**EECS2040 Data Structure Hw #1 (Chapter 1, 2 of textbook)**

**due date 4/8/2021**

**by 107061123, 孫元駿**

**Part 1 (40% of Hw1)**

1. (20%) Using the ADT1.1 *NaturalNumber* in the textbook pp.10, add the following operations to the *NaturalNumber* ADT: Predecessor, *IsGreater*, *Multiply*, *Divide*.

Predecessor(x): NaturalNumber ::= if(x == 0) return 0

else return x-1

IsGreater(x,y):Boolean ::= if(x > y) return TRUE

else return FALSE

Multiply(x,y):NaturalNumber ::= if((x \* y) < MAXINT) return x \* y

else return MAXINT

Divide(x,y):NaturalNumber ::= if(y == 0) return ERROR

if(x<y) return 0

else return 1 + Divide(x-y, y)

1. (20%) Determine the frequency counts for all statements (by step table) in the following two program segments:

code (a): code (b)

1. for(i=1;i<=n;i++) 1 i=1;
2. for(j=1;j<=I;j++) 2 while(i<=n)
3. for(k=1;k<=j;k++) 3 {
4. x++; 4 x++;

5 i++;

6 }

code (a):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | s/e | freq | subtotal |
| 1 | for(i=1;i<=n;i++) | 1 | +1 | +1 |
| 2 | for(j=1;j<=i;j++) | 1 |  |  |
| 3 | for(k=1;k<=j;k++) | 1 |  |  |
| 4 | x++; | 1 |  |  |
| Total | |  | | |

code (b)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | s/e | freq | subtotal |
| 1 | i=1; | 1 | 1 | 1 |
| 2 | while(i<=n) | 1 | n+1 | n+1 |
| 3 | { | 0 | n | 0 |
| 4 | x++; | 1 | n | n |
| 5 | i++; | 1 | n | n |
| 6 | } | 0 | n | 0 |
| total | | 3n+2 | | |

1. (20%) For the function Multiply() shown below,
2. Introduce statements to increment count at all appropriate points and compute the count

void Multiply(int \*\*a,int \*\*b, int \*\*c, int m, int n, int p){

for(int i=0;i<m;i++){

count++;

for(int j=0; j<p; j++){

count++;

c[i][j] = 0;

count++;

for(int k=0;k<n;k++){

count++;

c[i][j] += a[i][k] \* b[k][j];

count++;

} count++;

} count++;

} count++;

}

1. Simplify the resulting program by eliminating statement and compute the count

void Multiply(int \*\*a,int \*\*b, int \*\*c, int m, int n, int p){

for(int i=0;i<m;i++){

count+=2;

for(int j=0; j<p; j++) count+=3;

for(int k=0;k<n;k++) count+=2;

}

count++;

}

1. Obtain the step count for the function using the frequency method.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | s/e | freq | subtotal |
| 1 | for(int i=0;i<m;i++) | 1 |  |  |
| 2 | for(int j=0; j<p; j++) | 1 |  |  |
| 3 | { | 0 |  |  |
| 4 | c[i][j] = 0; | 1 |  |  |
| 5 | for(int k=0;k<n;k++) | 1 |  |  |
| 6 | c[i][j] += a[i][k] \* b[k][j]; | 1 |  |  |
| 7 | } | 0 |  |  |
| total | |  | | |

1. (20%) A complex-valued matrix X is represented by a pair of matrices (A, B) where A and B contains real values. Write a program that computes the product of two complex-valued matrices (A, B) and (C, D), where (A, B) \* (C, D) = (A+iB)\*(C+iD) = (AC-BD)+i(AD + BC). Determine the number of additions and multiplications if the matrices are all nxn.

(a)

// (Real, Imag)

// (A, B) \* (C, D) = (R, I)

// A, B, C, D are all two dimension int array to represent Matrix

// n means the n\*n Matrix

void(int\*\* A, int\*\* B, int\*\* C, int\*\* D, int\*\* R, int\*\* I, int n)

{

for(int i = 0; i < n; i++)

{

for(int j = 0; j < n; j++)

{

for(int k = 0; k < n; k++)

{

R[i][j] += A[i][k] \* C[k][j] - B[i][k] \* D[k][j];

I[i][j] += A[i][k] \* D[k][j] + B[i][k] \* C[k][j];

}

}

}

}

(b)

For the result of every element need times of addition, and times of multiply. We got Matrix, so that we have element. And every complex multiply may need multiply and addition.

