Problem: Board Game

Tile Game

- Two-sided Tiles

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

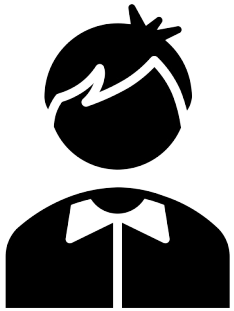
Front Side

9	12	6	4
8	5	+	7
11	−	÷	10
×	1	3	2

Back Side

Tile Game

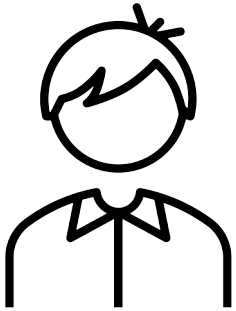
- At the start of game, two players can see the **<u>back side</u>** of the board for **<u>5 seconds</u>**.



Player 1

9	12	6	4
8	5	+	7
11	−	÷	10
×	1	3	2

Back Side



Player 2

Board is composed of
12 numbers from (1~20)
4 operations from (+,x,-,/,^)
are sampled without replacement

Tiles.txt file will be given.

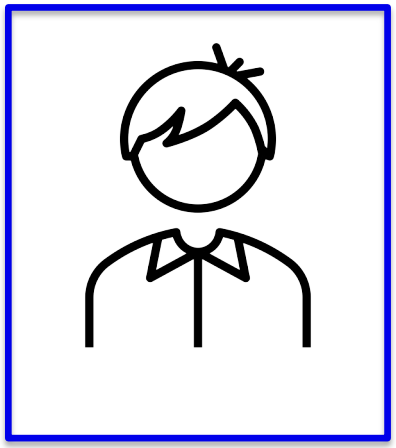
Tile Game

- Each player has an assigned color and can only use numbers that are matched own color.
  (Player 1: red / Player 2: blue)

Player 1

9	12	6	4
8	5	+	7
11	−	÷	10
×	1	3	2

Back Side



Player 2

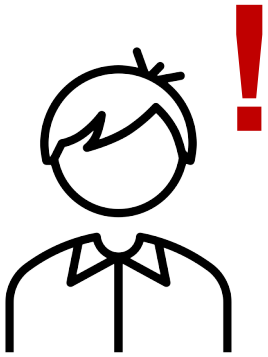
Tile Game

- After 5 seconds, players **<u>can not see</u>** the back side of the tile anymore.

Player 1

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Front Side



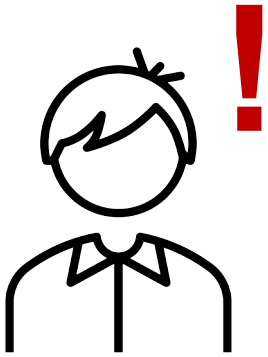
Player 2

Tile Game

- Then, the target number is generated.

Player 1

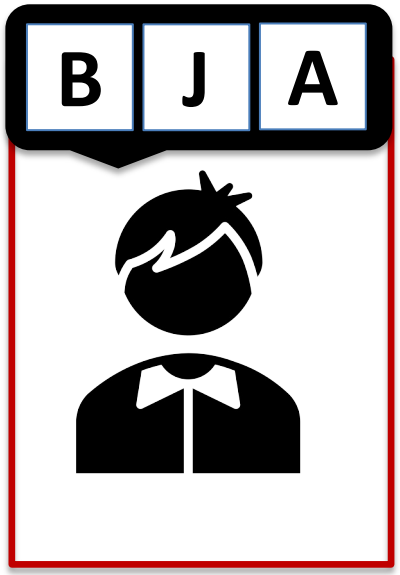
Front Side



Player 2

Tile Game

- Player should call 3 alphabets for making the target number. **In first round, player 1 starts first**



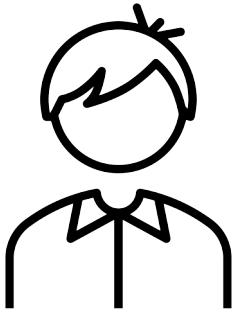
Player 1

Target Num: 14

9	12	C	D
E	F	G	H
I	–	K	L
M	N	O	P

Front Side

Round 1

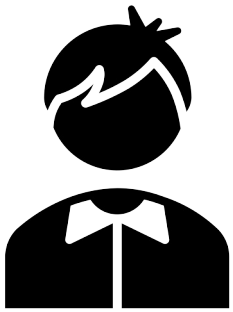


Player 2

$$12 - 9 = 3$$ **: Incorrect!**

Tile Game

- If the answer is incorrect, other player gets turn



Player 1

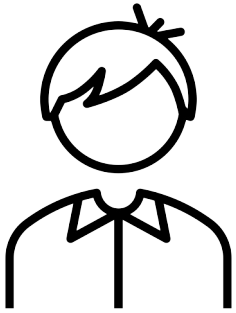
Target Num: 14

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Front Side

Round 1

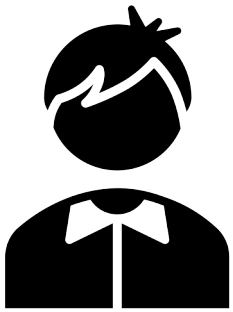
Turn



Player 2

Tile Game

- If the answer is correct, player 2 gets the winning point (+1) and the round ends.

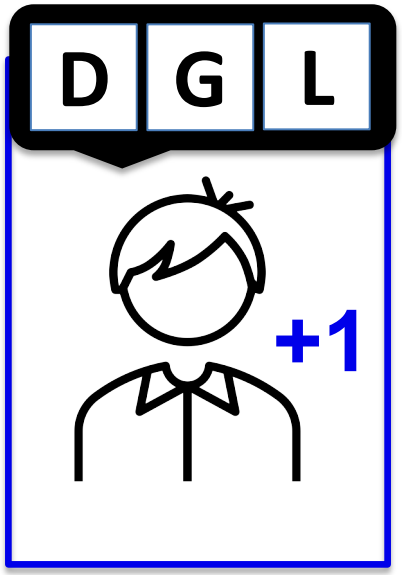


Player 1

Target Num: 14

A	B	C	4
E	F	+	H
I	J	K	10
M	N	O	P

Front Side

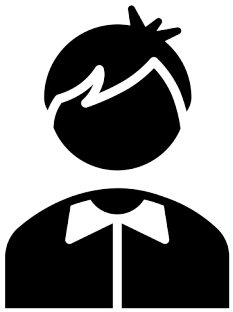


Player 2

$$\mathbf{4} + \mathbf{10} = 14$$ **: Correct!**

Tile Game

- In each round, the board **is changed**, two players can see the **back side** of the board for **3 seconds**.



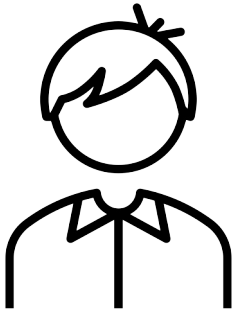
Player 1

9	12	6	4
8	5	+	7
11	−	÷	10
×	1	3	2

Back Side

3 seconds

Round 2

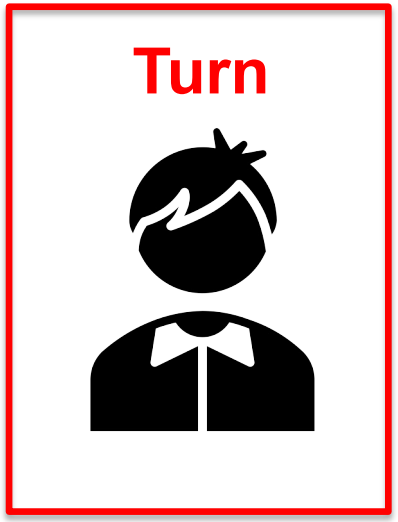


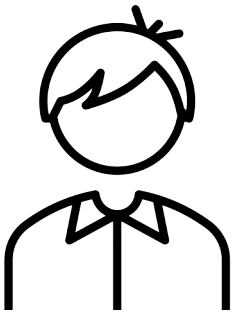
Player 2

Tile Game

- The new target number is generated with changed board for next round.
  - (Since the player 2 got the correct answer in round 1, now player 1 has the priority)

