**Apuntes Métodos Computacionales 2**

Semana 1 Viernes (31/03/2023)

**Data types:**

Int -> 4 bytes \* 8 bits

32 bits

Espacio de representación: 2^31-1, -2^31

Char

**Prefix and postfix:**

Hay dos tipos

x++

++x

**If statement:**

**While loop:**

También hay do-while

**For loop:**

DUDA: los arreglos se crean vacíos o con valor predet (i.e. 0)? RTA: “vacíos”.

También hay vectores (esos sí cambian len)

**Switch statement:**

*if* secuenciales más sencillos:

El *else* sería *default*

**And/Or**

&& y ||

**Arrays**

En los arrays no se pueden mezclar tipos de datos como en las listas de Python.

DUDA: Insert. RTA: se usa con vectores, para añadir elementos (Python style)

**Multidimensional Arrays:**

**Functions:**

Se debe indicar el tipo de retorno (si no retorna, se coloca *void*).

DUDA: Si se quiere pasar un parametron desconocido, se coloca *auto.* RTA: sirve también para inicializar variables con el tipo de datos adecuado dependiendo de lo que se le asigne (sin tener que especificarlo explícitamente).

**Default parameters:**

Igual que en python.

**OOP:**

Constructor: crea una instancia (objeto) de la clase. Al invocar una clase, se ejecuta automáticamente el constructor.

**Classes:**

Atributos suelen ser privados

Constructor es una función que tiene el mismo nombre que la clase (en Python el constructor es con init)y es privado.

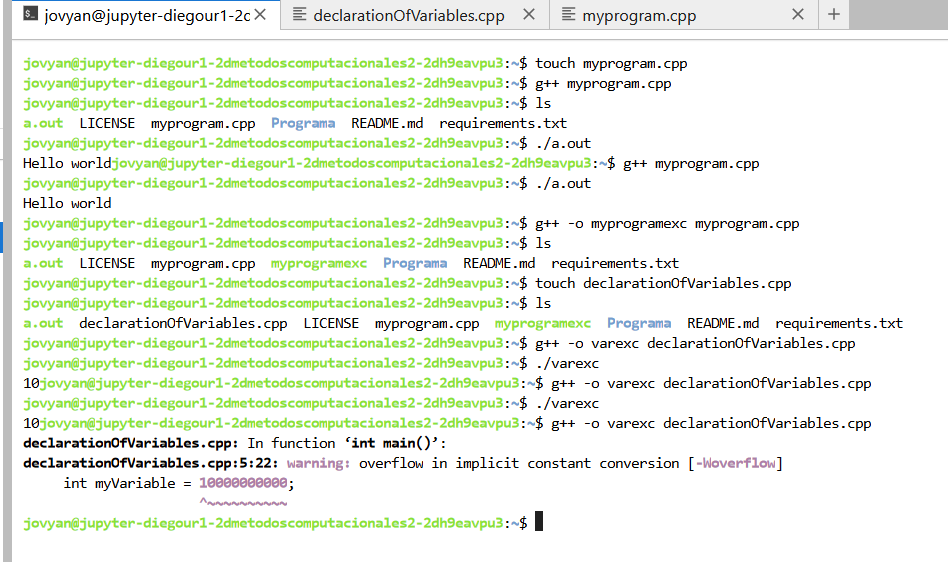
Al crear un objeto se llama al constructor automáticamente.

Además de *public* y *private*, hay *protected* (igual que en Java).

OTRAS DUDAS:

Buscar printf. RTA: printf("format string", arg1, arg2, ...); %d, %f y %s para int, float y string.

Cambiar tamaño de array. RTA: como tal no se puede por medio de una built-in, pero hay dos opciones: crear un nuevo array y trasladar los valores antiguos o usar un arreglo dinámico (ver Chat).



Texto

Descripción generada automáticamente

Semana 2 Miércoles

DUDA: this en c++? RTA: En Java, "this" se utiliza principalmente para evitar conflictos de nombres entre variables locales y miembros de la clase, mientras que en C++, se utiliza para acceder a los miembros de la clase a través de punteros (se usa this->nombre=pNombre).

