Programación en lenguaje C

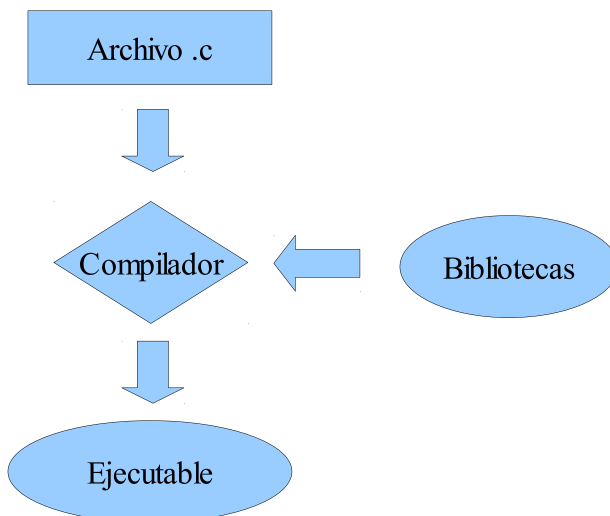
1 Introducción

Lenguaje de programación creado en 1972 por Dennis Ritchie, es la evolución del lenguaje de programación B.

Linux, kernel del sistema operativo GNU/Linux, esta escrito principalmente en el lenguaje de programación C.

C produce código más eficiente y veloz, ya que accede directamente al hardware de la máquina sobre la que corre y se compila el software para el sistema operativo específico en el que se ejecutará. Esto en contra parte con lo que hacen otros lenguajes, como Java, que se ejecutan sobre una máquina virtual, son más portables pero menos eficientes al tener la JVM entremedio.

Básicamente el proceso de compilación es el siguiente:



2 Algunas características del lenguaje

1. Permite realizar tareas de bajo nivel.
2. Lenguaje de pre-procesado (macros, múltiples archivos fuente).
3. Utilización de punteros.
4. Tipos de datos establecidos.
5. Definición de nuevos tipos de datos.
6. Programación estructurada (secuencia, selección e iteración).

3 Estructura de un programa en C

Inclusiones   #include <stdio.h>

Definiciones   #define PI 3.14

Declaraciones   # int h = 5;

Sentencias

```
int main() {
        int a;
        return 0; // ¿porque el retorno es 0?
}
```

Para compilar: gcc -Wall programa.c -o programa

Algunas convenciones:

1. Variables en DEFINE son en mayúsculas.
2. Una variable parte con minúsculas y cada palabra nueva de la variable inicia en mayúsculas (esto aplica también para las funciones).

4 Tipos de datos

4.1 Enteros

Tipo dato	Largo	Rango
short int	2 bytes	Signed: -32,768 to 32,767 unsigned: 0 to 65,535
int	4 bytes	Signed: -2,147,483,648 to 2,147,483,647 unsigned: 0 to 4,294,967,295
long	8 bytes	Signed: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 unsigned: 0 to 18,446,744,073,709,551,615

4.2 Carácter

Tipo dato	Largo	Rango
char	1 byte	signed: -128 to 127 unsigned: 0 to 255

4.3 Punto flotante

Tipo dato	Largo	Rango
float	4 bytes	+/- 3.4e +/- 38 (~7 digits)
double	8 bytes	+/- 1.7e +/- 308 (~15 digits)

4.4 String

En C el concepto de string no existe, sin embargo se puede utilizar un arreglo de caracteres para tales efectos.

char palabra[] = “hola”; => HOLA\0

\0 corresponde al fin del string, los siguientes casos son diferentes para efectos de como será utilizado el string:

char otra[256] = “hola”; es distinto a char otra2[256] = “hola\0”;

Hay que agregar manualmente el \0 para que no se siga leyendo el arreglo una vez se termino la palabra, además el largo del arreglo debe considerar la casilla para guardar este fin (1 byte).

5 Ámbito de variables

Al programar es muy importante considerar el ámbito de las variables, esto es donde será cada una visible y a cual se afectará si se realiza una modificación, se presenta un ejemplo donde se declara la misma variables diferentes veces para ejemplificar esto.

