Proyecto 5

Algoritmos y Estructuras de Datos I
Laboratorio

2 de noviembre de 2009

El proyecto consta de una serie de ejercicios para programar algoritmos en el lenguaje C.

La implementación en lenguaje C debe hacerse traduciendo los programas escritos en el formalismo del teórico producto de su derivación, tal como se vio en el teórico del laboratorio, a menos que en el enunciado se diga lo contrario. Los programas deben tomar los datos de entrada del usuario y mostrar los resultados en pantalla.

Para hacer los programas tener en cuenta:

- Antes de escribir el programa en lenguaje C, derivar o demostrar el algoritmo en el lenguaje del teórico a partir de su derivación.

  Una vez que tenga el algoritmo escrito en el lenguaje del teórico, recién allí escribirlo en el lenguaje C.

  Para evaluar el ejercicio los docentes verificarán que la traducción al lenguaje C corresponde al programa derivado o demostrado junto con su especificación. Por lo tanto al momento de la evaluación hay que presentar la especificación, la derivación (o demostración) y el programa resultante en el lenguaje del teórico.

  Se debe hacer la derivación o demostración de los programas a menos que se indique lo contrario en el ejercicio.

- En todos los programas utilizar la técnica de archivos separados según se indique.

- Recordar que la entrada debe chequear el cumplimiento de la precondición en la especificación del programa.

- Cuando se creen "headers" (archivos `.h`) hacer la cerradura `ifndef` como se explicó en el teórico.

- Compilar todos los archivos y linkear el programa con las opciones `-ansi`, `-pedantic` y `-Wall`. Ver que no haya mensajes de error o advertencias.

- No usar ninguna variable global.

- Hacer primero todos los ejercicios y después los puntos estrella.

- Para obtener `B+` todos los ejercicios y puntos estrella deben estar hechos en forma correcta.

- Para obtener `B` todos los ejercicios deben estar hechos en forma correcta.

1. Derive y programe en C el programa que dice si todos los elementos en un arreglo son positivos (ejercicio 19.3 del libro).

   Para ello:

   - Escribir en un archivo separado `bool.h` el tipo `bool`.

- Escribir el algoritmo en una función

  ```
  bool todos_positivos(int arr[], int n);
  ```

  donde `arr` es el arreglo y n es su longitud.
- Escribir esta función en los archivos separados `operaciones.h` y `operaciones.c`. En el primero debe estar la signatura (o prototipo) de la función y en el segundo su implementación.
- Escribir las funciones de entrada/salida en los archivos separados `in_out.h` e `in_out.c`. En el primero deben estar las signaturas (o prototipos) de la funciones y en el segundo sus implementaciones.

Punto $\star$: Hacer que la función devuelva la posición en el arreglo donde hay un número negativo (si es que lo hay). No hace falta demostrar este agregado.

2. Derive y programe en C el programa que dice si algún elemento en un arreglo es positivos (ejercicio 19.3 del libro).

   Para ello:

   - Escribir en un archivo separado `bool.h` el tipo `bool`.
   - Escribir el algoritmo en una función

     ```
     bool algun_positivos(int arr[], int n);
     ```

     donde `arr` es el arreglo y n es su longitud.
   - Agregar esta función a los archivos separados `operaciones.h` y `operaciones.c`. En el primero debe estar la signatura (o prototipo) de la función y en el segundo su implementación.
   - Agregar las funciones de entrada/salida a los archivos separados `in_out.h` e `in_out.c`. En el primero deben esta las signaturas (o prototipos) de la funciones y en el segundo sus implementaciones.

   Punto $\star$: Hacer que la función devuelva la posición en el arreglo donde hay un número negativo (si es que lo hay). No hace falta demostrar este agregado.

3. Derive y programe en C el programa que dice si los paréntesis en un string están balanceados (ejemplo 20.3 del libro).

