Tarea 5

Luis Fernando Cantú Díaz de León

A 14 de marzo de 2018

1 Preguntas

1.1 Estudia y entiende los archivos `definiciones.h` y `funciones.c` de ahí, en particular investiga por qué se usan "" en la línea que tiene `#include` en `funciones.c` en lugar de usar `< >`. Investiga el uso de `static` en la definición de variables externas de `funciones.c` .

Se usan comillas ("") en el *header file* cuando el archivo a incluir en el *header* es uno creado por el usuario. Los signos de mayor y menor que (<>) se usan para archivos *header* del sistema; busca el archivo indicado en una lista estándar de directorios del sistema.

En cuanto a `static`, esta palabra se utiliza en un programa con variables globales y funciones para limitar el uso de estas dos últimas al mismo programa.

1.2 Investiga sobre `BLAS`, `CBLAS`, `LAPACK`, `ATLAS` y las operaciones de nivel 1, nivel 2 y nivel 3 de `BLAS` y reporta sobre esta investigación que realizas. Es una investigación que contiene principalmente una descripción sobre los paquetes y niveles anteriores.

- BLAS: Basic Linear Algebra Subprograms. Son funciones que proveen al usuario de los fundamentos básicos para realizar operaciones vectoriales y matriciales.
- BLAS-Nivel 1: Lleva a cabo operaciones escalares, vectoriales y vector-vector.
- BLAS-Nivel 2: Lleva a cabo operaciones matriz-vector.
- BLAS-Nivel 3: Lleva a cabo operaciones matriz-matriz.
- CBLAS: Es la interfaz de BLAS en C.
- LAPACK: Linear Algebra PACKage. Es un paquete de software escrito en FORTRAN 90 y proporciona rutinas para resolver sistemas de ecuaciones lineales simultáneas, soluciones de mínimos cuadrados de sistemas de ecuaciones lineales, problemas de valores propios y problemas de valores singulares. También provee de las factorizaciones de matrices asociadas.

- ATLAS: Automatically Tuned Linear Algebra Software. Proporciona interfaces de BLAS y algunas rutinas de LAPACK para C y FORTRAN 77.

1.3 En la carpeta «analisis-numerico-computo-cientifico/C/BLAS/ejemplos/ ejecuta el programa `dot_product.c` y realiza pruebas con diferentes vectores definidos por ti.

Primero para los vectores definidos en la tarea.

```
gcc -Wall dot_product.c funciones.c -o programa.out -lblas
./programa.out 3
```

```
## ------------
## vector :
## vector[0]=1.00000
## vector[1]=0.00000
## vector[2]=-3.00000
## ------------
## vector :
## vector[0]=5.00000
## vector[1]=8.00000
## vector[2]=9.00000
## ------------
## resultado: -22.000000
```

Luego otros dos vectores diferentes. Cabe recalcar que lo único que se hizo fue modificar el tamaño de los mismos.

```
./programa.out 4
```

```
## ------------
## vector :
## vector[0]=1.00000
## vector[1]=0.00000
## vector[2]=-3.00000
## vector[3]=10.00000
## ------------
## vector :
## vector[0]=5.00000
## vector[1]=8.00000
## vector[2]=9.00000
## vector[3]=4.00000
## ------------
## resultado: 18.000000
```

Y finalmente, otros dos vectores con un tamaño diferente:

```
./programa.out 5
```

```
## ------------
## vector :
## vector[0]=1.00000
## vector[1]=0.00000
## vector[2]=-3.00000
## vector[3]=10.00000
## vector[4]=5.00000
## ------------
## vector :
## vector[0]=5.00000
## vector[1]=8.00000
## vector[2]=9.00000
## vector[3]=4.00000
## vector[4]=3.00000
## ------------
## resultado: 33.000000
```

1.4 Investiga sobre la subrutina de Fortran `ddot` (parámetros que recibe y la salida).

La subrutina `ddots` de FORTRAN sirve para llevar a capo una operación producto punto. Toma como argumentos los siguientes parámetros:

- N: número de elementos en el vector.
- DX, DY: son los vectores a multiplicar.
- INCX: tamaño de los espacios entre los elementos de DX.
- INCY: tamaño de los espacios entre los elementos de DY.

Regresa un escalar resultado del producto punto.

1.5 Haz un programa que utilice la subrutina `daxpy` de Fortran.

La subrutina `daxpy` hace la siguiente operación vectorial:

`daxpy - compute y := alpha * x + y`

Se modificó un poco el archivo `dot_product.c`. Específicamente, se le agregó el parámetro `alpha`. También es importante recalcar que la subrutina sobreescribe los valores del vector Y, y es ahí en donde arroja el *output*.

Probamos el programa con diferentes valores de *alpha*. Primero con $alpha = 1$.

```
gcc -Wall daxpy.c funciones.c -o daxpyy.out -lblas
./daxpyy.out 5 1
```

```
## ------------
## vector :
## vector[0]=1.00000
## vector[1]=0.00000
## vector[2]=-3.00000
## vector[3]=10.00000
## vector[4]=5.00000
## ------------
## vector :
## vector[0]=5.00000
## vector[1]=8.00000
## vector[2]=9.00000
## vector[3]=4.00000
## vector[4]=3.00000
## ------------
## resultado:
## vector[0]=6.00000
## vector[1]=8.00000
## vector[2]=6.00000
## vector[3]=14.00000
## vector[4]=8.00000
```

Luego con $alpha = 2$.

```
./daxpyy.out 5 2
```

```
## ------------
## vector :
## vector[0]=1.00000
## vector[1]=0.00000
## vector[2]=-3.00000
## vector[3]=10.00000
## vector[4]=5.00000
## ------------
## vector :
## vector[0]=5.00000
## vector[1]=8.00000
## vector[2]=9.00000
## vector[3]=4.00000
## vector[4]=3.00000
## ------------
## resultado:
## vector[0]=7.00000
```

```
## vector[1]=8.00000
## vector[2]=3.00000
## vector[3]=24.00000
## vector[4]=13.00000
```

Y finalmente con $alpha = 5$.

```
./daxpyy.out 5 5
```

```
## ------------
## vector :
## vector[0]=1.00000
## vector[1]=0.00000
## vector[2]=-3.00000
## vector[3]=10.00000
## vector[4]=5.00000
## ------------
## vector :
## vector[0]=5.00000
## vector[1]=8.00000
## vector[2]=9.00000
## vector[3]=4.00000
## vector[4]=3.00000
## ------------
## resultado:
## vector[0]=10.00000
## vector[1]=8.00000
## vector[2]=-6.00000
## vector[3]=54.00000
## vector[4]=28.00000
```