



UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

Año 2017 - 2^{do} Cuatrimestre

ALGORITMOS Y PROGRAMACIÓN I (95.11)

TRABAJO PRÁCTICO Nº1

TEMA: Calculador de desempeño académico

FECHA DE ENTREGA: 20 de septiembre de 2017

Integrantes:

- Aguirre Pedro, 97603 pedro.aguirre2395@gmail.com
- Cuenca Matilde, 99319 matcuenca84@gmail.com
- Viegas Bordeira Santiago, 99634 santiago.bordeira@gmail.com

1. Objetivo

El objetivo del trabajo consiste en la realización de un conjunto de aplicaciones en modo consola, escritos en lenguaje ANSI-C89, que permitan implementar un sistema de cálculo de métricas sobre el desempeño académico del usuario.

2. Alcance

Mediante el presente trabajo práctico se busca que el estudiante adquiera y aplique conocimientos sobre los siguientes temas:

- Programas en modo consola
- Directivas al preprocesador C
- Juego de caracteres ASCII
- Tipos enumerativos
- Control de flujo
- Salida de datos con formato
- Funciones
- Modulación
- Arreglos
- Cadena de caracteres

3.1 Funcionamiento del programa

La idea general del programa es la de, a partir de un menú interactivo con el usuario, poder obtener información sobre el nombre, apellido, carrera y promedio académico de quien lo ejecuta. El resultado final es la impresión por pantalla de un mensaje con información sobre los datos mencionados anteriormente. El mensaje respeta la siguiente estructura:

```
Nombre y Apellido, Padrón, Carrera, Materias, Promedio, Aplazos
```

3.2 Descripción del código y criterios utilizados

En el código fuente del programa realizado, se hizo uso de *typedef enum*, conocidos por tipos enumerativos en lenguaje de programación C. Ésta, es una palabra reservada que tiene la función de asignar un nombre alternativo a tipos existentes. En el código se utilizaron dos tipos enumerativos:

- `state_t` representa los estados de la máquina. Para crear la máquina de estados, se definió siguiente tipo enumerativo:

```
typedef enum{

    ST_IDLE,
    ST_MENU_PRINCIPAL,
    ST_MOD_ASIG,
    ST_MOD_REGISTRO,
    ST_NOMBRE,
    ST_PADRON,
    ST_CARRERA,
    ST_ELIMINAR_ASIG,
    ST_NUEVA_ASIG,
    ST_MOD_UNA_ASIG,
    ST_METRICA,
    ST_PROMEDIO,
    ST_MAXIMO,
    ST_MINIMO,
    ST_CANT_ASIG,
    ST_APLAZOS,
    ST_VOLVER,
    ST_FINALIZAR,
    ST_SALIR

} state_t;
```

- `status_t` representa las posibles fallas que pueden ocurrir en el desarrollo del programa.

```
typedef enum {

    ST_OK,
    ST_ERROR,
    ST_OPCION_INVALIDA,
    ST_PUNTERO_NULO,
    ST_SIN_CARRERA

} status_t;
```

Los tipos de enumerativos nombrados fueron definidos en dos *headers* llamados `tp1.h` y `tipos.h`, respectivamente, con el objetivo de hacer una programación modular. De esta manera, si otro módulo del programa usa alguna de estas definiciones, basta con incluir el mismo *header* y así no será necesario volver a repetir los *typedef*.

Se tuvo como consideración el uso de constantes simbólicas (definidas con la orden `#define`) para evitar generar un código “*hardcodeado*”. El término recién mencionado, “hace referencia a una mala práctica en el desarrollo de software que consiste en incrustar datos directamente en el código fuente del programa, en lugar de obtener esos datos de una

fueron fuente externa como un fichero de configuración o parámetros de la línea de comandos, o un archivo de recursos”.⁽¹⁾ Con esto se buscó simpleza y facilidad a la hora de corregir datos del código ya que si esto es lo que desea hacer, con solo modificar el valor de la constante definida en cuestión no es necesario ingresar a la lógica del programa.

Siempre que fue necesario, se definieron dichas constantes simbólicas en archivos de extensión `.h` agrupadas según un criterio de utilidad o uniformidad de los mismos. Dichas constantes simbólicas, son utilizadas por el archivo de extensión `.c`, por lo que los archivos `.h` deben ser incluidos en los archivos `.c`. Además, para las mismas, se utilizó una protección para evitar las inclusiones recursivas que pueden generar fallas en la lógica del programa.

