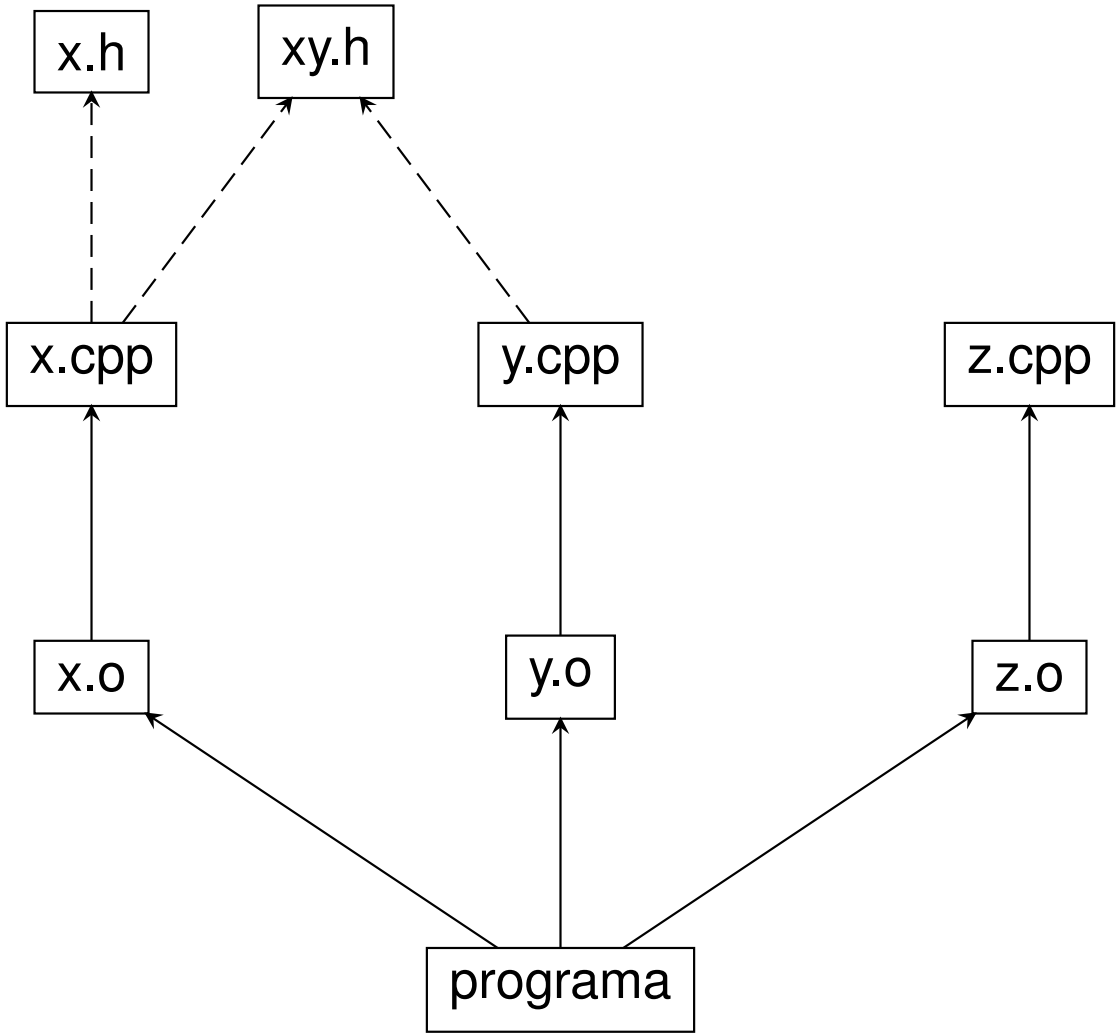
Compilación y recompilación

Francisco Palomo Lozano Inmaculada Medina Bulo

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Cádiz

Makefiles

Ejemplo de grafo de dependencias



Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

1. Sólo necesito recompilar los módulos modificados
2. El proceso es incremental

Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

1. Sólo necesito recompilar los módulos modificados
2. El proceso es incremental

Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

1. Sólo necesito recompilar los módulos modificados
   - Ya sea porque lo han sido directamente
   - O porque dependen, por ejemplo, por incluir cabeceras recompiladas
2. El proceso es automatizable

Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

1. Sólo necesito recompilar los módulos modificados
   - Ya sea porque lo han sido directamente
   - O indirectamente, por ejemplo, por incluir cabeceras modificadas
2. El proceso es automatizable

Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

1. Sólo necesito recompilar los módulos modificados
   - Ya sea porque lo han sido directamente
   - O indirectamente, por ejemplo, por incluir cabeceras modificadas
2. El proceso es automatizable
   - Existen varias herramientas disponibles, por ejemplo, Make
   - Se suministran dependencias y acciones en un formato apropiado

Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

1. Sólo necesito recompilar los módulos modificados
   - Ya sea porque lo han sido directamente
   - O indirectamente, por ejemplo, por incluir cabeceras modificadas
2. El proceso es automatizable
   - Existen varias herramientas disponibles, por ejemplo, Make

Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

1. Sólo necesito recompilar los módulos modificados
   - Ya sea porque lo han sido directamente
   - O indirectamente, por ejemplo, por incluir cabeceras modificadas
2. El proceso es automatizable
   - Existen varias herramientas disponibles, por ejemplo, Make
   - Se suministran dependencias y acciones en un formato apropiado

Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

1. Sólo necesito recompilar los módulos modificados
   - Ya sea porque lo han sido directamente
   - O indirectamente, por ejemplo, por incluir cabeceras modificadas
2. El proceso es automatizable
   - Existen varias herramientas disponibles, por ejemplo, Make
   - Se suministran dependencias y acciones en un formato apropiado

Compilación del proyecto

Compilación conjunta frente a separada

1. Compilación y enlazado para obtener el ejecutable en un paso

```
c++ -o programa x.cpp y.cpp z.cpp
```

2. Compilación y enlazado por separado

```
c++ -c x.cpp y.cpp z.cpp
c++ -o programa x.o y.o z.o
```

Ventajas de la compilación separada

1. Sólo necesito recompilar los módulos modificados
   - Ya sea porque lo han sido directamente
   - O indirectamente, por ejemplo, por incluir cabeceras modificadas
2. El proceso es automatizable
   - Existen varias herramientas disponibles, por ejemplo, Make
   - Se suministran dependencias y acciones en un formato apropiado

Makefile: primera versión

```
# El formato de una regla es el siguiente:
#
# objetivos: dependencias
#       TAB    acciones


programa: x.o y.o z.o
        c++ -o programa x.o y.o z.o

x.o: x.cpp x.h xy.h
        c++ -c x.cpp

y.o: y.cpp xy.h
        c++ -c y.cpp

z.o: z.cpp
        c++ -c z.cpp
```

Makefile: primera versión

```
# El formato de una regla es el siguiente:
#
# objetivos: dependencias
#       TAB    acciones


programa: x.o y.o z.o
        c++ -o programa x.o y.o z.o

x.o: x.cpp x.h xy.h
        c++ -c x.cpp

y.o: y.cpp xy.h
        c++ -c y.cpp

z.o: z.cpp
        c++ -c z.cpp
```

Makefile: primera versión

```
# El formato de una regla es el siguiente:
#
# objetivos: dependencias
#       TAB    acciones


programa: x.o y.o z.o
        c++ -o programa x.o y.o z.o

x.o: x.cpp x.h xy.h
        c++ -c x.cpp

y.o: y.cpp xy.h
        c++ -c y.cpp

z.o: z.cpp
        c++ -c z.cpp
```

Makefile: primera versión

```
# El formato de una regla es el siguiente:
#
# objetivos: dependencias
#       TAB    acciones


programa: x.o y.o z.o
        c++ -o programa x.o y.o z.o

x.o: x.cpp x.h xy.h
        c++ -c x.cpp

y.o: y.cpp xy.h
        c++ -c y.cpp

z.o: z.cpp
        c++ -c z.cpp
```

Makefile: segunda versión

```
# Empleo de macros predefinidas y especiales

CXX = c++



programa: x.o y.o z.o
	$(CXX) -o $@ $(LDFLAGS) $^

x.o: x.cpp x.h xy.h
	$(CXX) -c $(CXXFLAGS) $<

y.o: y.cpp xy.h
	$(CXX) -c $(CXXFLAGS) $<

z.o: z.cpp
	$(CXX) -c $(CXXFLAGS) $<
```

Makefile: segunda versión

```
# Empleo de macros predefinidas y especiales

CXX = c++



programa: x.o y.o z.o
	$(CXX) -o $@ $(LDFLAGS) $^

x.o: x.cpp x.h xy.h
	$(CXX) -c $(CXXFLAGS) $<

y.o: y.cpp xy.h
	$(CXX) -c $(CXXFLAGS) $<

z.o: z.cpp
	$(CXX) -c $(CXXFLAGS) $<
```

Makefile: segunda versión

```
# Empleo de macros predefinidas y especiales

CXX = c++



programa: x.o y.o z.o
	$(CXX) -o $@ $(LDFLAGS) $^

x.o: x.cpp x.h xy.h
	$(CXX) -c $(CXXFLAGS) $<

y.o: y.cpp xy.h
	$(CXX) -c $(CXXFLAGS) $<

z.o: z.cpp
	$(CXX) -c $(CXXFLAGS) $<
```