Ej:

```
int a;                  // variables global (1)
int funcion (int a) {   // parámetro de la función (2)
    return a;           // se usa (2)
}
int funcion2 () {
    int b;              // variable local (3)
    return a;           // se usa (1)
}
void main () {
    int a;              // variable local (4)
    a = b;              // error porque se usa (3) que es local de funcion2
    if(a) {
        int a = 2;      // variable local del bloque (5) distinta a (4)
    }
}
```

Se recomienda:

- No utilizar variables globales.
- Utilizar nombres descriptivos para las mismas.
- Si hay variables que tienen el mismo nombre que sea en contextos diferentes.

6 Operadores

6.1 Estándar

+ Suma
- Resta
/ División
* Multiplicación
% Módulo

6.2 Relacionales

> Mayor que
< Menor que
>= Mayor o igual que
<= Menor o igual que
== Igual a
!= Distinto a

6.3 Unarios

-- Decrementar en 1
++ Incrementar en 1

Ejemplo:

```
int a = 0, b;
b = a++;   // b=0, a=1
b= ++a;    // b=2, a=2
```

6.4 Lógicos

&& and
|| or
! not

6.5 Binarios

&	and
|	or
^	xor
~	complemento a uno
-	negación
<<	desplazamiento a la izquierda
>>	desplazamiento a la derecha

7 Conversión de datos o casting

Consiste en forzar a la conversión de un tipo de datos en específico.

Ejemplo:

```
int a = 92;
char b;
b = (char) a;
```

Esto será utilizado más adelante cuando se solicite memoria de forma dinámica y se indique con que propósito dicha memoria será utilizada.

8 Estructuras de control

```
if(condición) {
	// código si es verdadero
} else {
	// código si es falso (opcional)
}
```

Notación resumida para asignaciones:

```
int a = condición ? valor si es verdadero : valor si es falso;
```

```
switch (b) {
	case '1': {
		// código caso que sea 1
		break;
	}
	// mas opciones
	default: {
		// opción por defecto (opcional)
	}
}
```

Un switch es básicamente una serie de if anidados, es mas limpio visualmente esto, sin embargo se estima que puede ser menos óptimo.

9 Repetidores

```
while (condicion) {
	// código a repetir
}

do {
	// código a ejecutar al menos una vez
} while (condición) {
	// código a ejecutar si es verdadera la condición (opcional)
}

for (declaración; control; incremento) {
	// código a repetir
}
```

10 Comentarios

```
// comentario una línea
/* comentario
varias línea */
```

10.1 Comentar funciones

Es muy importante comentar el código fuente que uno desarrolla, alrededor de un 30% de lo escrito debieran ser comentarios. Al menos debemos comentar nuestras funciones, se recomienda:

```
/**
 * Nombre y que hace
 * @param
 * @return
 * @warning
 * @todo
 * @author
 * @date
 */
```

Con lo anterior si otro desarrollador toma el código fuente sabrá de que se trata el código sin tener que leerlo directamente.

10.2 Comentar en pre-procesado

Otra forma de comentar es antes que se genere el ejecutable final:

```
#IF (0)
// código
#END ID
```

Esto es útil al hacer debug a un programa pues podemos comentar código que ya posee comentarios.

11 Funciones

Una función permitirá agrupar código que deseamos utilizar en diversas oportunidades, otra utilidad es para encapsular código y entregarlo a otro desarrollador (o nosotros mismos) donde uno solo se preocupa de los parámetros de entrada y el retorno pero no de la lógica que hay entre medio.

Una función se define de la siguiente forma:

```
tipo_de_dato nombre (parámetros) {
	// código de la función
	return valor; // en caso que el tipo de dato de la función no sea void
}
```

Una función con tipo de dato igual a void significará que no devuelve ningún tipo de dato.

Funciones se pueden definir antes del main, si se hace después se deberá colocar el prototipo de la función antes del main para que el compilador sepa que la función existe (en alguna parte).

