Desarrollo de una biblioteca de ingreso de datos.

Programación I – Laboratorio I.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Esp. Ing. Ernesto Gigliotti*

Revisores: *Mg. Mauricio Dávila*

Versión : 1.3

Internacional.

Índice de contenido

1 Introducción

Hasta el momento hemos utilizado la función scanf en el lugar del código donde requeríamos pedir al usuario que ingrese un dato por la consola. Esto no solo requería llamar a la función, sino también imprimir previamente un mensaje, y posterior al ingreso del dato, una validación y quizá reintentos.

```
printf("Ingrese un numero de 1 a 99:");
scanf("%d",&numero);
if(numero<1 || numero >99 )
    printf("Error");
```

Esto provoca que cada vez que queremos pedirle al usuario información, repitamos una considerable cantidad de código que puede ser abstraído en una función.
Crearemos una biblioteca "utn.h" y "utn.c" en donde agregaremos las funciones que nos permitirán pedirle datos al usuario y validarlos.

1.1 Función para pedir un número por consola

Definiremos el siguiente prototipo:

```
int utn_getNumero(int* pResultado,
           char* mensaje,
           char* mensajeError,
           int minimo,
           int maximo,
           int reintentos)
```

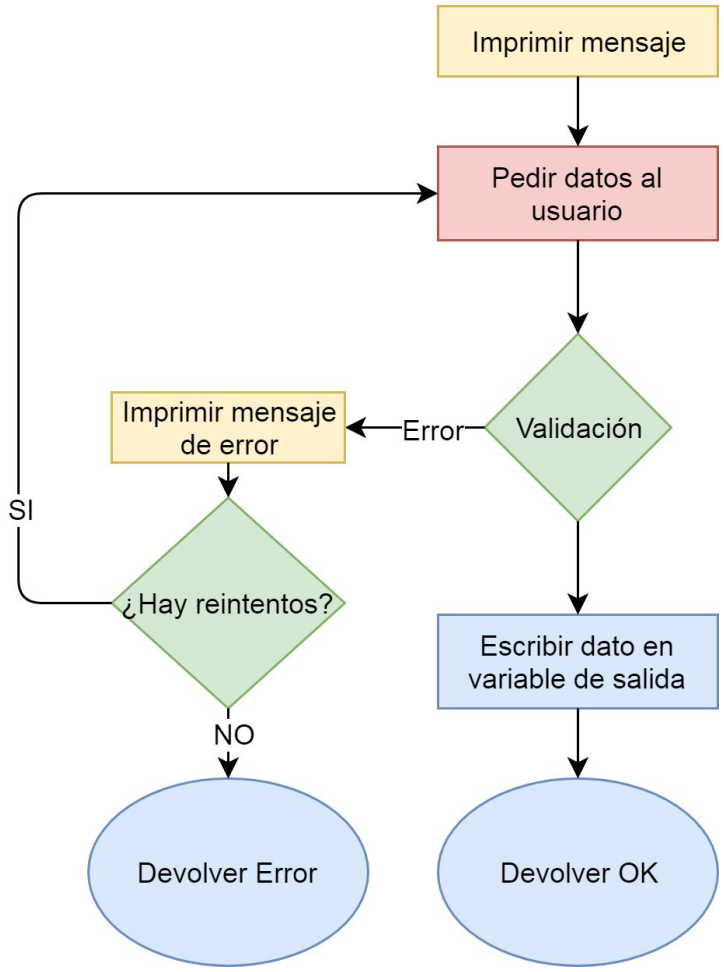
Esta función recibe los siguientes argumentos:

- pResultado: Puntero a variable donde se escribirá el valor ingresado en el caso de ser correcto.
- mensaje: Puntero a cadena de caracteres con mensaje a imprimir antes de pedirle al usuario datos por consola.
- mensajeError: Puntero a cadena de caracteres con mensaje de error en el caso de que el dato ingresado no sea válido.
- minimo: Valor mínimo admitido (inclusive)
- maximo: Valor máximo admitido (inclusive)
- reintentos: Cantidad de veces que se le volverá a pedir al usuario que ingrese un dato en caso de error.

