# UNIVERSIDAD AUTONOMA METROPOLITANA

## Programación Avanzada

## Práctica 6:

## "TDA – Lista Secuencial"

**Fecha de Elaboración:**
Martes, 29 de julio del 2008

**Fecha de Entrega:**
Jueves, 7 de agosto del 2008

# Contenido

# AVISOS:

- No se reciben reportes de las prácticas después de la fecha de entrega.
- En la página se encuentra el formato del reporte para las prácticas en donde se describen cada uno de los apartados que deberá considerarse. Se deberá seguir dicha plantilla para el desarrollo del reporte de las prácticas.
- Enviar a la cuenta de correo un paquete zipeado que contendrá:
  - Códigos fuentes (extensión **.c**).
  - Códigos Objetos (extensión **.o** )
  - Ejecutables (sin extensión )
  - En el reporte incluir por actividad (extensión **.doc)**:
    - **Desarrollo** de la actividad.
    - **Resultados** de la actividad.
- En el subject del correo escribir: ***Práctica 6 - <nombre completo>***.
- **Se entregará el reporte y los códigos fuentes impresos.**

# 1 Objetivos

1) Utilizar el diseño de programación ascendente y descendente en el enfoque de **programación modular** (enfoque "divide y vencerás") para la resolución de problemas mediante el uso de **módulos** que serán traducidos a funciones (y/o procedimientos) en el lenguaje C.
2) Utilizar la herramienta **make** para optimizar y automatizar la creación de proyectos en C bajo el ambiente de Linux
3) Implementar la especificación del **TDA Lista Secuencial** haciendo uso de un arreglo de elementos definido por el usuario.
4) Definir e implementar el **conjunto de valores** que especificaran el TDA Lista.
5) Implementar el **conjunto de operaciones** que permiten manipular el TDA.

# 2 Introducción

En el desarrollo de esta práctica se requiere que el alumno realice cada una de las actividades que se le indica.

Definir los algoritmos de búsqueda y ordenamiento iterativos que se vieron en clase.

Reutilización de los algoritmos de búsqueda y ordenamiento para datos básicos almacenados en un arreglo.

Reutilización de funciones que permiten la extracción de datos de un archivo de texto.

## 2.1 Complementos

Para completar el reporte de la práctica el alumno deberá de efectuar una breve investigación de los siguientes puntos (máximo 3 párrafos por punto):

- Utilización de la herramienta make en programas C para la creación de proyectos en C bajo el ambiente de Linux.
- El operador **?** en C.

## 2.2 Ayuda complementaria

- Referencia de la bibliografía contenida en la planeación del curso de Programación Avanzada.
- Notas del Curso de Introducción a al Programación 08-I.
- Guía Básica para el uso de C.
- La ayuda de la línea de comandos de Linux: ***man make***.
- Practica 5 publicada en la siguiente página del curso programación avanzada. URL:http://uamisoft.izt.uam.mx/uamisoft/doku.php?id=amm:cursos:pa:practicas:practica5

# 3 Desarrollo

La sección del desarrollo de la práctica se divide en una serie de actividades que permiten englobar, desarrollar y alcanzar cada uno de los objetivos de la práctica. Es necesario leer con atención cada una de las actividades a desarrollar.

## 3.1 Antecedentes: Creación de proyectos en C en un ambiente Linux con *Make*

**Un proyecto**, es un programa ya dividido en varias unidades de traducción (codigo fuentes). Un programa con estas características, es difícil darle mantenimiento debido a que, para generar una nueva versión hay que establecer las dependencias entre ellas para saber que unidades se tienen que compilar y después ligar para generar la nueva versión.

Para elaborar un proyecto se requiere definir los siguientes archivos:

**Librería.h**:

Es un archivo de encabezado que contienen los elementos particulares que se utilizaran en dentro del proyecto. Se pueden incluir las librerías de uso común, declarar constantes, variables globales, definir nuevos tiempo de datos, especificar las declaraciones de las funcione que se utilizaran a lo largo del proyecto, etc.

**Archivos.c**:

Contienen las funciones que se definieron en el archivo ***Librería.h,*** dichas funciones pueden estar definidas en distintitos archivos C.

La generación de proyectos requiere de efectuar la compilación de cada uno de los archivos fuentes, posteriormente el ligado para así obtener el ejecutable completo, estas actividades se pueden automatizar con una herramienta llamada **make**. La herramienta **make** busca optimizar y automatizar la construcción de programas.

Make utiliza un archivo, **makefile**, en donde se especifican las dependencias entre las unidades de traducción (entre archivos) y una vez realizado esto, se ejecuta con el comando:

]$ make

El siguiente código es el contenido de un archivo ***makefile***:

```
#---------------------------------------------------------------------
#compilador
#---------------------------------------------------------------------
cc=gcc
#---------------------------------------------------------------------
# Variables
#---------------------------------------------------------------------
FUENTES = Saludos.c Despedidas.c Principal.c LibreriaPrueba.h
OBJS = Principal.o Saludos.o Despedidas.o

#---------------------------------------------------------------------
# Dependencias
#---------------------------------------------------------------------
Activ1: $(OBJS)
```

```
	cc -o $@ $(OBJS)

Principal.o:
	cc -c Principal.c   LibreriaPrueba.h
Despedidas.o:
	cc -c Despedidas.c
Saludos.o:
	cc -c Saludos.c

limpia:
	rm $(OBJS)
```

Todas las dependencias mostradas en la figura, se representan en el archivo **makefile** después de los dos puntos (:), y si los archivos de los cuales depende la etiqueta no existen o son más antiguos que la etiqueta, se verifican sus dependencias para (re)generarlos, antes de ejecutar el comando de la siguiente línea. Por ejemplo, si es la primera vez que vamos a compilar a través del **make**, en la línea:

```
Activ1: $(OBJS)
  cc -o $@ $(OBJS)
```

Activ1 es el nombre del ejecutable y depende de los objetos definidos con la macro OBJS. Si no existen se generan dichos archivos, para posteriormente ejecutar la siguiente línea y aso obtener el ejecutable.

