Birla Institute of Technology & Science, Pilani
2nd Semester 2016-17 - CS F211 – Data Structures and Algorithms

Lab 2 - 29th Jan 2017
Topics – Abstract Data Types and Performance Measurements

1. Implementing Abstract Data Types in C

An Abstract Data Type is realized by separating the user concerns (types and interface) from provider concerns (implementation). In C,

- a header file (with suffix ".h") is used to define the data type and declare the functions / procedures operating on it
- a code file (with suffix ".c") is used to define the implementation (I.e. define the functions / procedures).

For instance, consider the ADT Linked List, and an insert operation over it.

We use linkedlist.h and linkedlistImpl.c to realize this ADT:

```
/* linkedlist.h */
typedef … *linkedList;
void  insert(linkedList);
…

/* linkedlistImpl.c */
void insert(linkedList list) {  return … ; }

…
```

Then, a (test) program that uses linkedList, say driver, can be implemented as a code file, say driver.c:

```
#include "linkedlist.h"
int main(int argc, char **argv)
{
  linkedList list;
  …
  insert(list, element);
   …
   return 0;
}
```

Problem 1: Use this approach to realize ADT DynamicList with operations *create*, *traverse*, *destroy* (i.e. deallocate everything) using linked list as the implementation. Driver code should read one of the following codes and call corresponding function (until -1 has been given as input).

0	create	Should read a set of integers from stdin (until -1 is given), insert them in DynamicList and call traverse function
1	traverse	Should print DynamicList in a line with a tab in between each element. End of the list (or empty list) should be represented by -2.

2	destroy	Should deallocate all nodes of DynamicList and call traverse function. In case of empty list it should print -3.

Test Case 1:

Input 2	Output 2
0 12 5 6 7 6 7 6 7 -1 1 2 -1	12 5 6 7 6 7 6 7 -2 12 5 6 7 6 7 6 7 -2 -2

Test Case 2:

Input 2	Output 2
0 1 -1 0 1 -1 0 3 -1 1 1 1 2 2 -1	1 -2 1 1 -2 1 1 3 -2 1 1 3 -2 1 1 3 -2 1 1 3 -2 -2 -3

Test Case 3:

Input 2	Output 2
0 12 5 1 6 2 -1 1 2 -1	12 5 1 6 2 -2 12 5 1 6 2 -2 -2

Test Case 4:

Input 2	Output 2
0 12 5 12 5 2 -1 2 1 1 2 -1	12 5 12 5 2 -2 -2 -2 -2 -3

Test Case 5:

Input 2	Output 2
1 2 1 0 12 5 1 6 2 -1 1 2 2 -1	-2 -3 -2 12 5 1 6 2 -2 12 5 1 6 2 -2 -2 -3

Test Case 6:

Input 2	Output 2
0 -2 -3 4 5 9 10000 100120 -1 0 12 -1 0 56 -1 0 12345 -1 1 0 5 -1 2	-2 -3 4 5 9 10000 100120 -2 -2 -3 4 5 9 10000 100120 12 -2 -2 -3 4 5 9 10000 100120 12 56 -2 -2 -3 4 5 9 10000 100120 12 56 12345 -2 -2 -3 4 5 9 10000 100120 12 56 12345 -2 -2 -3 4 5 9 10000 100120 12 56 12345 5 -2 -2 -2 -3

1 2 -1	

Problem 2: Modify above implementation to add following functions:

3	insertCycle	Should read an integer N from stdin and make last node of existing list to point to *Nth* node of the list, thus creating a cycle.
4	hasCycle	Should print 1 if cycle exists in the list, otherwise it should print 0.
5	traverseGeneric	Generic form of traverse function to handle linear (i.e. Acyclic) as well cyclic linked list.
6	destroyGeneric	Generic form of destroy function to handle linear (i.e. Acyclic) as well cyclic linked list.

Algorithm for hasCycle:
To detect a cycle in the DynamicList by reversal of links: repeatedly reverse every link in the list while traversing it; if you revisit the head node then there is a cycle.

2. Measuring (Dynamic) Memory Used and Time Taken

a. **Measuring Memory:** The amount of dynamic memory allocated can be kept track by programming. Define the following types:

- a record type named *myMem* containing two fields: (i) *ptr*, of type void * and (ii) *sz*, of type *size_t* (*size_t* is the return type of *sizeof* operator in C)
- a pointer type *myPtr*, which is of type pointer to *myMem*.

Define procedures *myAlloc* and *myFree* such that:

myPtr myAlloc(size_t reqSize) :

- callls *malloc(reqSize)* and if it succeeds:
  - increments global variable curHeapSize by reqSize the amount requested for allocation, and updates global variable maxHeapSize if curHeapSize exceeds the previous maximum.
  - allocates a myMem and sets it ptr field to what malloc returns and sz field to reqSize
  - returns the pointer to the allocated myMem

void myFree(myPtr p)

- calls *free* with the ptr field of what p is pointing to
- decrements curHeapSize, and
- deletes (i.e. deallocates) the record pointed to by p

In the end, you may need to print heap memory left (i.e. not freed) and maximum heap memory allocated during the entire execution. You shall need to print the values in bytes.

b. The amount of running time can be measured using the *time* function. Use #include <time.h> this function. To measure the execution time of a particular function, you need to measure the clock time using *time* function before and after the function call and then take difference of the two. Refer to man pages on how to use this function.

Exercise:

(a) Given a file containing a list of integers in increasing order, create a sorted list, Ls, implemented as a dynamically allocated array.

i. Measure the time taken for a random mix of K1 insertions and K2 deletions in Ls, for some K1>=0 and K2>=0.

ii. Repeat the previous step for different K1 and K2 values.

iii. Also, measure the total heap space allocated in each case.

(b) Repeat the previous exercise where Ls is implemented as a linked list.