UNIVERSIDAD
NACIONAL DEL NORDESTE

PROYECTO INTEGRADOR "TO DO"

grupo 3 (turno noche)
Velozo Agustin
Villalba Renata
Herrera Fabrizio
Saravia Joaquin
Sosa Camila

Algotitmos y estructura de datos 2

Introducción

El siguiente informe integrará todos los conceptos aprendidos a lo largo de la asignatura Algoritmos y Estructura de Datos II, como también, siendo implementados en la creación de un proyecto, creando una aplicación que permita crear una CheckList para ser utilizada como una “Lista To-Do''. Como motivo principal del desarrollo de la aplicación, se aspiró a solventar la falta de buena gestión del tiempo.

Si bien existen gran variedad de técnicas y estrategias que pueden ayudarnos y sobre las que se puede encontrar mucha información fácilmente, pero prácticamente ninguna tan sencilla y tan eficiente como mantener una lista de tareas.

Una lista de tareas es algo que todo el mundo puede usar y no es nada más que un lugar donde anotar todas las ideas o actividades que quieres desarrollar en otro momento, una lista de cosas pendientes.

Entonces a causa de que nuestro cerebro no es capaz de almacenar absolutamente todo lo que queremos recordar, necesitamos un soporte externo en el que volcar toda esa información (sea física o digital).

Por ello esta aplicación es capaz de crear CheckLists “padres” o “principales”, las cuales serían la tarea principal a cumplir, mientras que, a su vez, estas tareas tienen la capacidad de almacenar subtareas más pequeñas respecto a la principal. Cada uno de estos elementos de la checklist deberían poderse tachar, como una manera de representar que la tarea ya se ha realizado.

A su vez, cada uno de estos elementos puede ser creado, modificado y eliminado. Al finalizar el programa, estos elementos serán almacenados en una pequeña base de datos local realizando uso de los archivos.

Para lo cual estaremos aplicando en la “Lista To-Do“ las nuevas estructuras de datos y técnicas de programación conocidas de la asignatura, para experimentar las diferentes formas de utilizarlas y cómo pueden combinarse en diversas formas.

Aplicando temas como Listas enlazadas, Corte de Control, se debería implementar también Recursividad, Optimización y Eficiencia, la Gestión de módulos y librerías, entre otros. Daremos algún ejemplo de Listas, sus funciones más notorias, además se presentarán evidencias de corte de control, así como también se identificarán la gestión de módulos y librerías con una utilización precisa, clara y objetiva.

Desarrollo

Reuniones y comunicación

Para la realización de este proyecto nos hemos enfocado en usar distintos medios de comunicación para organizarnos probando diferentes herramientas para identificar con cual nos sentíamos más cómodos y ágiles. Terminamos optando por varios de ellos más específicamente: WhatsApp, Meet y Discord. Siendo que estos nos resultaron más eficientes y versátiles para una óptima comunicación.

Propuestas para el proyecto.

En esta etapa se propusieron distintas ideas acerca de lo que queríamos hacer como proyecto. Llegando a tener como propuestas distintos juegos, como “la viborita” o “el truco” en caso de necesitar cambiar de idea, al final terminamos optando por desarrollar una aplicación que permite crear distintas CheckLists para ser utilizada como una “Lista To–Do”.

Desarrollo de la propuesta.

Al comenzar a pensar en cómo implementar la idea de proyecto con los conocimientos nuevos de la asignatura fue necesario estar en constante formación. Ya que no se puede “HACER” sin antes “SABER HACER”, en este proceso de formación a través de las clases y ejercicios prácticos. Cada uno de nosotros fue aprendiendo a su ritmo, Algunos ya tenían una serie de conocimientos previos mientras que otros recién los fueron conociendo durante la cursada. Esta pequeña brecha nos llegó, en cierto modo, a retrasar en el transcurso del proyecto siendo que todos queríamos ser parte de la realización y nadie se quería quedar atrás, quienes ya tenían más avanzada su práctica fueron tomando la iniciativa en la codificación e implementación en lenguaje C del programa. De igual manera eso no evitó que todos

pudiéramos participar en los momentos de tomar decisiones sobre el rumbo del proyecto o de buscar soluciones ante problemas que fueron surgiendo.

- Análisis: ¿Qué queríamos lograr? ¿Que necesitamos para eso?:

Desarrollar un programa que pueda mantenerse por sí mismo en el que se apliquen las listas enlazadas como herramienta principal para almacenar todos los datos temporales durante su ejecución, mientras que también, un sistema de almacenamiento en archivos, siendo cada elemento de la lista, uno de sus registros.