Round 2





Player 1 Front Side Player 2

Tile Game

- Calculate all possible outputs (**natural number**) calculated with two numbers (**same color**) and one operator.
- Target Number is selected from the intersection of the two output arrays (**red & blue**)

$$2 + 6 = 8$$
$$\vdots$$
$$9 + 12 = 21$$
$$\vdots$$
$$7 - 2 = 5$$
$$\vdots$$
$$2 \times 6 = 12$$
$$\vdots$$

9	12	6	4
8	5	+	7
11	−	÷	10
×	1	3	2

$$1 + 3 = 4$$
$$\vdots$$
$$10 - 5 = 5$$
$$\vdots$$
$$10 \div 1 = 10$$
$$\vdots$$
$$4 \times 3 = 12$$
$$\vdots$$

$$[\,8\ \cdots, 21, \cdots \mathbf{5}, \cdots, \mathbf{12}, \cdots\,] \quad \cap \quad [\,4, \cdots, \mathbf{5}, \cdots, 10, \cdots \mathbf{12}, \cdots\,]$$

Target numbers are valid if both players can calculate.

Tile Game

- Elements of arrays can not be unique.
- Intersection should consider non-distinct element cases.
  - Ex) red= [1,1,1,2,3,4,4,4,5,5,6,6,7]

    blue = [1,1,3,3,4,5,5,6,7,7,7,7,7]

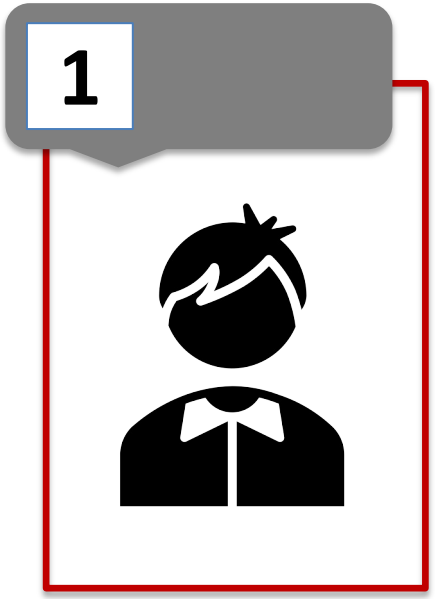
    Intersection= [1,1,3,4,5,5,6,7]
  - Hint: Use two pointers for indexing and compare!
- Algorithm:
  - 1) Generate two arrays which consists of all possible outputs with (same colored) numbers and ops.
  - 2) Sorting two arrays
  - 3) Finding intersection with two sorted arrays
  - 4) Randomly select one number from generated intersection

Tile Game

- The game is composed of **N(≥3)** rounds, and the player who gets more points becomes winner.
- At the beginning of each turn, player can use chances
- Chances
  - Chance can be used at most **once** in each **turn**.
  - Chance-1: Skip this turn
  - Chance-2: Show the back of the board
  - Chance-3: Double Chance
- Two Chance-1 and Two Chance-2 are given to each player
- Note that the number of Chance-3 is unlimited.

Tile Game

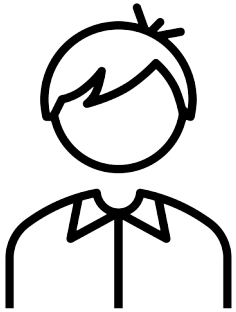
- Chance 1: Turn-over



Player 1

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

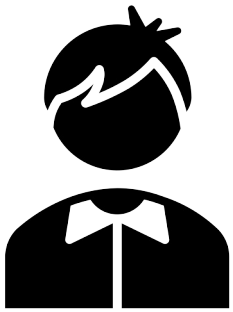
Front Side



Player 2

Tile Game

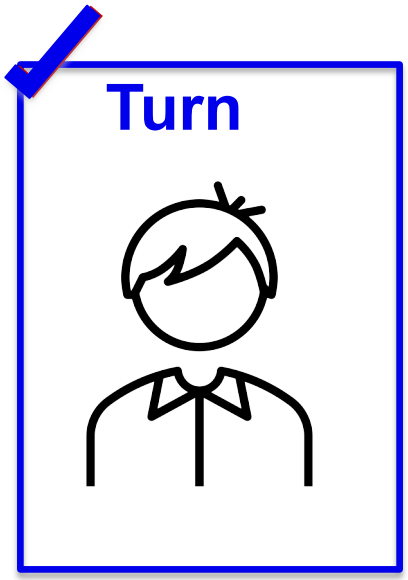
- Chance 1: Turn-over



Player 1

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

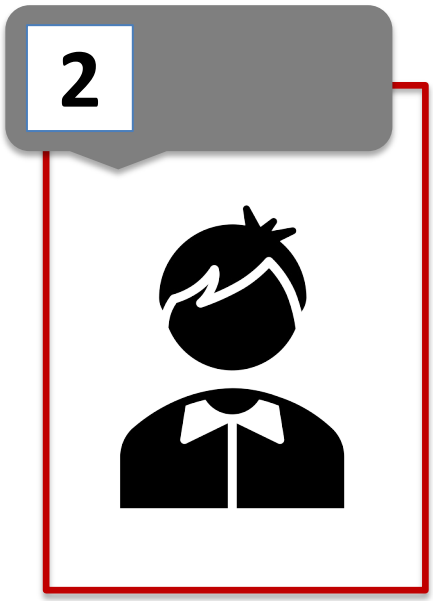
Front Side



Player 2

Tile Game

- Chance 2: Show me the back of the board

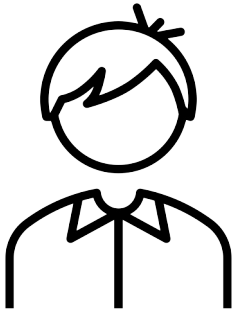


Player 1

5 seconds

9	12	6	4
8	5	+	7
11	−	÷	10
×	1	3	2

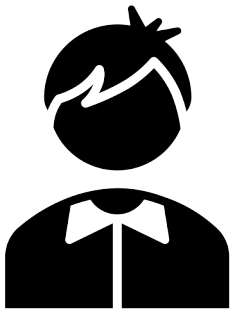
Back Side



Player 2

Tile Game

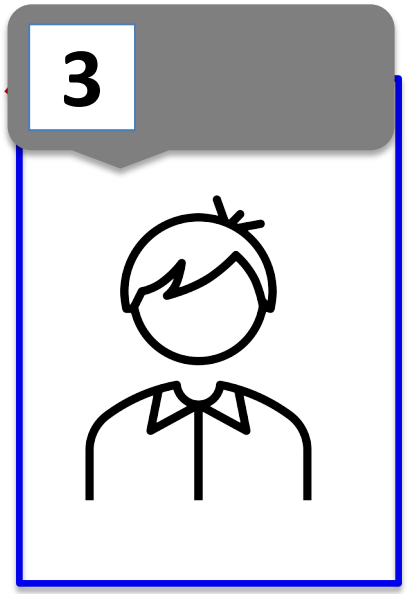
- **Chance 3: Double chance**



Player 1

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

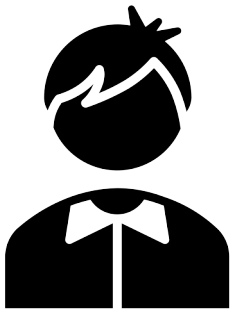
Front Side



Player 2

Tile Game

- Chance 3: Double chance (get 2 points)

