***TP N° 7: Programación Avanzada – Uso de Heap***

***Ejercicio 2***

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <math.h>

#include "getnum.h"

#define TRUE 1

#define FALSE 0

*/\* Lee de la entrada estándar con el formato <n>m, donde n y m son números*

*\*\* entre 2 y 10 y devuelve esos valores como parámetros*

*\*/*

void leerBases ( int \*baseOrigen, int \*baseDestino );

*/\* lee un número entero en una base determinada \*/*

int leerNumero( int base);

*/\* Recibe un número entero con dígitos de una base y lo convierte a otra base,*

*\*\* dejando el resultado en otro numero.*

*\*/*

int convierte ( int entrada, int baseOrigen, int baseDestino);

*/\* Recibe un número positivo en cualquier base [2..10] y pasa a base 10 \*/*

int pasaABase10 ( int entrada, int base);

*/\* Recibe un número positivo en base 10 y lo pasa a cualquier base [2..10] \*/*

int pasaDeBase10 ( int entrada, int base);

int

main (void)

{

int base1, base2, numero, salida;

leerBases(&base1, &base2);

while ( numero = leerNumero(base1) )

{

salida = convierte(numero, base1, base2);

printf("\nEl numero convertido es %d\n",salida);

}

return 0;

}

void

leerBases ( int \*baseOrigen, int \*baseDestino )

{

int aux1, aux2, flag;

do

{

printf("\nIngrese bases en formato <n>m: ");

flag = scanf ("<%d>%d",&aux1,&aux2);

BORRA\_BUFFER;

if ( flag==2 && ( aux1<2 || aux1>10 || aux2<2 || aux2>10))

flag = 0;

} while ( flag != 2);

\*baseOrigen = aux1;

\*baseDestino = aux2;

}

int

leerNumero( int base)

{

int flag, aux, respuesta;

do

{

respuesta = getint("Ingrese entero en base %d (0 para terminar): ",

base);

flag = TRUE;

aux = abs(respuesta);

while ( aux > 0 && flag )

{

if ( aux % 10 >= base )

flag = FALSE;

aux /= 10;

}

if ( !flag )

printf("Ingreso incorrecto de datos\n");

} while ( !flag );

return respuesta;

}

int

convierte ( int entrada, int baseOrigen, int baseDestino )

{

int intermedio, signo = 1;

if ( baseOrigen == baseDestino )

return entrada;

if ( entrada < 0 )

{

signo = -1;

entrada \*= signo;

}

*/\* Primero convierte a base 10 y luego a la base deseada \*/*

intermedio = pasaABase10(entrada, baseOrigen);

intermedio = pasaDeBase10(intermedio, baseDestino);

return intermedio \* signo;

*/\* Otra forma,* ***return pasaDeBase10(paseABase10(entrada, baseOrigen), baseDestino);*** *\*/*

}

int

pasaABase10 ( int entrada, int base)

{

long auxiliar=0, anterior=0, potencia=1;

*/\* Convierte el numero de entrada a una variable numérica en base 10*

*\*\* Con "anterior" se valida que el calculo de "auxiliar" no genere overflow.*

*\*/*

while ( entrada > 0 && anterior<=auxiliar)

{

anterior = auxiliar;

auxiliar += potencia \* (entrada % 10);

potencia \*= base;

entrada /= 10;

}

if ( anterior <= auxiliar)

return auxiliar;

else

return 0;

}

int

pasaDeBase10 ( int entrada, int base)

{

int digito, salida = 0;

int potencia = 1;

while( entrada > 0 )

{

digito = entrada % base;

entrada/= base;

salida += digito \* potencia;

potencia \*= 10;

}

return salida;

}

La versión anterior funciona correctamente siempre y cuando se trabaje con números pequeños. La siguiente implementación trabaja con strings y es un poco más compleja pero no tiene las limitaciones presentadas en el código anterior.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define BORRA\_BUFFER while (getchar() != '\n')

*/\* Máxima cantidad de dígitos de un número \*/*

#define MAX\_DIGIT 40

*/\* Lee de la entrada estándar con el formato <n>m, donde n y m son números*

*\*\* entre 2 y 10 y devuelve esos valores como parámetros*

*\*/*

void leerBases ( int \*baseOrigen, int \*baseDestino );

*/\* Lee en un string un número entero en una base determinada*

*\*\* si el número ingresado es 0 devuelve 0, 1 en caso contrario*

*\*/*

int leerNumero(char numero[], int base);

*/\* Recibe un string con dígitos de una base y lo convierte a otra base, dejando el*

*\*\* resultado en otro string. En caso de poder convertir devuelve 1, 0 si no*

*\*/*

int convierte(char entrada[],int baseOrigen,int baseDestino,

char salida[]);

*/\* Recibe un string en cualquier base [2..10] y pasa a base 10 \*/*

int pasaABase10 ( char entrada[], int base, char salida[]);

*/\* Recibe un string en base 10 y lo pasa a cualquier base [2..10] \*/*

int pasaDeBase10 ( char entrada[], int base, char salida[]);

int

main (void)

{

char entrada[MAX\_DIGIT], salida[MAX\_DIGIT];

int base1, base2;

leerBases(&base1, &base2);

while ( leerNumero(entrada, base1) )

{

if (convierte(entrada, base1, base2, salida))

printf("\nEl numero convertido es %s\n",salida);

else

printf("\nNumero fuera del rango representable\n");

}

return 0;

