

# Auxiliar 1

## Introducción a objetos en C++

Profesora: Nancy Hitschfeld-Kahler

Auxiliar: Vicente González

*Basado en las diapositivas de Sergio Salinas*

CC7515-1 – Computación en GPU

March 26, 2024

- 1 Introducción a C++
- 2 Actividad
- 3 Clases en C++
  - Clases abstractas
  - Creación de un objeto
- 4 Constructores
- 5 Biblioteca estándar de C++
- 6 Overloading
- 7 Compilación
- 8 Tests
- 9 Bibliografía

# Introducción a C++

- Desarrollado por Bjarne Stroustrup en 1983 como una extensión del lenguaje C.
- Lenguaje de bajo nivel con control de memoria necesarias para aprovechar la arquitectura de la GPU.
- Se recomienda usar el estándar de C++ moderno (C++ 11 en adelante).

# Hello world

hello.cpp

```
#include <iostream>
```

```
int main() {  
    std::cout << "Hello, world!" << std::endl;  
    return 0;  
}
```

```
//g++ hello.cpp -o hello
```

# Actividad

- Crear un pequeño proyecto
- Ejecutables y librerías
- Aprender a hacer clases en C++
  - Pokemon
  - Movimientos
  - Tipos
- Uso de CMake y Google Test
- Tips del auxiliar para trabajar más cómodamente :D

Vamos a crear un proyecto “típico” de C++, con las siguientes carpetas:

- `src`: Donde van todos los archivos `.cpp` y `.h` para compilar el proyecto.
- `include`: Reservado para headers “públicos” cuando se hacen librerías o usan.
- `external`: Dependencias del proyecto.
- `test`: Donde dejar los tests del proyecto.
- `build`: Donde van a ir nuestros ejecutables y cositas de CMake.

Clone la repo <https://github.com/Seivier/CPPokemon.git>



# Clases en C++

# Introducción a Clase en C++

- En C++, un objeto es una instancia de una clase.
- Una clase es una estructura de datos que define un conjunto de atributos y métodos que operan sobre esos atributos.
- Los objetos se pueden crear dinámicamente en tiempo de ejecución usando punteros y el operador `new`.
- Los objetos se eliminan usando el operador `delete` para liberar la memoria asignada a ellos.
- Se pueden usar tanto `class` como `struct`

```
----- pkm.h -----  
struct Pokemon {  
    virtual void speak() const = 0;  
    virtual const std::string& name() const = 0;  
    virtual int hp() const = 0;  
    virtual int maxHp() const = 0;  
    virtual const Type& type() const = 0;  
};
```

Una *interface* en C++ !!!

- Cree las clases APokemon, AMovement y AType donde encapsule el funcionamiento común
- Para APokemon puede crear variables privadas para la vida y el tipo.
- Para AMovement puede crear variables privadas para el poder, el tipo y la precisión del ataque.
- Para AType puede establecer el comportamiento por defecto de las funciones.

## Ejemplo

```
class APokemon {  
public:  
    // ...  
    int hp() const override;  
private:  
    // ...  
    int _hp; // o mHp;  
};
```

Recuerde que la declaración va en un .h y la implementación en un .cpp

# Constructores

# Tipos de constructores y destructores

```
// Constructor por defecto
```

```
Rectangulo(): ancho(0), alto(0) {}
```

```
// Constructor con parámetros
```

```
Rectangulo(double ancho, double alto) {
```

```
    this->ancho = ancho;
```

```
    this->alto = alto; }
```

```
// Constructor copia
```

```
Rectangulo(const Rectangulo& r) {
```

```
    this->ancho = r.ancho;
```

```
    this->alto = r.alto; }
```

```
// Destructor
```

```
~Rectangulo() {
```

```
    std::cout << "Se ha destruido un rectangulo." << std::endl;
```

```
}
```

# Tipos de constructores y destructores

- Cree constructores para sus clases, donde inicialice cada variable
- Aseguresé de crear todo lo necesario en el constructor
- Debe hacerse cargo de la memoria pedida en el destructor

## IMPORTANTE!

Usted no puede usar una clase abstracta como miembro de la clase, un lugar de eso deber usar un puntero