Diseño:

El desarrollo conceptual del “COMO” se basa en las listas enlazadas y en que cada lista enlazada herede un “Hijo” (como sub-tarea) y un “Siguiente”, similar a la estructura de un árbol binario.

- Cada elemento debe tener un título o nombre y una descripción que podría contener explícitamente la tarea a realizar o detalles de esta como el lugar o el horario. A su vez en cada tarea también habría subtareas para completar la misma, se marcaria con un carácter el estado de las tareas siendo “X” el cual indica que la tarea se ha finalizado y “O” indicando que la tarea aún está pendiente. De este modo, cada que la tarea principal sea completada, las sub-tareas también se completan, y a la inversa.

Por ejemplo: Una tarea donde se muestren los pendientes de la asignatura

Titulo	Descripcion	Estado
Tarea Principal	Pendientes AED	O (pendiente)
Subtarea 1	TP 5 TAD	X (finalizado)
Subtarea 2	Realizar Teorico 6	X (finalizado)
Subtarea 3	TP 7, Complementarios	X (finalizado)
Subtarea 4	TP Integrador	O (pendiente)

Algoritmos 2 de un alumno:

Codificación

Transcripción de la idea a ciertas instrucciones a interpretar por la computadora, escritas respetando la sintaxis de un lenguaje de programación, precisamente C, obteniéndose así el programa o código fuente. Teniendo ya en claro lo que queríamos lograr y definiendo los requerimientos necesarios, la estructura principal y el resultado que esperábamos fuimos pacientemente llevándolo a código dando así inicio a la etapa de codificación.

Ejecución y depuración:

Durante la misma se presentaron inconvenientes menores los cuales detallaremos a continuación:

1) Al momento de comenzar a implementar las listas enlazadas, el pasaje por parámetros a ciertas funciones provocaba un error, en otras palabras, no almacenaba en la lista el dato que se ingresaba. Eso se solucionó fácil, buscando

donde era que se encontraban los parámetros mal colocados, implementando doble puntero en cada uno de ellos.

2) Cuando se pudo ingresar y el display, es decir, lo que se imprimía por pantalla, era correcto, se quiso mejorar la modularización, implementando archivos TAD. Allí el tipo de elemento almacenado en la lista empezó a hacer conflicto con la clase principal. Para poder solucionar esto, verificamos todo el código en búsqueda de una función o declaración errónea, y borramos todo lo relacionado al tElemento, dejándolo a eso, por un lado, y solo las funciones de navegación en el archivo main.

3) Al implementar la navegación con las teclas W y S, el display no se actualizaba, es decir, todos los elementos de la lista se quedaban marcados. Para arreglar eso, se pasó por parámetro la posición.

4) Al aplicar el almacenamiento de las listas enlazadas, no hubo problema, lo que provocó problema fue a la hora de leer estos datos almacenados. Al ser una lista que se ingresa por el principio, los nodos hijos, en lugar de ser leídos como hijos de sus respectivos padres, se leían como hijos del padre siguiente al suyo. Eso se arregló pasando por parámetro siempre la primera posición a la hora de leer un hijo del archivo. Tras esto, el programa funcionaba bien, aunque el inconveniente pasó a ser que, cada vez que corría el programa, los elementos invierten su orden. Como solución, se aplicaron funciones recursivas, las cuales recorren la lista enlazada, almacenando los elementos desde el final hacia el principio.

5) Al principio, cuando queríamos eliminar un nodo, era posible eliminar hasta el elemento que permitía crear uno nuevo, por lo que el programa quedaba obsoleto y terminaba por retornar un error. Además, al eliminar el último elemento, su nombre pasaba al elemento creador. Para solucionar estos inconvenientes,

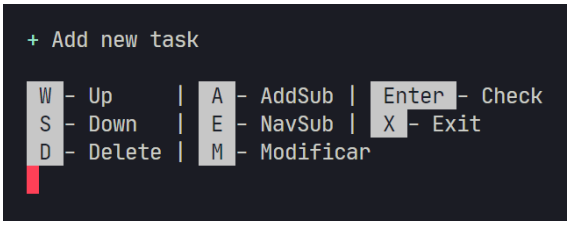
delimitamos la capacidad del método eliminar, de modo a que, si la posición en la que se encuentra es la última, quiere decir que estaba en el elemento creador, por lo que no podrá eliminarse.

