//

// Shraddha Jamadade

// Student ID: 110287963

// Date: 10/30/2017

// 117 Assignment 5

//

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//

// CODE DESCRIPTION:

// Assignment to implement simulation for dynamic memory i.e. heap for garbage collection(allocation by manipulating linked

// lists using insertion sort algorithm and deallocation using lazy garbage collection mechanism with mark and sweep phases).

// Linked lists used:

// 1. lisp1, with head pointer L1(initial value -1)

// 2. lisp2, with head pointer L2(initial value -1)

// 3. free-list(initially contains all cells/), with head pointer Free(initial value 1)

// It is a menu driven program with opts for:

// 1. Print\_memory(to display memory contents and values of pointers)

// 2. Insert(Head\_pointer, key)

// 3. Delete(Head\_pointer, key)

// 4. Garbage\_Collect(mark and sweep garbage collection)

// Input: User input is any one opt from the menu

// Inputs for this assignment:

// Eight insertion operations keeping the order: insert (L1, 3); insert (L1, 1); insert (L2, 4); insert (L1,5);

// insert (L2, 2); insert (L2, 9); insert (L2, 8); insert (L1, 4)

// Invoke print memory opt to show memory contents and pointer values after each operation.

// Delete cmd using the following sequence of operations: delete (L1, 4); print\_memory; delete (L2, 8); print\_memory;

// delete (L1, 1); print\_memory; delete (L2, 4); print\_memory; delete (L1, 5); print\_memory; garbage-collect( ); print\_memory

// Output: Executes the given input cmds to to insert, delete and to display memory contents.

// Program Name: 117Assignment5.cpp

// Compile and run: GNU GCC Compiler(Code Blocks)

// x86\_64-w64-mingw32-g++.exe -Wall -fexceptions -g -c C:\Users\SHRADDHA\Documents\117\CSci117Assignment5\main.cpp -o obj\Debug\main.o

// x86\_64-w64-mingw32-g++.exe -o bin\Debug\CSci117Assignment5.exe obj\Debug\main.o

//

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//

// CODE:

#include<iostream> //for input output stream

#include<stdio.h> //For input output functions

#include<stdlib.h> //For pointer operations

using namespace std;

struct node //creates a structure called node where each node stores a key, next value, mark bit and next index.

{

int key; //values to be inserted

struct node\* next; // next pointer

int mark; //initially contains all values as 0

int indexNext;

};

struct node mem[10]; //Instantiation of the memory with 10 spaces

struct node L1,L2; //Instantiation of the head pointers for the two lists

struct node Free; //Instantiation of the head pointer for the free-list

int index=0; //To hold the index of the next element

// Function to insert elements into the linked lists. It takes input as the cmd: //insert(Head\_pointer, key)

void Insert(struct node\* Head\_pointer,int key)

{

int i;

struct node \*previous; // Pointer to hold the previous node

struct node \*t=Head\_pointer->next; // Temporary pointer t

if(index==0) //if it is the first node

{

Head\_pointer->next=&mem[0]; //Head\_pointer points to index of next element

Head\_pointer->indexNext=index+1;

mem[0].key=key;

mem[0].indexNext=-1;

index++;

return;

}

mem[index].key=key;

if(Head\_pointer->next==NULL) //if next address of the Head is null that is it is the first element

{

Head\_pointer->next=&mem[index];

Head\_pointer->indexNext=index+1; //Head points to the index of the next element

mem[index].indexNext=-1;

index++; //index=index+1, incremented to next element

return; //exit from function

}

if(mem[index].key<Head\_pointer->next->key) //Performing insertion sort operation

{

Head\_pointer->next=&mem[index]; //Head\_pointer points to the next address

i=Head\_pointer->indexNext; //next address is stored in i variable

Head\_pointer->indexNext=index+1; //Head\_pointer points to the index of next element

mem[index].next=t; //value stored in t

mem[index].indexNext=i; //New element points to the next index of next element

}

else //traverse till next element is greater than current element

{

while(mem[index].key>t->key && t->next!=NULL)

{

previous=t;

t=t->next;

}

if(mem[index].key<t->key)

{

previous->next=&mem[index];

i=previous->indexNext;

previous->indexNext=index+1;

mem[index].next=t; //New element points to the next address of next element mem[index].indexNext=i; //New element points to the next index of next element

}

else

{

t->next=&mem[index];

t->indexNext=index+1;

mem[index].indexNext=-1;