}

void

leerBases ( int \*baseOrigen, int \*baseDestino )

{

int aux1, aux2, cant;

do

{

printf("\nIngrese bases en formato <n>m: ");

cant = scanf ("<%d>%d",&aux1,&aux2);

BORRA\_BUFFER;

if ( cant==2 && ( aux1<2 || aux1>10 || aux2<2 || aux2>10))

cant = 0;

} while ( cant != 2);

\*baseOrigen = aux1;

\*baseDestino = aux2;

}

int

leerNumero( char numero[], int base)

{

int flag, i;

char formato[6];

*/\* Armar el string de formato para leer un máximo de caracteres*

*\*\* Formato tendrá por ejemplo el siguiente contenido: "%4s" o "%6s", etc \*/*

sprintf (formato, "%%%ds",MAX\_DIGIT);

do

{

printf ("Ingrese un entero en base %d (0 para terminar): ",base);

flag = scanf(formato,numero);

BORRA\_BUFFER;

if ( flag )

{

for (i=0; numero[i] && flag; i++)

if ( numero[i] < '0' || numero[i] >= base +'0')

flag = 0;

}

if ( !flag ) printf("Ingreso incorrecto de datos\n");

} while ( flag != 1);

*/\* si el numero es 0, se considera finalizada la entrada \*/*

flag = strcmp(numero,"0") == 0 ? 0: 1;

return flag;

}

int

convierte(char entrada[], int baseOrigen, int baseDestino, char salida[])

{

char intermedio[MAX\_DIGIT];

int flag;

*/\* Primero convierte a base 10 y luego a la base deseada \*/*

if ( flag = pasaABase10(entrada, baseOrigen, intermedio))

flag = pasaDeBase10(intermedio, baseDestino, salida);

return flag;

}

int

pasaABase10 ( char entrada[], int base, char salida[])

{

int i;

long auxiliar=0, anterior=0, potencia=1;

*/\* Convierte el número de entrada a una variable numérica en base 10*

*\*\* Con* ***anterior*** *se valida que el cálculo de* ***auxiliar*** *no genere overflow*

*\*/*

for(i = strlen(entrada)-1; i >= 0 && anterior<=auxiliar; i--)

{

anterior = auxiliar;

auxiliar += potencia \* ( entrada[i] - '0' );

potencia \*= base;

}

if ( anterior <= auxiliar)

{

sprintf( salida, "%ld", auxiliar); */\* paso a string \*/*

return 1;

}

else

return 0;

}

int

pasaDeBase10 (const char entrada[], int base, char salida[])

{

long auxiliar;

int i, j, digito;

char aux;

auxiliar = atol(entrada);

*/\* Guardar los restos en orden inverso al correcto \*/*

for(i=0; auxiliar >= base && i < MAX\_DIGIT; i++)

{

digito = auxiliar % base;

auxiliar /= base;

salida[i] = digito + '0';

}

if ( i == MAX\_DIGIT )

return 0; */\* No alcanza la longitud del string \*/*

salida[i] = auxiliar + '0';

salida[i+1] = 0;

*/\* Invertir el string de resultado \*/*

for (j=0; j <= i/2; j++)

{

aux = salida[j];

salida[j] = salida[i-j];

salida[i-j] = aux;

}

return 1;

}

Otra versión de leerBases, esta vez sin utilizar variables auxiliares para realizar la lectura de las bases.

void

leerBases ( int \*baseOrigen, int \*baseDestino )

{

int flag;

do

{

printf("\nIngrese bases en formato <n>m: ");

flag = scanf ("<%d>%d", baseOrigen, baseDestino);

BORRA\_BUFFER;

if ( flag==2 && ( \*baseOrigen < 2 || \*baseOrigen > 10 ||

\*baseDestino < 2 || \*baseDestino > 10))

flag = 0;

} while ( flag != 2);

}

***Ejercicio 3***

La función ***misterio1*** concatena dos strings (el segundo string recibido al final del primer string) como la función de biblioteca estándar ***strcat***.

***Ejercicio 4***

La función ***misterio2*** retorna la longitud real del string recibido, como la función de biblioteca estándar ***strlen***.

***Ejercicio 5***

En esta primera versión implementamos los vectores de alumnos como matrices de caracteres, donde cada fila contiene el nombre de un alumno. En este caso la cantidad de columnas debe ser como mínimo la longitud del nombre más largo.

*/\* Cantidad máxima de alumnos \*/*

#define MAX\_ALUMNOS 40

*/\* Máxima longitud del nombre de los alumnos \*/*

#define MAX\_LARGO 25

typedef char TAlumnos[MAX\_ALUMNOS][MAX\_LARGO];

typedef char TCurso[];

*/\* Recibe un arreglo con los alumnos y otro con el curso al que pertenecen*

*\*\* y devuelve dos arreglos con los alumnos separados por curso \*/*

void separaCursos ( TAlumnos alumnos, TCurso cursos, TAlumnos cursoG,

TAlumnos cursoH);

void

separaCursos ( TAlumnos alumnos, TCurso cursos, TAlumnos cursoG,

TAlumnos cursoH)

{

int i=0;

int indG=0, indH=0; */\* índices de cursoG y cursoH \*/*



while ( alumnos[i][0] != 0 ) 

{

if ( cursos[i] == 'G' )

strcpy ( cursoG[indG++], alumnos[i]);

else

strcpy ( cursoH[indH++], alumnos[i]);

i++;

}

*/\* Marcar el final del vector con un string vacío \*/*

cursoG[indG][0] = 0;

cursoH[indH][0] = 0;

}

Ejemplo de programa que utiliza la función anterior.

int

main (void)

{

TAlumnos alumnos= {"Carlos G","Susana W","Jorge P","Carla B","Sandra S","Miriam T",""};

TCurso curso = "GGHGHG";

TAlumnos cursog;

TAlumnos cursoh;

.....

