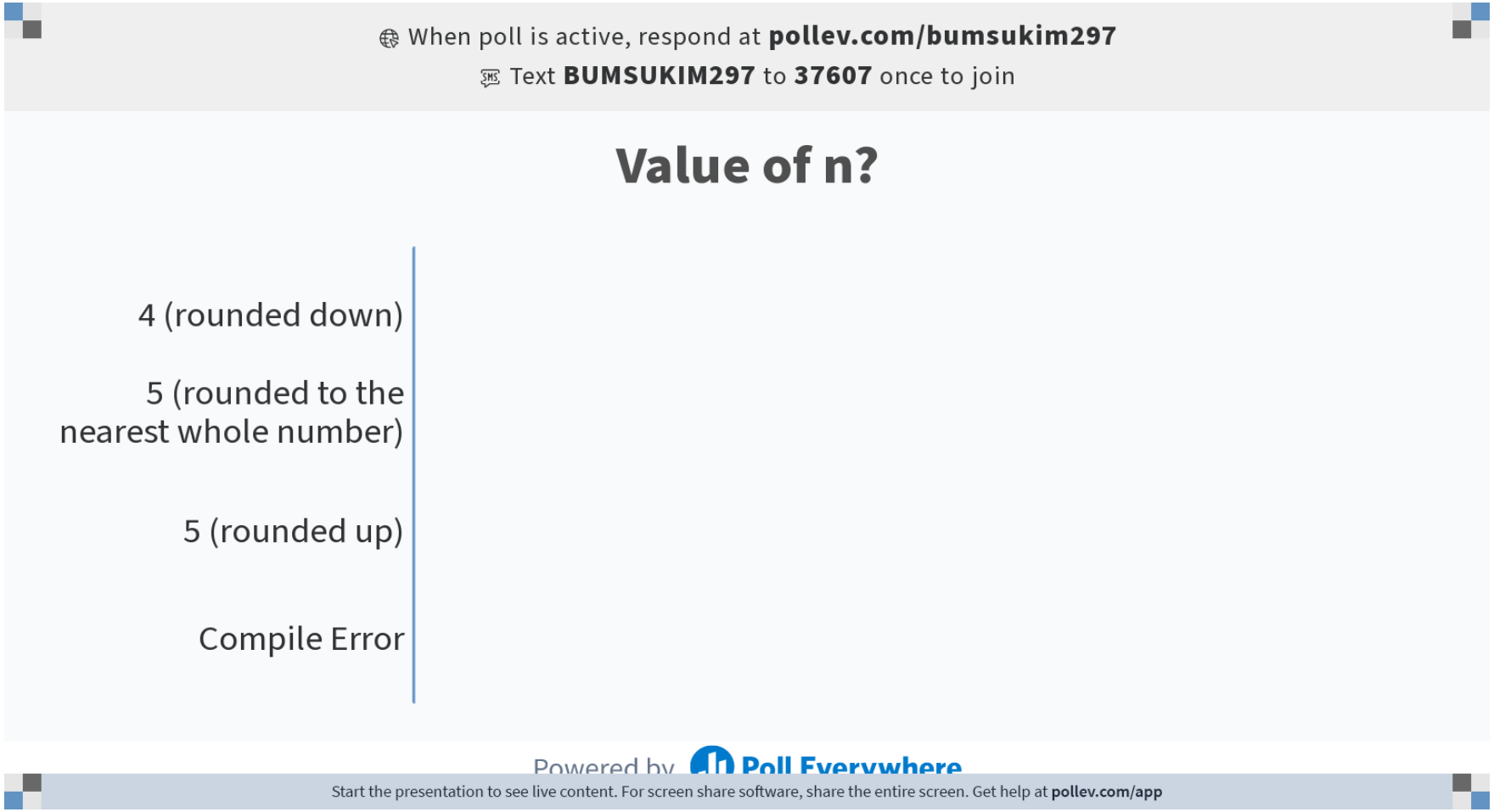
- To run your code,
- [Ctrl] + [F5] : Run without debugging (this should be used by default)
- [F5] : Start debugging (when things go wrong, you can use this)

Integer Operations (Review)

- Recall that **+**, **-**, and * work as we expect from everyday math
- However, **/** for two integers does “integer division”
  - We only take the quotient, and the remainder is discarded
- On the other hand, **%** finds the remainder
- For example, 14/3 == 4 and 14%3 == 2, because 14 = 3*4 + 2
- Maybe a similar question:

```
int n = 4.567; // what is the value of n?
```

```
int n = 4.567; // what is the value of n?
```



