UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Seminario de Solución de Problemas de Estructuras de Datos II

Act01 Oferta académica :: Paradigma Estructurado y Modular

Jesús Uriel Guzmán Mendoza 212451601
Profesor: López Cisneros, Juan José
Sección D06, 2020-B
Tiempo de realización: 25200 segundos

Oferta académica :: Paradigma Estructurado y Modular

Descripción de la actividad

Se creará un programa que posiblemente almacene los siguientes datos: nombre de materia, código de materia, nrc, nombre de profesor, días y horas de impartición, sección, entre otros.

Debe tener las siguientes especificaciones:

1. El almacenamiento se realizará a partir de un "struct" o más
2. Prohibido el uso de variables globales
3. Utilizar funciones y su paso de parámetros
4. Debe considerar la creación de un archivo ‘header’ y de su definición de funciones en otro archivo por cada "struc" generado, además del archivo que contiene el "main()"
5. Se debe trabajar con tipos de datos "char" no "string"

Objetivo (s) de la actividad

El objetivo de esta actividad es poner a prueba nuestras habilidades de conocimientos teóricos y prácticos de el paradigma estructurado y modular, o sea, hacer un programa de tal manera en la que únicamente utilicemos los principios y reglas en este tipo de paradigma y no utilizar nada que provenga de C++, también el objetivo pienso que es ver qué tanto hemos aprendido en la carrera csobre C y los diversos paradigmas, ya que no se acostumbra a enseñar mucho sobre los paradigmas y solamente nos ponen a programar, por lo que en esta tarea se pondrá a práctica todas nuestras habilidades para mandar como parámetros los punteros, no utilizar programación orientada a objetos, etc.

Fundamentos teóricos estudiados

Principalmente, los fundamentos teóricos que estudié fueron aquellos que necesitaba en ese momento preciso a la hora de programar o que no me acordaba, mis fuentes principales fueron canales de programación de indios y foros de Stack Overflow.

No utilicé ningún libro ya que a estas alturas sí tengo fresco lo que es programar en C, sin embargo, habían algunas cosas que nunca aprendí en la carrera que sí tuve que investigar como la los punteros hacia otros punteros debido a que no hay referencias como tal en C y algunas cosas sobre memoria.

Explicación del método de desarrollo y su solución (se sugiere ejemplificar con diagramas y dibujos)

La explicación del método de desarrollo fue imaginar cómo sería la manera más eficiente y que consumiera el menor número de líneas de código posibles para llevar al cabo satisfactoriamente esta tarea con todos los requerimientos abordados.

Se diría que primero imaginé qué tipo de estructuras necesitaría, llegué a la conclusión de que solamente necesitaba los headers de mi lista simplemente ligada y otro para mis materias. Estuve pensando en crear un header para un menú, sin embargo, pensé que esto no tenía sentido en programación estructurada ya que solamente terminaría importanto funciones de un h sin ninguna otra razón justificada tal como modular las estructuras y sus funciones como en mi lista simplemente ligada y mis materias, por lo que mejor puse mis métodos de menú en el main ya que era lo que me tenía más sentido.

Debido a que había leído que tenía que ser lo más eficiente posible en cuanto a memoria , decidí utilizar una lista simplemente ligada para almacenar toda la información, ya que es una estructura excelente para manejar memoria ya que elimina el nodo directamente y libera memoria a la misma vez, haciéndolo una solución muy dinámica. También opté por la lista simplemente ligada ya que nunca había

programado una C y me quise poner el reto de programarla en C debido a los punteros y mandarlo como referencia, lo cual fue un éxito y ahora puedo decir que sé manejar punteros mejor en C gracias a eso.

Explicación de los fragmentos esenciales de código

```
void showSubject(Subj subj) {
  printf("\n");
  printf("\tName: %s\n", subj.name);
  printf("\tID: %s\n", subj.ID);
  printf("\tNRC: %s\n", subj.NRC);
  printf("\tProfessor name: %s\n", subj.professorName);
  printf("\tAvailable days: \n");

  char days[7][50] = {
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
    "Sunday"
  };

  for (int i = 0; i < 7; ++i)
    if (subj.availableDays[i])
      printf("\t\t%s\n", days[i]);

  printf("\tStart time: %s\n", subj.startTime);
  printf("\tEnd time: %s\n", subj.endTime);
  printf("\tSection: %s\n", subj.section);
  printf("\n");
}
```