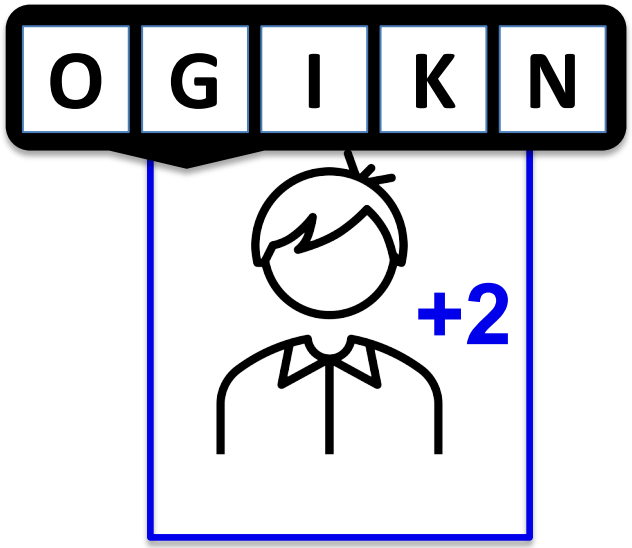


Player 1

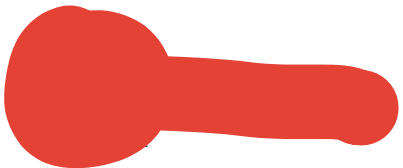
Target Num: 14

A	B	C	D
E	F	+	H
11	J	÷	L
M	1	3	P

Front Side



Player 2



$$3 + 11 \div 1 = 14$$

Tile Game

- The game is composed of N(≥3) rounds, and the player who gets more points becomes winner.
- Each **round** is composed of 4-steps:
  - 1) Target number generation
  - 2) Chance selection
    - Player chooses whether to use 3 different chances or not.
  - 3) Tile selection
    - Player enters three unique alphabets to complete equation.
  - 4) Result validation
    - Program checks whether the player's answer is correct or not.
    - If correct, round is done. If wrong, turn is changed.

Implementation Details

Implementation Details

- Input: Tile.txt

```
R9 R12 R6 B4
R8 B5 W+ R7
B11 W- W/ B10
Wx B1 B3 R2
```

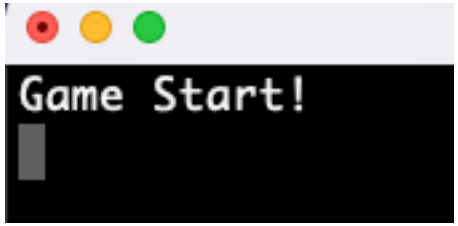
1) First character in each element denotes color
(R: red, B: black, W: default)

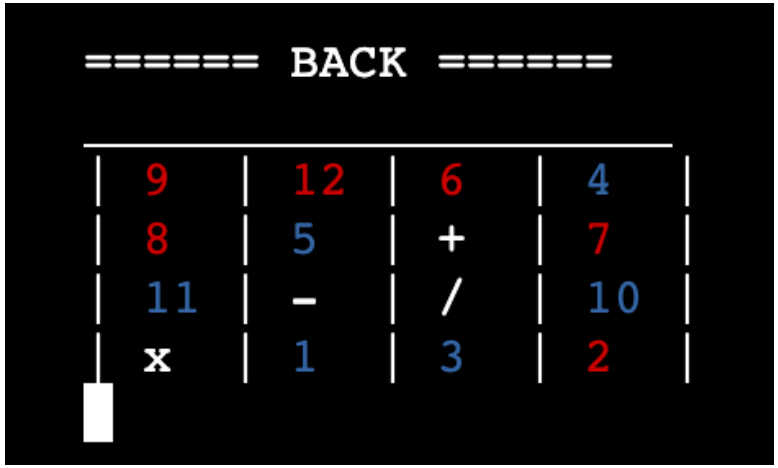
2) Other characters denote numbers or operations

3) Board is composed of
12 numbers from (1~20)
4 operations from (+,x,-,/,^)
sampled without replacement
(sampling code is given)

Implementation Details

- After starting the game, your program shows the back side of the board for 5 seconds.





1) Print "Game Start!"

2) Clear Terminal

3) Show the back side of the board for 5 seconds

4) Clear Terminal

Implementation Details

- Print colored text in console: use **ANSI escape code**

```
#include <stdlib.h>
#include <iostream>
using namespace std;

#define COLOR_RED     "\033[31m"      /* Red */
#define COLOR_DEFAULT   "\033[0m"
#define COLOR_BLUE     "\033[34m"      /* Blue */

int main()
{
    int a[4] = {1, 3, 4, 5};
    cout << COLOR_BLUE << a[0] << " a" << endl;
    cout << "b" << endl;
    cout << COLOR_RED << a[1] << COLOR_DEFAULT << " c" << endl;
    cout << "d" << endl;

    return 0;
}
```

Outcome

```
1
3
4
```

You can simply use pre-defined color code.

Implementation Details

- **Step 1.** Target number generation
  - Recap: Round is composed of 4-steps:

    1) Target number generation
    2) Chance selection
    3) Tile selection
    4) Result validation

  - Each alphabet-tile can be used only once.
  - Note that target number is generated before selecting chances:
    ➔ All possible **natural numbers** that can be made by **two** numbers and **one** operation in the board.
    (No need to consider three num / two operation)

Implementation Details

- **Step 1.** Target number generation
  - Generating function will be given: (Do not fix it)

```
int Game::generate_target_num(vector<int> intersection){
        int size_intersection= intersection.size();
        int random_idx;
        if (size_intersection!=0) {
                random_idx = rand() % size_intersection;
                target_num= intersection[random_idx];
                return target_num;
        } else {return -1;}
}
```

  - **TO DO:**
    1) Implement *generate_target_num*, possible_target_num, find_intersection_non_distinct, sorting.
    2) In *Game::run()*, you should also consider extreme case if size of intersection is 0, in that case, you should change the color of board and find intersection iteratively until size of intersection > 0.

       (HINT: use while)

Implementation Details

- **Step2.** Chance selection

```
Player 1's turn!

Round 1 / 5
Current score:  [Player 1: 0 | Player 2: 0 ]

Target Number:  12

Player 1's Remaining Chance List:
Chance 1:2,  Chance 2:2,  Chance 3: unlimited

Use Chance?
Press 0: No chance
Press 1: Skip this turn!
Press 2: Show back sides of all alphabet tiles!
Press 3: Double Chance!
>>
```

1) Clear the Terminal

2) Show the player's turn, rounds and current scores
(Player 1 starts first in the first round)

3) New target number is printed

4) Show chance lists and the remaining chances for corresponding player

Implementation Details

- **Step2.** Chance selection: (Chance 1) Skip turn

```
Player 1's turn!

Round 1 / 5
Current score:  [Player 1: 0 | Player 2: 0 ]

Target Number:  12

Player 1's Remaining Chance List:
Chance 1:2,  Chance 2:2,  Chance 3: unlimited

Use Chance?
Press 0: No chance
Press 1: Skip this turn!
Press 2: Show back sides of all alphabet tiles!
Press 3: Double Chance!
>> 1
```

```
Player 2's turn!

Round 1 / 5
Current score:  [Player 1: 0 | Player 2: 0 ]

Target Number:  12

Player 2's Remaining Chance List:
Chance 1:2,  Chance 2:2,  Chance 3: unlimited

Use Chance?
Press 0: No chance
Press 1: Skip this turn!
Press 2: Show back sides of all alphabet tiles!
Press 3: Double Chance!
>>
```

1) Turn is changed (Player 1 -> Player 2)
2) Note that the player 2's remaining number of chance-1 is still 2. (Only the player-1's remaining number of chance-1 is changed)

Implementation Details

- **Step2.** Chance selection : (Chance 2) Show the back side

```
Player 1's turn!

Round 1 / 5
Current score:  [Player 1: 0 | Player 2: 0 ]

Target Number:  6

Player 1's Remaining Chance List:
Chance 1:2,  Chance 2:2,  Chance 3: unlimited

Use Chance?
Press 0: No chance
Press 1: Skip this turn!
Press 2: Show back sides of all alphabet tiles!
Press 3: Double Chance!
>> 2
====== BACK ======
____________________
| 9  | 12 | 6  | 4  |
| 8  | 5  | +  | 7  |
| 11 | -  | /  | 10 |
| x  | 1  | 3  | 2  |
```

1) If the player uses chance 2, the program should print the back-side of the board for 5-seconds.

