Computación

Año 2013

Prof: T. S. Grigera — JTP: C. Grunfeld — AD: G. Sieben

Práctica 8 — Lenguaje C II

Esta práctica abarca los siguientes temas:

- Entrada/salida a archivos (`fopen`, `fclose`, `fprintf`, `fscanf`, `fgetc`, `fputc`, `fgets`). Entrada/salida sin formato (`fread` y `fwrite`). Tipo de datos `struct` (estructuras). `typedef` y `union`.
- Declaración vs. definición de funciones: prototipos. Alcance o ámbito (scope) de variables. Variables globales. Módulos: funciones y variables externas. Símbolos públicos y privados: `static`. Variables automáticas y estáticas y segundo significado de `static`. Argumentos de funciones: noción de pasaje por valor y por referencia. Recursión.
- Punteros. Punteros y arreglos. Cadenas de caracteres como arreglos de `char`. Algunas funciones para manipular cadenas. Asignación dinámica de memoria. Punteros a `struct`.

Bibliografía:

- B. W. Kernighan y D. M. Ritchie, *El lenguaje de programación C (2da ed.)*, Prentice-Hall Hispanoamericana, Mexico (1991).
- *C Programming web tutorial*, `http://www2.its.strath.ac.uk/courses/c/tableofcontents3_1.html`

Problema 1. Punteros.

a) Escriba el código en C que haga lo siguiente:

i) Defina un puntero a `int` llamado `addr`.

ii) Asigne la dirección de una variable `float` llamada `bal` al puntero a `float temp`.

iii) Assigne la letra ='w'= a la variable a la que apunta el puntero `let`.

b) ¿Cuál es el la salida del siguiente fragmento?

```
int  count = 10, *temp, sum = 0;

temp = &count;
*temp = 20;
temp = &sum;
*temp = count;
printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum );
```

c) Después del siguiente fragmento

```
int *p;
int i;
int k;
i = 42;
k = i;
p = &i;
```

¿cuál o cuáles de las siguientes instrucciones cambiará el valor de `i` a 75?

- `k=75`
- `*k=75`
- `*p=75*`
- `p=75`

d) Utilizando punteros, escriba una función o fragmento de código que imprima los caracteres de una cadena `char s[]` en orden inverso.

Problema 2. Escriba una función para leer números de un archivo de datos que contenga una serie de números reales a razón de uno o más por línea, separados por espacios. El prototipo debe ser

```
int lee_nro(char archivo[],double *num);
```

donde `archivo` contendrá el nombre del archivo a leer y en `num` se devolverá el número leído. La función en sí devolverá verdadero si se ha podido leer un número correctamente, y falso en caso contrario. Note que la función deberá devolver sucesivamente los distintos números del archivo cada vez que es llamada, de modo que pueda utilizarse por ejemplo como sigue:

```
double a;
while (lee_nro(argv[1],&a))
  printf("%g\n",a);
```

Ayuda: recuerde la sentencia `static`.

Problema 3. Utilice la función del problema anterior para escribir un programa que lea un archivo con números reales y los grabe en otro archivo, de a uno por línea.

Problema 4. Asignación dinámica de memoria. En aplicaciones numéricas ocurre con relativa frecuencia la necesidad de leer un conjunto de números reales y almacenarlos en un arreglo para luego procesarlos. En general es deseable que el programa sea capaz de leer tantos números como puedan almacenarse en la memoria, sin fijar un límite predeterminado dentro del programa: por esta razón no es conveniente utilizar arreglos con espacio asignado estáticamente, del tipo

```
#define NMAX 2000

double data[NMAX];
```

La asignación dinámica de memoria (mediante `malloc()` y `free()`), aunque algo más complicada, es mucho más conveniente. Escriba una función que sea capaz de leer un archivo que contenga como primera línea un número entero que indique cuántos reales hay a continuación. El prototipo debe ser

```
int leer(char nombre[],double **a,int *N);
```

El valor devuleto por la función se utilizará para indicar errores, mientras que `N` será el número de datos leídos y `a` será un puntero a esos datos. La función podrá utilizarse así:

```
double *d;
int    i,N;
if (leer("datos.dat",&d,&N)==0) exit(1); /* error de lectura */
/* Se han leído N datos, procesar */
for (i=0; i<N; i++) {
   /* Procesar dato numero i */
   d[i]*=2;
   ...
}
/* Fin, liberar la memoria */
free(d);
```

Note el uso de `**` (puntero a puntero) para que la función pueda modificar el valor del *puntero* `d`. Puede ayudarse con la función escrita en el problema 2.

Problema 5. Escriba una versión mejorada de la función anterior, que tenga idéntico prototipo pero que no necesite leer el número `N` del archivo, sino que lo determine por su cuenta, simplemente leyendo todos los reales presentes en el archivo. **Ayuda:** utilice la función `realloc()`.