A continuación se explicará el desarrollo de esta función junto con todas las funciones internas auxiliares que se requieren.

2 Desarrollo de la función

La función contará con el siguiente diagrama:



Dentro de la función, deberemos imprimir el mensaje, antes de que el programa se bloquee con la función scanf que le pedirá datos al usuario.
Luego se validará que el número ingresado se encuentre dentro del rango determinado por los argumentos "maximo" y "minimo".

En el caso de que se considere el número como válido, se ejecutará la sentencia "break" y se saldrá del bucle "while". En esta condición la variable "reintentos" todavía es mayor a cero, por lo que se escribe en el puntero pasado como argumento, el valor ingresado, el cual se encuentra en la variable "num".

Primer desarrollo de la función utn_getNumero()

```
int utn_getNumero(int* pResultado,
           char* mensaje,
           char* mensajeError,
           int minimo,
           int maximo,
           int reintentos)
{
     int ret;
     int num;

     while(reintentos>0)
     {
          printf(mensaje);
          scanf("%d",&num);
          if(num<=maximo && num>=minimo)
              break;

          reintentos--;
          printf(mensajeError);
     }

     if(reintentos==0)
     {
         ret=-1;
     }
     else
     {
         ret=0;
         *pResultado = num;
     }

     return ret;
}
```

2.1 Validación simple de datos ingresados

En el ejemplo anterior, se utilizó la función scanf para obtener los datos ingresados por el usuario. La desventaja de utilizar esta función es que si esperamos recibir un número (utilizando el modificador de formato %d) y escribimos un texto, la función no escribirá en la variable, ya que no podrá convertir el texto a número. Puede probarse este comportamiento haciendo un programa que ejecute nuestra función e ingresando una palabra en vez de un número.

Este error es indicado por el valor de retorno de scanf, el cual coincide con la cantidad de variables que pudo escribir. Una manera de validar si el usuario realmente ingresó un número y este pudo ser guardado en la variable "num" es preguntar si scanf devolvió 1 (indicando que pudo escribir la variable).

Modificación de la función validando que el valor ingresado sea numérico:

```
int utn_getNumero(int* pResultado,
           char* mensaje,
           char* mensajeError,
           int minimo,
           int maximo,
           int reintentos)
{
     int ret;
     int num;

     while(reintentos>0)
     {
          printf(mensaje);
          if(scanf("%d",&num)==1)
          {
              if(num<=maximo && num>=minimo)
                  break;
          }
          reintentos--;
          printf(mensajeError);
     }

     if(reintentos==0)
     {
         ret=-1;
     }
     else
     {
         ret=0;
         *pResultado = num;
     }

     return ret;
}
```

Si probamos nuevamente la función ingresando un texto en vez de un número con el siguiente programa:

```
int main()
{
    int n;
    utn_getNumero(&n,"ingrese el numero:","error",1,9,3);
    printf("num ingresado:%d",n);
    return 0;
}
```

Nos encontraremos con el siguiente resultado al ingresar la palabra “hola”:

```
ingrese el numero:hola
erroringrese el numero:erroringrese el numero:errornum ingresado:2
Process returned 0 (0x0)   execution time : 2.284 s
Press any key to continue.
```

Como se observa, luego de ingresar "hola" y presionar ENTER, se muestra el mensaje de error, y al volver a ejecutar scanf, la misma toma como si se hubiera ingresado un texto y presionado ENTER una y otra vez, hasta agotar los reintentos y dejar de ejecutar la función.

Esto es debido a que al producirse el error en la lectura del texto ingresado por consola, la función scanf no descarta el texto previamente ingresado y este queda en el buffer de entrada "standard input" del programa. Al volver a ejecutar scanf, vuelve a tratar de leer el texto que todavía está almacenado allí y vuelve a dar el mismo error. Para borrar el texto almacenado en el buffer del standard input, deberemos ejecutar la función fflush(). De esta manera al volver a ejecutar scanf, el programa volverá a quedarse bloqueado esperando que el usuario ingrese algo por consola.

