PIC 10A 2B

TA: Bumsu Kim

Today…

- Loop Examples
  - Prompting Until a Match is Found
  - Finding the First Match
  - Comparing Adjacent Values
- Random Numbers
  - Applications to Loop Exercises
- HW 3 Hints

Prompting Until a Match is Found

- Write a program that keeps prompting a user until providing valid input
    - e.g. The input is valid if it is a number between 0 and 100 (exclusive)

Prompting Until a Match is Found

- Write a program that keeps prompting a user until providing valid input
  - e.g. The input is valid if it is a number between 0 and 100 (exclusive)

```
bool valid = false;
double input;
while (!valid) {
    cout << "Please enter a positive value < 100: ";
    cin >> input;
    if (0 < input && input < 100) { valid = true; }
    else { cout << "Invalid input." << endl; }
}

cout << "You validly entered: " << input << endl;
```

Prompting Until a Match is Found

- Write a program that keeps prompting a user until providing valid input
  - e.g. The input is valid if it is a number between 0 and 100 (exclusive)

```
bool valid = false;
double input;
while (!valid) {
    cout << "Please enter a positive value < 100: ";
    cin >> input;
    if (0 < input && input < 100) { valid = true; }
    else { cout << "Invalid input." << endl; }
}

cout << "You validly entered: " << input << endl;
```

More on this exercise

- Better coding style/practice?
- What happens if you input multiple invalid words (a line separated by white spaces), such as “120 S Wilshire Blvd” here?

UCLA

Finding the First Match

- Write a program that finds the position of a given character in the user input
  - e.g. Character to find = white space

Finding the First Match

- Write a program that finds the position of a given character in the user input
  - e.g. Character to find = white space

```
string str = "Hello everyone, my name is Shu.";
bool found = false;
int position = 0;
while (!found && position < str.length()) {
    string ch = str.substr(position, 1);
    if (ch == " ") { found = true; }
    else { position++; }
}
if (position == str.length()) { cout << "Not found." << endl; }
else { cout << position << endl; } // 5
```

Finding the First Match

- Write a program that finds the position of a given character in a string
  - e.g. Character to find = white space

```
string str = "Hello everyone, my name is Shu.";
bool found = false;
int position = 0;
while (!found && position < str.length()) {
    string ch = str.substr(position, 1);
    if (ch == " ") { found = true; }
    else { position++; }
}
if (position == str.length()) { cout << "Not found." << endl; }
else { cout << position << endl; } // 5
```

More on this exercise

- What if we want to find a substring?
- What if the inputs are given by the user?
- [Challenge] Can you do the same task with a while loop with an empty body?

- Hint: “Short-circuit evaluation”
  - (Expr_A) && (Expr_B) → if Expr_A is false, don’t execute/evaluate Expr_B at all
  - (Expr_A) || (Expr_B) → if Expr_A is true, don’t execute/evaluate Expr_B at all

Comparing Adjacent Values

- Write a program that finds adjacent duplicates
    - e.g. 1 7 2 9 9 4 9

Comparing Adjacent Values

- Write a program that finds adjacent duplicates
  - e.g. 1 7 2 9 9 4 9

```
double input;
double previous=0;
while (cin >> input) {
    if (input == previous) { cout << "Duplicate
        input" << endl; }
    previous = input;
}
```

Random Numbers

Syntax, Libraries, and Usage

rand() <cstdlib>

- `rand()` generates a (pseudo-) random integer,
  - Between `0` and `RAND_MAX`
    - RAND_MAX depends on the library
    - But at least 0x7fff = $2^{15} - 1 = 32767$ on any stand library implementation
  - Using a “seed”

- Using the same seed, `rand()` will generate the same sequence of random numbers
  - More on this later

Using `rand()` – Floating Point numbers

- Since it generates an `int` between `0` and `RAND_MAX`, you can generate a random number between 0 and X (inclusive) by

  - `rand_num = (rand()*1./RAND_MAX)*X;`

Type Conversion

Normalized to 0 ~ 1

  - Or,

  - `rand_num = (static_cast<double>(rand())/RAND_MAX)*X;`

    which is preferable

Using rand() – Floating Point numbers

- If you want a real number between A and B,
  - The size of interval = (B – A)
  - Starts from A

  - `rand_num = ((rand()*1./RAND_MAX)*(B-A) + A;`

$$\text{rand\_num} = (\underbrace{(\text{rand()*1./RAND\_MAX})*(\text{B-A})}_{\text{Between 0 and (B-A)}} + \text{A};$$

Using `rand()` – Integers

- For integers, we can use the % (mod/remainder) operator

- Random int between 0 and N (inclusive):
  - `n = rand()%(N+1)`
  - The remainder is always between 0 and N
    - Note that, there are **N+1** different integers in [0,N]