El programa está configurado en idioma español, aunque por omisión, el programa se configura en idioma inglés.

En cuanto a las funciones que se implementaron en el programa, éstas fueron doce. El programa contiene una función encargada del procesamiento de datos devolviendo algún caso de estado que es chequeado en `main`. En el caso de devolver un estado erróneo, `main` se encarga de invocar una función denominada `procesar_errores()`. Esta función contiene una construcción del lenguaje C para decisiones múltiples, denominada `switch`, donde para cada “*case*” (casos con escenarios esperados e inesperados) de la misma se contemplan todos los posibles errores del programa para los cuales se debe realizar alguna acción. Luego de detectar qué tipo de problema ha ocurrido, se imprime por pantalla un mensaje que informa al usuario el motivo por el cual el programa no ha sido ejecutado con éxito. Estos mensajes están definidos en archivos denominados `espanol.h` e `ingles.h`, donde se almacenarán en idioma español e inglés respectivamente.

Para la lectura de las opciones ingresadas por el usuario dentro del menú del programa fue necesario implementar una función que se llamó `leer_int()`. Esta función simplemente se encarga de leer un número entero ingresado por el usuario, almacenarlo y retornar un estado, que dependerá de si la función se pudo realizar con éxito o si hubo un error en su desarrollo.

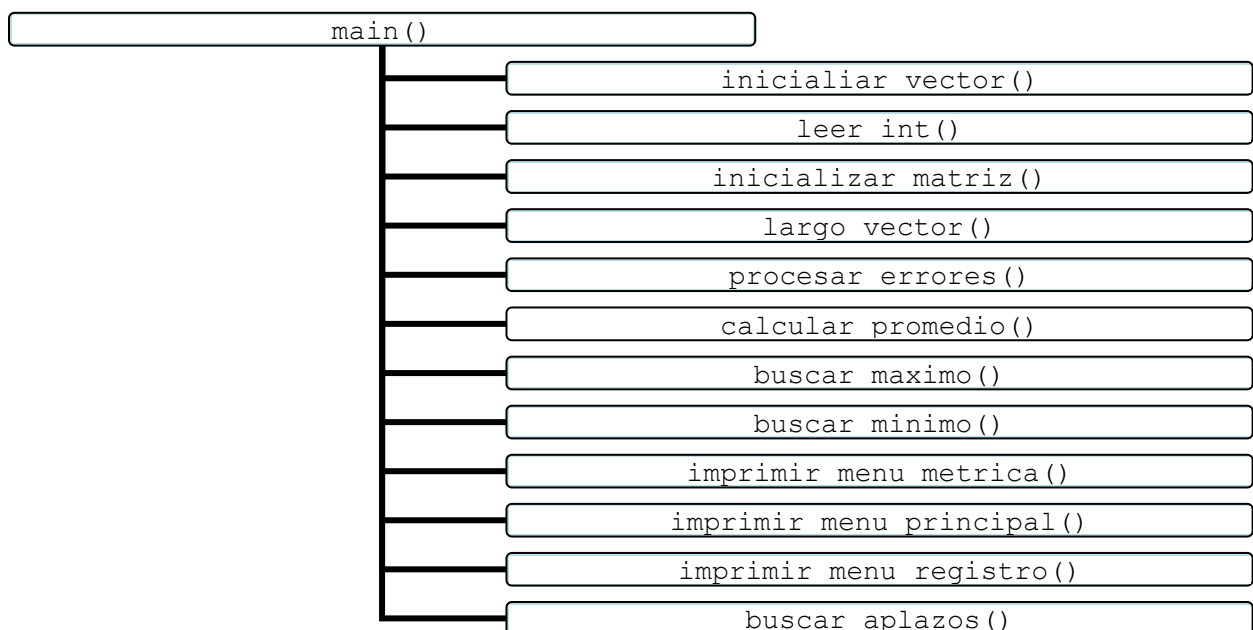
Se desarrollaron dos funciones capaces de inicializar tanto una matriz como un vector. Dichas funciones, pondrán a todos los elementos pertenecientes al vector o a la matriz en cero (“0” en el código). Para el caso de la matriz se utilizó `inicializar_matriz()` y para el vector `inicializar_vector()`. Además se realizó una función, `largo_vector()`, capaz de contar la cantidad de elementos cargados por el usuario gracias a la inicialización mencionada

anteriormente ya que permitió, recorriendo el vector que se le pasa, comparar con '\0'. El resultado de ejecución de dicha función será un número que representa la cantidad de elementos ingresados en dicho vector (será el número de elementos distintos de '\0'). Esta función, al igual que la mencionada con anterioridad, devuelve un estado.