Ejemplo:

```
	int calcula (int, int);
	int main () {
		int dif = calcula (5, 6);
		return 0;
	}
	int calcula (int a, int b) {
		return (a-b);
	}
```

11.1 printf()

Permite mostrar texto por la salida estándar. Su uso básico es:

```
printf("hola");
```

Para mostrar variables se debe utilizar una sintaxis especial, ejemplos:

```
printf("%d", entero);    // imprime un valor entero
printf("%s", string);    // imprime un string, arreglo de char
printf("%f", real);      // imprime un float o double
```

Si queremos introducir caracteres especiales podemos utilizar:

\n Salto de línea

\t Tabulador

12 Punteros

Una diferencia grande de C con otros lenguajes (como Java o PHP) es el soporte para utilizar punteros. Un puntero corresponde a una variable que en su contenido tiene la dirección de memoria hacia donde esta apuntando, en estricto rigor no contiene el valor sino la forma de acceder a él. Esto será útil para pasar parámetros por referencia a funciones y poder retornar más de un elemento, o por ejemplo cuando queramos recorrer elementos en estructuras de datos como listas o colas. Por lo cual identificaremos la dirección de memoria, el puntero y el valor como elementos diferentes.

Ejemplo:

```
int a = 5;              // memoria: 0x001, valor = 5
int *b = null;          // valor = null
b = &a;                 // valor = 0x001
printf("%d\n", b);      // imprime 0x001
printf("%d\n",*b);      // imprime 5
```

Los punteros permitirán a funciones devolver más de un valor mediante modificaciones que se pueden hacer a variables dentro la función y ser retornadas mediante variables que se pasaron por referencia. Además una función también puede devolver un punteros.

Ejemplos:

```
int funcion (int *a, int *b);   // recibe a y b por referencia
int *f (int a, int b);          // devuelve puntero a un entero
int (*f) (int a, int b);        // puntero a una función que devuelve un
                                // entero
```

13 Ingresar datos mediante parámetros al programa

Se permite el ingreso de parámetros mediante:

```
./programa par1 par2 par3 … parN
```

Para poder utilizar estos parámetros se reciben en la función main de la siguiente forma:

```
int main (int argc, char *argv[]) {
        // codigo
}
```

argc contiene la cantidad de parámetros pasados al programa más uno, y argv es un arreglo de punteros a carácter con los parámetros pasados. Notar que los valores en argv son de tipo carácter por lo cual se deberán convertir a otro tipo de datos en caso que se requiera.

14 scanf

Permite ingresar texto por la entrada estándar. Su uso básico es:

```
scanf("%s", string);
```

Notar que scanf requiere como variable(s) la dirección de memoria donde se colocará el dato leído desde la entrada estándar, en el caso de los string (arreglo de caracteres) el nombre del arreglo (sin llaves) es un puntero, en el caso de otras variables se debe indicar la dirección de memoria utilizando & previo a la variable, ejemplos:

```
scanf("%d", &entero);  // lee un valor entero
```

Se puede leer más de un parámetro de la siguiente forma:

```
scanf("%s %d", string, &entero);   // lee un string y un valor entero
```

15 Leer y escribir archivos

Para leer y escribir ficheros en archivos existen diferentes funciones, en general estás parten con una eltra "f" y el tipo de dato a utilizar es FILE*.

Ej:

```
// leer archivo carácter a carácter
FILE *archivo;
archivo = fopen("archivo.txt", "r");
while(!feof(archivo)) printf("%c", fgetc(archivo));
fclose(archivo);
```

Otra alternativa es leer (o escribir) grupos de datos utilizando fscanf y fprintf

Ej:

```
fscanf(in, "%d %s %s", &run, nombre, apellido);
fprintf(out, "%d %s %s\n", run, nombre, apellido);
```

16 Estructuras de datos

C permite definir nuevas estructuras de datos mediante la utilización de la instrucción struct.

Ej:

```
struct alumno {
    int run;
    char nombre[100];
    char apellido[100];
};
struct alumno a; // a es una estructura de datos de tipo alumno
```

Para evitar tener que utilizar la palabra struct cada vez podemos definir un nuevo tipo de datos y luego usarlo cada vez que lo necesitemos.

