Organización de Computadoras
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Segundo Cuatrimestre de 2015

Proyecto Nº 1
Programación en Lenguaje C

El objetivo principal del proyecto es implementar funciones para la administración de un bloque de memoria.

Para esto se debe implementar:

- un TDA Lista utilizando memoria dinámica mediante `free` y `malloc`.
- un TDA Memoria Heap que implementa la administración de un bloque propio de memoria y utiliza el TDA Lista anterior.
- un TDA Pila utilizando el espacio de memoria implementado.
- un programa de prueba que utiliza el TDA Pila anterior.

1. TDA Lista

Implementar en C utilizando punteros un TDA Lista cuyos elementos sean pares de enteros. La lista debe ser simplemente enlazada sin centinelas, con posición directa. La implementación debe contener las operaciones:

- `tLista crearLista()` Crea una lista vacía y la devuelve.
- `void insertarAtras(tLista* L, tNodo* pos, tElemento x)` Inserta el elemento `x` en la posición inmediata siguiente a `pos` en la lista.
- `void insertarPpio(tLista* L, tElemento x)` Inserta el elemento `x` en la primera posición de la lista.
- `tNodo* siguiente(tNodo* pos)` Devuelve la posición siguiente a la posición `pos`. Devuelve `NULL` si `p` es `NULL` o si no existe posición siguiente.
- `tNodo* primera(tLista L)` Devuelve la primera posición de la lista `L`.
- `tNodo* ultima(tLista L)` Devuelve la última posición de la lista `L`.
- `void eliminar(tLista* L, tNodo* pos)` Elimina el elemento de la posición `pos` de la lista.
- `tElemento* elemento(tNodo* pos)` Devuelve un puntero al elemento que ocupa la posición `pos` de la lista.
- `int listaVacia(tLista L)` Devuelve verdadero ($\neq 0$) si la lista está vacía, y falso (0) si la lista contiene al menos un elemento.

donde los tipos `tElemento`, `tNodo` y `tLista` están definidos de la siguiente manera:

```
typedef struct Elemento{
    int a;
    int b;
} tElemento;
```

```
typedef struct Nodo {
    tElemento elem;
    struct Nodo* next;
} tNodo;

typedef struct Lista {
    tNodo* header; //puntero al primer nodo de la lista
    tNodo* tail; //puntero al último nodo de la lista
    int size;
} tLista;
```

2. TDA Memoria Heap

Implementar un TDA Memoria Heap que simule la asignación y liberación de memoria en un proceso, mediante la asignación y liberación de bytes en un bloque de memoria declarado de la siguiente forma:

```
char* mem
```

La memoria heap debe ser única para toda aplicación que utilice este espacio de memoria.

El TDA debe mantener información sobre bytes libres a través de una lista enlazada con información sobre los bloques libres. ***Puede agregar estructuras auxiliares que considere necesarias para la administración de los bloques libre y ocupados.*** Para la implementación de esta lista se debe utilizar el TDA Lista implementado anteriormente.

Se deben proveer las siguientes operaciones:

- `int inicializarMemoria(int max)` Crea e inicializa un espacio de memoria de m bytes, siendo m el menor múltiplo de 8 mayor o igual que `max`. Esta función debe inicializar las listas de bloques libres de forma tal que toda la memoria esté disponible. Retorna 0 si la creación fue exitosa, $\neq 0$ en caso de error, según el siguiente código de error: 1 no hay suficiente memoria, 2 la memoria ya fue inicializada, 3 cualquier otro error.

- `void* asignar(int cant)` Reserva `cant` bytes de memoria contigua y devuelve el puntero al primer byte. Retorna NULL en caso de no poder encontrar `cant` bytes contiguos.

- `void liberar(void* bloque)` Libera el bloque de memoria apuntado por `bloque` y actualiza las estructuras auxiliares.

  Al liberar memoria se deben *fusionar* los bloques contiguos de memoria libre para asegurar mayor éxito en futuras invocaciones a `asignar`.

3. TDA Pila

Implementar en C el TDA para representar una pila de enteros. La pila debe implementarse como una lista enlazada dentro del espacio de memoria definido en el ejercicio anterior, es decir, no se deben utilizar las operaciones `malloc` y `free`, sino que deben utilizarse las operaciones `asignar` y `liberar` que provee el TDA Memoria Heap. Debe proveer las siguientes operaciones:

1. `tPila* crearPila()` Retorna una pila vacía nueva. Retorna NULL en caso de no poder crear la pila.

2. `int tope(tPila* P)` Retorna el entero que se encuentra en el tope de la pila. Su comportamiento no está definido en caso de que la pila se encuentre vacía.

3. `int desapilar(tPila** P)` Elimina el entero que se encuentra en el tope de la pila y lo retorna.

4. `int apilar(int a, tPila** P)` Inserta el entero `a` en el tope de la pila. Retorna 0 si la inserción fue exitosa, $\neq 0$ en caso contrario.

5. `int pilaVacia(tPila* P)` Devuelve verdadero ($\neq 0$) si la pila esta vacía, falso (0) en caso contrario.