MANUAL DEL USUARIO:

Para la mejor interpretación por parte del usuario, se creó un manual en el que se explican explícitamente cada una de las funcionalidades de cada tecla y cómo es que se debe utilizar la aplicación.

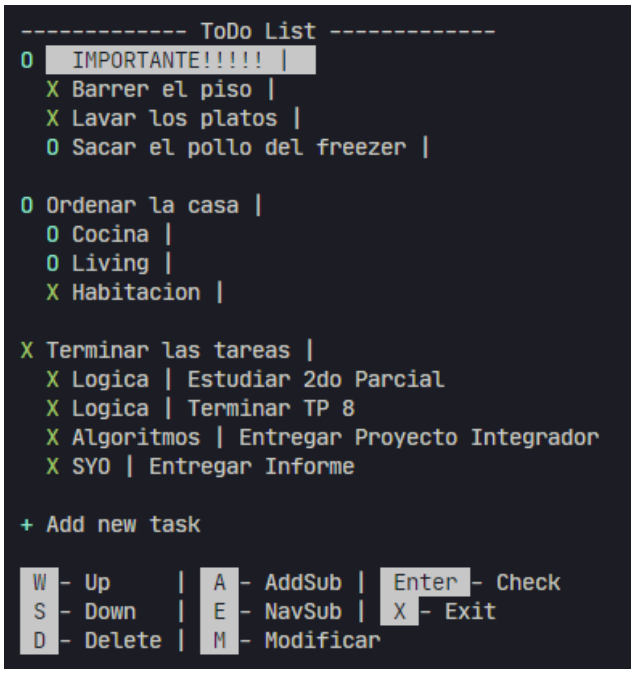
Al ejecutar nos encontraremos con la siguiente interfaz , y en ella se detectan una serie de opciones :

Nombre	Tecla	Descripción
Subir	W	Desplazarse hacia arriba
Bajar	S	Desplazarse hacia abajo
Borrar	D	Borrar Padre
AddSub	A	Agregar SubTarea
NavSub	E	Navegar entre SubTareas
Modificar	M	Modificar Elemento
Check	Enter	Tachar tarea
Exit	X	Atras - Salir



Si desea agregar una tarea se deben seguir los siguientes pasos:

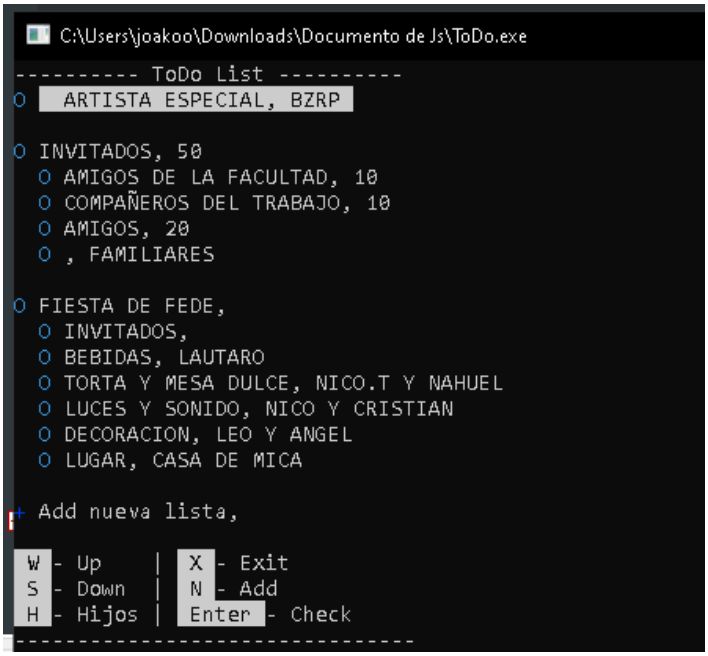
1) Para añadir una tarea principal, se debe dirigir al elemento “Add new task”. En caso de querer añadir una SubTarea, se debe parar sobre la tarea principal y presionar “N”.
2) Ingrese el “NOMBRE DE LA TAREA” (o subtarea) y presione Enter.
3) Ingrese el “DESCRIPCIÓN DE LA TAREA” (o subtarea) y presione Enter.



