

Herramientas Computacionales para la Astroinformática

Cristian A. Vega-Martínez (oficina: IMIP, Académicos 2)
Facundo A. Gómez



Anuncio

- Se compartió un documento llamado Portafolios_del_curso.pdf, que describe cómo se debe crear y mantener el repositorio de estudio del curso, el cual se utilizará como instrumento de evaluación.

Fe de erratas

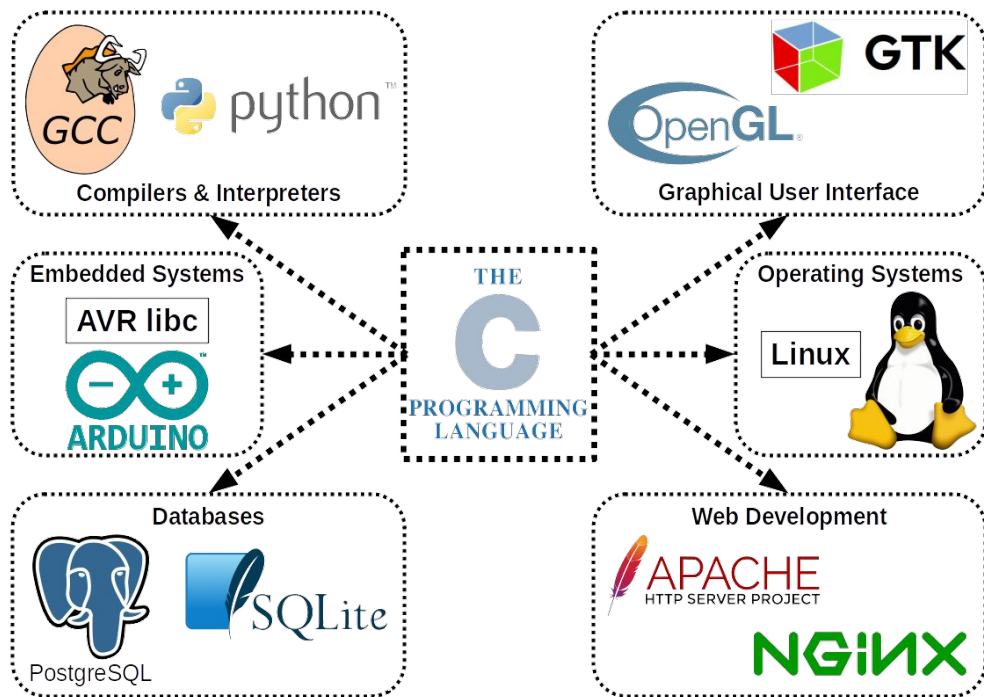
- En el documento “0 - Presentación e Introducción.pdf”, así como durante la clase correspondiente, se indicó un día **feriado** (27/oct) para las presentaciones mid-term. Por ello, las fechas de presentaciones serán:
 - **Martes 24 de octubre**
 - **Jueves 26 de octubre.**

Introducción al lenguaje C

Compilación, linkeo y librerías

Lenguaje C

C es un lenguaje de programación de propósito general, originalmente desarrollado por Dennis Ritchie entre 1969 y 1972 en los Laboratorios Bell.



Algunas características a destacar:

- Es ampliamente utilizado en desarrollo de sistemas operativos, compiladores, aplicaciones, etc.
- Usa tipos de datos estáticos.
- Dispone de las estructuras típicas de lenguajes de alto nivel. También dispone de construcciones del lenguaje que permiten un control a bajo nivel.
- Acceso a memoria de bajo nivel (punteros).
- Conjunto reducido de palabras clave.

Sintaxis básica

- La sintaxis del código es insensible a múltiples saltos de línea y espacios (indentación optativa).
- Cada instrucción se debe delimitar con un `;` al final.
- Cada conjunto de instrucciones debe encerrarse entre llaves `{ }`.
- El código debe organizarse en funciones, entre las cuales debe existir una `main`.
- Las instrucciones para el preprocesador son indicadas con `#`.
- Los comentarios se indican con `//` (para una línea) o encerrados entre `/*` y `*/` para múltiples líneas.

Ejemplo: Hola mundo más simple en C:

```
#include <stdio.h>

int main() {
    printf("Hola mundo");
    return 0;
}
```

Compilación con GCC

GCC (GNU Compiler Collection) es un **conjunto** de compiladores desarrollados por el proyecto GNU. Ha evolucionado para soportar **múltiples lenguajes** de programación, incluidos C, C++, Fortran, Ada y otros, aunque su principal uso es la compilación de C/C++. Es **software libre** y es **ampliamente utilizado** en sistemas basados en Unix, como Linux. Es **multiplataforma** y puede aplicar **optimizaciones** durante la compilación.