# Biblioteca estándar de C++

# Biblioteca estándar de C++

Conjunto de funciones, objetos y clases que proporcionan una amplia variedad de características y funcionalidades para el lenguaje C++. Por ejemplo:

- Entrada/salida: operaciones de entrada y salida, como leer o escribir en archivos o en la consola.
- Contenedores: estructuras de datos para almacenar y manipular colecciones de objetos, como vectores, listas, mapas, etc.
- Algoritmos: funciones para realizar operaciones comunes en contenedores, como ordenar, buscar, mezclar, etc.
- Tipos de datos: tipos de datos comunes, como cadenas de caracteres, booleanos, números, etc.
- Funciones matemáticas: funciones matemáticas comunes, como seno, coseno, exponencial, etc.

La Biblioteca estándar de C++ está disponible en cualquier compilador que cumpla con el estándar de C++. Para usarla, se debe incluir el archivo de cabecera correspondiente.

# Bibliotecas de la librería estandar

- `<iostream>`: para entrada/salida de consola
- `<vector>`: para el uso de vectores dinámicos
- `<string>`: para el uso de cadenas de texto
- `<algorithm>`: para el uso de algoritmos de ordenación, búsqueda, etc.
- `<unordered_map>` y `<map>`: para el uso de mapas y diccionarios
- `<set>`: para el uso de conjuntos
- `<cmath>`: para el uso de funciones matemáticas como `sqrt()`, `cos()`, `sin()`, etc.
- `<chrono>`: para el uso de medidas de tiempo
- `<memory>`: para el uso de puntero inteligentes
- etc

Más en [https://en.cppreference.com/w/cpp/standard\\_library](https://en.cppreference.com/w/cpp/standard_library)

- Existen 3 tipos de punteros “inteligentes” en la STL
- Funcionan como puntero normales desde afuera
- El uso correcto de estos garantiza el despreocuparse por el manejo de memoria
- `std::shared_ptr<T>`: Permite crear un puntero compartido, va contando las referencias y autodestruye cuando el contador llega a 0
- `std::weak_ptr<T>`: Permite crear un puntero único, el cual no permite compartir la información que contiene con otros punteros
- `std::weak_ptr<T>`: Permite crear un puntero débil, se comporta de manera similar a un puntero clásico y sirve para referencia cíclicas

Reemplace sus punteros clásicos por `std::unique_ptr<T>`, puede usar `std::make_unique<T>` para crear uno

Ejemplo

```
class C {  
public:  
    int x, y, z;  
    C(int x, int y, int z);  
}  
  
int main() {  
    std::unique_ptr<C> p;  
    p = std::make_unique<C>(10, 20, 30); // llama al constructor  
    std::cout << p->x << std::endl; // "10"  
}
```

- Permite manejar strings de manera cómoda
- Funciona como cualquier string de otro lenguaje tipado

Añada la siguiente función a Pokemon

```
_____ pkm.h _____  
struct Pokemon {  
    // ...  
    virtual const std::string& name() const = 0;  
}
```

E implementela en la clase abstracta

- Los vectores de la STL son arreglos típicos de otros lenguajes
- Permiten manipular arreglos de tamaño dinámico alojados en el heap
- Usando esto y los punteros inteligentes puede evitar tratar con `new` y `delete` casi en su totalidad



Añada a APokemon un arreglo de movimientos Movement

# Overloading

# Overloading de operadores

El **overloading de operadores** es una funcionalidad en C++ que permite a los desarrolladores definir una función que se comporta como un operador. Por ejemplo, para sumar dos números complejos, podemos sobrecargar el operador `+` de la siguiente manera:

```
class Complejo {  
public:  
    Complejo operator+(const Complejo& c) const {  
        return Complejo(real + c.real, imag + c.imag);  
    }  
private:  
    double real;  
    double imag;  
};
```

En este caso, estamos sobrecargando el operador `+` para que sume dos números complejos. El operador se define dentro de la clase `Complejo` y se utiliza el operador `+` para definir la función.

