Estructuras de Datos y Algoritmos

Tecnólogo en Informática

INSTRUCTIVO DEL COMANDO MAKE

Contenido

➢ Introducción

Un programa escrito en C está normalmente compuesto por varios archivos.
Estos archivos se van modificando según se va completando o cambiando el programa. Cada vez que se modifica algún archivo del programa, éste debe recompilarse para generar un ejecutable del programa actualizado. La modificación de un archivo del programa no implica que se deban recompilar todos los archivos del mismo, únicamente se debe recompilar el archivo modificado, así como otros archivos que puedan depender de él. La utilidad **make** determina automáticamente qué archivos del programa deben ser recompilados y los comandos que se deben utilizar para realizar esta tarea. En programas muy grandes nos ayuda a mantener el ejecutable totalmente actualizado. Cuando lo utilizamos, no debemos recordar las dependencias entre archivos del programa ni los comandos necesarios para compilar cada parte del mismo. La utilidad **make** no sólo sirve para compilar programas en C, también se puede utilizar:

- Con otros lenguajes cuyo compilador se pueda ejecutar en una "shell" (en la línea de comandos).
- Para tareas de instalación y actualización de aplicaciones.

Para usar **make** debemos escribir un archivo llamado *makefile*. Este archivo describe las relaciones entre los archivos y los comandos que se deben ejecutar con cada uno de ellos. Si en un directorio existe el archivo *makefile*, para ejecutar la utilidad **make** simplemente debemos escribir en la línea de comandos:

```
> make
```

Con este comando **make** busca el archivo *makefile* y se encarga de ejecutar las ordenes contenidas en el mismo.

➢ El archivo *makefile*

La utilidad **make** normalmente ejecuta las reglas contenidas en un archivo llamado *makefile*. Como se verá más adelante, es posible indicar a **make** que lea reglas de archivos con un nombre diferente a éste. Un *makefile* simple está compuesto por reglas que tienen la siguiente forma:

```
etiqueta ... : dependencias...
      comando
      comando
      ...
```

Una *etiqueta* es un rótulo que generalmente se refiere a un nombre de archivo.
Este archivo es aquel que queremos actualizar con esta regla. También hay etiquetas que indican la acción que la regla realiza; por ejemplo *limpiar* (ver en el ejemplo siguiente). Una *dependencia* es un archivo que se usa en la regla. Si este archivo se modifica la regla se activa. Una regla puede depender de varios archivos. Un *comando* es la acción que make ejecuta si la regla se activa. Una regla puede ejecutar más de un comando.

Nota muy importante: Una línea con un comando **empieza** siempre con un **tabulador** (no por varios caracteres espacio).

Una regla tiene un nombre (*etiqueta*) y dice como recompilar un archivo (mediante comandos) si algún archivo del que depende se modifica (*dependencias*).

➢ Ejemplo de *makefile* sencillo

Éste es un ejemplo de un *makefile* que permite generar un ejecutable llamado *prog.exe* a partir de un programa en C que tiene 3 archivos: el programa principal *prog.cpp*, un archivo fuente *vector.cpp*, y su archivo de cabecera *vector.h*.

```
todo: prog.exe
vector.o: vector.h vector.cpp
      g++ -c vector.cpp -Wall
prog.o: prog.cpp vector.h
      g++ -c prog.cpp -Wall
prog.exe: prog.o vector.o
      g++ prog.o vector.o -o prog.exe
limpiar:
      rm prog.o
      rm vector.o
      rm prog.exe
```

Nota: El modificador –Wall activa todos los posibles warnings del compilador.

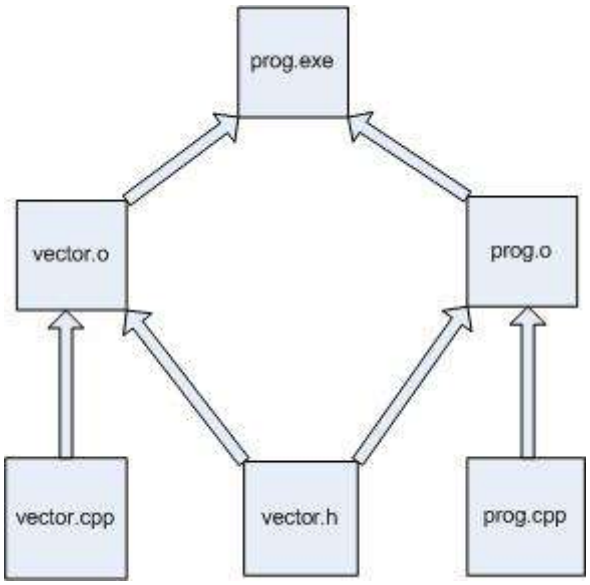
Este *makefile* permite hacer dos cosas: crear un ejecutable actualizado del programa y borrar el archivo ejecutable junto con todos los archivos objeto del mismo. Para crear el ejecutable actualizado del programa hay que escribir en la línea de comandos:

```
> make
```

En el ejemplo anterior tenemos una primera regla llamada 'todo'. Esta regla no tiene comandos y sí tiene una dependencia 'prog.exe'. Esta regla nos indica sólo que hay que ver si el archivo *prog.exe* está actualizado. La regla que se encarga de actualizar el archivo *prog.exe* está más abajo y tiene como etiqueta 'prog.exe'. La segunda regla tiene como etiqueta 'vector.o'. El comando que contiene la regla es: `g++ -c vector.cpp –Wall`, que hace que el compilador construya el archivo objeto *vector.o*, y tiene las dependencias *vector.cpp* y *vector.h*. Este comando se ejecutará si el archivo *vector.cpp* o el archivo *vector.h* han sido modificados. La tercera regla es semejante a la anterior. La cuarta regla tiene como etiqueta 'prog.exe'. El comando de la regla hace que se construya el archivo ejecutable 'prog.exe'. Sus dependencias son *prog.o* y *vector.o*. Esta regla se ejecutará si los archivos indicados en las dependencias han sido modificados.