[1]Este método sirve para mostrar cada materia, no tiene mucho de complicado,

[1]

simplemente son impresiones sobre un struct, no retorna nada ya que se asume en mi código que si la vas a usar es porque la estructura es válida.

```
bool addSubject(Node** subjects) {
  Subj subj;
  readSubject(&subj);
  printf("%s\n", subj.name);
  return listAdd(subjects, subj);
}
```

2

[2] Función para agregar una materia a mi lista simplemente ligada, tiene como parámetro un puntero hacia un puntero, lo cual es interesante porque simula una referencia hacia un puntero, a esta función le mando la dirección de memoria de la lista y funciona perfectamente.

```
bool modifySubject(Node** subjects) {
  Node* it;
  char NRC[50];

  printf("NRC: ");
  scanf("%s", NRC);

  for (it = *subjects; it; it = it->next)
    if (strcmp(it->subj.NRC, NRC) == 0) {
      char op;
      do {
        printf("1) Name\n");
        printf("2) ID\n");
        printf("3) Professor name\n");
        printf("4) Available days\n");
        printf("5) Start time\n");
        printf("6) End time\n");
        printf("7) Section\n");
        printf("0) Exit\n");
        printf("Attribute to modify: ");
        scanf(" %c", &op);
        getchar();
```

[3]

```
bool showAllSubjects(Node* subjects) {
  for (Node* it = subjects; it; it = it->next)
    showSubject(it->subj);

  return subjects;
}
```

[4]

[3] Esta función es para modificar un elemento de la lista, no pego todo el código ya que es muy largo y creo que se entiende que simplemente por cada opción, voy a buscar el elemento y modificar su atributo, esta función busca por el NRC, el cual es la llave primaria de la materia.

[4] Esta función muestra todos los elementos utilizando la función previamente discutida sobre mostrar un elemento, simplemente itera sobre la lista y muestra todos los contenidos.

```
bool showAllSubjectsOrdered(Node* subjects) {
  Node* copy = NULL;

  for (Node* it = subjects; it; it = it->next)
    listAdd(&copy, it->subj);

  for (Node* it = copy; it; it = it->next) {
    Node* mn = it;
    for (Node* it2 = it->next; it2; it2 = it2->next)
      if (strcmp(it2->subj.name, mn->subj.name) < 0)
        mn = it2;

    Subj temp = it->subj;
    it->subj = mn->subj;
    mn->subj = temp;
  }

  showAllSubjects(copy);

  return copy;
}
```

[5]

[5] Esta función crea una copia de la lista y la ordena utilizando el método de selección. La complejidad es O(n^2), sin embargo, es fácil y corto de codificar y cumplía perfectamente el objetivo. Después de ordenarla, muestro todos los elementos en la copia ordenada.

```
bool findSubject(Node* subjects) {
  Node* it;
  char NRC[50];

  printf("NRC: ");
  scanf("%s", NRC);

  for (it = subjects; it; it = it->next)
    if (strcmp(it->subj.NRC, NRC) == 0) {
      showSubject(it->subj);
      return true;
    }
  return false;
}
```

6

```
bool tempDelSubject(Node** subjects, Node** deletedSubjects) {
  char NRC[50];

  printf("NRC: ");
  scanf("%s", NRC);

  for (Node* it = *subjects; it; it = it->next)
    if (strcmp(it->subj.NRC, NRC) == 0)
      return listAdd(deletedSubjects, it->subj) && listDelete(subjects, NRC);

  return false;
}
```

7

[6] Simplementa busca una materia por NRC, si la encuentra, la muestra y regresa un booleano si la encontró o no.

[7] Este simplemente recibe punteros hacia las dos listas, una para las materias y otra para la “basura”, simplemente agrego el elemento a la basura para recuperarlo posteriormente y lo borro de la lista actual.

```
bool recoverSubject(Node** subjects, Node** deletedSubjects) {
  char NRC[50];

  printf("NRC: ");
  scanf("%s", NRC);

  for (Node* it = *deletedSubjects; it; it = it->next)
    if (strcmp(it->subj.NRC, NRC) == 0)
      return listAdd(subjects, it->subj) && listDelete(deletedSubjects, NRC);

  return false;
}
```

[8]

```
bool permDelSubject(Node** subjects) {
  char NRC[50];

  printf("NRC: ");
  scanf("%s", NRC);

  for (Node* it = *subjects; it; it = it->next)
    if (strcmp(it->subj.NRC, NRC) == 0)
      return listDelete(subjects, NRC);

  return false;
}
```

[9]

[8] Función para recuperar una materia, simplemente busca en la basura por un NRC, y si lo encuentra, lo agrega de vuelta a la lista de materias y lo remueve de la basura.