More on `int` operators

- Special operators for integral types (for now):
  - ++, ++, --, and --
  - (Post/pre)-(increment/decrement) operators

```
int n = 3;
++n; // pre-increment
n++; // post-increment
--n; // pre-decrement
n--; // post-decrement
```

- Pre and Post operators are different (in terms of their returned values)
- In general, pre-increment/decrement is preferred because, internally, there's a non-necessary copy in post-operators

More on operators

- Compound assignments: +=, -=, *=, /=

- Usage: x = x [operator] y;
  - [operator] can be +, *, -, / (and others)

```
int n = 3;
n += 2; // n = n + 2
n -= 2; // n = n - 2
n *= 2; // n = n * 2
n /= 2; // n = n / 2
```

- Q: What happens if you do `n %= 2` when `n` is even/odd (respectively)?

Floating-point Numbers

- We have different internal (bit) representations for x and y here:

```
double x = 5;
int y = 5;
```

- Let’s talk more about floating point numbers

Floating-point Numbers

- Floating-point types: uses the "scientific notation," in base 2 numbers
  - Ex) 1902849287.4356 → 1.902×10^{9}
  - Ex) 0.000034453234 → 3.445×10^{-6}
  - Loses precision, but more efficient for finite storage

- $\text{Mantissa} \times \text{Base}^{\text{Exponent}}$

- Of course, the Base is 2, and the mantissa and the exponent are represented in base 2 numbers

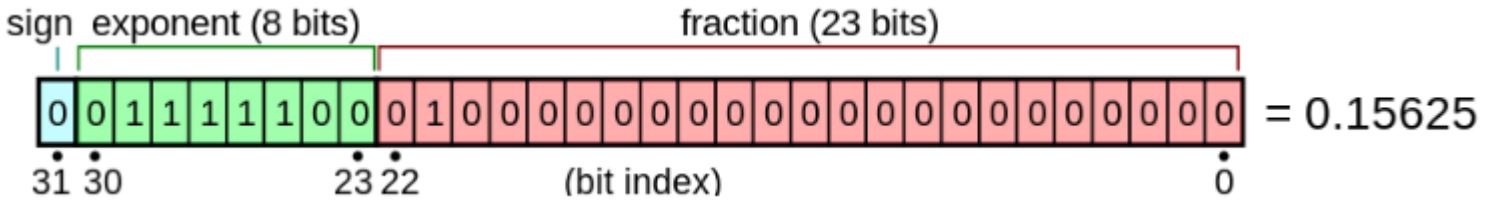
- For instance, "4.35" is stored approximately as 4.34999999999999645
  - Example:

```
int n = 4.35 * 100;
```

UCLA

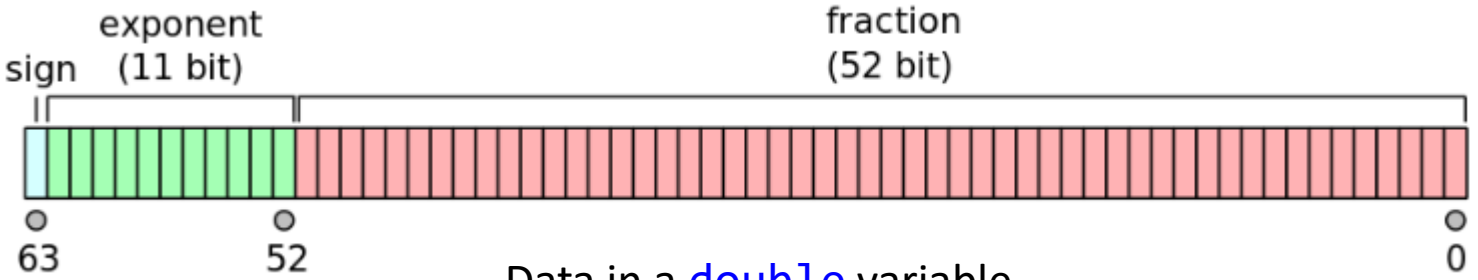
Floating-point Numbers

- Floating-point types:
  - `float`: 4 bytes
  - `double`: 8 bytes



Data in a `float` variable

- Double means “double-precision”



Data in a `double` variable

- 23-bit mantissa → about 7 significant digits in decimal (`float`)
- 52-bit mantissa → about 15 significant digits in decimal (`double`)

UCLA

The type `double` – operators

- Operators such as +, -, *, / are defined for the type `double`
- Fortunately, they are not as tricky as the case of `int`
- Use them like real number arithmetic operators, just don't forget that they have finite precision, and upper/lower limits
- There is a small number ***eps*** such that **1 + *eps* == 1** !!
  - Called a machine epsilon, and sets the limit of precision in your number system
  - Example in Live Coding

