Trabajo Práctico N°1:
¡Gran pesca Pokemón!

Algoritmos y Programación II
Cátedra Méndez-Camejo
FIUBA

- Alumna: María Pilar Gaddi.
- Padrón: 105682

Introducción:

La entrenadora de pokemónes y líder del gimnasio, Misty, organizo un evento de pesca que se llevará a cabo en su acuario y, para ello, necesita un programa que pueda trasladar aquellos pokemónes que cumplan con ciertas condiciones desde el arrecife (donde todos los pokemones, en principio, se encuentran allí) hasta su propio acuario, para que todos los seleccionados pudieran competir y mostrar sus habilidades.

El presente trabajo consta de **2 archivos**:

- **Pokemones.c**, donde transcurre el flujo principal del programa
- **Evento_pesca.c**, donde se implementan las funciones de la biblioteca **evento_pesca.h**, brindada por la catedra.

Con respecto a la implementación de las funciones que se haya en los mismos, nuestro programa tiene dos pilares:

- Un **vector dinámico** que representa el **arrecife**, ubicado en el **HEAP**. En cada elemento del mismo se encuentran los **pokemones que pertenecen al arrecife**.
- Un **vector dinámico** que representa el **acuario**, ubicado en el **HEAP**. En cada elemento del mismo se encuentran los **pokemones que pertenecen al acuario.**

En principio, la información de los pokemones que irán en el vector arrecife será extraída de un archivo de texto que nos dará Misty, y que **debe ser el segundo argumento de la línea de comando**, al ejecutar el programa.

Cabe aclarar que, **en caso de que la apertura del archivo de texto falle**, el programa solo mostrará un mensaje de aviso y finalizará, a priori liberará la memoria anteriormente reservada por malloc().

En caso de que:

- **La apertura del archivo de texto se haya realizado correctamente**
- **La extracción de los pokemones a partir del archivo de texto (para que sean incorporados al vector de pokemones en arrecife) se haya realizado correctamente**
- **La creación del acuario (acción realizada por malloc(), es decir, que este no falle) haya sido exitosa**

El programa seguirá su flujo y, primeramente, se mostrará el listado de los pokemones sin filtrar. A partir de ahí, el usuario **podrá elegir:**

- **Filtrar los pokemones**, ubicados en principio en el vector arrecife, que cumplen con ciertas habilidades representadas en 5 funciones booleanas y que se encuentran en el archivo pokemones.c:

    - **Tiene_mejor_color**
    - **Tiene_cuatro_letras**
    - **Tiene_velocidad_par**
    - **Es_edicion_limitada**
    - **Tiene_velocidad_promedio**

- **No filtrar los pokemones** (de esta forma también finaliza el programa, con un mensaje de aviso).

Luego de que los pokemones sean filtrados (primera opción):

Aquellos que **sigan en el arrecife** se van a ir **mostrando por pantalla en forma de lista, con diferentes formatos**. Estos mismos seguirán estando en el vector arrecife.

Aquellos pokemones que **hayan sido trasladados al acuario** (ya que alguna de las funciones booleanas mencionadas anteriormente devolvió true y la cant_seleccionada no había alcanzado su máximo), no seguirán estando en el vector arrecife y serán trasladados al vector acuario.

Como elegí realizar el traslado de pokemones:

Para que los pokemones puedan **comenzar a ser trasladados** desde el arrecife hasta el acuario, se debe cumplir la condición de que la **cantidad de pokemones en condiciones de ser filtrados, sea igual o mayor a la cant_seleccion (esta cantidad también debe ser mayor 0).**

En caso de realizarse el traslado, la cantidad máxima de pokemones para ser trasladados será la siempre la de cant_seleccion
Agrego el pokemon seleccionado al vector de pokemones en el acuario (mediante la función agregar_pokemon),

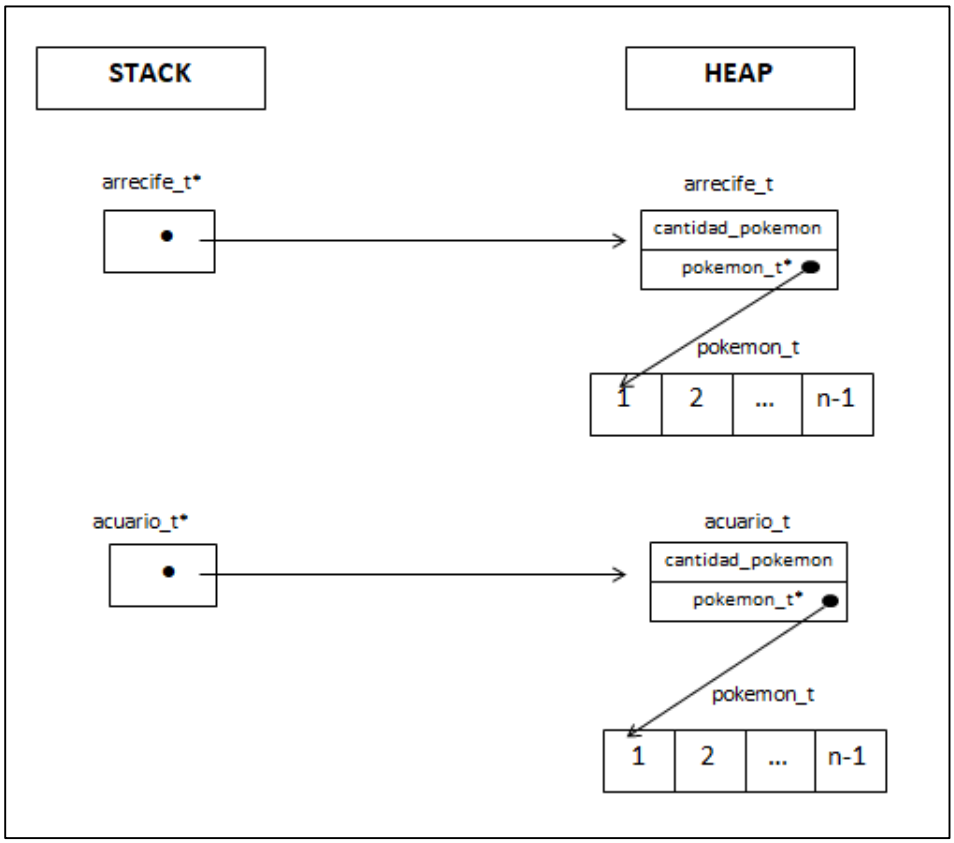
Procedimiento:

- Aumento (uno en uno) la cantidad de pokemones en el acuario
- Mediante la función agregar_pokemon, agrando el bloque de memoria al cual apunta el puntero que le paso como parámetro a realloc() (acuario->pokemon) y luego avanzo una determinada **cantidad** de elementos en el vector (cantidad que la función recibe como parámetro) para posicionar allí al nuevo pokemon.
- Elimino ordenadamente el pokemon seleccionado del vector arrecife (mediante la función eliminar_elemento) y luego achico el bloque de memoria al cual apunta el puntero que le paso como parámetro a realloc() (arrecife->pokemon), mediante la función achicar_arrecife.

Luego de realizarse todos los traslados y las muestras por pantalla necesarias, se procede a **cargar aquellos pokemones que se encuentran en el vector acuario en un archivo de texto .txt** para que Misty pueda tener una lista de aquellos que se encuentran en su acuario.

Finalmente, se termina liberando toda la memoria reservada por malloc durante la ejecución del programa.

Representación de la memoria del STACK y el HEAP en el programa:



Forma de compilación:

El programa debe ser compilado bajo la siguiente línea de compilación:

gcc *.c -Wall -Werror -Wconversion -std=c99 -o evento_pesca

Forma de ejecución:

El programa debe ser ejecutado bajo la siguiente línea de ejecución:

./evento_pesca nombre_arch_arrecife

Siendo el **nombre_arch_arrecife** el nombre del archivo de arrecife que se desea utilizar (argv[1])

Explicación de algunos conceptos…

Punteros:
Un puntero es una variable que **almacena una dirección de memoria** de otra variable. De esta forma, cuando los utilizamos (pasaje de parámetros por referencia), cada modificación que se le haga a un parámetro dentro de una función, también se ve plasmada en la función de la cual se llamo.

Aritmética de punteros:

Para acceder a la dirección de memoria de una variable utilizo el operador **& (operador de dirección)**

Para acceder al valor que almacena la variable a la que apunta un puntero, se utiliza el operador *** (operador de indirección).** También este mismo operador permite declarar un tipo de dato puntero.

Ejemplo:

Caso 1: Contenido de la dirección de memoria apuntada por pokemon_t*
Caso 2: Direccion de memoria a la que apunta pokemon_t*

```
//Contenido de la direccion de memoria de la "i" celda del vector

printf("%s", (arrecife->pokemon+i)->especie);
printf("%s", (arrecife->pokemon)[i].especie);
printf("%s", ((*arrecife).pokemon)[i].especie);
printf("%s", ((*arrecife).pokemon+i)->especie);

//Los ultimos 4 ejemplos son equivalentes (tienen el mismo contenido de la direccion de memoria).

//Direccion de memoria de la "i" celda del vector

printf("%p", &((arrecife->pokemon+i)->especie));
printf("%p", &((arrecife->pokemon)[i].especie));
printf("%p", &(((*arrecife).pokemon)[i].especie));
printf("%p", &(((*arrecife).pokemon+i)->especie));
printf("%p", ((arrecife->pokemon+i)));

//Los ultimos 5 ejemplos son equivalentes (tienen la misma direccion de memoria).
```

Punteros a funciones:

Los punteros a funciones son variables que guardan una dirección de memoria de una función. Estos sirven para poder “parametrizar” el código.

En el TP se utilizaron punteros a funciones, por ejemplo, para poder mostrar con diferentes formatos las listas a medida que los pokemones se iban filtrando. Tambien se utilizaron en las funciones booleanas que servian de filtro.

Malloc y realloc:

La función malloc() le indica al sistema operativo que necesita usar memoria dinámica (del HEAP). Además, reserva *size* bytes que se le pasan como parámetro y devuelve la dirección del primer byte de memoria que reservo o NULL en caso de que haya fallado. La estructura es la siguiente:

void *malloc(size_t size)

Además, viene acompañada siempre de la función free() , que se encarga de liberar ese espacio de memoria apuntado por el puntero devuelto previamente

por malloc() y inicializar ese puntero a NULL. En caso de que puntero == NULL (previamente a free()), la operación no se realiza. La estructura es la siguiente:

void free(void *puntero)

La función realloc() modifica el tamaño del bloque de memoria apuntado por puntero size bytes. La estructura es la siguiente:

void *realloc(void* puntero, size_t size)