Soluciones ejercicio preparcial Informática I

Ing. Ignacio Bonelli, Ing. Héctor Levi, Leandro Kollenberger

Sábado 4 de Julio de 2015

1. Un vector es un conjunto o arreglo secuencial de variables de un mismo tipo con una cantidad de elementos constante. Se utiliza en casos en los que sea necesario declarar muchas variables del mismo tipo. Se declara de la siguiente manera:

   ```
   tipodato NombreVector[cantidaddeelementos];
   ```

   Por ejemplo:

   ```
   int V[5];
   char str[20];
   double pepe[712];
   unsigned long long int carlitos[10];
   ```

   Es posible acceder a los elementos del vector de dos maneras posibles: con notación vectorial, o con notación puntero.
   Esto es factible ya que el nombre del vector además es un puntero que contiene la dirección de memoria del comienzo del vector, y el lenguaje nos provee de una manera versátil (comunmente denominada "dualidad puntero-vector") de acceder a estos elementos.
   Por ejemplo:

   ```
   // Tipo vector
       V[4]; // Se lee: "V sub cuatro"
   // Tipo puntero
       *(V+4); /* Se lee: "El contenido de lo apuntado por V, con un offset
                  o corrimiento de cuatro elementos" */
   ```

2. No, no es posible, ya que el nombre de un vector es una constante que representa su dirección de comienzo, la cual es inmutable. En otras palabras, si bien el nombre de un vector es un puntero a su inicio en memoria, este vector es a su vez una constante, y no puede ser modificada por el programa del usuario.
   Comunmente este tipo de asignación incorrecta conlleva a un error del compilador del tipo "nombredelvector no es un l-value". Esto significa que la variable ubicada a la izquierda (left) de un signo igual (asignación) no es un "valor a la izquierda", es decir, un valor asignable.

3. Un puntero es una variable que contiene la dirección de memoria de un elemento del tipo de dato que le corresponde. En otras palabras, en la dirección contenida en una variable tipo puntero encontraremos una variable del tipo de dato a la que el puntero hace referencia.
   Por ejemplo:

   ```
   int *p; // Puntero a un tipo de dato "int"
   char *q; // Puntero a un tipo de dato "char"
   p = 0x2000;
   q = 0x2004;
   ```

   En este caso, `p` nos indica que en la dirección de memoria `0x2000` (es decir, 2000 en hexadecimal) se encuentra una variable del tipo `int`, y que al acceder a ella debemos interpretarla de tal manera.
   Asimismo, la variable `q` (que apunta a la dirección `0x2004`) nos dice que allí hay una variable del tipo `char`.
   Ya que el nombre de un vector es la dirección de su comienzo en la memoria, podemos asignar ese valor a un puntero del mismo tipo de dato, y trabajar con ese vector de exactamente la misma manera que lo haríamos con el vector mismo.

Por ejemplo:

```
int V[10];
int *p;
V[0] = 12; // Escribo el valor "12" en el primer elemento del vector
p = V; /* Asigno la direccion de comienzo del vector al puntero que
          declare anteriormente */
p[0] = 22; /* Sobreescribo el valor del primer elemento del vector,
              en este caso con un "22". */
*(p+0) = 4; // Idem, con notacion tipo puntero.
```

4. La estructura de control de flujo *switch* sirve para reemplazar una gran cantidad de estructuras `if`, `else if` y `else` que comparan una variable entera con múltiples valores posibles, por un bloque de código mucho más ordenado. Estos valores posibles son indicados en sentencias `case`, los cuales se ejecutarán en caso de que la variable cumpla esa condición. Una particularidad de la estructura `switch` es que la ejecución del código continúa hasta el fin de la estructura o hasta que encuentre una sentencia `break`, incluso atravesando otros `case`. Además de esto, provee un caso particular optativo denominado `default`, que se ejecutará en caso de que la variable comparada no sea igual a ninguno de los casos definidos. Por ejemplo:

```
switch(variable)
{
    case 1:
        // codigo que se ejecuta en caso de que "variable" valga "1"
    case 2:
        // codigo que se ejecuta en caso de que "variable" valga "1" o "2"
        break;
    default:
        // esto se ejecutara en caso de que "variable" no sea "1" ni "2"
}
```

5. La declaración `char linea[20];` es un vector de 20 elementos del tipo `char` llamado "linea", lo cual según lo visto anteriormente:

   - "linea" es una variable del tipo puntero a `char`, que contiene la dirección de comienzo del vector.
   - Cada uno de los elementos del vector están ordenados de manera contigua en memoria, y cada uno de ellos ocupa `sizeof(char) = 1 byte`.

   Por lo tanto:

   (a) `*linea` es el contenido de lo apuntado por `linea`, un puntero a `char`. Esto corresponderá al primer elemento del vector, de tipo `char`.

