Proyecto 2
TAD arreglo de enteros

Algoritmos y Estructuras de Datos II

17 de marzo de 2006

Hacer una implementación del TAD arreglo de enteros y utilizarlo para devolver el reverso del arreglo representado.

1. Escribir en archivos separados la definicion del tipo arreglo de enteros, sus constructores, destructor y demás procedimientos teniendo en cuenta que se debe poder proyectar su tamaño. Para ello seguir el siguiente esquema de archivos:

   - En el archivo arrInt.h definir el tipo como una estructura (arreglo + cantidad de elementos) y escribir los prototipos de los constructores, destructor y demás procedimientos con todas las anotaciones. El archivo debe quedar como el que figura en el apéndice .
   - En el archivo arrInt.c, implementar todos los procedimientos declarados en el archivo anterior. Para ello algunas de las cosas que hay que tener en cuenta son:
     - En los constructores utilizar las funciones `malloc` o `calloc` para la generación dinámica de memoria. Acordarse de utilizar `free` en los destructores. Para ver como funcionan y que librerias hay que incluir, consultar las *man pages* y las *info pages*.
     - En el procedimiento `arrInt_fillRand` usar las funciones `rand` y `srand` para la generación de números aleatorios. Para `srand` puede elegir como semilla el tiempo medido en segundos provisto por la función `time`. Para ver como funcionan y que librerias hay que incluir, consultar las *man pages* y las *info pages*.
     - En el procedimiento `arrInt_swap` primero definir la precondición y la postcondición de intercambiar y luego escribir el algoritmo.
     - Obviamente la implementación de la función `arrInt_length` debe ser de orden constante.

2. Escribir en otro archivo el bloque *main* que implemente una interface con el usuario donde, el mismo pueda ejecutar cuantas veces pueda:

   a) Crear un arreglo de la longitud que desee si es que no existe uno construido (y no fue debidamente destruido). En este caso preguntar después si se desea llenarlo por teclado (procedimiento `arrInt_fillStdIn`) o llenarlo al azar (`arrInt_fillRand`) . Al final hay que mostrar el arreglo por pantalla (procedimiento `arrInt_showStdOut`).

   b) Hacer el swap de dos elementos y mostrar el resultado (procedimientos `arrInt_swap` y `arrInt_showStdOut`).

   c) Invertir el orden de los elementos del arreglo (procedimiento `arrInt_reverse`) y mostrar el resultado.

   d) Destruir el arreglo.

   e) Terminar el programa. En este momento si hay un arreglo construido hay que destruirlo.

Nota: El usuario no debe poder ejecutar ningún procedimiento si no existe un arreglo creado.

Para hacer todos los programas seguir la siguientes directivas generales:

- Ocultar toda la información posible sobre la implementacion del TAD al escribir el archivo con el bloque `main`.
- Poner los bloques `ifndef` en cada archivo .h.
- En el archivo arrInt.h
  - Especificar con pre y pos cada prototipo de los procedimientos expresando solo propiedades abstractas sobre arreglos.
  - Agregar descripción y uso.
  - Incluir con la directiva `#include` la mínima cantidad de headers que le harán falta al programador para poder usar el TAD.
- En la implementación del TAD (archivos arrInt.c)
  - Especificar (con pre y pos) cada función teniendo en cuenta que, a diferencia del punto anterior, se puede expresar propiedas sobre la estructura que implementa al TAD.
  - Demostrar que los procedimientos `arrInt_swap` y `arrInt_reverse` son correctos en base a estas especificaciones.
  - Usar la librería assert para testear la parte de las precondiciones y postcondiciones que sean posibles.
- Hacer Makefile compilando todos los archivos con las opciones `-ansi`, `-pedantic` y `-Wall`. Ver que no haya mensajes de error o advertencias.
- No usar ninguna variable global.