}

}

index++; //go to next index

}

// Function to delete elements from the linked lists. It takes input as the cmd: //delete(Head\_pointer, key)

void Delete(struct node\* Head\_pointer,int key)

{

struct node \*previous,\*next; //pointers next and previous->to store next and previous

//values while traversing lists

struct node \*t=Head\_pointer->next; //pointer t to temporarily store address of next node

previous=next=Head\_pointer->next; //assign previous and next pointer to the next node

if(Head\_pointer->next->key==key) //find whether the next node of the Head\_pointer

//contains the key

{

Head\_pointer->next=next->next; //Head\_pointer points to the address of next element

Head\_pointer->indexNext=next->indexNext; //Head\_pointer points to the index of next //element

Head\_pointer->next->indexNext=-1;

return ; //exit the function

}

while(t->key!=key) //continue the while loop till key is not equal to required key

{

previous=t;

t=t->next;

next=t->next;

}

if(t->next==NULL) //if it is the last element

{

previous->next=NULL;

previous->indexNext=-1;

t->indexNext=-1;

return;

}

t->next=NULL;

previous->next=next; //after traversing, the previous node points to the address of the next node

previous->indexNext=t->indexNext; //after traversing, the previous node points to the index of //the next node

t->indexNext=-1; //change next index to -1

}

//Function to print the elements from the 2 lists

void Print()

{

struct node \*t=L1.next; //pointer t points to the first element of lisp1 and stores its

// value temporarily.

printf("\nLisp1:\n"); //print

if(L1.next==NULL) //if lisp1 is empty (condition check)

{

printf("List 1 is Empty\n"); //print message

}

else

{

while(t->next!=NULL) //while loop executes till there is no next element(i.e. till end of list)

{

printf("%d ",t->key); //print elements

t=t->next;

}

printf("%d",t->key);

}

t=L2.next; //pointer t points to the first element of lisp2 and stores its value temporarily.

printf("\nLisp2:\n"); //print

if(L2.next==NULL) //if lisp2 is empty (condition check)

{

printf("List 2 is Empty\n"); //print message

}

else

{

while(t->next!=NULL) //while loop executes till there is no next element(i.e. till end of list)

{

printf("%d ",t->key); //print elements

t=t->next;

}

printf("%d",t->key);

}

}

//Function for garbage collection. There are 2 phases:

//Mark phase traces all nodes starting from head node (marks reachable nodes)

//Sweep phase collects unmarked nodes from lowest address and returns to free-list

//Last In First Out operations are performed. Garbage node is collected and stored at the head.

//On memory allocation, the first node is assigned.

void garbageCol()

{

int i;

struct node \*t; //MARK PHASE

struct node \*p1; //pointer p1 points to list1 elements

struct node \*p2; //pointer p2 points to list2 elements

p1=L1.next;

p2=L2.next;

while(p1->next!=NULL) //while it is not the last node(while all nodes are reachable in list1)

{

p1->mark=1;

p1=p1->next;

}

p1->mark=1;

while(p2->next!=NULL) //while it is not the last node(while all nodes are reachable in list2)

{

p2->mark=1;

p2=p2->next;

}

p2->mark=1;

p1=&mem[0]; //SWEEP PHASE

Free.next=NULL;

//traverse the memory and check the free blocks (Free pointer should point at free block)

for(i=0;i<10;i++)

{

if(mem[i].mark==0) //if it is reachable but still marked as 0(unmarked)

{

t=Free.next;

Free.next=&mem[i];

mem[i].next=t;

}

}

}

//Function to print memory, called when opt selected. Prints key, next element index and

//mark value

void printMemory()

{

int i;

for(i=0;i<10;i++) //loop uptil the size of the list i.e. 10

{

printf("\nkey: %d, next pointer:%d, mark:%d",mem[i].key,mem[i].indexNext,mem[i].mark);

}

}

//Function to print free memory space, Function called when opt selected

void freeMemoryDisplay()

{

int i;

printf("\nAvailable Free mem:\n");

for(i=0;i<10;i++) //loop uptil the size of the list i.e. 10

{

if(mem[i].mark==0)

printf("\nFree at:mem[%d]",i);

}

}

//Main function. It displays the user selection opts.

