CSE213L: DSA Lab

LNMIIT, Jaipur

Training Set 03

See DSA theory (classroom.google.com with code wonhzxk) for a partial implementation of a deque ADT using linked list data-structure. In this training set, students will use a copy of one of their C programs for correct and balanced use of pairs of parentheses, brackets and braces.

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. Tutor will ask you to show at least one `assert()` that is not simply an version of those that are 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 is:

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

Write a C program that reads program `copiedProg.c` from 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`").

Your 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` to use your implementation of deque 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.

Your program should finally report if your program was well balanced 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.

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);
        if (tmp == NULL)
                hdr.nextL = NULL;
        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;
}
```