**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)**

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

return 1 + Divide(x-y, y)

2.

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 | | |

3.

(a)

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++;

}

(b)

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++;

}

(c)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 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 | |  | | |

4.

(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.

5.

(a)

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)

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

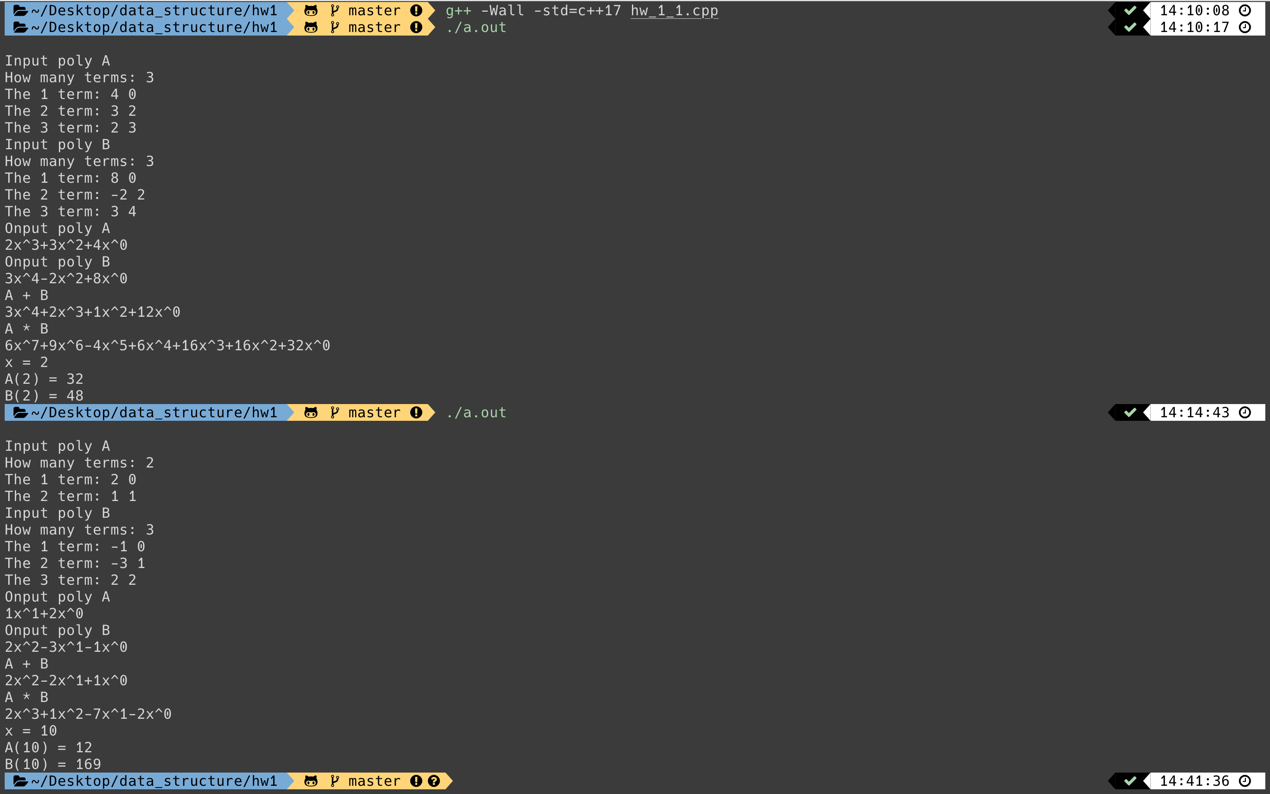
You should submit:

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

(b) Show the execution trace of your program.

1.

The execution trace of my program



2.

The execution trace of my program





3.

The execution trace of my program