- Random int between A and B:

  = A + (random int between 0 and (B-A))
  - `n = A + rand() % (B-A+1)`

A Seed for `rand()`

- The function `rand()` generates a sequence of random numbers using a seed
- The seed can be set by `srand(some_number)`;
- Ex) `some_number` == 1 → `rand()` gives
  - 41
  - 18467
  - 6334
  - 26500
  - 19169
  - 15724 …

  - Exercise: Use rand() several times without srand() commands, and verify that you get the sequence above

Seed for `rand()`

- With the same seed, you'll have the same sequence of random numbers

- So, to get a random-like numbers (pseudo-random numbers), use different seed every time you run the code

- Commonly used trick: use a ***current time*** as a seed
  - `time(nullptr)` in `<ctime>` library returns the current time in **seconds**, since *00:00, Jan 1 1970 UTC*
  - Ex) `time(nullptr) == 1666067370` when this slide was created

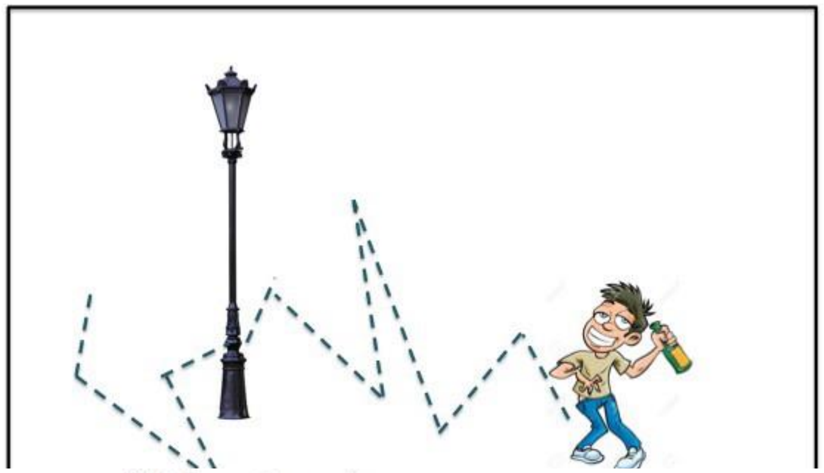
Summary

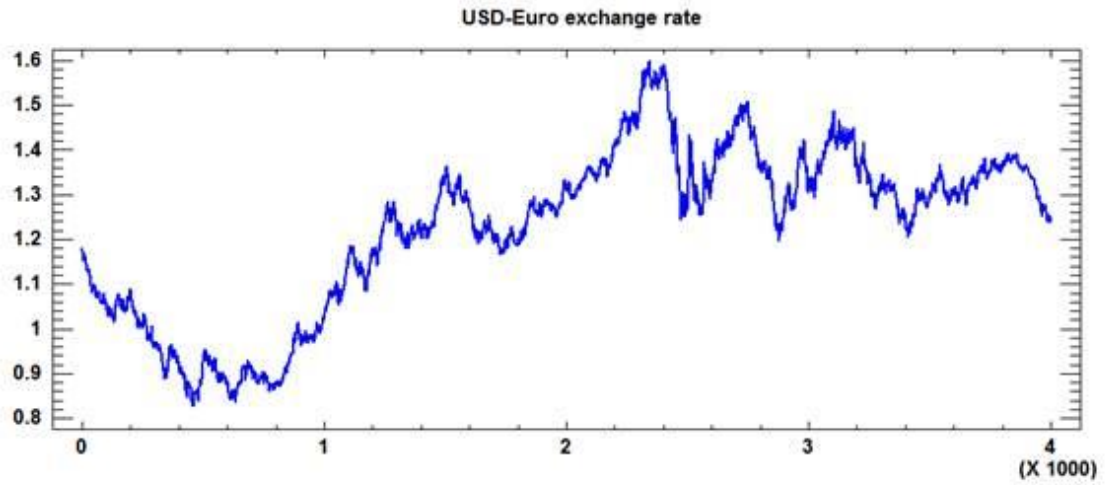
- #include `<cstdlib>` for `rand()` and `srand()`
- #include `<ctime>` for `time()`
- Use `time(nullptr)` to generate a seed:
  - `srand(time(nullptr));`
  - And then use `rand()`
- Intervals:
  - `r_double = (rand()*1./RAND_MAX)*(B-A) + A;`
  - `r_int = A + rand()%(B-A+1);`

Exercise – Random Walk Simulator

(will be revisited on Thursday)

