UNIVERSITY of GUELPH

CIS2520 Data Structures
Fall 2015, Assignment 3

A. Reversing a list using recursion

Download **A1key.zip**. *It packs two folders:* **List_Student_S** *and* **List_Student_L**.
This question concerns the files in **List_Student_L**.

1) Add the following function to the *Lists of Students* library. Implement it using recursion.

```
/*************************************************************************
 * FUNCTION NAME: Reverse
 * PURPOSE: Reverses a List
 *          (the first Item becomes the last and vice versa).
 * ARGUMENTS: The address of the List to be reversed (List *L)
*************************************************************************/
extern void Reverse (List *L);
```

2) Modify **myProgram.c** to test your implementation of **Reverse**. With the current test file **test.txt**, the output of the program should be as indicated below.

```
List is empty; list is not full; list is of size 0:
List is not empty; list is not full; list is of size 1:
      John  75%
List is not empty; list is not full; list is of size 2:
      John  75%
      Mary  80%
List is not empty; list is not full; list is of size 3:
      Pete  90%
      John  75%
      Mary  80%
```

```
List is not empty; list is not full; list is of size 4:
      Pete  90%
      Liz   85%
      John  75%
      Mary  80%
List is not empty; list is not full; list is of size 3:
      Pete  90%
      Liz   85%
      John  75%
List is not empty; list is not full; list is of size 2:
      Pete  90%
      John  75%
List is not empty; list is not full; list is of size 1:
      John  75%
List is empty; list is not full; list is of size 0:
```

B. Comparing the running times of sorting algorithms

Download **A1key.zip**. *It packs two folders:* **List_Student_S** *and* **List_Student_L**.
Create a copy **List_int_S** *of the folder* **List_Student_S**.
This question concerns the files in **List_int_S**.

1) Delete all the files except **ListType.h**, **ListInterface.h** and **ListImplementation.c**, and modify these three files to create a *Lists of Integers* library. The library should be able to handle relatively big lists.

2) Add the functions **BubbleSort1**, **BubbleSort2** and **MergeSort** (as described in Appendix) to your *Lists of Integers* library.

3) Write a program **test.c**. To run the program, one should type:

```
$ ./test.out
```

The program uses the function **clock** (see lecture notes, slide 5.4). It reads the data stored in **test.txt** (i.e., a list of integers) and it displays various results, as illustrated below.

test.txt

```
5 8 2 1 9
```

Output of the program when **test.txt** *is as described above:*

```
5 8 2 1 9 to sort
1 2 5 8 9 BubbleSort1 in <running_time>
1 2 5 8 9 BubbleSort2 in <running_time>
1 2 5 8 9 MergeSort in <running_time>
```

4) Write a program **sort.c**. To run the program, one should type:

```
$ ./sort.out <number_m_of_lists> <size_n_of_lists>
```

The program compares the three sorting algorithms. It considers m lists of size n. One list should be (1,2,3,…,n); one list should be (n,n−1,…2,1); each one of the other lists should be randomly populated. The program uses the functions **clock**, **rand**, **srand** and **time**. It measure the best, average and worst running times of **BubbleSort1**, **BubbleSort2** and **MergeSort**.

Output of the program:

```
BubbleSort1
<best running time>
<average running time>
<worst running time>
BubbleSort2
<best running time>
<average running time>
<worst running time>
MergeSort
<best running time>
<average running time>
<worst running time>
```

NOTE — Your folder **List_int_S** *should now contain the following text files:*
ListType.h, **ListInterface.h**, **ListImplementation.c**, **test.c**, **sort.c**,
a **makefile**, *and the test file* **test.txt** *as shown in question* **3)**.

C. Creating a *Queues of Students* library

Download **A1key.zip**. *It packs two folders:* **List_Student_S** *and* **List_Student_L**.
Create a copy **Queue_Student_S** *of the folder* **List_Student_S**.
This question concerns the files in **Queue_Student_S**.