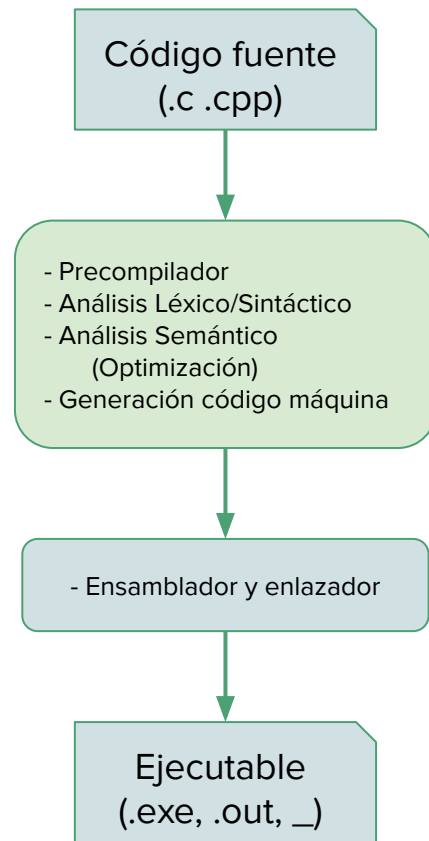
Compilación completa:

```
> gcc hola.c -o programa
```

Compilación con linker separado:

```
> gcc -c hola.c
```

```
> gcc -o programa hola.o
```



Compilación con GCC

Activación de Optimizaciones: se utiliza el argumento -O (más detalles: RTFM).

```
> gcc -O2 hola.c -o programa
```

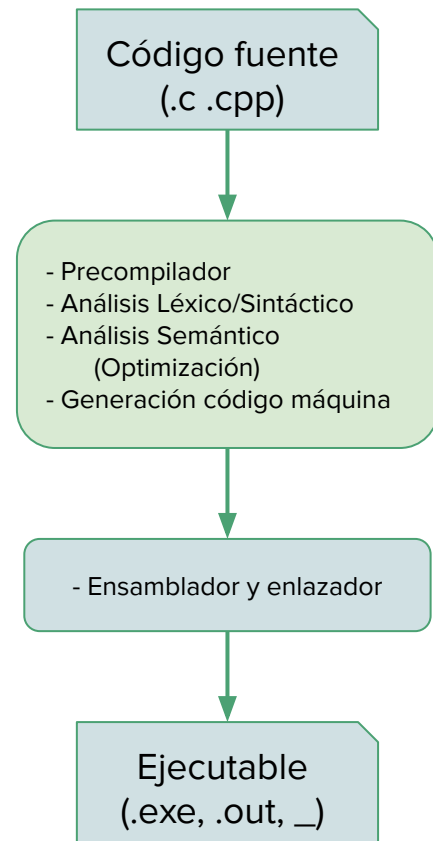
Mostrar **advertencias:** se pueden activar todas con -Wall
(“all” se puede reemplazar por advertencias específicas; -Wno... las desactiva)

```
> gcc -o programa -Wall hola.c
```

Incorporación de librerías: se utiliza el argumento -l (más el nombre de la lib).
Ejemplo: compilación con la librería matemática math.h:

```
> gcc -o programa hola.c -lm
```

Las librerías son buscadas en los directorios de sistema por su nombre (directorios **lib/**), más todos los que se le indiquen con -L. Esta opciones las utiliza el enlazador.



Compilación con GCC - Librerías

Las librerías incluyen un archivo **header** (**.h** o **.hpp**), escrito en C/C++, que se distribuye en código fuente, y contiene, declaraciones, definiciones, documentación, etc. Se suelen ubicar en los directorios **include/**, más los que se indiquen con **-I**.

Las librerías son de **dos tipos: estáticas y dinámicas**.

Librerías estáticas: son archivos compilados, en código objeto (extensión **.a**), que contienen las funciones y procedimientos de la librería. Estos son incorporados directamente en el archivo ejecutable durante el enlazado.

