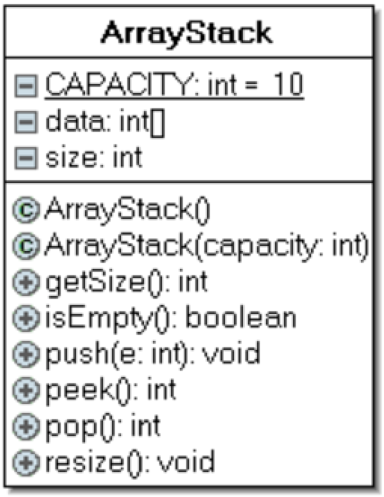
Data Structures and Algorithms Laboratory		
Laboratory 6: Stacks	**School of Information Technology**	
Name: Pattarapon Bunchuai	**ID:** 6431503044	**Section:** 3
Date:	**Due date:** **on the LMS**	

Objective

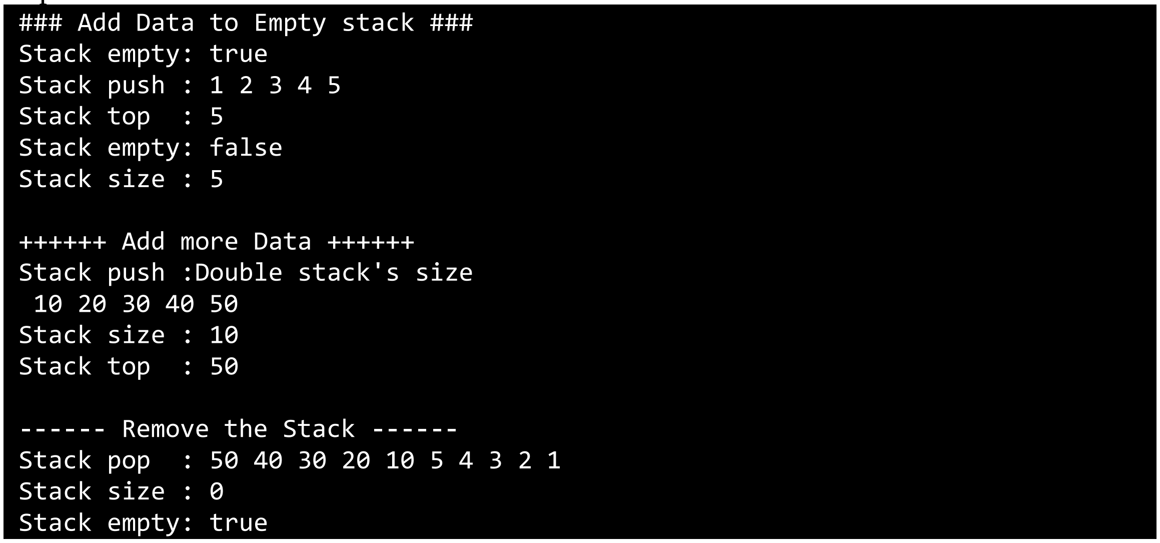
- To create a stack program based on an array
- To develop stack's methods such as push and pop
- To implement the Matching Parentheses
- To implement the Converting Infix to Postfix

Exercise 1(In-Class): From the given class diagram, create an array-based stack and complete the program to get the results as shown.

Diagram:



Expected result:



```
### Add Data to Empty stack ###
Stack empty: true
Stack push : 1 2 3 4 5
Stack top  : 5
Stack empty: false
Stack size : 5

++++++ Add more Data ++++++
Stack push :Double stack's size
 10 20 30 40 50
Stack size : 10
Stack top  : 50

------ Remove the Stack ------
Stack pop  : 50 40 30 20 10 5 4 3 2 1
Stack size : 0
Stack empty: true
```

