

## Mathematical Methods and Numerical Techniques I — Homework Problems

1. *Reading Assignment*

Read Boas, remaining parts of Chapter 1.

2. *Products of Vectors*

(10 P.)

Write a C++ program that accepts the components of two vectors  $\mathbf{a}$  and  $\mathbf{b}$  as keyboard input, and delivers their scalar product  $\mathbf{a} \cdot \mathbf{b}$  as well as their vector product  $\mathbf{a} \times \mathbf{b}$  as output on the screen. (Comment, compile, and test it, too!)

3. *Unit Conversion*

(15 P.)

The conversion factor between inches and centimeters is 1 in = 2.54 cm. Also, one foot is 12 inches. Implement (test, compile) C++ code that takes your height in centimeters as keyboard input, and delivers your height in feet and inches on the screen. (*Example*: My height is 183.5 cm, or 6 ft., 0.25 in.)

4. *Greatest Common Divisor*

(15 P.)

In the lecture, we discussed an algorithm that calculates the greatest common divisor (GCD) of two integer numbers. Construct a C++ program that accepts two numbers as keyboard input, calculates their GCD, and prints it on the screen, Comment, build, and test it.

5. *Prime Numbers*

(25 P.)

On the back of this page you will find a C++ program that produces a list of prime numbers.

(a) What exactly does this program accomplish, and how does it go about it? Analyze the source code, and comment on the algorithm used. Can you think of ways of improving it? (10 P.)

(b) The distribution of prime numbers among the integers is a central subject of *number theory*. A rather mysterious problem in this field is the distribution of *prime number twins*, pairs of prime numbers  $\{p, p + 2\}$  that differ only by 2 (e. g., 3 and 5, 5 and 7, 11 and 13, but also 99989 and 99991). It is not even known whether there are infinitely many of these pairs. — Modify the C++ program on the back of the page as to count and return a list of all prime number *twins* not exceeding some given number. Comment on your changes. (15 P.)

```

/* The FindPrimes Program */

#include<iostream>
using namespace std;

int main()
{
/* Upper Limit */
    long Limit;
    cout << "Upper Limit For Prime Search: ";
    cin >> Limit;

/* Number of Primes */
    long Counter = 0;

/* Current Number Examined */
    long Current = 1;

/* Announce Prime Output */
    cout << "Primes: ";

/* Loop Through Values */
    do
    {
/* Increment Current Number */
        ++Current;

/* Initialize Factor Variable */
        long Factor = 2;

/* Initialize Flag */
        bool IsAPrime = true;

/* Check Whether Current Number Is Prime */
        while ((Factor * Factor <= Current) && (IsAPrime == true))
        {
            if ((Current % Factor) == 0)
            {
/* Number Was Evenly Divisible */
                IsAPrime = false;
            }
            else
            {
/* Check Next Number */
                ++Factor;
            }
        }

/* Count and Output Prime */
        if (IsAPrime == true)
        {
            cout << Current << " ";
            ++Counter;
        }
    }
    while (Current < Limit);

/* Return Number of Primes */
    cout << "\n\nTotal Number of Primes Not Exceeding " << Limit << ": " << Counter << "\n";

/* Terminate Program */
    return 0;
}

