Introduction to Data Structures

Jinkyu Lee

Dept. of Computer Science and Engineering,
Sungkyunkwan University (SKKU)

Homework 2A

- 40 points for coding evaluation
  - Submission format
    - File name: yourid_HW2A.c
      - Example: 2000123456_HW2A.c
    - File type: .c (NOT .cpp)
  - Submission site: https://icampus.skku.edu
    - Week 5: [Homework] 2A (code)

- 1 point for report
  - The report is not evaluated in detail but evaluated as Pass/Fail
  - Template: Homework Report Template.docx
  - Submission format
    - File name: yourid_HW2A.pdf
      - Example: 2000123456_HW2A.pdf
    - File type: .pdf
  - Submission site: https://icampus.skku.edu
    - Week 5: [Homework] 2A (report)

- Due date
  - 10/13 23:59 (no late submission accepted)

Rules for homework

- You should follow instructions.
  - Complier
    - You will get no/less point if your program cannot be complied with the specified complier
  - Input/output format
    - You will get no/less point if TA's automatic evaluation program cannot parse your input or output.
  - Permitted modification scope
    - You will get no/less point if you modify code outside of the permitted modification scope
  - All other rules
    - You will get severe penalty or no/less point if you violate the given rules.

Complier and input/output rules for homework

- Every implementation homework will be evaluated by TA’s automatic evaluation program with the following complier.
  - Complier: GCC 7.X, 8.X, 9.X or 10.X
    - https://gcc.gnu.org/
  - You will get no/less point if your program cannot be complied with GCC 7.X, 8.X, 9.X or 10.X.
    - For example, do not rely on visual studio.
  - You can use standard library such as *stdlib.h* and *math.h*.

- Input/output format
  - You will get no/less point if TA’s automatic evaluation program cannot parse your input or output according to the following rules.
  - Use `stdin` and `stdout`

Problem

- Problem: Conversion of Infix to Postfix, and its evaluation of 2-digit hexadecimal (16-ary) numbers.
  - A 2-digit hexadecimal number is expressed as 00, 01, 02, … 09, 0A, 0B, 0C, 0D, 0E, 0F, 10, 11, 12, …, 1A, 1B, 1C, 1D, 1E, 1F, …., F0, F1, … F9, FA, FB, FC, FD, FE, or FF.
    - Be careful that each character is either 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E or F. (i.e., decimal numbers and upper-case letters A-E).
  - There are four operators: +, -, *, and /.
  - For given Infix expression of 2-digit hexadecimal numbers, you should (i) convert it to Postfix expression, and (ii) evaluate the Postfix expression.
  - The length of input is at most 50 characters without any blank.

Problem

- After calculating a sub-expression, the result should be expressed as positive (or zero) 2-digit hexadecimal numbers. You don’t need to care for a number larger than FF or smaller than 00.
  - e.g., there is no input like 99 * 99, which is larger than FF.
  - e.g., there is no input like A0-B7, which is smaller than 00.
- For division, you only care for the quotient.
  - e.g., B / 3 = 3, not 3.66.

Input/Output

- Input (no space)

```
2A+05*1A–0F
```

- Output (no space)

```
2A051A*+0F-
9D
```

Template

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

#define MAX_STACK 100

typedef enum {false , true} bool;

typedef struct {
    char small;
    char large;
    // *Variable "operator"
    // used in ConvInfixToPostfix function
    char operator;
}Hexa_num;

typedef struct {
    Hexa_num items[MAX_STACK];
    int top;
}Stack;

/* Modify from here */
// if you need any user-defined function
/* Modify to here */

Hexa_num add(Hexa_num b1,Hexa_num b2);
Hexa_num subtract(Hexa_num b1,Hexa_num b2);
Hexa_num multiply(Hexa_num b1,Hexa_num b2);
Hexa_num division(Hexa_num b1,Hexa_num b2);
```

```
void InitStack (Stack *pstack);
bool IsFull(Stack *pstack);
bool IsEmpty(Stack *pstack);
Hexa_num Peek(Stack *pstack);
void Push(Stack *pstack,Hexa_num item);
void Pop(Stack *pstack);
void ConvInfixToPostfix(char* exp,char* convExp, int
len);
Hexa_num EvalPostfix(char* exp,int len);
void print_Hexa_num(Hexa_num result);



int main() {
    char infix_exp[100];
    char postfix_exp[100];
    Hexa_num result;

    scanf("%s",infix_exp);


ConvInfixToPostfix(infix_exp,postfix_exp,strlen(infi
x_exp));
    printf("%s\n%s\n",infix_exp,postfix_exp);
    result =
EvalPostfix(postfix_exp,strlen(postfix_exp));
    print_Hexa_num(result);
    return 0;
}
/* Modify from here */
// implementation of functions
/* Modify to here */
```

Template

- You cannot modify the template except the space between /*Modify from here*/ and /*Modify to here*/
  - Do not remove /*Modify from here*/ and /*Modify to here*/
  - TA will copy the space and evaluate your code.
  - In the space, you need to implement the following functions.
  - You may add user-defined functions.

```
Hexa_num add(Hexa_num b1,Hexa_num b2);
Hexa_num subtract(Hexa_num b1,Hexa_num b2);
Hexa_num multiply(Hexa_num b1,Hexa_num b2);
Hexa_num division(Hexa_num b1,Hexa_num b2);

void InitStack (Stack *pstack);
bool IsFull(Stack *pstack);
bool IsEmpty(Stack *pstack);
Hexa_num Peek(Stack *pstack);
void Push(Stack *pstack,Hexa_num item);
void Pop(Stack *pstack);
void ConvInfixToPostfix(char* exp,char* convExp, int len);
Hexa_num EvalPostfix(char* exp,int len);
void print_Hexa_num(Hexa_num result);
```

Evaluation

- Evaluation
  - TA will test several cases.
  - Read Pages 8~9 (regarding template) carefully.
  - For each test case,
    - If your C code results in an answer within 10 seconds on a platform with average computing power,
      - If your output is perfect for both postfix expression and evaluation result,
        - You get 100%.
      - Else if your output is perfect only for postfix expression,
        - You get 50%.
      - Else if your output is perfect only for evaluation result,
        - You get 50%.
      - Else,
        - You get 0%.
    - Else,
      - You get 0%.