ArrayStack:

```
class ArrayStack {
        //Note that this is only for an integer stack
        //------------ Data -------------------
        private final static int CAPACITY = 10; //default size
        private int[] data; //array for stack data
        private int size = 0; //stack's size

        //------------ Method -------------------
        //constructor with default capacity
        public ArrayStack() {
                data = new int[CAPACITY];
        }

        //constructor with a given capacity
        public ArrayStack(int capacity) {
                data = new int[capacity];
        }

        //current stack's size, not a capacity
        public int getSize() {
                return size;
        }

        //empty?
        public boolean isEmpty() {
                if (size == 0) {
                        return true;
                }

                return false;
        }
```

```
    //push
    public void push(int e) {
        if (size == data.length) {
            resize();
        }

        data[size] = e;
        size ++;
    }

    //peek
    public int peek() {
        if (size == 0) {
            return -1;
        }

        return data[size-1];
    }

    //pop
    public int pop() {
        if (size == 0) {
            return -1;
        }

        size--;
        return data[size];
    }

    //resize
    public void resize() {
        System.out.println("Double stack's size");
        //create a new array of double size
        int[] temp = new int[2 * data.length];
        //copy old array data to new array
        System.arraycopy(data, 0, temp, 0, data.length);
        //assign the stack to new array
        data = temp;
    }
}
```

MainClass

```
public class MainStack {
       public static void main(String[] args) {
           ArrayStack stack = new ArrayStack(5);
           System.out.println("### Add Data to Empty stack ###");     //Add data
           System.out.println("Stack empty: "+stack.isEmpty());
           System.out.print("Stack push :");
           for(int i=1;i<=5;i++){
             stack.push(i);                                          //push
             System.out.print(" "+i);
           }
           System.out.println();

           System.out.println("Stack top  : "+stack.peek());        //check Top
           System.out.println("Stack empty: "+stack.isEmpty());    //check Empty
           System.out.println("Stack size : "+stack.getSize());    //check Size

           System.out.println("\n++++++ Add more Data ++++++");       //Add more data
           System.out.print("Stack push :");
           for(int j=10;j<=50;j+=10){
             stack.push(j);                                          //push
             System.out.print(" "+j);
           }
           System.out.println("\nStack size : "+stack.getSize());
           System.out.println("Stack top  : "+stack.peek());

           System.out.println("\n------ Remove the Stack ------");    //Remove Stack
           System.out.print("Stack pop  :");
           int s=stack.getSize();                                    //get Size
           for(int k=1;k<=s;k++){
             System.out.print(" "+stack.pop());                      //pop and display
           }
           System.out.println("\nStack size : "+stack.getSize());  //check Size
           System.out.println("Stack empty: "+stack.isEmpty());    //check Empty
       }
}
```

