# Data Structure Assignment 3

## Paper Homework

(Textbook p.136 Exercise 6)

(a) Write the prefix form of the expressions in Excercise1.

(a) a * b * c

(b) -a + b - c + d

(c) a * -b + c

(d) (a + b) * d + e / (f + a * d) + c

(e) a && b || c || !(e > f)     (assuming C precedence)

(f) !(a && !( (b<c) || (c > d))) || (c < e)

(Just write down the final answer)

## General Information

- Deadline : 2018/11/2 (Please submit to TA after class)
- Late homework will not be accepted.
- Please write on **A4** papers, if there is more than one page, staple them together, and write your student id & name on each page.
- Any copies will be scored as zero. Do not plagiarize

## Programming Homework1

(textbook p.119 Exercise 4)

4. A *double-ended queue(deque)* is a linear list in which additions and deletions may be made at either end. Obtain a data representation mapping a *deque* into a one-dimensional array. Write functions that add and delete elements from either end of the deque.

The input consists of several lines. Each line consists of an instruction (and an integer number n).

- push_front ***n*** : add an element ***n*** from front of *deque*.
- push_back ***n*** : add an element ***n*** from back of *deque*.
- pop_front : delete an element from front of *deque*. push_back 3
- pop_back : delete an element from back of *deque*.
- show : print out all the elements in *deque* from front to back.

The number of instructions will not exceed 50.

**Note : You can not use c++ STL <deque> to finish your homework!**

### Input :

```
push_front 1
push_front 2
push_back 3
pop_front
push_back 4
pop_back
show
```

### Output :

```
1 3
```

---

| **Instruction** | | **deque** | | |
|---|---|---|---|---|
| push_front 1 | -> | (1) | | |
| push_front 2 | -> | (2 1) | | |
| push_back 3 | -> | (2 1 3) | | |
| pop_front | -> | (~~2~~ 1 3) | -> | (1 3) |
| push_back 4 | -> | (1 3 4) | | |
| pop_back | -> | (1 3 ~~4~~) | -> | (1 3) |

## Programming Homework2

(textbook p.136 Exercise 6)

6. Another expression form that is easy to evaluate and is parenthesis-free is known as prefix. In prefix notation, operators precede their operands. Figure 3.17 shows several infix expressions and their prefix equivalents. Notice that the order of operands is the same in infix and prefix

| **Infix** | **Prefix** |
|---|---|
| a*b/c | /*abc |
| a/b-c+d*e-a*c | -+-/abc*de*ac |
| a*(b+c)/d-g | -/*a+bcdg |

**Figure 3.17**: Infix and prefix expressions

(b) Write a C/C++ function that evaluates a prefix expression, *expr*. (Hint: Scan *expr* from right to left) **(Save as *eval.c/eval.cpp*)**

(c) Write a C/C++ function that transform an infix expression, *expr*, into its prefix expression. **(Save as *prefix.c/prefix.cpp*)**

***eval.c/eval.cpp***

**Input :**

```
/ * 2 3 6
- + - / 9 3 4 * 2 5 * 7 8
- / * 10 + 5 3 8 2
```

**Output :**

```
1
-47
8
```

***prefix.c/prefix.cpp***

**Input :**

```
2 * 3 / 6
9 / 3 - 4 + 2 * 5 - 7 * 8
10 * ( 5 + 3 ) / 8 - 2
```

**Output :**

```
/ * 2 3 6
- + - / 9 3 4 * 2 5 * 7 8
- / * 10 + 5 3 8 2
```

**General Information**

- Deadline : 2018/11/16 23:55.
- Upload your assignment to Moodle system.
- Upload file format : “student-ID_Name.rar” or “student-ID_Name.zip”
  Ex. “F12345678_王小明.rar”
- Your file should consist of the following items : Source Code & Readme file(Program description)
- Late homework will not be accepted.
- Any copies will be scored as zero. Do not plagiarize.