CISS245: Advanced Programming
Assignment 1

Name: _____

Q1. Write the string comparison function

```
int str_cmp(char x[], char y[]);
```

Add this prototype to `mystring.h` and implement the function in `mystring.cpp`.

The function returns 0 when the two strings `x` and `y` are the same, i.e., the characters of `x` up to `'\0'` matches exactly the characters of `y` up to `'\0'`. If they are different, then 1 is returned.

The file `main.cpp` should look like this:

```
#include <iostream>
#include <limits>
#include "mystring.h"

const int MAX_BUF = 1024;


void test_str_cmp()
{
    char s[MAX_BUF];
    char t[MAX_BUF];

    std::cin.getline(s, MAX_BUF);
    std::cin.getline(t, MAX_BUF);

    std::cout << str_cmp(s, t) << std::endl;
}


int main()
{
    int i = 0;
    std::cin >> i;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    switch (i)
```

```
    {
        case 0:
            test_str_cmp();
            break;
    }

    return 0;
}
```

Test 1.

```
0
abc
abc
0
```

Test 2.

```
0
 abc
abc
1
```

Test 3.

```
0
a bc
abc
1
```

Test 4.

```
0
hello world
hello world
0
```

You are strongly advised to try more test cases of your own.

Q2. Write the string copy function:

```
void str_cpy(char x[], char y[]);
```

The prototype should be added to `mystring.h` while the implementation of the function is in `mystring.cpp`.

The function copies `y` to `x`. For instance, if `y` is the string `"hello world"`, after calling `str_cpy(x, y)`, then `x` is `"hello world"`.

The following `main.cpp` must be included

```
#include <iostream>
#include <limits>
#include "mystring.h"

const int MAX_BUF = 1024;


void test_str_cmp()
{
    // earlier test function
}


void test_str_cpy()
{
    char x[MAX_BUF];
    char y[MAX_BUF];

    std::cin.getline(y, MAX_BUF);
    str_cpy(x, y);
    std::cout << x << std::endl;
    return;
}


int main()
{
    int i = 0;
    std::cin >> i;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    switch (i)
    {
        case 0:
```

```
            test_str_cmp();
            break;
        case 1:
            test_str_cpy();
            break;
    }
    return 0;
}
```

Test 1.

```
1
hello world
hello world
```

Test 2.

```
1
1 2 3
1 2 3
```

You are strongly advised to try more test cases of your own.

Q3. Write a "find character in string" function

```
int str_chr(char x[], char c);
```

that returns the index where `c` first occurs in `x`. If `c` does not occur in `x`, then -1 is returned.

For instance if `x` is `"hello world"` and `c` is `'o'`, then `str_chr(x, c)` returns 4.

```
#include <iostream>
#include <limits>
#include "mystring.h"

const int MAX_BUF = 1024;


void test_str_cmp()
{
    // earlier test function
}


void test_str_cpy()
{
    // earlier test function
}


void test_str_chr()
{
    char x[MAX_BUF];
    char c;

    std::cin.getline(x, MAX_BUF);
    std::cin >> c;

    std::cout << str_chr(x, c) << std::endl;
    return;
}


int main()
{
    int i = 0;
    std::cin >> i;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

```
    switch (i)
    {
        // earlier cases
        case 2:
            test_str_chr();
            break;
    }
    return 0;
}
```

Test 1.

```
2
hello world
w
6
```

Test 2.

```
2
hello world???
?
11
```

You are strongly advised to try more test cases of your own.

Q4. Write a "find string in string" function

```
int str_str(char x[], char y[]);
```

that returns the index where `y` first occurs in `x`. If `y` does not occur in `x`, then -1 is returned.

For instance if `x` is `"hello world"` and `y` is `" wor"`, then `str_str(x, y)` returns 5.

```
#include <iostream>
#include <limits>
#include "mystring.h"

const int MAX_BUF = 1024;


// earlier test functions


void test_str_str()
{
    char x[MAX_BUF];
    char y[MAX_BUF];

    std::cin.getline(x, MAX_BUF);
    std::cin.getline(y, MAX_BUF);

    std::cout << str_str(x, y) << std::endl;
    return;
}


int main()
{
    int i = 0;
    std::cin >> i;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    switch (i)
    {
        // earlier cases
        case 3:
            test_str_str();
            break;
    }
    return 0;
```

```
}
```

Test 1.

```
3
hello world
 wor
5
```

Test 2.

```
3
hello world
or
7
```

Test 3.

