Universidad de Buenos Aires
Facultad De Ingeniería
Año 2018 - 1.er Cuatrimestre

Algoritmos y Programación I (95.11)

TRABAJO PRACTICO N.º 2 - Indexación automática de archivos MP3
FECHA: 04/07/2018
ALUMNOS:
MORA, Tomás Andrés - #99420
<tom.mora96@gmail.com>
LANÚS, Facundo - #99422
<facundolanus@gmail.com>

Índice

TRABAJO PRÁCTICO N.º 2 – Indexación automática de archivos MP3

1) Objetivo

El objetivo de este T.P. consiste en desarrollar un aplicativo de consola con comandos en línea de órdenes y escrito en lenguaje ANSI C, que permita construir un índice de los archivos MP3 indicados como parámetro, ordenado de acuerdo con un criterio particular y presentado en formato texto (CSV, HTML, XML).

2) Alcance

Mediante el presente T.P. se busca que el alumno adquiera y aplique conocimientos sobre los siguientes temas:

- programas en modo consola;
- argumentos en línea de órdenes (*CLA*);
- modularización;
- makefile;
- archivos de texto y binarios;
- memoria dinámica;
- punteros a función;
- tipo de dato abstracto, T.D.A. Vector. T.D.A. *ad hoc;*
- estructura del encabezado de un archivo MP3 (*ID3v1);*
- métodos de ordenamiento;
- estructura básica de un archivo CSV;
- estructura básica de un documento XML;
- estructura básica de un documento HTML (optativo).

3) Desarrollo

En este T.P. se pide escribir un programa ejecutable que genere documentación sobre pistas de audio en formato MP3.

El programa ejecutable, denominado "`mp3explorer.exe`" (WinXX) o "`mp3explorer`" (Unix), debe ser invocado de la siguiente forma:

WinXX:
```
mp3explorer –fmt <formato> –sort <criterio> -out <salida> <arch 1>…<arch N>
```

UNIX:
```
./mp3explorer –fmt <formato> –sort <criterio> -out <salida> <arch 1>…<arch N>
```

Los comandos en línea de órdenes utilizados por la aplicación son los indicados a continuación.

Formato del índice a generar

Comando	Descripción	Valor	Tipo de dato	Observaciones
-fmt	Formato del índice a generar	"csv"	Cadena de caracteres	Obligatorio
		"xml"	Cadena de caracteres	Obligatorio
		"html"	Cadena de caracteres	Optativo

Criterio de ordenamiento para los temas

Comando	Descripción	Valor	Tipo de dato	Observaciones
-sort	Criterio de ordenamiento	“name”	Cadena de caracteres	Nombre del tema (obra).
		“artist”	Cadena de caracteres	Autor de la obra.
		“genre”	Cadena de caracteres	Género de la obra.

Si se ordena, por ejemplo, por género, no se pide aplicar un nuevo criterio para todos los elementos con el mismo género.

Archivo de salida <salida>

Es la ruta del archivo índice que se desea construir.

Archivos de entrada <arch j>

Se trata del archivo MP3 j-ésimo de entrada.

Nota: Se puede asumir que los comandos en línea de órdenes estarán en cualquier orden (en pares), si esta estrategia simplifica el desarrollo de la presente aplicación, pero se debe consignar la decisión en el informe.

Operación

El programa debe analizar los archivos MP3 suministrados al programa y generar una tabla en memoria con los atributos de cada tema, para su posterior ordenamiento e impresión del índice en el formato correspondiente.

Formato de entrada

Se debe extraer la información de cada tema a partir del *frame header* del archivo MP3 mediante la lectura de los últimos 128 bytes del archivo. Se sugiere utilizar la función de biblioteca `fseek`(). Se debe trabajar con archivos ID3v1**.**

Formato de salida

a) Formato CSV:

El formato del documento CSV de salida a generar es:

```
<nombre de tema>|<artista>|<género>
…
<nombre de tema>|<artista>|<género>
```

Se trata de un archivo CSV sin encabezado, campos sin calificador y carácter separador “|”.

b) Formato XML:

El formato del documento XML de salida a generar es:

```
<?xml version="1.0" ?>
<tracks>
    <track>
        <name>...</name>
        <artist>...</artist>
        <genre>...</genre>
    </track>
        ...
```

```
        ...
</tracks>
```

Nota: A los efectos de este T.P. se permite obviar la transformación de juegos de caracteres y entidades para un documento XML.

c) Formato HTML (optativo):

Formato HTML tabular libre, a elección del desarrollador.