   Para ello:

   - Escribir en un archivo separado `bool.h` el tipo `bool`.
   - Escribir el algoritmo en una función

     ```
     bool parentesis_balanceados(char *str);
     ```

     donde `str` es la cadena de caracteres.
   - Agregar esta función a los archivos separados `operaciones.h` y `operaciones.c`. En el primero debe esta la signatura (o prototipo) de la función y en el segundo su implementación.
   - Agregar las funciones de entrada/salida a los archivos separados `in_out.h` e `in_out.c`. En el primero deben esta las signaturas (o prototipos) de la funciones y en el segundo sus implementaciones.

4. Derive y programe en C el programa que calcula la suma del segmento de suma máxima (ejemplo 19.9 del libro).

   - Escribir el algoritmo en una función

     ```
     int suma_maxima(int arr[], int n);
     ```

     donde `arr` es el arreglo y n es su longitud.
   - Agregar esta función a los archivos separados `operaciones.h` y `operaciones.c`. En el primero debe esta la signatura (o prototipo) de la función y en el segundo su implementación.
   - Agregar las funciones de entrada/salida a los archivos separados `in_out.h` e `in_out.c`. En el primero deben esta las signaturas (o prototipos) de la funciones y en el segundo sus implementaciones.

   Punto $\star$: Al momento de programar los algoritmos anteriores se puede definir el tamaño del arreglo como una constante `N` (definida mediante `#define`). ¿Se puede hacer que este tamaño sea decidido por el usuario?. Lea la info page de libc donde se habla de arreglos, responda a la pregunta y lea si la posible solución tiene algún inconveniente. Programar todos los ejercicios anteriores según la respuesta.

5. Hacer dos implementaciones del TAD Número Complejo:

   a) Escribir un archivo separado para los números reales llamado real.h e implementarlo como un sinónimo del tipo `double`.

   b) Escribir en archivos separados la definicion del tipo complejo como un par parte real y parte imaginaria, su constructor y demás funciones (TAD) siguiendo el esquema de los archivos:
   - En el archivo complejo.h definir el tipo como una estructura y escribir los prototipos. El archivo debe quedar como el que figura en el apéndice A.
   - En el archivo complejo.c, incluir el archivo complejo.h e implementar todas sus funciones.

   c) Escribir en otro archivo el bloque *main* que implemente una interface con el usuario donde el mismo pueda cargar 2 números complejos `x` e `y` y muestre por pantalla el valor:

   ```
   a * x + b * y + c
   ```

   con

   ```
   a = 2 + i * 3
   b = 8 + i
   c = 5 + i * 7
   ```

   d) Después de terminar este programa, hacer otra implementación donde el número complejo se represente por su módulo y su ángulo. Para hacer esto, cambiar únicamente el archivo complejo.c y la definición de la estructura en el archivo complejo.h.

   e) Al hacer estas dos implementaciones intentar ocultar toda la información posible sobre la implementacion del TAD al escribir el archivo con el bloque `main`.

   f) No hace falta derivar los algoritmos.

   Punto $\star$: Con el archivo complejo.h del apéndice ¿se puede ocultar la implementación del TAD? Si no es así, ¿se le ocurre alguna manera de hacerlo?

Apéndice A

```
#ifndef COMPLEJO_H
#define COMPLEJO_H

#include "real.h"

struct scompl {
  real rl, img;
};

typedef struct scompl compl;

compl
compl_create(const real rl, const real img);
/*
DESC: Constructor del tipo
*/

real
compl_preal(const compl c);
/*
DESC: Devuelve la parte real.
*/

real
compl_pimag(const compl c);
/*
DESC: Devuelve la parte imaginaria.
*/

compl
compl_suma(compl c, const compl x);
/*
DESC: Suma x a c.
*/

compl
compl_resta(compl c, const compl x);
/*
DESC: Resta x a c.
*/

compl
compl_prod(compl c, const compl x);
/*
DESC: Multiplica x a c.
*/

void
compl_print(const compl c);
/*
```

```
DESC: Imprime en pantalla c con formato a + i b.
*/

compl
compl_leer(void);
/*
DESC: Lee por teclado y construye.
*/

compl
compl_clone(const compl x);
/*
DESC: Construye una copia.
*/


#endif /* COMPLEJO_H */
```