**Punteros:**

&score indica la dirección en memoria donde se almacena el int score. Al imprimierlo, se muestra un número hexadecimal.

Un pointer se define con \*. Una variable puntero guarda la dirección en memoria.

El puntero guarda la dirección.

**Pass by value vs pass by reference:**

Dentro de las funciones no cambia el valor de las variables afuera (en Python sí, pero en C++ no).

Lo que se hace es pasarle la dirección a la función. Dentro de la función se cambia el valor a la variable.

a -> ptrA => &a

a <- ptrA => \*prtA (de referencia)

Una cosa es inicializar el pointer con \* y otra cosa es usar \* para acceder al valor.

**Función swap:**

a=1, b=2 -> a=2, b=1

en Python sería

*def swap (a, b):*

*temp = a*

*a = b*

*b = temp*

en cpp se debe hacer by reference y no by value:

DUDA: arreglar input en VSC. RTA: listo.

Semana 2 Viernes (14/04)

**Stack vs heap memory:**

Stack: se guarda en bloques continuos. Más fácil acceso. Eliminación de las variables se hace de forma automática.

Heap: se guarda en orden diferentes. La eliminación de variables no es automática (manual).

El Heap tiene más memoria que el Stack (esa es su ventaja).

**Create a variable in heap:**

Se usa new para crear variable en heap. Si no se usa new, se está trabajando en stack.

Para eliminar la variable se usa delete.

Si no, hay memory leak.

DUDA: memory leak. RTA: es un problema que ocurre en programas de computadora cuando la memoria asignada dinámicamente no se libera adecuadamente.

En Python y Java no se necesita delete (el garbage collector).

**Makefiles:**

Forma de estructurar proyectos grandes.

Preprocessor instruction: #include (para librerías) y #define (para constante).

DUDA: #define. RTA: es una directiva de preprocesador que se utiliza para definir macros o constantes simbólicas. Las macros pueden ser utilizadas para reemplazar texto o funciones en el código, mientras que las constantes simbólicas son valores fijos que no cambian durante la ejecución del programa.

Estas instrucciones se envían antes de que compile el archivo. Existen de manera más global.

Las variables del código existen en el archivo y las constantes existen en todo lado.

**#ifndef**:

Si no está definido, defínalo

**File guards and headers:**

Archivos headers: son como archivos cpp pero se definen “.h”.

.cpp y .h

En .h solo se nombran las funciones existentes.

En .cpp se implementan las funciones.

.h

perimeter()

area()

.cpp

perimeter{

a+b+c

}

DUDA: arreglar compilar varios archivos enlazados en VSC. RTA: listo. Toca desde consola para enlazarlos, como en se hace en el makefile:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Error al define dos veces

**Solution: use ifndef**

**Function overloading:**

Otras formas son con template (mejor) y auto.

**Makefile:**

Su estructura es:

Se tienen targets: prerequisites

targets son archivos objetivos.

Ejemplos:

G++ -c no los compila por completo, pero sí los pasa a lenguaje de máquina.

-c crea un archivo.o que no está relacionado con los otros archivos.

-o

**Ejemplo:**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

Si hubiera mil archivos, el g++ … sería enorme. Para eso se usa un makefile.

Makefile es archivo sin extensión.

En Binder no se reconoce el tab, entonces se usa un ; para remplazar el tab.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ahora sí el Make file bien

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

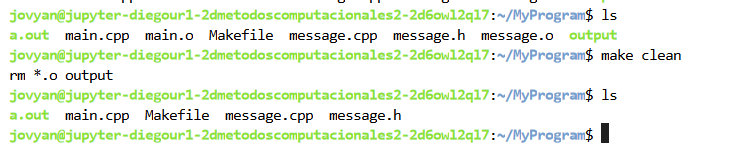
Descripción generada automáticamente

Pero quedan muchos archivos.

