PART I
The purpose of PART I is to practice:

- input/output
- if statements and constructing the appropriate logic that is needed to solve the problem
- writing functions and passing values

Problem 1: Intervals
For this problem, assume all parameters are integers. An interval \([a, b]\) represents the set of numbers between \(a\) and \(b\) inclusive. If \(a > b\), we assume that the interval (set) is empty.

(a) Write a function called intervalEmpty that takes \(a\) and \(b\) as parameters and returns true if \([a, b]\) is empty and false otherwise.

Solution:

```cpp
bool intervalEmpty(int a, int b) {
    return (a>b);
}
```

(b) Write a function called intervalIntersect that takes \(a\), \(b\), \(c\), and \(d\) as parameters, and:

- outputs the intersection of intervals \([a, b]\) and \([c, d]\) as an interval. Use \([1, 0]\) to denote an empty intersection.
- returns the number of elements that belong to both intervals \([a, b]\) and \([c, d]\)
Solution:

```c
int intervalIntersect(int a, int b, int c, int d) {
    int low;
    int high;

    // the larger of a and c
    if (a<=c)
        low=c;
    else
        low=a;

    // the smaller of b and d
    if (b<=d)
        high=b;
    else
        high=d;

    // check if intersection is empty
    if (intervalEmpty(low, high)) {
        low=1;
        high=0;
    }

    cout<<'['<<low<<','<<high<<']';
    return high-low+1;
}
```

(c) In the main function, write a program to prompt the user to input $a$, $b$, $c$, and $d$ and output:

- whether $[a, b]$ is empty or not
- whether $[c, d]$ is empty or not
- the intersection of $[a, b]$ and $[c, d]$ and the number of integer elements in that intersection

Example: If the two intervals are $[1,0]$ and $[2,3]$:

Interval $[1,0]$ is empty
Interval $[2,3]$ is not empty
The intersection of $[1,0]$ and $[2,3]$ is $[1,0]$ with 0 integer elements

Example: If the two intervals are $[1,10]$ and $[5,12]$:

Interval $[1,10]$ is not empty
Interval $[5,12]$ is not empty
The intersection of $[1,10]$ and $[5,12]$ is $[5,10]$ with 6 integer elements

Example: If the two intervals are $[1,2]$ and $[4,6]$:

Interval $[1,2]$ is not empty
Interval $[4,6]$ is not empty
The intersection of $[1,2]$ and $[4,6]$ is $[1,0]$ with 0 integer elements

Solution:
int main() {
    int a;
    int b;
    int c;
    int d;
    cout<<"Input 4 integers to make two intervals [a,b] and [c,d]: ";
    cin>>a;
    cin>>b;
    cin>>c;
    cin>>d;
    cout<<"Interval ["<<a<<','<<b<<"] is ";
    if (intervalEmpty(a,b))
        cout<<"empty\n";
    else
        cout<<"not empty\n";
    cout<<"Interval ["<<c<<','<<d<<"] is ";
    if (intervalEmpty(c,d))
        cout<<"empty\n";
    else
        cout<<"not empty\n";
    cout<<"The intersection of ["<<a<<','<<b<<"] and ["<<c<<','<<d<<"] is ";
    int n=intervalIntersect(a,b,c,d);
    cout<<"with "<<n<<" integer elements\n";
}

PART II
The purpose of PART II is to practice:

- loops
- simple conditionals
- writing functions and passing values

Problem 2: Fair and Square...

(a) Write a function called square2 that takes an integer \( n \) as a parameter and returns the sum of the first \( n \) odd numbers starting from 1 to and ending in \( 2n-1 \).

