Tema 5: Herramientas de programación y diseño modular

Programación 2

Curso 2013-2014

Notas

Índice

Notas

El proceso de compilación (1/2)

- La tarea de traducir un programa fuente en ejecutable se realiza en dos fases:
  - Compilación: El compilador traduce un programa fuente en un programa en código objeto (no ejecutable)
  - Enlace: El enlazador (*linker*): junta el programa en código objeto con las librerías del lenguaje (C/C++) y genera el ejecutable
- En C++, se realizan las dos fases con la siguiente instrucción:

```
g++ programa.cc -o programa
```

Notas

El proceso de compilación (2/2)

El compilador de C++ (`g++`) admite muchos más parámetros. Por ejemplo:

- `-Wall` : Muestra todos los warnings, no sólo los más importantes (recomendada)
- `-g` : Compila en un modo que facilita encontrar los errores mediante un depurador (recomendada en P2)
- `-c` : Sólo compila, generando código objeto pero sin hacer el enlace
- `-o`: Indica el nombre del ejecutable
- `-version` : Muestra la versión actual del compilador

Ejemplo: `g++ -Wall -g programa.cc -o programa`

Notas

Depuracion

Un depurador (`debugger`) es un programa que nos ayuda para encontrar y corregir bugs (errores) en nuestro código.

- **GDB**. Inicia nuestro programa, lo para cuando lo pedimos y mira el contenido de las variables. Si nuestro ejecutable da un fallo de segmentación, nos dice la línea de código dónde está el problema.
- **Valgrind**. Detecta errores de memoria (acceso a componentes fuera de un vector, variables usadas sin inicializar, punteros que no apuntan a una zona reservada de memoria, etc.)
- Más ejemplos en Linux: DDD, Nevimer, Electric Fence, DUMA, etc.

Notas

Diseño modular

- Cuando el programa crece es necesario dividirlo en módulos
- Cada módulo agrupa una serie de funciones con algo en común
- El módulo puede reutilizarse en otros programas: reusabilidad de código
- Importante: permite probar las funciones de cada módulo por separado

Notas

Compilación separada (1/2)

- Cuando un programa se compone de varios módulos, lo que debe hacerse para obtener el ejecutable es:
  1. Compilar los módulos fuente por separado, obteniendo varios ficheros en código objeto
  2. Enlazar los ficheros en código objeto (los módulos compilados) con las librerías y generar un ejecutable

Notas

Compilación separada (2/2)

- El programa principal '`main`' usa y comunica los módulos
- Un módulo '`m`' se implementa con dos ficheros fuente:
  - `m.h` : contiene constantes, tipos y prototipos de funciones visibles fuera del módulo
  - `m.cc` : contiene el cuerpo de las funciones (y puede que constantes y tipos internos al módulo)

Notas

El fichero de cabecera 'm.h'

```
// m.h
// Informacion del autor, fecha, cambios, ...

#ifndef _m_
#define _m_

// constantes

// tipos

// prototipos:  int f(int,double);

#endif
```

Notas

El fichero 'm.cc'

```
// m.cc
// Informacion del autor, fecha, cambios, ...

#include <iostream>
// ... (otros ficheros de cabecera estandar)

using namespace std;

#include "m.h"
// ... (otros ficheros de cabecera propios)

// codigo de las funciones
```

Notas

Ejemplo

Supongamos que hemos hecho un programa (para Linux) llamado '`copiarMemoriaUSB.cc`' que monta una memoria USB, copia el contenido de esa memoria en un directorio de nuestro disco duro y después formatea la memoria y la desmonta. Cuando termina muestra por pantalla cuántos archivos tenía y qué tamaño ocupaban.

```
#include <iostream>
using namespace std;

const int MAXPMONTAJE = 200;

typedef struct {
  int narchivos, tamano;
  char puntoMontaje[MAXPMONTAJE];
} memUSB;

bool MontarUSB(memUSB &m) { ... }
...
```

Notas

Ejemplo (continuación)

```
...
bool CopiarADirectorio(memUSB m,char dir[]) { ... }
void FormatearUSB(memUSB m) { ... }
void DesmontarUSB(memUSB m) { ... }
void MostrarDatosUSB(memUSB m) { ... }

int main()
{
   memUSB mem;

   if (MontarUSB(mem))
      if (CopiarADirectorio(mem,"/tmp/memusb")) {
        FormatearUSB(mem);
        DesmontarUSB(mem);
        MostrarDatosUSB(mem);
      }
}
```

Notas

Ejemplo (continuación)

Supongamos que deseamos preparar nuestro programa para poder utilizar las funciones de manejo de la memoria USB en otros programas. Para ello habría que crear un módulo "`memoriaUSB`", con dos ficheros fuente:

- `memoriaUSB.h`: contendrá los tipos definidos, constantes y prototipos de funciones que queremos que sean visibles desde otros módulos o programas
- `memoriaUSB.cc`: contendrá solamente el cuerpo de las funciones, y posiblemente otras funciones, constantes y tipos de uso interno en el módulo

