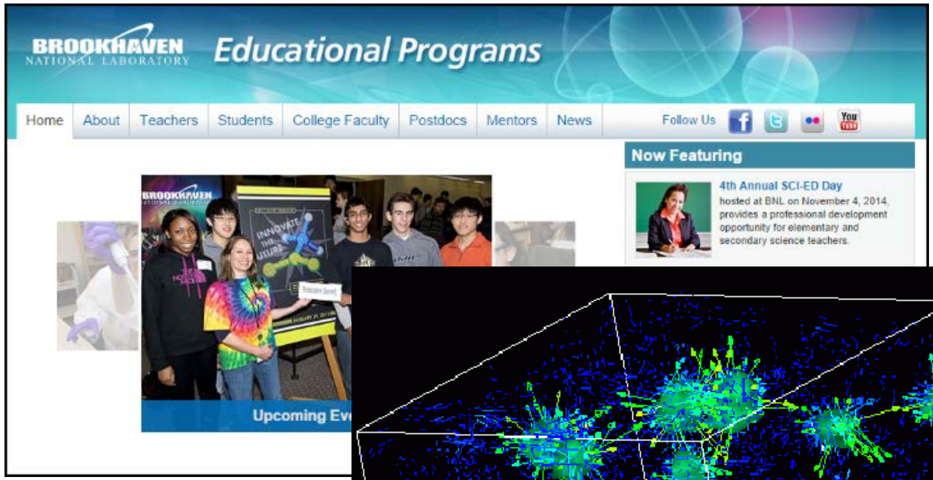
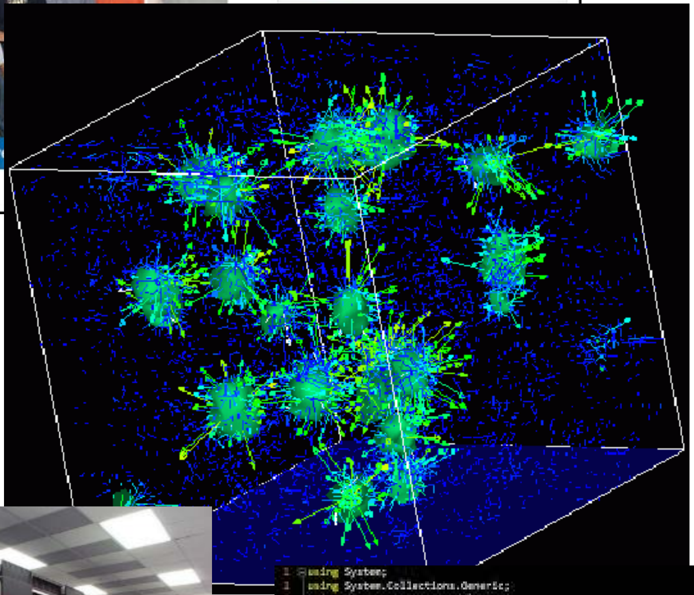
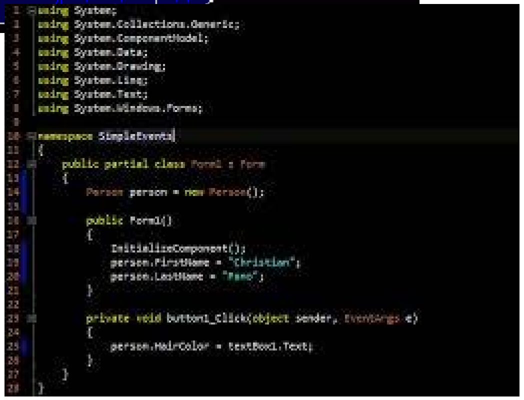


4th Annual SCI-ED Day hosted at BNL on November 4, 2014, provides a professional development opportunity for elementary and secondary science teachers.





```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SimpleEvents
{
    public partial class Form1 : Form
    {
        Person person = new Person();

        public Form1()
        {
            InitializeComponent();
            person.FirstName = "Christian";
            person.LastName = "Removed";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            person.HairColor = textBox1.Text;
        }
    }
}
```

Survey of Scientific Computing (SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

