	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	M.C. JOSÉ MAURICIO MATAMOROS DE MARIA Y CAMPOS
Asignatura:	ESTRUCTURA DE DATOS Y ALGORITMOS I
Grupo:	12
No de Práctica(s):	07
Integrante(s):	ARELLANES CONDE ESTEBAN
No. de Equipo de cómputo empleado:	16
No. de Lista o Brigada:	01
Semestre:	2022-2
Fecha de entrega:	18 abril, 23:59
Observaciones:	

CALIFICACIÓN: _____

Práctica 7. Listas

Estructuras de Datos y Algoritmos I

Autor: Arellanes Conde Esteban

1. Objetivo

El alumno revisará las definiciones, características, procedimientos y ejemplos de las estructuras lineales lista ligada y lista basada en arreglos tanto en sus versiones de extremos abiertos como circulares para poder comprenderlas a fin de ser capaz de implementarlas.

2. Introducción

Una lista simple ya sea ligada, doblemente ligada, ordenada, circular, con arreglos o memoria dinámica debe de tener funciones de inserción, borrado, y búsqueda. Dependiendo de cuál estructura escojamos o necesitemos podemos o no tener un ‘‘append’’ y un ‘‘preppend’’ (inserción normal o reversa) y que se rigen por θ(n) donde n es el número de elementos. Por lo que en el peor de los casos se recorrería toda la lista sin las funciones de inserción anteriormente mencionadas. De igual manera sucede con las funciones de búsqueda pero como pequeña ventaja es que si es una lista ligada podemos empezar a recorrer desde alguno de los dos extremos convirtiendose así en un θ(n/2).

3. Actividad 1:

El programa ‘‘prog1.c’’ requería de una estructura de datos de tipo lista para poder operar, pero esta no estaba implementada. Completando el programa ‘‘prog1.c’’, implementando la estructura de datos correspondiente y ejecutando el programa podemos responder las preguntas que se presentan a continuación.

¿Qué estructura de datos requería el programa? Explique [1 punto]: El programa ‘‘prog1.c’’ requería de una lista simple ligada (o bien con arreglos) debido a las operaciones de borrado, inserción y búsqueda que se necesitaban.

¿Qué salida produce el programa? [1 punto]: La salida que produjo el programa ‘‘prog1.c’’ fue la siguiente:

```
List size: 90
List size: 87
List size: 89

L: 615 ->681 ->205 ->645 ->17 ->93 ->684 ->174 ->113 ->685 ->513 ->
765 ->555 ->423 ->218 ->780 ->996 ->139 ->240 ->237 ->836 ->974 ->
576 ->165 ->33 ->205 ->214 ->501 ->346 ->495 ->467 ->699 ->464 ->42
->143 ->147 ->567 ->255 ->831 ->80 ->20 ->737 ->502 ->589 ->869 ->
79 ->108 ->86 ->914 ->433 ->567 ->841 ->950 ->952 ->45 ->163 ->804 ->
390 ->657 ->622 ->767 ->707 ->85 ->81 ->160 ->850 ->583 ->726 ->
456 ->414 ->157 ->827 ->502 ->659 ->767 ->370 ->508 ->845 ->830 ->5
->593 ->110 ->262 ->511 ->302 ->211 ->814 ->698 ->-1 ->NULL
```

¿Cómo se implementó la función `insert`? Anote y explique su código a continuación [2 puntos]:
Para la función de inserción (‘‘`void insert`’’) necesitamos de insertar un elemento nuevo por implementándola por medio del miembro del nodo siguiente del primer elemento en una lista completamente nueva.

```
void insert(list* l, int ix, int x) {
size_t i;
if (l == NULL) exit(0);
node *n = (node*)calloc(1, sizeof(node));
n->value = x;
if(l->count == 0) preppend(l, x);
node *prev = l->first;
while (i < ix && prev ->next) { prev = prev->next; ++i; }
n->next = prev->next; prev->next = n; ++l->count;}
```