## 3.2 Antecedentes: TDA Lista Secuencial - Especificación

### 3.2.1 Conjunto de valores.

Una lista sobre un Dominio D es una secuencia $a_1, a_2, \ldots a_n$ donde n es un número natural y $a_i \in D$.

Se dice que:

-El elemento $a_i$ ocupa la posición i. Donde las posiciones i son tal que $1 \leq i \leq n$.

- $a_i$ **precede** a $a_{i+1}$ ( $1 \leq i \leq n-1$ )
- $a_i$ **sucede** a $a_{i-1}$ ( $1 < i \leq n$ )

### 3.2.2 Conjunto de operaciones.

- **Crea_Lista ( )**
- **Inserta (L, elemento, posicion)**
- **Elimina (L, posicion)**
- **Busca (L, elemento)**
- **Longitud ( L )**
- **Fin ( L )**
- **Consulta ( L , posicion )**
- **Despliega ( L )**
- **Es_Vacia ( L )**
- **Es_Llena ( L )**
- **Haz_Nula ( L )**
- **Destruye ( L )**
- **Primer ( L )**
- **Resto (L )**

## 3.3 Actividad 1 "Creación de un proyecto y utilización de la herramienta Make"

*Con el desarrollo de esta actividad se alcanzará los objetivos 1 y 2 de la práctica.*

Cree una carpeta referente a la practica y la actividad: /Practica6/Actividad1/ y descargue los siguientes archivos a ser analizadazos:

**Saludos.c**

- ***void Saludo(int i)***
  Recibe como parámetro un entero para enviar a pantalla un mensaje y el contenido de dicho parámetro.
- ***int Saludos (int numSaludos )***
  De acuerdo al valor del parámetro, se invoca la función Saludos.

**Despedidas.c**

- ***void Despedida()***
  Envía a pantalla un mensaje.
- ***void Despedidas ( int Arr [], int longitud)***
  De acuerdo al valor del parámetro numSaludos, se invoca la función Despedida() y se modifica el valor de acuerdo al valor cont.

**Pricipal.c**

- ***void Leer_Arreglo(int Arreglo[TAMAX], int longitud)***
  Solicita al usuario que teclee cada elemento que será almacenado en el arreglo.
- ***void Imprimir_Arreglo(int Arreglo[TAMAX], int longitud)***
  Envía a pantalla el contenido de un arreglo.

**LibreriaPrueba,h**

- ***Especificación de librerías de uso común.***
- ***Declaración de constantes***
- ***Declaración y definición de tipo de datos nuevos***
- ***Declaración de variables globales***
- ***Definición de funciones por archivo de todo el proyecto***

**Makefile**

- Contienen las 'instrucciones' para automatizar la compilación y ligado de los archivos contenidos en un proyecto.

### 3.3.1 Diseño Modular - Por Archivos.

Desarrolle el diagrama modular de acuerdo al código proporcionado.

### 3.3.2 Compilación con el archivo makefile proporcionado.

Efectué la generación automatizada de los archivos objetos (*o) y del ejecutable de la aplicación. Desde la línea de comando teclee lo siguiente:

```
{~/Practica6/Actividad1}[gustavob@quetzal20]$ make
cc -c Principal.c   LibreriaPrueba.h
cc -c Saludos.c
cc -c Despedidas.c
cc -o Activ1 Principal.o Saludos.o Despedidas.o
```

Para eliminar los archivos de objetos teclee la siguiente línea:

```
{~/Practica6/Actividad1}[gustavob@quetzal20]$ make limpia
rm Principal.o Saludos.o Despedidas.o
```

```
{~/Practica6/Actividad1}[gustavob@quetzal20]$ ls
Activ1        LibreriaPrueba.h     Makefile     Saludos.c
Despedidas.c  LibreriaPrueba.h.gch  Principal.c
```

Si efectúa modificaciones en los archivos fuentes, debe de borrar los archivos obj previamente generados y volver a generar el ejecutable.

Ejecute el programa generado y observe su funcionamiento.

Para la entrega del reporte debe de indicar todos aquellos ‘eventos’ que se generaron para obtener el ejecutable del proyecto, referente a la actividad 1, y la solución de los mismos.

## Actividad 2 “TDA – Lista Secuencial”

*Con el desarrollo de esta actividad se alcanzará los objetivos del 1 al 5 de la práctica.*

Cree un directorio, dentro del directorio Practica6, con el nombre ListaSecuencial. Descargue los archivos proporcionados y analícelos.

### 3.3.3 Menú TDA Lista Secuencial

Integre el uso de un menú que permita la interacción del usuario con las operaciones de manipulación del TDA Lista Secuencial.

### 3.3.4 Archivo make.

Desarrolle un archivo make para la compilación y ligado del proyecto.

### 3.3.5 Diseño Modular

Desarrolle el diagrama modular de acuerdo al código proporcionado y al desarrollo de los módulos que hacen falta.