.....

separaCursos(alumnos, curso, cursog, cursoh );

.....

.....

return 0;

}

En la siguiente versión los nombres se almacenan en vectores de punteros a char, por lo que su tratamiento es distinto.

#define MAX\_ALUMNOS 40

typedef char \* TAlumnos[MAX\_ALUMNOS];

typedef char TCurso[];

void

separaCursos ( TAlumnos alumnos, TCurso cursos, TAlumnos cursoG, TAlumnos cursoH)

{

int i=0;

int indG=0, indH=0; */\* indices de cursoG y cursoH \*/*

while (alumnos[i][0] != 0 )

{

if ( cursos[i] == 'G' )

cursoG[indG++] = alumnos[i];

else

cursoH[indH++] = alumnos[i];

i++;

}

cursoG[indG]= alumnos[i]; */\* Ambas opciones son correctas ya que \*/*

cursoH[indH]= ""; */\* representan el string nulo \*/*

}

***Para pensar:***

* ¿Funciona con esta versión el main() del ejemplo anterior?
* En caso de no funcionar, ¿qué modificaciones requiere?

***Ejercicio 6***

Para saber qué hace cada función, remitirse al diseño propuesto en la guía de trabajos prácticos.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include "getnum.h"

#include "random.h" */\* Utiliza la biblioteca random desarrollada en clase teórica \*/*

#define PAUSA { while((getchar()) != '\n'); }

#define DIGITOS 5

#define MAXNIVEL 10

void generaAleatorio( int incognita[] );

int elegirNivel( void );

void leerNumero( int numero[] );

int coincideNumero( int incognita[], int numero[]);

int cantidadBien( int incognita[], int numero[]);

int cantidadRegular( int incognita[], int numero[]);

int

main (void)

{

int incognita[DIGITOS], numero[DIGITOS];

int acierto = 0; */\* Flag para indicar que el jugador adivinó \*/*

int paso, i, nivel;

randomize();

nivel = elegirNivel();



generaAleatorio(incognita);

for (paso=0; paso < nivel && !acierto; paso++)

{

printf("Queda%s %d oportunidad%s\n", (nivel-paso>1)?"n":"",

nivel-paso, (nivel-paso>1)?"es":"");

leerNumero(numero);

acierto = coincideNumero(incognita, numero);

}

if ( acierto )

printf ("\nAdivinaste el numero en %d pasos\n",paso);

else

{

printf ("\nLo siento, el numero es ");

for (i=0; i < DIGITOS; i++)

printf ("%d",incognita[i]);

printf("\n");

}

return 0;

}

void

generaAleatorio( int incognita[] )

{

*/\* Armamos un arreglo auxiliar con los números del 1 al 9*

*\*\* Cada número que sale se lo reemplaza por el último elemento y se "achica"*

*\*\* el arreglo. De esta forma aseguramos que en cada oportunidad el número*

*\*\* elegido no salió anteriormente, haciendo eficiente el algoritmo.*

*\*/*

char arregloAux[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int n = 9, i, numAux;

for (i=0; i < DIGITOS; i++)

{

numAux = randInt(0,n-1);

incognita[i] = arregloAux[numAux];

arregloAux[numAux] = arregloAux[--n];

}

return;

}

int

elegirNivel( void )

{

int n;

do

n = getint("\nIngrese cantidad de chances: ");

while ( n < 1 || n > MAXNIVEL );

return n;

}

void

leerNumero( int numero[] )

{

int i, j, num, numOK;

do

{

num= getint("Ingrese un numero de %d dígitos del 1 al 9 sin repetir:",

DIGITOS);

for (i=DIGITOS-1, numOK = 1; i>=0 && numOK ; i--, num/=10)

{

numero[i] = num % 10;

if (numero[i] == 0)

numOK = 0;

else

for (j=DIGITOS-1; j>i && numOK ; j--)

if (numero[i] == numero[j])

numOK = 0;

}

}while (!numOK || num!=0);

/\*Si num!=0 entonces el número tenía dígitos de más \*/

return;

}

int

coincideNumero(const int incognita[], int numero[])

{

int bien, regular;

bien = cantidadBien(incognita, numero);



if (bien == DIGITOS)

return 1;

regular = cantidadRegular(incognita, numero);

printf ("Bien: %d\n", bien);

printf ("Regular: %d\n", regular);

return 0;

}

int

cantidadBien(const int incognita[], int numero[])

{

int i;

int bien = 0;

for (i=0; i < DIGITOS; i++)

bien += ( incognita[i]==numero[i]);

return bien;

}

int

cantidadRegular(const int incognita[], int numero[])

{

int i,j;

int regular = 0;



for (i=0; i < DIGITOS; i++)

for (j=0; j < DIGITOS; j++)

if ( i != j )

regular += ( incognita[i] == numero[j]);

return regular;

}

***Para pensar:***

¿Qué se puede criticar del diseño del ejercicio en relación con la eficiencia ?

***Ejercicio 7***

Para una explicación de lo que hace cada función, consultar el diseño presentado en la guía de TP.