Makefile: segunda versión

```
# Empleo de macros predefinidas y especiales

CXX = c++



programa: x.o y.o z.o
	$(CXX) -o $@ $(LDFLAGS) $^

x.o: x.cpp x.h xy.h
	$(CXX) -c $(CXXFLAGS) $<

y.o: y.cpp xy.h
	$(CXX) -c $(CXXFLAGS) $<

z.o: z.cpp
	$(CXX) -c $(CXXFLAGS) $<
```

Makefile: segunda versión

```
# Empleo de macros predefinidas y especiales

CXX = c++



programa: x.o y.o z.o
	$(CXX) -o $@ $(LDFLAGS) $^

x.o: x.cpp x.h xy.h
	$(CXX) -c $(CXXFLAGS) $<

y.o: y.cpp xy.h
	$(CXX) -c $(CXXFLAGS) $<

z.o: z.cpp
	$(CXX) -c $(CXXFLAGS) $<
```

Makefile: tercera versión

```
# Definición de nuevas macros

CXX = c++

OBJETOS = x.o y.o z.o

programa: $(OBJETOS)
        $(CXX) -o $@ $(LDFLAGS) $(OBJETOS)

x.o: x.cpp x.h xy.h
        $(CXX) -c $(CXXFLAGS) $<

y.o: y.cpp xy.h
        $(CXX) -c $(CXXFLAGS) $<

z.o: z.cpp
        $(CXX) -c $(CXXFLAGS) $<
```

Makefile: tercera versión

```
# Definición de nuevas macros

CXX = c++

OBJETOS = x.o y.o z.o

programa: $(OBJETOS)
        $(CXX) -o $@ $(LDFLAGS) $(OBJETOS)

x.o: x.cpp x.h xy.h
        $(CXX) -c $(CXXFLAGS) $<

y.o: y.cpp xy.h
        $(CXX) -c $(CXXFLAGS) $<

z.o: z.cpp
        $(CXX) -c $(CXXFLAGS) $<
```

Makefile: cuarta versión

```
# Empleo de macros y reglas implícitas

CXX = c++

OBJETOS = x.o y.o z.o

programa: $(OBJETOS)
        $(CXX) -o $@ $(LDFLAGS) $(OBJETOS)

x.o: x.h xy.h


y.o: xy.h
```

Makefile: cuarta versión

```
# Empleo de macros y reglas implícitas

CXX = c++

OBJETOS = x.o y.o z.o

programa: $(OBJETOS)
        $(CXX) -o $@ $(LDFLAGS) $(OBJETOS)

x.o: x.h xy.h


y.o: xy.h
```

Makefile: quinta versión

```
# Empleo de macros, reglas implícitas y objetivos falsos

CXX = g++
CXXFLAGS = -ansi
OBJETOS = x.o y.o z.o

all: programa

programa: $(OBJETOS)
        $(CXX) -o $@ $(LDFLAGS) $(OBJETOS)

x.o: x.h xy.h
y.o: xy.h

clean:
        $(RM) programa $(OBJETOS) core *~
```

Makefile: quinta versión

```
# Empleo de macros, reglas implícitas y objetivos falsos

CXX = g++
CXXFLAGS = -ansi
OBJETOS = x.o y.o z.o

all: programa

programa: $(OBJETOS)
        $(CXX) -o $@ $(LDFLAGS) $(OBJETOS)

x.o: x.h xy.h
y.o: xy.h

clean:
        $(RM) programa $(OBJETOS) core *~
```

Makefile: quinta versión

```
# Empleo de macros, reglas implícitas y objetivos falsos

CXX = g++
CXXFLAGS = -ansi
OBJETOS = x.o y.o z.o

all: programa

programa: $(OBJETOS)
        $(CXX) -o $@ $(LDFLAGS) $(OBJETOS)

x.o: x.h xy.h
y.o: xy.h

clean:
        $(RM) programa $(OBJETOS) core *~
```

Makefile: quinta versión

```
# Empleo de macros, reglas implícitas y objetivos falsos

CXX = g++
CXXFLAGS = -ansi
OBJETOS = x.o y.o z.o

all: programa

programa: $(OBJETOS)
        $(CXX) -o $@ $(LDFLAGS) $(OBJETOS)

x.o: x.h xy.h
y.o: xy.h

clean:
        $(RM) programa $(OBJETOS) core *~
```

Bibliografía

D. Morse y R. McGrath
Documentación de GNU Make 3.80 en el sistema Info
`emacs (C-h i m make)` | `info make` (2002)

D. Morse y R. McGrath
Página resumen del manual de GNU Make 3.80
`man make` (2002)