Punto optativo 1a: En este TAD los procedimientos como `arrInt_swap` o `arrInt_fillRand` reciben como parámetro el objeto de tipo `arrInt` que modificarán y no devuelven nada (`void`). ¿Qué pasa si sigo este esquema y quiero hacer un procedimiento que cambie el entero que almacena la cantidad de elementos? (por ejemplo un procedimiento que copie un arreglo en otro; esta comentado en el apéndice). Cambie la implementación del TAD para que esto sea posible sin cambiar las signaturas de los procedimientos.

Punto optativo 1b: Si hizo el punto anterior haga también el procedimiento `arrInt_copy` utilizando la función de librería `realloc` (ver *man page*) para aumentar el tamaño del arreglo si es necesario.

Punto optativo 2: Cuando incluyo arrint.h en el archivo que contiene el bloque `main` puedo acceder a los campos del registro (`ai.a` y `ai.n`). ¿ Hay alguna forma para que esta información no sea accesible y así ocultar la implementación del TAD?. Si cree que si haga el proyecto con este cambio.

Punto optativo 3: ¿Hay alguna forma de definir un TAD arreglo donde se pueda cambiar "facilmente" el tipo de los elementos que almacena? (la idea es tener un tipo parametrico como las listas de `a` en haskell). Si cree que sí, pregunte si la idea es correcta y haga el proyecto con este nuevo TAD.

Apéndice A

```
#ifndef ARRINT_H
#define ARRINT_H

struct sArrInt {
  int * a;
  int n;
};

typedef struct sArrInt arrInt;

/* Nota: a=A => created(a) */

/* Constructores y destructor */

arrInt
arrInt_create(const int ne);
/*
DESC: Constructor del tipo
PRE: { ne >= 0 /\ ! created(a) }
   a = arrInt_create(ne);
POS: { Hay memoria ==> a = A /\ length(a)=ne }
*/

arrInt
arrInt_clone(const arrInt a);
/*
DESC: Construye una copia.
PRE: { a=A /\ ! created(b) }
   b = arrInt_clone(a);
POS: { Hay memoria ==> a=A  /\ b=a }
*/

void
arrInt_destroy(arrInt a);
/*
DESC: Destructor de tipo. Cambia a.
PRE: { a=A }
   arrInt_destroy(a);
POS: { ! created(a) }
*/


/* Proyector */

int
arrInt_length(const arrInt a);
/*
DESC: Devuelve el tamanio.
```

```
PRE: { a=A }
   l = arrInt_length(a);
POS: { a=A /\ l=length(a) }
*/

/* Procedimientos */

void
arrInt_swap(arrInt a, const int i, const int j);
/*
DESC: Hace el swap entre las posiciones i y j. Cambia a.
PRE: { a=A /\ 0 <= i,j < length(a) }
   arrInt_swap(a,i,j);
POS: { a=swap(A,i,j) }
*/

void
arrInt_reverse(arrInt a);
/*
DESC: Hace el reverse de a. Cambia a.
PRE: { a=A }
   arrInt_reverse(a);
POS: { (FORALL i: 0<= i < length(a): a[i] = A[length(A)-i-1])
       /\ length(a)=length(A) }
*/

void
arrInt_fillStdIn(arrInt a);
/*
DESC: Llena a por teclado. Cambia a.
PRE: { a=A }
   arrInt_fillStdIn(a);
POS: { llenado a por teclado /\ length(a)=length(A) }
*/

void
arrInt_fillRand(arrInt a);
/*
DESC: Llena a con valores al azar. Cambia a.
PRE: { a=A }
   arrInt_fillRand(a);
POS: { llenado a con valores al azar /\ length(a)=length(A) }
*/

void
arrInt_showStdOut(const arrInt a);
/*
DESC: Muestra por pantalla los valores en a.
PRE: { a=A }
   arrInt_showStdOut(a);
```

```
POS: { a=A /\ Valores de A en pantalla }
*/

/* void  */
/* arrInt_copy(arrInt b, const arrInt a); */
/* /\* */
/* DESC: Copia el objeto a en b.  */
/* PRE: { a=A /\ b=B /\ length(a) <= length(b)} */
/*    arrInt_copy(b,a);   */
/* POS: { a=A  /\ b=a }    */
/* *\/ */

#endif /* ARRINT_H */
```