2) Turn is not changed.

Implementation Details

- **Step3/4** Tile selection / Result validation

```
Player 1's turn!

Round 1 / 5
Result:  [Player 1: 0 | Player 2: 0 ]

Target Number:  12
Enter alphabets for flipping tile on board
====== BOARD ======

--------------------
| A  | B  | C  | D  |
| E  | F  | G  | H  |
| I  | J  | K  | L  |
| M  | N  | O  | P  |
>> CMP
Result: 12
Correct: Round done!
====== BOARD: One-flipped ======

--------------------
| A  | B  | 6  | D  |
| E  | F  | G  | H  |
| I  | J  | K  | L  |
| x  | N  | O  | 2  |
```

1) Clear the Terminal

2) Show the player's turn, rounds, current scores, and target number

3) Show the front-side of the board

4) Enter three alphabets at a time.

5) After entering each alphabet, corresponding element (number or operation) will be flipped.
: Flip identically when result is false.

Implementation Details

- Next round

```
====== BACK ======

-------------------
| 9  | 12 | 6  | 4  |
| 8  | 5  | +  | 7  |
| 11 | -  | /  | 10 |
| x  | 1  | 3  | 2  |
```

1) Clear terminal

2) Board is changed (Position of color is shuffled)
   : Shuffling rule is fixed (You don't have to implement it)

3) Show the changed board(only color, 5 seconds)

Implementation Details

- Next round

```
Player 2's turn!

Round 2 / 5
Current score:  [Player 1: 1 | Player 2: 0 ]

Target Number:  3

Player 2's Remaining Chance List:
Chance 1:2,  Chance 2:2,  Chance 3: unlimited

Use Chance?
Press 0: No chance
Press 1: Skip this turn!
Press 2: Show back sides of all alphabet tiles!
Press 3: Double Chance!
>>
```

1) Clear terminal

2) When the new round begins, priority is changed.
   - If the player 1(2) gets a point in the previous round, now it's time for player 2(1).

3) New target number is printed.

Implementation Details

- **Step3/4** (Chance 3) Double case

```
Player 2's turn!

Round 2 / 5
Result:  [Player 1: 1 | Player 2: 0 ]

Target Number:  3
Enter alphabets for flipping tile on boar
====== BOARD ======
--------------------
| A  | B  | C  | D  |
| E  | F  | G  | H  |
| I  | J  | K  | L  |
| M  | N  | O  | P  |
>>HJEKP
Result: 3
Correct: Round done!
====== BOARD: Selected-flipped ======
--------------------
| A  | B  | C  | D  |
| 8  | F  | G  | 7  |
| I  | -  | /  | L  |
| M  | N  | O  | 2  |
```

1) If chance-3 is used, the player can use 5-alphabets for calculating the target number.
(Thus, input size should be five, in contrast to 3 [standard case])

2) If the answer is correct, the player will get **2 points**, not 1 point.

Implementation Details

- Final result

```
Result of player 1
Score: 3
Result of player 2
Score: 5
Player 2 win
Thanks for playing the game!
20XX-XXXXX Name
```

1) After all rounds are done, print the final score of player 1 and player 2

2) Player who gets more scores becomes a winner

3) Draw case can be possible

Implementation Details

- Exception case
  - 1. Incorrect case
    - Incorrect-case **1**: Invalid formula
      EX-1) [+ 7 8]
      EX-2) [7 8 x]
      ➔ Print ("False! Invalid Formula") and then turn is changed immediately

    - Incorrect-case **2**: same alphabet is used more than once
      EX-1) [A M A], [A A M], [M A A]
      ➔ Print ("False!: You entered an alphabet more than once!") and then turn is changed.

Implementation Details

- Exception case
  - 1. Incorrect case
    - Incorrect-case **3**: Unmatched color

      Ex) Player 1 (Red): 3 + 10

      ➔ Print ("Color unmatching : Change turn") and then turn is changed.

  - 2. When all the chances are used

    EX-1) If chance-1 is entered when there are no more chances left

    ➔ Print ("No remaining chance-1") and then player is given opportunities to re-enter the chance type (Turn is not changed)

Details with examples are described in appendix

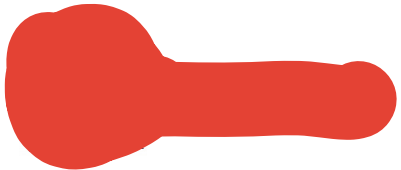
Implementation Details

- Exception case
  - In some cases, exceptions occur concurrently:
    - Ex) If A: number , C: operation,
      - User input: A A C
      - In this case, both incorrect case-1 (invalid formular) and incorrect case-2 (repeated alphabets) occur at the same time

You don't have to consider the priority among the exceptions.

It is okay if your exception is appropriate for the situation.

Problem: Calculator using recursion



Calculator

- Calculator
  - For board game, we need calculator.
  - Simple Calculator
    - For considering board game only, you don't have to implement complicated calculator

    - Length of input formular is fixed (3,5).
    - Inputs are natural numbers:

      ➔The number of cases are limited.

      ➔Comparing priorities between operations is **simple**

      ➔ You can simply implement with **if-statement** only.

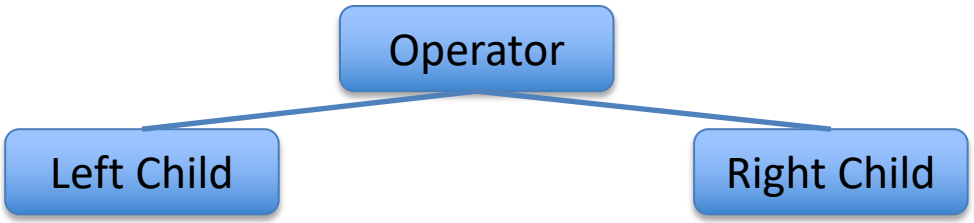
Calculator

- Calculator
  - General Calculator
    - But what if length of input formular is larger than 5?
    - What if inputs can be real numbers?
  - Simple Calculator VS General Calculator
    - You can select among two calculator.
    - In overall grading, calculator accounts for 30%.
      - If you implement Simple calculator, you can get at most 80% in calculator part.
      - If you implement General calculator, you can get at most 100% in calculator part.

We strongly recommend that you implement the Simple calculator first.

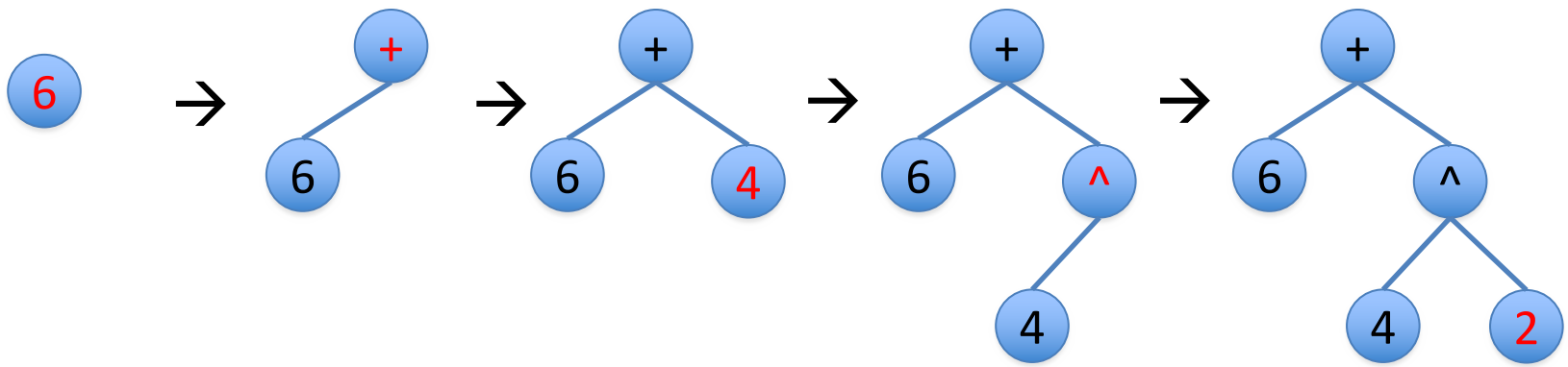
Calculator

- Constructing tree structure to compute the expression
- Two elements can form a node
  1. Number
     - Add to current tree
  2. Operator
     - Consider the priority and make new sub-tree
       - Higher priority → Deeper in the tree