Ej:

```
typedef struct alumno Alumno;
Alumno a; // a es una estructura de datos de tipo alumno
```

Otra forma de declarar la estructura y crear el nuevo tipo de datos es de la siguiente forma:

```
typedef struct alumno {
    int run;
    char nombre[100];
    char apellido[100];
} Alumno;
```

Cuando queramos utilizar la estructura lo hacemos accediendo a sus atributos mediante el punto:

Ej:

```
a.run = 1; strcpy(a.nombre, "Nombre"); strcpy(a.apellido, "Apellido");
```

16.1 Asignación dinámica de memoria

Es posible pedir memoria de forma dinámica utilizando calloc o malloc, la forma de utilización es la siguiente:

alumnos = (Alumno) calloc (10, sizeof(Alumno));

alumnos = (Alumno) malloc (sizeof(Alumno)*10);

La diferencia principal entre calloc y malloc es que el segundo no limpia la memoria cuando es asignada en cambio el primero si lo hace asignando valor null a la misma.

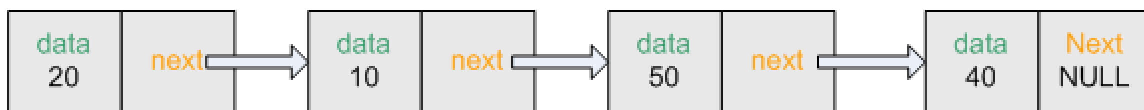
Si los datos son creados de la forma anterior ahora deben ser utilizados como punteros, ya no usando punto, sino → esto se hace de la siguiente forma (pensando en el ejemplo anterior):

alumnos[0]->run = 1;

El problema sin embargo sigue siendo que se debe conocer de forma a priori el tamaño del arreglo ya sea porque se declarará como tal o se indicará el tamaño (en el ejemplo 10) al solicitar el espacio.

16.2 Estructuras de datos básicas

Dentro de las estructuras de datos más estudiadas se encuentran las colas, listas y pilas las cuales se explicarán a continuación. La característica fundamental a utilizar en su implementación será que son de tamaño variable, esto ya que las estructuras de datos que conformar cada una de las colas, lista o pilas será solicitada de forma dinámica al sistema (usando malloc) y mediante punteros se crearán los enlaces entre las estructuras.



Linked list

Ilustración 1: Lista simplemente enlazada

Lo anterior se consigue con un nodo que es la misma estructura anterior pero con un nuevo campo que es un puntero a una estructura del mismo tipo, de esta forma se puede apuntar a la estructura siguiente en la lista y tenerla "distribuida" por la memoria, el apuntar a NULL indica el fin de la lista.

Algunos conceptos

- Nodo: estructura de dato parte de la lista (cola o pila) que contiene los punteros al siguiente nodo.
- Simplemente enlazada: se guarda un puntero con la ubicación del siguiente nodo.
- Doblemente enlazada: se guarda un puntero con la ubicación del nodo anterior y el siguiente.
- Cabeza: nodo inicial de la lista.
- Lista circular: lista que su último nodo apunta al primero.

Colas

Una cola es un conjunto de estructuras donde los nodos entran al final de la cola y son retirados desde el inicio.

Listas

Una lista es similar a una cola, sin embargo los elementos pueden ser colocados y sacados de la lista en cualquier ubicación (inicio, fin o en una posición específica).

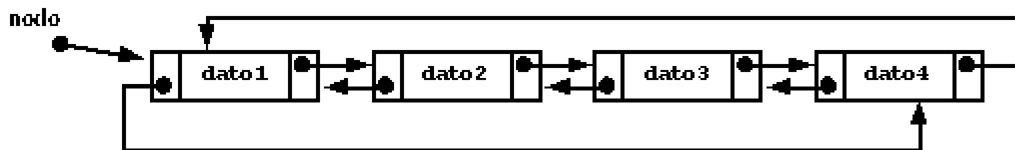


Ilustración 2: Lista doblemente enlazada circular

Como se mencionó anteriormente la principal ventaja de utilizar una lista versus un arreglo es que estas tienen tamaño dinámico y permiten fácilmente ingresar un nuevo nodo en cualquier parte de la misma.

Pilas

Una pila corresponde a una estructura donde los nodos son colocados y sacados en el mismo extremo, esto significa que si coloco en una pila A, B, C, D y ahora quiero sacar los elementos estoy obligado a hacerlo en el orden D, C, B, A. El método push permitirá agregar un elemento a la pila y el método pop sacar uno de la misma.

