



Survey of Scientific Computing (SciComp 301)

Dave Biersach
Brookhaven National
Laboratory
dbiersach@bnl.gov



```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9
10 namespace SimpleEvents
11 {
12     public partial class Form1 : Form
13     {
14         Person person = new Person();
15
16         public Form1()
17         {
18             InitializeComponent();
19             person.FirstName = "Christian";
20             person.LastName = "Pano";
21         }
22
23         private void button1_Click(object sender, EventArgs e)
24         {
25             person.MainColor = textBox1.Text;
26         }
27     }
28 }
```

Exam 1
Total of 100 points

15 pts

1. Implement Heron's Method

In the **q01** folder, edit the C++ console application to calculate and display the square root of a random integer greater than one million, using **Heron's Method**, to 8 digits of precision to the right of the decimal point.

Specifically you must implement the missing code in the function **heron()** to return the estimate of the square root of the parameter **s**.

You may stop iterating when your current x^2 estimate is within 1×10^{-06} of **s**

$x = \text{initial guess of } \sqrt{S}$

$$S = (x + \Delta_x)^2 = x^2 + (2x)\Delta_x + \Delta_x^2$$

$$\Delta_x = \frac{S - x^2}{2x + \Delta_x}$$

$$\text{Assume } \Delta_x \ll x \therefore \Delta_x \approx \frac{S - x^2}{2x}$$

$$x_{\text{revised}} = x + \Delta_x$$

$$x_{\text{revised}} \approx x + \frac{S - x^2}{2x}$$

$$\approx \left[\frac{2x^2}{(2x)} + \frac{S - x^2}{2x} \right] \approx \frac{1}{2} \left(\frac{S + x^2}{x} \right)$$

$$x_{\text{revised}} \approx \text{Mean} \left(\frac{S}{x}, x \right)$$

20 pts

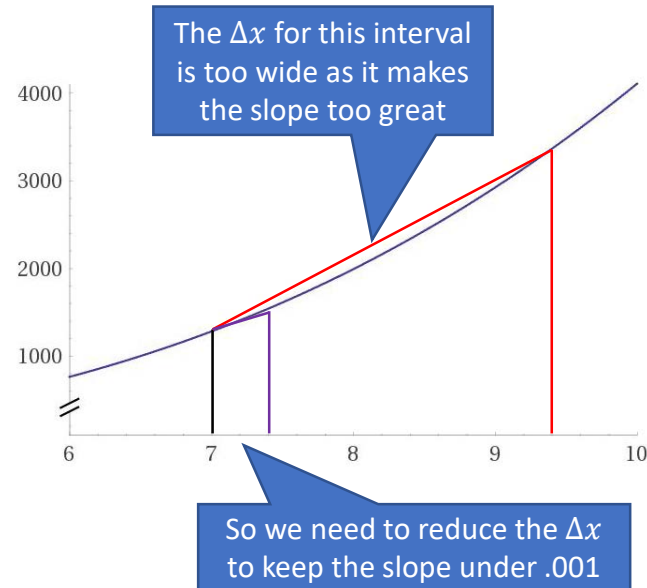
2. Adaptive Quadrature

In the **q02** folder, edit the C++ console application to calculate the following integral using the **midpoint rule**

$$F(x) = \int_0^{10} 5x^3 - 9x^2 - 11$$

Specifically you must complete the function **midPointAdaptive()** that while using the midpoint rule will ensure that the maximum slope of any given interval on the curve will be less than **0.001**

The function **midPointFixed()** is set to use 1 million fixed width intervals. How does the adaptive quadrature compare to this in terms of relative % error?



Adaptive quadrature

Adaptive quadrature is a **numerical integration** method in which the **integral** of a **function** $f(x)$ is **approximated** using static quadrature rules on adaptively refined subintervals of the integration **domain**. Generally, adaptive algorithms are just as efficient and effective as traditional algorithms for "well behaved" integrands, but are also effective for "badly behaved" integrands for which traditional algorithms fail.

10 pts

3. Sum of Multiples

In the **q03** folder, edit the C++ console application to calculate and display the sum of all natural numbers less than 1,900 that are fully (cleanly, evenly) divisible by both 7 and 11

```
// sum-multiples.cpp
#include "stdafx.h"
using namespace std;


int main()
{
    int termsMax = 1900;
    int sum = 0;

    // Implement your code here

    cout.imbue(locale(""));

    cout << "The sum of all natural numbers less than " << termsMax << endl
         << "that are fully divisible by both 7 and 11 is " << sum << endl << endl;

    return 0;
}
```



5 pts

4. Temperature Converter

In the **q04** folder, edit the C++ console application copied from Session 02 Lab 03 to display the same range in temperatures as before, but this time converting from Celsius degrees to Fahrenheit degrees

Be sure to make the human readable output make sense because you are flipping which scale is on which side of the equality symbol

