CS3723 Pgm Assignment #1: Storage Management in C (30 pts)

In this assignment, you will create **heap storage management functions** based on using reference counts and understanding metadata. You will write functions which manage the reference counts for user data nodes:

* When a node is created, it sets the count to 1 (this is like a variable referencing data causing the count to be 1).
* A node (i.e., *from* node) can be associated with another node (i.e., *to node*) by changing a pointer attribute in the *from* node. This will cause the *to* node to have its reference count increased. If the *from* node was already referencing another node via that particular pointer attribute, that original referenced node should have its reference count decreased. If the new *to* *node* value is NULL, don't attempt to increase its reference count.
* We can also add references to nodes similar to a node having another variable referencing it.

You will also have to use the metadata to print nodes. (See the sample output.)

It is likely that a question on the midterm exam will be easier if you do this programming assignment.

So that we can actually *examine* nodes that have been freed, instead of using malloc() and free(), use the following functions provided by the driver:

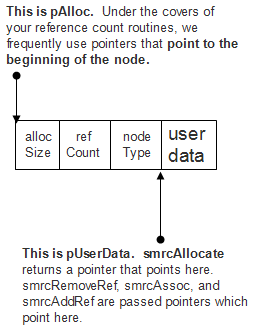
AllocNode \* smAlloc(StorageManager \*pMgr, short shTotalSize)

Returns the address of the allocated memory (not the address of the user data area).

void smFree(StorageManager \*pMgr, char \*pUserData, SMResult \*psmResult)

Since we want to examine "freed" memory, this doesn't actually free the memory. It simply marks the memory as free by changing the sign on shAllocSize. Notice that this is passed the address of the user data area not the address of the allocated node.

In the past when you used malloc() and free(), malloc() returned a pointer for your *user data* and you passed free() a pointer to your *user data.* As you know from lecture, malloc() actually allocated more bytes than you requested, including an attribute for the total size. The following diagram shows that our allocated nodes(for this assignment) contain an allocated size, reference count, and node type:



To help reduce the code that you have to write and to reduce debugging difficulty, I have provided the following:

cs3723p1Driver.c - reads the input file, calls your reference count storage management functions, and uses a hash table to store the addresses of allocated memory (so that those can be subsequently associated with other nodes or freed). An important function is the setData function which uses metadata to set attributes in a node. The driver also provides functions for printing the heap memory to help with debugging.

cs3723p1.h - include file for this program. Some important typedefs:

NodeType – describes one node type: name, beginning subscript in metaAttrM array, total size

MetaAttr – describes one attribute in a node type: name, type, size in bytes, and offset

AllocNode - contains the node's size, node type, reference count, and the user's data

StorageManager - a structure that contains the address of the heap (pBeginStorage), free memory pointer, an array of NodeType entries, and an array of MetaAttr entries. It does not have a count of the number of entries in those arrays. Instead, sentinels are used to mark the end of the arrays.

SMResult - used by the smrc... functions to specify whether they executed successfully.

hashApi.cpp - C++ code to integrate C with the C++ Hash Table Class (unordered\_map). This supports functions getHash, putHash, eraseAll, and printAll. This is only used by the driver.

hexDump64.o - A function to produce hexDumps of data. This is used by the driver when printing the heap's contents. This file is available on the server at this location: **/usr/local/courses/cs3723/clark**. Copy that .o to your folder.

p1Input.txt - Input text file suitable for the driver. The driver uses **stdin** so redirect input from this file.

makefile - Please use this makefile to create your **p1** executable. Note that you should not use the hen servers. The makefile uses g++ instead of gcc. This file can be found at **/usr/local/courses/cs3723/clark**. To build the executable (with it automatically building the other pieces), type  
 **make p1**

**You will need to code** the following functions; however, due to modularity concerns, you may need to create 1 or more extra functions. Your code should be placed in cs3723p1.c. Look at the prototypes in the include file.

smrcAllocate - allocates a node

* It is passed the user data size (which isn't the size to allocate), the node type, and the user's data. How much do we allocate? this size plus ?
* Invokes the driver-provided smAlloc to actually provide the memory.
* Initializes an AllocNode:
  + Set its reference count to 1.
  + Set its node type.
  + Set its size.
  + Set its sbData.
* Returns a pointer (from the user's perspective) to the allocated memory. This is not the address of the AllocNode! (See the diagram.) If memory was not allocated, it should return NULL.

smrcRemoveRef - removes (i.e., decrements) a reference to the specified user data.