//It contains the menu driven function (while loop till user terminates function)

//makes function calls to appropriate functions on user selection

int main()

{

int i,opt; //initialization of structure variables

char cmd[20];

L1.next=NULL;

L2.next=NULL;

for(i=0;i<9;i++)

{

mem[i].key=0;

mem[i].indexNext=i+2;

mem[i].next=NULL;

}

mem[9].indexNext=-1;

mem[9].next=NULL;

do

{

printf("\nMenu:\n1.Print\_memory\n2.Insert\n3.Delete\n4.Display List Elemets\n5.Garbage\_Collect\n6.Print Free mem\n7.Exit\n"); //print the menu

printf("Enter your option\n");

scanf("%d",&opt); //take input from user

if(opt==2) //INSERTT

{

printf("\n Enter the insert command: insert(Head\_pointer, key)\n");

scanf("%s",cmd);

if(cmd[11]=='1')

{

Insert(&L1,atoi(&cmd[13]));

}

else

{

Insert(&L2,atoi(&cmd[13]));

}

}

else if(opt==3) //DELETE

{

printf("\nEnter the delete command: delete(Head\_pointer, key)\n");

scanf("%s",cmd);

if(cmd[11]=='1')

Delete(&L1,atoi(&cmd[13]));

else

Delete(&L2,atoi(&cmd[13]));

}

else if(opt==5) //GARBAGE COLLECTION

garbageCol();

else if(opt==4) //DISPLAY LIST ELEMENTS

Print();

else if(opt==1) //DISPLAY MEMORY CONTENTS

printMemory();

else if(opt==6) //DISPLAY FREE MEMEORY

freeMemoryDisplay();

else if(opt==7) //EXIT

exit(0);

} while(1); //to continue the loop, while with always true condition

}

//

//

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

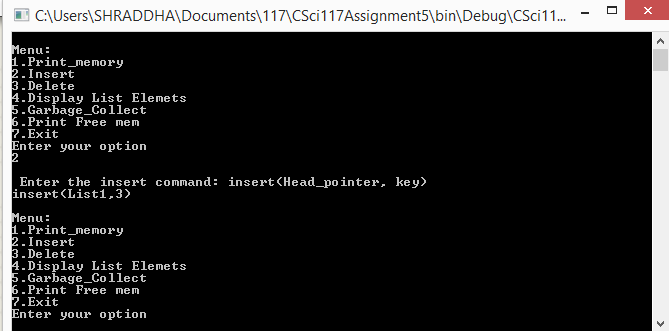
//

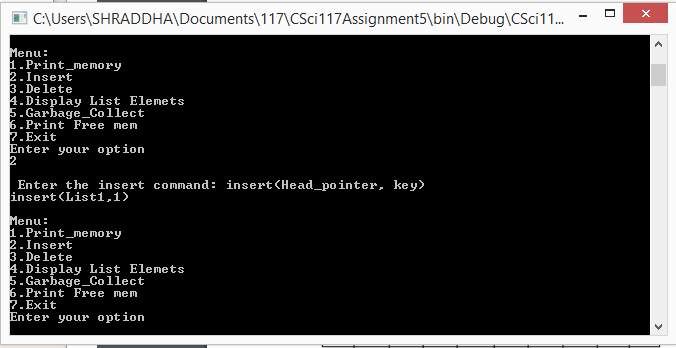
// OUTPUT

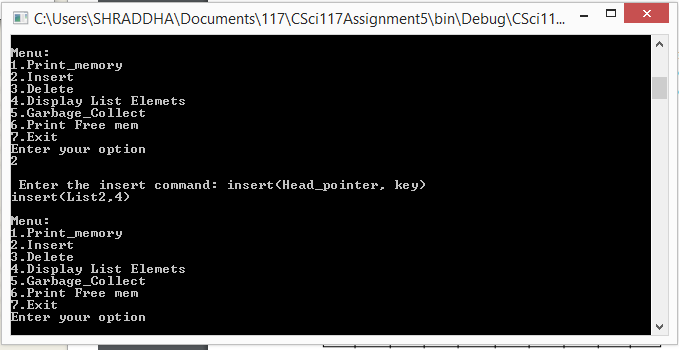
Check empty list condition:

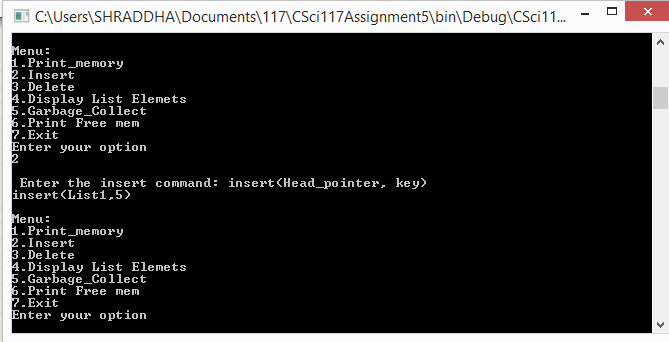


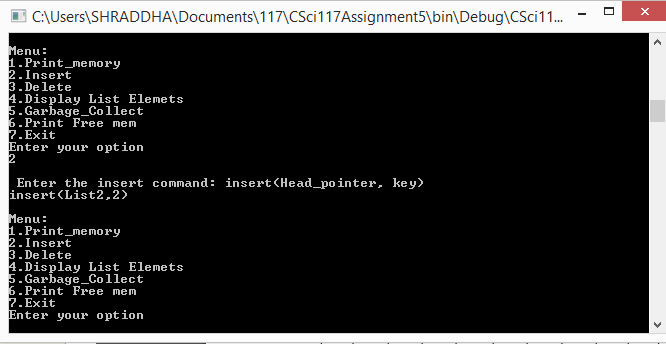
Insert operations in the given sequence:

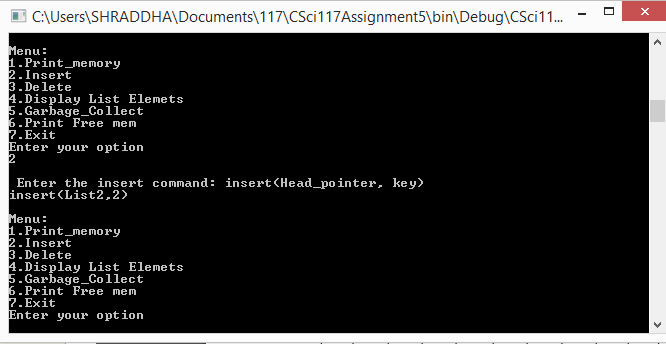


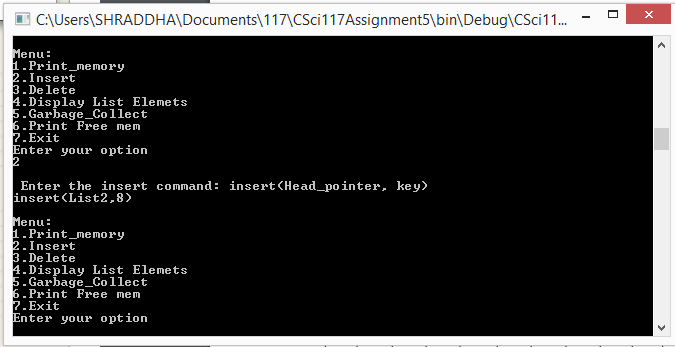


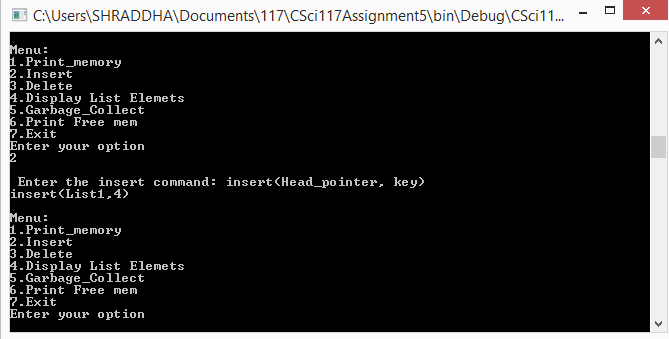




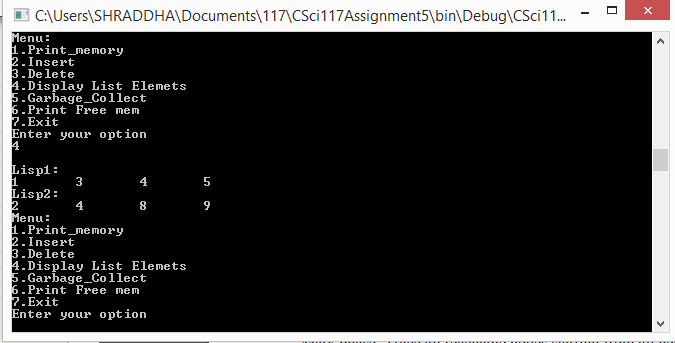




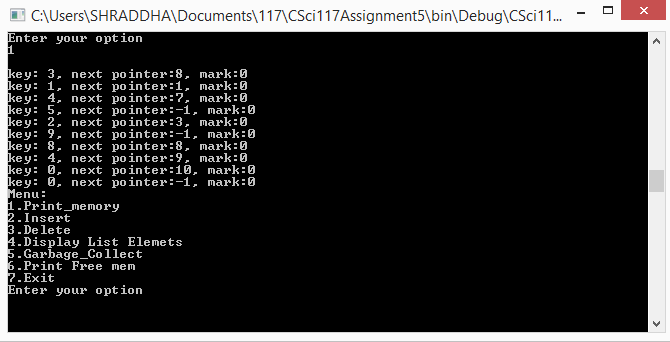




Display list contents



Print memory

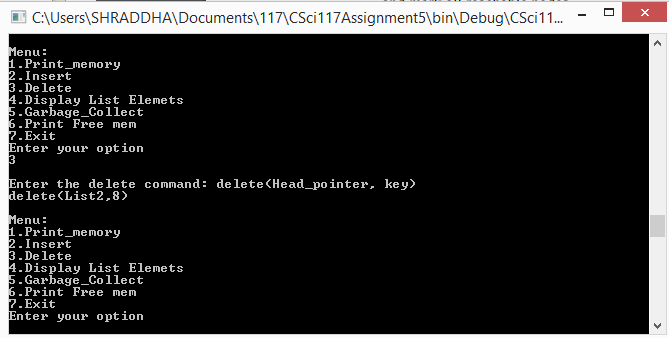


Print free memory

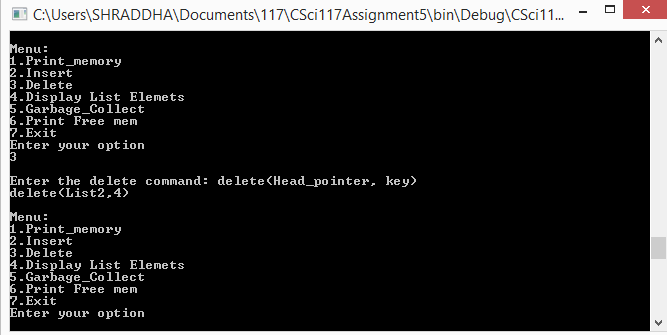


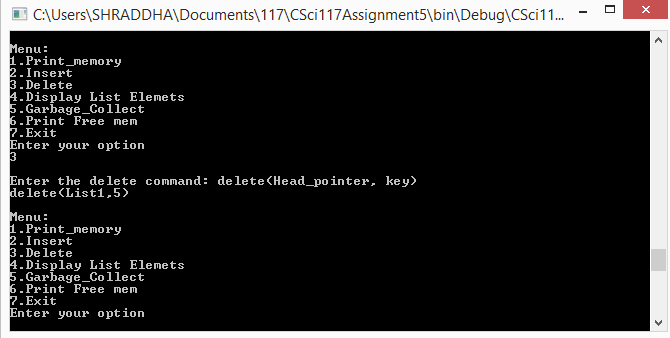
Delete in the given sequence



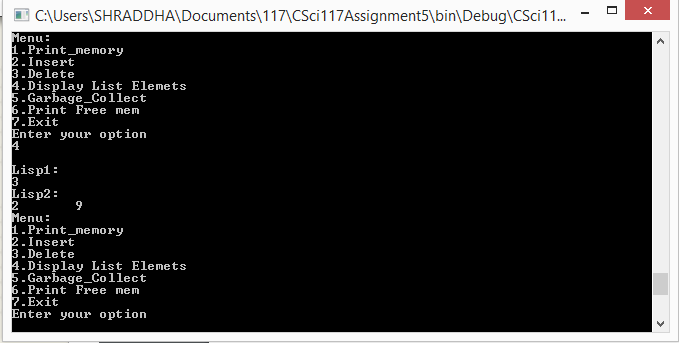




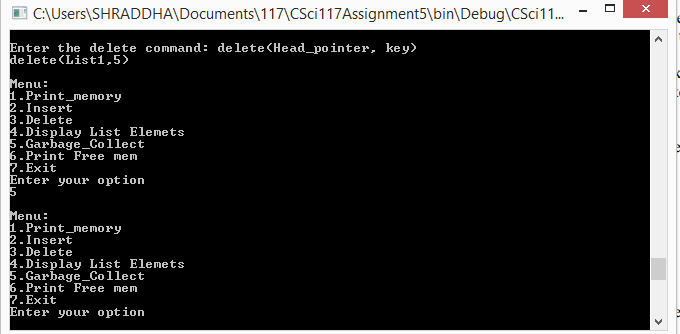




Display list after delete operations



Garbage collect



Print memory after delete