A diferencia del ejercicio anterior, se debe agregar a cada función que opera sobre los números un parámetro indicando la cantidad de dígitos que contiene. Más allá de eso, los algoritmos siguen siendo los mismos.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include "getnum.h"

#include "random.h"

#define MAXNIVEL 10

void generaAleatorio( int incognita[], int digitos);

int elegirNivel( void );

void leerNumero( int numero[], int digitos);

int coincideNumero( int incognita[], int numero[], int digitos);

int cantidadBien( int incognita[], int numero[], int digitos);

int cantidadRegular( int incognita[], int numero[], int digitos);

int

main (void)

{

int incognita[9], numero[9];

int digitos, nivel, i;

int acierto = 0; */\* Flag para indicar que el jugador adivino \*/*

randomize();

nivel = elegirNivel();

putchar ('\n');

*/\* Pedir al usuario la cantidad de dígitos (entre 1 y 9) \*/*

do

{

digitos = getint("Cantidad de digitos: ");

} while ( digitos <=0 || digitos >9 );

generaAleatorio(incognita,digitos);

for (i=1; i <= nivel && !acierto; i++)

{

leerNumero(numero, digitos);

acierto = coincideNumero(incognita, numero, digitos);

}

if ( acierto )

printf ("\nAdivinaste el numero en %d pasos\n",i-1);

else

{

printf ("\nLo siento, el numero es ");

for (i=0; i < digitos; i++)

printf ("%d",incognita[i]);

putchar('\n');

}

return 0;

}

void

generaAleatorio(int incognita[], int digitos )

{

*/\* Armamos un arreglo auxiliar con los números del 1 al 9*

*\*\* Cada número que sale se lo reemplaza por el último elemento y se "achica"*

*\*\* el arreglo. De esta forma aseguramos que en cada oportunidad el número*

*\*\* elegido no salió anteriormente, haciendo eficiente el algoritmo*

*\*/*

char arregloAux[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int n = 9, i, numAux;

for (i=0; i < digitos; i++)

{

numAux = randInt(0,n-1) ;

incognita[i] = arregloAux[numAux];

arregloAux[numAux] = arregloAux[--n];

}

}

int

elegirNivel( void )

{

int n;

do

n = getint("Cantidad de opciones: ");

while ( n < 1 || n > MAXNIVEL );

return n;

}

void

leerNumero( int numero[] )

{

int i, j, num, numOK;

do

{

num= getint("Ingrese un numero de %d dígitos del 1 al 9 sin repetir:",

DIGITOS);

for (i=DIGITOS-1, numOK = 1; i>=0 **&& numOK** ; i--, num/=10)

{

numero[i] = num % 10;

if (numero[i] == 0)

numOK = 0;

else

for (j=DIGITOS-1; j>i **&& numOK** ; j--)

if (numero[i] == numero[j])

numOK = 0;

}

}while (!numOK || num!=0);

*/\*Si num!=0 entonces el número tenía dígitos de más \*/*

return;

}

int

coincideNumero( const int incognita[], const int numero[], int digitos)

{

int bien, regular;

bien = cantidadBien(incognita, numero, digitos);

if (bien == digitos)

return 1;

regular = cantidadRegular(incognita, numero, digitos);

printf ("Bien: %d\n", bien);

printf ("Regular: %d\n", regular);

return 0;

}

int

cantidadBien(const int incognita[], const int numero[], int digitos)

{

int i;

int bien = 0;

for (i=0; i < digitos; i++)

bien += ( incognita[i]==numero[i]);

return bien;

}

int

cantidadRegular( const int incognita[], const int numero[], int digitos)

{

int i,j;

int regular = 0;

for (i=0; i < digitos; i++)

for (j=0; j < digitos; j++)

if ( i != j )

regular += ( incognita[i] == numero[j]);

return regular;

}

***Ejercicio 8***

Para una explicación de lo que hace cada función, consultar el diseño presentado en la guía de TP.

*/\* Simulador de Bingo para dos jugadores \*/*

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include "random.h"

*/\* La siguiente macro lee caracteres hasta que recibe un ENTER \*/*

#define PAUSA { while((getchar()) != '\n'); }

#define BOLILLAS 90

#define LONG\_LINEA 5

#define LINEAS 3

typedef int TipoLinea[LONG\_LINEA];

typedef TipoLinea TipoCarton[LINEAS];

void generarCarton ( TipoCarton carton);

int sacarBolilla (int bolillero[], int \*cantBolillas);

int controlarCarton (TipoCarton carton, int bolilla);

void imprimirCarton (TipoCarton carton);

int buscarBolilla (TipoCarton carton, int bolilla);

int controlarLineas (TipoLinea linea);

int jugar(int bolillero[], TipoCarton carton1, TipoCarton jugador2);

int

main (void)

{

TipoCarton carton1, carton2;

int bolillero[BOLILLAS];

int i;

int bingo;

randomize();



for (i=0; i < BOLILLAS; i++)

bolillero[i] = i+1;

generarCarton (carton1);

generarCarton (carton2);

bingo = jugar(bolillero, carton1, carton2);

for (i=1; i <= 2; i++)

if (bingo & i)

printf ("El jugador %d hizo bingo\n",i);

return 0;

}

int

jugar(int bolillero[],TipoCarton carton1, TipoCarton carton2)