A continuación, unos usos prácticos para usar nuestro To-Do Maker:

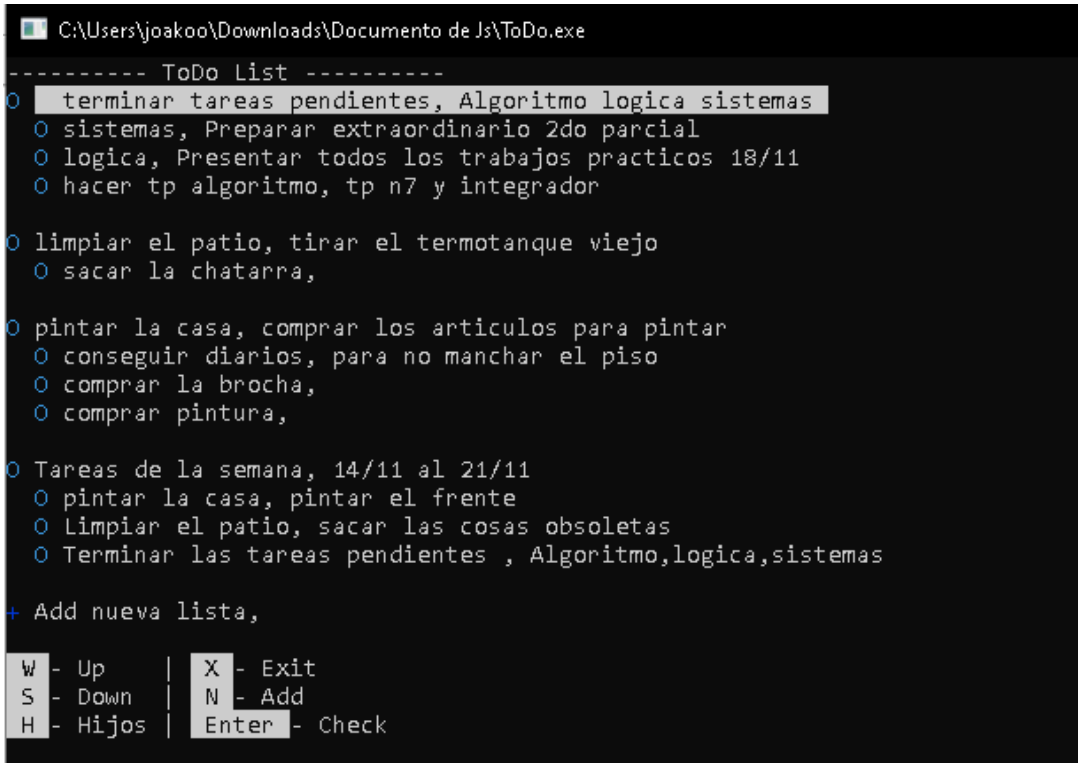
Ejemplo Nº1

Se acerca el cumpleaños de FEDE y sus Amigos planean organizar una fiesta sorpresa para él para dividirse las tareas utilizaron TO DO MAKER y les resultó mucho más sencillo llevar cuenta de lo que hacía falta para la fiesta.



Ejemplo Nº2

Se viene una semana complicada para David y debe organizarse para poder acordarse de todas las cosas que debe hacer durante el transcurso de esta.



Etapa 2: Estructuras compuestas enlazadas (implementación dinámica mediante listas enlazadas)

Algunas de las funciones que estuvimos utilizando son como;

1- inicializar lista:

```
50  // Metodos genericos de Listas Enlazadas
51  void inicializarLista(tLista * pTodoList){
52      pTodoList = NULL;
53  }
```

2- Insertar Primer elemento:

```
void insertarPrimerHijo(tLista * pPadre, tElemento pElem){
    tHijo * nuevoHijo;

    nuevoHijo = (tHijo*) malloc(sizeof(tHijo));

    nuevoHijo->elemento = pElem;
    nuevoHijo->siguiente = NULL;

    pPadre->hijo = nuevoHijo;

    printf("a");

}
```

3- insertar un elemento:

```
81  void insertarHijo(tLista * pPadre, tElemento pElem){
82
83      if(pPadre->hijo == NULL){
84          insertarPrimerHijo(pPadre, pElem);
85          return;
86      }
87
88      tHijo * nuevoHijo = (tHijo*) malloc(sizeof(tHijo));
89
90      nuevoHijo->elemento = pElem;
91      nuevoHijo->siguiente = pPadre->hijo;
92
93      pPadre->hijo = nuevoHijo;
94
95  }
96
```