IMPORTANTE: En Linux en lugar de fflush() se debe utilizar la función __fpurge(), para lo cual es necesario incluir el archivo stdio_ext.h .

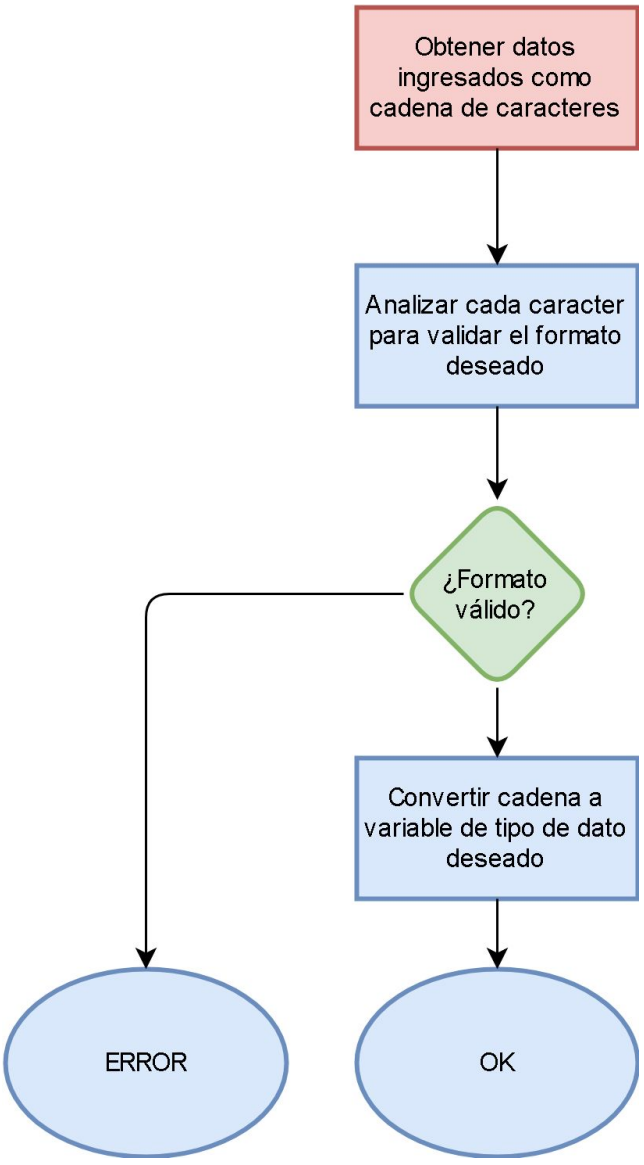
Modificación de la función borrando el buffer de texto ingresado en caso de error:

```
int utn_getNumero(int* pResultado,
           char* mensaje,
           char* mensajeError,
           int minimo,
           int maximo,
           int reintentos)
{
   int ret;
   int num;

   while(reintentos>0)
   {
        printf(mensaje);
        if(scanf("%d",&num)==1)
        {
            if(num<=maximo && num>=minimo)
                break;
        }
        fflush(stdin); //EN LINUX UTILIZAR __fpurge(stdin)

        reintentos--;
        printf(mensajeError);
   }

   if(reintentos==0)
   {
       ret=-1;
   }
   else
   {
       ret=0;
       *pResultado = num;
   }
   return ret;
}
```

2.2 Validación avanzada de datos ingresados

La función desarrollada en el punto anterior nos permite una validación simple del texto ingresado, scanf solo nos indica si pudo o no convertir el texto ingresado a un número, y guardarlo en una variable numérica del tipo int.

