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.

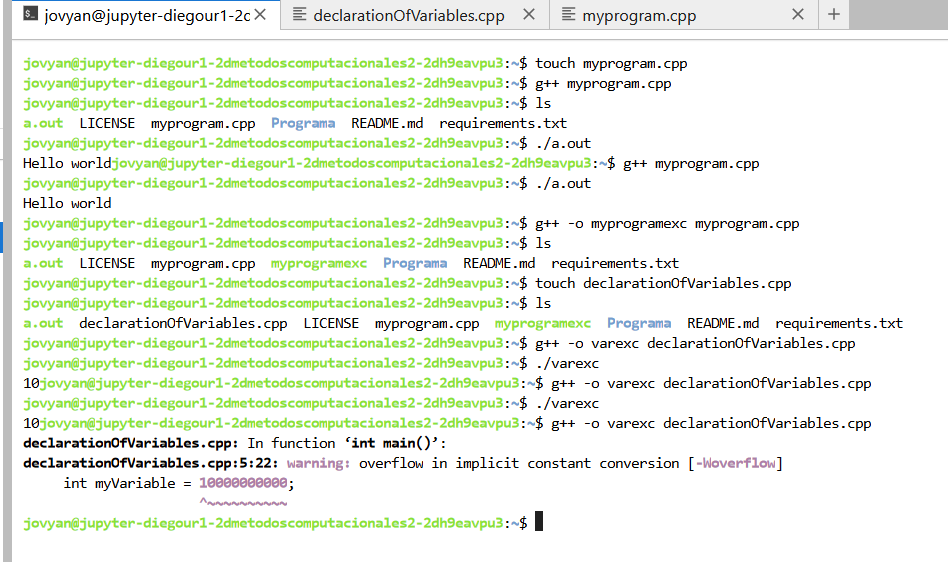
Al crear un objeto se llama al constructor automáticamente.

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++?

**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

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.

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

Se envían antes de que compile el archivo. Existen de manera más global.

Las variables del código existe 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

}

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

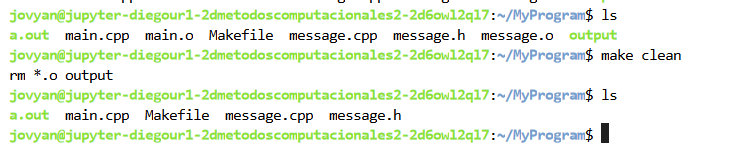
Descripción generada automáticamente

Pero quedan muchos archivos.

Entonces se usa clean

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente



**Ejercicio:**

// compllilar y ejecutar el cppp

// eso genra el xvalues

// y luego ejecuta python

// no hay que crear archivo python

// python nombrearhcivo.py