CSE213L: DSA Lab
LNMIIT, Rupa Ki Nangal
Training Set 04

In this training set, students will be guided to write a simple program for a game which resembles a game of musical chairs. The primary data-structured used in the program will be a doubly linked list. For details of data-structure ADT interface functions and their implementations, please refer to Section *6.5: Circular Doubly Linked List* in textbook by R Thareja.

However, the interface we need is somewhat different from the one discussed in DSA classes and the textbook. As has been instructed in the previous DSA Lab training sets, students will be required to implement the data-structure and the application program in separate files: `main.c` and `dlist.c`. The interface functions and other macros will be declared in file `dlist.h`. This file is listed here to help novice student programmers.

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

struct chair {
	struct chair *leftChair;
	char *player;
	struct chair *rightChair;
};

/*	Free all memory that is accessible to dlist
	from pointer marking its start. This includes memory
	with struct as well as with string player
*/
void initDList();

/*	Create a new chair for a player. Add it to the
	list. Terminate prog if memory not available
*/
void insertChair(char *player);

/*	Free person name string and chair too.
	Count the fisrt pointed chair as numbered 0.
	Displacement can be positive or negative from
	chair 0. Player pointer should not be NULL.
*/
char *removeChair(int displacement);

/*	Return name of the person in the chair */
char *whoInChair(int displacement);

/* Is there any chair left in the game? */
int isEmpty();
```

Please see the end of this document for an output produced by an implementation of this game.

Training Set 04: Task 01

Create a text file with names of 20 friends. Each name is a different name. Type one name in each line of the file. Let us call the file `players`. These names will be read from the file and inserted into a circular list that you will write in this training set. As this is a game like a game of musical chairs, nodes in the list are tagged as `chair`. Some snippets of code from file `main.c` are listed below.

Once the names have been read from file `players` into the list, the game can be played. The game will be played by rolling a dice. You already know from your childhood days that a roll of a dice returns a number from 1 to 6 with equal probability.

In each phase of this game, a dice will be rolled, and a player selected to be sacrificed and removed from the game. This player is called victim. The victim is selected as follows: The chair pointed to by the primary reference of the doubly linked list is chair 0. Rolled number is denotes the victims chair number. If the value rolled value is an odd integer, the victim is chosen by counting the rolled value using the left links starting from chair 0. Otherwise, the right link is used to count chairs. The victim is removed from the list and victim's name printed on the standard output.

Carefully read file `main.c` below and populate the bodies of `dlink.c` with functions to let the program compile correctly. The minimal implementation code needed in a function is usually the main block of the function with either no code or with a single return statement. For example,

```
char *whoInChair(int displacement) {
      return "None found";
}
```

Next, implement functions `isInit()` and `isEmpty()` after defining global variables for the starting chair in the data-structure. My program has variable declarations: `struct chair *pointAtChair0;`

My code from file `main.c` is copied below for students to read and use to start their program development.

```
#include <stdio.h>
#include <stdlib.h>
#include "dlink.h"

char tmp[101];

int rollDice() {
      int i = 1+rand()%6;
      return i;
}

int main(void) {
      char *name;
      int dice;

      initDList();
      FILE *playerF = fopen("players","r");
      assert(player != NULL); // File is open
```

```
	fgets(tmp, 100, playerF);
	while (!feof(playerF)) {
		// Get space filled with \0
		name = calloc(1,strlen(tmp)+1);
		// Newline should not copied
		strncpy(name, tmp, strlen(tmp)-1);
		insertChair(name);
		fgets(tmp, 100, playerF);
	}

	printf("\nAll palyers are in their chairs now. Game begins\n\n");

	while (!isEmpty()) {
		printf("%s is in chair 0. ", whoInChair(0));
		dice = rollDice();
		if (dice%2 == 1) // Odd numbers on left!
		dice *= -1;
		printf("Rid chair %d. ", dice);
		name = removeChair(dice);
		if (isEmpty())
			printf("\n\nWinner is %s\n", name);
		else
			printf("\nLosing player was %s.\n", name);
		free(name);
	}
	return 0;
}
```

Once you have set up the program and have the two interface functions named above working you may get this task marked by your tutor.

Training Set 04: Task 02

To complete this task, the students must implement all ADT interface functions except for function `char *removeChair(int displacement);` This function is a little awkward to implement and is put on hold for a later task (Task 03).

To prove the claim that your interface functions in file `dlink.c` are working correctly, you may modify code in file `main.c` and use ADT functions you have implemented to list some player names from the doubly linked list.

Include `assert()` statements in our code to catch and avoid mistakes and errors. Program without the defensive programming checks will be very difficult to develop. DO NOT BE LAZY.

Training Set 03: Task 03

Now is the time to implement a challenging task. It is function `char *removeChair(int displacement);` Study output of a game given below to identify different situations in which a chair could be removed from the list.

The best advice that can be given to the students is to identify each of these situations and keep solution code for each case separate. Use `assert()` declarations to catch mistakes early.

Some situations that the students are advised to note carefully before begin coding the function are listed below:

1. Removal of the last chair from the list
2. Removal of the second last chair from the list.
3. Removal (and identification) of the chair pointed to by the list's start pointer `pointAtChair0;`
4. Removal of other chairs.

```
Adding Ram Mohan
Adding Sita
Adding Bharat Kumar
Adding John
Adding Charles
Adding Reeta
Adding Geeta
Adding Radha
Adding Najma
Adding Raman
Adding Mohammad
Adding Pooja
Adding Achla

All players are in their chairs now. Game begins


Ram Mohan is in chair 0 now.
        Rid chair 6.    Freeing a chair; at least 2 chairs left.
        Losing player was Geeta.
Ram Mohan is in chair 0 now.
        Rid chair 6.    Freeing a chair; at least 2 chairs left.
        Losing player was Radha.
Ram Mohan is in chair 0 now.
        Rid chair -5.   Freeing a chair; at least 2 chairs left.
        Losing player was Najma.
Ram Mohan is in chair 0 now.
        Rid chair -5.   Freeing a chair; at least 2 chairs left.
        Losing player was Reeta.
Ram Mohan is in chair 0 now.
        Rid chair 6.    Freeing a chair; at least 2 chairs left.
        Losing player was Mohammad.
Ram Mohan is in chair 0 now.
        Rid chair -5.   Freeing a chair; at least 2 chairs left.
        Losing player was John.
Ram Mohan is in chair 0 now.
        Rid chair -1.   Freeing a chair; at least 2 chairs left.
        Losing player was Achla.
Ram Mohan is in chair 0 now.
        Rid chair -1.   Freeing a chair; at least 2 chairs left.
        Losing player was Pooja.
Ram Mohan is in chair 0 now.
        Rid chair -5. Freeing chair 0.
        Losing player was Ram Mohan.
Raman is in chair 0 now.
        Rid chair -3.   Freeing a chair; at least 2 chairs left.
        Losing player was Sita.
Raman is in chair 0 now.
        Rid chair 6. Freeing chair 0.
        Losing player was Raman.
Bharat Kumar is in chair 0 now.
        Rid chair 6. Freeing second last chair.
        Losing player was Bharat Kumar.
Charles is in chair 0 now.
        Rid chair 2. Freeing the last chair.

Winner is Charles
```