{

int huboLinea=0;

int bolilla;

int cantBolillas = BOLILLAS;

int lineas1, lineas2;

int bingo=0;

do

{

printf("\nJugador 1\n");

imprimirCarton(carton1);

printf("\nJugador 2\n");

imprimirCarton(carton2);

*/\* Extraer una bolilla \*/*

bolilla = sacarBolilla( bolillero, &cantBolillas);

printf("Bolilla extraida: %d\n",bolilla);

*/\* Controlar si alguno o los dos hizo linea \*/*

lineas1 = controlarCarton( carton1, bolilla);

lineas2 = controlarCarton( carton2, bolilla);

if (! huboLinea )

{

if ( lineas1 > 0 )

printf ("El jugador 1 hizo linea\n");

if ( lineas2 > 0 )

printf ("El jugador 2 hizo linea\n");

huboLinea += lineas1 + lineas2 ;

}

***/\* Encendemos un bit en la variable bingo por cada jugador***

***\*\* De esta forma podemos controlar si ambos hicieron bingo***

***\*\* con una sola variable***

***\*/***

bingo = (lineas1 == 3) | 2 \* (lineas2 == 3);

PAUSA

} while (bingo == 0);

return bingo;

}

void

generarCarton( TipoCarton carton )

{

int i,j;

int n;

int aux = BOLILLAS-1;

int arregloAux[BOLILLAS];

*/\* Usamos un vector auxiliar para no repetir numeros.*

*\*\* En cada paso lo que se genera en forma aleatoria es el INDICE del vector*

*\*\* y no el valor de la bolila. Cada vez que se genera el indice, tomamos*

*\*\* el valor almacenado en esa posicion, se copia el ultimo elemento a esa*

*\*\* posicion y se decrementa en uno la cantidad de elementos.*

*\*\* De esta forma es imposible que los numeros se repitan y nos aseguramos*

*\*\* que se ejecute en forma eficiente.*

*\*/*

for (i=0; i<BOLILLAS; i++)

arregloAux[i]=i+1;

for (i=0; i < LINEAS; i++)

for (j=0; j < LONG\_LINEA; j++)

{

n = randInt(0,aux);

carton[i][j] = arregloAux[n];

arregloAux[n] = arregloAux[aux--];

}

}

int

sacarBolilla (int bolillero[], int \*cantBolillas)

{

int aux;

int bolilla;

aux = randInt(1, \*cantBolillas ) ;

(\*cantBolillas)--;

bolilla = bolillero[aux];

bolillero[aux] = bolillero[\*cantBolillas];

return bolilla;

}

int

controlarCarton (TipoCarton carton, int bolilla)

{

int lineasCompletas=0;

if ( buscarBolilla( carton, bolilla ))

{

int i;

for (i=0; i < LINEAS; i++)

lineasCompletas += controlarLineas(carton[i]);

}

return lineasCompletas;

}

void

imprimirCarton (TipoCarton carton)

{

int i,j;

putchar ('\n');

for (i=0; i < LINEAS; i++)

{

for (j=0; j < LONG\_LINEA; j++)

if (carton[i][j])

printf ("%d\t",carton[i][j]);

else

printf ("%c\t",'X');

printf("\n");

}

}

int

buscarBolilla (TipoCarton carton, int bolilla)

{

int i,j;

int esta=0;

for (i=0; i < LINEAS && !esta; i++)

for (j=0; j < LONG\_LINEA && !esta; j++)

if ( carton[i][j] == bolilla ) {

esta = 1;

carton[i][j] = 0;

}

return esta;

}

int

controlarLineas (TipoLinea linea)

{

int i;

int aciertos;

for (i=0, aciertos=0; i < LONG\_LINEA; i++)

aciertos += ( linea[i] == 0 );

return (aciertos == LONG\_LINEA);

}

***Ejercicio 9***

En esta versión se debió cambiar la interfase de la función jugar para que en vez de dos cartones reciba un vector de cartones. Obviamente esta versión sirve también para dos jugadores. Se ve aquí que hubiese sido mejor definir desde un primer momento una interfase más flexible y extensible, previendo que los jugadores fueran más de dos.

Presentamos solo las funciones que cambian con respecto a la implementación anterior.

a) Solución usando N como constante simbólica

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include "random.h"

#include "getnum.h"

#define PAUSA { while((getchar()) != '\n'); }

#define BOLILLAS 90

#define LONG\_LINEA 5

#define LINEAS 3

#define MAX\_JUGADORES 10

typedef int TipoLinea[LONG\_LINEA];

typedef TipoLinea TipoCarton[LINEAS];

void generarCarton ( TipoCarton carton);

int sacarBolilla (int bolillero[], int \*cantBolillas);

int controlarCarton (TipoCarton carton, int bolilla);

void imprimirCarton (TipoCarton carton);

int buscarBolilla (TipoCarton carton, int bolilla);

int controlarLineas (TipoLinea linea);

int jugar(int bolillero[], TipoCarton cartones[], int n);

int

main (void)

{

TipoCarton cartones[MAX\_JUGADORES];

int bolillero[BOLILLAS];

int i, jugadores;

int bingo;

/\* setear la semilla \*/

randomize();

for (i=0; i < BOLILLAS; i++)

bolillero[i] = i+1;

*/\* Preguntar la cantidad de jugadores \*/*

do

{

jugadores = getint("Cantidad de jugadores (entre 1 y %d):",

MAX\_JUGADORES);

} while ( jugadores < 1 || jugadores > MAX\_JUGADORES );