[9] Esta función simplemente hace lo mismo, pero no la manda a la basura, por lo que ya es imposible de recuperar la materia después de haber sido borrada.

```
bool listAdd(Node** head, Subj subj) {
  if (!*head) {
    *head = (Node*)malloc(sizeof(Node));
    (*head)->subj = subj;
  } else {
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->subj = subj;
    temp->next = *head;
    *head = temp;
  }
  return *head;
}
```

[10]

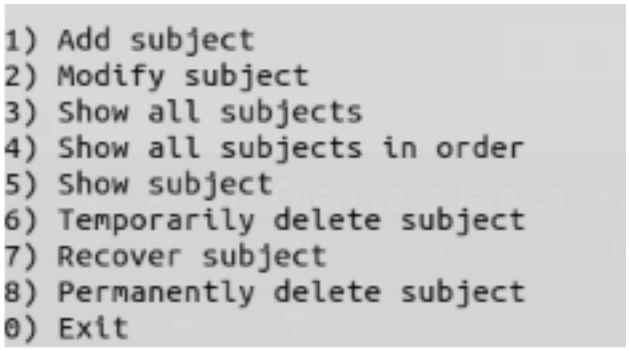
```
bool listDelete(Node** head, char NRC[]) {
  if (strcmp((*head)->subj.NRC, NRC) == 0) {
    Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    return true;
  }

  for (Node* it = *head; it && it->next; it = it->next)
    if (strcmp(it->next->subj.NRC, NRC) == 0) {
      Node* temp = it->next;
      it->next = it->next->next;
      free(temp);
      return true;
    }

  return false;
}
```

[11]

[10] Función para agregar un elemento a mi lista, si la lista está vacía, reserva nueva memoria para que nuestra cabeza sea alocada hacia ella, en caso contrario, la inserta al principio ya que es más rápido que colocarla al final, la diferencia es O(1) vs O(n), esto provoca que los elementos queden en el orden de una queue, sin embargo no importa ya que la lista es para almacenar y su orden no importa.

[11] Función para borrar un elemento de la lista, busca por el NRC, si el primer elemento corresponde, apunta ese nodo en particular a NULL, y si no, busca el siguiente

Capturas de pantalla del programa en ejecución (colores invertidos, si el fondo originalmente es negro)



[12]

nodo para ver si corresponde, y si es así, entonces re-enlaza los enlaces hacia el siguiente del siguiente, o sea que se “brinca” el nodo para ser eliminado. En ambos casos, libero la memoria después de remover, haciendo el programa eficiente en memoria.

[12] Menú principal

```
Name: 3
ID: 3
NRC: 3
Professor name: 3
Available days:
        Monday
        Tuesday
        Wednesday
        Thursday
        Friday
        Saturday
        Sunday
Start time: y
End time: y
Section: y


Name: 4
ID: 4
NRC: 4
Professor name: 4
Available days:
        Monday
        Wednesday
        Friday
        Saturday
        Sunday
Start time: 4
End time: 4
Section: 4


Name: 2
ID: 2
NRC: 2
Professor name: 2
Available days:
        Monday
        Tuesday
        Thursday
        Saturday
Start time: 2
End time: 2
Section: 2


Name: 1
ID: 1
NRC: 1
Professor name: 1
Available days:
        Monday
        Wednesday
```

[13]

[13] Añadiendo varias materias, agregué valores “dummy” solo para demostrar su funcionamiento.

```
Name: 1
ID: 1
NRC: 1
Professor name: 1
Available days:
        Monday
        Wednesday
        Friday
        Sunday
Start time: y
End time: y
Section: y


Name: 2
ID: 2
NRC: 2
Professor name: 2
Available days:
        Monday
        Tuesday
        Thursday
        Saturday
Start time: 2
End time: 2
Section: 2


Name: 3
ID: 3
NRC: 3
Professor name: 3
Available days:
        Monday
        Tuesday
        Wednesday
        Thursday
        Friday
        Saturday
        Sunday
Start time: y
End time: y
Section: y


Name: 4
ID: 4
NRC: 4
Professor name: 4
Available days:
```

[14]

[14] Ordenamiento por nombre de materia, cabe decir que solamente fue una copia la que se ordenó y no la lista original, esto para conservar la propiedad de que solo estoy mostrándolo ordenado, no necesariamente ordenando la lista original.

```
        Name: 3
        ID: 3
        NRC: 3
        Professor name: 3
        Available days:
                Monday
                Tuesday
                Wednesday
                Thursday
                Friday
                Saturday
                Sunday
        Start time: y
        End time: y
        Section: y


        Name: 4
        ID: 4
        NRC: 4
        Professor name: 4
        Available days:
                Monday
                Wednesday
                Friday
                Saturday
                Sunday
        Start time: 4
        End time: 4
        Section: 4


        Name: 1
        ID: 1
        NRC: 1
        Professor name: 1
        Available days:
                Monday
                Wednesday
                Friday
                Sunday
        Start time: y
        End time: y
        Section: y

1) Add subject
```

Recuperación del elemento 2 de la basura, cabe notar que el elemento está en un diferente orden del que estaba originalmente, ya que así funciona la lista que los almacena.