Entonces se usa clean

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente



**Ejercicio:**

// compilar y ejecutar el cpp

// eso genera el xvalues

// y luego ejecuta python

// no hay que crear archivo python

// python nombrearchivo.py

DUDA: -o y -c en c++. RTA: Un archivo objeto es un archivo binario intermedio que contiene el código compilado pero no enlazado. Enlazar es el proceso de combinar uno o más archivos objeto y, opcionalmente, bibliotecas para crear un archivo ejecutable o una biblioteca compartida. En resumen, un archivo objeto contiene el código de máquina compilado, y enlazar es el proceso de combinar estos archivos objeto para crear un archivo ejecutable completo. Enlazar es necesario porque un programa puede estar compuesto por múltiples archivos fuente, cada uno de los cuales puede contener definiciones de funciones, variables y otros elementos que deben ser referenciados o utilizados en otros archivos fuente. Enlazar garantiza que todas estas referencias se resuelvan correctamente y se combinen en un solo archivo ejecutable.

DUDA: ifndef define endif para qué? RTA: ver “Consulta#1 MC2” en Chat.

Texto

Descripción generada automáticamente

DUDA: template y auto. Template con & en la tarea. RTA: Los templates en C++ son una forma de lograr la programación genérica. Permiten escribir funciones o clases que pueden operar en diferentes tipos de datos sin tener que duplicar el código. Los templates son una forma de reutilizar código al permitir que los tipos de datos se definan en tiempo de compilación. La palabra clave auto en C++ permite que el compilador deduzca automáticamente el tipo de una variable a partir de su valor inicial. Es útil cuando se trabaja con tipos de datos complicados o cuando no quieres especificar explícitamente el tipo de dato. Ver “Templates y Referencias C++” en Chat.

DUDA: arreglar makefile en VSC. RTA: nel.

DUDA: OOP en Python. RTA: similar a Java. Pero con *\_\_x\_\_* y *self* en todos lados. El constructor es *\_\_init\_\_* y el *toString()* es *\_\_str\_\_*. El *self* es como un *this*.

Semana 3 Miércoles:

**Finalizar Cpp**

**Crear puntero a clase:**

¿Qué pasa si se hace ptrMytriangle.a?

Bota error.

En este caso se usa -> para acceder a los atributos.

Por ejemplo:

ptrMytriangle -> a

ptrMytriangle -> perimeter

this-> es el puntero de la clase.

this->a = 3 equivale a a = 3

this hace referencia ptrMytriangle

DUDA: this

Temas que faltaron:

-templates

-destructores: (similar a constructor, pero elimina las variables). En stack son automáticos, en heap toca definirlos para que eliminen los variables.

-filestream: para leer, copiar y escribir archivos.

**Fin Cpp**

**Transformada de Fourier**

Se tiene una función

Lo que dice Fourier es que una función f(x) periódica se puede escribir:

Sirve porque seno y coseno son ortonormales:

Porque sen(x) es impar y cos(x) es par.

Cómo se ve? Si f(x) es impar

La integral

El producto de impar con par da una función impar y al integral también da 0.

Ahora,

Cómo se ve? Ambas son seno (impares), pero el periodo es diferente (no se sobrelapan).l

[ Lele pancha :c ]

Es importante la integral.

Lo mismo con coseno

Entonces se tiene

De la suma solamente sobrevive una parte.

Para hallar a\_o se coge la f(x) y se integra, las sumatorias se anulan y queda a\_0.

Es importante para mecánica cuántica.

En el ejemplo de la función step queda

Ahora en Python.

¿Cómo demostrar que Fourier puede generar cualquier función periódica?

Tiene que ser una base para el espacio.

Esto es casi la transformada de Fourier.

Semana 3 Viernes:

**Transformada de Fourier**

Usando senos y cosenos:

Usando un omega fundamental

Hacer rectángulos con ancho muy pequeño para generar la integral.

Transformada de Fourier es una versión de la serie de Fourier, con periodo infinito o cuando los w tienden a 0.

Con ortogonalidad se puede hacer:

La Transformada de Fourier inversa queda:

Aunque suele usarse al revés:

T.F.

T.F.I.

**Ejemplo con coseno:**

Calcular TF de onda que tiene frecuencia omega\_0

¿Cuánto da?

Entonces

¿Qué significa? En omega\_0 hay resonancia. Es decir, hay un pico infinito solo en esa frecuencia.

**Ejemplo:**

Ahora se usará posiciones con números de onda:

Se calcula la TF

Da otra gaussiana. Con una menor desviación estándar: ahora es 1/sigma.