# Overloading de operadores

Añada la siguientes funciones a Pokemon en util/pkm.h

```
----- pkm.h -----  
struct Pokemon {  
    // ...  
    virtual Movement& operator[] (const std::size_t idx) const = 0;  
}  
std::ostream& operator<<(std::ostream& os, const Pokemon& pok);  
-----
```

E implementela en la clase abstracta

# Compilación

Creemos unas clases concretas para poder compilar el proyecto

- Los tipos Normal, Rock y Fighting.
- Los siguientes Pokémon:
  - Bidoof de tipo Normal con 60 de vida
  - Mankey de tipo Fighting con 100 de vida
- Los siguientes Movimientos:
  - Tackle de tipo Normal con 40 de poder y 100 de precisión
  - CrossChop de tipo Fighting con 100 de poder y 80 de precisión
  - Rollout de tipo Rock con 30 de poder y 90 de precisión

## Makefile

```
INC=include/
SRC=main.cpp pokemon.cpp movement.cpp
BUILD=build/

.PHONY: build clean

build:
    mkdir -p build/
    $(CC) $(LDFLAGS) $(SRC) -I$(INC) -o $(BUILD)main

clean:
    rm -rf build
```

# CMake o Makefile?

- CMake
  - Estándar moderno de C++
  - Complejo y estructurado
  - Multiplataforma
  - Relativamente intuitivo
  - Basado en macros y funciones
- Makefile
  - Pensado con *build tool* de Linux
  - Sencillo y directo
  - Basado en reglas
  - Ideal para automatizar comandos



```
Cmake en bash
```

---

```
$ cmake -S . -B build
    # S es para el codigo fuente
    # B es donde dejar los archivos generados

$ cmake --build build
    # build construye en ejectable o libreria

$ ./build/src/CPPokemonRun
    # con esto se ejecuta la aplicación
```

---

# Tests

# Testing con Asserts en C++

En C++, se pueden hacer pruebas unitarias utilizando asserts.

- Un assert es una macro que verifica una expresión y termina el programa si la expresión es falsa.
- Los asserts son útiles para detectar errores lógicos en el programa, como valores inválidos de parámetros o errores de cálculo.

Para utilizar los asserts, se debe incluir la biblioteca `cassert` y luego utilizar la macro `assert` con la expresión a evaluar:

```
#include <cassert>

int dividir(int a, int b) {
    assert(b != 0);
    return a / b;
}
```

Si la expresión `b != 0` es falsa, el programa terminará en ese punto y mostrará un mensaje de error.

Google Test es un framework de testing más sofisticado

- Diversos tipos de asserts
- Permite el uso de clases para crear suites de tests
- Similar a jUnit de Java o mUnit de Scala
- Su uso no es obligatorio pero muy recomendado

Para usarlo en conjunto con CMake:

<https://google.github.io/googletest/quickstart-cmake.html>

O puede ver el siguiente video:

<https://youtu.be/pxJoVRfpRPE?si=hqbv1whpbNtKcrzl>

CMake y Gtest serán explicado en más en profundidad en la próxima auxiliar

```
_____ root _____  
cmake_minimum_required(VERSION 3.20)  
  
project(CPPokemon)  
  
set(CMAKE_CXX_STANDARD 20)  
set(CMAKE_CXX_STANDARD_REQUIRED True)  
set(CMAKE_EXPORT_COMPILE_COMMANDS True)  
  
include(CTest)  
add_subdirectory(external)  
add_subdirectory(src)  
add_subdirectory(test)
```

src

```
add_library(${PROJECT_NAME} STATIC movement.cpp pokemon.cpp)
target_include_directories(${PROJECT_NAME} PUBLIC
↳  ${PROJECT_SOURCE_DIR}/include)

add_executable(${PROJECT_NAME}Run main.cpp)
target_link_libraries(${PROJECT_NAME}Run PUBLIC ${PROJECT_NAME})
```

extern

```
add_subdirectory(googletest)
```

```
_____ test _____  
if(BUILD_TESTING)  
  add_executable(tests test.cpp)  
  target_link_libraries(tests PRIVATE GTest::gtest_main  
    ↪  ${PROJECT_NAME})  
  add_executable(${PROJECT_NAME}::tests ALIAS tests)  
  
  include(GoogleTest)  
  gtest_discover_tests(tests)  
endif()
```

Puede ver en detalle el proyecto en la rama [reference](#)

## Bibliografía



# Bibliography

Stroustrup, B. (2018). A Tour of C++, Second Edition.

