Machine Problem 1: A High Performance Linked List

(Due: Before lab meeting in week 2)

Introduction:

In a traditional implementation of a linked list, memory is allocated for each newly inserted item. When an item on this list is deleted, the memory allocated for it is freed and given back to the operating system. Consequently, each allocation/de-allocation request for insertions or deletions respectively involves interacting with the operating system's memory manager. System calls have a large overhead associated with them. The processor must stop the execution of the user process, switch to executing system code, spend time performing the requested system function, and as such, the calls to the memory manager hinder the performance of the linked list. In this assignment, we will explore a solution to this problem which will produce a high performance and efficient implementation of a linked list.

In this assignment, the program you create will serve as the memory manager instead of relying on the operating system to do the dirty work. Your program should obtain/reserve a fixed amount of memory from the operating system's memory manager during initialization (this is done by calling either malloc or new). After initially acquiring memory from the system, your program should use this memory to manage the linked list throughout its execution. Consequently, extraneous and expensive calls to the system's memory manager will no longer be necessary.

Each element of the linked list occupies a specific amount of space, known as the basic block size and denoted by the variable b. The memory size as a whole is determined by the parameter m. As such, there can be at most m/b elements in the list. Any insertion requests that would attempt to insert more than m/b elements into the list should be immediately rejected.

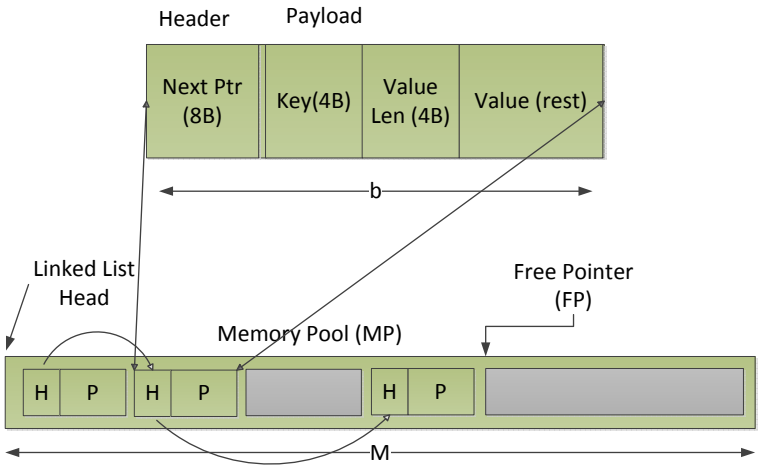
The list should be managed through a Head Pointer (HP) and a Free Pointer (FP). The former points to the head of the list, and the latter points to where the next insertion should happen.

Each linked list item can be separated into two sections: a header and a payload. The header contains necessary information for maintaining the list (i.e. the next pointer, the previous pointer (for doubly linked lists), other metadata, etc...). The payload portion consists of a key-value pair. The key is a 4 byte integer, and the value is of variable length, but has a maximum size that is determined by the header and key size.

Figure #1 visually demonstrates how a singly linked list should be organized. In the top

of the figure, a linked list item is shown in detail. Note that the size of pointers depend on the machine/OS type (i.e. in a 64-bit machine, pointers are 8-bytes as opposed to the 4-byte pointers present in 32-bit machines).

Figure #1: Structural view of a linked list in memory



Assignment:

You are to implement a singly-linked list with the mentioned features in C/C++.

- There should be 3 files in your program: main.c/cpp, linked_list.h, and linked_list.c/cpp. A sample main file will be provided for you to finish.
- Your implementation should contain at least the following functions (declared in the .h file and defined in the .c/.cpp file).

```
void Init(int M, int C); /* Initially obtains M bytes of memory by
calling the standard library function malloc() and initializes the
linked list */

void Destroy(); /* Destroys the linked list and returns memory to the
system by calling another library function, free(). */

int Insert(int x, char* value_ptr, int value_len); /* Inserts a
key-value pair, where the key is an integer, x, and the value is some
data pointed to by the pointer value\_ptr of length value\_len. You
should use the library function, memcpy() to copy the value. */

void Delete(int x); /*Deletes the first item from the beginning with
the key x. */

char* Lookup(int x); /* Finds the given key in a list and returns a
pointer to the first occurrence of that key. */

void PrintList(); /* Prints out all the items' key-value pairs
```

```
sequentially starting from the head of the list. Print only the key
and the value-length. Do not print the actual value as it could
contain binary/non-printable data. */
```

- Your implementation should include a program called testlist which reads the basic block size and the memory size (in bytes) from the command line, initializes the memory, makes some insertion/deletion calls, prints the list using the PrintList() function, and then destroys the allocated memory using Destroy(). Note that all these calls should go inside testlist's main function.
- Use the getopt() C library function to parse the command line for arguments. The usage for your program should be as follows:

./testlist [-b <blocksize>] [-s <memsize>]

-b <blocksize>	Defines the basic block size, b, in bytes. Default = 128 bytes
-s <memsize>	Defines size of memory allocated in bytes. Default = 512 kB

- Make sure that your program does not crash in any case. Here are a number of scenarios that your program should account for. In these cases, your program should simply print an errir, skip the given instrution, and continue to work (i.e. do not exit for any reason).
  - Deleting non-existent keys from the list.
  - Trying to insert keys after the given memory is full.
  - Trying to insert values that do not fit in the payload section.

Report Documentation

Provide a PDF report describing your findings and answering all the following questions. Do you notice any wastage of memory when items are deleted? If so, can your program avoid such wastage? How would you do so? Can you think of a scenario where there is space in the memory but no insertion is possible? What is the maximum size of the value when the pointers are 8 bytes?

Submission Instructions

Submit a zipped folder containing your code and the PDF report named MP1.pdf. There should be 3 c/cpp/h files: linked_list.h, linked_list.c/cpp, and main.c/cpp. Demonstrate your work during lab meetings. Make sure that your program runs on one of the CSE department's linux servers (e.g., linux2.cse.tamu.edu, compute.cse.tamu.edu, build.tamu.edu). Remember that we are using a new platform Vocareum (available at vocareum.com) for submitting machine problems. Please register as a student on this website, play around with it, and become familiar with it so that there will be no issues before the assignment's deadline. Please don't be afraid to talk to your TA or instructor if you have any issues.