So that we need multiplication times and addition times.

1. (20%) The Tower of Hanoi is a classical problem which can be solved by recurrence. There are three pegs and N disks of different sizes. Originally, all the disks are on the left peg, stacked in decreasing size from bottom to top. Our goal is to transfer all the disks to the right peg, and the rules are that we can only move one disk at a time, and no disk can be moved onto a smaller one. We can easily solve this problem with the following recursive method: If N = 1, move this disk directly to the right peg and we are done. Otherwise (N >1), first transfer the top N − 1 disks to the middle peg applying the method recursively, then move the largest disk to the right peg, and finally transfer the N −1 disks on the middle peg to the right peg applying the method recursively. Let T(N) be the total number of moves needed to transfer N disks.

(a) Prove that T(N) = 2T(N −1) + 1 with T(1) = 1.

Assume there are three pegs, call it A, B, C, if all N disks are at A peg, then we cost one step to move all disk from A to C, if , s.t .

In general, if we need to move disks from A to C, we need steps.

First, we move disk from A peg to B peg and cost steps, then we spend one step to move the largest disk from A to C, last we spend step again to move disks from B to C.

s.t we know with .

(b) Unfold this recurrence relation to obtain a closed-form expression for T(N). (T(N) is expressed in terms of function of N.)

**Part 2 Coding (60% of Hw1)**

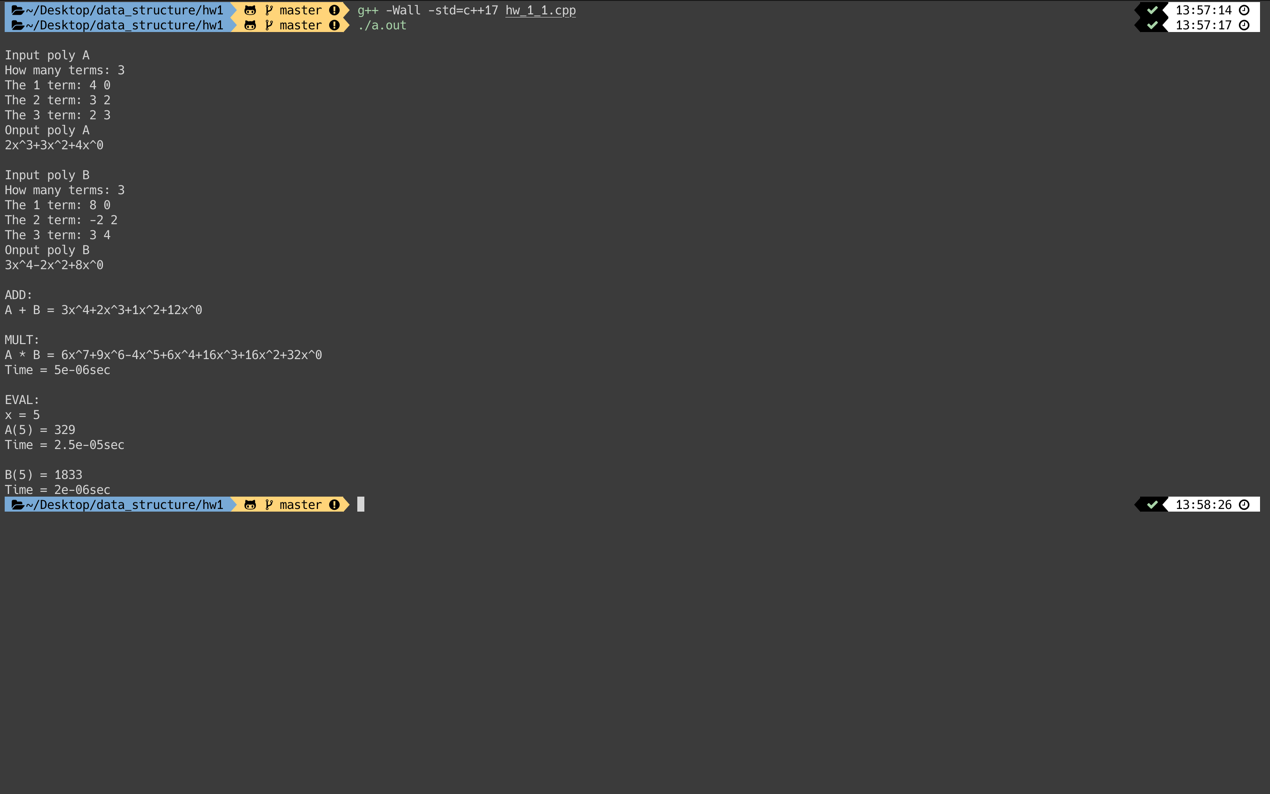
You should submit:

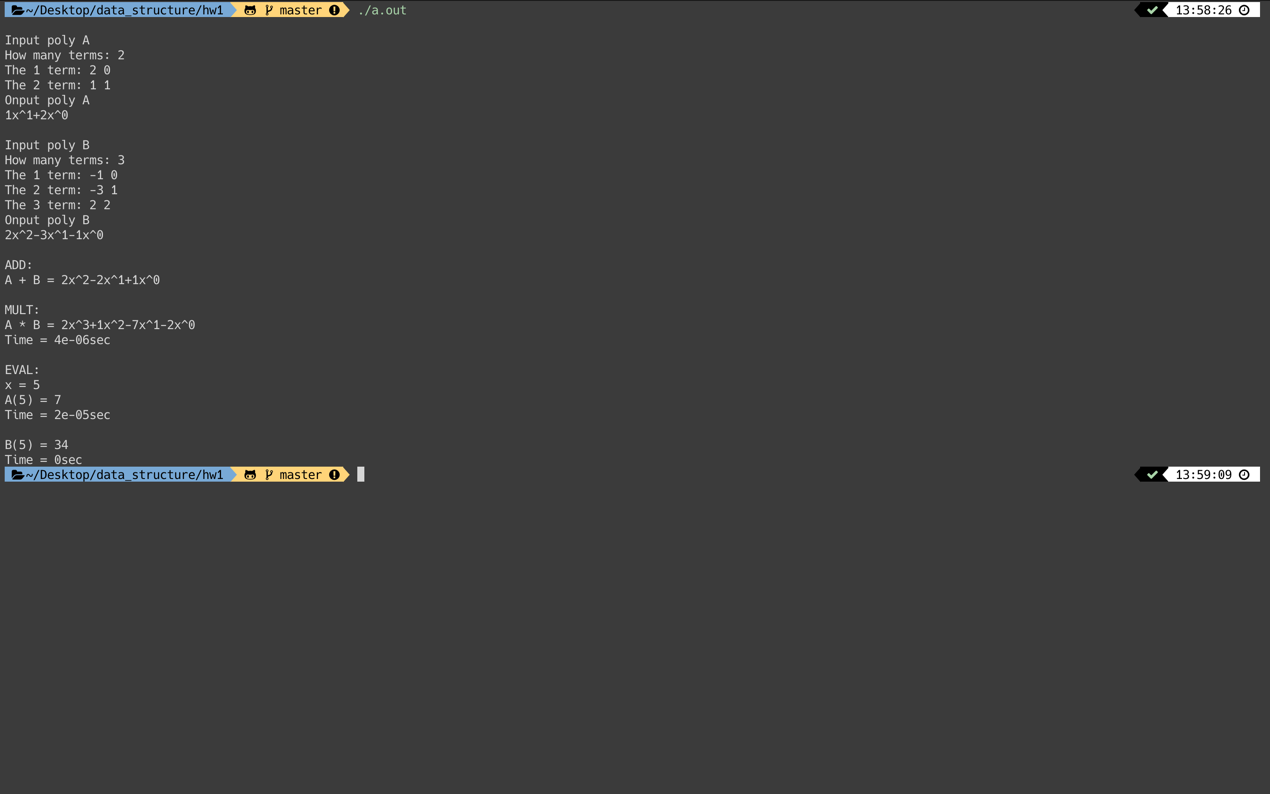
(a) All your source codes (C++ file).

(b) Show the execution trace of your program.

1. (30%) Write a C++ program to implement the **ADT2.3 Polynomial** (pp.88) using Representation 3 (dynamic array of (coef, exp) tuples). Implement the Mult(Polynomial p) and Eval(float x). Estimate the computing time for Mult and Eval function. Add two more functions to input and output polynomials via **overloading** the **>>** and **<< operators**.