4) Restricciones

La realización de este programa está sujeta a las siguientes restricciones:

- Se debe recurrir al uso del T.D.A. Vector para almacenar los elementos de información correspondientes a una pista de audio.
- Se debe recurrir al uso de punteros a función a fin de parametrizar la impresión del índice.
- Se deben utilizar funciones y una adecuada modularización.
- Se debe construir un proyecto mediante la utilización de *makefile*.
- Hay otras cuestiones que no han sido especificadas intencionalmente en este requerimiento, para darle al desarrollador la libertad de elegir implementaciones que, según su criterio, resulten más favorables en determinadas situaciones. Por lo tanto, se debe explicitar cada una de las decisiones adoptadas y el o los fundamentos considerados para ellas.

5) Entrega del Trabajo Práctico

Se debe presentar la correspondiente documentación de desarrollo. Ella se debe entregar en forma impresa y encarpetada, de acuerdo con la numeración siguiente:

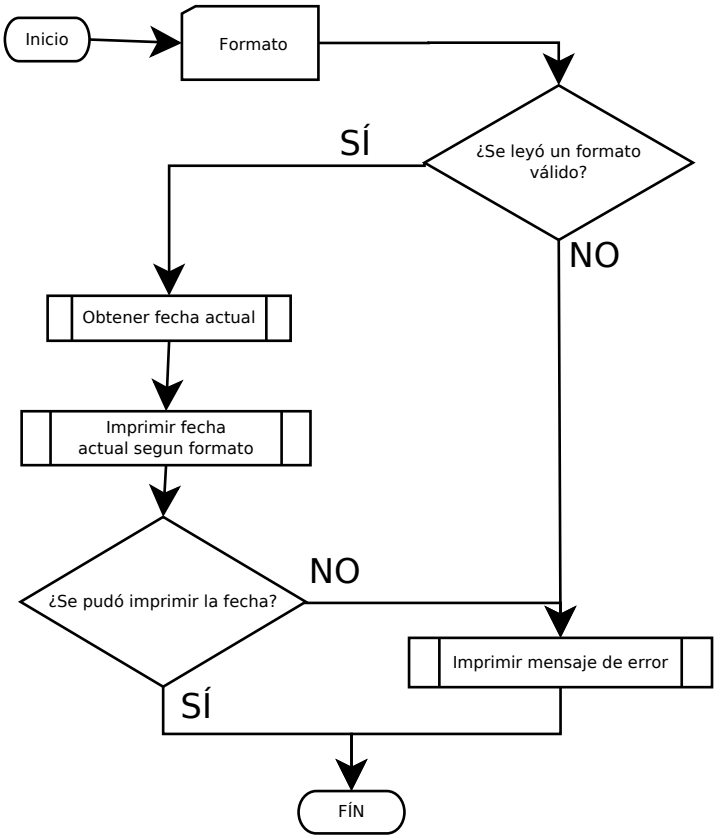
1) Carátula. Incluir una dirección de correo electrónico.
2) Enunciado (el presente documento).
3) Diagrama de flujo básico y simplificado del programa (1 carilla A4).
4) Estructura funcional del programa desarrollado (diagrama de funciones, 1 carilla A4).
5) Explicación de las alternativas consideradas y las estrategias adoptadas.
6) Resultados de la ejecución (corridas) del programa, en condiciones normales e inesperadas de entrada.
7) Reseña de los problemas encontrados en el desarrollo del programa y las soluciones implementadas para subsanarlos.
8) Conclusiones.
9) *Script* de compilación.
10) Código fuente.