Exercise 2(In-Class): Matching Parentheses

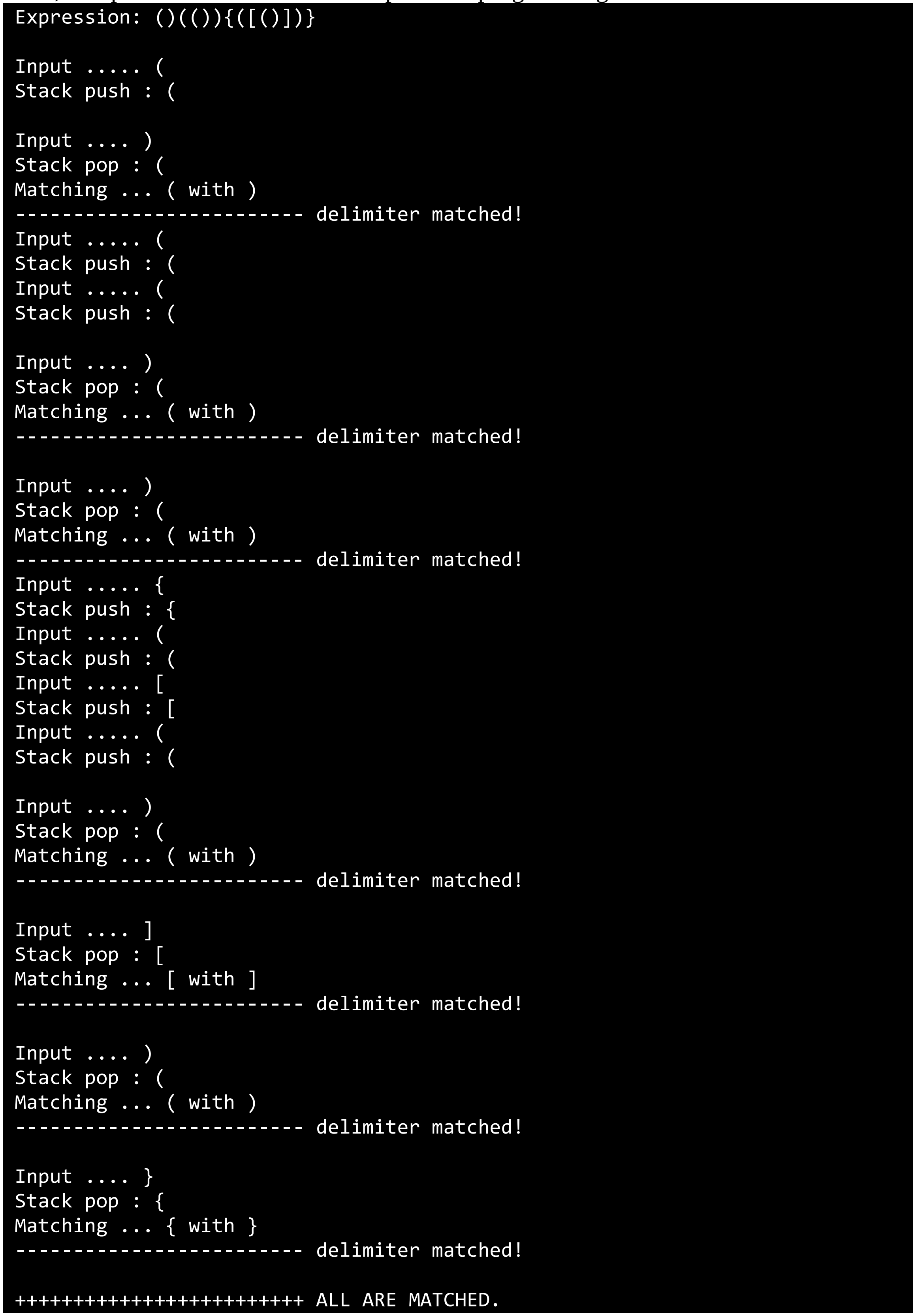
Modify the ArrayStack in Exercise 1 to become **a stack of character data type**. Write a new codes below.

ArrayStack and Stack Code:

```
class ArrayStackChar {
        //------------ Data  -------------------
        private final static int CAPACITY = 10; //default size
        private char[] data; //array for stack data
        protected int size = 0; //stack's size

        //------------ Method  -------------------
        //constructor with default capacity
        public ArrayStackChar() {
                data = new char[CAPACITY];
        }

        //constructor with a given capacity
        public ArrayStackChar(int capacity) {
                data = new char[capacity];
        }

        //current stack's size, not a capacity
        public int getSize() {
                return size;
        }

        //empty?
        public boolean isEmpty() {
                if (size == 0) {
                        return true;
                }

                return false;
        }

        //push
        public void push(char e) {
                if (size == data.length) {
                        resize();
                }

                data[size] = e;
                size ++;
        }

        //peek
        public char peek() {
                if (size == 0) {
```

```
            return ' ';
        }

        return data[size-1];
    }

    //pop
    public char pop() {
        if (size == 0) {
            return ' ';
        }

        size--;
        return data[size];
    }

    //resize
    public void resize() {
        System.out.println("Double stack's size");
        //create a new array of double size
        char[] temp = new char[2 * data.length];
        //copy old array data to new array
        System.arraycopy(data, 0, temp, 0, data.length);
        //assign the stack to new array
        data = temp;
    }

}
```