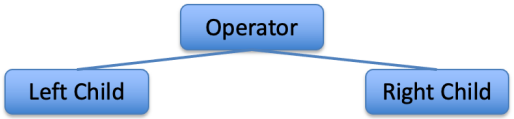
Calculator

- Example of tree structure : 6 + 4^2 x 3



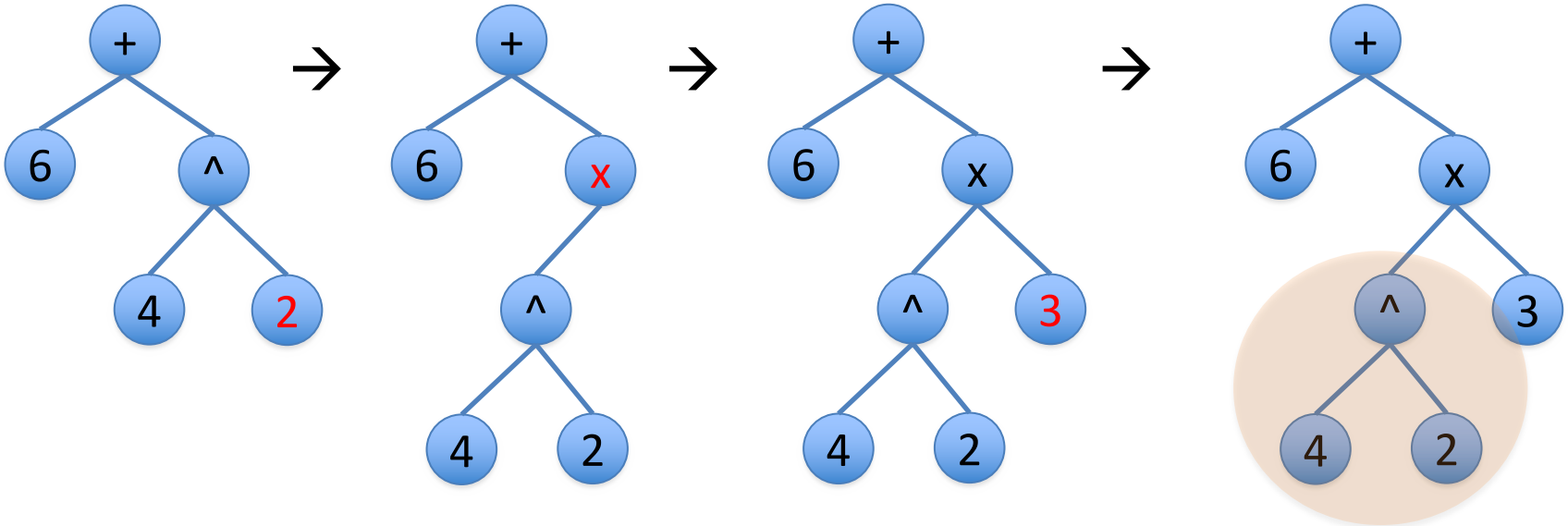
1. Number
   - Add to current tree

2. Operator
   - Consider the priority and make new sub-tree
   - Higher priority → Deeper in the tree



Calculator

- Example of tree structure : 6 + 4^2 x 3

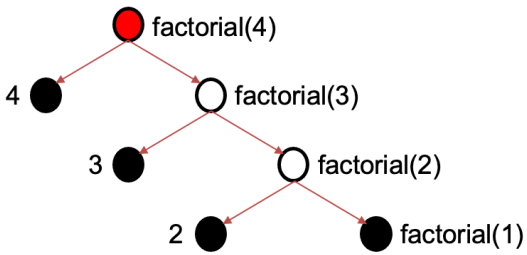


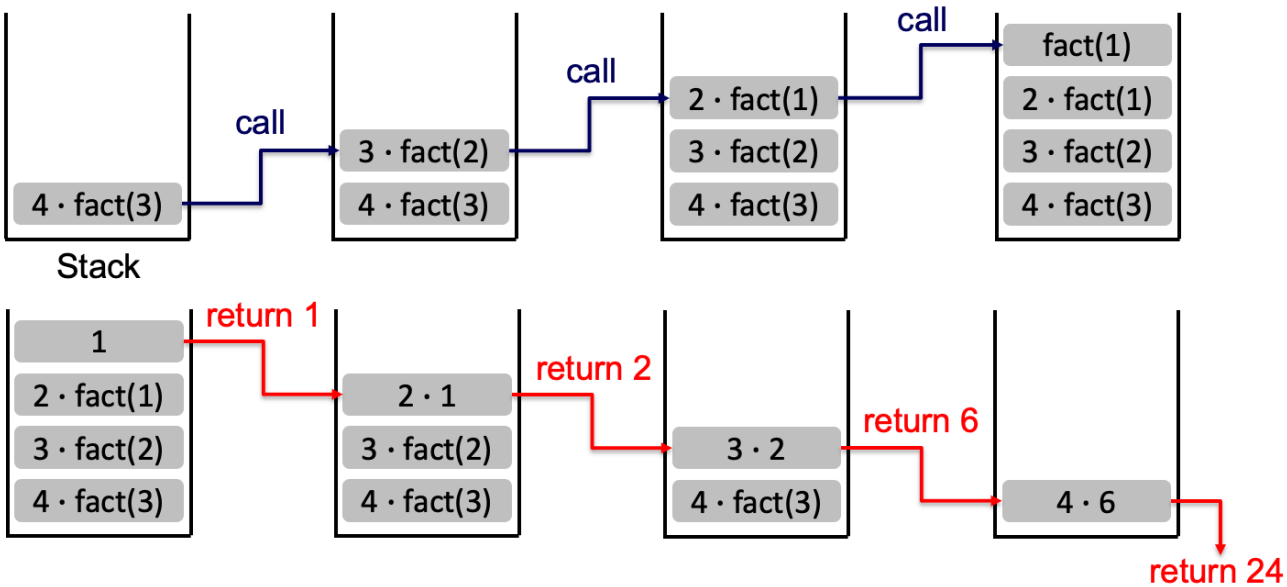
→ Deeper subtree should be computed first

How can we implement this Tree Structure??