Exam 2
Total of 100 points

5 pts

1. Solve a given 4x4 system

In the **q01** folder, edit the C++ console application to calculate the four unknowns for this system of equations:

$$x_1 + 2x_2 + x_3 - x_4 = 5$$

$$3x_1 + 6x_2 + 4x_3 + 4x_4 = 16$$

$$4x_1 + 8x_2 + 3x_3 + 4x_4 = 22$$

$$2x_1 + 4x_2 + x_3 + 5x_4 = 15$$

20 pts

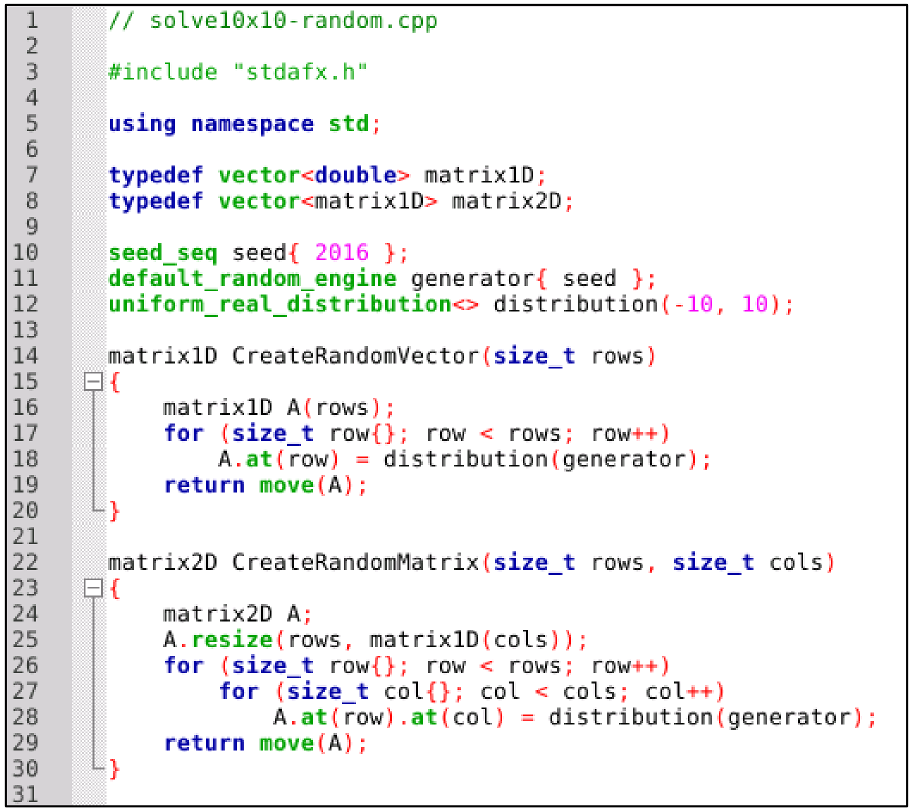
2. Create and solve a random 10x10 system

In the **q02** folder, edit the C++ console application to **generate** and solve a system of 10 random linear equations having 10 unknowns using Cramer's Rule

The coefficients and values should be taken from a uniform **real** distribution between [-10,10) with initial seed of 2016

Display all numbers with **four** digits of precision to the right of the decimal point

Copy any needed relevant code from the q01 solution files

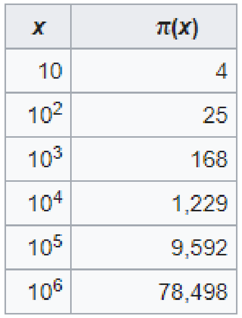


```cpp
// solve10x10-random.cpp

#include "stdafx.h"

using namespace std;

typedef vector<double> matrix1D;
typedef vector<matrix1D> matrix2D;

seed_seq seed{ 2016 };
default_random_engine generator{ seed };
uniform_real_distribution<> distribution(-10, 10);

matrix1D CreateRandomVector(size_t rows)
{
    matrix1D A(rows);
    for (size_t row{}; row < rows; row++)
        A.at(row) = distribution(generator);
    return move(A);
}

matrix2D CreateRandomMatrix(size_t rows, size_t cols)
{
    matrix2D A;
    A.resize(rows, matrix1D(cols));
    for (size_t row{}; row < rows; row++)
        for (size_t col{}; col < cols; col++)
            A.at(row).at(col) = distribution(generator);
    return move(A);
}

```

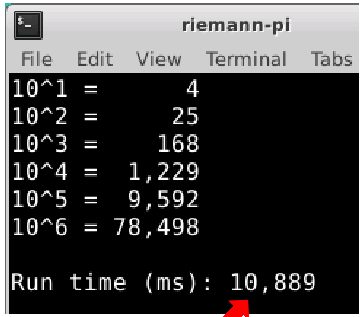
15 pts

3. Calculate Riemann’s $\pi(x)$

In the **q03** folder, edit the C++ console application to calculate and display $\pi(x)$ for each successive power of 10 from 1 to 6 inclusive