Then, complete the main class to complete the program to get the results as shown.



```
Expression: ()(()){([()])}

Input ..... (
Stack push : (

Input .... )
Stack pop : (
Matching ... ( with )
------------------------- delimiter matched!
Input ..... (
Stack push : (
Input ..... (
Stack push : (

Input .... )
Stack pop : (
Matching ... ( with )
------------------------- delimiter matched!

Input .... )
Stack pop : (
Matching ... ( with )
------------------------- delimiter matched!
Input ..... {
Stack push : {
Input ..... (
Stack push : (
Input ..... [
Stack push : [
Input ..... (
Stack push : (

Input .... )
Stack pop : (
Matching ... ( with )
------------------------- delimiter matched!

Input .... ]
Stack pop : [
Matching ... [ with ]
------------------------- delimiter matched!

Input .... )
Stack pop : (
Matching ... ( with )
------------------------- delimiter matched!

Input .... }
Stack pop : {
Matching ... { with }
------------------------- delimiter matched!

+++++++++++++++++++++++++ ALL ARE MATCHED.
```

Main Class Source code:

```
public class MainMatching {

      public static void main(String[] args){
          String opening="({[";
          String closing=")}]";
          String expression="()(()){([()])}";

          //Show input message
          System.out.print("Expression: "+expression);
          System.out.println("\n");

          //Create a char stack
          ArrayStackChar stack =  new ArrayStackChar();
          //split String to array of characters
          char[] exp = expression.toCharArray();

          //For each character in the array
          for(char c:exp) {
             //is it opening delimiter?
             //use indexOf() to find character in String, return -1 if not found,
otherwise return position
             //if this opening delimiter is found
             if(opening.indexOf(c) != -1) {
                    //push to stack
                    System.out.println("Input ..... " + c);
                    System.out.println("Stack push : " + c);
                    stack.push(c);
             }
             //is it closing delimiter?
             else if(closing.indexOf(c) != -1) {
                    //is stack empty? --> no matching opening delimiter
                    if(stack.isEmpty()) {
                          System.out.println("\n ***** MISMATCHED DELIMITER! ***** ");
                          //end the program
                          return;
                    }

                    //pop an element
                    System.out.println("\nInput .... " + c);
                    char p = stack.pop();                 ;
                    System.out.println("Stack pop : " + p);
                    System.out.println("Matching ... "+ p + " with "+ c);

                    //is the pop opening element is equal to the current closing
delimiter?
                    //here the positions of both delimiters must be equal
                    if(opening.indexOf(p) != -1) {
                          System.out.println("------------- delimiter matched!");
                    }
                    else {
                          System.out.println("\n ***** MISMATCHED DELIMITER! ***** ");
                          //end the program
                          return;
                    }
```

```
              }//end if else
           }//end of loop

           //---now all delimiters are checked---
           //is there any delimiter left in stack?
           //if stack is empty
           if(stack.isEmpty()){
             System.out.println("\n+++++++++++++++++++++++++ ALL ARE MATCHED.");
           }
           else{
             System.out.println("\n ***** MISMATCHED DELIMITER! ***** ");
             //end the program
             return;
           }

    }// end  of main method
}// end of main class
```

Question: What is the maximum number of parentheses that can be pushed to the stack AT ANY TIME when the algorithm runs? What are they?

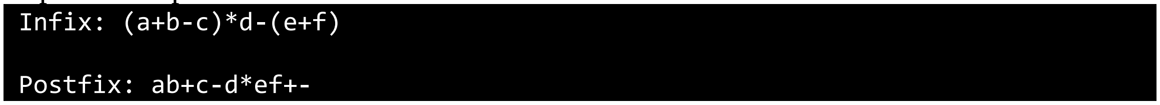
The maximum number of parentheses that can be pushed to the stack is 4. They are {([(

<u>Exercise 3 (Homework)</u>: Converting Infix to Postfix.