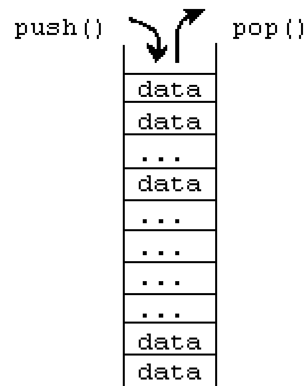


Ilustración 3: Pila con métodos pop y push

Ejemplo de lista

```
struct nodo {
	int id;
	struct nodo *siguiente;
};
struct nodo *lista = (struct nodo*) malloc(sizeof(struct nodo));
struct nodo *nodo2 = (struct nodo *) malloc(sizeof(struct nodo));
lista->siguiente = nodo2;
lista->id = 1;
lista->siguiente->id = 2;
```

17 Operaciones binarias

En C es posible realizar diversas operaciones con números binarios, revisar punto 5.5 para ver operadores.

Ej:

```
a = 1;      // 0 0 0 0 0 0 0 1
b = 2 ;     // 0 0 0 0 0 0 1 0
c = 5;      // 0 0 0 0 0 1 0 1
x = a|b;    // 0 0 0 0 0 0 1 1
x = a&c;    // 0 0 0 0 0 0 0 1
x = a^c;    // 0 0 0 0 0 1 0 0
x = c<<1;   // 0 0 0 0 1 0 1 0
```

Se pueden utilizar máscaras para encontrar ciertos bits.

Ej:

```
// encontrar unos
x = 10;     // 0 0 0 0 1 0 1 0
m = 1;      // 0 0 0 0 0 0 0 1
for(i=0; i<8; ++i) {
	if(x&m) ++unos;
	x>>1;
}
```

18 Macros

Ej:

```
#include <stdio.h>
#define PI 3.14
#define AREA_CIRCULO(x) PI*(x)*(x)
// es necesario utilizar (x) para aquellas variables de la macro
int main (void) {
	printf("Areá circulo radio 3: %f", AREA_CIRCULO(3));
	return 0;
}
```

19 Recursividad

La recursividad implica el llamado a una función desde la misma función. No se recomienda su uso en programas de alto desempeño, básicamente por su consumo de pila/stack de memoria.

Estructura:

```
funcion a () {
	condición de termino, return
	operaciones
	llamada recursiva
	operaciones (opcional)
}
```

Ej:

```
int factorial (int n) {
	if(n==2) return n;
	return n*factorial(n-1);
}
```

20 Bibliotecas .h

El uso de bibliotecas permitirá separar el código de nuestros programas, compartir funcionalidades, disminuir el tamaño de nuestro fuente donde se encuentra el main y re compilar solo lo que sea necesario.

Al programar se debe recordar que:

- Mantener archivos pequeños.
- main() de no más de 25 líneas.
- Funciones realizan solo una operación

Pensando en lo anterior las bibliotecas serán de gran utilidad, ya se han visto bibliotecas como la stdio.h o la stdlib.h, estás son propias del lenguaje en esta oportunidad se mostrará como crear una biblioteca propia.

Ej:

main.c

```
#include “mio.h”
int main (void) {
        hola();
        chao();
        return 0;
}
```

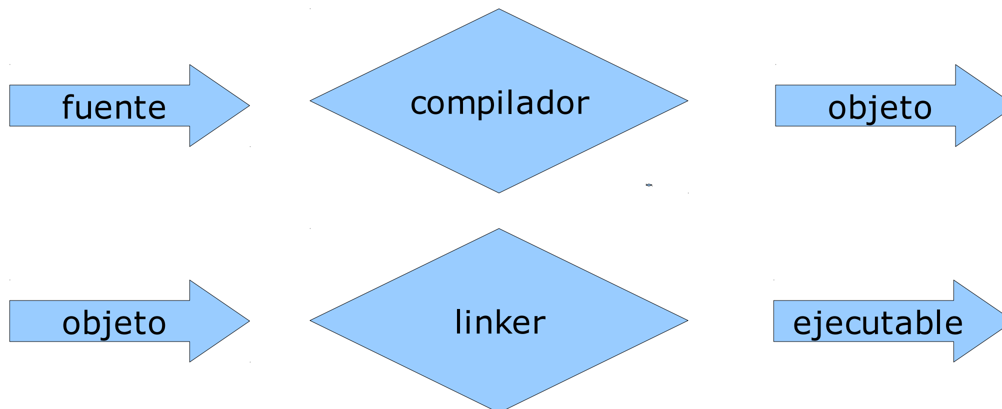
mio.h

```
#IFNDEF __MIO_H__
#define __MIO_H__
// se colocan prototipos de las funciones
void hola();
void chao();
#ENDIF
// dentro de los .h además de prototipos pueden ir estructuras de datos,
// definiciones, macros, variables globales (no recomendadas), etc.
```

