

EE441- Programming Assignment 1

Due Date: 7.11.2023, 23:55

Part 1 – Classes & Arrays

1. This class holds number 4 ,and an operator '+', and a member function to apply the given operation to a number. The input of the function is 5. Therefore, as a output we expect to see 9.

```
int main()
{
    int output, input = 5;
    Operation op(4, '+');

    output = op.apply_operation(input);
    cout << output;
```

Figure 1: Main of the part 1 question 1

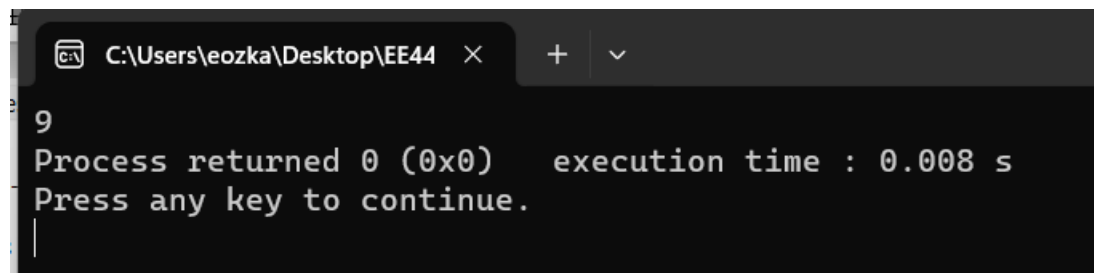


Figure 2: Output of the part 1 question 1

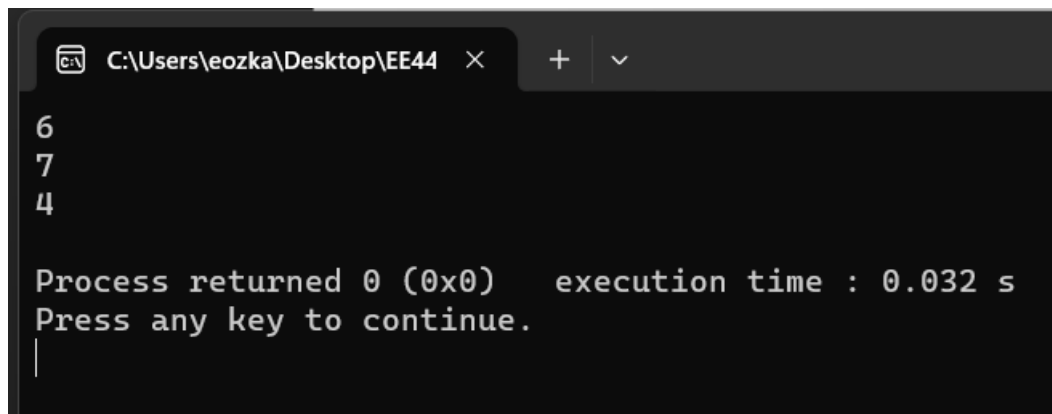
2. In this question, I implemend a class for operation queue. I put operation into the queue by using push_back() function, then using pop_front(), I applied the operation with input 2. I expected the output to be 6, 7 , 4 since it is a queue and first in operation will be applied firstly.

```
int main()
{
    Operation op1(3, '*'), op2(5, '+'), op3(2, '^');
    OperationQueue Q1;

    Q1.Push_Back(op1);
    Q1.Push_Back(op2);
    Q1.Push_Back(op3);

    while(!Q1.QEmpty())
    {
        Operation op;
        op = Q1.Pop_Front();
        cout << op.apply_operation(2) << endl;
    }
```

Figure 3 Main of the part 1 question 2



```
C:\Users\eoarka\Desktop\EE44 × + v
6
7
4

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
|
```

Figure 4

Output of the part 1 question 2

3.

```
double apply_operations(OperationQueue &myQ, double input)
{
    Operation temp;
    while (!myQ.QEmpty())
    {
        // pop the front element of queueu
        temp = myQ.Pop_Front();
        // apply operation one by one
        input = temp.apply_operation(input);
    }
    return input;
}
```

