CSE213L: Data Structures and Algorithms
LNMIIT, Jaipur
Training Set 01

We plan to organise each training set as a set of 3 tasks. You will be required to demonstrate each task during the lab session and get the marks recorded and signed with date and time in your lab record book by your tutor immediately for each task is completed. Each student is required to maintain a lab notebook to record their progress in ADT Lab.

In any lab session of 2 hours only 3 task completions will be recorded. Training set 01 is, however, different in this respect. If your session does not have a tutor, please get the work recorded by your instructor when you have your first "attendance" meeting with your instructor in a later week.

Typically, three tasks in a training set will match the maximum amount of progress we record in a single lab session. This recording may comprise tasks from two separate training sets. For example, a student may complete remaining work from a previous training in a lab before starting a new training set.

As in CP, students will progress on the training sets based on their abilities and other needs. Not everyone in the class need to be working on the same training set in a lab session. Student's learning needs determine what they do in the lab sessions.

Attendance meeting is in addition to the physically-present attendance that we record (as a tick in a register) in the lab for those present. It is a short discussion session with the instructor perhaps once every 2 or three week. In the meeting instructor will discuss your progress and preparedness for the works in the lab. They will also see and sign your lab record notebook in which you write details of your activities, programs and outcomes as you work in the lab, make preparations for work in the lab before coming to the lab, and what you did to solve the difficulties you noticed in the lab session.

The physical attendance will be marked each session. However, the meeting with the instructor may not occur each week as it requires a longer discussion with the instructor.

To summarise:

1. Each training is made of three tasks.
2. Lab tutors record, completion of each task separately.
3. No more than three task completions are recorded a lab session.
4. Physical attendance of the students will be marked in each session.
5. There are additional records maintained to ensure that the students are coming to the labs well prepared and also they build on the lessons practicals done in the lab.
6. Students will progress in the lab training at a pace that suits their learning needs the best.
7. Interviews (marked as Attendance in CIF Assessments section) with the instructors will be used to advice students if it will help the student to repeat some training lessons.

Some slides from DSA textbook are uploaded here for your reference. You also need a C book as the program requires you to learn new topics in C.

Training set 01: Task 01

In the tasks below, you will be reading inputs that may be an `int` value or a character. To help you perform this reading, you must read about `stdio` functions `sprintf()` and `sscanf()`.

The information is available on pages 245 and 246 of K&R ANSI C book. A link to the book is provided to you on this classroom site.

Write a program that reads an input line containing an arithmetic expression into an array. You write a function that returns a `struct` indicating if the next token in the expression is an integer value, an operator, a left parenthesis, a right parenthesis or the end of expression. Use `#define` to create symbolic names for constants (See page 14 in K&R). Other member of the `struct` will provide details for integer value or operator letter.

In your demonstration use the function to print the input expression with one output line for each token. See the last page of this document for coding hints.

Training Set 01: Task 02

1. Write a C program to evaluate a postfix expression that is typed as input through input stream `stdin`. See textbook for the algorithm if needed (Thareja: Figure 7.23)
2. In doing so you must implement Stack ADT interface functions: `push()`, `pop()`, `peek()`, `isEmpty()`, `isFull()`, `initStack()`.
3. Test your program with several inputs.
    a. 10 20 + 30 *
    b. 20 10 – 5 /
    c. 25 10 %

Training Set 01: Task 03

1. Copy your solution for Task 02 as a separate program that you will modify in this task. As you make these modifications your code for ADT interface functions is not likely to change at all. In fact since char is a sub-type for `int`, you may not require any change in the above listed functions.
2. Write program to convert infix expression to postfix notation.
3. Test your solutions with many expressions with nested parentheses expressions. Your output from Task 03 should provide input for Task 02 programs.

Training Set 01: Task 04 (Optional but Useful and Easy)

1. Use compilation command `gcc` with `-o` options to create two executables `task02` and `task03` as follows:

```
gcc task02.c -lm -o task02

gcc task03.c -lm -o task03
```

Command that feeds the output of task03 into task02 for evaluation is as follws:

```
./task03 | ./task02
```

You may even to one more interesting change. Create a text file with infix expression. Let us call this file `expression`. You may run your expression evaluator:

```
cat expression | ./task03 | ./task02
```

Input/Output of the program

(0 before an integer indicates that data is an operand)

```
Input expression: 12+ 2*(3*( 24+ 5))
0  12
+
0  2
*
(
0  3
*
(
0  24
+
0  5
)
)
```

Token

Token for an expression is a string of one or more characters that is meaningful single item. In the present problem these items are operands (integer values), parentheses, and operators

Partial implementation (Some parts have been removed)

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define LPAR '('
#define RPAR ')'
#define PLUS '+'
#define MULT '*'
#define MOD '%'
#define FINISH '\0'
#define INT '0'

struct token {
     char kind;
     int value;
};
```

```
char expr[100];
int where = 0;

void skipWhite() {
     while (isspace(expr[where])) where++;
}

void skipDigits() {
     while (isdigit(expr[where])) where++;
}

struct token getNextToken() {
          struct token token;
          skipWhite();
           … … CODE REMOVED … … …
          return token;
}

int main(void) {
     struct token token;
     int i;

     printf("Input expression:");
     fgets(expr, 99, stdin);

     token = getNextToken();

     while (token.kind!=FINISH)
     {
          printf("%c", token.kind);
          if (token.kind == INT)
               printf("  %d", token.value);
          printf("\n");
          token = getNextToken();
     }

     return 0;
}
```