Programmeertechnieken
Week 2

Kristian Rietveld

http://liacs.leidenuniv.nl/~rietveldkfd/courses/pt2016/

Universiteit Leiden
The Netherlands

Vorige week

- De UNIX omgeving: een Zwitsers zakmes.
- Bouwen van pipelines.
- Shell scripting & awk.

Deze week

- Scripttalen.
- Stappen in softwarecompilatie.
- Shared libraries & Linking.
- Makefiles.
- Build systems.

Scripttalen

- Vorige week maakten we al kennis met bash scripting en "awk".
- Wat nu als we geen geschikte standaard tool kunnen vinden? Hoe zetten we snel zelf een tool in elkaar?
- Er bestaan vele scripttalen: Python, perl, ruby, PHP, JavaScript, VBScript.
- Bij deze talen wordt er niet gecompileerd, de scripts worden tijdens run-time geinterpreteerd.

Python

- Op dit moment de meeste populaire scripttaal.
  - (Heeft in de laatste 10 jaar Perl ingehaald).
- Hierdoor standaard geinstalleerd op Linux, Mac systemen.
- Ontworpen door Guido van Rossum eind jaren '80 / begin jaren '90.
- Eenvoudig & portable.

Python (2)

- **Waarom zo populair?**
- Veel doen in weinig regels code.
- Code blijft leesbaar.
- Zeer uitgebreide standaard bibliotheek.
- Eenvoudig om zelf uitbreidingen te schrijven.
- Eenvoudig om uitbreidingen te schrijven die gebruik maken van bestaande bibliotheken geschreven in bijvoorbeeld C.

Python (3)

- We bespraken vorige week de "shebang" ("#!") regel.
- Gebruik dit ook voor Python scripts om hiervan "executables" te maken.

```
#!/usr/bin/env python

print "Hello world"
exit(0)
```

Een eerste Python programma

```
# Dit is een regel met commentaar ...
import math # voor de "pi" constante
print "Geef straal, daarna Enter ..",
straal = float(raw_input())
if straal > 0:
    print "Oppervlakte:",
    print math.pi * straal * straal
else:
    print "Niet zo negatief ..."
print "Einde van dit programma."
exit(0)
```

Een eerste Python programma

```
# Dit is een regel met commentaar ...
import math # voor de "pi" constante
print "Geef straal, daarna Enter ..",
straal = float(raw_input())
if straal > 0:
    print "Oppervlakte:",
    print math.pi * straal * straal
else:
    print "Niet zo negatief ..."
print "Einde van dit programma."
exit(0)
```

Een tweede Python programma

```
#!/usr/bin/env python
import sys

print "De argumenten aan dit programma
zijn:"
for i, arg in enumerate(sys.argv):
    print "{0}: {1}".format(i, arg)
exit(0)
```

Variabelen in Python

- Elke variabele in Python heeft een type.

```
>>> a, b, c = 9, 3.14, "strrrr"
>>> type(a)
<type 'int'>
>>> type(b)
<type 'float'>
>>> type(c)
<type 'str'>
>>> a = "strrr2" # oude waarde van a wordt overschreven
>>> type(a)
<type 'str'>
```

Conversie van getallen

- `float()` is een type conversie. Accepteert ook strings: `float("3.14")`.
- Andere typeconversies: `int()`, `complex()`, `str()`.
- Niet hetzelfde als een C cast, die kan bijvoorbeeld niet van string naar float!
- Operatie op twee verschillende typen resulteert in een impliciete conversie: type coercion.

print statement

- `print` zet data op het scherm.
- Keyword `print`, gevolgd door een lijst van expressies.
- Impliciete conversie naar strings.
- Spaties ingevoegd tussen uitvoeren van verschillende expressies.

print statement

```
>>> a = 110
>>> b = 12
>>> print "Test:", "a =", a, "b =", b, "en samen
maakt dat", a + b
Test: a = 110 b = 12 en samen maakt dat 122
```

Strings - Indexing en slicing

- Een string in Python is een object, net als de C++ "string" klasse.
  - Python heeft geen `char` type.
- In een ouderwetse C-string kunnen we met een index een individueel array element uitlezen.
- In Python kunnen we objecten van het type `str` ook "indexen".
  - `woord[1]`
- Je mag ook een start en eind index geven, bijvoorbeeld om een substring uit te lezen. We noemen dit slicing.
  - `zin[3:14]`
  - De eind-index telt **niet** mee!

