CS007A – Computer Science I – Midterm II Name_____
Write all responses on separate paper. Show all work for credit. Do not use a computer.

1. The **sieve of Eratosthenes** is a very old procedure to compute prime numbers up to a given number MAX. The algorithm starts with a list of numbers from 0 to MAX and proceeds as follows:

   1. *Delete 0 and 1 from the list.*
   2. *Go to the next not-deleted number N and delete all multiples of N from the list (not including N, i.e. 2*N, 3*N, …).*
   3. *Repeat step* ***2.*** *until you reach the end of the list.*

   At the end of this algorithm, all numbers remaining in the list are prime numbers.

   In the following, you should implement this algorithm step-by-step. Implement the list as an array **`A`** of Boolean values, such that **`A[i] = true`** means that number **`i`** is in the list.

   a) Give the declaration of the Boolean array. The size of the array should be 100.

   b) Initialize all array elements with the value **`true`**. Use a for-loop!

   c) Implement steps 1 to 3 of the algorithm described above.

2. Given that `a` is the name of an array whose first element has the address 0x00E0FF58, What is the output of the following code fragments (assuming everything is well defined)?

```
a. int a[2] = {0};
   cout << a << " " << &a[1];
b. short a[3]={8,9};
   cout << a[2] << " " << a+2 << endl;
c. double a[3]={8.,9.,2.64212e-308};
   cout << &a[2] << " " << (a+2)[0] << endl;
```

3. What is the output to the console of the following program?

```
#include <iostream>
using namespace std;

int xfunction(int n, long t)
{
   cout << "int";
   return n;
}

long xfunction(long n)
{
   cout << "long";
   return n;
}

int main()
{
   cout << xfunction(5);
   return 0;
}
```

4. Write a function named "`reduce`" that takes two positive integer arguments, call them "`num`" and "`denom`", treats them as the numerator and denominator of a fraction, and reduces the fraction. That is to say, each of the two arguments will be modified by dividing it by the greatest common divisor of the two integers.

   The function should return the value 0 (to indicate failure to reduce) if either of the two arguments is zero or negative, and should return the value 1 otherwise. Thus, for example, if `m` and `n` have been declared to be integer variables in a program, then

```
m = 25;
n = 15;
if (reduce(m,n))
  cout << m << '/' << n << endl;
else
  cout << "fraction error" << endl;
```

   will produce the following output: `5/3`

   Note that the values of m and n were modified by the function call. Similarly,

```
   m = 63;
   n = 210;
   if (reduce(m,n))
      cout << m << '/' << n << endl;
   else
cout << "fraction error" << endl;
```

   will produce the following output:

```
3/10
```

   Here is another example.

```
m = 25;
n = 0;
if (reduce(m,n))
   cout << m << '/' << n << endl;
else
   cout << "fraction error" << endl;
```

   will produce the following output:

```
fraction error
```

   The function `reduce` is allowed to make calls to other functions that you have written.

5. In this exercise you'll write code to fill in the missing parts of the following polynomial function evaluator:

```
1.  #include <iostream>
2.  #define MAXDEG 100
3.  using namespace std;

4.  //prototype for getCoeff() here
5.  //prototype for evalPoly() here

6.  int main()
7.  {
8.        double coeff[MAXDEG], x;
9.        int n;
10.       cout << "\nWhat is the degree of your polynomial? ";
11.       cin >> n;
12.       getCoeff(coeff,n);
13.       while(1)
14.       {
15.             cout << "\nEnter a value for x: ";
16.             cin >> x;
17.             cout << "\nf(" << x << ") = " << evalPoly(coeff, n, x);
18.       }
19.       return 0;
20. }

21. // define getCoeff() here

22. // define evalPoly() here
```

When the functions are correctly implemented a typical run of the program would look like this, where the polynomial is of degree 2 and the coefficients are specified so that $f(x) = 1 \cdot x^2 + 0 \cdot x + 1$:

```
What is the degree of your polynomial? 2
Enter the coefficient of x^0 :1
Enter the coefficient of x^1 :0
Enter the coefficient of x^2 :1

Enter a value for x: 1

f(1) = 2
Enter a value for x: 2

f(2) = 5
Enter a value for x:
```

a. Give the prototypes for `getCoeff()` and `evalPoly()` specifying the output type and the parameter list for each.
b. Give a function definition for `getCoeff()` that will fill the `coeff[]` array with user-defined coefficients for x^i as $0 \leq i \leq$ degree.
c. Give a function definition for that will return the value of $f(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$ for a given value of x.

