CSE213L: DSA Lab

LNMIIT, Jaipur

Training Set 03

The training set aims to train students on several basic but very useful programming concepts. These concepts to be introduced through this training set include:

1. Abstract Data Type (ADT) and interface functions,
2. Linked list data-structures,
3. Multi-file C programs (this was also used in Training Set 02)
4. Use of `assert()` declarations as part of defensive programming paradigm

An initial (but incomplete) implementation of a program is included at the end of this document. The given program is made of three separate files that must be placed in a single directory. In this training set, the student will test a C program for correct and balanced use of parentheses, brackets and braces pairs in the program.

Following commands are used to compile the given C program that is divided over multiple files:

```
>gcc main.c deque.c
```

Or

```
>gcc *.c
```

Training Set 03: Task 01

As Task 01 complete the implantations of all functions in file `deque.c`.

Add `assert()` checks in your codes as precaution against errors and mistakes. To help you the importance of the `assert()` declarations, the code provided to you has an error that is reported by a failed `assert()` declaration.

Tutor will ask you to show at least one `assert()` that is not simply an minor variation of an `assert()` declaration that is already in the provided code. Test your code before getting it marked by a tutor.

Training Set 03: Task 02

Choose one of your largest C programs. Make a copy of the program. Linux command for making copy of file is:

```
cp yourProg.c copiedProgram.c
```

As the primary task, write a C program that reads program `copiedProg.c` from its file, one char at a time and prints the read character as output on stream `stdout`. Revise your lessons from CP/CPL if you need more information on file handling. Be sure to open your file in read mode ("`r`"). Do not use a `stdio` function that has been deprecated (you compiler will caution you against such errors.)

Your program output should look like what the following command prints on your computer monitor

```
cat copiedProg.c
```

Once completed, get your task marked by your tutor.

Training Set 03: Task 03

In this task you will use ADT interface `deque.h` and your implementation of its functions in file `deque.c` in Task 01as a stack.

Modify your function(s) in Task 02 to stop printing the read program on screen. Instead it should push occurrences of left pairs of parentheses, brackets and braces into the stack. These will be removed when matching right pair is read from the program.

That is, each time a left symbol arrives in the input stream, it is pushed into the stack. When a right symbol is noted in the input stream, it should match the top symbol on stack. The top symbol is removed from the stack. If the top symbol in the stack does not match the right symbol read from the input an mis-matched symbol error is revealed.

Your program should finally report if your program was well-balanced or not in respect to parentheses, brackets and braces or not. Conditions indicating ill-balanced program are (any of the following marks the program as ill-balanced).

1. Right pair not matching the symbol removed from the stack.
2. A right pair arrives when stack is empty.
3. Stack not being empty when the program reading ends.

Vishu Malhotra
29 April 2020
The LNMIIT, Jaipur 302031.

File main.c

```
#include <stdio.h>
#include "deque.h"

int main(int argc, char *argv[]) {
        joinL(20);
        joinL(200);
        printf("%d\n", size());
        printf("%d \n", leaveR());
        printf("%d\n", size());
        printf("%d \n", leaveR());
        printf("%d\n", size());
        return 0;
}
```

File deque.h

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

struct node {
        int data;
        struct node *nextL;
        struct node *nextR;
};

/* Defiend in some othert file */
extern struct node hdr;

/* ADT interface functions */
void init();

void joinL(int d);
void joinR(int d);

int leaveL();
int leaveR();

int size();
```

File deque.c

```
#include "deque.h"

/* storage allocated here */
struct node hdr;

/* ADT interface functions */
/* THink of hdr as if it is on top
of all member nodes of deque.
Left and right of header is not symmetric to
those of member nodes */
void init() {
        // unused
        hdr.data = 0;
```

```
        hdr.nextL = hdr.nextR = NULL;
}

void joinL(int d) {
        printf("Going to join %d on left\n", d);
        struct node *new = malloc(sizeof(struct node));
        assert(new!=NULL); // Stop if problem

        new->data = d;

        if (hdr.nextL == NULL) {
                assert(hdr.nextR == NULL);
                hdr.nextL = hdr.nextR = new;
                new->nextL = new->nextR = NULL;
                printf("Joined %d on left\n", d);
                return;
        }

        assert(hdr.nextR != NULL);
        assert(hdr.nextL->nextL == NULL);
        hdr.nextL->nextL = new;
        new->nextR = hdr.nextL;
        new->nextL = NULL;
        hdr.nextL = new;
        printf("Joined %d on left\n", d);
}

void joinR(int d) {
        return;
}

int leaveL() {
        // Unimplemented
        return 0;
}
int leaveR() {
        struct node *tmp;
        printf("Someone leaving from Right\n");
        assert(hdr.nextL != NULL && hdr.nextR != NULL);
        int d = hdr.nextR->data;
        tmp = hdr.nextR->nextL;
        if (tmp != NULL)
                tmp->nextR = NULL;
        free(hdr.nextR);
        hdr.nextR = tmp;
        printf("From right %d left\n", d);

        assert (tmp != NULL || hdr.nextL == NULL);
        return d;
}

int size() {
        int i = 0;
        struct node *ptr = hdr.nextL;

        while (ptr != NULL) {
                i++; ptr = ptr->nextR;
        }
        return i;
}
```