Recursion : Stack

- Recall Factorial





We can implement Tree Structure by **Stack**

Implementation

- Functions
  - EvaluateTerm(const char s[], int& idx)
    - Checks the string and if it is not an operator converts the string to float number
  - StringToNumber(const char s[], int& idx)
    - It converts the string to float number
  - OperatorPriority(char op)
    - It returns the priority of given operator 'op'
  - CalculateOperator(double left, double right, char op)
    - It returns the result calculated by left and right using the given operator 'op'

Implementation

- Constants
  - const int CALC_MAX_EXPR_LENGTH
    - Maximum number of the characters in a single expression
  - const int CALC_MAX_OPERATOR_LEVEL
    - Maximum level of the priority of operators

- In CalculateExpression() function,
  - Base Case

    Return a number if the term is not an operator
  - Recursive Step

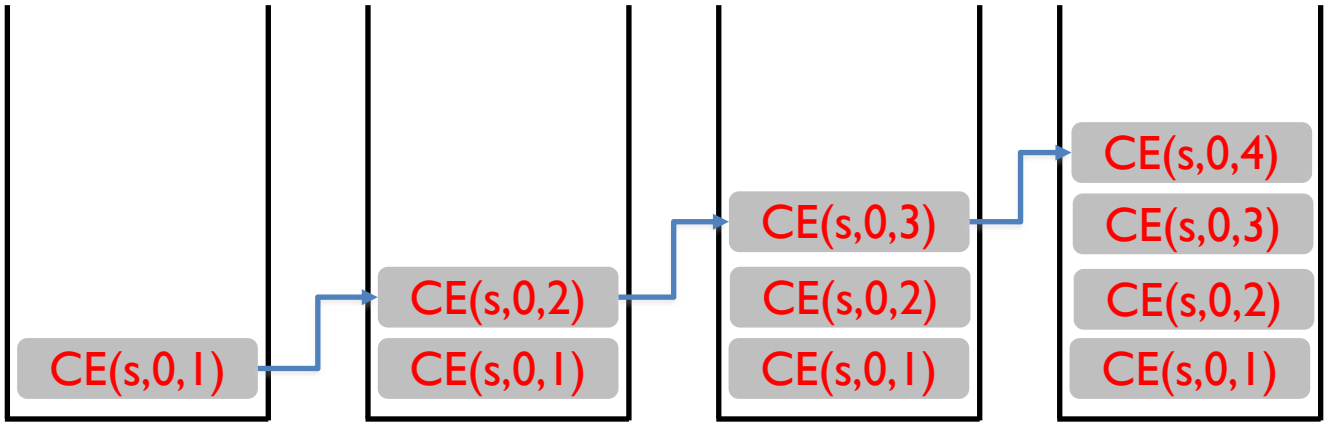
    1) Check the priority of operator and move the index to the next number

    2) Recursively set the left and right values of the operator

Algorithm detail

- CalculateExpression() CE(s, idx, level)
  - If the level exceeds a MAX_OPERATOR_LEVEL, it means that it is not an operator, so you should call the EvaluateTerms

idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3
	&idx						

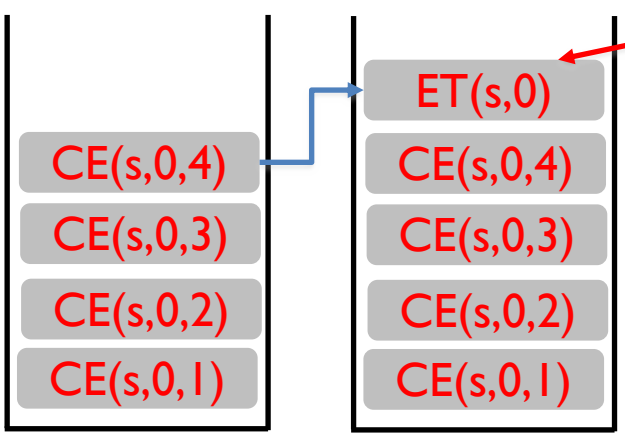


Algorithm detail

- CalculateExpression()
  - EvaluateTerms return what the terms are

idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3
	&idx						

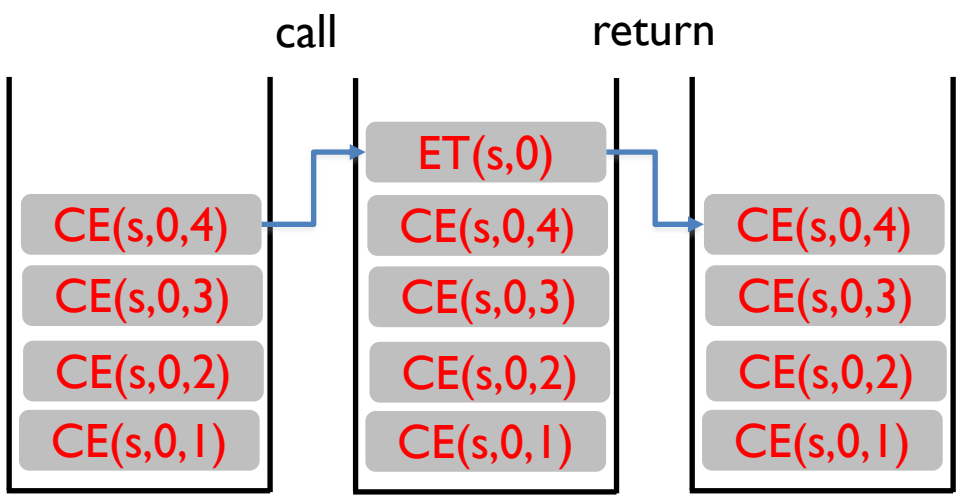
When we call ET(), idx is updated to the index of next operator



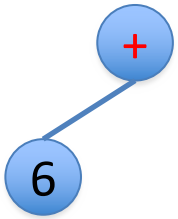
Algorithm detail

- CalculateExpression()
  - EvaluateTerms return what the terms are

idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3
		&idx					



Left=6
op= "+"



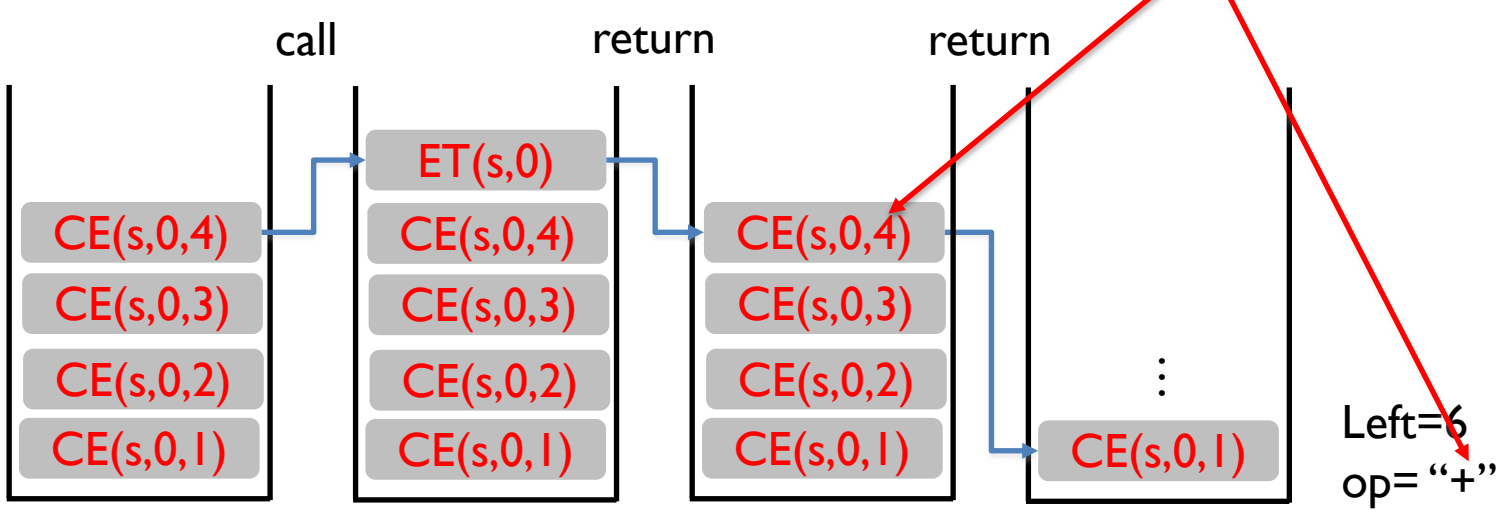
Algorithm detail

- CalculateExpression()
  - EvaluateTerms return what the terms are

idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Compare the level of operator



Algorithm detail

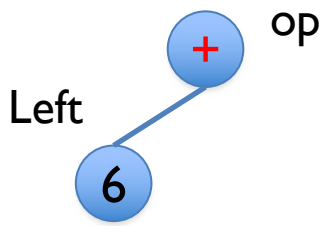
- CalculateExpression()

idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

Stack

&idx

Priority=1



CE(s,0,1)

While returning CE, if the priority of the operator and the current level are the same, index++ and call CE again

