**Universidad Autónoma de Baja California**



**Algoritmos y estructuras de datos**

**Practica #1. Arreglos de carácter**

**Alumno: Caudillo Sánchez Diego**

**Matricula: 1249199**

**Grupo: 551**

**Docente: Alma Leticia Palacios Guerrero**

**Fecha de entrega: 15/Feb/2019**

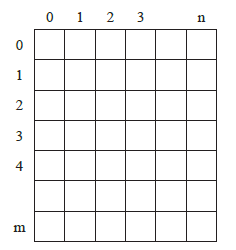
**Competencia**

Utilizar creativamente los conceptos de arreglos de caracteres y sus operaciones para elaborar una propuesta novedosa y eficiente para problemas de ingeniería, preservando la integridad de la información.

**Introducción**

Un *array* o *arreglo* (lista o tabla) es una secuencia de datos del mismo tipo. Los datos se llaman elementos del *array* y se numeran consecutivamente 0, 1, 2, 3 ... El tipo de elementos almacenados en el *array* puede ser cualquier dato simple de Java o de un tipo previamente declarado como una clase. Normalmente, el *array* se utiliza para almacenar tipos tales como *char*, *int* o *float*. Un *array* puede contener, por ejemplo, la edad de los alumnos de una clase, las temperaturas de cada día de un mes en una ciudad determinada o el número de personas que residen en una comunidad. Cada ítem del *array* se denomina *elemento*. Los elementos de un *array* se numeran, como ya se ha comentado, consecutivamente 0, 1, 2, 3, ... Estos números se denominan *valores índice* o *subíndice* del *array*. El término “subíndice” se utiliza ya que especifica, igual que en matemáticas, una secuencia tal como a0, a1, a2... Estos números localizan la posición del elemento dentro del *array*, proporcionando *acceso directo* al *array*.

Los *arrays multidimensionales* son aquellos que tienen más de una dimensión y, en consecuencia, más de un índice. Los más usuales son los de dos dimensiones, conocidos también por el nombre de *tablas* o *matrices*. Sin embargo, es posible crear *arrays* de tantas dimensiones como requieran sus aplicaciones, ya sean tres, cuatro o más. Un *array* de dos dimensiones *(m* × *n)* equivale a una tabla con múltiples filas y múltiples columnas.



Estructura de un array de dos dimensiones

En el *array* bidimensional de la figura, si las filas se etiquetan de 0 a m y las columnas de 0 a n, el número de elementos que tendrá el *array* será el resultado del producto (m+1) \* (n+1). El sistema de localizar un elemento es por las coordenadas representadas por su número de fila y su número de columna *(a, b)*.**Codificación**

/\*

Materia: Algoritmos y estructuras de datos

Matricula: 1249199

Fecha de entrega: 15/Feb/2019

DESCRIPCION

El IMSS identifica a todos los trabajadores que cotizan con un número de seguro social

(NSS). Este número no es aleatorio, se conforma por 11 dígitos que se general a partir

de los datos del trabajador. Los dos primeros dígitos son el número que corresponde a

la delegación del IMSS en la que se registró el trabajador por primera vez. Los siguien-

tes 2 dígitos son los dos últimos dígitos del año en que el trabajador se registró por

primera vez. Los siguientes dígitos son los dos últimos del año de nacimiento del traba-

jador. Luego siguen 4 dígitos que es un número consecutivo de inscripción asignado por

la oficina administrativa del IMSS. Finalmente, el IMSS asigna un dígito verificador que

resulta de multiplicar los 10 números anteriores por la secuencia 1-2-1-2-1-2-1-2-1-2.

INSTRUCCIONES

Se pide que diseñe e implemente una función que reciba los datos necesarios para

obtener el NSS y retorne una cadena con el NSS de la persona.

NOTAS

◘ Todas las capturas deben estar en funciones y validadas.

◘ Debe haber una función por cada tipo de dato.

◘ Los números de delegación son proporcionados por el usuario,

son números de dos dígitos.

◘ Debe utilizar la misma función para capturar todos los datos que sean del mismo

tipo, es decir, solo debe haber una función para enteros, otra para cadenas y una

más para flotantes.

◘ Las funciones de captura solo pueden capturar un dato a la vez.

◘ Cada función debe realizar solo una tarea.

◘ Evite el uso de variables globales.

