Ejercicios iniciales de Punteros

*0) Para analizar el siguiente programa:

- Escriba el modelo de administración de memoria de un programa en c++ (sin agregar las variables en primer lugar)
- Sobre el modelo escrito haga el trabajo de seguir en tiempo de ejecución paso a paso el programa (es decir que tiene que seguir la ejecución línea por línea) y cree las variables en el modelo de memoria mostrando:
    - Donde se crean las variables (stack, heap o segmento de código)
    - En qué momento se crean las variables
    - Qué valores van teniendo las variables en cada paso
    - Cuando se destruyen las variables

```
int Sumar(int primerOperador, int segundoOperador);

int main()
{
  int primerOperador;
  int segundoOperador;
  int resultadoSuma;
  primerOperador = 400;
  segundoOperador = 6000;
  resultadoSuma = Sumar(primerOperador, segundoOperador);
  return 0;
}

int Sumar(int primerOperador, int segundoOperador)
{
  int resultadoSuma = primerOperador + segundoOperador;
  return resultadoSuma;
}
```

*1) Definición de variable puntero: Diga con sus propias palabras y acompañada con varios ejemplos una definición de un puntero. Finalmente trate de dar una definición formal de la definición de una variable puntero.

*2) Dadas las siguientes declaraciones:

```
int x;
int array[MAX];
x = array[4];
```

Analice y responda:

• Qué traduce el compilador para acceder al cuarto elemento del arreglo y asignárselo a x.

• Escriba otra forma de hacer lo mismo que utilice punteros (sin utilizar los corchetes [])

*3) Dadas las siguientes declaraciones:

```
int *ip;
```

```
int *ip;
int **ipp;
int (*ip4)[4];
int i;
int j;
int ventas[3][4];
```

explicar con sus palabras las siguientes expresiones:

a) `ip4 = ventas;`
b) `ip = (int *)ventas;`
c) `ipp = (int **) ventas;`
d) `*(*(ip4 + i) + j)`
e) `*(*(ventas + i ) + j)`

Vuelva al ejercicio y trate de dar una explicación más formal y general.

*4) Defina que es un array (con sus propias palabras. Luego trate de dar una definición más formal). ¿Qué diferencia hay entre el nombre de un array (el identificador de la variable array) y un puntero?

*5) ¿Cómo sabe el compilador el tamaño de un objeto al que apunta un puntero?

*6) Escribir un programa que imprima cada uno de los elementos de un arreglo dos dimensiones (una matriz) utilizando un puntero para acceder a los mismos, en lugar de utilizar subíndices. Utilizar el siguiente arreglo y los punteros indicados abajo:

```
int matriz[3][4] = { {1,2,3,4 }, { 5,6,7,8}, {9,10,11,12 } } ;
int *puntero1;
int (*puntero2)[4];
int fila, col;
```

*7) Explicar qué pasa si se olvida el carácter nulo (o barra cero) como último carácter de una cadena de caracteres.

*8) Escribir un programa en el que se defina un arreglo de 10 punteros a float, se lean diez números en las ubicaciones en las que hacen referencia los punteros. Se sumen todos los números y se almacene el resultado en una dirección a la que haga referencia un puntero. El programa deberá mostrar el contenido de todas las variables, tanto los punteros como los números de tipo float.

*9) Explicar qué es una variable tal como la que se define en la siguiente declaración y de un ejemplo de valor posible para la misma:

```
int * * * miVariable;
```

y trate de representarla gráficamente.

*10) Explicar el significado de las siguientes declaraciones:

```
int (*uno)[12];
int *dos[12];
```

*11) Escriba las declaraciones para las siguientes variables:

- un puntero a carácter
- un array de 10 enteros
- una referencia a un array de 10 enteros
- un puntero a un array de cadenas de caracteres
- un puntero a un puntero a carácter
- una constante entera
- un puntero a una constante entera y un puntero constante a un entero.

Inicie cada uno de ellos con un valor.