Figure 5 Code of the part 1 question 3

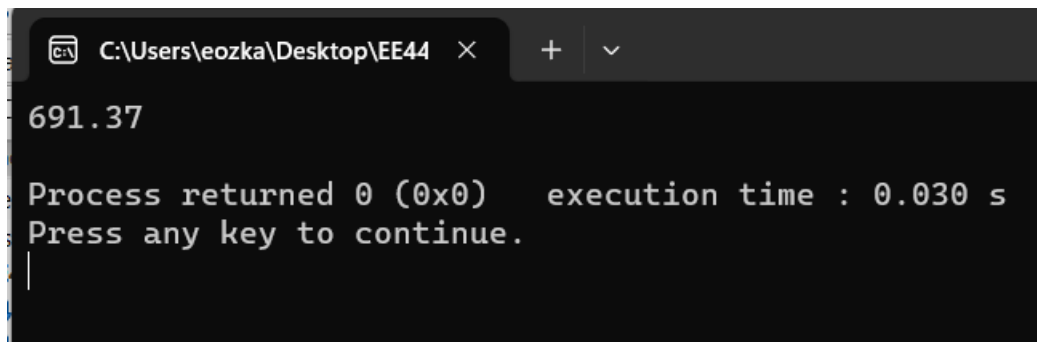
4. In order to calculate $3x^2 + 9x + 5$ this function, I manipulated the function as $3*((x + 1.5)^2 - 7/12)$ and put the operation objects by starting the inside of manipulated function. In op3 object double casting performed due to avoid loss the information.

```
int main()
{
    double x = 13.7;
    Operation op1(1.5, '+'), op2(2, '^'), op3(7.0/12, '-'), op4(3, '*');
    OperationQueue Q1;
    Q1.Push_Back(op1);
    Q1.Push_Back(op2);
    Q1.Push_Back(op3);
    Q1.Push_Back(op4);

    cout<< apply_operations(Q1, x)<<endl;

    return 0;
}
```

Figure 6 Main of the part 1 question 4



```
C:\Users\eoarka\Desktop\EE44 x + v
691.37
Process returned 0 (0x0) execution time : 0.030 s
Press any key to continue.
```

Figure 7 Output of the part 1 question 4

Part 2 – Recursion & Algorithmic Complexity

1.

In this question to model three rod I implemented a class. In this class the top information of nonzero values are holded and they are change when the push and pop functions are called. My class for rods is below.

```
const int MaxStackSize = 20;
template <class T>
class Rod2D
{
private:
    T rodlist[3][MaxStackSize]; // Rodlist for indicate 3 rod in the game
    int top[3];                 // Top array to point the nonzero top of the rod
public:
    //Constructor
    Rod2D(void);
    // modification operations
    void Push(const T& item, int rodnum);
    // Pop nonzero top element of selected rod
    T Pop(int rodnum);
    //just copy top item of selected rod without modifying stack contents
    T Peek(int rodnum) const;
    // return the nonzero top of the selected rod
    int returnTop(int rodnum) const;
    // fill with zero to indicate empty stack positions
    void PushZero(int rodnum, int idx);
    // return a disc of the selected rod at desired position
    T ShowElement(int rodnum, int idx) const;
};
```

Figure 9 Class to model rods

In hanoi constructor, the rods are filled with zeros then the discs are put in the first rod. The figure below shows the hanoi game which is constructed with 5 discs.

For the move function, the indices are checked whether they are bigger than and equal to or smaller than 3. To check if the move is legal or illegal, I just check the condition that diameter of top disc of source rod smaller than the top disc in the destination rod. In the algorithm move disc from rod to same rod is legal since it does not change anything.