6) Bibliografía

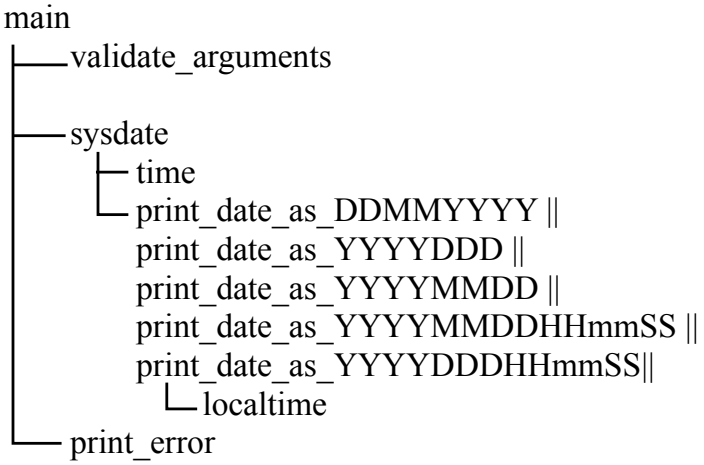
Se debe incluir la referencia a toda bibliografía consultada para la realización del trabajo: libros, artículos, etc.

7) Última fecha de entrega: sábado 30/06/2018

2. Estructura lógica



3. Estructura funcional



4. Consideraciones y estrategias

Una de las primeras consideraciones tenidas en cuenta fue la organización de los argumentos de línea de comandos. Se consideró la posibilidad de tener un orden fijo de argumentos o no. Al advertir que ya se había implementado una solución parecida en un trabajo anterior y que ésta permite la asignación de los argumentos a una estructura de configuración mediante macros, se implementó un orden fijo de argumentos.

Para el ordenamiento del vector de pistas, se utilizó la función de biblioteca estándar *qsort()*, dado que se consideró más práctico utilizar una función ya implementada que codificar una propia. Además, el algoritmo de ordenamiento de esta función es más eficiente que los utilizados por cualquiera de las funciones que se podrían haber codificado, los cuales son de orden cuadrático.

Para el encabezo y el pie de página del archivo xml a exportar, el programa lee archivos xml externos y los exporta al archivo de salida. Se implemento de esta manera, en vez de que el encabezo y el pie de página sean parte del arreglo de contexto xml, debido a que de esta forma si se quisiera modificar el contenido de estos (por ejemplo para utilizar otro lenguaje de marcado basado en xml), dicha modificación no requiere la recompilación del programa.

Cabe aclarar que para el desarrollo de este trabajo no se tuvo en cuenta que si alguno de los campos de la pista tiene algún caracter especial de xml (por ejemplo, &) sería necesario hacer una transformación de caracteres mediante entidades de xml para que el archivo exportado sea un documento xml válido.

El desarrollo de este trabajo se llevó a cabo utilizando la herramienta de control de versiones Git, que permitió la visualización y edición del mismo código fuente a ambos desarrolladores de manera remota.

5. Problemas de desarrollo y soluciones

En cuanto a los problemas encontrados durante el desarrollo, se destacaron tres. Estos estuvieron referidos a la utilización de herramientas de control de versiones, la determinación de la memoria estática utilizada por los campos del tipo de dato abstracto *Track* y la implementación de la función de biblioteca *qsort()*.

Si bien la utilización de herramientas de control de versiones fue, en general, provechosa, en las primeras instancias resultó ser una complicación, debido a que fue necesario aprender los comandos requeridos para su operación. Además, la edición en simultáneo del mismo código fuente significó que se debió reescribir código anteriormente implementado. A medida que se adquirió experiencia en su utilización, estas herramientas resultaron ser una ventaja en vez de un complicación.

Una vez que se decidió que las cadenas de caracteres del tipo de dato abstracto *Track_t* utilizaran memoria estática, fue necesario definir la cantidad de caracteres que tendrán. En un primer momento se definió como tal cantidad el valor de las macros pertenecientes al módulo *mp3*, que representan la cantidad de caracteres utilizados en el encabezado del archivo mp3 para cada campo, según el estándar ID3v1. Esto, sin embargo, dio por resultado el funcionamiento erróneo

del programa, de manera tal que si el largo de alguno de los campos de las pistas mp3 era igual o mayor a la cantidad definida por la macro, al momento de exportar la información de esa pista, dicha información del campo en cuestión incluía también la información del campo siguiente. Esto se debía a que, cuando se importaba una cadena de caracteres desde el archivo mp3, el caracter nulo que se incluía al final de esta no era incluido en la cadena guardada en la instancia del tipo de dato abstracto *Track_t*, ya que la cantidad de caracteres definida no era suficiente para guardar la cadena completa y el caracter nulo. Para solucionar este problema, se les asignó a los campos la cantidad de caracteres utilizados en el encabezado más uno.