6. `tPila* subPila(int (*f)(int, int), int a, tPila* P)` Crea una nueva pila con todos los elementos `e` de la pila `P` tales que `f(e, a)` es verdadero($\neq 0$).

donde el tipo `tPila` está definido de la siguiente manera:

```
typedef struct Pila {
    int elem;
    struct Pila* next;
} tPila;
```

4. Programa de prueba

Se debe realizar un programa que reciba como argumento un archivo de enteros y un criterio de selección, y escriba en un nuevo archivo, o muestre por pantalla, los enteros del archivo que cumplen con dicho criterio.

Este programa debe leer el archivo completo ya almacenarlo en una estructura tipo `tPila`. Utilizando la función `subPila` se deben obtener los números que cumplen la condición especificada. Además, en el programa principal se deben implementar las funciones de comparación (mayor, menor o múltiplo de un número dado). Por último, se debe crear un archivo nuevo o mostrar por pantalla el resultado obtenido.

Este programa debe hacer uso únicamente del TDA Pila implementado
No debe utilizar el TDA Lista.

El programa implementado (al que llamaremos `seleccionar`) debe conformar la siguiente especificación al ser invocado desde la línea de comandos:

$ seleccionar [-h] [<criterio><archivo entrada>] [<archivo salida>]

Los parámetros entre corchetes denotan parámetros opciones. El significado de las diversas opciones es el siguiente:

- En caso de no especificar parámetros, especificar parámetros inválidos o especificar el parámetro -h como primer argumento en la línea de comandos, el programa debe mostrar una pequeña ayuda por pantalla. Esta ayuda consiste en un breve resumen del propósito del programa junto con una reseña de las diversas opciones disponibles. En el caso de parámetros inválidos, además de la ayuda, y de ser posible, debe indicarse cuál fue el error en los parámetros.

- Si se especifica un archivo de salida, deben escribirse los números seleccionados en dicho archivo. En caso contrario, deben mostrarse por pantalla.

- El criterio especificado debe ser uno de los siguientes:

  - `>N` Selecciona los números mayores a `N`.
  - `<N` Selecciona los números menores a `N`.
  - `%N` Selecciona los números que son múltiplo de `N`.

- El archivo de entrada debe ser una secuencia de enteros, uno por cada línea del archivo.

Por ejemplo, a partir del archivo *números.txt* que tiene los números:

1
5
6
3
25
63
12
11
21

en la siguiente tabla se muestra qué debería mostrar por pantalla cada una de la siguientes invocaciones:

Invocación	Resultado por pantalla
$ seleccionar %3 numeros.txt	6 3 63 12 21
$ seleccionar ">20" numeros.txt	25 63 21
$ seleccionar "<20" numeros.txt	1 5 6 3 12 11

Sobre la implementación

- Los archivos fuente principales se deben denominar **pila.c, lista.c, memoria.c y seleccionar.c** respectivamente.

  Recordemos que en el caso de librerías, también se deben adjuntar los respectivos archivos de encabezados **pila.h, lista.h y memoria.h**, los cuales han de ser incluidos en los archivos fuente de los programas que hagan uso de estas librerías.

- Es importante que durante la implementación del proyecto se haga un uso cuidadoso del espacio de memoria, tanto para la reservar (`malloc`) la memoria necesaria, como para liberar (`free`) la memoria de variables que ya no son necesarias.

- Se debe respetar los nombres de tipos y los encabezados de funciones especificados en el enunciado. *Los proyectos que no cumplan esta condición quedarán automáticamente desaprobados*

Sobre el estilo de programación

- El código implementado debe reflejar la aplicación de las técnicas de programación modular estudiadas a lo largo de la carrera.

- En el código, entre eficiencia y claridad, se debe optar por la claridad. Toda decisión en este sentido debe constar en la documentación que acompaña al programa implementado.

- El código debe estar *indentado* y adecuadamente comentado.

Sobre la documentación

Los proyectos que no incluyan documentación estarán automáticamente desaprobados.
La documentación debe:

- estar dirigida principalmente a usuarios con conocimientos de programación.

- explicar brevemente los programas realizados, así como las decisiones de diseño tomadas, y toda otra observación que se considere pertinente.

- incluir explicación de todas las funciones implementadas, indicando su prototipo y el uso de los parámetros de entrada y de salida.

Sobre la entrega

- Las comisiones deben estar conformadas por exactamente 3 alumnos registrados con la cátedra. La fecha límite para informar a la cátedra quiénes son los integrantes de cada comisión es el **Miércoles 30 de septiembre de 2015**.

  No se aceptarán cambios a último momento en los integrantes.

- El código fuente y el informe del proyecto deberán ser enviados en un archivo **zip** por mail a la asistente de la cátedra hasta las **8:00hs** del día **Miércoles 14 de octubre de 2015**. Tanto el asunto del mail como el nombre del archivo comprimido debe ser el siguiente:

  [OC2015] Proyecto 1 - apellido de los integrantes de la comisión

  Toda comisión que no cumpla este punto estará automáticamente desaprobada.

- El día **Miércoles 14 de octubre de 2014**, de **10 a 12hs**, en el aula de clase se deberá entregar un folio plástico (no se aceptarán carpetas), cerrado con cinta adhesiva, conteniendo los siguientes elementos:

  - Una carátula que identifique claramente a los integrantes de la comisión.
  - Una impresión en doble faz de los archivos ".c" y ".h" enviados por mail.
  - La documentación del proyecto impresa en doble faz.

  No se aceptarán discrepancias entre el código fuente impreso y el enviado por mail.