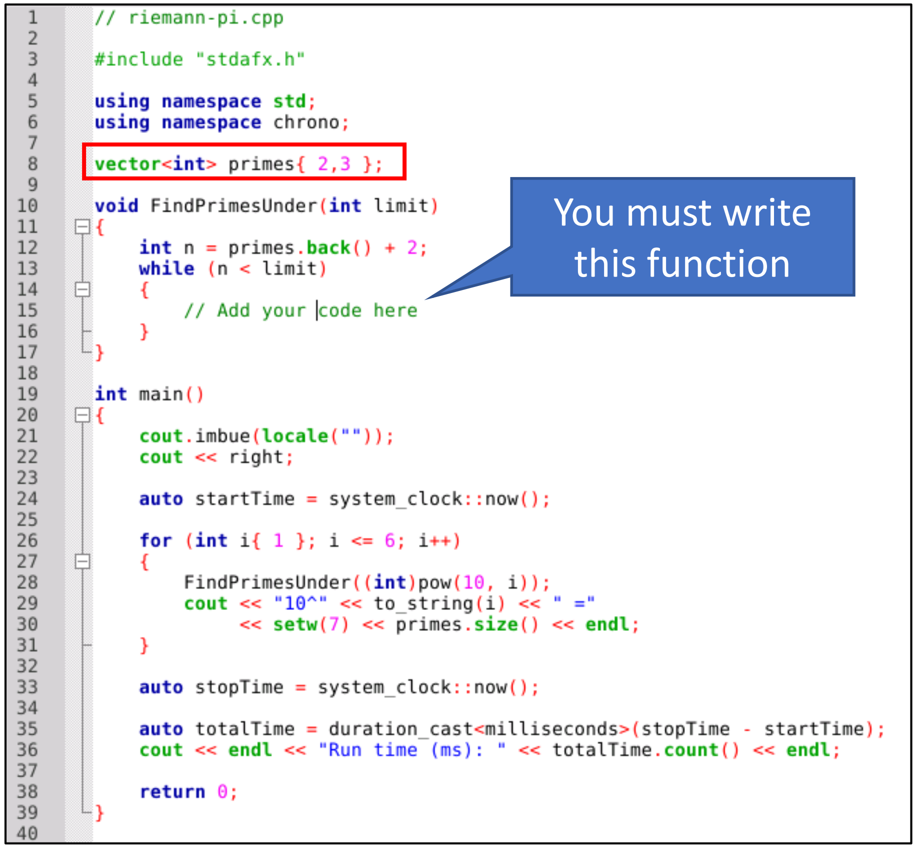


x	*π*(*x*)
10	4
10^2	25
10^3	168
10^4	1,229
10^5	9,592
10^6	78,498



```
10^1 =       4
10^2 =      25
10^3 =     168
10^4 =   1,229
10^5 =   9,592
10^6 =  78,498

Run time (ms): 10,889
```

Special bonus points if your code runs faster than mine!

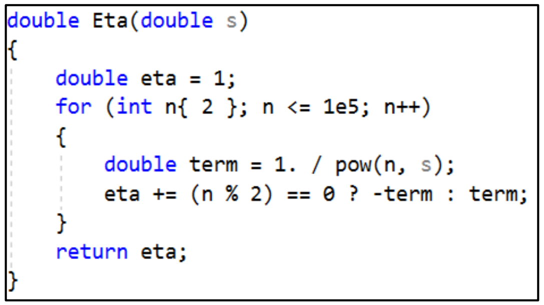


```
// riemann-pi.cpp

#include "stdafx.h"

using namespace std;
using namespace chrono;

vector<int> primes{ 2,3 };

void FindPrimesUnder(int limit)
{
    int n = primes.back() + 2;
    while (n < limit)
    {
        // Add your code here
    }
}

int main()
{
    cout.imbue(locale(""));
    cout << right;

    auto startTime = system_clock::now();

    for (int i{ 1 }; i <= 6; i++)
    {
        FindPrimesUnder((int)pow(10, i));
        cout << "10^" << to_string(i) << " ="
             << setw(7) << primes.size() << endl;
    }

    auto stopTime = system_clock::now();

    auto totalTime = duration_cast<milliseconds>(stopTime - startTime);
    cout << endl << "Run time (ms): " << totalTime.count() << endl;

    return 0;
}
```