El funcionamiento de este *makefile* es el siguiente:
Al ejecutar la utilidad **make** sin argumentos se activa la primer regla del *makefile*. Esta regla es 'todo', la cual no tiene comandos que ejecutar, pero sí tiene una dependencia, 'prog.exe'. La regla 'todo' unicamente activa la regla 'prog.exe'. La regla 'prog.exe' a su vez se activa si los archivos *prog.o* o *vector.o* se han modificado. Antes de ejecutar el comando asociado a la regla 'prog.exe' se comprueba si es necesario activar las reglas asociadas a los archivos 'vector.o' y ' prog.o'. La regla 'prog.exe' llama a las reglas 'vector.o' y 'prog.o'. La regla 'vector.o' tiene dependencias de los archivos *vector.cpp* y *vector.h*. Como se ve ya no hay reglas asociadas a estos archivos por lo que la orden asociada a la regla 'vector.o' se ejecutará si alguno de los archivos indicados en sus dependencias han sido modificados. La regla 'prog.o' funciona exactamente igual que la regla 'vector.o'. Para generar el archivo *prog.exe* en primer lugar se observa si se ha

modificado alguno de los siguientes archivos: *vector.cpp*, *vector.h* y *prog.cpp*. En caso afirmativo se actualizan los archivos objeto *vector.o* y *prog.o*. En último lugar se construye el archivo ejecutable *prog.exe* a partir de los archivos objeto previamente actualizados. Un esquema de como el *makefile* indica las dependencias entre los archivos que forman el programa y el proceso de actualización que sigue hasta construir un ejecutable *prog.exe* acorde con los archivos fuente (**.cpp* y **.h*) que se están manejando en ese momento puede ser:



Como se ve, al ejecutar la utilidad **make** se activan todas las reglas del *makefile* excepto la regla 'limpiar'. Esta regla se utiliza para eliminar los archivos objeto y el archivo ejecutable y puede tener sentido para forzar una compilación completa del programa. El comando necesario para borrar el ejecutable y los archivos objeto es:

```
>make limpiar
```

La utilidad **make** sin argumentos ejecuta la primera regla del *makefile*. Para ejecutar una regla distinta hay que pasarle como argumento el nombre de la regla que queremos activar. Si por ejemplo sólo quisiéramos obtener el archivo *vector.o* podríamos ejecutar:

```
>make vector.o
```

➢ Variables de un *makefile*

En el *makefile* anterior hemos usado en varias líneas el comando 'g++' y la lista de objetos 'vector.o prog.o'. Estas palabras se repiten muchas veces dentro del texto del *makefile*. Estas cadenas se pueden sustituir por variables que se definen dentro del *makefile*. Una variable se define de la forma siguiente:

```
nombre_variable= lista palabras a las que sustituye
```

Se hace referencia a una variable dentro del *makefile* de la forma siguiente:

```
$(nombre_variable)
```

Dentro de un *makefile*, cada vez que se encuentre la referencia a una variable, ésta equivale a la lista de palabras indicadas en el momento en que se definió la misma. El *makefile* anterior se puede sustituir por uno equivalente de la manera siguiente:

```
obj= vector.o prog.o
CC= g++
WARNING = -Wall
LIB=
todo: prog.exe
vector.o: vector.h vector.cpp
	$(CC) -c vector.cpp $(WARNING)
prog.o: prog.cpp vector.h
	$(CC) -c prog.cpp $(WARNING)
prog.exe: $(obj)
	$(CC) $(obj) -o prog.exe $(LIB)
limpiar:
	rm prog.exe
	rm $(obj)
```

Las variables facilitan la escritura y evitan repetir listas de palabras iguales en líneas distintas (con el peligro de que no todas las líneas tengan la misma lista de palabras). Por otro lado, facilitan el mantenimiento de un *makefile*.
Obsérvense las variables CC, WARNING y LIB. Si quisiéramos utilizar otro compilador (por ejemplo 'turboc') sólo debemos modificar el valor de la variable CC. Lo mismo ocurriría si quisiéramos modificar el tipo de 'warnings' que queremos conocer al compilar. Sólo habría que modificar la variable WARNING. Si necesitáramos añadir alguna biblioteca para enlazar ('linkeditar') sólo deberíamos añadir a la definición de la variable LIB las bibliotecas necesarias.

➢ Cómo utilizar make con archivos cuyo nombre no es *makefile*

La utilidad **make** puede utilizar cualquier archivo que contenga reglas como las anteriormente descritas. Por defecto **make** siempre busca archivos con el nombre *makefile*. Si queremos que ejecute reglas de un archivo con otro nombre se debe utilizar la opción **-f**. Por ejemplo, si tengo un archivo llamado 'regla.txt' y quiero ejecutar las reglas contenidas en él, el comando que se utilizo es:

```
> make -f reglas.txt
```

➢ Comentarios en un *makefile*

Podemos colocar comentarios en un *makefile* antecediendo las líneas a comentar por el carácter '#'.