1) Rename the files **ListType.h**, **ListInterface.h** and **ListImplementation.c**. Call them **QueueType.h**, **QueueInterface.h** and **QueueImplementation.c**, and modify them to create a *Queues of Students* library where queues are implemented using circular arrays. The function declarations in **QueueInterface.h** should be:

```
extern void Initialize (Queue *Q);
extern void Enqueue (Item I, Queue *Q);
extern void Dequeue (Queue *Q);
extern int Full (Queue *Q);
extern int Empty (Queue *Q);
extern int Size (Queue *Q);
extern void Head (Queue *Q, Item *I);
extern void Tail (Queue *Q, Item *I);
extern void Destroy (Queue *Q);
```

2) Update the **makefile**, and rewrite **test.txt** and **myProgram.c** according to the example below.

test.txt

```
Enqueue John 75
Enqueue Mary 80
Dequeue
Enqueue Pete 90
Enqueue Liz 85
Enqueue Bob 60
Dequeue
Dequeue
Dequeue
Dequeue
```

Output of the program when **test.txt** *is as described above:*

```
Queue is empty; queue is not full; queue is of size 0:
Queue is not empty; queue is not full; queue is of size 1:
      John  75%
Queue is not empty; queue is not full; queue is of size 2:
      John  75%
      Mary  80%
```

```
Queue is not empty; queue is not full; queue is of size 1:
      Mary  80%
Queue is not empty; queue is not full; queue is of size 2:
      Mary  80%
      Pete  90%
Queue is not empty; queue is not full; queue is of size 3:
      Mary  80%
      Pete  90%
      Liz   85%
Queue is not empty; queue is full; queue is of size 4:
      Mary  80%
      Pete  90%
      Liz   85%
      Bob   60%
Queue is not empty; queue is not full; queue is of size 3:
      Pete  90%
      Liz   85%
      Bob   60%
Queue is not empty; queue is not full; queue is of size 2:
      Liz   85%
      Bob   60%
Queue is not empty; queue is not full; queue is of size 1:
      Bob   60%
Queue is empty; queue is not full; queue is of size 0:
```

NOTE — Your folder **Queue_Student_S** *should now contain the following text files:* **QueueType.h**, **QueueInterface.h**, **QueueImplementation.c**, **StudentType.h**, **StudentInterface.h**, **StudentImplementation.c**, **myProgram.c**, **makefile**, *and the test file* **test.txt** *as shown in question* **2)**.

SUBMISSION

Make sure the revised folders **List_Student_L**, **List_int_S** and **Queue_Student_S** contain text files only (**.h**, **.c**, **makefile**, **test.txt**). Make sure all the file and function header comments have been updated according to the requested changes. Place the three folders along with a **README.txt** text file and the completed **Academic Integrity file** in a root folder **CIS2520_LastNameFirstName_A3**. Zip the root folder and upload it to *Moodle*. Check the course website for additional instructions.

MARKING SCHEME

A = 20% B = 50% C = 30%

APPENDIX

function BubbleSort1 (A)

```
for j=1 to A.length−1
    for i=1 to A.length−j
        if A[i−1] > A[i]
            swap A[i−1] and A[i]
```

function BubbleSort2 (A)

```
k = A.length
repeat
    swapped = false
    for i = 1 to k−1
        if A[i−1] > A[i]
            swap A[i−1] and A[i]
            swapped = true
    k = k−1
until not swapped
```

function MergeSort (A, first, last)

```
if first<last
    middle=(first+last)/2          // greatest integer less than or equal to (first+last)/2
    MergeSort(A,first,middle)
    MergeSort(A,middle+1,last)
    Merge(A,first,middle,last)
```

function Merge (A, first, middle, last)

```
for i=0 to middle−first
    L[i]=A[first+i]
L[middle−first+1]=+∞

for j=0 to last−middle−1
    R[j]=A[middle+j+1]
R[last−middle]=+∞

i=j=0
for k=first to last
    if L[i]≤R[j]
        A[k]=L[i]
        i=i+1
    else
        A[k]=R[j]
        j=j+1
```