Strings - Indexing en slicing

```
>>> s = "een lange test string"
>>> s[2]
'n'
>>> s[-4]
'r'
>>> s[3:8]
' lang'
>>> s[6:]
'nge test string'
```

Lijsten

- Een `list`-object is een geordende lijst van variabelen.
- Typen van de elementen mag verschillend zijn.

```
>>> a = [0, 1, 2, 3, 4, 5, 6, 7]
>>> len(a)
8
>>> a[6]
6
>>> a[2:5]
[2, 3, 4]
>>> a[3:]
[3, 4, 5, 6, 7]
>>> a[:6]
[0, 1, 2, 3, 4, 5]
```

Dictionaries

- Net als in awk, een associatieve array.
- Indexering mag met iets anders dan een geheel getal.

```
>>> d = dict()
>>> d["walter"] = "071-5270000"
>>> d["kris"] = "06-12345678"
>>> d["joop"] = "0123-524513"
# Value ophalen uit de dictionary aan de
hand van een key
>>> d["kris"]
'06-12345678'
```

for loops

- `for`-loops zijn in Python een stuk eenvoudiger.
- We drukken een iteratie van een lijst uit.
- De iteratievariabele neemt opeenvolgend de verschillende waarden van de lijst aan.
- Voor getallenreeksen kunnen we met `range()` automatisch lijsten genereren.
  - `range([start], stop, [step])`

for loops (2)

```
for karakter in ['a', 'b', 'c', 'd', 'e']:
    print karakter,
# drukt af: a b c d e

for i in [1, 2, 3, 4, 5]:
    print i

for i in range(5):
    print i
```

Inspringregels

- In C++ kun je slordig zijn met de layout van je code, Python is daar echter veel strikter in.
- Correct inspringen is een must, fout inspringen wordt bestraft met een `IndentationError`.
- Wanneer inspringen?
  - Om blokken van statements te vorm.
  - `if`-statements, loops en definieren van functies.
  - In C++ plaats je bij bijna al deze gevallen accolades!

Inspringregels (2)

- Binnen eenzelfde blok **moet** er op elke regel op dezelfde manier worden ingesprongen.
- De eerste regel die anders wordt ingesprongen maakt geen deel meer uit van dat blok.
- Advies: altijd 4 spaties, vermijd tabs.

Filter loop

- Open en lees alle bestanden gegeven op command-line.
- Of anders, lees `stdin` tot EOF. (Pipelines!)

```
#!/usr/bin/env python
import fileinput

for line in fileinput.input():
    line = line.rstrip()
    # Doe iets met "regel"
    print line[::-1]
```

Voorbeeld: Regels tellen

```
#!/usr/bin/env python
import fileinput

g = w = 0
for line in fileinput.input():
    if line.startswith("g"):
        g += 1
    if line.startswith("w"):
        w += 1

print "G regels:", g
print "W regels:", w
```

Plotten

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 2, 25)
y1 = 3 * x + 5
y2 = 5 * x ** 2 - 3

plt.plot(x, y1, color="blue", lw=1.0,
linestyle="solid", marker=".")
plt.plot(x, y2, color="red", lw=4.0,
linestyle="dotted")

plt.show()

exit(0)
```

Plotten (2)