¿Qué ocurre si queremos validar otro tipo de formato en el texto ingresado? por ejemplo un teléfono o cuit, en el cual existen guiones o puntos como parte del texto. No se podrá utilizar la función scanf con el modificador de formato "%d". La lógica a seguir para resolver la validación de un formato en particular en el texto, es la siguiente:



Obtendremos los datos ingresados por el usuario como una cadena de caracteres, es decir que almacenaremos caracter por caracter el texto ingresado. Luego podremos analizar el texto ingresado en este formato, para ver si cumple con el formato deseado (texto con solo números, con un solo punto, con un guión en una posición en particular, etc.)
Una vez que la validación es correcta, podremos convertir ese texto en una variable de otro tipo en el caso que sea posible (por ejemplo si son solo números podremos convertir el texto a

número y guardarlo en una variable del tipo int, o float)

Como puede observarse, lo que antes era solo una línea de código:

```
if(scanf("%d",&num)==1)
```

Ahora se debe transformar en algo más complejo, por lo que haremos esto en una nueva función. Para este ejemplo en particular, en donde queremos obtener un número en una variable del tipo int, crearemos la función "getInt" la cual hará todo lo indicado en el diagrama anterior y suplantará a la línea del if con la llamada a scanf.
Modificación de la función utilizando la función getInt en vez de scanf:

```
int utn_getNumero(int* pResultado,
           char* mensaje,
           char* mensajeError,
           int minimo,
           int maximo,
           int reintentos)
{
   int ret;
   int num;

   while(reintentos>0)
   {
        printf(mensaje);
        if(getInt(&num)==1)
        {
            if(num<=maximo && num>=minimo)
                break;
        }
        fflush(stdin); //EN LINUX UTILIZAR __fpurge(stdin)

        reintentos--;
        printf(mensajeError);
   }

   if(reintentos==0)
   {
       ret=-1;
   }
   else
   {
       ret=0;
       *pResultado = num;
   }

   return ret;
}
```

Deberemos definir la función "getInt" la cual será la siguiente:

```
static int getInt(int* pResultado)
{
    char buffer[64];
    scanf("%s",buffer);

    *pResultado = atoi(buffer);
    return 1;
}
```

En esta primera versión de getInt, obtenemos el texto ingresado utilizando scanf con el modificador de formato "%s". El texto quedará almacenado como una cadena de caracteres en la variable buffer.
A continuación deberíamos llamar a alguna función que se encargue de validar la cadena según algún criterio, por ejemplo que solo tenga caracteres numéricos. Por el momento omitiremos este paso y supondremos que el texto ingresado es numérico.
Por último, se debe transformar la cadena de caracteres a un valor del tipo int. Para ello utilizamos la función "atoi" la cual recibe una cadena y devuelve un valor del tipo int. Para utilizar esta función deberemos incluir la biblioteca "stdlib.h".

2.3 Validación de la cadena de caracteres ingresada

A continuación validaremos que la cadena obtenida sea numérica, para ello desarrollaremos la función "esNumerica" la cual recibirá una cadena y devolverá 1 en el caso de que los caracteres sean todos numéricos:

```
static int esNumerica(char* cadena)
{
	int i=0;
	int retorno = 1;
	if(cadena != NULL && strlen(cadena) > 0)
	{
		while(cadena[i] != '\0')
		{
			if(cadena[i] < '0' || cadena[i] > '9' )
			{
				retorno = 0;
				break;
			}
			i++;
		}
	}
	return retorno;
}
```

Esta función recorre todos los caracteres de la cadena, si alguno no es numérico, el bucle termina y la variable "ret" quedará cargada con -1. Si el bucle termina por la detección del caracter de fin de cadena (\0) la variable ret se carga con 1.

Ahora podemos utilizar esta función en getInt:

```
static int getInt(int* pResultado)
{
    int ret=-1;
    char buffer[64];
    scanf("%s",buffer);
    if(esNumerica(buffer))
    {
        *pResultado = atoi(buffer);
        ret=1;
    }
    return ret;
}
```

De esta forma completamos la función utn_getNumero, la cual utiliza internamente las funciones:

getInt: Pide un texto al usuario, lo almacena como cadena, valida y convierte el texto a numero y lo devuelve como int.
esNumerica: Recibe una cadena de caracteres y devuelve 1 en el caso de que el texto este compuesto solo por números.