Solution: here are two possible solutions.

int square2(int n) {
    int s=0;
    for (int i=1; i<=n; i=i+1) // loop n times
        s=s+2*i-1; // the i-th odd number is 2i-1
    return s;
}
int square2(int n) {
    int s=0;
    int i=1; // start with the first odd number
    while (i<=2*n-1) { // as long as less of equal to 2n-1
        s=s+i;
        i=i+2; // increment by 2 to get the next odd number
    }
    return s;
}

(b) Compare this function to the function square that we have seen in class. To do this, verify in main that both functions return the same value for all $n = 0 \ldots 100$. One way is to print the values side by side in a loop. [optional] Try to find a better way using a loop and an if statement.

**Solution:** here are two solutions. The first outputs the results side by side, the second uses if.

```cpp
int main() {
    for (int i=0; i<=100; i=i+1) {
        cout<<square(i)<<' ';
        cout<<square2(i)<<'\n';
    }
}
```

```cpp
int main() {
    bool agree=true;
    for (int i=0; i<=100; i=i+1)
        if (square(i)!=square2(i)
            agree=false;
    if (agree)
        cout<<"both functions agree on all inputs from 0 to 100\n";
    else
        cout<<"the two functions do not agree on all inputs from 0 to 100\n";
}
```

Where the square function is as seen in class:

```cpp
int square(int n) {
    return n*n;
}
```
Problem 3: Square root
We have seen in class a function to compute the square root of a number \( x \) based on Newton’s method:

```cpp
bool closeEnough(float a, float b) {
    return (-0.001<=a-b && a-b<=0.001);
}

float sqrt(float x, float guess) {
    while (!closeEnough(guess*guess, x) {
        cout<<guess<<'\n'; //not needed, but to see
        //how guess is changing
        guess = (guess + x/guess)/2;
    return guess;
}

Implement a sqrt function based on the following idea: we bound the square root of \( x \) from the left and the right. Initially, the square root of \( x \) must satisfy:

\[
0 \leq \sqrt{x} \leq \max(x, 1)
\]

So if we initially let \( a = 0 \) and \( b = \max(x, 1) \), then the square root of \( x \) is in the interval \([a, b]\). To assign \( b \), an if statement can compare \( x \) to 1. Now let \( m \) be the middle point of the interval \([a, b]\) (we can use the average function to find it). While \( m^2 \) is not close enough to \( x \) we repeatedly perform the following (otherwise, we return \( m \)):

- if \( m^2 \leq x \), we assign \( a \) the value of \( m \), i.e. the interval becomes \([m, b]\)
- if \( m^2 \geq x \), we assign \( b \) the value of \( m \), i.e. the interval becomes \([a, m]\)
- update \( m \) to be the middle of the interval \([a, b]\)

Therefore, in addition to \( m \), we need two variables to keep track of how the interval is changing.

Note 1: We exit the loop when \( m^2 \) is close enough to \( x \), say within 0.001.

Note 2: The size of the bounding interval is halfed each time, but mathematically Newton’s method converges faster. To check this, insert a cout statement as illustrated above to track the iterations, and try both functions to compare the number of iterations (for the first version, you may start with \( x \) itself as the guess).

Example: Here’s how the interval and \( m \) change when computing the square root of \( x = 0.5 \).
Solution:

```cpp
float average(float x, float y) {
    return (x+y)/2;
}
```

```cpp
float sqrt(float x) {
    float a=0;
    float b;
    if (x>1)
        b=x;
    else
        b=1;
    float m=average(a,b); //or simply (a+b)/2;
    while (!closeEnough(m*m,x)) {
        if (m*m<=x)
            a=m;
        if (m*m>=x)
            b=m;
        m=average(a,b); //or simply (a+b)/2;
    }
    return m;
}
```

Instructions to submit homework
Have a separate program for each problem. For each program, upload it to the following website:
http://www.cs.hunter.cuny.edu/~saad/courses/c++/taxi.html
If your program compiles successfully, you will receive a 5-digit TAXI code. Put this TAXI code as a comment in the beginning of the corresponding C code file.

```cpp
// TAXI code here
```

```cpp
#include <iostream>
using ...
```

```
//the rest of the file...
```

Submit the file through Blackboard. You will find an appropriate column to upload it in the Grade Center under the Assignments section.