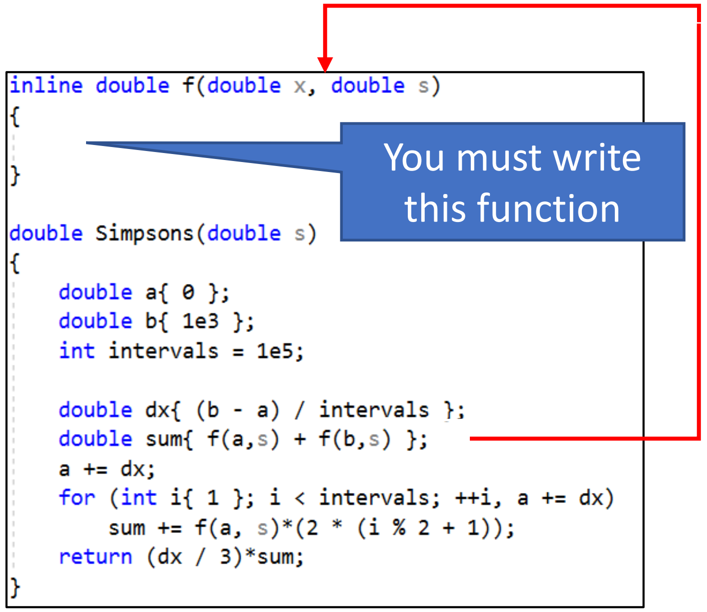
10 pts

4. Calculate Gamma from Eta

$$\eta(s) = \frac{1}{\Gamma(s)} \int_0^\infty \frac{x^{s-1}}{e^x + 1} dx$$



```cpp
double Eta(double s)
{
    double eta = 1;
    for (int n{ 2 }; n <= 1e5; n++)
    {
        double term = 1. / pow(n, s);
        eta += (n % 2) == 0 ? -term : term;
    }
    return eta;
}
```



```cpp
inline double f(double x, double s)
{

}

double Simpsons(double s)
{
    double a{ 0 };
    double b{ 1e3 };
    int intervals = 1e5;

    double dx{ (b - a) / intervals };
    double sum{ f(a,s) + f(b,s) };
    a += dx;
    for (int i{ 1 }; i < intervals; ++i, a += dx)
        sum += f(a, s)*(2 * (i % 2 + 1));
    return (dx / 3)*sum;
}
```

In the **q04** folder, edit the C++ console application to display 5! using only $\eta(s)$ **and the value of the integral**

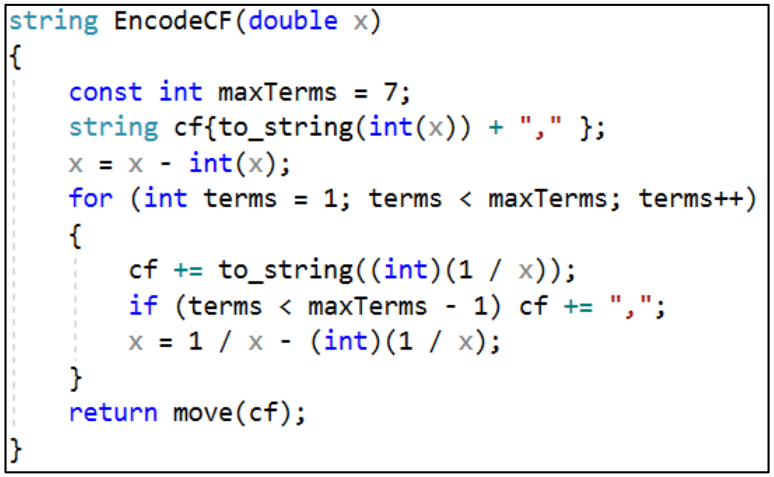
15 pts

5. Expand a Standard CF

In the **q05** folder, edit the C++ console application to display the first **seven** terms of the standard continued fraction encoding for each value of x_n:

$$x = \frac{1 + \sqrt{4n^2 - 4n + 5}}{2}$$

$$n \in \mathbb{Z}^+, n < 10$$



```
string EncodeCF(double x)
{
    const int maxTerms = 7;
    string cf{to_string(int(x)) + "," };
    x = x - int(x);
    for (int terms = 1; terms < maxTerms; terms++)
    {
        cf += to_string((int)(1 / x));
        if (terms < maxTerms - 1) cf += ",";
        x = 1 / x - (int)(1 / x);
    }
    return move(cf);
}
```