Figure 10 Hanoi game constructed with 5 discs

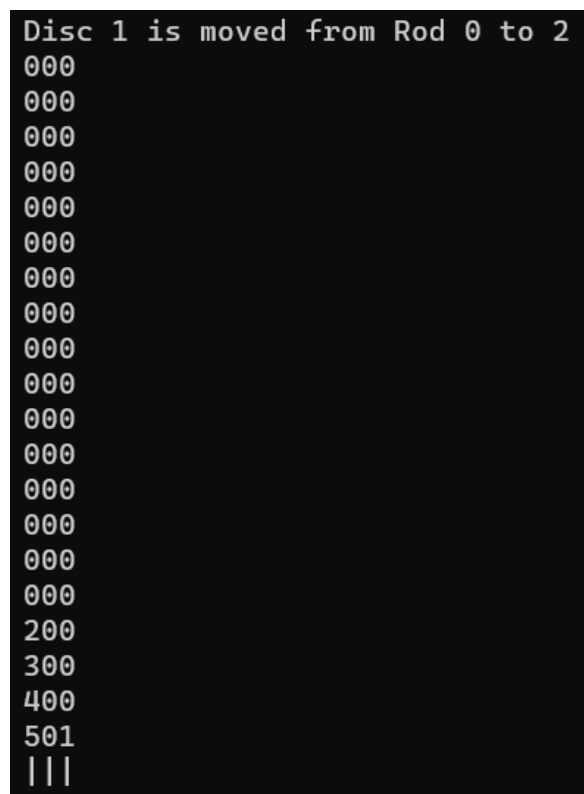


Figure 11 After calling move(0,2) function

To solve hanoi with n discs recursively, the algorithm is constructed as follows:

- Solve hanoi for first n-1 disc and move them from source rod to middle rod
- Move nth disc from source to destination rod
- Solve hanoi for first n-1 disc and move them from middle rod to destination rod

First the solve_hanoi function takes hanoi object and gets the number of disc in the first rod dynamically. The function calls solve_hanoi_recursively, which is explained above.

The figure 12 shows the functions for solve hanoi and complexity calculations for recursive hanoi.

```

void solve_hanoi_recursive(Hanoi&Game, int DiscNum,int source,int middle,int destination)
{
    // Termination
    if (DiscNum == 1) // t.b : Stopping condition
        // Move the first disc (Biggest) from source the destination rod
        Game.move(source,destination); // T2 : comp. of move function O(1)
    else
    {
        // Solve hanoi for the top (n-1) discs from source to middle rod
        solve_hanoi_recursive(Game,DiscNum-1,source,destination,middle); // T(n-1)
        // Move the nth disc from source to destination
        Game.move(source,destination); // T2
        // Solve hanoi for the top (n-1) discs from middle to destination rod
        solve_hanoi_recursive(Game,DiscNum-1,middle,source,destination); // T(n-1)
    }
}

void solve_hanoi(Hanoi& Game)
{
    // Give names to rods
    int source = 0,middle = 1,destination = 2;
    // Number of discs in rods
    int *numOfDiscs, DiscNum;
    int arr[3]; // Array to hold number of the discs in the three rod

    // Get the number of discs at each rod
    numOfDiscs = Game.returnRodTops(arr);
    DiscNum = *(numOfDiscs);
    // Solve hanoi recursively
    solve_hanoi_recursive(Game,DiscNum,source,middle,destination);
}

