CS 137: Assignment #10

Due on Tuesday, Dec 5, 2023, at 11:59 PM

Submit all programs using the Marmoset Submission and Testing Server located at
https://marmoset.student.cs.uwaterloo.ca/

Victoria Sakhnini

Fall 2023

Notes:

- Use the examples to guide your formatting for your output. Remember to terminate your output with a newline character.
- For this assignment, you may use any content covered until the end of Module 13.
- <math.h> is not allowed.
- Your solution may not pass marmoset tests for some of the tests due to the wrong implementation or a memory leak. My advice is that even if you pass your own tests and are very positive about your implementation. Use Valgrind to detect memory leaks before submitting your solutions. gcc won't detect memory leaks, and maybe your local compiler won't catch them, either.

Problem 1

Consider the following definition:

```
typedef struct dlnode {
        int data;
        struct dlnode *next;
         struct dlnode *prev;

} dlnode;

typedef struct ndlst {
        dlnode *head;  //points to the first element; otherwise, NULL
         dlnode *tail;  //points to the last element; otherwise, NULL
         int  len;      // number of elements in the list
} ndlst;
```

to define a doubly linked list where each node points to the previous and to the next nodes if exist; otherwise, it points to NULL. Also, the head points to the first element of the list, and the tail points to the last element in the list to make working with this list more efficient (such as when adding/removing from the front/end of the list).

Create a C program `doublyList.c` that completes the following functions (The first one was completed for you):

```
struct ndlst *dlistCreate(void){
     ndlst *ret = malloc(sizeof(ndlst));
     ret->head = NULL;
     ret->tail = NULL;
     ret->len = 0;
     return ret;
}

void dlistDestroy(ndlst *lst){ }
```

//returns the number of elements in the list

```
int dlistLength(ndlst *lst){  }
```

//Pre: `lst` is a valid list with a length of at least `n` where `n>=1`

//Post: `n`th item removed

```
void dlistRemoveElem(ndlst *lst, int n){  }
```

// `lst` is a valid list

```
void dlistAddToFront(ndlst *lst, int elem){  }
```

// `lst` is a valid list

```
void dlistAddToEnd(ndlst *lst, int elem){  }
```

`// lst` is a valid list, to be sorted in ascending order

```
void dlistSort(ndlst *lst){  }
```

// rotate the list `n` times to the left, assume `lst` not empty

```
void dlistRotateLeft(ndlst *lst, int n){
```

// rotate the list `n` times to the right, assume `lst` not empty

```
void dlistRotateRight(ndlst *lst, int n)
```

You are to submit this file containing all implemented functions (that is, you must delete the test cases portion/`main` function, as well as the two print functions provided below.). However, **<u>you must keep the required included libraries and structure definitions.</u>**

The sample code for testing is below. The **<u>two</u>** provided print functions are beneficial to check that you are building the list correctly with the correct links for `next` and `prev` pointers.

```
void dlistPrint(ndlst *lst)  // I am using this function to print
                             // results but don't submit this function
{
    dlnode *node = lst->head;
    for (; node; node = node->next)
            printf("%d  ",node->data);
    printf("\n");
}

void dlistPrintReverse(ndlst *lst)   // I am using this function to
                         // print results but don't submit this function
{
    dlnode *node = lst->tail;
    for (; node; node = node->prev)
            printf("%d  ",node->data);
    printf("\n");
}

int main(void)
{
        ndlst *lst1 = dlistCreate();
        assert(dlistLength(lst1) == lst1->len);
        assert(dlistLength(lst1) == 0);
        dlistAddToEnd(lst1, 10);
        dlistAddToFront(lst1, -20);
        dlistAddToFront(lst1, 0);
        dlistAddToEnd(lst1, 9);
        dlistAddToFront(lst1, -9);
        dlistAddToFront(lst1, 7);
        dlistAddToEnd(lst1, 40);
        assert(dlistLength(lst1) == lst1->len);
        assert(dlistLength(lst1) == 7);
        dlistPrint(lst1);
        dlistPrintReverse(lst1);
        dlistRemoveElem(lst1, 5);
        dlistRemoveElem(lst1, 1);
```

```c
        dlistRemoveElem(lst1, 5);
        dlistAddToEnd(lst1, 100);
        assert(dlistLength(lst1) == lst1->len);
        assert(dlistLength(lst1) == 5);
        dlistPrint(lst1);
        dlistPrintReverse(lst1);
        dlistSort(lst1);
        dlistPrint(lst1);
        dlistRotateLeft(lst1, 2);
        dlistPrint(lst1);
        dlistRotateRight(lst1, 3);
        dlistPrint(lst1);
        dlistDestroy(lst1);
        return 0;
}

```

The output of the code above:

```
7  -9  0  -20  10  9  40

40  9  10  -20  0  -9  7

-9  0  -20  9  100

100  9  -20  0  -9

-20  -9  0  9  100

0  9  100  -20  -9

100  -20  -9  0  9
```

Problem 2

Duck-Duck-Goose is a game where a group of people sit around in a circle while one person walks around the circle at a time.