4- Eliminar un nodo:

```
327  void eliminarPadre(tLista ** pTodoList, int posPadre){
328
329      tLista * nodoAEliminar;
330
331      if(posPadre == 1){
332
333          nodoAEliminar = (*pTodoList);
334          (*pTodoList) = (*pTodoList)->siguiente;
335
336      }else{
337
338          tLista * aux;
339          aux = (*pTodoList);
340          int i;
341
342          for(i = 1; i < posPadre -1; i++){
343              aux = aux->siguiente;
344          }
345
346          nodoAEliminar = aux->siguiente;
347          aux->siguiente = nodoAEliminar->siguiente;
348
349      }
350
351      free(nodoAEliminar);
352      nodoAEliminar = NULL;
353
354  }
```

```
356  void eliminarHijos(tHijo ** pHijo){
357
358      if((*pHijo)->siguiente != NULL){
359          eliminarHijos(&(*pHijo)->siguiente);
360      }
361      free(*pHijo);
362
363      printf("Hijo eliminado \n");
364
365  }
```

5- Visualizar elementos:

```c
void mostrarHijos(tLista * pPadre, int hijoSelect){

    int cant = 1;

    tHijo * aux = pPadre->hijo;
    tElemento elemento;

    while(aux != NULL){

        elemento = aux->elemento;

        if(cant == hijoSelect){
            if(aux->elemento.isChecked){
                printf(BG_BLACK GREEN "  X ");
                printf(BG_WHITE BLACK "  %s | %s \t\n", elemento.datos.titulo, elemento.datos.descripcion);
            }else{
                printf(BG_BLACK CYAN "  O ");
                printf(BG_WHITE BLACK "  %s | %s \t\n", elemento.datos.titulo, elemento.datos.descripcion);
            }
        }else{
            if(aux->elemento.isChecked){
                printf(BG_BLACK GREEN "  X ");
                printf(WHITE "%s | %s \t\n", elemento.datos.titulo, elemento.datos.descripcion);
            }else{
                printf(BG_BLACK CYAN "  O ");
                printf(WHITE "%s | %s \t\n", elemento.datos.titulo, elemento.datos.descripcion);
            }
        }

        cant++;

        aux = aux->siguiente;
```

Etapa 3: Corte de control

El corte de control se utiliza cuando pueden detectarse, dentro del archivo ordenado, grupos de registros que corresponden a un mismo campo de control. Podemos detectar cuando finalizan los elementos de entrada de un grupo antes de comenzar el siguiente.

Es la razón por la cual fue incorporada al proyecto “To-Do” ya que nos ayudaba a identificar a cada subtarea o tarea como elementos de entrada (registros), así, logra distinguir una subtarea de sus hijos.

Etapa 4: Recursividad

La recursividad puede aplicarse en los campos donde la definición de la construcción de una “cosa” puede llevarse a cabo a partir de la “cosa” en sí.

Es por eso que la utilizamos, definiendo un algoritmo a partir de sí mismo. Lo utilizamos por ejemplo en una función que fue la de “eliminarHijos(tHijo **)”, donde eliminamos las subtareas, empleando una estructura de control condicional. O también, a la hora de guardar los elementos de la lista en el archivo, guardando desde el último elemento hacia el primero.

```
void eliminarHijos(tHijo ** pHijo){

    if((*pHijo)->siguiente != NULL){
        eliminarHijos(&(*pHijo)->siguiente);
    }
    free(*pHijo);

    printf("Hijo eliminado \n");

}
```

Etapa 5: Tipo Abstracto de datos.

El concepto de abstracción se refiere a organizar la realidad para hacerla más comprensible. Es por ello que la implementamos, para centrar la atención en las

características más relevantes a los problemas que nos topamos durante el desarrollo del proyecto. Creamos varias librerías como “colors.h”, “element.h” y “toDoStruct.h” para disponerlas del modo más conveniente para su mejor comprensión.

Al usarlas facilitamos la reusabilidad del código, además de que, si necesitábamos hacer modificaciones, íbamos directamente hasta el TAD y aplicamos los cambios, y eso no llegaba a afectar al programa que lo utilizaba.

En el caso de la librería “colors.h”, se la utilizó en la librería “element.h”, como en “toDoStruct.h” y por último en el programa principal de “ToDo.c” con la función de brindar de colores a la hora de imprimir por pantalla aportando un toque más atractivo y estético, acción que se realiza y argumentos a través de los cuales toma información y devuelve el color.