```

Base condition: $T(1) = 1$

$$T(n) = 2T(n-1) + 1$$

$$T(n-1) = 2T(n-2) + 1$$

$$T(n-1) = 2(2T(n-3) + 1) + 1$$

$$= 2^2 T(n-3) + 2 + 1$$

$$T(n) = 2^3 T(n-3) + 2^2 + 2 + 2^0$$

Generalization

$$T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \dots + 1$$

$n - k = 1$
put $k = n - 1$

$$T(n) = 2^{n-1} T(1) + 2^{n-2} + 2^{n-3} + \dots + 1$$

$$= 2^{n-1} + 2^{n-2} + \dots + 2^2 + 2^1 + 1 = 2^n - 1$$

$T(n) = O(2^n - 1) = O(2^n)$

Figure 13 Complexity calculation of recursive solve hanoi function

```

Disc 2 is moved from Rod 0 to 2
Disc 1 is moved from Rod 1 to 2
Disc 3 is moved from Rod 0 to 1
Disc 1 is moved from Rod 2 to 0
Disc 2 is moved from Rod 2 to 1
Disc 1 is moved from Rod 0 to 1
Disc 4 is moved from Rod 0 to 2
Disc 1 is moved from Rod 1 to 2
Disc 2 is moved from Rod 1 to 0
Disc 1 is moved from Rod 2 to 0
Disc 3 is moved from Rod 1 to 2
Disc 1 is moved from Rod 0 to 1
Disc 2 is moved from Rod 0 to 2
Disc 1 is moved from Rod 1 to 2
000
000
000
000
000
000
000
000
000
000
000
000
000
000
000
000
000
000
000
000
001
002
003
004
|||

```

Figure 13 Output of solve hanoi function

2.

The algorithm for print_backwards as follows:

- i. Run the function until see the null-terminated string. Our termination condition of recursive algorithm is (character [n] == '\0').
- ii. If char is not equal to null terminated string, call the function for string has last n-1 character.
- iii. After calling function, print the current character.

The figure 14 complexity of this algorithm.