- Let's call the person walking around the circle `P` As `P` passes people sitting down, they will tap on the sitting person's head and say either "duck" or "goose".
- If `P` says "duck", nothing will happen, and they will continue walking.
- If `P` says goose, the sitting person will get up and chase `P` around the circle.
- If `P` is touched before running around the circle once, they must stay as `P`.
- Otherwise, the previously sitting person becomes `P`.
- In this alternative of the game, the loser will instead be eliminated from playing, and the winner will become `P`.

For this question, you will simulate a game of duck-duck-goose.

The struct to represent a person is as follows:

```
typedef struct Person {
    int id;
    int speed;
    struct Person* next;
}Person;
```

Create a program that takes in as input lines...

```
    person1 speed1
    person2 speed2
    .
    .
    .

    personn speedn
```

where `personn` is a unique integer ID to represent the nth person and `speedn` is a positive integer to represent the speed of the nth person. Construct a link list of people in the order they are given that forms a ring such that the last person points to the first person in the list.

`P` will start off as the first person to read in from standard input. After reading in all participants, the program should do the following:

I) `P` will call `n` people "duck" and the `n + 1`th person goose where `n` is the `id` of `P`. As `P` goes from person to person, print either

```
c duck d
```

or

```
c goose! d
```

Where `c` is the `id` of `P` and where `d` is the `id` of the sitting person `P` passes.

`II)` When `P` calls goose, the winner is the person with the highest speed value.

If the speeds are the same, the winner is the "goosed" person. That person will become the new `P,` and the loser will leave the circle. Continue playing duck-duck-goose, moving on to the person after the winner until one person is left.

`III)` At the end, print

```
winner! d
```

where `d` is the `id` of the winner.

Example Input1

```
1 2

2 4

3 3

4 1

?
```

Example Output1

```
1 duck 2

1 goose! 3

3 duck 4

3 duck 2

3 duck 4

3 goose! 2

2 duck 4

2 duck 4

2 goose! 4

winner! 2
```

Example Input2

```
1 2

2 3

3 4

4 5

5 6

6 7

?
```

Example Output2

```
1 duck 2

1 goose! 3

3 duck 4

3 duck 5

3 duck 6

3 goose! 2

3 duck 4

3 duck 5

3 duck 6

3 goose! 4

4 duck 5

4 duck 6

4 duck 5

4 duck 6

4 goose! 5

5 duck 6

5 duck 6

5 duck 6

5 duck 6

5 duck 6

5 goose! 6

winner! 6
```

Example Intput3

```
3 6
1 5
2 7
4 4
6 1
5 3
?
```

Example Output3

```
3 duck 1
3 duck 2
3 duck 4
3 goose! 6
3 duck 1
3 duck 2
3 duck 4
3 goose! 5
3 duck 1
3 duck 2
3 duck 4
3 goose! 1
3 duck 2
3 duck 4
3 duck 2
3 goose! 4
3 duck 2
3 duck 2
3 duck 2
3 goose! 2
winner! 2
```

Example Intput4

```
1 5

2 7

3 5

4 1

?
```

Example Output4

```
1 duck 2

1 goose! 3

3 duck 4

3 duck 2

3 duck 4

3 goose! 2

2 duck 4

2 duck 4

2 goose! 4

winner! 2
```

Create a C program `duckgoose.c` that completes the following functions :

```
Person updatePerson(int id, int speed);

void addPerson(Person** start, Person* new);

Person* play(Person* start)// frees any person who leaves the game.
```

You are to submit this file containing all implemented functions (that is, you must delete the test cases portion/`main` function). However, **you must keep the required included libraries and structure definitions.**

The provided main function to play the game:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct Person{
    int id;
    int speed;
    struct Person* next;
}Person;

Person updatePerson(int id, int speed) {
}

void addPerson(Person** start, Person* new) {
}
// Returns the winner
Person* play(Person* start) {
}

int main(){
    int amount;
    int p, s;
    // List of people
    Person* lop = NULL;
    // Read in participants
    while (scanf("%d %d", &p, &s) == 2){
        Person* np = (Person*)malloc(sizeof(Person));
        *np = updatePerson(p, s);
        addPerson(&lop, np);
    }
   Person* winner = play(lop);
    free(winner);
}

```