Ocurre en **principio de incertidumbre**.

Mucha incertidumbre en posición y poca incertidumbre en momentos, y viceversa.

**La relación de la Transformada de Fourier con la Mecánica Cuántica.**

La ecuación de Schrödinger

Partícula libre, cuya solución es

Una partícula libre se puede ver como una onda con

La función de onda es

La probabilidad de que se encuentre en todo el espacio es 1.

También está el espacio de momentos. Es la probabilidad de que la partícula tenga cierto momento.

Como no había nada de certidumbre de la posición, hay total certidumbre de su momento.

También está el caso de los paquetes de onda. En ese caso la función de onda se ve así:

La probabilidad termina siendo una gaussiana con desviación sigma. Ahora se pasa al estado de momentos.

También es una gaussiana. Pero la desviación es diferente. En esta la desviación es (1/sigma)\*(hbar/2).

En el caso de la partícula libre, para que esté totalmente localizada, no cuadra con la Relatividad. Ahí entra Dirac y se crea la ecuación de Dirac.

Semana 4 Miércoles:

**Transformada Discreta de Fourier (DFT)**

Teníamos la serie de Fourier

Para periodos infinitos se obtiene la transformada

Es útil para encontrar las frecuencias.

La serie de Fourier también se puede escribir de una formam exponencial:

Si se tiene una frecuencia fundamental

Se puede crear como bins en un histograma de h(f) vs f. Donde cada rectángulo está en un nf\_0, cada rectángulo tiene un

Para la discreta se van a sumar de las áreas c\_n

Como va de -inf a inf, sería imposible implementarlo. Entonces toca acortarlo. Se aprovecha la periodicidad del exponencial.

Cada vez que el exponente vale 2pi, se va a repetir.

Se divide el espacio entre 0 a 1 en N pedazos de 1/N.

Se divide el espacio , entonces la frecuencia fundamental queda

Debido a la periodicidad, solo toca ir de 0 a N. Se coge el periodo y se parte en N partes.

Importante:

Se hace un linspace para varios tiempos: f(t\_0), f(t\_1), f(t\_2),…

Donde cualquier tiempo cumple .

Entonces la transformada de Fourier queda

Se divide espacio de tiempos y de frecuencias por N. Para cada tiempo hay una frecuencia.

El término k sirve para calcular recorrer el tiempo.

n significa que se está recorriendo sobre el tiempo (t).

x\_m es la ampmlitud de la onda f(t).

k recorre el espacio de frecuencias (kf\_0).

X\_k significa cuánto se activa esa frecuencia, su amplitud h(kf\_0).

¿Cómo se ve esto en programación?

El x\_n es un producto punto con el exponencial

La implementación es sencilla.

Se activa en 3, 10. ¿Por qué en 30 y 37?

La discreta es muy pesada.

Por eso, se inventó la **FFT**.

Es la misma Discreta lenta, pero va dividiendo el espacio.

Los coeficientes c\_n más altos se mantienen, y los más bajos se eliminan (son menos importantes).

Semana 4 Viernes:

Fin de transformada de Fourier.

Problema con la transformada es que es muy pesada.

Con sonido es fácil, con imágenes es complicado.

FFT reduce la complejidad de O(n^2) a O(nlogn)

Aplicación: random Fourier Features (RFF): del 2010. Sirve métodos de Kernel y AI.

Solución de ecuaciones diferenciales.

En el curso veremos tanto ordinarias como parciales.

En las ordinarias se hace métodos de Runge-Kutta.

En parciales diferencias finitas.

**Ordinarias:**

1er orden:

movimiento acelerado con fricción del aire:

Crecimiento de población:

2do orden:

Resorte:

Péndulo:

Primero vemos de 1er orden:

Se puede pasar todo lo que no sea derivada a un lado.

**Regla de Euler:**

La derivada se puede aproximar como

Se escribe la derivada de forma numérica.

Similar a Newton-Raphson, pero no se manda hasta 0 sino a otro número.

El h es el delta, el avance.