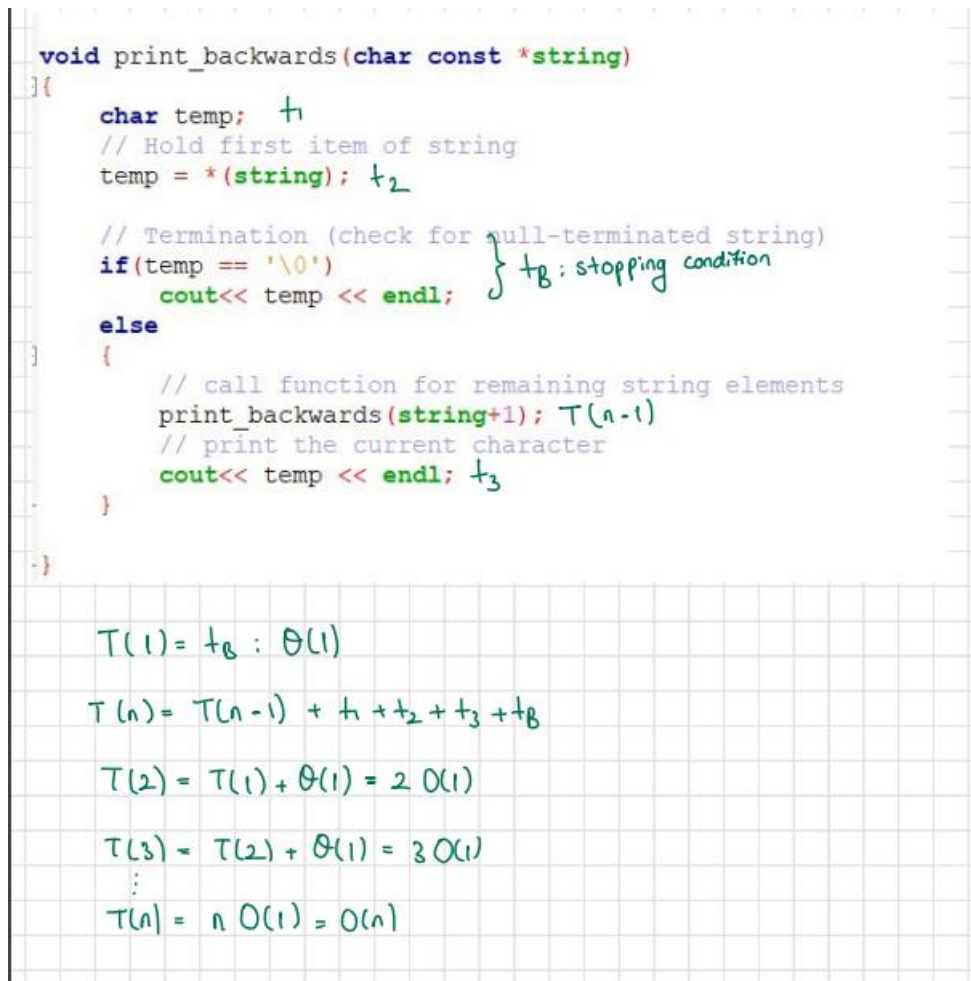


Figure 14 Complexity calculation

As a input “EE441 Data Structures” is given the function.

serutcurtS ataD 144EE6

Figure 15 Output of print_backwards

3. The complexity of GCD algorithm is $O(\log(a \cdot b)) = O(\log a + b) = O(\log n)$

```

int gcd(int n, int m)
{
    int temp;
    // Take the absolute value of the numbers if they are negative
    if(n<0)
        n = abs(n);
    if(m<0)
        m = abs(m);
    // Return zero if any number is zero
    if((n == 0) || (m == 0))
        return 0;

    while (m != 0) {
        temp = m;
        m = n % m;
        n = temp;
    }
    return n;
}

```

Figure 15 GCD Algorithm

4. The complexity of LCM algorithm is $O(\log n)$ since I use the gcd algorithm and complexities of other lines is constant.

```
int lcm(int n, int m)
{
    // Take the absolute value of the numbers if they are negative
    if(n<0)
        n = abs(n);
    if(m<0)
        m = abs(m);
    // Return zero if any number is zero
    if((n == 0) || (m == 0))
        return 0;
    // If they are equal return any one of them
    if (n == m)
        return m;
    return (m*n)/gcd(n,m);
}
```

Figure 15 GCD Algorithm

```
// greatest common divisor of two integers
gcd1 = gcd(12,18);
cout <<" " << endl;
cout <<"gcd: " <<gcd1 << endl;
cout <<" " << endl;
// least common multiple of two integers
lcm1 = lcm(12,18);
cout <<"lcm: " << lcm1;
```

Figure 16 Main function implementation of gcd and lcm functions

```
gcd: 6
lcm: 36
```

Figure 17 Output of functions

5.

n	time (ns)
1	3e-07
2	3.3e-07
3	4.1e-07
4	7.5e-07
5	1.2e-06
6	2.19e-06
7	3.94e-06
8	7.5e-06
9	1.807e-05
10	3.107e-05
11	5.698e-05
12	0.00011738
13	0.00022831
14	0.00045664
15	0.00093829
16	0.00199535
17	0.00373856
18	0.00769503
19	0.0154275
20	0.0305162