CS007A – Computer Science I – Midterm II Solutions

1. The **sieve of Eratosthenes** is a very old procedure to compute prime numbers up to a given number MAX. The algorithm starts with a list of numbers from 0 to MAX and proceeds as follows:

   1. *Delete 0 and 1 from the list.*
   2. *Go to the next not-deleted number N and delete all multiples of N from the list (not including N, i.e. 2*N, 3*N, …).*
   3. *Repeat step* ***2.*** *until you reach the end of the list.*

   At the end of this algorithm, all numbers remaining in the list are prime numbers.

   In the following, you should implement this algorithm step-by-step. Implement the list as an array **`A`** of Boolean values, such that **`A[i] = true`** means that number **`i`** is in the list.

   a) Give the declaration of the Boolean array. The size of the array should be 100.
   SOLN:

```
const int MAX = 100;
bool A[MAX];
```

   b) Initialize all array elements with the value **`true`**. Use a for-loop!
   SOLN:

```
for(int i = 0; i < MAX; ++i)
   A[i] = 1; // or A[i] = true;
```

   c) Implement steps 1 to 3 of the algorithm described above.
   There are a nearly infinite variety of ways to implement this algorithm. Here's a novel approach:

```
// exam 2 q1 eratosthenes - Todor Nikolov's code

#include <iostream>
using namespace std;

int main() {
      const int MAX = 100;  //MAX...the governing number for iterations.
      bool A[MAX]; // algo outcome: value is true iff the number is prime
      for(int i=0; i < MAX; ++i)
            A[i] = 1;  // all numbers are assumed prime unless shown otherwise
      A[0]=0; A[1] =0; // The first two numbers are assumed to not be prime
      for(int i = 2; i < MAX;++i)   { // for all integers from 2 and above..
            if(!(A[i]))  // if we already know it's not prime, skip over it
                  continue;
            for(int j = 2*i; j< MAX; j+=i) //i is prime, all multiples are not
                  A[j]=0; // false, a multiple of i is not prime.
      }
      for(int i = 0; i < MAX; ++i) {  // Print out the pattern of primes:
            if(A[i]) cout << '1';
            else
                cout << (char)32;  // blank
            if(!(i%20))  // weird logic, huh?
                cout << endl;
      }
      return 0;
}
Output:
```

```
 11 1 1   1 1   1 1
  1     1 1     1
1 1   1     1     1
1     1   1 1     1
  1     1       1
```

2. Given that **a** is the name of an array whose first element has the address `0x00E0FF58`, what is the output of the following code fragments (assuming everything is well defined)?
   a. `int a[2] = {0};`
      `cout << a << " " << &a[1];`
      ANS: The first line declares **a** to be an array or two `int`s that is initialized to all zeroes.
      The `cout` command says to output the address of the first element, a space and then address of the second element of the array of the array. Since each integer requires 4 bytes this would produce `00E0FF58 00E0FF5C.`
   b. `short a[3]={8,9};`
      `cout << a[2] << " " << a+2 << endl;`
      ANS: `a` is initialized to have 3 elements, but only the first two have been initialized, so the third element of the array is set by default to 0. Thus a[2] is 0. So the code prints a "0" follwed by a space the code prints a space has not been asigned and by default `cout << a[2]` prints `0`. Then a space is printed and `a+2` is the address of the third element of the array. Since a short is only two bytes, `a+2` is the address of the memory location 4 bytes beyond, so this will print out `0 00E0FF5C`
   c. `double a[3]={8.,9.,2.64212e-308};`
      `cout << &a[2] << " " << (a+2)[0] << endl;`
      ANS: A double uses 8 bytes, so the address of `a[2]` will be 16 bytes greater than that of `a`. Thus the output will be `00E0FF68 2.64212e-308`