Debido a la complejidad de la función *qsort()* y a la inexperiencia en su utilización, resultó problemático utilizarla para el ordenamiento del vector de pistas. Precisamente, las primitivas de comparación de pistas que son llamadas por esta función intentaban comparar datos que eran erróneos. Era claro que estas primitivas accedían de manera incorrecta a la memoria que contenía a dichos datos de manera incorrecta. Luego de horas de intentar solucionar el problema, fue resuelto modificando el molde *(cast)* que las primitivas de comparación aplican sobre los punteros a *void* que reciben como argumento. Inicialmente los punteros a *void* se moldeaban a punteros a un dato de tipo *ADT_Track_t*; al modificar este molde a punteros dobles a un dato *ADT_Track_t* se corrigió el problema. El problema se debía a que la función *qsort()* le entrega a las primitivas de comparación punteros a los elementos del vector que está ordenando. Como los elementos de este vector son de tipo punteros a dato *ADT_Track_t*, los argumentos que reciben las primitivas de comparación son, justamente, punteros dobles a *ADT_Track_t*.

6. Conclusiones

Se puede concluir que la utilización de estructuras de datos más complejas, como son los tipos de dato abstracto, permiten una separación entre la lógica de un algoritmo y la implementación de las primitivas de las estructuras, lo que da por resultado un código más simple y reutilizable. Además, ayudan a la modularización del código, lo que facilita el trabajo en equipo.

La utilización de la función *qsort()*, si bien resultó problemática, ahorró la codificación de una función de ordenamiento propia. Además, haber solucionado los problemas que trajo significa que, en un futuro, será posible utilizar esta función sin dificultad, que es más eficiente.

En cuanto a los tiempos de desarrollo: la implementación de los tipos de dato abstracto utilizados fue lo que más tiempo insumió, seguido por la validación de los argumentos. La corrección de errores también llevó un tiempo considerable. La implementación de la lógica principal del programa se llevó a cabo en relativamente poco tiempo. Esto se corresponde con lo que se aprecia en los diagramas: mientras que el diagrama de flujo (que representa la estructura lógica) es relativamente simple, salvo por las validaciones, el diagrama funcional muestra la gran cantidad de primitivas que son utilizadas por el programa.

En cuanto a las estrategias de diseño, la implementación de los tipos de dato

abstracto corresponden a una estrategia ascendente, donde se construyeron baterías de funciones primitivas, asociadas a una estructura de datos particular, para ser utilizadas a medida que sea necesario. Para la implementación de la lógica principal, se utilizó una estrategia de desarrollo descendente, donde las funciones utilizadas eran implementadas a medida que su necesidad surgía para realizar un proceso particular de la lógica.

7. Bibliografía

Estándar ID3v1: http://www.id3.org/ID3v1
(Última consulta: 03/07/2018)

Ray, *Learning XML*, Estados Unidos, O'Reilly Media, 2003.

Kernighan, Ritchie, *The C programming language*, Estados Unidos, Prentice Hall, 1988.

8. Resultados de ejecución

main.c

```c
#include <stdio.h>
#include <string.h>
#include "main.h"
#include "config.h"
#include "types.h"
#include "sysdate.h"
#include "errors.h"

/*Diccionario de literales de los formatos de fecha que puede ingresar el usuario.
  Su clave es el tipo enumerativo date_format_t.*/
char * date_formats[MAX_DATE_FORMATS] = {
    DATE_FORMAT_STR_DDMMYYYY,
    DATE_FORMAT_STR_YYYYDDD,
    DATE_FORMAT_STR_YYYYMMDD,
    DATE_FORMAT_STR_YYYYMMDDHHmmSS,
    DATE_FORMAT_STR_YYYYDDDHHmmSS};

extern config_t config;

int main(int argc, char * argv[])
{
    status_t st;
    if((st = validate_arguments(argc, argv, &config)) != OK)
    {
        print_error(st);
        return st;
    }

    if((st = sysdate(config.date_format)) != OK)
    {
        print_error(st);
        return st;
    }

    return OK;
}

status_t validate_arguments(int argc, char * argv[], config_t *config)
{
    size_t i;

    if(config == NULL)
        return ERR_NULL_POINTER;

    if(argc != CMD_ARG_AMOUNT)
        return ERR_PROGRAM_INVOCATION;

    if(strcmp(argv[CMD_ARG_POS_FORMAT_TOKEN], CMD_ARG_FORMAT_TOKEN))
        return ERR_PROGRAM_INVOCATION;

    for(i = 0; i < MAX_DATE_FORMATS; i++)
    {
        if(!strcmp(argv[CMD_ARG_POS_FORMAT_VALUE], date_formats[i]))
        {
            config->date_format = i;
            return OK;
        }
    }

    return ERR_INVALID_FORMAT;
}
```