Para calcular la métrica se hizo uso de cinco funciones propias de este menú. En primer lugar, la función que ha sido comentada `largo_vector()`, que indica la cantidad de asignaturas que ha ingresado el usuario; luego, `calcular_promedio()`, encargada de obtener el promedio de una serie de notas académicas ingresadas por el usuario; `buscar_maximo()` y `buscar_minimo()` encargadas de recorrer un vector buscando máximos y mínimos respectivamente y por último, `buscar_aplazos()` cuya función es calcular la cantidad de aplazos en base a las notas ingresadas por el usuario.

Por último, para la impresión de los menús de opciones a elegir se hizo uso de una función capaz de reunir todos los mensajes e imprimirlos en cuyo caso corresponda. Al presentar más de un menú de opciones en el programa se utilizaron nombres distintos para cada función aunque la lógica de todas es la misma, imprimir una serie de opciones.

A continuación, se muestra en un diagrama cómo es la invocación de las funciones:



Teniendo en cuenta las características del código, cabe mencionar el no uso de la función de biblioteca `scanf()` para lectura de datos de entrada. Esta función lee datos formateados de la entrada estándar. Cuando se habla de datos formateados, se hace referencia a que los datos se pueden presentar como enteros, flotantes, caracteres, entre otros y la función almacena esta

información y devolverá el número de datos que se han introducido correctamente. Las desventajas y problemáticas que surgen en el uso de esta función son varias. En primer lugar, la particularidad de dejar un carácter almacenado en el *buffer*. `scanf()` lee del *buffer* hasta encontrar el tipo de dato preestablecido y, en caso de haber un `'\n'` (salto de línea), éste lo deja almacenado. Esto genera inconvenientes a la hora de realizar una nueva lectura, ya que habrá un elemento previamente almacenado dando lugar a lecturas incorrectas y almacenamientos no deseados.

A su vez, otra desventaja es que `scanf()`, mientras busca los valores de entrada “salta” sobre los espacios en blanco; “ignora los blancos y tabuladores que estén en su cadena de formato (Kernighan y Ritchie, 1991:177)”. Por último, otra cuestión por la que no fue utilizada esta herramienta fue que se va de cotas. El hecho de no pedir la cantidad de caracteres que debe leer, genera que la función lea hasta encontrar algo erróneo o se ingrese el carácter de salto de línea; puede generar una lectura de algo no deseado. La robustez de un programa se ve alterada por el uso de esta función. El hecho de ser robusto implica poder desempeñarse sin grandes errores frente a circunstancias no previstas. Lo mínimo que debe hacer un programa es funcionar, si esto no es posible, el programa falla y no es más muchas líneas de código. Esta cuestión se ve reflejada en lo dicho por Ghezzi, Jazayeri y Mandrioli en *Fundamentals of Software Engineering*, “*a program is robust if it behaves “reasonably,” even in circumstances that were not anticipated...*”. Un programa es robusto si se comporta razonablemente aún en circunstancias que no son anticipadas.

Una alternativa que se pensó como solución a este inconveniente fue en principio, el uso de la función `getchar()` empleado de la siguiente manera:

```
while((c = getchar()) != '\n' && c!=EOF)
```

Sin embargo, si bien el programa compila con normalidad, el uso de esta alternativa no fue adoptada por este grupo de trabajo por su complejidad. La solución que se tuvo en cuenta con el fin de generar una lectura de código menos problemática y con un margen de error mucho menor fue el uso de la función `fgets()`. Esta función, trabaja de forma similar a la función `gets()`, con la diferencia que esta última permite establecer el máximo de caracteres que pueden leerse. Primero se establece dónde se quiere copiar la línea leída y a continuación el máximo número de caracteres, incluyendo el `'\n'`, que se pueden leer. Esto hace que la función lea los caracteres ingresados hasta detectar un `'\n'` o hasta que haya copiado el máximo de elementos asignados. Como último parámetro de la función se debe establecer de dónde se obtendrán los datos; éste será `stdin` (*standard input*) que por defecto es el teclado.

