AERSP 424: Advanced Computer Programming

Submission Instructions: Homework 2
Due: 1/22/19

Submission Instructions:

- Submit the file with the .cpp extension containing your C++ source code.
- Notes on submissions:
  - Submit single cpp file for questions 1,2,3
  - Submit one pdf for questions 4 and 5. NO HANDWRITTEN SUBMISSIONS.
  - Complete submissions should include a cpp file and a pdf.
- Teams of up to 3 are allowed. Put the names of all team members on a single submission.

```
int transform( int array[], int size);
int main()
{
     int anArray[] = {0,1,2,3,4};
     int result = 0;
     result = transform(anArray, 5);
     for(int i=0; i<size; i++ )
        cout << array[i] << endl;
     cout << result << endl;
     return 0;
}

int transform( int array[], int size )
{
     int sum=0;
     for(int i=0; i<size; i++ ) {
        sum += array[i]
        array[i]=array[i]*array[i];
     }
     return sum;
}
```

1. Copy the code above:
   a. Compile the program. Show the result.

```
0
1
4
9
16
10
```

b. Create another function which transforms **double** array types and returns a **double** value.

```
double transform( double array[], int size);
int main()
{
      SAME AS BEFORE…
}

double transform( double array[], int size )
{
      double sum=0;
      for(int i=0; i<size; i++ ) {
         sum += array[i];
         array[i]=array[i]*array[i];
      }
      return sum;
}
```

c. Create a function called **transformByValue** which transforms an **double** valued array which is pass by value and returns the result as an **double**.

```
double transformByValue(double x, double y) {
      if(y==0)
            return 1;
      else {
            int i=1;
            while (i<y) {
                  x*=x;
                  i++;
            }
            return x;
      }
}
```

d. Create a function called **transformByReference** which transforms an **double** valued parameter which is passed by reference and returns a **double**.

```
double transformByReference(double& x, double y) {
      if(y==0)
            return 1;
      else {
            int i=1;
            while (i<y) {
                  x*=x;
                  i++;
            }
```

```
        return x;
    }
}
```

e. In main create a **double** variable called **dVal** set to 5.1. Print the value of **dVal** to console. Call the function **transformByValue** with **dVal** as the parameter. Now print the value of **dVal** to console again. Has the value changed?

```
double dVal=5.1;
double result;
cout << "Before: "<<dVal << endl;
result = transformByValue(dVal,2);
cout << "After: "<<dVal << endl;
```

No, the value hasn't changed.

f. In main create a **double** variable called **dRef** set to 3.2. Print the value of **dRef** to console. Call the function **transformByReference** with **dRef** as the parameter. Now print the value of **dRef** to console again. Has the value changed?

```
double dRef=3.1;
cout << "Before: "<<dRef << endl;
result = transformByReference(dRef,2);
cout << "After: "<<dRef << endl;
```

Yes, the value has changed.

2. Create a function that implements Ackermann's function:

$$A(m,n) = \begin{cases} n+1 & if\ m = 0 \\ A(m-1,1) & if\ m > 0\ and\ n = 0 \\ A\big(m-1, A(m,n-1)\big) & if\ m > 0\ and\ n > 0 \end{cases}$$

a. Create a recursive function called **Ackermann** takes two **unsigned long** parameters and returns a single **unsigned long** value. The function calculates the value of Ackermann's function for a given input.

```
unsigned long Ackermann(unsigned long m, unsigned long n) {
    if(m==0) {
        return (n+1);
    }
    else if(m>0 && n==0) {
        return Ackermann(m-1,1);
    }
    else if(m>0 && n>0){
```

```
            return Ackermann(m-1, Ackermann(m,n-1));
        }
        else {
            cout << "Invalid Parameter"<<endl;
            return -1;
        }
    }
```

b. Feed random numbers into the function for both n and m between the values of 0 and 3. List the numbers and their values.

```
will vary:
Ackermann(3,3) = 61
Ackermann(3,2) = 29
Ackermann(3,1) = 13
Ackermann(3,0) = 5
Ackermann(2,3) = 9
Ackermann(2,2) = 7
Ackermann(2,1) = 5
Ackermann(2,0) = 3
Ackermann(1,3) = 5
Ackermann(1,2) = 4
Ackermann(1,1) = 3
Ackermann(1,0) = 2
```

3. Create a function called **`BabylonianMethod`** which calculates the square root ($\sqrt{S}$) of a number using the Babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method) by taking two **`double`** as a parameters one representing *S* and the second representing an initial guess and returning a **`double`** of the estimate of $\sqrt{S}$. Using the same function name but different parameter types and return types to add functions with:
    a. **`double`** parameter, **`double`** return
    b. **`float`** parameter, **`float`** return
    c. **`int`** parameter, **`int`** return
    d. **`unsigned long`** parameter, **`unsigned long`** return
    e. In main call each version of the function.

```
double BabylonianMethod(double S, double initialGuess) {

    double guess=initialGuess;
    for(int i=0;i<6;i++) {
        guess = 0.5*(guess + (S/guess));
    }
```

```
return guess;
}
```

4. What is the output of the following? **Add comments briefly explaining each line**

a.
```
int main()
{
   int x=20, y =35;
   x = y++ + x++;
   y = ++y + ++x;
   cout << x << " "<<y << endl;
   return 0;
}
```

```
56 93
```

b.
```
int main()
{
   int check=2;
   switch(check) {
      case 1:
         cout << "George ";
      case 2:
         cout << "Edward ";
      case 3:
         cout << "Richard ";
      default:
         cout << "Henry ";
   }
   cout << endl;
   return 0;
}
```

```
Edward Richard Henry
```

c.
```
int main()
{
   int m=-10, n =20;
   n = (m < 0) ? 0 : 1;
   printf("%d %d", m,n);
   return 0;
}
```

```
-10 0
```

5. Identify and correct the errors in each of the following:

a.
```
cout >> output;
```

```
   cout << output;
```

b.
```
while (x = 4 ) {
   ++c;
```

```
   while (x == 4 ) {
        ++c;
   }
```

c.
```
if (y>3)
   sum =+ 5;
```

```
   if(y>3)
     sum+= 5;
```

d.
```
if ( hw == 1)
      cout << "Good" << endl;
   else;
      cout << "Bad" << endl;
   endif;
```

```
   if ( hw == 1)
      cout << "Good" << endl;
   else
      cout << "Bad" << endl;
```