*12) Cual es el tamaño del array str del siguiente ejemplo:

```
char str[] = "Una cadena corta";
```

*13) Defina una tabla con los nombres de los meses del año y el número de días de cada mes. Escriba dicha tabla papel. Defina las estructuras de datos en c++ dos veces:

- utilizando un array de char para los nombres y un array para los números de días
- utilizando un array de estructuras, en el que cada estructura contiene el nombre del mes y el número de días.

*14) Defina un array de cadena**s** de caracteres (es de dos dimensiones) donde cada cadena de caracteres es un mes del año. Escriba dos funciones:

- Una función que lo carge
- Una función que lo imprima

*15) ¿Qué error conceptual grave comete la siguiente implementación?

```
const int Tope = 100;
char* ObtenerCopia(char * cadena)
{
   char copia[Tope];
   int i=0;
   for(i=0; cadena[i]!='\0'; ++i)
   {
     copia[i]=cadena[i];
   }
   copia[i] = '\0';
   return &(copia[0]);
}
int main(){
  char *copia = ObtenerCopia("Hola que tal");
  for(int i=0; copia[i]!='\0'; i++){
   cout<<"caracter numero "<<i<<"   : "<<copia[i]<<endl;
  }
}
```

*16) Implementar las iteraciones de los punteros de las siguientes funciones de dos maneras distintas

```
int Sumatoria(int* enteros, int cantidadEnteros);
int Copiar(char *cadenaOrigen, char *cadenaDestinoCopia);
```

*17) Dado el siguiente código

```
int main(){
  int operador1 = 100;
  int operador2 = 200;
  int *punteroOperador1 = &operador1 ;
```

```
  int *punteroOperador2 = &operador2;

  cout<<"Operador 1: "<<operador1<<endl;
  cout<<"Operador 2: "<<operador2<<endl;

  cout<<"Valor apuntador por punteroOperador1: "<<*punteroOperador1<<endl;
  cout<<"Valor apuntador por punteroOperador1: "<<*punteroOperador2<<endl;
}
```

- Declarar e implementar una función que intercambie los **valores** de las variables operador1 y operador 2. ¿De cuántas formas diferentes puede invocar esa función y que se obtenga el intercambio deseado?
- Declarar e implementar una función que intercambie los valores de las variables punteroOperador1 y punteroOperador2
- Si tuviera el siguiente vector de enteros: int vector[3] = {40,1,10} ¿qué función podría invocar y como la invocaría para intercambiar el valor de la posición 0 por el de la 10? ¿Que función no podría invocar y por qué razón?

*18) Vuelva a implementar todas las funciones de cadena de caracteres de la guia 1 (void Copiar(char origen [], char destino[]); pero:

- Utilizando aritmética de punteros
- Haciendo las implementaciones lo más compactas que el lenguaje lo permita.

*19) Utilizando el tipo de dato BarajaCartas trabajado en la guía anterior. Se pide implementar las funciones del siguiente programa:

```
int main(int argc, char *argv[])
{
   BarajaCartas baraja;
   CargarBaraja(&baraja);
   MostrarBaraja(&baraja);
   MezclarBaraja(&baraja);
   MostrarBaraja(&baraja);

   return 0;
}
```

*20) Implementar las siguientes funciones de cadena de caracteres, pero hacer las implementaciones mas compactas del universo utilizando los conceptos de aritmetica de punteros

```
void Copiar(char* origen, char*destino);
int ContarCaracteres(char * cadena);
int ConcatenarEnTexto(char * texto, char *subcadena);
```

Ejercicios con memoria dinámica

Funciones creacionales

Las funciones creacionales (o tambien conocidas en inglés como factory methods) tienen la responsabilidad de crear una instancia válida de una entidad, entendiendo que entidad puede ser una estructura, un TDA, un Módulo o un objeto o cualquier encapsulamiento lógico que representa una abstracción. Por instancia válida entendemos que la variable que representa esa entidad está lista para ser usada y está construida de una manera que conceptualmente es correcta.
Las funciones creacionales son una buena práctica de programación para encapsular las responsabilidades de creaciones de entidades, puesto que crear una entidad es una responsabilidad y las responsabilidades se encapsulan en funciones que realicen esa sola responsabilidad (también conocido como principio de responsabilidad única o Single Responsability).
Una función creacional debe recibir por parametro todo lo que necesite para crear la instancia válida deseada.