```
import pandas
import matplotlib.pyplot as plt
import sys

D = pandas.read_csv(sys.stdin, sep=" ",
header=None, index_col=0)
D.plot(kind="bar", rot=30, legend=False)
plt.title("Mijn plot")
plt.show()
exit(0)
```

Plotten naar een bestand

- Om de plot op te slaan in een bestand:
  - Vervang `plt.show()`
  - met `plt.savefig("mijnplot.pdf")`

Python Modules

- Er zijn al zeer veel Python modules geschreven ...
  - Zip & TAR files lezen.
  - Database toegang (SQL)
  - Internet modules: e-mail, HTTP, FTP
  - Website frameworks
  - UNIX / Mac / Windows specifieke modules
  - Verschillende packages voor wetenschappelijke disciplines
  - Enz ...

Zelf modules schrijven

- Het is eenvoudig om zelf Python modules te schrijven.
  - Maak een ".py" file met classes, functies. Deze kun je vervolgens vanuit dit bestand importeren.

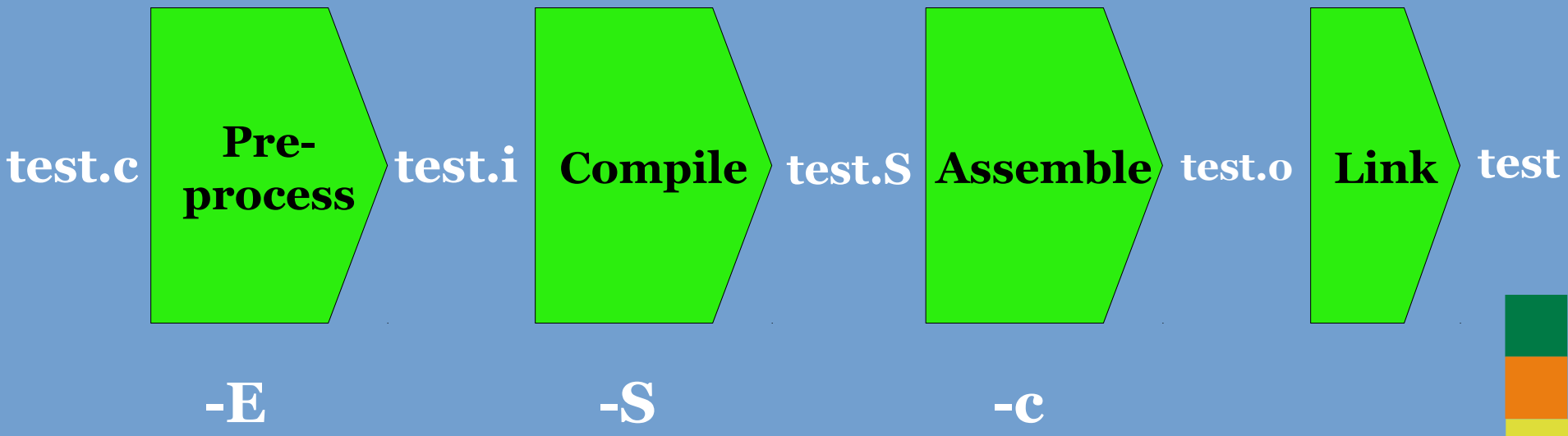
- Wat nu als je al C code hebt die je wilt gebruiken?
- Dit is ook geen probleem: er is dan een "language binding" nodig.
- "Glue code": Python en C code aan elkaar "lijmen".
- Zelf schrijven met Python API, of genereren.
  - SWIG, PyBindGen, Boost::python.

Wat kan nog meer in Python?

- Functies, klassen, methoden, inheritance, operator overloading, …
- Generators, list comprehension, lambda functies, exceptions, …
- Meer leren: Python Tutorial: https://docs.python.org/2/tutorial/
- Of wellicht dictaat Python voor Natuur- en Sterrenkundigen: http://liacs.leidenuniv.nl/~rietveldkfd/courses/pmpy2015/

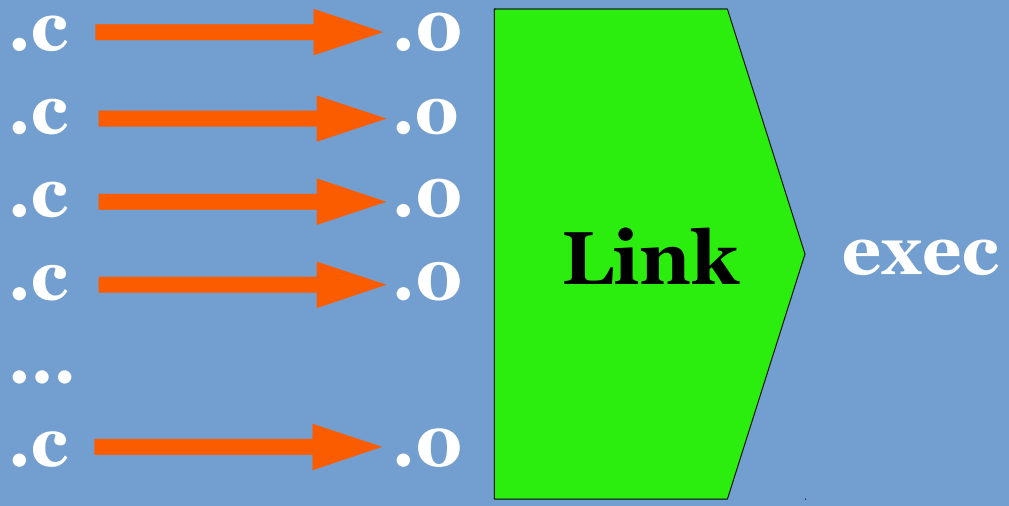
Stappen in softwarecompilatie

- Wat gebeurt er wanneer we een simpel programma compileren?
- Normaal gesproken worden alle stappen doorlopen.



Meerdere source files

- Wanneer programma's groter worden, zal de source code in meer dan één “source file” worden geplaatst.



- Meerdere .c files worden gecombineerd in één executable.

Header files

- Header files zijn nodig zodra programma's gaan bestaan uit meerdere bestanden.
- Om een functie welke is gedefinieerd in een ander bestand aan te roepen, hebben we een declaratie nodig.
- Om de declaratie niet in meerdere bestanden te herhalen, zetten we deze in een header file.

Voorbeeld

module1.h:

```
struct Obj { int a, b, c };
int telop(int a, int b);
void doeiets(struct Obj *o);
```

module1.c:

```
#include “module1.h”

int telop(int a, int b)
{
  return a + b;
}

void doeiets(struct Obj *o)
{
  o->a = 16;
}
```

test.c:

```
#include “module1.h”

int main(void)
{
   int c = telop(245, 3);

   struct Obj o;
   doeiets(&o);

   return 0;
}
```

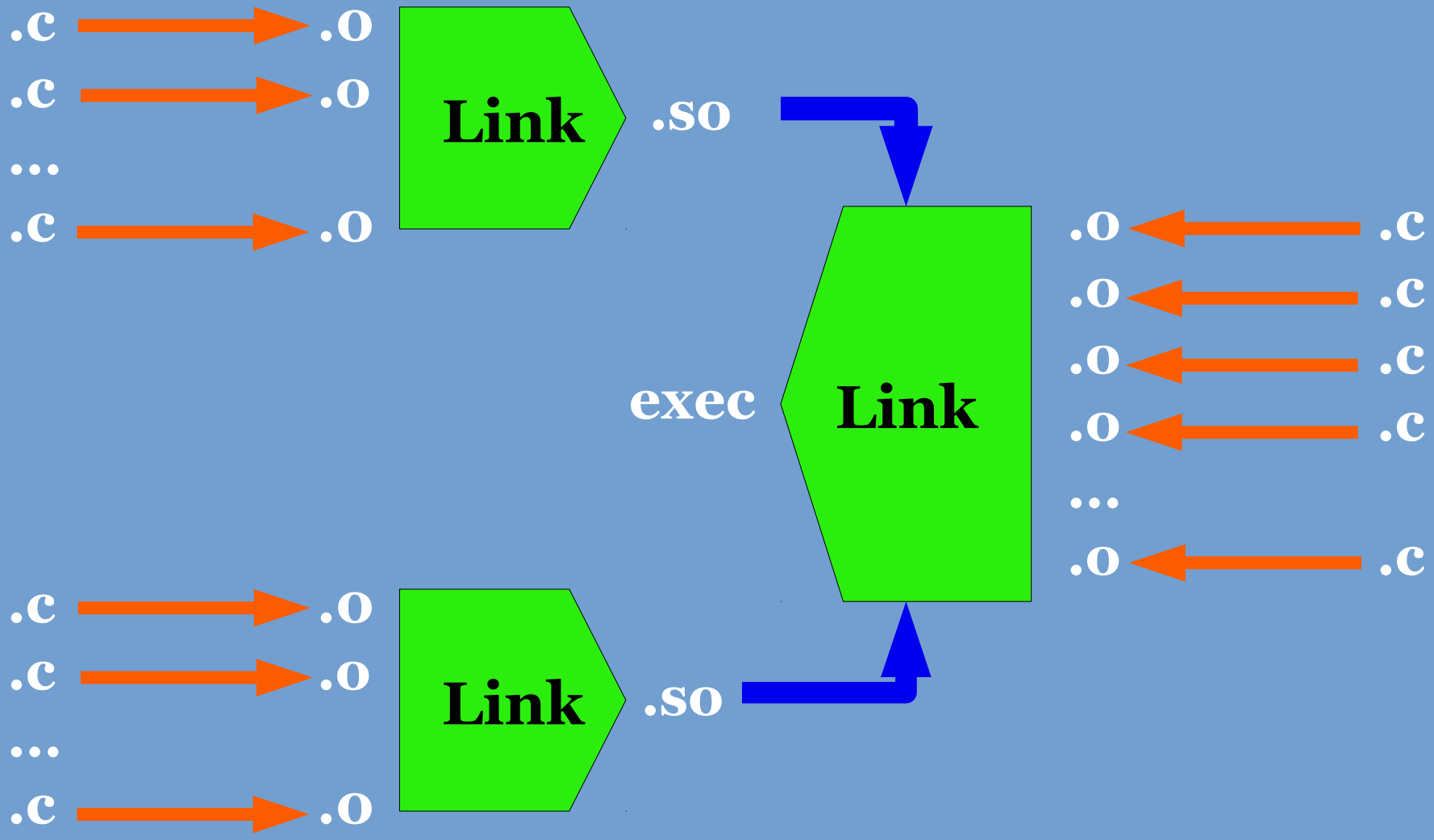
Libraries

- Als software nog verder groeit, komen we tot een punt waarop we niet alle source code meer in een enkele executable linken.
- Waarom doen we dat niet meer?
  - (1) Dit zou betekenen dat bij elke verandering, de gehele executable opnieuw moet worden gelinkt.
  - (2) Er zijn meerdere executables die dezelfde code hergebruiken. We zouden dan de code dubbel opslaan.
  - Voorbeeld: GUI library.

Libraries (2)

- Oplossing: code die wordt hergebruikt in meerdere programma's, wordt opgeslagen in een aparte module. Dit soort modules noemen we "libraries".
- Voorbeelden:
  - C library (printf, puts, fopen, ...).
  - C++ standard library (cin, cout, ...).
  - Image I/O libraries.
  - GUI libraries.
  - Zie /lib, /usr/lib ...
- Executables worden gelinkt tegen een library.

Libraries (3)



Static vs. dynamic libraries

- Static library: is een collectie object-bestanden die *in* een executable moeten worden gelinkt tijdens het compileren.
  - Op UNIX vaak "archive files" (.a).
- Dynamic library: is een losstaande module die kan worden gedeeld (shared object, ".so", ".dylib" op Mac).
  - Windows DLL: Dynamic-Link Library.
