Problem 1. What can you say about the sign of the expression:

$$f(u) = \Big(2u+1\Big)ln\Big(1+\frac{1}{u}\Big) - 2 \quad u > 0.$$

Solution.

We are going to solve this problem graphically. First step - asymptote calculation. As far as we know there are three types of asymptotes(horizontal, vertical and oblique).

Vertical asymptote is obviously $u = 0$, because denominator of $\frac{1}{u}$ could not be equal to zero, this lead to asymptote equation listed above.

Horizontal aymptote calculation:

$$\lim_{u\to\infty}\Big(2u+1\Big)ln\Big(1+\frac{1}{u}\Big) = \lim_{u\to\infty}\Big(2uln\Big(1+\frac{1}{u}\Big) + ln\Big(1+\frac{1}{u}\Big)\Big) = 2\lim_{u\to\infty}\Big(uln\Big(1+\frac{1}{u}\Big)\Big) +$$

$$+\lim_{u\to\infty}\Big(ln\Big(1+\frac{1}{u}\Big)\Big) = 2\lim_{u\to\infty}\Big(uln\Big(1+\frac{1}{u}\Big)\Big) + 0 = 2ln\Big(\lim_{u\to\infty}\Big(u\Big(1+\frac{1}{u}\Big)\Big) = [u>0] =$$

$$2ln\Big(\lim_{u\to\infty}\Big(1+\frac{1}{u}\Big)^u\Big) = [\lim_{u\to\infty}\Big(1+\frac{1}{u}\Big)^u = e] = 2ln(e) = 2.$$

Using calculations above, we have come to the next conclusion: $\lim_{u\to\infty} f(u) = 2 - 2 = 0$. So, horizontal aymptote equation is $y = 0$.

Oblique asymptote is equal to horizontal asymptote: $y = 0$. Now, to obtain sign of the expression we need to calculate value of $f(u)$ function in u-point such as $u > 0$.

For this purpose we choose point $u = \frac{1}{2}$. Calculation of value of f function in this point are listed below: $f(\frac{1}{2}) = 2ln(3) - 2 = 2(ln(3) - 1)$. The only remain thing — is to obtain the sign of $ln(3) - 1$ expression:

$$ln(3) \quad v \quad 1$$

$$3 \quad v \quad e$$

$$3 \quad > \quad e$$

So, the sign of the expression $f(\frac{1}{2}) > 0$ which lead to conclusion that some part of the $f(y)$ function is higher than zero.

In this section we will answere the question: will grapth of $f(u)$ function intersect X axis on the positive infinity or not.

Expand the function $f(u)$ in a Taylor series at infinity. First of all notice, that Taylor expansion for $\frac{1}{u}$, where u tends to infinity is equal to Taylor expansion for $t = \frac{1}{u}$, where to tends to zero, thats why:

$$ln\Big(1+\frac{1}{u}\Big)|_{x\to\infty} = \Big[t = \frac{1}{u}\Big] = ln\Big(1+t\Big)|_{t\to 0} = \Big[Taylor.Series\Big] = t - \frac{t^2}{2} + \frac{t^3}{3} + o\Big(t^3\Big) =$$

$$= \frac{1}{u} - \frac{1}{2u^2} + \frac{1}{3u^3} + o(\frac{1}{u^3}),$$

thus the expansion of the Taylor series has the form:

$$f(u) = \Big(2u+1\Big)ln\Big(1+\frac{1}{u}\Big) - 2 = \Big(2u+1\Big)\Big(\frac{1}{u} - \frac{1}{2u^2} + \frac{1}{3u^3} + o\Big(\frac{1}{u^3}\Big)\Big) - 2 =$$

$$= 2 - \frac{1}{u} + \frac{2}{3u^2} + \frac{1}{u} + \frac{1}{2u^2} + \frac{1}{3u^3} - 2 + o\Big(\frac{1}{u^3}\Big) = \frac{1}{6u^2} + \frac{1}{3u^3} + o\Big(\frac{1}{u^3}\Big).$$

From the last equation we can see that value of $f(u)$ function on infinty is always bigger then zero $\Big(u > 0 \to \Big(\frac{1}{6u^2} + \frac{1}{3u^3}\Big) > 0\Big)$. Besides, the second derivative $f(u)'' = \frac{x^2}{(x+1)^2}$ is always bigger then zero which means that $f(u)$ graph concave. This means that sign of the $f(u)$ expression is positive for $u > 0$.

Answer: sign of the $f(u)$ expression is positive for $u > 0$.