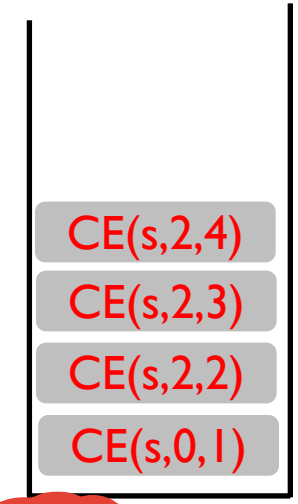
Algorithm detail

- CalculateExpression()

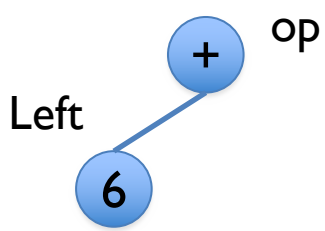
idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack



next level

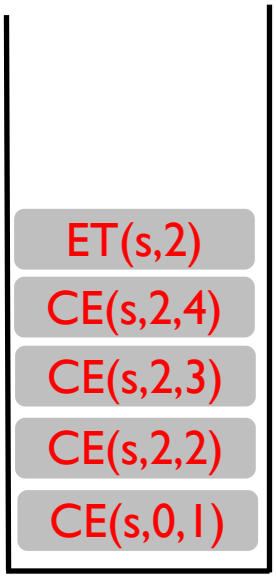


When we call ET(), idx is updated to the index of next operator

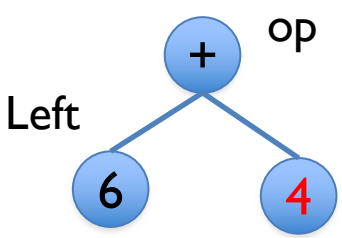
idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack



→4



Algorithm detail

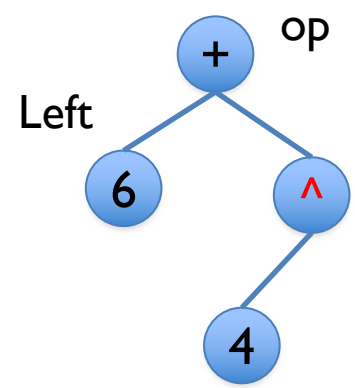
- CalculateExpression()

idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

Stack

&idx

Priority=3



CE(s,2,3)

CE(s,2,2)

CE(s,0,1)

While returning CE, if the priority of the operator and the current level are the same, index++ and call CE again

Algorithm detail

- CalculateExpression()

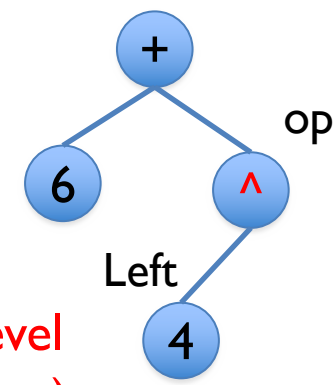
idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack

CE(s,4,4)
CE(s,4,3)
CE(s,2,3)
CE(s,2,2)
CE(s,0,1)

same level (for '^' op)



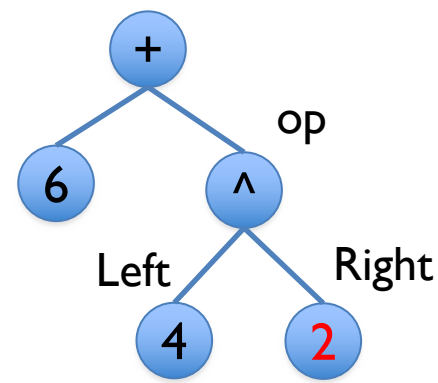
idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack

ET(s,4)
CE(s,4,4)
CE(s,4,3)
CE(s,2,3)
CE(s,2,2)
CE(s,0,1)

→2



Algorithm detail

- CalculateExpression()

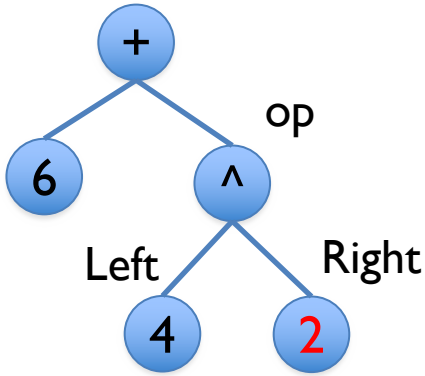
idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack

CE(s,2,3)
CE(s,2,2)
CE(s,0,1)

Left:4
Op:^
Right: 2

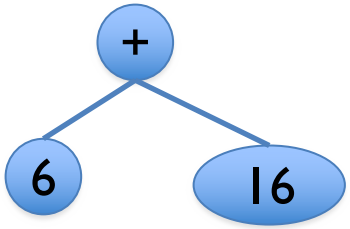


idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack

CE(s,2,2)
CE(s,0,1)



→ Since priority of "X" is 2, which is same as the level, we call CE again

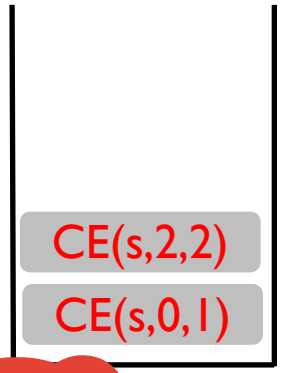
Algorithm detail

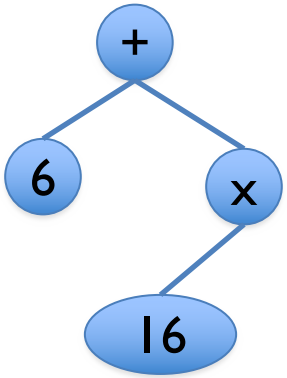
- CalculateExpression()

idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack

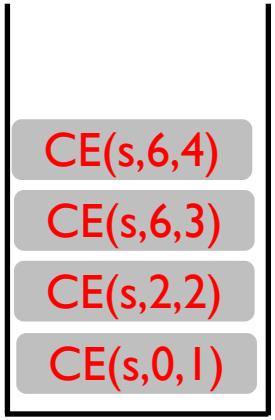




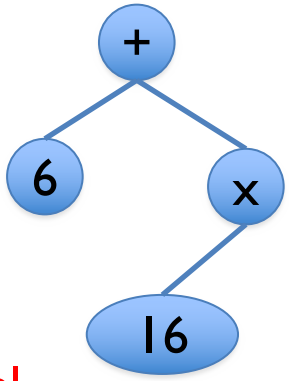
idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack



next level



Algorithm detail

- CalculateExpression()

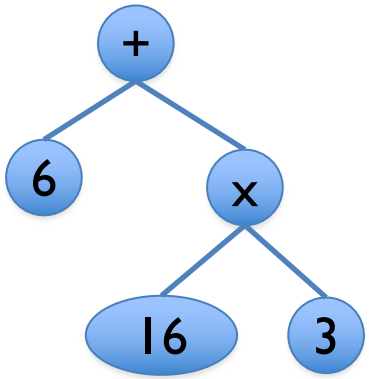
idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack

ET(s,6)
CE(s,6,4)
CE(s,6,3)
CE(s,2,2)
CE(s,0,1)

→3



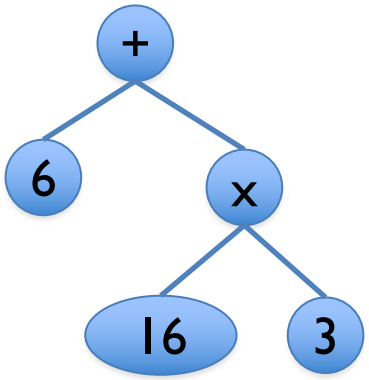
idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

Stack

CE(s,2,2)
CE(s,0,1)