¿Cómo se implementó la función `delete`? Anote y explique su código a continuación [2 puntos]:
Lo único que hay que hacer en el caso en el que queramos eliminar un elemento de la lista es reconectar el elemento anterior con el siguiente al eliminado de la lista.

```
void delete(list* l, int ix) {
size_t i;
if (l->count < 1) exit(0);
node *n = l->first;
while ((ix > i) && n) { n = n->next; ++i; }
node* tmp = n->next;
n->next = n->next->next; free(tmp); --l->count;}
```

4. Actividad 2:

El programa ‘‘`prog2.c`’’ requería de una estructura de datos de tipo lista para operar, pero esta no estaba implementada. Modificando la implementación del programa de la Actividad 1 de tal forma que éste permitiera recorrer la lista en un bucle infinito y utilizándola para que este mismo programa (‘‘`prog2.c`’’) pueda funcionar correctamente.

¿Qué estructura de datos requería el programa? Explique [1 punto]: El programa de manera similar al anterior requería de una lista simple ligada, o bien con arreglos.

¿Qué salida produce el programa? [1 punto]: La salida que produjo el programa ‘‘prog2.c’’ fue la siguiente:

```
L: 887 ->887 ->887 ->887 ->887 ->887 ->887 ->887 ->887 ->887 ->NULL
L: 887 ->778 ->887 ->778 ->887 ->778 ->887 ->778 ->887 ->778 ->NULL
L: 887 ->778 ->916 ->887 ->778 ->916 ->887 ->778 ->916 ->887 ->NULL
L: 887 ->778 ->916 ->794 ->887 ->778 ->916 ->794 ->887 ->778 ->NULL
L: 887 ->778 ->916 ->794 ->336 ->887 ->778 ->916 ->794 ->336 ->NULL
L: 887 ->778 ->916 ->794 ->336 ->387 ->887 ->778 ->916 ->794 ->NULL
L: 887 ->778 ->916 ->794 ->336 ->387 ->493 ->887 ->778 ->916 ->NULL
L: 887 ->778 ->916 ->794 ->336 ->387 ->493 ->650 ->887 ->778 ->NULL
L: 887 ->778 ->916 ->794 ->336 ->387 ->493 ->650 ->422 ->887 ->NULL
*Nota: Esta línea es mostrada unas 74 veces aprox.
List size: 84
L: 887 ->778 ->916 ->794 ->336 ->NULL
List size: 81
List size: 83
L: 28 ->691 ->60 ->764 ->927 ->541 ->427 ->173 ->737 ->212 ->369 ->568 ->430
->783 ->531 ->863 ->124 ->68 ->136 ->55 ->NULL
```

¿Qué modificaciones realizó a la estructura de datos empleada? Anote y explique su código a continuación [2 puntos]: El programa funciona de manera similar al anterior de la actividad 1 y las modificaciones únicamente fueron el bucle infinito de la función ‘‘list_print_segment’’ y un par de funciones adicionales.

```
void list_print_segment(list* l, int offset, int count) {
printf("L: ");
node* n = list_first(l);
int ix = 0;
while((n != NULL) && (ix < offset)) { ++ix; n = next(n); }
while((n != NULL) && (count > 0)) {
printf("%d ->", value(n)); n = next(n); --count;}
printf("NULL\n");}
```

5. Conclusión y Referencias

5.1. Conclusión:

Las estructuras de listas no son particularmente muy rápidas pero sí lo son al momento de almacenar y eliminar información que necesitemos y combinado a alguna otra estructura podría ser muy beneficioso para quién desee implementarla en algún proyecto. Podemos a su vez concluir que se cumplieron los objetivos.

5.2. Referencias:

- Thomas H. Cormen. (2009). Introduction to Algorithms 3e. United States of America: Massachusets Institute of Technology.
- Stephen R. Davis. (2015). Beginning Programming with C++ For Dummies. New Jersey, United States of America: John Wiley & Sons.
- Código completo prog1.c: https://onlinegdb.com/3norG8ji3
- Código completo prog2.c: https://onlinegdb.com/djsmYfH_h