*/\* generar los cartones \*/*

for (i=0; i < jugadores; i++)

generarCarton (cartones[i]);

bingo = jugar(bolillero, cartones, jugadores);

for (i=0; i < jugadores; i++)

if (bingo & (1 << i))

printf ("El jugador %d hizo bingo\n",i+1);

return 0;

}

int

jugar(int bolillero[], TipoCarton cartones[], int n)

{

int huboLinea=0, bolilla, cantBolillas = BOLILLAS;

int lineas, resultado, i, bingo=0;

do

{

*/\* Imprimir los cartones de cada jugador \*/*

for (i=0; i < n; i++)

{

printf("\nJugador %2d\n", i+1);

imprimirCarton(cartones[i]);

}

*/\* Extraer una bolilla \*/*

bolilla = sacarBolilla( bolillero, &cantBolillas);

printf("Bolilla extraida: %d\n",bolilla);

*/\* Controlar si alguno hizo linea \*/*

*/\* Otra opcion seria tener un vector auxiliar en donde*

*\*\* v[I]= 1 o 0 si el jugador I hizo linea o no*

*\*/*

for (lineas=0, i=0; i < n; i++)

{

resultado = controlarCarton( cartones[i], bolilla) ;

lineas |= resultado << i;

*/\* Encendemos un bit en la variable bingo por cada jugador \*/*

if ( resultado == 3 )

bingo |= 1 << i;

}

if (! huboLinea )

for ( i=0; i < n; i++)

if ( lineas & (1 << i))

{

printf ("El jugador %2d hizo linea\n", i+1);

huboLinea = 1 ;

}

PAUSA

} while (bingo == 0);

return bingo;

}

b) Solución usando N ingresado por el usuario – Solo se modifica *main* (en negrita)

int

main (void)

{

**TipoCarton \* cartones;**

int bolillero[BOLILLAS];

int i, jugadores;

int bingo;

/\* setear la semilla \*/

randomize();

for (i=0; i < BOLILLAS; i++)

bolillero[i] = i+1;

*/\* Preguntar la cantidad de jugadores \*/*

**do**

**{**

**jugadores = getint("Cantidad de jugadores :");**

**} while ( jugadores < 1 );**

**/\* Crea el arreglo dinámico de cartones \*/**

**if ((cartones = malloc(sizeof(TipoCarton)\*jugadores))) == NULL)**

**{**

**printf(“Hubo un error al reservar memoria.\n”);**

**return 1;**

**}**

*/\* generar los cartones \*/*

for (i=0; i < jugadores; i++)

generarCarton (cartones[i]);

bingo = jugar(bolillero, cartones, jugadores);

for (i=0; i < jugadores; i++)

if (bingo & (1 << i))

printf ("El jugador %d hizo bingo\n",i+1);

free(cartones);

return 0;

}

***Ejercicio 10***

En esta versión del ahorcado no se guarda registro de las letras ya dichas por el jugador. En caso de que repita una letra que no pertenece a la palabra se vuelve a computar el error.

***Ejercicio 11***

1. La función ***aMayusculas*** espera un puntero a char y en su lugar recibe un puntero a int (la dirección del segundo elemento de v). El segundo valor del arreglo es 2, que ocupa 4 bytes y sus valores son 02-00-00-00 (en little-endian), por lo tanto en el ciclo se aplica toupper al valor 02 (que no lo altera por no ser un ASCII de letra minúscula). Al tratar al parámetro como puntero a char, cada incremento aumenta el puntero en un byte entonces el próximo control del ciclo da falso y la función termina retornando la dirección del valor 02.

La primera invocación a ***printf*** toma dicha dirección como string y como el valor 02 no es un ASCII imprimible y el siguiente es 00, imprime un carácter “basura” y termina.

Con la segunda invocación ocurre lo mismo pero con el primer elemento de v. El valor 01 no es un ASCII imprimible y el siguiente es 00.

1. El puntero ***p*** apunta a una zona de datos constante entonces la función ***aMayusculas*** aborta al intentar alterarla.
2. Imprime :

**LA MUNDO**

**HOLA MUNDO**

Esto se debe a que la función ***aMayusculas*** recibe ***p*** por lo tanto altera todo el cartel, pero ***printf*** (la primera invocación) recibe ***p+2***. La segunda invocación de ***printf*** recibe ***p*** entonces aparece el cartel completo.

d) El programa aborta por desreferenciar p que vale NULL. Una función recibe el valor derecho de una variable (una copia de su valor), por lo que luego de las siguientes líneas

char \*p=NULL;

resize(p, 100);

es imposible que p valga otra cosa que NULL. Lo correcto sería que resize retorne el nuevo puntero, o sea

| void \* resize(void \*p, unsigned int bytes) {  return realloc(p, bytes);  }  int  main(void) {  char \*p=NULL;  p = resize(p, 100);  strcpy(p, "hello world");  aMayusculas(p);  printf("%s\n", p);  return 0;  } |
| --- |

o que reciba el valor izquierdo del puntero:

| void resize(void \*\*p, unsigned int bytes) {  \*p = realloc(\*p, bytes);  }  int  main(void) {  char \*p=NULL;  resize(&p, 100);  strcpy(p, "hello world");  aMayusculas(p);  printf("%s\n", p);  return 0;  } |
| --- |

***Ejercicio 12***

a) Ocupa 24 o 48 bytes. Esto es porque ***p*** es una matriz de punteros a char. Cada puntero ocupa 4 u 8 bytes y la matriz tiene 6 elementos (2 filas x 3 columnas).

