PIC 10A 1C. Week 7b Exercises. TA: Eric Kim.

1. AT-AT

Write a program that, given a user-given sentence, replaces all 'a' with '@':

```
Enter a sentence: Dan ate apples, haha!
D@n @te @pples, h@h@!
```

Rule: Only use a single **range for-loop** to iterate over the input string.

2. Don't Britta This!

Consider the following class interfaces defined in `person.h` and `studyroom.h`:

```
class Person {
private:
    string myname; unsigned int myage;
public:
    Person(string name, unsigned int age);
    unsigned int get_age();
    string get_name();
};
class StudyRoom {
private:
    vector<Person> myv; // Stores People
public:
    StudyRoom();
    void add_person(Person p); // Add person to room
    double compute_avg_age(); // Avg age of people in study room
};
```

Write a program that does the following:

1. Ask the user for a name and an age.
2. Create a Person object, and add it to the StudyRoom.
3. Ask the user if he/she is finished. If not, then go back to 1.
4. Else, output the average age of the people in the StudyRoom (in fixed precision, with 2 decimal points).

Example:

```
Name? Jeff Winger
Age? 34
Done? y/n: n
Name? Britta Perry
Age? 30
Done? y/n: n
Name? Troy Barnes
Age? 21
Done? y/n: y
Avg age: 28.33
```

Rule: Only use a single **do-while** loop.

3. The Overzealous Censor Officer

Write a program that replaces every other character in a string with '*'. Example:

```
Enter a string: Frankly my dear
F*a*k*y*m* *e*r
```

Rule: Only use a single **for loop**.

4. Blackjack-Lite

Write the following program:

1. Ask the user for an integer, and add it to a running sum.
2. If the current sum is greater than 21, output "Bust!" and exit.
3. Else, if the current sum is equal to 21, output "Blackjack!", and exit.
4. Else, go back to (1).

Example:

```
[You: 0] Integer? 7
[You: 7] Integer? 3
[You: 10] Integer? 11
Blackjack!
```

```
[You: 0] Integer? 6
[You: 6] Integer? 9
[You: 15] Integer? 8
Bust!
```

Rule: Only use a single **while** loop - no other looping constructs allowed.

5. #justpic10Athings

We would like to write a program that, given a string of '+' and '#', computes a more compact version of the original string. For instance, here are some expected outputs:

`"+++##"` `Becomes: "+3#2"`	`"++++#++"` `Becomes: "+4#1+2"`
`"+#"` `Becomes: "+1#1"`	`"+++++"` `Becomes: "+5"`
`""` `Becomes: ""`	`"#++#####+++"` `Becomes: "#1+2#3+4"`

Write a program that, given such a user-inputted string, outputs its compressed version:

```
Enter a string: ++###
+2#3
```

Aside: This is a form of run-length encoding, a technique used to compress data into a smaller (yet equivalent) form. For instance, when you compress a file to a .zip file, the compression program is likely using this principle to achieve a much smaller file size!

As you can imagine, some types of data are more amenable to compression that others. A file with lots of long runs, ie "+++++++++", compress well, whereas files with only short runs, ie "+#+#+#+", compress poorly.