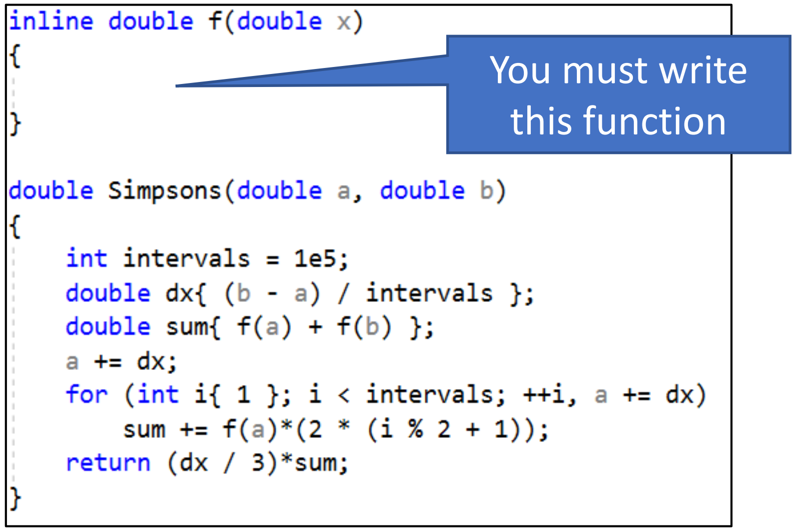
This is something I personally discovered. It reminds me of the Golden Ratio

10 pts

6. Integrate the Standard Normal

In the **q06** folder, edit the C++ console application to use Simpson's Rule to *reasonably* estimate the area under the **standard normal** curve from -1 to 1

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



```
inline double f(double x)
{

}

double Simpsons(double a, double b)
{
    int intervals = 1e5;
    double dx{ (b - a) / intervals };
    double sum{ f(a) + f(b) };
    a += dx;
    for (int i{ 1 }; i < intervals; ++i, a += dx)
        sum += f(a)*(2 * (i % 2 + 1));
    return (dx / 3)*sum;
}
```

Is this area the same as the probability of a normally distributed random variable falling within the **first** standard deviation away from the mean?

5 pts

7. Decrypt Ciphertext

In the **q07** folder, edit the C++ console application to decrypt the file “ciphertext.txt”

Use the tools & techniques from Session 14

10 pts

8. Finding an Open Reading Frame (ORF)

https://en.wikipedia.org/wiki/Open_reading_frame

http://vlab.amrita.edu/?sub=3&brch=273&sim=1432&cnt=1

DNA (Deoxyribonucleic acid) is the genetic material that contains all the genetic information in a living organisms. The information is stored as genetic codes using adenine (A), guanine (G), cytosine(C) and thymine (T). During the transcription process, DNA is copied to mRNA. Each of these base pairs will bond with a sugar and phosphate molecule to form a nucleotide. **Three nucleotides** that code for a particular amino acid during translation is called as a **codon**. An Open Reading Frame (ORF) goes from the **start codon** to a **stop codon** and encodes a specific protein. By analyzing the ORF we can predict the possible amino acids that might be produced during translation.

Given a sequence, create 6 different reading frames labeled +1, +2, +3, -1, -2 and -3:

- The first reading frame is obtained by considering the sequence in words of 3
- The second reading frame is formed after skipping the first nucleotide and then grouping the remaining sequence into words of 3 nucleotides
- The third reading frame is formed after skipping the first 2 nucleotides and then grouping the remaining sequence into words of 3 nucleotides
- The other negative 3 reading frames can be found only after finding the **reverse complement of the sequence**
- The same process as for the +1, +2 and +3 strands is repeated for the -1, -2 and -3 strands, but now using the reverse complement sequence