mio.c

```
#include "mio.h"
#include <stdio.h>
// cuerpo de las funciones
void hola () {
	printf("Hola\n");
}
void chao () {
	printf("Chao\n");
}
```

20.1 Compilación



Ej:

```
gcc -Wall -c  mio.c -o mio.o
gcc -Wall -c main.c -o main.o
gcc -Wall mio.o main.o -o ejecutable
```

21 Makefile

Simplifica la tarea de compilación, pues agrupa los pasos de compilar en una menor cantidad de comandos a ejecutar.

La estructura de un archivo Makefile es la siguiente (utilizando el ejemplo anterior de bibliotecas):

```
CC="gcc"
FLAG="-Wall"

all: mio main
	$(CC) $(FLAG) mio.o main.o -o biblioteca

mio:
	$(CC) $(FLAG) -c mio.c -o mio.o

main:
	$(CC) $(FLAG) -c main.c -o main.o

clean:
	rm -f mio.o main.o biblioteca
```

“mio:” y “main:” son la acción a realizar, por defecto make busca all, el cual necesita de mio y main previamente.

Si se utilizara “mio.o: o “main.o:” esto automáticamente buscaría los fuentes .c con el mismo nombre y los compilaría.

22 Softwares de apoyo a la programación

A continuación se mencionan 3 software que pueden colaborar con el desarrollo de sus aplicaciones (información de cada software extraída desde Wikipedia).

22.1 GDB

GDB o GNU Debugger es el depurador estándar para el sistema operativo GNU. Ofrece la posibilidad de trazar y modificar la ejecución de un programa. El usuario puede controlar y alterar los valores de las variables internas del programa. Funciona en modo consola pero existe (entre otras) la interfaz gráfica DDD (Data Display Debugger).

22.2 Doxygen

Doxygen es un generador de documentación para C y C++ entre otros lenguajes. Su nombre es un acrónimo de dox(document) gen(generator), generador de documentación para código fuente.

Ej:

```
doxygen -g config // crea configuración inicial en el archivo config
// editar config según necesidades de documentación
doxygen config
```

22.3 Subversion

Subversion es un sistema de control de versiones diseñado específicamente para reemplazar al popular CVS. Se puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras.

A cierto nivel, la posibilidad de que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada, "si se ha hecho un cambio incorrecto a los datos simplemente deshaga ese cambio".

23 Ejercicios

Los siguientes son diferentes ejercicios que ayudarán a practicar lo visto en los distintos puntos de este apunte.

1. Cree un archivo holamundo.c que imprima un "Hola mundo". ¿Por qué debo incluir stdio.h?
2. Cree un programa que guarde un valor entero y traté de guardar el número 18,000,000,000. ¿Qué sucede?
3. Cree un programa que genere un string de largo 256 y guarde una palabra sin fin de string, imprímala seguida de un guión ¿qué sucede?
4. Cree un programa que implemente una función para calcular la potencia de un número, prototipo: int exponencial (int base, int exponente);
5. Cree un programa que implemente una función que copie un string en otro.
6. Cree un programa que reciba un número como parámetro y mediante una función lo convierta a entero.
7. Crear un programa que cargue en una lista los datos de un archivo de texto con la estructura "%d %s %s" (ejemplo run, nombre y apellido), luego muestre un menú que permita: agregar un nodo a la lista, borrar un nodo X de la lista, mostrar la lista, guardar la lista en un archivo (con la misma estructura) y salir del programa. Usar makefile y bibliotecas .h
8. Escribir un programa al que se le ingrese por parámetro una dirección IP y su máscara de red y muestre por pantalla la red a la que pertenece dicha IP. Usar makefile y bibliotecas .h