Implicit and Explicit Casts

- Always use EXPLICIT casts

```
double x = 1;  // bad: implicit casting
int y = 1.5;   // worse: loss of information
double y = 1.0; // good
int z = static_cast<int>(1.5); // good
```

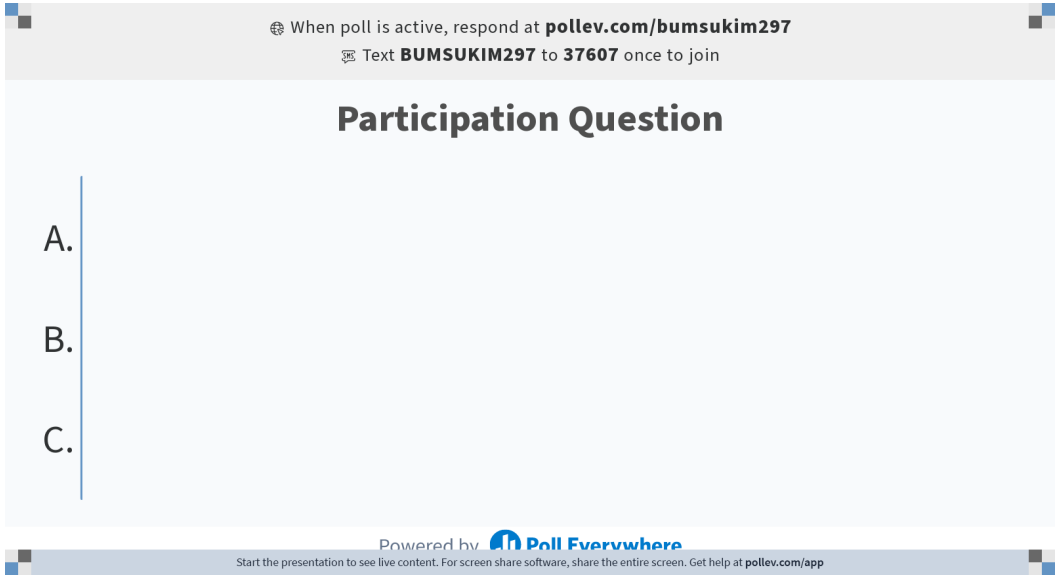
- Use `static_cast` when you should make it clear

- You will see more confusing implicit typecasts later
- So let's abide by a good coding practice

Participation Question

- Suppose that `runTime` is a double variable storing the number of seconds (down to the nearest hundredth of a second) it takes someone to complete a marathon.

- Consider the code below that is to compute their time, rounded to the nearest second, in the format of hours, minutes, and seconds (seconds and minutes should not exceed 60).
  Does the code work?




```
int a = static_cast<int>(runTime + 0.5);
int hours = a / 3600;
int minutes = (a % 3600) / 60;
int seconds = a % 60;

cout << "Rounded run time: "
       << hours << " hour(s), " << minutes
       << " minute(s), " << seconds << " second(s)";
```

A) No, there is a logical error and the result is wrong.
B) Yes, but the coding style needs improvement
C) Yes, and the style is fine.

Homework Tips

- If you're not using VS for your HW (e.g. CLion, Xcode, etc.) then make sure that your code run on VS 2022 (in PICLAB)
  - You should create a project to run your code in VS
  - Review how to create a project in VS

- File names are *extremely* important when you're submitting a HW
  - It is because that the reader will use an automated system to run your code
  - If the file names don't match, it will fail to run and you will lose points for it

- The output accuracy (including formatting) is also very important

```
cout << "Hello, World!" << endl; // if the output is supposed to be this,
cout << "Hello World!" << endl;  // this is wrong
cout << "Hello, world!" << endl; // this is also wrong
```

UCLA

Exercise – Product of Digits

- Write a program to input a positive integer from user and calculate the product of digits
- Your code should work for all integers ranging from 100 to 999
- Input and output should be exactly:

```
Input an integer (100 - 999):
[USER ENTERS AN INTEGER FROM 100 TO 999]
The product of digits is X.
```

- An example:

```
Input an integer (100 - 999):
[USER ENTERS 132]
The product of digits is 6.
```

UCLA

String Manipulations

- Recall the type `char`, used to store a single character
- Strings can be thought as a `char` array (i.e., a sequence of `char` variables), with some *features*

```
string str1 = "ABC";
```