```

## 2. Products of Vectors

(This is a program evaluating the scalar and vector products of two vectors. It is a straightforward exercise that should allow you to familiarize yourself with the oddities of writing C++ code.)

```
/* Products of Vectors */

#include<iostream>
using namespace std;

int main()
{
/* Vector Components */
    double ax, ay, az;
    double bx, by, bz;

/* Input Components - Vector a */
    cout << "Vector a, x Component: ";
    cin >> ax;
    cout << "Vector a, y Component: ";
    cin >> ay;
    cout << "Vector a, z Component: ";
    cin >> az;
    cout << "\n";

/* Input Components - Vector b */
    cout << "Vector b, x Component: ";
    cin >> bx;
    cout << "Vector b, y Component: ";
    cin >> by;
    cout << "Vector b, z Component: ";
    cin >> bz;
    cout << "\n";

/* Form Scalar Product */
    double ScalarProduct = ax * bx + ay * by + az * bz;

/* Form Vector Product */
    double VectorProduct_x = ay * bz - az * by;
    double VectorProduct_y = az * bx - ax * bz;
    double VectorProduct_z = ax * by - ay * bx;

/* Output Results */
    cout << "Scalar Product of a, b: " << ScalarProduct << "\n\n";

    cout << "Vector Product of a, b:\n\n";
    cout << "x Component: " << VectorProduct_x << "\n";
    cout << "y Component: " << VectorProduct_y << "\n";
    cout << "z Component: " << VectorProduct_z << "\n";

/* Terminate Program */
    return 0;
}
```

### 3. Unit Conversion

Here's how this program works: First, it reads the height in cm as input from the keyboard, and finds the corresponding value in inches. The `while{...}` loop then subtracts 12 inches from the result and increments the variable representing feet until the remainder is less than a foot. Finally, the result is delivered on the screen.

```
/* Conversion from cm to ft/in */

#include<iostream>
using namespace std;

int main()
{
/* Variables */
    double MetricHeight;
    double Inch;
    int Feet = 0;

/* Conversion Factor */
    double cm_per_inch = 2.54;

/* Input Metric Height */
    cout << "Your Height [cm]: ";
    cin >> MetricHeight;
    cout << "\n";

/* Conversion to Inches */
    Inch = MetricHeight / cm_per_inch;

/* Isolate Feet */
    while (Inch >= 12.0)
    {
/* Subtract 12 Inches, Add 1 Foot */
        Inch -= (double)12;
        ++Feet;
    }

/* Output Results */
    cout << "Your Height Is " << Feet << " ft., " << Inch << " in.";

/* Terminate Program */
    return 0;
}
```

## 4. Greatest Common Divisor

(This is the program that we discussed at length in class. It is here just as an exercise in actually writing C++ code.)

```
/* The GCD Program */

#include<iostream>
using namespace std;

int main()
{
/* Declare Variables For Integers */
    long Number1, Number2;

/* Input Both Numbers */
    cout << "First Number: ";
    cin >> Number1;

    cout << "Second Number: ";
    cin >> Number2;

/* Exchange Numbers If in Wrong Order */
    if (Number1 < Number2)
    {
        long Aux = Number1;
        Number1 = Number2;
        Number2 = Aux;
    }

/* Apply GCD Algorithm Loop */
    do
    {
/* Find Remainder in Division */
        long Aux = (Number1 % Number2);
/* Replace Bigger Number by Smaller Number */
        Number1 = Number2;
/* Replace Smaller Number by Remainder */
        Number2 = Aux;
    }
    while (Number2 != 0);

/* Output Result */
    cout << "Their GCD is " << Number1 << "\n";

/* Terminate Program */
    return 0;
}
```

## 5. Prime Numbers

- (a) This program finds all prime numbers not exceeding the variable `Limit` entered by the user, prints them out on the screen, and also counts their total number (stored in the variable `Counter`). The outer `do{...}while` loop runs checks every whole number (stored in the variable `Current`) between 2 and `Limit` for primality, keeps track of the numbers of primes found, and puts them on the screen. The actual primality test takes place inside the inner (nested) `while{...}` loop: The boolean variable (or *flag*) `IsAPrime`, initially set to `true`, indicates whether a factor has been found that divides `Current` evenly (in which case it is not a prime). For the check itself, `Current` is divided by every integer number (the variable `Factor`) between 2 and the square root of `Current` (this is sufficient because at least one of the factors of a composite number is always smaller or equal to its square root – otherwise, the product any two factors would be bigger than the number itself), until either a factor is found (not a prime), or all possible factors have been eliminated (`Current` is a prime).