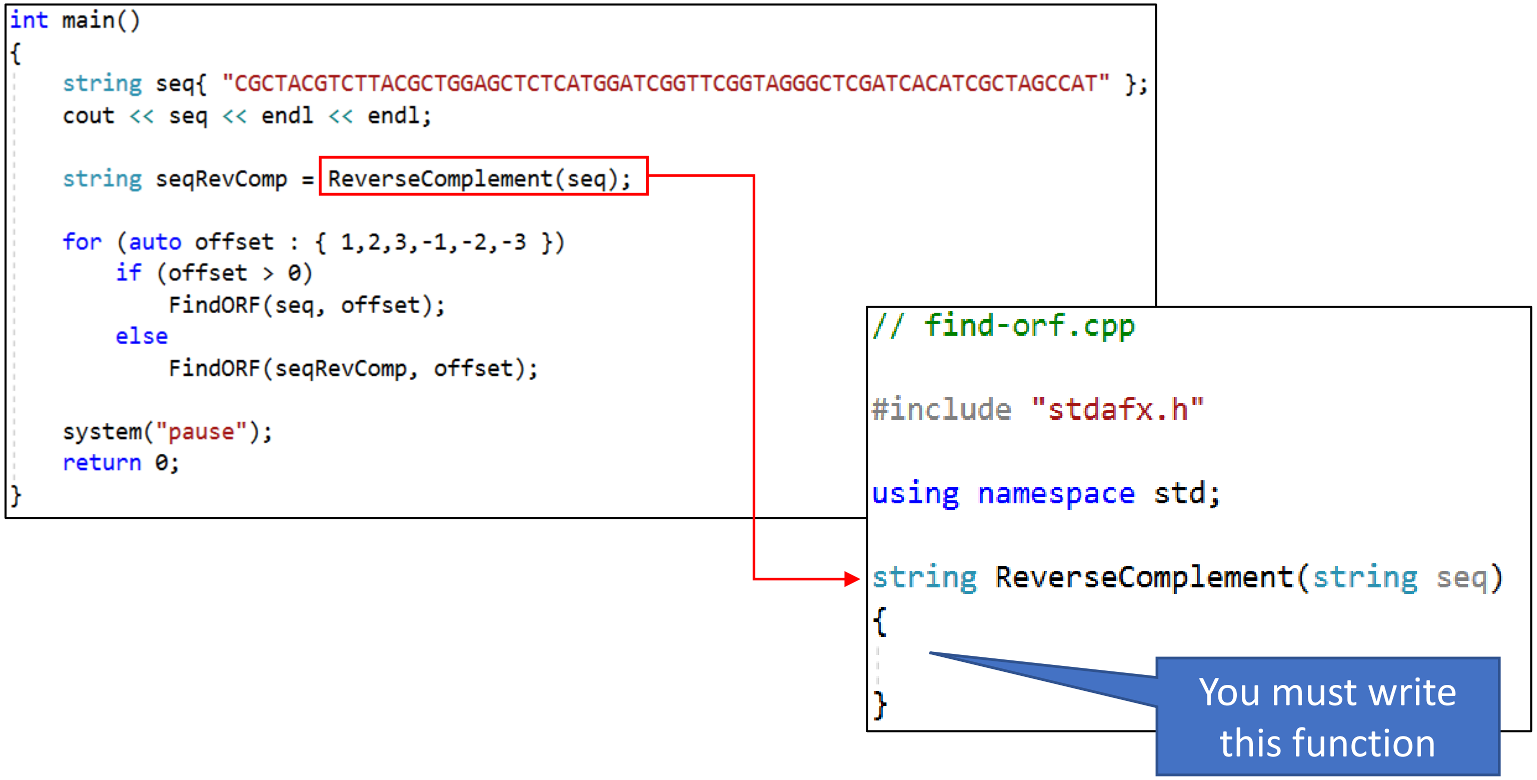
To identify each open reading frame (ORF):

- Find the start codon and stop codons in each reading frame
- Display the sequence stretch beginning with a start codon and ending in a stop codon