You should try out at least two runs of your program (execution trace) to **demonstrate** the Add, Mult, Eval and input, output functions.

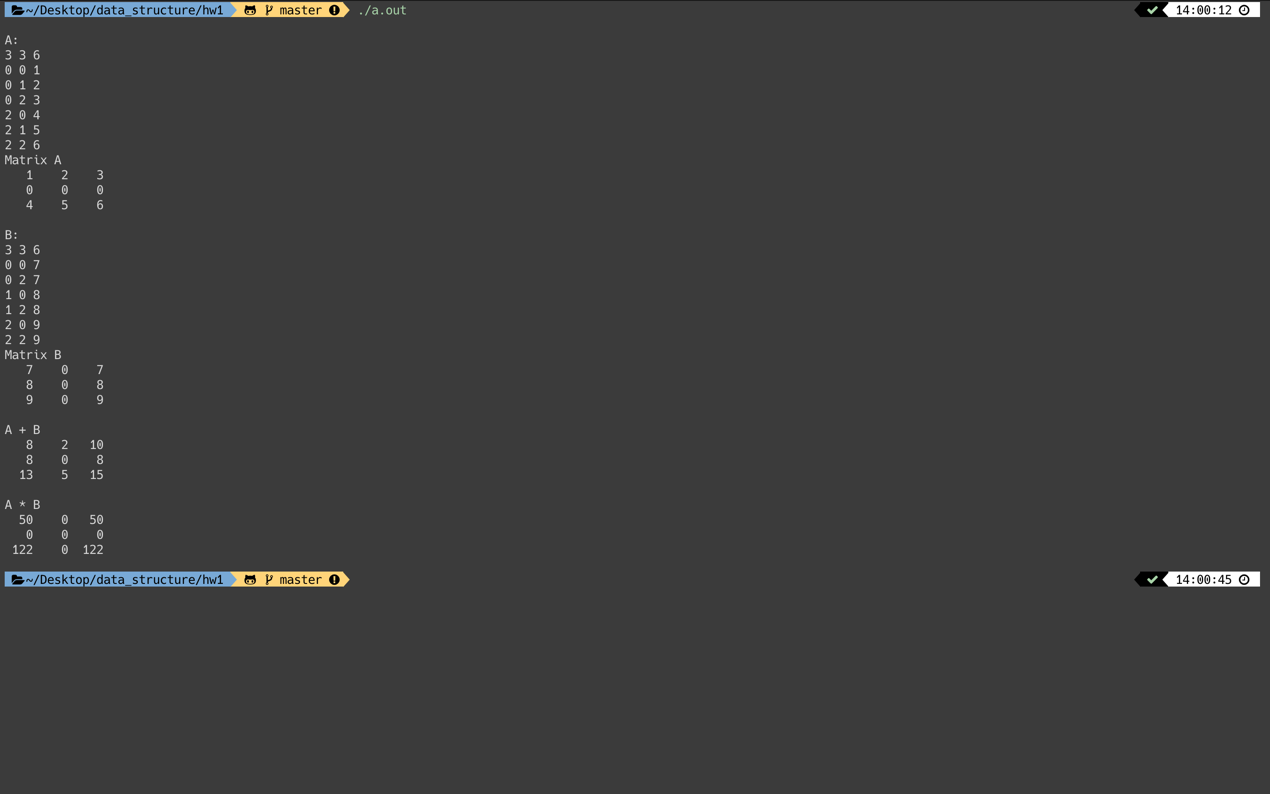




1. (35%) Write a C++ program to implement the **ADT2.4 SparseMatrix** in textbook (pp.97) (with Transpose implemented by FastTranspose). You should build you program based on the example codes in the book and implement the Add function and functions to input, output a sparse matrix by **overloading** the **>>** and **<<** **operators**.

You should try out at least two runs of your program to demonstrate the Add, Mult, Eval and input, output functions.





1. (35%) Write a C++ program to implement the **ADT2.5 String** (pp.114) (with Find function implemented by FastFind). In addition, write two more functions: String::Delete(int start, int length); //remove length characters beginning at start

String::CharDelete(char c); //returns the string with all occurrence of c removed.

You should try out at least two runs of your program to demonstrate all those functions.