Your code changes must successfully compile and run as expected

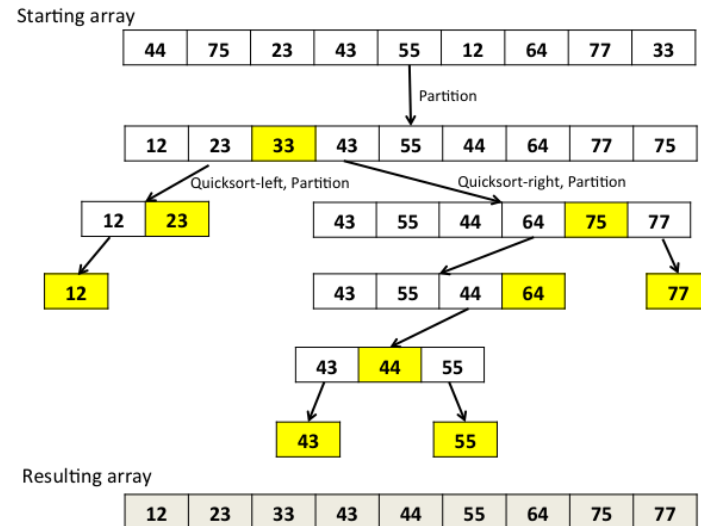
10 pts

5. Quicksort Optimization

Research Hoare's Quicksort algorithm and understand the benefit of partitioning using the *median of three* for the pivot element

In the **q05** folder, edit the C++ console application to implement the **MedianOfThree()** function: *a*, *b*, *c* are the values, *ai*, *bi*, *ci* are the indexes

Your function should return the *index* of the median of the three passed in values



```
template <typename T> size_t MedianOfThree(T &a, T &b, T &c,  
    size_t ai, size_t bi, size_t ci)  
{  
    // Implement your code here  
    return ci;  
}
```

5 pts

6. Lowest Common Multiple

In the **q06** folder, edit the C++ console application to calculate the lowest common multiple (LCM) of two integers *a* & *b*, using only basic arithmetic (*no looping*) and the greatest common divisor (**GCD**) function which is provided

```
// lcm-gcd.cpp
#include "stdafx.h"
using namespace std;


int GCD(uint64_t a, uint64_t b)
{
    return b == 0 ? a : GCD(b, a % b);
}

int LCM(uint64_t a, uint64_t b)
{
    // Implement your code here
}

int main()
{
    uint64_t a = 447618;
    uint64_t b = 2011835;

    cout.imbue(locale(""));
    cout << "LCM of " << a
        << " and " << b << " is "
        << LCM(a, b) << endl;


    return 0;
}
```



5 pts

7. Vector Addition

In the **q07** folder, edit the C++ console application to calculate the addition of two vectors. In particular, complete the function **SumVectors()** by providing values for each element of **vec3** using some form of looping construct



```
// vector-addition.cpp

#include "stdafx.h"

using namespace std;

template <typename T>
void DisplayVector(string name, vector<T> vec)
{
    cout << "Vector " << name << " = {";
    for (size_t i{}; i < vec.size() - 1; i++)
        cout << vec.at(i) << ", ";
    cout << vec.back() << "}" << endl << endl;
}

template <typename T>
vector<T> AddVectors(vector<T> vec1, vector<T> vec2)
{
    vector<T> vec3(vec1.size());
    // Implement your code here
    return vec3;
}

int main()
{
    vector<int> a{ -2, 3, 6, 113, 49, 0, 123 };
    vector<int> b{ 18, 13, 990, 2, -55, -9, 14 };

    DisplayVector("a", a);
    DisplayVector("b", b);
    DisplayVector("c", AddVectors(a, b));


    return 0;
}
```


20 pts

8. Hamming Weight

In the **q08** folder, edit the C++ console application to calculate the Base 2 *Hamming Weight* of a given natural number.

You must write the function
PopCount()



```
// hamming-weight.cpp

#include "stdafx.h"
#include <bitset>

using namespace std;

int PopCount(int n)
{
    int onebits = 0;
    // Insert your code here
    return onebits;
}

int main()
{
    int n = 95601;

    cout << "The Hamming weight of " << n
          << " in base 2 is " << PopCount(n)
          << endl << endl;

    bitset<17> b(n);

    cout << n << " in base 10 is "
          << b.to_string() << " in base 2" << endl;
    cout << "Therefore the popcount is " << b.count()
          << endl << endl;

    return 0;
}
```

10 pts

9. Multimodal Sets

Some lists can be multimodal.

If two or more elements appear an equal number of times within a list, the list is said to be multimodal.

In the **q09** folder, edit the C++ console application to return all of the modes of a list.

Specifically, you must implement the **FindModes()** function by populating the **modes** vector from the passed in **data** vector

```
// multi-modal.cpp
#include "stdafx.h"
using namespace std;

vector<int> FindModes(vector<int> data)
{
    // Prepare return vector
    vector<int> modes;


    // Implement your code here

    return modes;
}

int main()
{
    // Prepare data vector
    vector<int> data {7,1,2,3,7,4,5,6,7,1,2,1,1,9,7,10,2,2};

    cout << "The modes of the data are: " << endl;
    for (auto& n : FindModes(data))
        cout << n << endl;

    return 0;
}
```



10 pts

10. Circle Lattice Points

In the **q10** folder, edit the C++ console application to calculate the integer lattice points in a circle.

Specifically you must write the **LatticePoints()** function, which receives an integer radius, and must count all the lattice points within that radius.

This is known as the Gauss Circle Problem, and Gauss was the first one to prove the number of lattice points is bounded by this expression:

$$N(r) = \pi r^2 + E(r)$$

where

$$|E(r)| \leq 2\sqrt{2\pi r}$$