sysdate.c

```c
#include <stdio.h>
#include <time.h>
#include "sysdate.h"
#include "main.h"
#include "types.h"
#include "errors.h"

status_t (*date_printers[MAX_DATE_FORMATS]) (time_t, FILE *) = {
    (*print_date_as_DDMMYYYY),
    (*print_date_as_YYYYDDD),
    (*print_date_as_YYYYMMDD),
    (*print_date_as_YYYYMMDDHHmmSS),
    (*print_date_as_YYYYDDDHHmmSS)};

status_t sysdate(date_format_t date_format)
{
    status_t st;
    time_t  calendar_time;

    calendar_time = time(NULL);
    if((st = (*date_printers[date_format])(calendar_time, stdout)) != OK)
        return st;

    return OK;
}

status_t print_date_as_DDMMYYYY(time_t time, FILE * stream)
{
    char s[MAX_DATE_SIZE + 1];
    char * format = "%d %m %Y";
    struct tm * time_struct;

    if(stream == NULL)
        return ERR_NULL_POINTER;

    time_struct = localtime(&time);
    if(!strftime(s, MAX_DATE_SIZE, format, time_struct))
        return ERR_CANNOT_PRINT_DATE;

    fputs(s, stream);
    return OK;
}

status_t print_date_as_YYYYDDD(time_t time, FILE * stream)
{
    char s[MAX_DATE_SIZE + 1];
    char * format = "%Y %j";
    struct tm * time_struct;

    time_struct = localtime(&time);

    if(!strftime(s, MAX_DATE_SIZE, format, time_struct))
        return ERR_CANNOT_PRINT_DATE;

    fputs(s, stream);
    return OK;
}

status_t print_date_as_YYYYMMDD(time_t time, FILE * stream)
{
    char s[MAX_DATE_SIZE + 1];
    char * format = "%Y %m %d";
    struct tm * time_struct;

    time_struct = localtime(&time);

    if(!strftime(s, MAX_DATE_SIZE, format, time_struct))
```

```
68          return ERR_CANNOT_PRINT_DATE;
69
70      fputs(s, stream);
71      return OK;
72  }
73
74  status_t print_date_as_YYYYMMDDHHmmSS(time_t time, FILE * stream)
75  {
76      char s[MAX_DATE_SIZE + 1];
77      char * format = "%Y %m %d %H %M %S";
78      struct tm * time_struct;
79
80      time_struct = localtime(&time);
81
82      if(!strftime(s, MAX_DATE_SIZE, format, time_struct))
83          return ERR_CANNOT_PRINT_DATE;
84
85      fputs(s, stream);
86      return OK;
87  }
88
89  status_t print_date_as_YYYYDDDHHmmSS(time_t time, FILE * stream)
90  {
91      char s[MAX_DATE_SIZE + 1];
92      char * format = "%Y %j %H %M %S";
93      struct tm * time_struct;
94
95      time_struct = localtime(&time);
96
97      if(!strftime(s, MAX_DATE_SIZE, format, time_struct))
98          return ERR_CANNOT_PRINT_DATE;
99
100     fputs(s, stream);
101     return OK;
102 }
```

main.h

```h
#ifndef MAIN__H
#define MAIN__H

#include <stdio.h>
#include "config.h"
#include "errors.h"
#include "types.h"

#define CMD_ARG_AMOUNT 3
#define CMD_ARG_POS_FORMAT_TOKEN 1
#define CMD_ARG_FORMAT_TOKEN "-fmt"
#define CMD_ARG_POS_FORMAT_VALUE 2

#define MAX_DATE_FORMATS 5

/*Literales de los formatos de fecha que puede ingresar el usuario*/
#define DATE_FORMAT_STR_DDMMYYYY "DDMMAAAA"
#define DATE_FORMAT_STR_YYYYDDD "AAAADDD"
#define DATE_FORMAT_STR_YYYYMMDD "AAAAMMDD"
#define DATE_FORMAT_STR_YYYYMMDDHHmmSS "AAAAMMDDHHmmSS"
#define DATE_FORMAT_STR_YYYYDDDHHmmSS "AAAADDDHHmmSS"

/*Valida argumentos y setea la configuración. config no puede ser nulo.
  La bandera de formato debe ser la indicada por CMD_ARG_FORMAT_TOKEN
  La cadena de formato ingresada debe estar en el diccionario de formatos de fecha.*/
status_t validate_arguments(int argc, char * argv[], config_t *config);

#endif
```