* Passed a pointer (from the user's perspective) to the allocated memory. This is not the address of the AllocNode!
* Decrements the reference count for the corresponding AllocNode.
* If the reference count reaches zero, it must free the AllocNode:
  + If the user node type references pointers, decrement the ref count on any directly referenced user data nodes*. (*This will be a recursive call to smrcRemoveRef.*)*
  + Frees the AllocNode which reached a ref count of 0 by calling smFree. Note that smFree keeps the "freed" node around and sets it size to the negative of the size.

smrcAssoc - sets a user pointer in the specified user data node to a new referenced user data node.

* The referenced attribute must be a pointer. If not, set the rc to RC\_ASSOC\_ATTR\_NOT\_PTR.
* If that specified pointer attribute is already referencing something, that referenced user data node needs to be smrcRemoveRef.
* Unless it is NULL, the new referenced user data node needs to have its ref count increased.
* Change the user pointer in the specified user data node to point to the new referenced user data node or NULL (if that was specified).

smrcAddRef - adds a reference to the specified user pointer.

* For the node accessed via the user pointer, the reference count is increased by one

printNode - prints the specified node (see sample output) and returns a list of pointers that it references.

* Passed a pointer (from the user's perspective) to the allocated memory.
* Print the node's size, reference count, node type name.
* For each attribute in the user data based on the metadata:
  + print the attribute name, type, and its value.
* This function **must use the metadata** to obtain the attributes in the user data portion of nodes. If you hard code using Customer and/or LineItem you will lose at least 50% of the credit. (Examine the driver's **setData** function for more information.)
* Note that the size might be a negative number if the node had been previously freed.

Notes:

1. To help understand how to use the metadata, examine the driver's setData function. Also examine the driver's smFree to better understand user data addresses versus allocated addresses.
2. To simplify grading, please include your C code and output in a zip file with your name in mixed case with last name followed by first (e.g., KingJoe.zip).
3. Your code must follow my **programming standards**.
4. You must make certain your code works on a fox server and can be compiled by the specified makefile.
5. For Microsoft Visual Studio Users:

* Since I provided the hexDum64 as a .o file, you can't use it. Inside the driver, rename dumbHexDump to hexDump. Please remember to delete it before running on the UTSA Linux server.
* If you need a wider Console Window:
  + Once the console window displays (you may want a break point in your code so that it doesn't disappear), click the top left corner of the console window.
  + Properties
  + Layout
  + Change the Screen Buffer Size to 120
  + Change the Window Size to 120

Sample Output:

Metadata

Node Type Beg Attr Sub Total Sz

Customer 0 48

Attribute Name Type Offset Size

customerId S 0 12

name S 12 20

pFirstItem P 32 4

pNextCust P 36 4

balance D 40 8

LineItem 5 32

Attribute Name Type Offset Size

productId S 0 10

iQtyReq I 12 4

dCost D 16 8

pNextItem P 24 4

>>> ALLOC C111 Customer 111,Sal A Mander,NULL,NULL,100.00

>>> PRTNODE C111

Alloc Address Size Node Type Ref Cnt Data Address

0059A130 54 0 1 0059A136

Attr Name Type Value

customerId S 111

name S Sal A Mander

pFirstItem P 00000000

pNextCust P 00000000

balance D 100.000000

>>> ALLOC C222 Customer 222,Barb Wire,NULL,NULL,200.00

>>> PRTNODE C222

Alloc Address Size Node Type Ref Cnt Data Address

0059A166 54 0 1 0059A16C

Attr Name Type Value

customerId S 222

name S Barb Wire

pFirstItem P 00000000

pNextCust P 00000000

balance D 200.000000

>>> ALLOC PPF001 LineItem PPF001,5,9.95,NULL

>>> PRTNODE PPF001

Alloc Address Size Node Type Ref Cnt Data Address

0059A19C 38 1 1 0059A1A2

Attr Name Type Value

productId S PPF001

iQtyReq I 5

dCost D 9.950000

pNextItem P 00000000

\*

\* associate customer 111 with a next pointing to 222

\*

>>> ASSOC C111 pNextCust C222

\* customer 111's ref cnt should still be 1, but its pNextCust should point to 222

>>> PRTNODE C111

Alloc Address Size Node Type Ref Cnt Data Address

0059A130 54 0 1 0059A136

Attr Name Type Value

customerId S 111

name S Sal A Mander

pFirstItem P 00000000

pNextCust P 0059A16C

balance D 100.000000

\* customer 222's ref cnt should now be 2

>>> PRTNODE C222

Alloc Address Size Node Type Ref Cnt Data Address

0059A166 54 0 2 0059A16C

Attr Name Type Value

customerId S 222

name S Barb Wire

pFirstItem P 00000000

pNextCust P 00000000

balance D 200.000000