[15] Eliminación temporal de la materia con el nombre “2”.

```
Name: 2
ID: 2
NRC: 2
Professor name: 2
Available days:
        Monday
        Tuesday
        Thursday
        Saturday
Start time: 2
End time: 2
Section: 2


Name: 3
ID: 3
NRC: 3
Professor name: 3
Available days:
        Monday
        Tuesday
        Wednesday
        Thursday
        Friday
        Saturday
        Sunday
Start time: y
End time: y
Section: y


Name: 4
ID: 4
NRC: 4
Professor name: 4
Available days:
        Monday
        Wednesday
        Friday
        Saturday
        Sunday
Start time: 4
End time: 4
Section: 4
```

[16]

[16] Eliminación permanente la materia 1

```
Name: 2
ID: 2
NRC: 2
Professor name: 2
Available days:
        Monday
        Tuesday
        Thursday
        Saturday
Start time: 2
End time: 2
Section: 2


Name: 3
ID: 3
NRC: 3
Professor name: 3
Available days:
        Monday
        Tuesday
        Wednesday
        Thursday
        Friday
        Saturday
        Sunday
Start time: y
End time: y
Section: y


Name: 4
ID: 4
NRC: 4
Professor name: 4
Available days:
        Monday
        Wednesday
        Friday
        Saturday
        Sunday
Start time: 4
End time: 4
Section: 4
```

[17]

[17] Se intenta recuperar el elemento número 1, sin embargo, ya no existe, porque fue eliminado permanentemente.

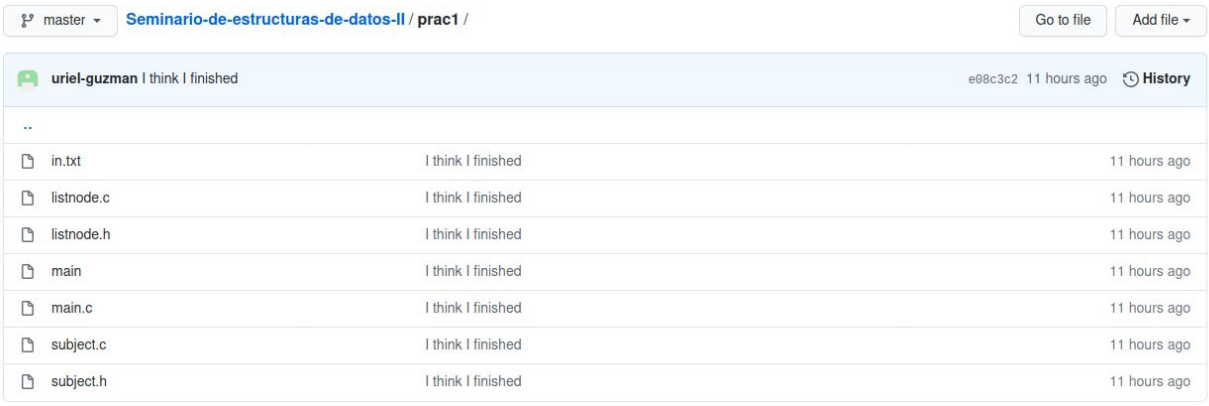
```
Name: 2
ID: 2
NRC: 2
Professor name: 2
Available days:
        Monday
        Tuesday
        Thursday
        Saturday
Start time: 2
End time: 2
Section: 2


Name: 3
ID: 3
NRC: 3
Professor name: 3
Available days:
        Monday
        Tuesday
        Wednesday
        Thursday
        Friday
        Saturday
        Sunday
Start time: y
End time: y
Section: y


Name: 99
ID: 98
NRC: 4
Professor name: 97
Available days:
Start time: 96
End time: 95
Section: 94
```