Since every number can be represented as a product of primes, it would be sufficient for the primality check to test only the *primes* in the range between 2 and the square root of `Current` (which, by the way, have been previously found by the program) as possible factors of `Current`. For that purpose, however, we need to find a way to store (and access) the prime numbers found by the program.

```
/* The FindPrimes Program */

#include<iostream>
using namespace std;

int main()
{
    /* Upper Limit */
    long Limit;
    cout << "Upper Limit For Prime Search: ";
    cin >> Limit;

    /* Number of Primes */
    long Counter = 0;

    /* Current Number Examined */
    long Current = 1;

    /* Announce Prime Output */
    cout << "Primes: ";

    /* Loop Through Values */
    do
    {
        /* Increment Current Number */
        ++Current;

        /* Initialize Factor Variable */
        long Factor = 2;

        /* Initialize Flag */
        bool IsAPrime = true;
```

```

    /* Check Whether Current Number Is Prime */
    while ((Factor * Factor <= Current) && (IsAPrime == true))
    {
        if ((Current % Factor) == 0)
        {
            /* Number Was Evenly Divisible */
            IsAPrime = false;
        }
        else
        {
            /* Check Next Number */
            ++Factor;
        }
    }

    /* Count and Output Prime */
    if (IsAPrime == true)
    {
        cout << Current << " ";
        ++Counter;
    }
}
while (Current < Limit);

/* Return Number of Primes */
cout << "\n\nTotal Number of Primes Not Exceeding " << Limit
<< ": " << Counter << "\n";

/* Terminate Program */
return 0;
}

```

- (b) In order to find prime twins, the following modifications have been made: First, a new variable `LastPrime` is declared (and initially set to 2, the smallest prime). This variable stores the most recent prime. Later in the program, an additional `if` block checks whether any newly found prime (stored in `Current`) exceeds the previous prime (stored in `LastPrime`) by 2. In this case, the pair of prime twins is printed on the screen, and their (new) counter `TwinCounter` is incremented. In any case, `LastPrime` is then updated to reflect the most recent prime before the next number is examined.

Here is a possible solution, with all major changes and additions marked in **bold**:

```

/* The FindPrimeTwins Program */

#include<iostream>
using namespace std;

int main()
{
    /* Upper Limit */
    long Limit;
    cout << "Upper Limit For Prime Twin Search: ";
    cin >> Limit;

    /* Number of Primes */
    long Counter = 0;

    /* Number of Prime Twins */
    long TwinCounter = 0;
}

```

```

/* Current Number Examined */
    long Current = 1;

/* Initialize LastPrime Variable */
    long LastPrime = 2;

/* Announce Prime Output */
    cout << "Prime Twins:\n";

/* Loop Through Values */
    do
    {
        /* Increment Current Number */
            ++Current;

        /* Initialize Factor Variable */
            long Factor = 2;

        /* Initialize Flag */
            bool IsAPrime = true;

        /* Check Whether Current Number Is Prime */
            while ((Factor * Factor <= Current) && (IsAPrime == true))
            {
                if ((Current % Factor) == 0)
                {
                    /* Number Was Evenly Divisible */
                    IsAPrime = false;
                }
                else
                {
                    /* Check Next Number */
                    ++Factor;
                }
            }

        /* Count Primes, Check and Output Prime Twins */
            if (IsAPrime == true)
            {
                /* Is it a Prime Twin? */
                if ((Current - 2) == LastPrime)
                {
                    cout << LastPrime << " \t" << Current << "\n";
                    ++TwinCounter;
                }

                /* Count Prime */
                ++Counter;

                /* Remember Last Prime */
                LastPrime = Current;
            }
    }
    while (Current < Limit);

/* Return Number of Primes, Prime Twins */
    cout << "\n";
    cout << "Total Number of Primes Not Exceeding " << Limit
        << ": " << Counter << "\n";
    cout << "Total Number of Prime Twins Not Exceeding " << Limit
        << ": " << TwinCounter << "\n";

/* Terminate Program */
    return 0;
}

```