```
3
hello world
wor
-1
```

Test 4.

```
3
abc def defg defghi
defgh
13
```

You are strongly advised to try more test cases of your own.

Q5. Write a lowercase function:

```
void str_lower(char x[], char y[]);
```

that copies the character of the string `y` to `x` except that uppercase characters are replaced by lowercase.

For instance if `y` is `"HelLo worlD ... 123!"`, then after calling `str_lower(x, y)`, `x` is
`"hello world ... 123!"`.

```
#include <iostream>
#include <limits>
#include "mystring.h"

const int MAX_BUF = 1024;


// earlier test functions


void test_str_lower()
{
    char x[MAX_BUF];
    char y[MAX_BUF];

    std::cin.getline(y, MAX_BUF);
    str_lower(x, y);

    std::cout << x << std::endl;
    return;
}


int main()
{
    int i = 0;
    std::cin >> i;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    switch (i)
    {
        // earlier cases
        case 5:
            test_str_lower();
            break;
```

```
    }
    return 0;
}
```

Test 1.

```
5
hEllo woRld
hello world
```

Test 2.

```
5
1 2 3 4 5
1 2 3 4 5
```

You are strongly advised to try more test cases of your own.

Q6. Write a string "tokenizing" function:

```
bool str_tok(char x[], char y[], char delimiters[]);
```

that does the following.

If `y` is `"hello world"` and `delimiters` is `" "`, when after calling `str_tok(x, y, delimiters)`, `x` becomes `"hello"`, `y` becomes `"world"`. Furthermore `true` is returned. Basically the characters in `delimiters` is used to cut up the string `y` (once). At this point, if `str_tok(x, y, delimiters)` is called again, `x` becomes `"world"`, `y` becomes `""`, and the function return `true`. If we call `str_tok(x, y, delimiters)` a third time, `x` becomes `""`, `y` stays as `""`, and `false` is returned.

Note that `delimiters` can contain more than one characters. For instance, suppose `y` is
`"hello world,galaxy,universe!"`. If `delimiters` is `" ,"`, then after the first call of `str_tok(x, y, delimiters)`, `x` becomes `"hello"`, `y` becomes `"world,galaxy,universe!"`, and the function return `true`. After the second call of `str_tok(x, y, delimiters)`, `x` becomes `"world"`, `y` becomes `"galaxy,universe!"`, and the function return `true`. After the third call of `str_tok(x, y, delimiters)`, `x` becomes `"galaxy"`, `y` becomes `"universe!"`, and the function return `true`. After the fourth call of `str_tok(x, y, delimiters)`, `x` becomes `"universe!"`, `y` becomes `""`, and the function return `true`. After the fifth call of `str_tok(x, y, delimiters)`, `x` becomes `""`, `y` stays as `""`, and the function return `false`. In this example, `' '` and `','` are used to cut up `y`.

Note that if `y` is `",123"` and `delimiters` is `","`, then on calling `str_tok(x, y, delimiters)`, `x` is `""`, `y` is `"123"`, and `true` is returned.

Note also that because there are multiple characters in delimiters, it's possible for string `y` to be cut up by the delimiter characters in different ways. The character that is used is the one that produces the shortest `x`. For instance in the above case where `y` is
`"hello world,galaxy,universe!"` and `delimiters` is `" ,"`. `y` can be cut up by `' '` (at index 5), by `','` (at index 11), and by `' '` (at index 18). The delimiter character that is actually used is `' '` (at index 5) because that creates the shortest left substring `x`.

```
#include <iostream>
#include <limits>
#include "mystring.h"

const int MAX_BUF = 1024;



// earlier test functions
```

```
void test_str_tok()
{
    char x[MAX_BUF];
    char y[MAX_BUF];
    char delimiters[MAX_BUF] = " ,.";

    std::cin.getline(y, MAX_BUF);
    bool b = str_tok(x, y, delimiters);

    std::cout << b << ' ' << x << ' ' << y << std::endl;
    return;
}


int main()
{
    int i = 0;
    std::cin >> i;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    switch (i)
    {
        // earlier cases
        case 6:
            test_str_tok();
            break;
    }
    return 0;
}
```

Make sure you try some of your own test cases.

Q7. Write a chat bot. The chat bot will learn to recognize the user. Here's a test run:

```
Hi, what is your name?
My name is John.
Hi John. How are you?
```

The chat bot must recognize the name of the user when given responses of the following form:

- `My name is John.`
- `I am John.`
- `I'm John.`
- `John.`

and also when the user enters extraneous spaces such as

- `My  name  is  John.`
- `I  am  John.`
- `I'm  John.`
- `John.`

and also when the user forgot to enter the period (`'.'`) and when the user forgets to capitalize correctly such as

- `my  name  is  john.`
- `i  am  john.`
- `i'm  john.`
- `john.`

It's helpful to write a `str_capitalize()` function such that `str_capitalize(x)` will replace `x[0]` with its uppercase. It's also useful to have a function that strips away left trailing space, `str_lstrip` (the left strip function) so that if you execute `str_lstrip(x)` where `x` is `"  hello world  "`, `x` becomes `"hello world  "`.

It's also convenient to have a corresponding `str_rstrip` (the right strip function). If you execute `str_rstrip(x)` where `x` is `"  hello world  "`, `x` becomes `"  hello world"`.

With the left and right strip functions, it's easy to create `str_strip` (the strip function) which performs both the left and right strip. If you execute `str_strip(x)` where `x` is `"  hello world  "`, `x` becomes `"hello world"`.

Q8. The next thing that the chat bot does is to be helpful to the user:

```
Hi, what is your name?
My name is John.
Hi John. How are you?
I'm sad.
I'm sorry to hear that. Why are you sad?
```

Again, the chat bot must recognize the following variations:

- `I'm sad.`
- `i'm sad.`
- `I am sad.`
- `i am sad.`
- `Sad.`
- `sad`

as well as without period and with extraneous spaces. Your chat bot must also recognize `depressed` and `miserable` (you can add other synonyms if you like). So here's another chat session:

```
Hi, what is your name?
John
Hi John. How are you?
I'm depressed.
I'm sorry to hear that. Why are you depressed?
```

The chat bot must also respond to the following emotions: `happy`, `glad`. (You can add others if you like.) Here's another chat session:

```
Hi, what is your name?
My name is John.
Hi John. How are you?
I'm happy.
I'm glad to hear that. Why are you happy?
```

Your chat bot must also recognize words like `very`, `extremely`, and `really`.

```
Hi, what is your name?
My name is John.
Hi John. How are you?
I'm very sad.
I'm sorry to hear that. Why are you so sad?
```

(Note the new word `so` when the user adds `very`.)

Once the user respond, the program does this:

```
Hi, what is your name?
My name is John.
Hi John. How are you?
I'm very sad.
I'm sorry to hear that. Why are you so sad?
My dog died.
I'm sorry. Tell me about your dog.
```

Here's another execution:

```
Hi, what is your name?
My name is John.
Hi John. How are you?
I'm very sad.
I'm sorry to hear that. Why are you so sad?
My goldfish died.
I'm sorry. Tell me about your goldfish.
```

Your program should respond to a general statement like:

```
Hi, what is your name?
My name is John.
Hi John. How are you?
I'm very sad.
I'm sorry to hear that. Why are you red sad?
My [pet/friend/relative/etc.] died.
I'm sorry. Tell me about your [pet/friend/relative/etc.].
```

On the happy side, your program should do this: You program should respond to a general statement like:

```
Hi, what is your name?
My name is John.
Hi John. How are you?
I'm very happy.
I'm glad to hear that. Why are you so happy?
I got a raise.
I'm glad. Tell me about your raise.
```

or

```
Hi, what is your name?
My name is John.
Hi John. How are you?
I'm very happy.
I'm glad to hear that. Why are you so happy?
I bought a car.
I'm glad. Tell me about your car.
```

You only need to handle `got`, `bought`, and `received`.

Q9. The last thing to add is a certain amount of randomness. Instead of always

```
Hi, what is your name?
```

vary it randomly with

```
Hello, what is your name?
```

(Of course use the random number generator). And instead of

```
I'm sorry to hear that. Why are you sad?
```

or

```
I'm glad to hear that. Why are you happy?
```

vary it randomly with

```
I'm so sorry to hear that. Why are you sad?
```

or

```
I'm so glad to hear that. Why are you happy?
```

and

```
I'm sorry to hear that. Tell me why you are sad.
```

or

```
I'm glad to hear that. Tell me why you are happy.
```

You are of course encouraged to write more string functions to clean up your main program.

This simple chat bot is just an illustration of what you can do to simulate human behavior – and of course it is also a practice on string processing. To create a strong AI bot requires a lot more work. Obviously you need to know lots of CS theory, algorithms, especially algorithms related to language theory and linguistics, and especially AI.