Sequence of char objects
Each `char` can be accessed by the **[ ] operator** (subscript operator)
e.g. `str1[1]`

```
str1.length();
```

This expression returns the length of the string

```
str1.substr(0, 2);
```

This expression returns the substring starting from 0^{th} position with length 2

```
str1 + "DEF";
```

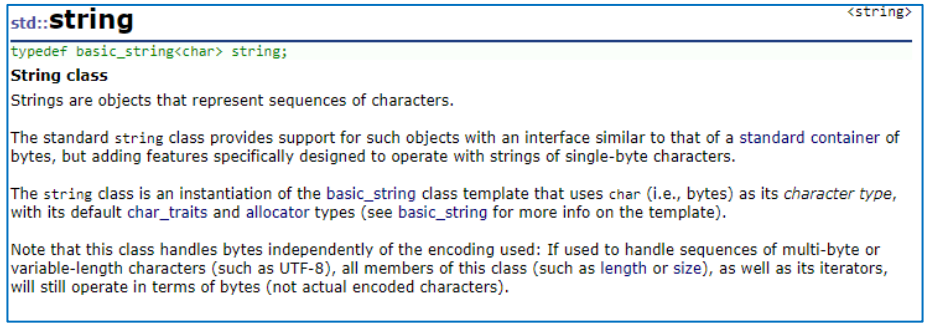
This expression returns the string formed by concatenating `str1` and `"DEF"`, so the resulting string contains `"ABCDEF"`

UCLA

String Manipulations

- There are a lot of useful features defined for string
- You can find them in the lecture notes

- Good Reference: http://cplusplus.com/
  - Teaching *how* to fish is in fact more efficient
  - Right now it can be hard for you to read the documentation
  - But you will be much more comfortable at reading it by the end of this course
  - e.g. Documentation for `string`:



std::**string** <string>

```
typedef basic_string<char> string;
```

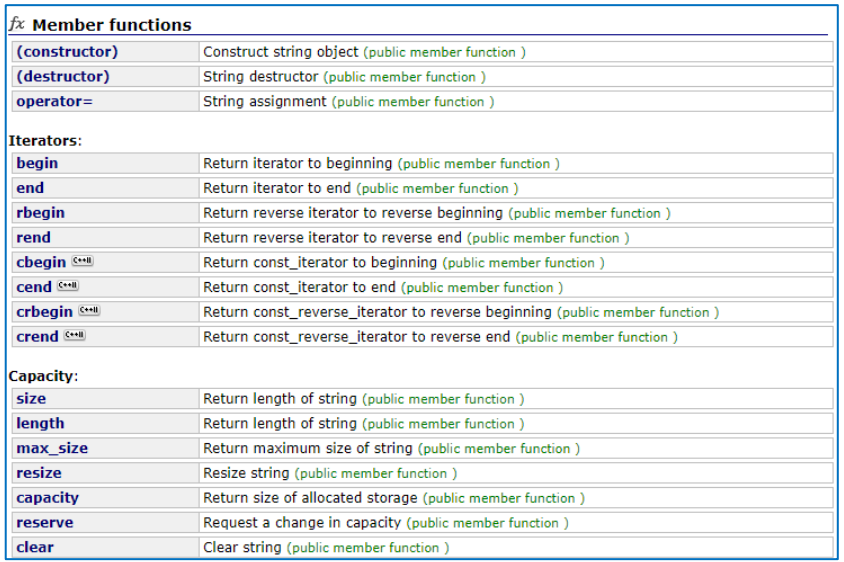
String class

Strings are objects that represent sequences of characters.

The standard `string` class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters.

The `string` class is an instantiation of the basic_string class template that uses `char` (i.e., bytes) as its *character type*, with its default char_traits and allocator types (see basic_string for more info on the template).

Note that this class handles bytes independently of the encoding used: If used to handle sequences of multi-byte or variable-length characters (such as UTF-8), all members of this class (such as length or size), as well as its iterators, will still operate in terms of bytes (not actual encoded characters).