10 pts

8. Finding an Open Reading Frame (ORF)

http://vlab.amrita.edu/?sub=3&brch=273&sim=1432&cnt=1



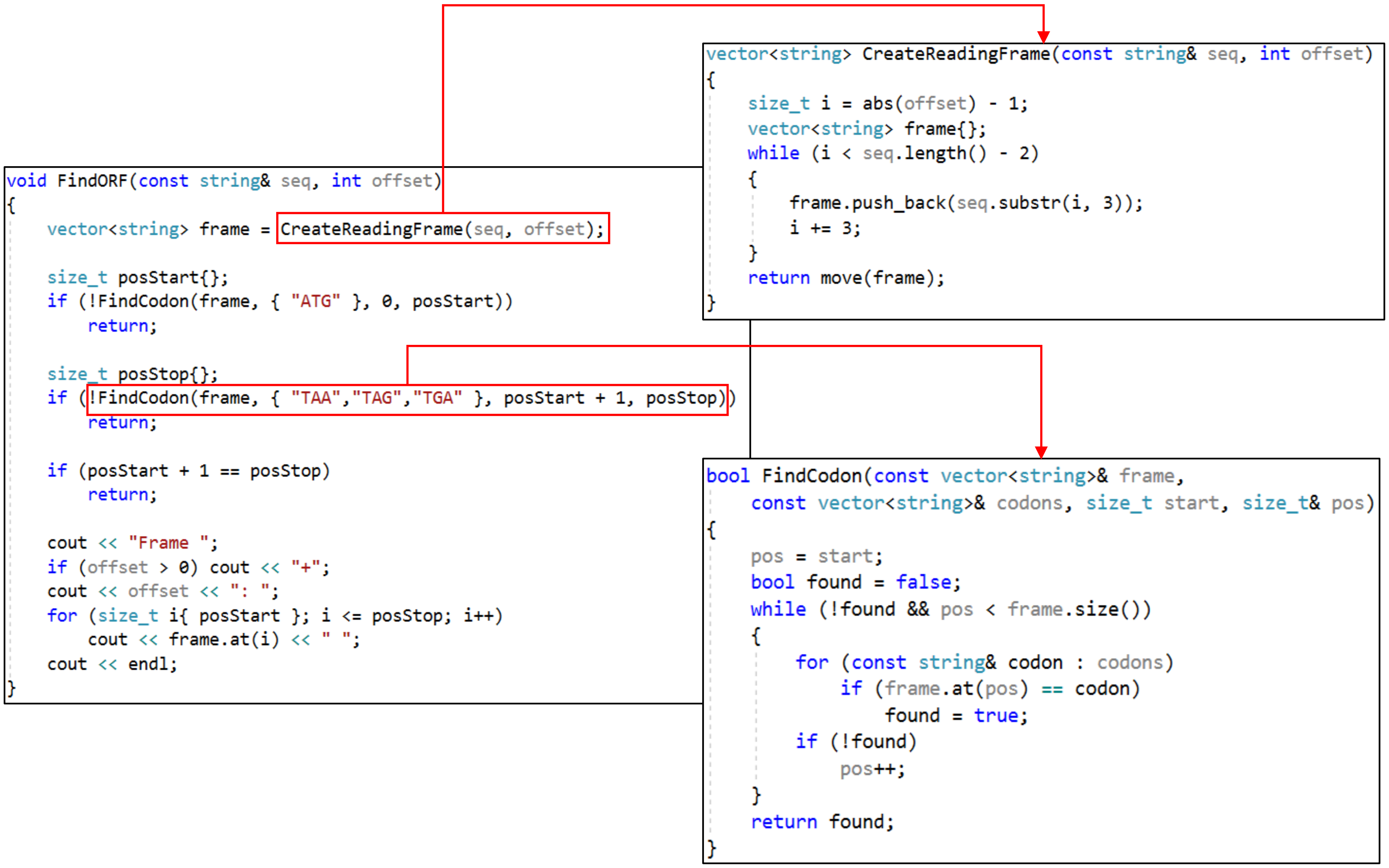
```
int main()
{
    string seq{ "CGCTACGTCTTACGCTGGAGCTCTCATGGATCGGTTCGGTAGGGCTCGATCACATCGCTAGCCAT" };
    cout << seq << endl << endl;

    string seqRevComp = ReverseComplement(seq);

    for (auto offset : { 1,2,3,-1,-2,-3 })
        if (offset > 0)
            FindORF(seq, offset);
        else
            FindORF(seqRevComp, offset);

    system("pause");
    return 0;
}
```

```
// find-orf.cpp

#include "stdafx.h"

using namespace std;

string ReverseComplement(string seq)
{

}
```

In the **q08** folder, edit the existing C++ console application to generate the **reverse complement** of a given DNA sequence

10 pts

8. Finding an Open Reading Frame (ORF)



```cpp
void FindORF(const string& seq, int offset)
{
    vector<string> frame = CreateReadingFrame(seq, offset);

    size_t posStart{};
    if (!FindCodon(frame, { "ATG" }, 0, posStart))
        return;

    size_t posStop{};
    if (!FindCodon(frame, { "TAA","TAG","TGA" }, posStart + 1, posStop))
        return;

    if (posStart + 1 == posStop)
        return;

    cout << "Frame ";
    if (offset > 0) cout << "+";
    cout << offset << ": ";
    for (size_t i{ posStart }; i <= posStop; i++)
        cout << frame.at(i) << " ";
    cout << endl;
}
```

```cpp
vector<string> CreateReadingFrame(const string& seq, int offset)
{
    size_t i = abs(offset) - 1;
    vector<string> frame{};
    while (i < seq.length() - 2)
    {
        frame.push_back(seq.substr(i, 3));
        i += 3;
    }
    return move(frame);
}
```

