Práctica de ejercicios # 10 - Punteros y arrays

Estructuras de Datos, Universidad Nacional de Quilmes

18 de junio de 2024

Aclaraciones:

- ***Los ejercicios fueron pensados para ser resueltos en el orden en que son presentados.*** *No se saltee ejercicios sin consultar antes a un docente.*
- ***Recuerde que puede aprovechar en todo momento las funciones que ha definido****, tanto las de esta misma práctica como las de prácticas anteriores.*
- ***Pruebe todas sus implementaciones****, al menos en una consola interactiva.*
- *Es sumamente aconsejable* ***resolver los ejercicios utilizando primordialmente los conceptos y metodologías vistos en videos publicados o clases presenciales****, dado que los exámenes de la materia evaluarán principalmente este aspecto. Si se encuentra utilizando formas alternativas al resolver los ejercicios consulte a los docentes.*

1. Registros

Ejercicio 1

Definir el tipo de dato Persona, como un puntero a un registro con el nombre y la edad de la persona. Realizar las siguientes funciones:

- `Persona consPersona(string nombre, int edad)`
  Devuelve a una persona nueva, con el nombre y la edad dados
- `string nombre(Persona p)`
  Devuelve el nombre de una persona
- `int edad(Persona p)`
  Devuelve la edad de una persona
- `void crecer(Persona p)`
  Aumenta en uno la edad de la persona.
- `void cambioDeNombre(string nombre, Persona p)`
  Modifica el nombre una persona.
- `bool esMayorQueLaOtra(Persona p1, Persona p2)`
  Dadas dos personas indica si la primera es mayor que la segunda.
- `Persona laQueEsMayor(Persona p1, Persona p2)`
  Dadas dos personas devuelve a la persona que sea mayor.

Ejercicio 2

Modelaremos los tipos de datos *Pokemon*, como un *TipoDePokemon* (agua, fuego o planta, sinónimo de *string*) y un porcentaje de energía (que inicia en 100); y *Entrenador*, como un nombre, una cantidad de pokémon y un array de pokémon. Así, la representación es la siguiente:

```
typedef string TipoDePokemon;

struct PokeSt {
   TipoDePokemon tipo;
   int vida;
}

typedef PokeSt* Pokemon;

struct EntrenadorSt {
   string nombre;
   Pokemon* pokemon;
   int cantPokemon;
}

typdef EntrenadorSt* Entrenador;
```

Dicho esto, implementar la siguiente interfaz de *Pokemon*:

- `Pokemon consPokemon(TipoDePokemon tipo)`
  Dado un tipo devuelve un pokémon con 100 % de energía.
- `TipoDePokemon tipoDePokemon(Pokemon p)`
  Devuelve el tipo de un pokémon.
- `int energia(Pokemon p)`
  Devuelve el porcentaje de energía.
- `void perderEnergia(int energia, Pokemon p)`
  Le resta energía al pokémon.
- `bool superaA(Pokemon p1, Pokemon p2)`
  Dados dos pokémon indica si el primero, en base al tipo, es superior al segundo. Agua supera a fuego, fuego a planta y planta a agua. Y cualquier otro caso es falso.

Una vez hecho eso, implementar la siguiente interfaz de *Entrenador*:

- `Entrenador consEntrenador(string nombre, int cantidad, Pokemon* pokemon)`
  Dado un nombre, una cantidad de pokémon, y un array de pokémon de ese tamaño, devuelve un entrenador.
- `string nombreDeEntrenador(Entrenador e)`
  Devuelve el nombre del entrenador.
- `int cantidadDePokemon(Entrenador e)`
  Devuelve la cantidad de pokémon que posee el entrenador.
- `int cantidadDePokemonDe(TipoDePokemon tipo, Entrenador e)`
  Devuelve la cantidad de pokémon de determinado tipo que posee el entrenador.
- `Pokemon pokemonNro(int i, Entrenador e)`
  Devuelve el pokémon número i de los pokémon del entrenador.
  Precondición: existen al menos $i - 1$ pokémon.
- `bool leGanaATodos(Entrenador e1, Entrenador e2)`
  Dados dos entrenadores, indica si, para cada pokémon del segundo entrenador, el primero posee al menos un pokémon que le gane.

2. Array Lists

Ejercicio 3

Dada la siguiente representación de listas, llamada *ArrayList*:

```
struct ArrayListSt {
   int cantidad;   // cantidad de elementos
   int* elementos; // array de elementos
   int capacidad;  // tamaño del array
}

typedef ArrayListSt* ArrayList;
```

Definir la siguiente interfaz de este tipo de listas:

- `ArrayList newArrayList()`
  Crea una lista con 0 elementos.
  Nota: empezar el array list con capacidad 16.
- `ArrayList newArrayListWith(int capacidad)`
  Crea una lista con 0 elementos y una capacidad dada por parámetro.
- `int lengthAL(ArrayList xs)`
  Devuelve la cantidad de elementos existentes.
- `int get(int i, ArrayList xs)`
  Devuelve el iésimo elemento de la lista.
- `void set(int i, int x, ArrayList xs)`
  Reemplaza el iésimo elemento por otro dado.
- `void resize(int capacidad, ArrayList xs)`
  Decrementa o aumenta la capacidad del array.
  Nota: en caso de decrementarla, se pierden los elementos del final de la lista.
- `void add(int x, ArrayList xs)`
  Agrega un elemento al final de la lista.
- `void remove(ArrayList xs)`
  Borra el último elemento de la lista.

Ejercicio 4

Definir las siguientes funciones utilizando la interfaz de *ArrayList*:

1. `int sumatoria(ArrayList xs)`
   Devuelve la suma de todos los elementos.
2. `void sucesores(ArrayList xs)`
   Incrementa en uno todos los elementos.
3. `bool pertenece(int x, ArrayList xs)`
   Indica si el elemento pertenece a la lista.
4. `int apariciones(int x, ArrayList xs)`
   Indica la cantidad de elementos iguales a x.
5. `ArrayList append(ArrayList xs, ArrayList ys)`
   Crea una nueva lista a partir de la primera y la segunda (en ese orden).

6. `int minimo(ArrayList xs)`
   Devuelve el elemento más chico de la lista.