4. Resultados de ejecución y reseñas sobre problemas encontrados en el programa

Durante el desarrollo del código, se fueron presentando grandes números de fallas y errores al momento de compilar como por ejemplo, olvido de definición de alguna macro, la errónea implementación de punteros y memoria dinámica.

Uno de los mayores desafíos fue el de lograr comprender la causa de un salto de línea inesperado observado al imprimir por pantalla. Esta falla se daba una vez obtenido una cadena de caracteres con la función de librería `fgets()`. Consultando bibliografía y con la ayuda de Internet, se comprendió que la causa era que esta función almacena en la cadena de caracteres los elementos ingresados por el usuario, seguido el carácter salto de línea (`'\n'`) y por último el `'\0'`. Esto generaba que, al momento de imprimir la cadena, se imprimiera también el salto de línea, por lo que fue necesario remover dicho salto de línea de la cadena. Para lograr esto, se hizo uso de la función `strchr(cadena, c)`, que devuelve un puntero a la primer aparición del carácter “c” en la cadena “cadena”. Una vez localizado el salto de línea, se lo reemplazó con un `'\0'`.

A continuación, algunas imágenes de los resultados de ejecución:

```
Por favor, elija una opción
1)Modificar registro personal
2)Modificar asignatura
3)Métrica
4)Finalizar
0)Salir
2
Asignaturas cargadas:
- algebra (8)
- algoritmos (10)
Por favor, elija una opción
1)Ingresar nueva asignatura
0)Volver
1
Por favor, ingrese su asignatura
fisica
Por favor, ingrese su nota
2
Asignaturas cargadas:
- algebra (8)
- algoritmos (10)
- fisica (2)
Por favor, elija una opción
1)Ingresar nueva asignatura
0)Volver
0
Por favor, elija una opción
1)Modificar registro personal
2)Modificar asignatura
3)Métrica
4)Finalizar
0)Salir
4
Juan Perez, 99632, Ing. Electrónica, 3, 6.666667, 1
```

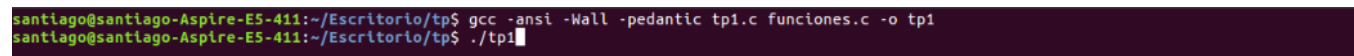
```
Bienvenido/a Por favor, elija una opción
1)Modificar registro personal
2)Modificar asignatura
3)Métrica
4)Finalizar
0)Salir
2
No tiene ninguna asignatura cargada
1)Ingresar nueva asignatura
0)Volver
1
Por favor, ingrese su asignatura
Algoritmos y Programacion
Por favor, ingrese su nota
10
Asignaturas cargadas:
- Algoritmos y Programacion (10)
Por favor, elija una opción
1)Ingresar nueva asignatura
0)Volver
0
Por favor, elija una opción
1)Modificar registro personal
2)Modificar asignatura
3)Métrica
4)Finalizar
0)Salir
```

5. Conclusión

Con este trabajo práctico se concluye que el desarrollo de un menú de opciones requiere de buenas prácticas de programación que se intentaron adquirir a lo largo del desarrollo del programa, como por ejemplo la implementación de funciones, ya que facilita no solo la comprensión del código fuente sino también hace a la simpleza del mismo y al correcto funcionamiento del programa.

5. Script de compilación

```
gcc -ansi -Wall -pedantic tp1.c funciones.c -o tp1
```



```
santiago@santiago-Aspire-E5-411:~/Escritorio/tp$ gcc -ansi -Wall -pedantic tp1.c funciones.c -o tp1
santiago@santiago-Aspire-E5-411:~/Escritorio/tp$ ./tp1
```

6. Bibliografía

(1) Descripción del término “harcodeado”, Wikipedia, https://es.wikipedia.org/wiki/Hard_code

- KERNIGHAN, BRIAN W., RIETCHIE, DENNNIS M., El Lenguaje de Programación en C, Segunda Edición, México 1991.
- GHEZZI, Carlo – JAZAYERI, Mehdi – MANDRIOLI, Dino, Fundaments of Software Engineering, New Jersey, 1991.