Librería “colors.h “:

```
//═══Color font code═══/
#define BLACK   "\x1B[30m"
#define RED     "\x1b[31m"
#define GREEN   "\x1b[32m"
#define YELLOW  "\x1b[33m"
#define BLUE    "\x1b[34m"
#define MAGENTA "\x1b[35m"
#define CYAN    "\x1b[36m"
#define WHITE   "\x1B[37m"
#define ORANGE  "\x1B[38;2;255;128;0m"
#define ROSE    "\x1B[38;2;255;151;203m"
#define LBLUE   "\x1B[38;2;53;149;240m"
#define LGREEN  "\x1B[38;2;17;245;120m"
#define GRAY    "\x1B[38;2;176;174;174m"
#define RESET   "\x1b[0m"
```

Librería “ToDoStruct.h” :

```
// ------------------------ Definicion de tipos/estructuras
typedef struct nodoHijo{
    tElemento elemento;
    struct nodoHijo * siguiente;
}tHijo;

typedef struct nodo{
    tElemento elemento;
    struct nodo *siguiente;
    tHijo * hijo;
}tLista;

// ------------------------ Prototipado de metodos
void inicializarLista(tLista *);
int dimensionLista(tLista *);

// Metodos para los nodos 'principales' o 'padres'
void insertarPrimerNodo(tLista **, tElemento);
void insertarNodo(tLista **, tElemento);
void eliminarPadre(tLista **, int);
tLista * buscarNodo(tLista *,int);

// Metodos para los nodos 'hijos'
void insertarPrimerHijo(tLista *,tElemento);
void insertarHijo(tLista *,tElemento);
int dimensionHijo(tHijo *);
tHijo * buscarHijo(tLista *, int);
void eliminarHijos(tHijo **);

void mostrarTodos(tLista *, int);
void mostrarHijos(tLista *, int);
void padreSeleccionado(tLista * , int, int);
tElemento modificarElemento(tElemento);
```

Librería “element.h”:

```c
typedef char tString[200];

typedef struct{
    tString titulo;
    tString descripcion;
}tDatos;

typedef struct{
    bool isChecked;
    bool esPadre;
    tDatos datos;
}tElemento;

tElemento crearElemento(int);
tElemento constructorElemento(int esPadre, int check, tString titulo, tString descripcion);
```

Etapa 6: Lectura y almacenamiento de archivos

Para leer el archivo, lo abrimos con el método de “rb” (read binary) y utilizamos el método de corte de control para identificar si es padre o hijo.

Para guardar, sobrescribimos (wb) el archivo utilizando una función recursiva que y recorre la lista enlazada hasta el final y regresa guardando cada uno de sus elementos como así también con las subtareas (hijos).

Conclusión

Durante este trayecto de llevar a cabo la realización de este proyecto, hemos aprendido a construir programas que requieran del uso de estructuras de datos compuestas, manipulación de archivos, el uso de recursividad, construir y desarrollar tipos de datos abstractos, seleccionar y usar diferentes tipos de estrategias para la lectura y almacenamiento de archivos, Además de saber combinar las estructuras y las técnicas para ser aplicadas de manera efectiva.

Como resultado de identificar y haber podido resolver los inconvenientes que nos surgían, adquirimos un mejor desempeño a la hora de aprender de forma autónoma, continua y de manera colaborativa. Nos ayudó en la forma de comunicarnos, a aprender diferentes modos de comunicación permitiéndonos descubrir, indagar, investigar sobre su manejo, como el funcionamiento, además de desarrollar habilidades respecto de la lógica, análisis y reflexión.

Estos aspectos mencionados son de importancia para nosotros ya que este proyecto nos permitió adquirir nociones básicas para auto gestionar proyectos de programación de software futuras.

Bibliografía

- https://www.youtube.com/watch?v=-Shr2s0gYao
- https://www.youtube.com/watch?v=WxoGvBzWuGs
- https://www.youtube.com/watch?v=lVmjX34TKFg
- https://www.youtube.com/watch?v=gwHbGOyjGmo
- https://virtual-moodle.unne.edu.ar/mod/resource/view.php?id=49839
- https://virtual-moodle.unne.edu.ar/mod/resource/view.php?id=349075
- https://virtual-moodle.unne.edu.ar/mod/resource/view.php?id=45768
- https://virtual-moodle.unne.edu.ar/mod/resource/view.php?id=364151
- https://www.infor.uva.es/~mserrano/EDI/cap2.pdf