sysdate.h

```h
#ifndef SYSDATE__H
#define SYSDATE__H

#include <stdio.h>
#include <time.h>
#include "types.h"
#include "errors.h"


status_t sysdate(date_format_t date_format);

status_t print_date_as_DDMMYYYY(time_t time, FILE * stream);
status_t print_date_as_YYYYDDD(time_t time, FILE * stream);
status_t print_date_as_YYYYMMDD(time_t time, FILE * stream);
status_t print_date_as_YYYYMMDDHHmmSS(time_t time, FILE * stream);
status_t print_date_as_YYYYDDDHHmmSS(time_t time, FILE * stream);

#endif
```

errors.h

```c
#ifndef ERRORS__H
#define ERRORS__H

#include <stdio.h>

#define MAX_ERRORS 5

typedef enum{
    OK,
    ERR_PROGRAM_INVOCATION,
    ERR_INVALID_FORMAT,
    ERR_NULL_POINTER,
    ERR_CANNOT_PRINT_DATE}
    status_t;

status_t print_error(status_t er);

#endif
```

errors.c

```c
#include <stdio.h>
#include "types.h"
#include "errors.h"
#include "messages.h"

char * errors[MAX_ERRORS] = {
    MSG_OK,
    ERR_MSG_PROGRAM_INVOCATION,
    ERR_MSG_INVALID_FORMAT,
    ERR_MSG_NULL_POINTER,
    ERR_MSG_CANNOT_PRINT_DATE};

status_t print_error(status_t err)
{
    fprintf(stderr, "%s\n", errors[err]);
    return OK;
}
```

config.h

```h
#ifndef CONFIG__H
#define CONFIG__H

#include <stdio.h>
#include "types.h"

typedef struct{
    date_format_t date_format;}
    config_t;

#endif
```

config.c

```c
#include <stdio.h>
#include "config.h"

config_t config;
```

types.h

```h
#ifndef TYPES__H
#define TYPES__H

#include <stdio.h>
#include <time.h>

typedef enum{
    DATE_FORMAT_DDMMYYYY = 0,
    DATE_FORMAT_YYYYDDD = 1,
    DATE_FORMAT_YYYYMMDD = 2,
    DATE_FORMAT_YYYYMMDDHHmmSS = 3,
    DATE_FORMAT_YYYYDDDHHmmSS = 4}
    date_format_t;

#define MAX_DATE_SIZE 20

#endif
```

messages.h

```h
#ifndef MESSAGES__H
#define MESSAGES__H

#define MSG_OK "OK."
#define ERR_MSG_PROGRAM_INVOCATION "Error al invocar el programa."
#define ERR_MSG_INVALID_FORMAT "El formato ingresado no es valido."
#define ERR_MSG_NULL_POINTER "Puntero nulo."
#define ERR_MSG_CANNOT_PRINT_DATE "No se pudo imprimir la fecha."

#endif
```

makefile

```
CFLAGS = -ansi -pedantic -Wall
CC = gcc

all: make_sysdate clean

make_sysdate: main.o sysdate.o errors.o config.o
    $(CC) $(CFLAGS) -o sysdate main.o sysdate.o errors.o config.o

main.o: main.c main.h config.h sysdate.h types.h errors.h
    $(CC) $(CFLAGS) -o main.o -c main.c

sysdate.o: sysdate.c sysdate.h types.h errors.h
    $(CC) $(CFLAGS) -o sysdate.o -c sysdate.c

errors.o: errors.c errors.h messages.h
    $(CC) $(CFLAGS) -o errors.o -c errors.c

config.o: config.c config.h
    $(CC) $(CFLAGS) -o config.o -c config.c

clean:
    rm *.o
```