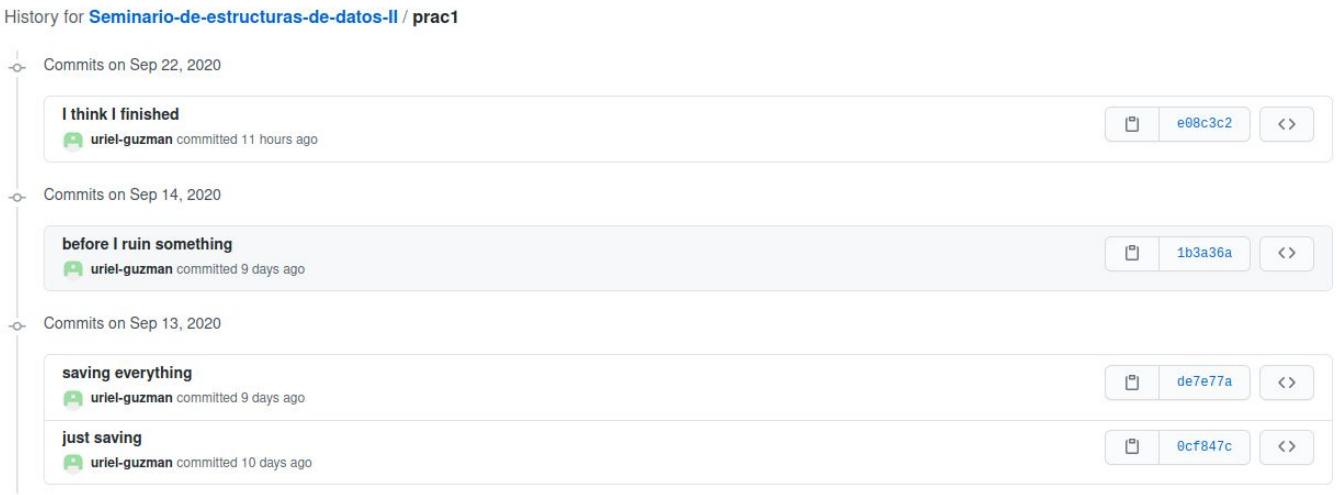
[18]

[18] Se modifica la última matria con el NRC de 4, cabe decir que el NRC no fue disponible para modificar, ya que es su llave primaria y no debería modificarse.

Evidencia de trabajo de un control de versiones (si aplica)

Se puede evidenciar el trabajo en mi Githun con el siguiente enlace: https://github.com/uriel-guzman/Seminario-de-estructuras-de-datos-II/tree/master/prac1





19

[19] Unas capturas sobre el historial de cambios que dediqué a esta tarea, junto al nombre de los commits que hice. Estoy consciente de que los mensajes de los commits no son útiles para un entorno de trabajo ya que no son serios y no describen nada, pero principalmetne utilizo Github para respaldo de tareas.

Problemas encontrados (enfrentados), y cómo se resolvieron

Algunos problemas encontrados fueron los siguientes:

Me fue un reto utilizar la lista simplemente ligada en C, principalmente por el paso de parámetros, estoy bastante cómodo con la teoría e implementación en C++ de esta estructura de datos, sin embargo, no había programado una en C. Los principales problemas fueron que no entendía muy bien cómo funcionaban los punteros, pero ahora estoy mucho más cómodos con ellos.

Otro reto fue hacer que funcionaran mis lecturas de caracteres con espacios, tuve varios problemas, enter ellos, problemas con el scanf("%[^\n], str), el cual estaba ocasionando bugs debido a que deja un salto de línea flotando, así que recurrí a gets, pero leí en Stack Overflow que no era seguro, y finalmente opté por fgets, el cual parece ser la opción más segura para leer secuencias de caracteres con espacios debido a que te pide el tamaño máximo del buffer.

Recomendaciones / sugerencias para mejorar la práctica

Mis recomendaciones serían que se aclarara un poco más lo que se pide para la práctica, ya que algunas cosas parecí entenderlas y después resultó que trabajé de más al implementar una lista simplemente ligada, también fue mi culpa por por preguntar.

Conclusiones

Fuera de lo anterior dicho, considero que fue una práctica útil que me hizo mejor programador en C, aprendí cosas nuevas y útiles y diría que fue una buena práctica para poner a pruba mis conocimientos de punteros y programación estructurada.

Referencias bibliográficas en APA

- paxdiablo (2009, Aug 08). How do you allow spaces to be entered using scanf? . Stack Overflow. https://stackoverflow.com/questions/1247989/how-do-you-allow-spaces-to-be-entered-using-scanf
- mycodeschool. (2012, Jan 13). Introduction to pointers in C/C++. Youtube. https://www.youtube.com/watch?v=h-HBipu_1P0