1. Implementar la siguiente función creacional

```
//Precondicion: @cadena es una cadena de caracteres
//Postcondicion: Devuelve una instancia válida de cadena de una
//cadena que es copia de @cadena
char* CrearCopia(char* cadena);
```

2. Implementar las siguientes 2 funciones creacionales. PAra entender el dominio del problema ver la definicion de zipic en la guia 1.

```
struct Zipic{
  int* elementos;
  int tamanioZipic;
};

//Precondición: @cadenaZipic es una cadena de caracteres
//Postcondicion: Si @cadenaZipic representa un Zipic valido
//devuelve una instancia de Zipic valido. De lo contrario
// devuelve NULL
//Parametros
//   @cadenaZipic: cadena de caracteres que representa un
//zipic donde cada elemento del zipic esta separado por
//el caracter @separadorElementosZipic
//   @separadorElementosZipic: separador de elementos zipic
//utilizado en@cadenaZipic
Zipic* CrearZipic(char* cadenaZipic, char
separadorElementosZipic);
```

```
struct Zipic{
  int* elementos;
  int tamanioZipic;
};

//Precondición: @zipic es una instancia valida
//Postcondicion: Devuelve otra instancia equivalente a @zipic
Zipic* CrearCopiaZipic(Zipic* zipic);
{
}
```

3. Implementar las siguientes funciones creacionales

```
struct Continente{
  char* nombre;
  int cantidadPaises;
};

struct Pais{
  char* nombre;
  int cantidadHabitantes;
  Continente* continente;
};


Continente* CrearContinente(const char* nombre, int cantidadPaises);

Continente* CrearContinenteVacio(const char* nombre);

Pais* CrearPais(Continente* continente, const char* nombrePais, int cantidadHabitantes);

Pais* CrearPais(const Continente* continente, const char* nombrePais, int
cantidadHabitantes);

Pais* CrearPais(char* nombrePais, int cantidadHabitantes, char* continentePertenencia);
```

4. Implementar las siguientes funciones creacionales

```
struct Campo{
  char *valorCampo;
}

struct Registro{
  Campo *campos;
  int cantidadCampos;
}

Registro* CrearRegistro(char* cadenaRegistro, char separadorCampos);

Registro* CrearRegistros(char* cadenaRegistros, char separadorRegistros, char
separadorCampos);
```

5. Implementar las siguientes funciones creacionales

```
enum Coordenada{
  Norte=1,
```

```
  NorEste=2,
  NorOeste=3,
  Sur=4,
  SurEste=5,
  SurOeste=6,
  Este=7,
  Oeste = 8
};
struct PaisLimitrofe{
  char* nombrePais;
  Coordenada limitaAl;
};
struct Pais{
  char* nombre;
  PaisLimitrofe *paisesLimitrofes;
  int cantidadPaisesLimitrofes;
};


Pais* CrearPais(char *nombre,  PaisLimitrofe *paisesLimitrofes, int
cantidadPaisesLimitrofes);

Pais* CrearPais(char *nombre, char *paisesLimitrofes, char separadorPaises, char
sepradorCampos);

PaisLimitrofe* CrearCopiaPaisLimitrofe(PaisLimitrofe* paisLimitrofe);
```

6. Tipo de dato Email
    a. Definir una estructura que represente un corre electronico. Un email tiene: Destinatarios (direccion de mails), un origen (direccion de email), asunto y un texto (cuerpo del mensaje)
    b. Definir varias funciones creacionales para este tipo de dato
    c. Definir una operacion que permita agregar un email destinatario para un instaancia de un correo electronico