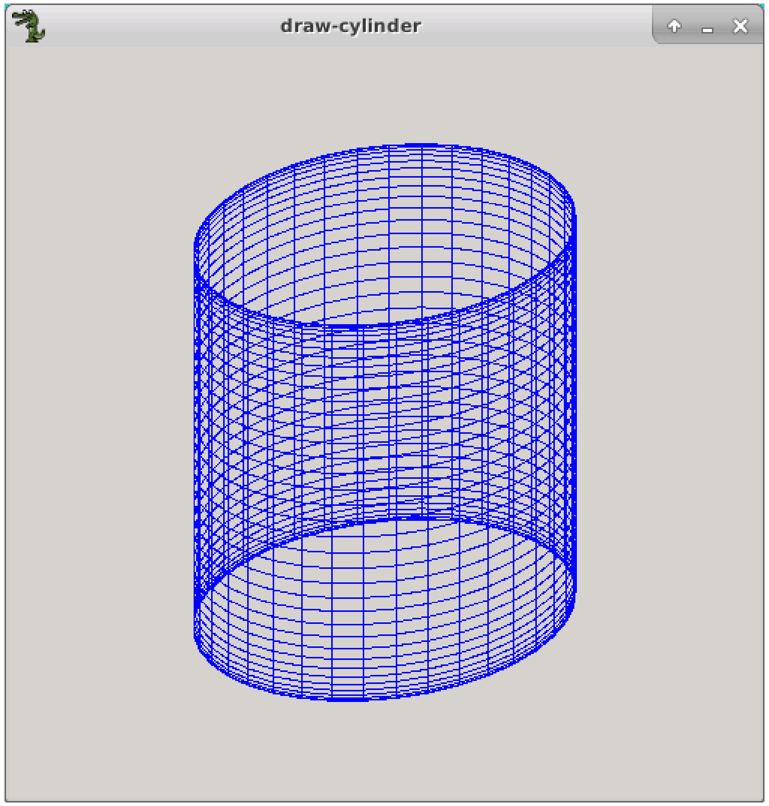
```cpp
bool FindCodon(const vector<string>& frame,
    const vector<string>& codons, size_t start, size_t& pos)
{
    pos = start;
    bool found = false;
    while (!found && pos < frame.size())
    {
        for (const string& codon : codons)
            if (frame.at(pos) == codon)
                found = true;
        if (!found)
            pos++;
    }
    return found;
}
```

5 pts

9. Draw a 3D Cylinder

In the **q09** folder, edit the C++ Allegro application to draw an open-ended 3D cylinder with radius 100 and length 200

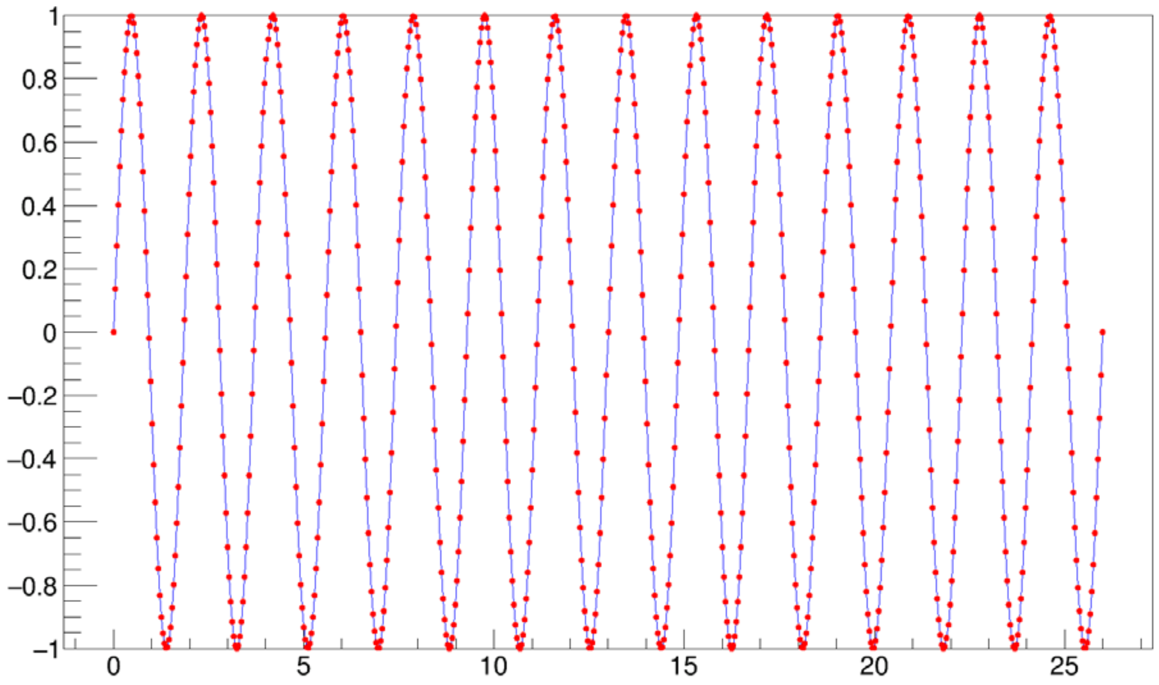
Make it a wireframe drawing. No back face culling or facet shading is required.



5 pts

10. Sample a Sinusoid

In the **q10** folder, edit the C++ ROOT application to draw a sine wave with exactly **7 crests** per every **13 units**, over the domain $0 \leq x \leq 26$



1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
5
10
15
20
25