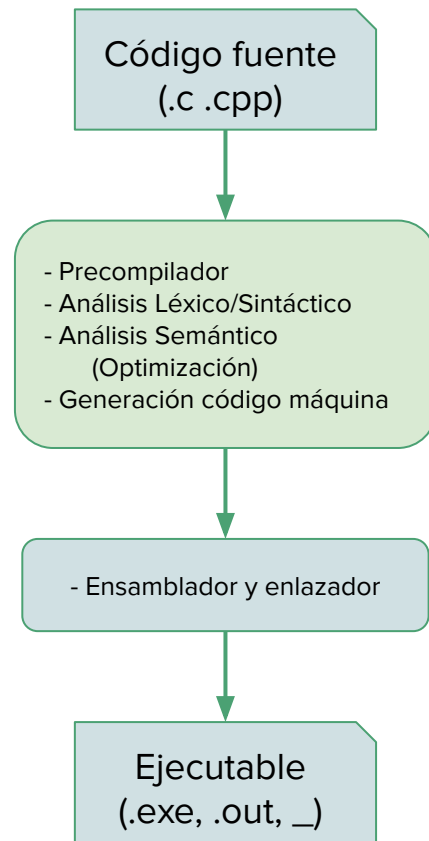
Permite crear ejecutables sin dependencias, pero de mayor tamaño.

Librerías dinámicas (shared): son archivos compilados, en código objeto (extensión **.so** o **.dll**), que contienen las funciones y procedimientos de la librería. Estos **no** son incorporados en el ejecutable, sino que **son llamadas (enlazadas) durante la ejecución**.

Crea ejecutables **dependientes**.

Las librerías se pueden actualizar sin recompilar el programa.

El sistema operativo usa la variable de entorno **LD_LIBRARY_PATH** para localizar librerías dinámicas adicionales.



Sintaxis de C: preprocesador

Instrucciones para el preprocesador se ejecutan antes de la revisión y compilación.

#include : Incorpora una librería (header) en la compilación.

Ejemplos: **#include <stdio.h>**,
#include "mylib.h"

#define : Definir un identificador (variable o macro) que es reemplazado por el preprocesador antes de compilar.

Ejemplo: **#define PI 3.14159**
#define CUADRADO(x) ((x) * (x))

Estos también se pueden definir al compilar con GCC usando -D.

#ifdef, #ifndef, #endif, #else : Permiten condicionar la compilación según identificadores definidos. Esto permite “(des)activar” secciones de código desde la compilación. Estas instrucciones no deberían usarse para regular flujo usual de código.

Sintaxis de C: **variables**

Las variables en C requieren ser **declaradas** (donde se especifica su tipo y reserva memoria) e **inicializadas** (donde se “llena” la memoria asignada). Ejemplo:

```
int var;    // declaración  
var = 0;    // inicialización
```

Esto también se puede realizar en una sola línea:

```
int var = 0;
```

Entre los tipos de datos básicos que se pueden utilizar en C encontramos:

```
int, (short, long, unsigned) int, float, double, char, FILE.
```

Las **strings** (arrays de char) y archivos (FILE) son tipos de datos *compuestos*, que deben ser manipulados por funciones específicas del lenguaje.

C además admite crear tipos de datos nuevos con **struct**.

Sintaxis de C: operadores

C admite **operadores aritméticos** (+, -, *, /) para operaciones entre variables, respetando las reglas de precedencia matemática. También permite uso de ().

Ejemplo:

```
int foo = 1 + 5 * 3;    // esto vale 16 y no 18.  
int bar = (1 + 5) * 3;  // esto si vale 18
```

Otras operaciones matemáticas requieren uso de librerías (como math.h).

También incorpora los **operadores lógicos**: and (&&), or (||), not (!), y operadores de comparación: igual (==), distinto (!=), mayor/menor/igual que (>, <, >=, <=) los cuales son utilizados en evaluación de condiciones.

Sintaxis de C: control de flujo

Se pueden aplicar condicionales en C con **if** y **else** de forma análoga a otros lenguajes, delimitando los campos con ; y { } cuando sea necesario.

Ejemplo:

```
int x = 1, y = 2;    // así declaramos varias a la vez
if (x <= y ) {       // { } innecesarias en instrucción única
    printf("x es mayor que y\n");
} else
    printf("y es mayor que x\n");
```

Sintaxis de C: control de flujo

Se pueden definir loops con **for**(*inicio*; *condición*; *paso*){...}. Ejemplo:

```
for (int i=0; i<10; i++) {  
    printf("El valor de i es: %d \n", i);  
}
```

También se puede usar **while**(*condición*){...} y **do** {...} **while**(*condición*); para crear loops solo indicando una condición. Ejemplo:

```
int i=0;  
while (i<10) {  
    printf("El valor de i es: %d \n", i++);  
}
```

Más Operadores

https://es.wikipedia.org/wiki/Anexo:Operadores_de_C_y_C%2B%2B

En C/C++ se introducen una variedad de operadores que simplifican sintaxis.

