Birla Institute of Technology and Science
CS F211, Data Structure and Algorithms

Second Semester 2018 – 19

Assignment

Learning Outcome: (a) How objects and pointers are implemented in RAM memory

(b) How defragmentation is done.

A linked list is a ADT in which the objects are arranged in a linear order. Unlike an array, in which the linear order is determined by the sequence of indices, the order in a linked list is determined by a pointer with each object. As shown in Figure 1, each node of a doubly linked list ***L*** contains an object with an attribute key and two other pointer attributes: *next* and *prev*. For a node ***x*** in the list, *x.key* contains its key value, *x.next* points to its successor and *x.prev* points to its predecessor in the linked list. If *x.prev* = NULL, the node x has no predecessor and is therefore the first node, or head, of the list. If *x.next* = NULL, the node x has no successor and is therefore the last element, or tail, of the list. An attribute *L.head* points to the first element of the list. If *L.head* = NULL, the list is empty.

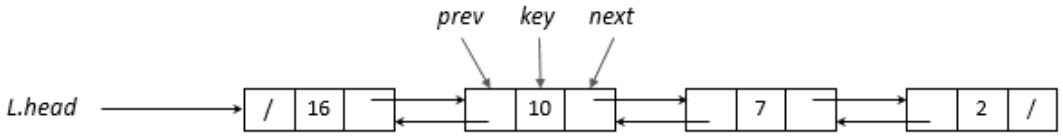

Figure 1

[Note: "/" represent NULL]

In computer memory, the words are typically addressed by integers from *0* to *N-1*, where *N* is an applicably large number. In many programming languages, an *object* occupies a contiguous set of locations in computer memory. A pointer is used to address the first memory location of the object, and we can access other memory locations within the object by adding an offset to the pointer. In figure 2, there is an object ***obj1*** comprised of three attributes *attr1*, *attr2* and *attr3* of integer data types. There is a pointer *P* pointing to the first memory location (i.e., 100) of *Obj1*. The offsets for *attr1*, *attr2* and *attr3* are 0, 4 and 8 respectively.

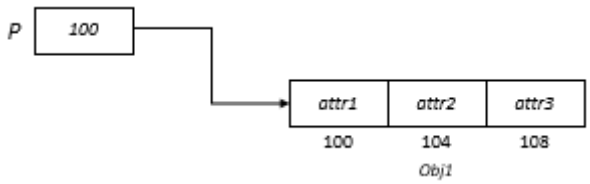

Figure 2

The same strategy can be used to implement objects, such as doubly linked list nodes, in programming environments that do not provide explicit pointer data types. Figure 3 shows how to use a single array A to store the linked list from figure 1. An object occupies a contiguous subarray A[*j*…*k*]. Each attribute of the object corresponds to an offset in the range from *0* to *k - j*, and a pointer to the object is the index *j*. In figure 3, each list node is an object that occupies a contiguous subarray of length 3 within the array. The offsets corresponding to attributes *key*, *next* and *prev* (in this order) are 0, 1 and 2, respectively. A pointer to an object is the index of the first element of the object.

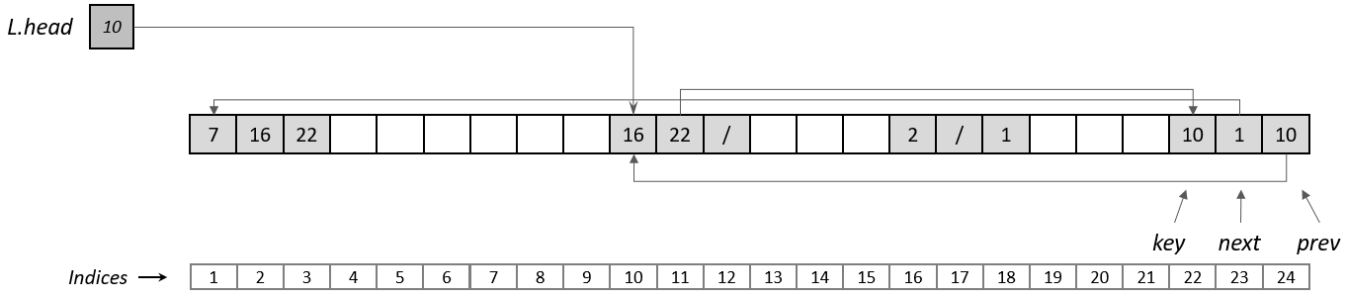

Figure 3

Allocating and freeing objects:

Suppose that an array has length *3*k* (*m* nodes, where *m = 3*k*) and that at some moment the dynamic set contains *n* <= *m* objects. Then *n* objects represent elements currently in the dynamic set, and the remaining *m – n* objects are free; the free objects are available to represent elements inserted into the dynamic set in the future, or to represent another dynamic set.

We keep the free objects in a doubly linked list, which we call the *free list*, in which *prev* field is unused. The free list uses only the *next* pointer attribute of each node, which stores the index of *next* free node within the list. The head of the free list is held in the global variable *free*. When the dynamic set represented by doubly linked list *L* is nonempty, the free list may be intertwined with list *L*, as shown in Figure 4. Note that each object in the representation is either in list *L* or in the free list, but not in both. Also, although *prev* field if present in each node of the free list, it is not used. You can consider it having some garbage values.

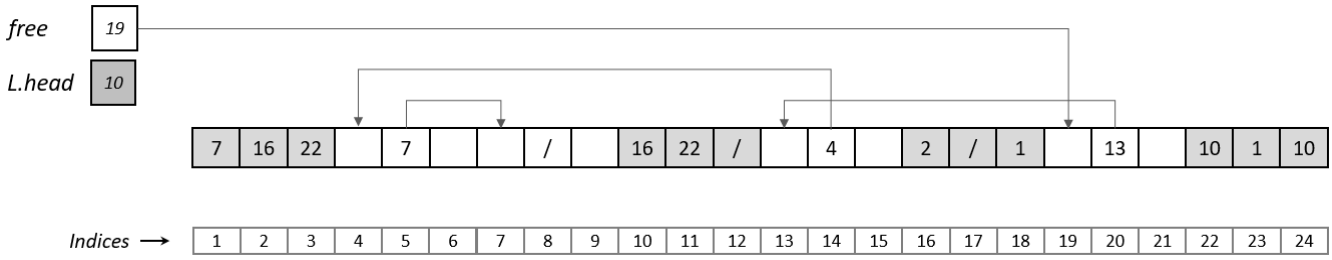

Figure 4

The free list acts like a stack: the next object allocated is the last one freed. We can use a list implementation of the stack operations PUSH and POP to implement the procedures for allocating and freeing objects, respectively. We assume that the global variable *free* used in the following procedures points to the first element of the free list.

The free list initially contains all *n* unallocated objects. Once the free list has been exhausted, running the ALLOCATE-OBJECT *function* signals an error. We can even service several linked lists with just a single free list. Figure 5 shows two linked lists and a free list intertwined through *key*, *next*, and *prev* fields.

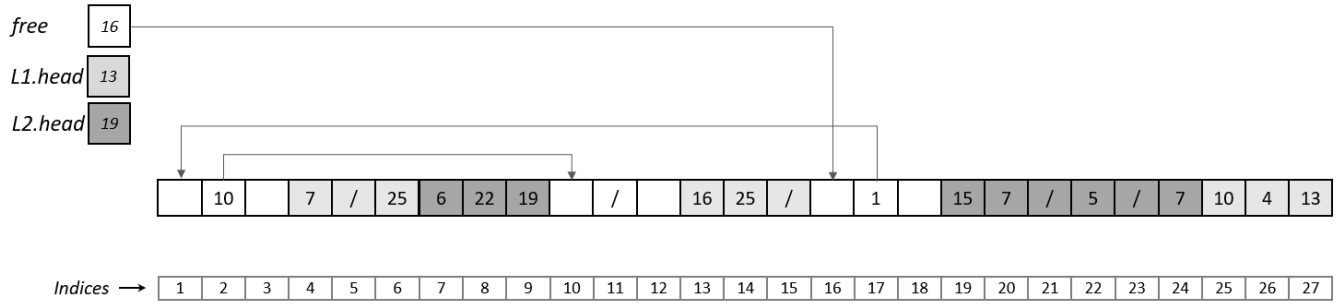

Figure 5

For this assignment you will write a menu driven program to implement multiple dynamic sets (in the form of multiple doubly linked lists) in an array. You can visualize an array as a RAM memory. So, this assignment will make you understand how linked objects are stored in memory. Of course, in actual RAM these can be heterogeneous objects, but for the sake of simplicity, for this assignment we will assume homogeneous objects in the form of simple integers. It other words, each node of a doubly linked list has three fields: object (i.e. an integer), next, and prev.