b)

| **Expresión** | **Expresión Equivalente** | **Valor** |
| --- | --- | --- |
| \*\*\*p | p[0][0][0] | ‘a’ |
| \*\*p[1] | p[1][0][0] | ‘j’ |
| \*\*(p[1]+2) | p[1][2][0] | ‘x’ |
| \*(\*(p+1)+1)[7] | p[1][8][0] | /\* error \*/ |
| (\*(\*(p+1)+1))[7] | p[1][1][7] | ‘w’ |
| \*(p[1][2]+2) | p[1][2][2] | ‘z’ |

### *Ejercicio 13*

1. El puntero **numero** no está inicializado ni permite reservar espacio para almacenar un entero.
2. No se pueden asignar punteros de distinto tipo y además **realPtr** no fue inicializado. Aunque fuera inicializado, la solución no es castearlo, ya que de todas formas la información almacenada seguiría siendo de punto flotante y al recuperarla se obtendría basura.
3. La variable **y** no es un puntero. A **x** se le debe asignar la **dirección** **de** **y,** de la siguiente forma **x = &y** .
4. No se puede incrementar el puntero ya que es el nombre del arreglo. La forma correcta de hacerlo es con subíndices:

char s[] = "esta es una cadena de caracteres";

int i;

for (i=0 ; s[i] != '\0'; i++)

printf("%c ", s[i]);

o con un puntero extra:

char s[] = "esta es una cadena de caracteres";

char \*aux;

for (aux = s ; \*aux != '\0'; aux++)

printf("%c ", \*aux);

1. Es necesario castear al puntero ***genericoPtr*** antes de desreferenciarlo, ya que es un puntero a void.

result = \* **(short\*)**genericoPtr + 7;

f) La variable **s** no está inicializada

g) Si realloc tuvo que mover el bloque de memoria, entonces s ya no apunta a una zona reservada en el heap,. por lo que la proposición free(s) provocará un error de segmentación. Si realloc no tuvo que desplazar la zona de memoria (había 80 bytes libres a continuación de los 20 bytes), entonces el error se producirá cuando se ejecute free(aux), ya que s y aux valen lo mismo, por lo que al intentar liberar por segunda vez la misma zona se producirá un error.

##### Ejercicio 14

1. La dirección del primer elemento de x, es decir, dirección de **x**[0]
2. Como x es la dirección de un elemento (el primero) si le sumo un valor debo obtener la dirección de otro elemento, ya que el tipo de dato se mantiene, en este caso, al sumar dos el puntero se mueve en dos elementos obteniendo la dirección de **x**[2], es decir, el tercer elemento.
3. Si x es la dirección del primer elemento, al desreferenciarlo obtengo el elemento, es decir el valor 10.
4. Sumo dos al valor obtenido en el ítem anterior, o sea 12.
5. Corresponde a desreferenciar la dirección obtenida en el ítem b), es el valor 30.
6. x[4] estaría almacenado en $1100 (considerando como tamaño de un int 4 bytes).

### *Ejercicio 15*

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 1.1 | | | | 1.2 | | | | 1.3 | | | | 2.1 | | | |
| 101 | 2.2 | | | | 2.3 | | | |  |  |  |  |  |  |  |  |
| 102 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

La variable ***tabla*** es un arreglo de 2 componentes, donde cada una es un arreglo de 3 floats.

1. Dirección de la primera fila o componente de **tabla** : El valor de una variable de tipo arreglo es la dirección de su primer componente y tabla está compuesta de **filas**, entonces es la dirección de la primera **fila**. El tipo de dato es **float \*\*** y su valor es **1000**.
2. Dirección de la segunda fila de **tabla** : Si tabla es la dirección de la primer fila, al sumarle un valor se obtiene la dirección de otra fila. En este caso es la fila siguiente. El tipo de dato es **float \*\*** y su valor es **100C**.
3. **&tabla**[1][0], o lo que es lo mismo la dirección del primer elemento de la segunda fila: Si tabla + 1 es la dirección de la primera fila, al desreferenciarla obtengo dicha fila. A su vez cada fila es un arreglo, entonces como el nombre de un arreglo es la dirección de su primer componente, obtengo la dirección del primer elemento de dicho arreglo. El tipo de dato es **float \*** y su valor es **100C**. Es importante notar que existe coincidencia en valor con el ítem b) , NO así en cuanto al tipo de dato.
4. **&tabla**[1][1]: Si **\*(tabla+1)** es la dirección de una componente de una fila, al sumarle cualquier valor tengo que obtener la dirección de otra componente de fila. En este caso es la dirección de la segunda componente de la segunda fila. El tipo de dato es **float \*** y su valor es **1010**.
5. **&tabla**[0][1]: El tipo de dato es **float \*** y su valor es **1004**.
6. **tabla**[1][1]. Es desreferenciar la dirección del punto d). El tipo de dato es **float** y su valor es **2.2**.
7. **tabla**[0][1]. Idem con el punto e). El tipo de dato es **float** y su valor es **1.2**.
8. **tabla**[1][0]. Idem con el punto c) . El tipo de dato es **float** y su valor es **2.1**.
9. **tabla**[0][1] + . Se le suma uno al valor obtenido en el punto g). El tipo de dato es **float** y su valor es **2.2**.
10. Es la segunda fila de **tabla** (o lo que es lo mismo la dirección del primer elemento de la segunda fila de **tabla**)**.** Es equivalente al punto c). El tipo de dato es **float \*** y su valor es **100C**.
11. **&tabla**[0][1]. Es equivalente al punto e). El tipo de dato es **float \*** y su valor **1004**.
12. **tabla**[1][0] + 1: Es equivalente al punto h), sumándole uno. El tipo de dato es **float** y su valor es **3.1**.
13. **tabla**[1][0]: Es equivalente al punto h). El tipo de dato es **float** y su valor es **2.1**.
14. **tabla**[0][1]: Es equivalente al punto g). El tipo de dato es **float** y su valor es **1.2**.