Entonces la regla de Euler es

También está la **regla de Euler mejorada**:

Teorema fundamental del Cálculo

Otra formad del TFC es

El mejorado calcula la integral con el método del trapecio:

Se aproxima el n+1 con la regla de Euler.

Esto es como Runge-Kutta con trapecio. Hay mejores aproximaciones con Simpson.

El método de Euler mejorado es entonces:

**Implementación** en Python:

Se necesitan valores iniciales.

La próxima clase se ve Runge-Kutta, es un método de 4to orden que da una mejor aproximación. También hay de 8vo orden.

**Animaciones**

¿plot vs ax?

Al hacer plt.figure() se crea un canvas (marco general)

Al hacer un axis, se divide el general en secciones.

Figure es el canvas, el ax es cada gráfico dentro del figure.

Subplots(2) crea dos plots

Ax[0] y ax[1]

Subplots(2, 2) crea una matriz con gráficas.

Ahora animación:

Se crea una figura general y se le crea un eje (que se va graficando en el tiempo). Se van sobreponiendo imágenes.

Se usa la función FuncAnimation(fig, func=updatet, frames=range(100))

fig es el canvas general

func es una función generador (callable), iterador.

En el siguiente update va graficadndo el siguiente i. El i es un linspace del tiempo, que actualiza el plot. En el ejemplo es [:i] para ir graficando todos los puntos (no solo uno)

Frames indican cuántos frames se hacen.

Si se elige mucho tiempo, se va a demorar mucho la simulación. Recomendación: máximo 100 de frame.

Se debe agregar el rc al comienzo para que se genere la simulación directamente en el JupyterNotebook.

Semana 5 Miércoles:

**Ejercicio clase pasada**

**Tema: Runge-Kutta**

La idea es solucionar ecuaciones diferenciales

Método de Euler es una primera aproximación (moviéndose un )

Luego está el método de Euler mejorado

La idea de este es ir calculando la derivada y moverse a lo largo de ella.

Runge-Kutta es mejor que Euler mejorado pero sigue la misma idea.

Runge-Kutta 4to grado:

Se mueven cuatros veces en pasos de menos de h, pero en total se mueve h por cada paso.

[El de 8vo grado es aún mejor. El 16 aún mejor pero complejo computacionalmente.]

¿De dónde viene el h/6?

Se integra usando el método de Simpson.

¿De dónde viene Simpson?

Interpolación de Lagrande de orden 2.

Simspon hace esto:

Pero lo usamos distinto:

El k\_3 aprovecha la aproximación anterior para mejorar.

Para mejorar esto, se podría usar Simpson 3/8, que es interpolación 3er grado, pero es más costoso computacionalmente.

**Computacionalmente**

Los pesos son dados por Simpson.

**Problema tigre vs gacela**

Runge-Kutta se aproxima bastante bien.

**Introducción a ED de 2do orden**

Ahora hay segunda derivada

La idea es convertirla en dos ecuaciones de 1er orden. Para eso se define

Esas dos se resuelven con RK, y así se resuelve la inicial.

Semana 5 Viernes:

**Runge Kutta para ecuaciones de 2do orden**

Se convierte en un sistema de dos ecuaciones

Aunque una ecuación de segundo grado también depende de la derivada

Se necesitan condiciones iniciales

Aquí nos desplazamos a lo largo de y, v, t.

Se aplica la operación de Runge Kutta para evolucionar de forma vectorial.

Evolucionan vectorialmente.

**Ejemplo:**

Péndulo

La ligadura ya está en las ecuaciones

Se quiere resolver el sistema, el cual tiene 6 incógnitas. Entonces no se puede resolver. Entonces toca hacer fuerzas.

Cuando hagan la simulación, eliminar el video, porque queda muy pesado para BN (50MB). Podría ser enviar un video de YB.

Enviar solo el NB y el video de YB.

También se puede hacer de 3 cuerpos.

El sistema de ecuaciones es de 18 ecuaciones.

Hay 9 condiciones iniciales de posiciones y 9 velocidades iniciales.

Se cuadran constantes útiles K1 y K2.

No resolver las ecuaciones a mano, sino que usar Simpy. Es como Mathematica, para coger ecuaciones y las resuelva por uno. ¿Cómo usar Simpy? Con symbols.