  - Het echte linken vindt pas plaats tijdens run-time.
  - De "dynamic linker" gaat op dat moment op zoek naar de libraries en dan vindt het echte linken plaats.
  - De dynamic library mag worden veranderd onafhankelijk van de executable, zolang de library "ABI compatible" blijft.
  - (Waarom? Zie volgende week.)

Gebruiken van libraries

- Om een bestaande library te kunnen gebruiken, moeten we ...
  - ... weten welke functies erin zijn gedefinieerd (namen).
  - ... weten welke parameters deze functies verwachten.
  - API: Application Programming Interface.
- Net als bij een programma bestaande uit meerdere source files, hebben we header files met declaraties nodig.
- Standaardlocatie header files: /usr/include.

Linken met bestaande libraries

- We hebben een aantal compileropties nodig om dit voor elkaar te krijgen, vooral als een library *niet* op de standaard locatie (/usr) is geinstalleerd.
- **"-I"** geeft de locatie op waar de compiler naar header files moet zoeken. Nodig tijdens de compilatiefase (.c -> .o).
- **"-L"** geeft de locatie op waar de compiler naar libraries moet zoeken.
- **"-l"** geeft de naam op van de library die moet worden gelinkt.
  - De naam is voldoende, de compiler zet er zelf "lib" voor en ".so" achter.
  - Bijv. "-lgtk-3" zorgt ervoor dat "libgtk-3.so.0" wordt gevonden.

Voorbeeld

- We hebben "libcool" geinstalleerd in /home/pt2016/prefix.
  - Header files (cool.h): /home/pt2016/prefix/include.
  - Library libcool.so: /home/pt2016/prefix/lib.