Left:16
Op:x
Right: 3

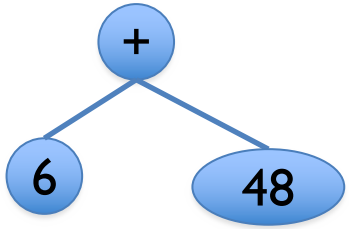


Algorithm detail

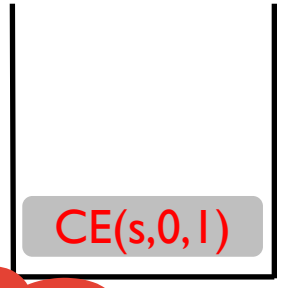
- CalculateExpression()

idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx



Stack



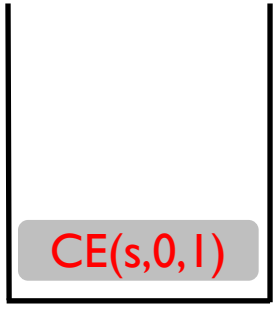
Left:6
Op:+
Right: 48

idx	0	1	2	3	4	5	6
s	6	+	4	^	2	X	3

&idx

54

Stack



Left: 54 → return

Submit

- Rules for Project 1
  - **DO NOT FIX**

1. Given function:

   generate_target_num(), color_change_board(), shuffle() function

2. Declarations of Simple_calculator class / General_calculator class

➔ (names, parameters and return-type of member function should not be fixed)

3. Names, parameters and return-type of

   possible_target_num(), sorting(), find_intersection(), is_integer() function.

4. Output printing

  - **YOU CAN FIX**

    Other implementations are free to modify

Submit

- Rules for Project 1
  - Implementation
    - You can not include any other header files. Ex) <algorithm>
    - Use "sleep()" whenever you use "system ("clear")" so that users can check the changes.
    - You do not have to worry about printing message or board, **except the final outputs.**
    - **One** chance selection step **at each turn.** Namely, after selecting chance, tile selection proceeds. Ex) Use chance 2 => Use chance 3: X
  - Exception
    - In some cases, several exceptions are occurred concurrently

    ➔ Don't have to consider priority among exceptions.
    - You do not have to consider other exception cases.

Submit

- Rules for Project 1
  - There are 6 test cases(4 cases are opened) for "Board game"(70%), and 6 test cases(3 cases are opened) for "Calculator"(30%)
  - Test your code & Push "제출" button on Elice
  - Submit a zip file which includes source codes and video to Elice
  - No need to submit a report, but, you should submit a video for test case : "20XX-XXXXX_proj1.mov" (not .avi)
    - Check the next slide for the input sequence of the test case
  - CPP files name: "20XX-XXXXX_game.cpp" / "20XX-XXXXX_calculator.cpp"
  - The submission title should be "20XX-XXXXX_proj1.zip"

Submit

- Test case for recording the video

1. 2
2. PMC
3. 0
4. ABC
5. 3
6. AMDKB
7. 0
8. GNO
9. 1
10. 3
11. FGBKD
12. 0
13. AGF
14. 3
15. EGHJD
16. 2
17. CKC
18. 0
19. FGH

If you can't complete the code about this input, choose one of the 3 questions posted on Elice and record a video

Ex) "20XX-XXXXX_proj1_Q2.mov"

Score (N/10)

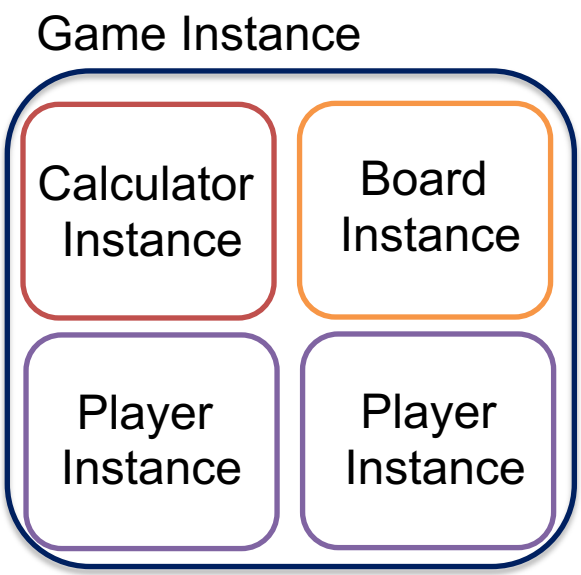
- Q1 : 3
- Q2, Q3 : 6

Implementation Details

- Explanation about skeleton codes
  - Game Class

```
class Game {
public:
    Game(const char board_front[][TILE_BOARD_LENGTH], pair<string,char> board_back[][TILE_BOARD_LENGTH]);
    void run();
    bool play_turn();
    void print_final_result() const;
    bool handle_tile_exception(string alphabets,int input_size,int& current_player_idx) ;
    int generate_target_num(vector<int> intersection);
    vector<int> possible_target_num(char color) ;
    void sorting (vector<int>& arr);
    vector<int> find_intersection( vector<int> left_arr, vector<int> right_arr,int left_len, int right_len) const;
    bool is_integer (double number) const;

private:
    int current_player_idx;
    Player players[2];
    Board board;
    //Simple_Calculator calculator;
    General_Calculator calculator;
    int round;
    int target_num;
};
```



Note that 'Game' Class has Calculator, Player, Board instances.

Implementation Details

- (Incorrect case 1) Invalid formula

```
Player 1's turn!

Round 4 / 5
Result:  [Player 1: 2 | Player 2: 2 ]

Target Number:  11
Enter alphabets for flipping tile on board
====== BOARD ======

| A | B | C | D |
| E | F | G | H |
| I | J | K | L |
| M | N | O | P |
>>KIN
====== BOARD: Selected-flipped ======

| A | B | C | D |
| E | F | G | H |
| 11 | J | / | L |
| M | 1 | O | P |
False!: Invalid formula : Change turn
```

1) Valid formula for the three alphabets case

   Num | Op | Num

2) Valid formula for the five alphabets case

   Num | Op | Num | Op | Num

3) If the input formula is invalid, your program should print "False! : Invalid formula" and change turn

Implementation Details

- (Incorrect case 2) Tile reuse

```
Player 2's turn!

Round 4 / 5
Result:  [Player 1: 2 | Player 2: 2 ]

Target Number:  11
Enter alphabets for flipping tile on board
====== BOARD ======

_____________________
| A | B | C | D |
| E | F | G | H |
| I | J | K | L |
| M | N | O | P |
>>CGC
False!: You entered same alphabet twice: Change turn
```

User input : CGC

1) The alphabets can not be entered more than once.

2) If the input alphabets are repeated, print "False!: You entered an alphabet more than once!" and turn is changed.

Also applied to the double chance case !!

Implementation Details

- (Incorrect case 3) Wrong answer

```
Target Number:  11
Enter alphabets for flipping tile on board
====== BOARD ======

_____________________
| A  | B  | C  | D  |
| E  | F  | G  | H  |
| I  | J  | K  | L  |
| M  | N  | O  | P  |
>>AGB
Result: 21
False!: Incorrect answer :  Change turn
====== BOARD: Selected-flipped ======

_____________________
| 9  | 12 | C  | D  |
| E  | F  | +  | H  |
| I  | J  | K  | L  |
| M  | N  | O  | P  |
```

1) Although the formula is correct, it can be a wrong answer.

2) Then print "False! : Incorrect answer " and then turn is changed.

Implementation Details

- (Incorrect case 4) Unmatched color

```
Player 2's turn!

Round 4 / 5
Result:  [Player 1: 2 | Player 2: 2 ]

Target Number:  11
Enter alphabets for flipping tile on board
====== BOARD ======

_____________________
| A | B | C | D |
| E | F | G | H |
| I | J | K | L |
| M | N | O | P |
>>IKN
====== BOARD: Selected-flipped ======

_____________________
| A | B | C | D |
| E | F | G | H |
| 11 | J | / | L |
| M | 1 | O | P |
Color unmatching : Change turn
```

1) The Player should choose a number that matches their color.

2) Otherwise, print "Color unmatching : Change turn" and then turn is changed.