   (b) `linea` como vimos recién es un puntero a char, que contiene la dirección de comienzo del vector.

   (c) `&linea` es la dirección de `linea`, a su vez un puntero al comienzo de un vector de `char`, por lo tanto es un puntero a puntero a `char`, es decir, un `char **`.

6. En este caso, la función `strlen` está siendo invocada incorrectamente, ya que debe tomar como único parámetro (o argumento) un puntero a `char` (la dirección de comienzo del "string" del cual queremos saber la longitud) pero se le está pasando la dirección de ese puntero, por lo tanto un `char **`. La corrección más simple es eliminar el operador `&` (dirección de).

7. (a) Esa línea cumple la función de indicarle al precompilador de C la inclusión del archivo de cabecera `stdio.h`, ubicado en las carpetas del sistema. De esta forma el precompilador agregará los contenidos de este archivo a nuestro código fuente actual.

   (b) Este archivo contiene los prototipos de las funciones de la librería estándar de C orientados al manejo de entrada y salida de datos, como por ejemplo `streams` de datos (teclado) o archivos.

   (c) El código binario (también denominado "objeto") de `printf()` se encuentra alojado en la librería estándar de C, ubicada junto al resto de las librerías del sistema. Al momento de compilar nuestro programa, el `linker` se encargará de *"linkear"* nuestros archivos objeto binario de nuestro programa que acabamos de compilar junto con la librería de C, y producirá como salida un archivo ejecutable.

8. (a) `'a';` ocupa un solo `byte`, ya que es un caracter único, el caracter 'a'.
   (b) `"a";` ocupa dos `bytes`, ya que representa una cadena de caracteres con dos elementos, el caracter 'a' y el caracter nulo que se utiliza para demarcar el fin de esta cadena.

9. Este fragmento de código imprimirá `"Son diferentes!!"` por pantalla, debido a que `p` y `q` representan las direcciones de comienzo de memoria de dos cadenas de caracteres totalmente diferentes, que de casualidad, contienen los mismos valores.
   En caso de que se requiera comparar la igualdad de dos cadenas de caracteres, se deberá recurir a la función de librería `strcmp`, definida en el archivo de cabecera `string.h`.

10. Todas las variables de tipo puntero, independientemente del tipo de dato al cual apuntan, ocupan el mismo espacio en memoria debido a que dentro de ellas está alojada una dirección de memoria. El tamaño de estas variables estará vinculado al tamaño del bus de direcciones del procesador en el cual estaremos trabajando. En nuestras PC, esto suele ser de 32 o 64 `bits`.

11. (a) Este código no compilará, debido a que el parámetro de formato `"%d"` que se le está indicando a la función `sscanf` corresponde a una variable del tipo entero, pero `a` y `b` son del tipo `float`. Lo mismo ocurre con `printf` y la variable `resultado`.
    (b) Fuera de los errores anteriormente indicados, el compilador no indicará ningún `warning`.
    (c) `gcc sumar.c -o ../programa.exe -Wall`
    (d) El problema más importante que puede llegar a tener este programa al momento de ejecutarse es que únicamente se comprueba el caso de que la cantidad de argumentos pasados por línea de comando sea uno. En caso de que se hayan pasado dos argumentos, (por ejemplo, `./programa.exe 32.5`) el segundo `sscanf` provocará una falla de segmentación, debido que `argv[2]` será un puntero nulo.
    (e) `stderr` es la salida estándar de error. Normalmente está direccionada por el sistema opeartivo a la línea de comandos actual, pero es útil imprimir los errores del programa a este `stream` debido a que el usuario puede querer redireccionar esta salida de errores a un archivo de texto tipo `log`, para su posterior análisis.

12. `chmod -x programa.exe` o `chmod 644 programa.exe`, en caso de que se quiera permisos de lectoescritura para el usuario dueño del archivo, y de solo lectura para todo el resto.

13. En este fragmento de código nunca se comprueba el estado de retorno de `fopen()`, el cual puede ser erróneo. En este caso `fopen()` devolverá un puntero a `NULL`, lo cual es inválido, y al momento de leer de ese descriptor de archivo inexistente con `fgets()`, se provocará una violación de segmento.

14. No, Linux es el `kernel` o núcleo de un sistema operativo. Únicamente se encarga de la administración de los recursos de hardware y del ordenamiento de las tareas que estén corriendo en un mismo momento.

15. GCC es un conjunto de compiladores del proyecto GNU. Tiene soporte para lenguajes C, C++, Objective-C, Ada, Fortran, entre otros.
    GDB es un `debugger` del proyecto GNU.

16. Un intérprete de comandos es una interfaz de usuario textual. Acepta la introducción de comandos textuales, normalmente de teclado, los cuales son interpretados y luego ejecutados. Entre los más comúnes, se encuentran `bash`, `zsh` y `ksh` en entornos UNIX y `COMMAND.COM` en entornos DOS.