\*/

//Librerias a utilizar

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

/\*Estructura con los datos requeridos para generar un NSS\*/

typedef struct

{

char anio\_reg[5];

char delegacion[3];

char anio\_nac[5];

char num\_consecutivo[5];

}info\_t;

//Headers

void covierteCadena(char\* seguro\_social, int\* persona);

void capturarDatos(info\_t \*persona);

int valida(int dato, int min, int max);

int generarNumeroVerificador(info\_t persona, int\* nss);

int sumatoria(int dato, int count);

int verificarRangoValores(char \*dato, int min, int max);

void errorMessage(void);

void verificaFormato(char\* seguro\_social);

/\* INICIA LA FUNCION MAIN \*/

int main(int argc, char const \*argv[])

{

/\*Declaración de variables\*/

char seguro\_social[12]; // cadena para guardar los 11 digitos del NSS

info\_t persona; // Estructura con la informacion para poder crear un nuevo NSS

int nss[6]; // arreglo auxiliar de enteros para determinar el digito verificador

srand(time(NULL)); // semilla para generar numero aleatorios.

capturarDatos(&persona);

nss[4] = generarNumeroVerificador(persona, nss);// Se guarda el numero verificador en el array que contiene los primeros 10 numeros del NSS.

covierteCadena(seguro\_social, nss);

if(strlen(seguro\_social) < 11) verificaFormato(seguro\_social); // si la cadena es de tamaño menor a 10, entonces se le da formato.

puts(seguro\_social); // impresion de la cadena

return 0;

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DESCRIPCION:

Función que concatena la informacion en forma de cadena para asi

obtener el NSS de una persona.

PARAMETROS

- seguro\_social: cadena de caracteres donde se va a guardar el NSS (destino)

- nss: arreglo de enteros con el numero de seguro social, para convertirlos

a cadena y asignarlos en el arreglo destino.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void covierteCadena(char\* seguro\_social, int\* nss)

{

char aux[6];

itoa(nss[0], aux, 10); // convierte los digitos guardados en el array de enteros.

strcpy(seguro\_social, aux); // Después los copia en la cadena llamada seguro social.

itoa(nss[1]%100, aux, 10); // A partir de esto, se concatenan las demas cadenas

// realizando el mismo método con la función itoa que

// convierte de entero a cadena.

strcat(seguro\_social, aux);

itoa(nss[2]%100, aux, 10);

strcat(seguro\_social, aux);

itoa(nss[3], aux, 10);

strcat(seguro\_social, aux);

itoa(nss[4],aux, 10);

strcat(seguro\_social, aux);

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DESCRIPCION:

Función que sirve para capturar los datos requeridos para crear un nuevo NSS.

PARAMETROS

◘ persona: estructura, en la cual vienen los campos para llenar la informacion

de la persona a la cual se le ve a generar un NSS.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void capturarDatos(info\_t \*persona)

{

do

{

printf("Numero de delegacion: "); gets(persona->delegacion); // Captura de la cadena con la funcion gets

// se utiliza el operador flecha ya que la estructura

// es un apuntador y esa es la manera correcta en como

// se accede a los datos dentro de la estructura.

if(verificarRangoValores(persona->delegacion,1,99) != 0) errorMessage();

} while(verificarRangoValores(persona->delegacion,1,99) != 0);

do

{

printf("A%co de nacimiento: ", 164); gets(persona->anio\_nac);

if(verificarRangoValores(persona->anio\_nac,1900,2019) != 0) errorMessage();

} while (verificarRangoValores(persona->anio\_nac,1900,2019) != 0);

do

{

printf("A%co de registro: ",164); gets(persona->anio\_reg);

if(verificarRangoValores(persona->anio\_reg,atoi(persona->anio\_nac),2019) != 0) errorMessage();

} while(verificarRangoValores(persona->anio\_reg,atoi(persona->anio\_nac),2019) != 0);

itoa(rand()%(9999+1-1000)+1000, persona->num\_consecutivo,10);

setbuf(stdin, NULL);// Se limpia el buffer de entrada para que no haya alguna intervencion en un futuro dentro del programa.

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DESCRIPCION:

Función la cual genera el número verificador del NSS, el cual se determina con

un algoritmo específico, dado por el Seguro Social.

PARAMETROS

◘ persona: esta es la estructura con la informacion de la persona

la cual se requiere para hacer las operaciones.

