Proyecto 5
Parte I

Algoritmos y Estructuras de Datos I
Laboratorio

14 de noviembre de 2007

1. Derivar (se vio en el teórico) e implementar en lenguaje C el algoritmo de la división, el cual calcula el cociente y resto de la división entera dado el dividendo y el divisor. El programa escrito en C debe respetar la derivación teniendo en cuenta las traducciones del formalismo al lenguaje vistas en el teórico del taller.

   Para hacer el programa en C hay que tener en cuenta los siguientes lineamientos:

   - El algoritmo de la división debe estar en una función separada. Esta función debe contener solo el algoritmo derivado en el teórico y sin incluir ningún algoritmo de entrada salida.
   - Hacer tambien dos funciones extras, una para ingresar el dividendo y divisor por teclado (mostrando mensajes de error si no se verifica la precondición del algoritmo), y otra para mostrar el cociente y el resto por pantalla.
   - Programar la función `main` con estas tres funciones unicamente.
   - Para devolver los resultados de las dos primeras funciones, simular pasaje de argumentos por referencia en C (punteros).
   - No usar variables globales.

2. Especificar y derivar un programa que decida si un número se encuentra en un arreglo de enteros.

   Implementar el programa calculado en C. El programa debe solicitar el ingreso de todos los elementos del arreglo y del número cuya existencia en el arreglo se quiere verificar.

   Pistas: Para la implementación del programa en C, la longitud del arreglo se puede determinar de dos maneras:

   a) a través de una directiva para el preprocesador (la longitud del arreglo queda determinada en el momento de compilación):

```
#include <stdio.h>
#define N 5
/* N es la cantidad de elementos del arreglo */
int main() {
```

```
int arreglo[N]; ...
```

b) declarando una variable entera (sea `n`) cuyo valor se definirá durante la ejecución del programa y (en un punto posterior a la asignación de esa variable) declarando un arreglo de longitud `n`. De esta manera podemos indicarle al programa la longitud del arreglo en tiempo de ejecución:

```
...
int n;
n = 8;
int arreglo[n];
...
```

La variable n puede ser definida también por entrada externa (ej. `scanf`).

Caveat lector (de la página de info de gcc sección "C extension" "variable length"):

> Variable-length automatic arrays are allowed in ISO C99, and as an extension GCC accepts them in C89 mode and in C++. (However, GCC's implementation of variable-length arrays does not yet conform in detail to the ISO C99 standard.)
>
> Los arreglos de longitud variable están permitidos en ISO C99, y GCC los acepta como una extensión en el modo C89 y en C++. Sin embargo, la implementación de arreglos de longitud variable en GCC no sigue detalladamente al estándar ISO C99.

Pregunta para evaluar el proyecto: ¿ Qué problemas tienen los arreglos de tamaño variable?

Ayuda: Ver info page de libc en la sección "Automatic Storage with Variable Size".

3. En este ejercicio se reimplementará el proyecto 3 de diccionario en el lenguaje C. Solo se implemetará la versión con lista de asociaciones.

   Para hacer cada TAD se deberá tener en cuenta:

   - Cada TAD se deberá escribir en un par de archivos separados, un .h y un .c.
   - Para implementar correctamente los TAD's, cada .h debera exportar únicamente las funcionalidades del TAD que define, ocultando todos los aspectos que tienen que ver con su implementación.
   - Se usarán strings de tamaño fijo pero arbitrario, lo mismo que el tamaño de la lista.

   Utilizar el tipo `bool` dado en clase. A continuación se detallan los TAD's a implementar.

   a) Reimplementar el TAD Data. Básicamente hay implementar estas funciones en C:

```
#define DATA_MAX_LENGH 50

/* como no estamos usando punteros,
   la definición de la estructura queda aca */
typedef struct {
    ...
} Data;

/* crea un Dato a partir de un string */
Data data_fromString (char *s);

/* convierte un dato en string */
void data_toString (Data d, char *s);

/* devuelve el largo del string almacenado */
int data_lenght (Data d);
```

b) Reimplementar el TAD Key. Básicamente hay implementar estas funciones en C:

```
#define KEY_MAX_LENGH 20

typedef struct {
    ...
} Key;

/* devuelve el largo máximo definido por KEY_MAX_LENGH */
int key_maxLen (void);

/* construye un Key desde un string */
Key key_fromString (char *s);

/* convierte un Key a string */
void key_toString (Key k, char *s);

/* devuelve la longitud del string en k */
int key_length (Key k);

/* devuelve TRUE si las claves son iguales */
bool key_eq (Key k1, Key k2);
```

Notar que, se debe definir y usar `key_eq()`.

c) Como C carece de tuplas, hay que implementarlo. La signatura debe ser:

```
typedef struct {
    ...
} Tuple;
```

```
/* constructor */
Tuple tuple_fromKeyData (Key k, Data s);

/* devuleve la primer componente */
Key fst (Tuple t);

/* devuleve la segunda componente */
Data snd (Tuple t);
```

d) Reimplementar el TAD ListaAsoc cuya signatura sea:

```
#define LIST_SIZE 100

typedef struct {
    ...
} ListaAsoc;

/* constructor */
ListaAsoc la_empty (void);

/* agrega una clave y un dato
   la lista no está ordenada
   notar que devuelve la lista modificada */
ListaAsoc la_add (ListaAsoc l, Key k, Data d);

/* devuelve TRUE si existe la clave en la lista */
bool la_exists (ListaAsoc l, Key k);

/* busca un dato según la clave
   debe llamarse sólo si la_exists (l, k) devuelve TRUE
   si no lo encuentra devuelve cualquier verdura */
Data la_search (ListaAsoc l, Key k);

/* borra la clave y el valor asociado
   devuelve la lista modificada */
ListaAsoc la_del (ListaAsoc l, Key k);
```

Nota: no se reimplementa `_toListPart()`.

Ejercicio estrella: Como se puede hacer para imprimir la lista?

e) Reimplementar el TAD Dict con la lista de asociaciones. La signatura debe ser:

```
typedef char Word[20];
typedef char Def[100];

typedef struct {
    ...
} Dict;
```

```
/* constructor */
Dict dict_empty (void);

/* agrega palabra y definición
   devuelve el diccionario modificado */
Dict dict_add (Dict d, Word w, Def f);

/* devuelve TRUE si la palabra está en el diccionario */
bool dict_exists (Dict d, Word w);

/* devuelve la definición de una palabra
   llamar solo si dict_exists (d, w) da TRUE
   si no está devuelve cualquier verdura */
void dict_search (Dict d, Word w, Def *f);

/* saca la palabra y la definición del diccionario
   devuelve el diccionario modificado */
Dict dict_del (Dict d, Word w);

/* imprime todos los pares (clave, valor) en el diccionario */
void dict_pprint (Dict d);
```

Ejercicio estrella: porqué `dict_search` no devuelve `Def`? Qué debería hacerse para que la signatura sea:

```
/* devuelve la definicion de una palabra
   llamar solo si dict_exists (d, w) da 1
   si no esta devuelve cualquier verdura */
Def dict_search (Dict d, Word w);
```

Se puede hacer que la signatura sea:

```
/* devuelve la definicion de una palabra
   llamar solo si dict_exists (d, w) da 1
   si no esta devuelve cualquier verdura */
void dict_search (Dict d, Word w, Def d);
```

f) Desarrollar una interfaz similar a la que se entregó para el proyecto 3. Notar que no se puede ni cargar ni grabar en disco ni imprimir el diccionario (a menos que se haga el ejercicio estrella del punto 4). Debe implementar la funcionalidad de buscar una palabra.