Write a menu driven program to perform following operations:

1. Create a new list
2. Insert a new node in a given list in sorted order: *Here list and node object is taken as input*
3. Delete an element from a given list: *Here list and node object is taken as input*
4. Count total elements excluding free list
5. Count total elements of a list: *Here list is taken as input*

6. Display all lists
7. Display free list
8. Preform defragmentation
9. Press 0 to exit

Defragmentation: Observe that after few insertions and deletions, free list will have scattered nodes. Defragmentation (or compaction) is a process of relocating/rearranging the non-contiguous data to restore them into contiguous manner. On other words, after defragmentation, free list will have nodes in sequence. For example, in figure 5, all elements of L1 and L2 lists will be placed together in contiguous manner after executing defragmentation.

Deliverables

1. Break your program in multiple files to implement modularity.
2. Learn yourself how to implement **make-files**
3. While evaluating, if I run **make**, your program should execute.
4. Deadline for submitting the assignment is February 04, 11 PM. Submission details will be uploaded later. You will be submitting your assignment on an anti-plagiarism software (like turnitin, etc) whose details will be provided later. If your code is copied or plagiarized, proper disciplinary action will be taken.

Marking Scheme: Your program should RUN completely and marks will be 0M / 4M / 8M (without defragmentation part) / 12M (everything along with defragmentation part).

Layout of the menu driven program

Your program should continuously display following options until user presses 0.

Select an option:

1. **Create a new list**
2. **Insert a new element in a given list in sorted order**
3. **Delete an element from a given list**
4. **Count total elements excluding free list**
5. **Count total elements of a list**
6. **Display all lists**
7. **Display free list**
8. **Perform defragmentation**
9. **Press 0 to exit**

O/P:

Select an option: 1
The sequence number of the newly created list is: n
Enter key value to be inserted in the newly created list-*n*: *here user inputs integer m*
Program outputs "SUCCESS" or "FAILURE: MEMORY NOT AVAILABLE".

Again Menu is displayed:

Select an option: 2
List you want to insert in: *here user inputs list number n*
Enter the key value: *here user inputs integer m*
Program outputs "SUCCESS" or "FAILURE: MEMORY NOT AVAILABLE".

Again Menu is displayed:

Select an option: 3
List you want to delete from: *here user inputs list number n*
Enter the key value: *here user inputs integer m*
Program outputs "SUCCESS" or "FAILURE: ELEMENT NOT THERE / LIST EMPTY".

Again Menu is displayed:

Select an option: 4
Program outputs: Total number of nodes in all lists are M.

Again Menu is displayed:

Select an option: 5
Enter the list number: *here user inputs list number n*
Program outputs: Total number of nodes in list n are M.

Again Menu is displayed:

Select an option: 6

Program outputs all the lists present in memory (or array in our case) in following format. This output is wrt Figure 5.

Elements of list-1 are:

key	next	prev
16	25	NIL
10	4	13
7	NIL	25

Elements of list-2 are:

key	next	prev
15	7	NIL
6	22	19
5	NIL	7

Again Menu is displayed:

Select an option: 7

Program outputs all nodes of a free list in following format. This output is wrt Figure 5.

Elements of free list are:

[16, 1, 10]

Again Menu is displayed:

Select an option: 8

Here your program performs defragmentation and outputs SUCCESS / FAILURE. Observe that defragmentation does not require all nodes of a particular list to be stored contiguously. Rather, all allocated nodes of all the lists should be stored contiguously.

We can check the SUCCESS of this operation by calling other appropriate menu items.