◘ nss: este es arreglo de enteros, el cual sera llenado con los datos

que se encuetran en la estructura, pero convertidas a entero.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int generarNumeroVerificador(info\_t persona, int\* nss)

{

int suma = 0;

// guarda en nss los valores de la información capturada como numeros enteros.

nss[0] = atoi(persona.delegacion);

nss[1] = atoi(persona.anio\_reg);

nss[2] = atoi(persona.anio\_nac);

nss[3] = atoi(persona.num\_consecutivo);

suma = sumatoria(nss[0],2) + sumatoria(nss[1]%100,2) + sumatoria(nss[2]%100,2)+ sumatoria(nss[3],4); // suma de los digitos multiplicados por 1 y 2.

if (suma%10 > 0) return (10 - suma%10);

else return 0;

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DESCRIPCION:

◘ Función que multiplica los primeros 10 números compuestos de NSS para

poder determinar el numero verificador, del modo: x 1 2 9 4 6 6 7 8 9 5 .

1 2 1 2 1 2 1 2 1 2

1+4+9+8+6+12+7+16+9+10 = 82

Devuelve la sumatoria de todos los números.

PARAMETROS

◘ dato: número entero que se utiliza para hacer la multiplicación.

◘ count: número de veces que se va hacer la iteración.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int sumatoria(int dato, int count)

{

int sumatoria = 0, aux=0, i; // auxiliares para resolver la suma y multiplicación

for(i = 0; i < count; i++) // La cantidad de iteraciones esta dada dependiendo del tipo de dato, puede ser de 2 o 4.

// Aunque es flexible para cualquier número, pero es nuestro caso son los que se utiliza

{

aux = (dato%10); // La variable aux obtiene el ultimo digito con la operacino %10.

if(i%2 == 0) aux \*= 2; // si el numero de iteracion es par, aux se multiplica por 2, esto para cumplir la regla para determinar el digito verificador.

sumatoria += aux; // la variable sumatoria se le acumula el valor de aux, ya que es la variable que contiene la sumatoria.

dato /= 10; // el dato se divide entre 10 para poder pasar al numero siguiente y volver a realizar las operaciones anteriores.

}

return sumatoria; // Devuelve el valor de la sumatoria

}

int verificarRangoValores(char \*dato, int min, int max)

{

if(atoi(dato) < min || atoi(dato) > max) return -1;

return 0;

}

/\*

DESCRIPCION

Mensaje de errror en caso de que los datos introducidos no sean validos.

PARAMETROS:

◘ La función no contiene parametros.

\*/

void errorMessage(void)

{

printf("Datos ingresados son incorrectos!");

setbuf(stdin, NULL);

getchar();

}

/\*

DESCRIPCION

Función que verifica el formato del NSS. Ejemplo: en la captura de la delegación, si

se captura un 1, el formato correcto debe ser 01.

El trabajo que realiza la función es crear una array auxiliar e inicializarlo con un caracter '0'

y despues concatenar el resto de la cadena, para así darle el formato correcto.

PARAMETROS:

◘ seguro\_social: cadena con el numero de seguro social, pero sin formato.

\*/

void verificaFormato(char\* seguro\_social)

{

char str\_aux[11];

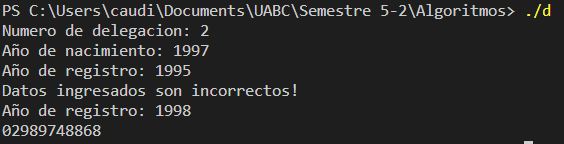
str\_aux[0] = '0';

strcat(str\_aux, seguro\_social);

strcpy(seguro\_social, str\_aux);

}

**Evidencia de ejecución**



Ejecución validando que la fecha de registro no sea menor a la fecha de nacimiento. También se observa que el número de delegación se ha capturado con un solo digito. Y al final se le da formato agregándole 0 al inicio de la cadena.

**Conclusión**

Con la realización de la practica se puede concluir que el problema planteado no es difícil, pero las validaciones de las funciones te hacen pensar en los posibles errores que un usuario pueda cometer a la hora de estar capturando la información requerida en el programa. Pero a la vez es de mucha importancia ya que en la vida real un error de programación y sin validar funciones puede traer problemas muy grandes. Es por eso la importancia que se tiene al validar funciones, al mismo tiempo que debes pensar como resolverlas lo mas optimo posible sin darle tantas vueltas a los datos modificándolos una y otra vez. En esta practica se aplican los conocimientos adquiridos en cursos pasados de programación. Por ser la primera practica del curso, sirve de mucho para reforzar el conocimiento y poner en práctica los aprendido.

**Bibliografía**

Luis Joyanes & Ignacio Zahonero. (2008). *Estructura de datos en Java*. Madrid: McGraw-Hill.

<http://www.cplusplus.com/reference/cstdlib/itoa/>

<http://www.cplusplus.com/reference/cstdlib/atoi/>