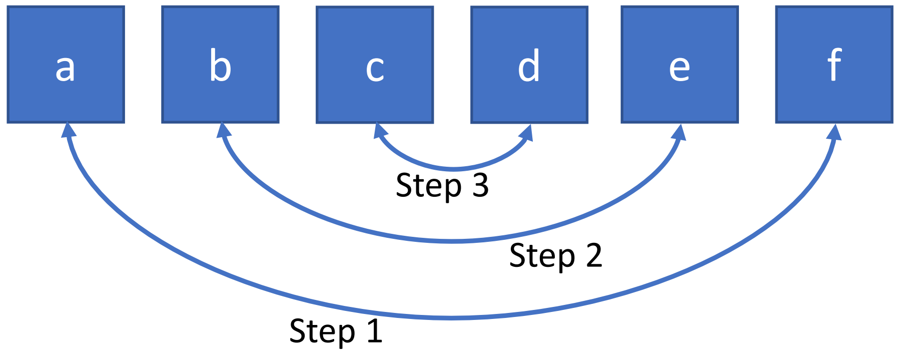
- Suppose someone walks in a constant speed, but chooses the direction randomly in each step
- The path (to be more precise, this *stochastic process*) is called a random walk
- We will simulate a 1-D random walk:
  - An object can only move forward or backward at each step
  - When the object hits the wall (upper/lower bounds), it stops walking
  - The time that it hits the wall is called the "stopping time"
  - Important in mathematical finance





HW3 Hints

- Problem 1:
  - Consider “swapping” two characters in the string



Original string:	abcdef
after Step 1:	fbcdea
after Step 2:	fecdba
after Step 3:	fedcba

Then stop!

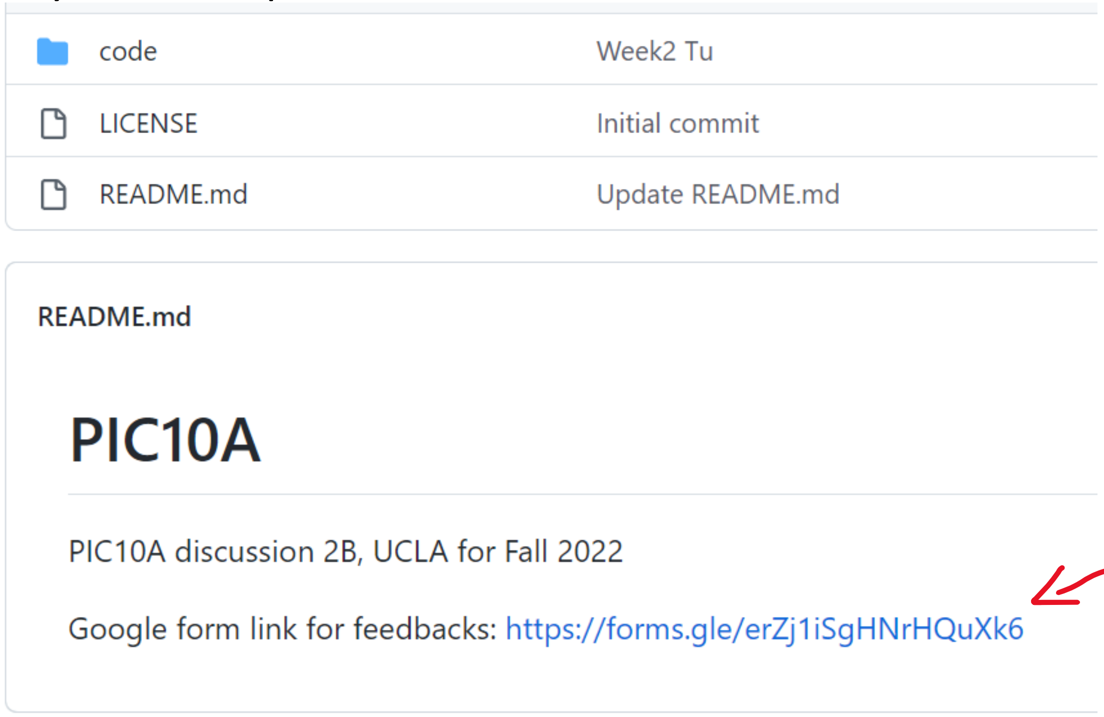
HW3 Hints

- Problem 2:
  - Recall the “Prime Factorization” exercise from Week 3 Thursday
  - We checked a condition “if a factor divides the number” at each iteration
  - Primality test is similar – if the prime factorization algorithm gives you a single number as an output, the number is prime
    - i.e. the same condition can be used for this problem too!

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:



code	Week2 Tu
LICENSE	Initial commit
README.md	Update README.md

README.md

PIC10A

PIC10A discussion 2B, UCLA for Fall 2022

Google form link for feedbacks: https://forms.gle/erZj1iSgHNrHQuXk6

Click this link

UCLA