3. What is the output to the console of the following program?

```
#include <iostream>
using namespace std;

int xfunction(int n, long t)
{
   cout << "int";
   return n;
}

long xfunction(long n)
{
   cout << "long";
   return n;
}

int main()
{
   cout << xfunction(5);
   return 0;
}
```

SOLN: This program illustrates function overloading. Since the call to `xfunction()` in `main()` takes only one argument, it is assumed that the argument is a long and the second version of `xfunction()` is called, producing the output:

```
long5
```

4. Write a function named "`reduce`" that takes two positive integer arguments, call them "`num`" and "`denom`", treats them as the numerator and denominator of a fraction, and reduces the fraction. That is to say, each of the two arguments will be modified by dividing it by the greatest common divisor of the two integers.

   The function should return the value 0 (to indicate failure to reduce) if either of the two arguments is zero or negative, and should return the value 1 otherwise. Thus, for example, if `m` and `n` have been declared to be integer variables in a program, then

```
m = 25;
n = 15;
if (reduce(m,n))
  cout << m << '/' << n << endl;
else
  cout << "fraction error" << endl;
```

will produce the following output: `5/3`

Note that the values of m and n were modified by the function call. Similarly,

```
m = 63;
n = 210;
if (reduce(m,n))
   cout << m << '/' << n << endl;
else
cout << "fraction error" << endl;
```

will produce the following output:

```
3/10
```

Here is another example.

```
m = 25;
n = 0;
if (reduce(m,n))
   cout << m << '/' << n << endl;
else
   cout << "fraction error" << endl;
```

will produce the following output:

```
fraction error
```

The function `reduce` is allowed to make calls to other functions that you have written.

ANS:

```
bool reduce(int &m, int &n) {
   int min;
   if(n==0 || m==0)  // only reduce with non zero values
         return 0;
   if(n>m)
         min = m;
   else min = n;
   for (int i = min; i > 1; i--) {
         if(m%i==0 && n%i==0) {  //if m and n are divisible by i then reduce
               m/=i;
               n/=i;
         }
   }
   return 1;
}
```

5. In this exercise you'll write code to fill in the missing parts of the following polynomial function evaluator:

```
1.  #include <iostream>
2.  #define MAXDEG 100
3.  using namespace std;

4.  //prototype for getCoeff() here
5.  //prototype for evalPoly() here

6.  int main()
7.  {
8.     double coeff[MAXDEG], x;
9.     int n;
10.    cout << "\nWhat is the degree of your polynomial? ";
11.    cin >> n;
12.    getCoeff(coeff,n);
13.    while(1)
14.    {
15.            cout << "\nEnter a value for x: ";
16.            cin >> x;
17.            cout << "\nf(" << x << ") = " << evalPoly(coeff, n, x);
18.    }
19.    return 0;
20. }

21. // define getCoeff() here

22. // define evalPoly() here
```

When the functions are correctly implemented a typical run of the program would look like this, where the polynomial is of degree 2 and the coefficients are specified so that $f(x) = 1 \cdot x^2 + 0 \cdot x + 1$:

```
What is the degree of your polynomial? 2
Enter the coefficient of x^0 :1
Enter the coefficient of x^1 :0
Enter the coefficient of x^2 :1

Enter a value for x: 1

f(1) = 2
Enter a value for x: 2

f(2) = 5
Enter a value for x:
```

a. Give the prototypes for `getCoeff()` and `evalPoly()` specifying the output type and the parameter list for each.
SOLN:
`void getCoeff(double [],int);`
`double evalPoly(double [],int,double);`

b. Give a function definition for `getCoeff()` that will fill the `coeff[]` array with user-defined coefficients for x^i as $0 \leq i \leq$ degree.

```
void getCoeff(double c[], int d)
{
   for(int i = 0; i <= d; ++i)
   {
         cout << "\nEnter the coefficient of x^" << i << " :";
         cin >> c[i];
   }
}
```

c. Give a function definition for that will return the value of $f(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$ for a given value of x.

```
double evalPoly(double c[],int d, double x)
{
   double pofx = 0, powx = 1;  // pofx is the value of the polynomial at x
                               // powx is x^i
   for(int i = 0; i <= d; ++i)
   {
         pofx += c[i]*powx;
         powx *= x;
   }
   return pofx;
}
```