2.4 Agregando límites de acceso a memoria

Si bien la versión obtenida hasta el momento de la función utn_getNumero es bastante aceptable, tiene un problema que es muy importante de resolver: Si el texto ingresado es más grande que el buffer definido en getInt de 64 caracteres, la función scanf seguirá escribiendo en porciones de memoria del programa, pudiendo provocar el comportamiento errático del mismo.

```
char buffer[64];
scanf("%s",buffer);
```

Este problema no tiene una solución provista por la función scanf. Por lo que se debe dejar de usarla.

Utilizaremos la función fgets en su lugar, la misma cuenta con el siguiente prototipo:

```
char* fgets(char* str, int n, FILE* stream)
```

La función fgets recibe como argumentos el array de char donde se escribirá, el tamaño de dicho array (esto permite limitar que no se escriba en posiciones de memoria más allá de dicho array) y el puntero a una estructura del tipo FILE, en donde pasaremos la variable “stdin” para que se lea de la consola (esta función también se utiliza para leer texto de archivos).
La función devolverá el mismo puntero str pasado como argumento o NULL en caso de error.

Reemplazaremos la función scanf por fgets en la función getInt:

```
static int getInt(int* pResultado)
{
    int ret=-1;
    char buffer[64];
    fgets(buffer,sizeof(buffer),stdin);
    if(esNumerica(buffer))
    {
        *pResultado = atoi(buffer);
        ret=1;
    }
    return ret;
}
```

Esta solución evitará que el usuario intencionada o no intencionadamente escriba zonas de memoria del programa al ingresar un texto más grande que el buffer definido.

2.5 Diferencias entre fgets y scanf

La diferencia principal que existen entre estas dos funciones es que al ingresar un texto y presionar ENTER, la función scanf escribirá en el buffer provisto el texto sin el carácter que representa el enter (\n), mientras que la función fgets sí escribirá dicho carácter.
Por ejemplo si se ingresa el texto "hola"+ENTER, así quedarán cargados los buffers:

Con scanf:

'H'	'O'	'L'	'A'	'\0'

Con fgets:

'H'	'O'	'L'	'A'	'\n'	'\0'

Sin embargo, si se escribe un texto más largo que el tamaño pasado a fgets, la función terminará la cadena con '\0' pero no existirá el '\n', ya que el mismo no entró en el buffer.

Por ejemplo si se ingresa el texto "hola1234"+ENTER, y el buffer es de 6 lugares, así quedará cargado:

Con fgets:

'H'	'O'	'L'	'A'	'1'	'\0'

Debemos eliminar el ENTER al final de la cadena en el caso de que exista, para lograr que se comporte igual que scanf y además para evitar que la función de validación esNumero nos dé un error, ya que al encontrar el '\n' en la cadena, la misma retornará que la cadena no es numérica.

A continuación se muestra una solución simple la cual consiste en agregar un '\0' en la última posición de la cadena (determinada por la función strlen). Para ello escribimos una nueva función que llamaremos "myGets" la cual utiliza internamente fgets() y contempla esta situación.

```
static int myGets(char* cadena, int longitud)
{
	int retorno=-1;
	if(cadena != NULL && longitud >0 && fgets(cadena,longitud,stdin)==cadena)
	{
		fflush(stdin); // fflush o __fpurge
		if(cadena[strlen(cadena)-1] == '\n')
		{
			cadena[strlen(cadena)-1] = '\0';
		}
		ret=0;
	}
	return retorno;
}
```

Esta función la llamaremos en lugar del fgets en la función getInt().