Write a program to convert the given infix expression to become the postfix expression.

<u>Expression</u>: (a+b-c)*d–(e+f)

Expected output:



```
Infix: (a+b-c)*d-(e+f)

Postfix: ab+c-d*ef+-
```

ArrayStack and Stack Code:

```
class ArrayStackChar {
        //------------ Data -------------------
        private final static int CAPACITY = 10; //default size
        private char[] data; //array for stack data
        protected int size = 0; //stack's size

        //------------ Method -------------------
        //constructor with default capacity
        public ArrayStackChar() {
                data = new char[CAPACITY];
        }

        //constructor with a given capacity
        public ArrayStackChar(int capacity) {
                data = new char[capacity];
        }

        //current stack's size, not a capacity
        public int getSize() {
                return size;
        }

        //empty?
        public boolean isEmpty() {
                if (size == 0) {
                        return true;
                }

                return false;
        }

        //push
        public void push(char e) {
                if (size == data.length) {
                        resize();
                }
```

```
		data[size]=e;
		size++;
	}

	//peek
	public char peek(){
		if(size==0){
			return ' ';
		}

		return data[size-1];
	}

	//pop
	public char pop(){
		if(size==0){
			return ' ';
		}

		size--;
		return data[size];
	}

	//resize
	public void resize(){
		System.out.println("Double stack's size");
		//create a new array of double size
		char[] temp=new char[2 *data.length];
		//copy old array data to new array
		System.arraycopy(data, 0, temp, 0, data.length);
		//assign the stack to new array
		data=temp;
	}

}
```

Main Class Source code:

```
public class MainInfix2Postfix {

	public static void main(String[] args){
		String operator="+-*/";
		String expression="(a+b-c)*d-(e+f)";

		System.out.println("Infix: "+expression);

		//Create a char stack
		ArrayStackChar stack=new ArrayStackChar();
```

```java
        //split String to array of characters
        char[] exp = expression.toCharArray();

        System.out.print("\nPostfix: ");
        //For each character in the array
        for (char c : exp) {

            //When char c is not operator
            if (operator.indexOf(c) == -1) {

                //Check char c is left parenthesis '(' ?
                if (c == '(') {
                    stack.push(c);
                }

                //Check char is right parenthesis ')' ?
                else if (c == ')') {

                    //Pop item out until reach ')' or until stack is  empty
                    for (char e = stack.pop(); e != '('; e = stack.pop()) {

                        if (stack.isEmpty()) {   //Stack empty
                            System.out.println("expression is not matched");

                            break;
                        }

                        System.out.print(e);
                    }
                }

                //When char c is Alphabet
                else {
                    System.out.print(c);
                }
            }

            //When char c is operator
            else {

                //Check, is stack is empty or not ?
                if (stack.isEmpty()) {
                    stack.push(c);
                }
                else {
                    //Check top of stack
                    char check = stack.peek();

                    //If top is operator
                    if (operator.indexOf(check) != -1) {
                        //Pop the exist operator in stack and push new
operator to stack.
                        char p = stack.pop();
                        System.out.print(p);
                        stack.push(c);
                    }

                    //If top is left parenthesis '('
                    else if(check == '('){
                        stack.push(c);
                    }
                }
            }

        } //end for each character

        //now all operators and operands are checked
        //is there any operator left in stack?
        //if stack is NOT empty
        if(!stack.isEmpty()){

            //Top is left parenthesis '('
            if(stack.peek() == '('){
                System.out.print("expression is not matched");
            }

            //Top is operator
            else {
                char p = stack.pop();
                System.out.print(p);
            }

        } //end if stack is NOT empty

    }//end main method

}//end class
```

Question: What is the maximum number of symbols that will be pushed to the stack AT ONE TIME during the conversion of this expression? What are they?

The maximum number of symbols that will be pushed to the stack is 3. They are – (+