### *Ejercicio 16*

a)

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 |  | ‘r’ | ‘o’ | ‘j’ | ‘o’ | 0 | ‘v’ | ‘e’ | ‘r’ | ‘d’ | ‘e’ | 0 | ‘a’ | ‘z’ | ‘u’ | ‘l’ |
| 101 | 0 | ‘b’ | ‘l’ | ‘a’ | ‘n’ | ‘c’ | ‘o’ | 0 | ‘n’ | ‘e’ | ‘g’ | ‘r’ | ‘o’ | 0 | ‘a’ | ‘m’ |
| 102 | ‘a’ | ‘r’ | ‘i’ | ‘l’ | ‘l’ | ‘o’ | 0 |  |  |  |  |  |  |  |  |  |

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 200 |  |  |  |  | 1001 | | | | 1006 | | | | 100C | | | |
| 201 | 1011 | | | | 1018 | | | | 101E | | | |  |  |  |  |
| 202 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. Dirección del primer elemento de color, es decir, dirección de un puntero al cartel *“rojo”* o bien **&color[0]**. El tipo de dato es **char \*\*,** el valor es **2004**.
2. Dirección de un puntero al tercer cartel o bien **&color[2]**. El tipo de dato es **char \*\*,** el valor es **200C**.
3. Puntero al primer cartel, o lo que es lo mismo, la dirección de la primera componente del primer cartel. Es equivalente a decir **color[0]**. El tipo de dato es **char \***, el valor es **1001**.
4. Puntero al tercer cartel, o bien, dirección de la primera componente del tercer cartel. Es equivalente a decir **color[2]**. El tipo de dato es **char \***, el valor es **100C**.
5. Son equivalentes. Ambos son la dirección donde comienza la cadena “amarillo” (la dirección de la primera ‘a’). El tipo de dato es **char \*** y su valor es **101E**.
6. Es **color[0][0]** es decir la letra ‘r’. **color** es la dirección del primer cartel, **\*color** es la dirección de la primera componente del primer cartel y **\*\*color** es el valor de la primera componente del primer cartel. El tipo de dato de **\*\*color** es **char**, su valor numérico corresponde al código ASCII de la **‘r’**.
7. Equivalente al punto anterior , pues **color[0]** es la primera componente de **color** (es un puntero a char), y al desreferenciarla se obtiene la primera letra apuntada, la **‘r’**.
8. Como se explica en el punto g) \*\*color es el caracter 'r', por lo que el resultado de esta expresión es el caracter 't'. El tipo de dato es **char**.

***Ejercicio 18***

a) La copia se realizó sobre una variable automática que al salir de la función se destruye. Para hacer una copia es necesario reservar memoria en el heap.



b) Hay cuatro errores, el primero es la definición incorrecta de resp. El segundo corresponde a la cantidad de memoria reservada que se expresa en bytes, por lo tanto debe ser la cantidad de elementos multiplicada por el tamaño de los mismos. Tercero, la función no retorna un tipo ***int*** sino ***int \* , cuarto*** está mal el índicey por último (pero no es obligatorio) se agrega la validación del retorno de ***malloc***.



##### Ejercicio 19

Esta versión no es muy eficiente porque recorre el string para saber su longitud. ¿Qué ventaja tiene?, que en el caso de ser n muy grande (mucho mayor que la longitud del string) tal vez no haya un bloque de n bytes libre en el heap, pero sí haya un bloque para el string resultado.

La siguiente versión privilegia la eficiencia, reservando de entrada el máximo posible

**char \***

**copiaPrefijo(const char \* s, unsigned int n)**

**{**

**char \* aux;**

**if ((aux = malloc(n+1)) == NULL)**

**return NULL;**

**unsigned int i;**

**for(i = 0; s[i] != '\0' && i<n; i++)**

**aux[i] = s[i];**

**aux[i] = 0;**

**aux = realloc(aux, i+1);**

**return aux;**

**}**

Una tercera variante, usando realloc de a bloques (solución aportada por Martín Augusto Ippolito)

| char \* copiaPrefijo(const char \* s, unsigned int n){  char \* newString = NULL;  int i;  for(i = 0; i < n && s[i] != '\0'; i++){  if( !(i % BLOCK) ) {  char \* aux = realloc(newString, i + BLOCK);  if (aux == NULL) {  free(newString);  return NULL;  }  newString = aux;  }  newString[i] = s[i];  }  newString = realloc(newString, i + 1);  newString[i] = '\0';  return newString;  } |
| --- |

El usuario de esta función debe liberar la copia del string cuando deje de usarlo.

***Ejercicio 20***

En el enunciado de este ejercicio se pide que el código de la función sea independiente de la definición del vector de alumnos. La única forma de respetar esto es utilizando un typedef para definir el vector que se adapte a cada una de los tipos propuestos.

***Ejercicio 21***

La función solicitada se incluye en el ejercicio 20.