A continuación se expone la biblioteca completa "utn.c" con el código de las funciones explicadas:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static int esNumerica(char* cadena);
static int getInt(int* pResultado);
static int myGets(char* cadena, int longitud);
/**
 * \brief      Lee de stdin hasta que encuentra un '\n' o hasta que haya copiado en cadena
 *             un máximo de 'longitud - 1' caracteres.
 * \param pResultado Puntero al espacio de memoria donde se copiara la cadena obtenida
 * \param longitud Define el tamaño de cadena
 * \return Retorna 0 (EXITO) si se obtiene una cadena y -1 (ERROR) si no
 *
 */
static int myGets(char* cadena, int longitud)
{
	if(cadena != NULL && longitud >0 && fgets(cadena,longitud,stdin)==cadena)
	{
		fflush(stdin); //  LINUX-> __fpurge o WIN-> fflush o MAC-> fpurge
		if(cadena[strlen(cadena)-1] == '\n')
		{
			cadena[strlen(cadena)-1] = '\0';
		}
		return 0;
	}
	return -1;
}

/**
 * \brief Verifica si la cadena ingresada es numerica
 * \param pResultado Puntero al espacio de memoria donde se dejara el resultado de la funcion
 * \return Retorna 0 (EXITO) si se obtiene un numero entero y -1 (ERROR) si no
 *
 */
static int getInt(int* pResultado)
{
    int retorno=-1;
    char buffer[64];

    if(pResultado != NULL)
    {
       if(myGets(buffer,sizeof(buffer))==0 && esNumerica(buffer))
       {
			*pResultado = atoi(buffer);
			retorno = 0;
		}
    }
    return retorno;
}
```

```
/**
 * \brief Verifica si la cadena ingresada es numerica
 * \param cadena Cadena de caracteres a ser analizada
 * \return Retorna 1 (vardadero) si la cadena es numerica y 0 (falso) si no lo es
 */
static int esNumerica(char* cadena)
{
	int i=0;
	int retorno = 1;
	if(cadena != NULL && strlen(cadena) > 0)
	{
		while(cadena[i] != '\0')
		{
			if(cadena[i] < '0' || cadena[i] > '9' )
			{
				retorno = 0;
				break;
			}
			i++;
		}
	}
	return retorno;
}

/**
 * \brief Solicita un numero al usuario, leuego de verificarlo devuelve el resultado
 * \param pResultado Puntero al espacio de memoria donde se dejara el resultado de la funcion
 * \param mensaje Es el mensaje a ser mostrado
 * \param mensajeError Es el mensaje de Error a ser mostrado
 * \param minimo Es el numero maximo a ser aceptado
 * \param maximo Es el minimo minimo a ser aceptado
 * \return Retorna 0 si se obtuvo el numero y -1 si no
 */
int utn_getNumero(int* pResultado, char* mensaje, char* mensajeError, int minimo, int
maximo, int reintentos)
{
	int bufferInt;
	int retorno = -1;
	while(reintentos>0)
	{
		reintentos--;
		printf("%s",mensaje);
		if(getInt(&bufferInt) == 0)
		{
			if(bufferInt >= minimo && bufferInt <= maximo)
			{
				*pResultado = bufferInt;
				retorno = 0;
				break;
			}
		}
		printf("%s",mensajeError);
	}
	return retorno;
}
```

3 Pasos a seguir

Este documento provee las bases para la realización de una biblioteca que permita pedir al usuario toda tipo de datos, no solamente números enteros positivos como en el ejemplo. Los pasos a seguir son incorporar el resto de las funciones, siguiendo la misma lógica que la utilizada, las funciones que se usarán en forma externa pueden ser:

utn_getNumero
utn_getNumeroConSigno
utn_getNumeroConDecimales
utn_getTelefono
utn_getDNI
utn_getCUIT
utn_getEmail
utn_getTexto

Cada una de estas funciones utilizará funciones internas, para obtener datos por consola y para validarlos, aquí se dan algunos ejemplos de algunas funciones internas necesarias:

utn_getNumero:
- getInt
- esNumero

utn_getNumeroConDecimales:
- getFloat
- esNumeroDecimal

utn_getTexto
- getString

utn_getEmail:
- getString
- esEmail