Member functions

(constructor)	Construct string object (public member function)
(destructor)	String destructor (public member function)
operator=	String assignment (public member function)
Iterators:	
begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)
Capacity:	
size	Return length of string (public member function)
length	Return length of string (public member function)
max_size	Return maximum size of string (public member function)
resize	Resize string (public member function)
capacity	Return size of allocated storage (public member function)
reserve	Request a change in capacity (public member function)
clear	Clear string (public member function)

UCLA

Exercise (String Manipulation I)

- Q) What is the output for the following code?

```
string s1, s2, s3;
s1 = "PIC";
s2 = "10A";
s3 = s1 + s2;
s3 += "\n";
cout << s2.length() << " " << s3.length();
```

UCLA

Exercise (String Manipulation I)

- Q) What is the output for the following code?

```
string s1, s2, s3;
s1 = "PIC";
s2 = "10A";
s3 = s1 + s2;
s3 += "\n";
cout << s2.length() << " " << s3.length();
```

- A) 3 7

```
string s1, s2, s3;
s1 = "PIC";    // length == 3
s2 = "10A";
s3 = s1 + s2; // s3 == "PIC10A"    (length == 6)
s3 += "\n";    // s3 == "PIC10A\n" (length == 7)
cout << s2.length() << " " << s3.length();
```

UCLA

Exercise (String Manipulation II)

- Q) Consider the following program.

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  string s = "a more perfect union";
  cout << s[1];
  cout << s[3];
  cout << s[5] << "\n";
}
```

What is the output?

A. oe
B. a m
C. amo
D. Neither of the above

Exercise (String Manipulation II)

- Q) Consider the following program.

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  string s = "a more perfect union";
  cout << s[1];
  cout << s[3];
  cout << s[5] << "\n";
}
```

What is the output?

A. oe
B. a m
C. amo
D. Neither of the above

A) A.
C++ indexing always starts from 0

UCLA

Exercise – Number to Month Names

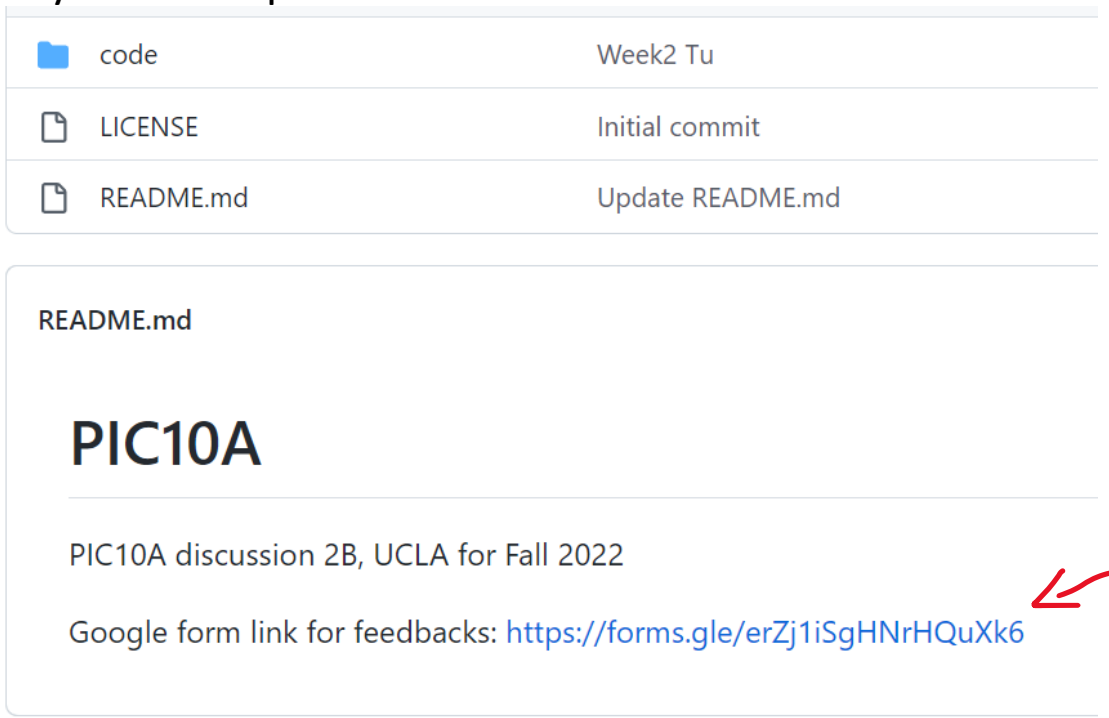
- This exercise is from the textbook

Exercise P2.19. Write a program that transforms numbers `1`, `2`, `3`, …, `12` into the corresponding month names `January`, `February`, `March`, …, `December`. *Hint:* Make a very long string `"January February March ..."`, in which you add spaces such that each month name has *the same length*. Then use `substr` to extract the month you want.

Your Feedback is welcome

- Don’t hesitate to give a feedback on the discussion
- Use the link on my Github repo, or the link below:
  - https://forms.gle/erZj1iSgHNrHQuXk6

My Github repo on the web looks like:



UCLA