Problem 2. Please indicate the issues you see from the code above. Fix those issues and rewrite the class definition

Solution:

```
class Complex
{
public:
  Complex();
  Complex(const double& re);
  Complex(const double& re, const double& im);
  ~Complex() {}

  friend const Complex& operator++(Complex& i);
  friend const Complex& operator++(Complex& i, int);
  friend const Complex operator+(const Complex& left, const Complex& right);

  friend std::ostream & operator<<(std::ostream & os, const Complex & obj);

private:
  double re_;
  double im_;
};


Complex::Complex() : re_(0), im_(0) { }

Complex::Complex(const double & re) :
        re_(re), im_(0) { }

Complex::Complex(const double & re, const double & im):
        re_(re), im_(im) { }

const Complex & operator++(Complex & i)
{
  ++i.re_;
  return i;
}

const Complex & operator++(Complex & i, int)
{
  Complex oldValue(i);
  ++i.re_;
  return oldValue;
}

const Complex operator+(const Complex & left, const Complex & right)
{
  return Complex(left.re_ + right.re_, left.im_ + right.im_);
}

std::ostream & operator<<(std::ostream & os, const Complex & obj)
{
  os << obj.re_ << " + i * " << obj.im_;
```

```
  return os;
}
```

- c). I wrote int instead of double automatically and do not noticed the error.
- d). When one pass argument as a value, it creates a copy of the object. And when one pass argument as a constant reference it deals with object (not copy) which could not be changed, which in current task is apropriate. But it does not play a significant role in current task.

Problem 3. Military tanks ride only by asphalt roads on the planet Barsoom. While moving, tanks destroy roadbed under themselves. A very modern tank Armata made a trip from the Capital to Zodanga city. We know that minimal road width on the Barsoom didn't change after this trip. What values can Armata's width take?

Solution.

Main idea, which is taken into account is: a map of the cities on the planet Barsoom could be represented as an oriented graph, each edge of which store roads width as na additional parameter.

But before algorithm discussion, first thing we need to do — we need randomly generate roads between the cities. This tasks solved in *Generator* class.

Depending on the number of input number of cities(n) and number of roads(N) we has two different algorithms of roads generation:

- If $N > n(n-1)/2$, or input number of roads is bigger them maximum value of roads between n cities. In this case we generate connected graph win random roads width.
- If $N < n(n-1)/2$, we generate two random numbers in range $[0, n)$(Indexes of start and finish of the road) and a random value of roads width. If this values are already generated we generate a new one untill all number of generated roads is equal to N.

When we have read the conditions from generated file, for task solution we must perform next steps (this is main algorithm in this task):

1. Construct a graph of the map, based on the data reading from file. And in the same time obtain minimum road width.
2. Capital city always has index 0, while Zodanga has index (n-1)(It could be easily changed in the code).
3. Peform the solution procedure.

Solution procedure consists of several steps:

- Vector each element of which stores maximum possible width from start point to the current node (Initially filled with negative values).
- Vector of bool, indicates has been tank in current node or not.
- Width in start point is equal to zero.
- Looping through all nodes
  - Find the index of node with the biggest width from all not visited yet nodes.
  - Set node with this index as visited.
  - Update width in current node.

When we update width in current node some some rules must be taken into account:

1. When leaving the starting node to its neighbors width of neighbors is equal to the difference of the road, and the minimum road width.
2. Along the way saved the minimum possible value of the width (that the tank could pass entirely on the way).
3. In the final node, we choose the maximum from all possible minimum widths lead to this node.

```
void BFSFindMaxWidth(start, finish)
{
  ...
  Width in start point is equal to zero.
  for (all nodes)
  {
    stepFromId = Find node with maximum width;
    Mark stepFromId as visited.
    for (Looping through all the neighbours)
    {
      if (could visit this neighbour)
      {
        if (node Id == finishId)
          ...
        if (node Id == startId)
          ...
        else if (node in the path)
          ...
      }
    }
  max Armata's width = width at finish;
}
```

Algorithm complexity.

Let us assume, that the each node has b neighbours in average. So, we have two nested loops 'for' and the time complexity is about $O(n*b)$. In the worst case graph is connected and each node has *no* neighbours. So, we have two nested loops 'for' and the time complexity is about $O(n^2)$, where n — number of nodes in graph.

Space complexity is about $O(n)$ because there is only two additional vectors, size of each of them is equal to n:

- Vector each element of which stores maximum possible width from start point to the current node. Size is n.
- Vector of bool, indicates has been tank in current node or not. Size is n.

Answer: Time dependancy is about $O(n*b)$, where n —number of the cities, b — is the average value of neighbors, in whorste case $O(n^2)$. Space complexity is about $O(n)$.