Figure 18 Benchmark result of hanoi game

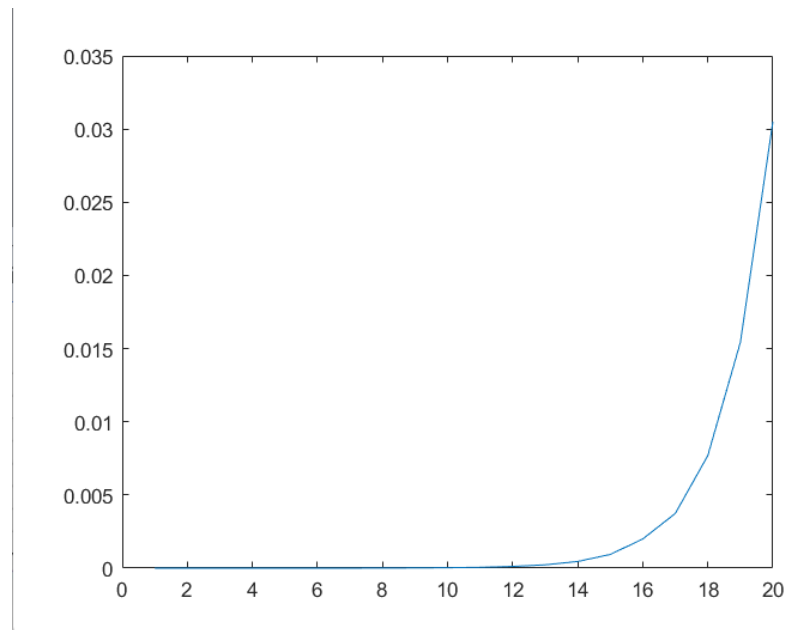


Figure 19 Plot of comlexity of hanoi game

As we can see from Figure 19, complexity of hanoi is $O(2^n)$.

n	time (ns)
1	0.00011568
2	0.00015525
3	0.00026823
4	0.00030624
5	0.0003759
6	0.00044161
7	0.00041747
8	0.00060292
9	0.00056198
10	0.00063278
11	0.00064555
12	0.00073523
13	0.00068966
14	0.00073246
15	0.00076255
16	0.00079796
17	0.00090456
18	0.00095864
19	0.0009188
20	0.0010551

Figure 20 Benchmark result of print_backwards

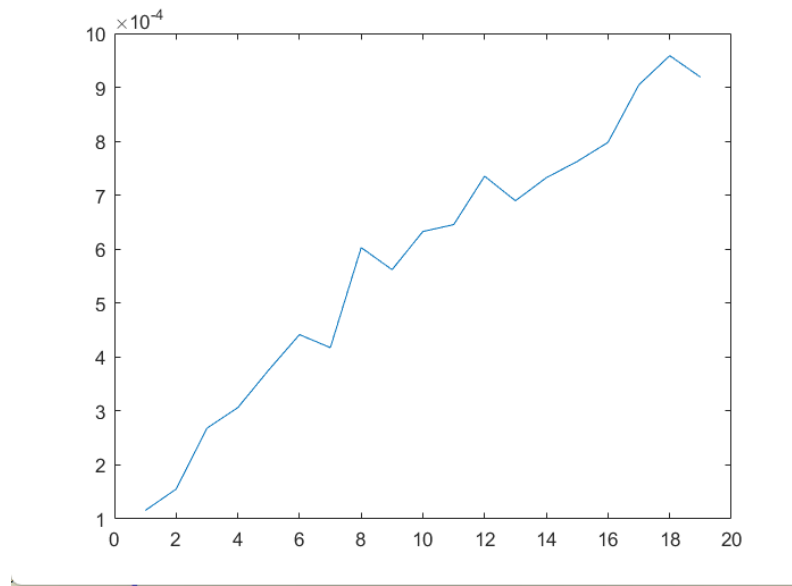


Figure 21 Plot of complexity of `print_backwards`.

As we can see in figure 21, the complexity of `print_backwards` is $O(n)$.

n	time (ns)	n	time (ns)
1	3e-08	1	3e-08
2	2e-08	2	2e-08
3	3e-08	3	2e-08
4	1e-08	4	1e-08
5	2e-08	5	1e-08
6	2e-08	6	1e-08
7	4e-08	7	2e-08
8	3e-08	8	1e-08
9	3e-08	9	2e-08
10	2e-08	10	2e-08
11	3e-08	11	2e-08
12	2e-08	12	1e-08
13	5e-08	13	2e-08
14	3e-08	14	2e-08
15	2e-08	15	1e-08
16	3e-08	16	2e-08
17	4e-08	17	3e-08
18	4e-08	18	2e-08
19	3e-08	19	2e-08
20	2e-08	20	1e-08

Figure 22 Benchmark results of `gcd` and `lcm` respectively

The complexities of `gcd` `lcm` algorithms are very small which is compatible with expectations.