Por ejemplo, para dos variables **x** e **y** definidas correctamente:

Nombre Operador	Sintaxis	Significado
Suma y asignación	a += b	a = a + b
Resta y asignación	a -= b	a = a - b
Multiplicación y asignación	a *= b	a = a * b
División y asignación	a /= b	a = a / b
Incremento postfijo (prefijo)	a++ (++a)	a += 1
decremento postfijo (prefijo)	a-- (--a)	a -= 1
y más... (aritméticos, lógicos, de punteros, de membresía, de bits).		

Los operadores pueden ser: **unarios**, **binarios**, o **ternarios**, según el número de argumentos que requieran (1, 2 o 3, respectivamente).

Arrays y memoria estática.

C permite definir arrays de datos del mismo tipo de forma **estática**. Por ejemplo:

```
int iarray[N];           // iarray es el nombre de mi variable.
```

Con N un valor definido. Los arrays son variables estructuradas donde los elementos se almacenan **de forma consecutiva** en la memoria.

Para inicializar un array al momento de su declaración se usan { }. Por ejemplo:

```
int iarray[5] = {0, 1, 2, 3, 4};           // El 5 puede omitirse acá
```

En cambio, para uno ya declarado, es necesario usar un loop:

```
for (int i = 0; i<5; i++) iarray[i] = i;
```

El operador de indexación [] es análogo al de Python, **parte desde cero**.

Arrays y memoria estática.

Arrays multidimensionales se deben definir utilizando índices múltiples. Por ejemplo:

```
int imatrix[N][M];    // N, M >=1
```

También se pueden inicializar tanto en su declaración

```
int imatrix[2][3] = {{0, 1, 2},  
                    {1, 2, 3}} // indentado para comprensión
```

o utilizando un doble loop:

```
for (i=0; i<2; i++)  
    for (j=0; j<3; j++)  
        imatrix[i][j] = i + j; // ejemplo de arriba
```

Antes de C99 **no se admitía el uso de variables** para declarar los tamaños de arrays, sino que sólo constantes (conocidas al compilar). Hay compiladores que aún aplican esta restricción.

Punteros y memoria dinámica

Una característica central de C es la capacidad de conocer/acceder explícitamente a la memoria donde se almacenan los datos y variables durante la ejecución.

Puntero: una variable que almacena la **dirección de memoria** de otra variable.

- Tienen similitud con el concepto de **referencia** de Python.
- Se usan los operadores `*` y `&` para (des)referenciar punteros. Ejemplo:

```
int var;    // Crea una variable var
int *pvar;  // Crea un puntero a un int
            // (pvar: puntero, *pvar: contenido)
pvar = &var; // pvar ahora apunta a var
scanf("%d \n", &var); // guarda stdin en var
```

Crear un puntero solo asigna la memoria del mismo, **no** de su contenido.

Punteros y memoria dinámica

Usando punteros se pueden definir array con memoria dinámica. Ejemplo:

```
int *counter;  
int Nele = 5;  
    // malloc reserva memoria:  
counter = malloc(Nele*sizeof(int));  
    // Ahora counter[*] se puede usar igual a un array estático:  
for (int i=0; i<Nele; i++)  
    counter[i] = i;  
    // Luego de usar, la memoria dinámica se debe liberar:  
free(counter);
```

Punteros y memoria dinámica

Utilización de punteros

- **Acceso directo a memoria**
Arrays de memoria dinámica.
Las variables tipo punteros también admiten operadores para moverse dentro de la memoria.
- **Paso por referencia**
Se pueden utilizar como argumentos y retornos de funciones.
- **Estructuras de datos dinámicas**
Útiles en la creación de objetos complejos como listas enlazadas, árboles, grafos.

Complicaciones comunes

- **Punteros no inicializados**
Tal como las variables, sin inicializar contienen información aleatoria de la memoria (apuntar a cualquier parte).
- **Desreferenciación de punteros nulos**
Intentar acceder a un valor a través de un puntero nulo resultará en Segmentation Fault.
- **Fugas de Memoria:**
Uso de arrays con memoria dinámica no liberados adecuadamente puede resultar en fugas de memoria.

Algunos recursos online

Libro de programación en C.

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C

Tutorial: librerías de C

http://www.tutorialspoint.com/c_standard_library/index.htm

Tutorial C++

<https://cplusplus.com/doc/tutorial/>

Colección de tutoriales gratuitos:

<https://www.tutorialspoint.com/tutorialslibrary.htm>

Tema para profundizar:

Make (Makefile)

¿Qué es?

¿Para qué sirve?

¿Cómo se usa?

Veremos ejemplos de Makefiles en la próxima clase.