Notas

El fichero de cabecera '`memoriaUSB.h`'

```
// memoriaUSB.h
#ifndef _memoriaUSB_
#define _memoriaUSB_

// constantes
const int MAXPMONTAJE = 200;

// tipos
typedef struct {
  int narchivos,tamano;
  char puntoMontaje[MAXPMONTAJE];
} memUSB;

// prototipos (con comentarios explicativos)
bool MontarUSB(memUSB &);
bool CopiarADirectorio(memUSB,char []);
void FormatearUSB(memUSB);
void DesmontarUSB(memUSB);
void MostrarDatosUSB(memUSB);
#endif
```

Notas

El fichero 'memoriaUSB.cc'

```
// memoriaUSB.cc
#include <iostream>

using namespace std;

#include "memoriaUSB.h"

// codigo de las funciones

bool MontarUSB(memUSB &m) { ... }
bool CopiarADirectorio(memUSB m,char dir[]) { ... }
void FormatearUSB(memUSB m) { ... }
void DesmontarUSB(memUSB m) { ... }
void MostrarDatosUSB(memUSB m) { ... }
```

Notas

El fichero 'copiarMemoriaUSB.cc'

```
// copiarMemoriaUSB.cc
#include <iostream>

using namespace std;

#include "memoriaUSB.h"

int main()
{
   memUSB mem;

   if (MontarUSB(mem))
      if (CopiarADirectorio(mem,"/tmp/memusb")) {
        FormatearUSB(mem);
        DesmontarUSB(mem);
        MostrarDatosUSB(mem);
      }
}
```

Notas

¿Cómo se compila el programa?

- Compilar cada módulo y el programa principal por separado:

```
g++ -Wall -g -c m1.cc
g++ -Wall -g -c m2.cc
g++ -Wall -g -c prog.cc
```

- Enlazar los módulos y el programa, y obtener el ejecutable:

```
g++ -Wall -g m1.o m2.o prog.o -o prog
```

- Atajo: (sólo para programas pequeños)

```
g++ -Wall -g m1.cc m2.cc prog.cc -o prog
```

Notas

Cuando el programa crece y crece . . .

Problema: un fichero de cabecera '`x.h`' se usa en varios `.cc` ¿Qué hay que hacer si se cambia algo en el `.h`?

- Opción 1: lo recompilo todo (a lo *"bestia"*)
- Opción 2: busco dónde se usa, y sólo recompilo esos módulos
- Opción 3: vale, pero ¿hay que hacerlo a mano? NO

Existe un programa llamado '`make`' que lo hace

Notas

Utilizando el programa 'make'

- Es un programa que ayuda a compilar programas grandes
- Permite establecer *dependencias*
- Compila un módulo cuando alguno de los ficheros de los que depende cambia

Notas

Creando un 'makefile'

- El 'makefile' es un fichero que especifica las dependencias entre los ficheros y qué hacer cuando cambia un fichero ('make' busca por defecto un fichero con ese nombre)
- Tiene un *objetivo* principal (normalmente el programa ejecutable) seguido de otros objetivos secundarios. Cada objetivo aparece junto con los ficheros de los que depende, seguidos de las instrucciones necesarias para construir el objetivo a partir de los ficheros
- El formato de cada objetivo es:

```
<objetivo> : <fichero 1> <fichero 2> ... <fichero N>
[tabulador]<instrucción 1>
[tabulador]<instrucción 2>
...
[tabulador]<instrucción M>
```

Notas

Creando un 'makefile' (2)

Ejemplo:

```
prog : m1.o m2.o prog.o
        g++ -Wall -g m1.o m2.o prog.o -o prog

m1.o : m1.cc m1.h
        g++ -Wall -g -c m1.cc
m2.o : m2.cc m2.h m1.h
        g++ -Wall -g -c m2.cc
prog.o : prog.cc m1.h m2.h
        g++ -Wall -g -c prog.cc
```

Notas

Como funciona 'make'

- Si se cambia `m2.cc` ...

```
$ make

g++ -Wall -g -c m2.cc
g++ -Wall -g m1.o m2.o prog.o -o prog
```

- Si se cambia `m2.h` ...

```
$ make

g++ -Wall -g -c m2.cc
g++ -Wall -g -c prog.cc
g++ -Wall -g m1.o m2.o prog.o -o prog
```

Notas

Constantes en `makefile`

Ejemplo anterior con constantes:

```
CC = g++
CFLAGS = -Wall -g
MODULOS = m1.o m2.o prog.o

prog : $(MODULOS)
        $(CC) $(CFLAGS) $(MODULOS) -o prog

m1.o : m1.cc m1.h
        $(CC) $(CFLAGS) -c m1.cc
m2.o : m2.cc m2.h m1.h
        $(CC) $(CFLAGS) -c m2.cc
prog.o : prog.cc m1.h m2.h
        $(CC) $(CFLAGS) -c prog.cc
clean:
        rm -rf $(MODULOS)
```

Más info: `http://es.wikipedia.org/wiki/Make`

Notas

Ejercicios

Ejercicio 1

Dado el siguiente programa

```
#include <iostream>
using namespace std;
const int MAXNUM = 20;

void ImprNumeros(int n)
{
  int i;
  for (i=0;i<n;i++)
     cout << "Numero: " << i << endl;
}

int main()
{
  ImprNumeros(MAXNUM);
}
```

divididlo en los módulos `numeros.h`, `numeros.cc`, `principal.cc`, y haced el `makefile` correspondiente.

Notas