```
gcc -Wall -I/home/pt2016/prefix/include \
-L/home/pt2016/include/lib \
-o cooltest cooltest.c -lcool
```

Voorbeeld

- Wanneer software tegen meerdere libraries linkt, loopt dit heel snel uit de hand …

Voorbeeld

- Wanneer software tegen meerdere libraries linkt, loopt dit heel snel uit de hand ...

```
gcc -DHAVE_CONFIG_H -I. -I../..  -DG_LOG_DOMAIN=\"Gimp-Config\"
-DGIMP_APP_VERSION_STRING=\"2.9\" -DDATADIR=\""/source/gimp/prefix/share"\"
-I../.. -I../.. -I../../app -I../../app -pthread
-I/source/gimp/prefix/include/gio-unix-2.0/ -I/source/gimp/prefix/include/glib-
2.0 -I/source/gimp/prefix/lib/glib-2.0/include    -pthread
-I/source/gimp/prefix/include/gegl-0.3 -I/source/gimp/prefix/include/json-glib-
1.0 -I/source/gimp/prefix/include/gio-unix-2.0/
-I/data/source/gimp/prefix/include/babl-0.1 -I/source/gimp/prefix/include/glib-
2.0 -I/source/gimp/prefix/lib/glib-2.0/include
-I/source/gimp/prefix/include/cairo -I/source/gimp/prefix/include/glib-2.0
-I/source/gimp/prefix/lib/glib-2.0/include -I/source/gimp/prefix/include/pixman-
1 -I/usr/include/freetype2 -I/usr/include/libpng12   -pthread
-I/source/gimp/prefix/include/gdk-pixbuf-2.0 -I/source/gimp/prefix/include/glib-
2.0 -I/source/gimp/prefix/lib/glib-2.0/include -I/usr/include/libpng12
-I/source/gimp/prefix/include -DGIMP_DISABLE_DEPRECATED
-DBABL_DISABLE_DEPRECATED -DGSEAL_ENABLE -DG_DISABLE_DEPRECATED
-DGDK_DISABLE_DEPRECATED -DGTK_DISABLE_DEPRECATED -DPANGO_DISABLE_DEPRECATED
-DGDK_MULTIHEAD_SAFE -DGTK_MULTIHEAD_SAFE  -g -O2 -Wall -Wdeclaration-after-
statement -Wmissing-prototypes -Werror=missing-prototypes -Wstrict-prototypes
-Wmissing-declarations -Winit-self -Wpointer-arith -Wold-style-definition
-Wmissing-format-attribute -Wformat-security -fno-common -fdiagnostics-show-
option -Wreturn-type   -MT gimprc-deserialize.o -MD -MP -MF .deps/gimprc-
deserialize.Tpo -c -o gimprc-deserialize.o gimprc-deserialize.c
```

Voorbeeld (2)

- Problemen die nu gaan ontstaan:
  - We willen onze software graag op verschillende systemen draaien.
  - Verschillende systemen hebben verschillende versies van de library.
  - En de library is op verschillende locaties geinstalleerd.
  - Enz. Enz.
- Oplossing: configureren voordat we gaan compileren.
- Configuratie management.
- Voor grote software systemen wordt dit *extreem* complex.

Zelf bibliotheken maken

- (1) Compileer object files met optie "-fPIC".
  - PIC: Position-Independent Code.
  - `gcc -Wall -fPIC -c obj1.c`
- (2) Link alle object files in een shared object. Gebruik optie "-shared".
  - `gcc -shared -Wall -fPIC -o libcool.so obj1.o obj2.o obj3.o`
- Vervolgens kan je tegen deze library linken.
  - `gcc -Wall -L/path/to/lib -I/path/to/include \ -o test test.c -lcool`

Dynamic library loading

- Bij het inladen van een programma dat een dependency heeft op (meerdere) shared libraries, moeten al deze shared libraries worden opgezocht.
- "Dynamic linking/loading".
- Wanneer een of meer benodigde libraries niet kunnen worden gevonden, kan het programma niet worden uitgevoerd.

Dynamic library loading (2)

- Belangrijk: er wordt alleen gezocht op standaard locaties!
- Wanneer we een library gebruiken geinstalleerd op een niet-standaard locatie, dan moeten we deze locatie bekend maken aan de loader.
- Environment variabele `LD_LIBRARY_PATH`.

```
export LD_LIBRARY_PATH="/path/to/our/lib"
```

Dynamic library loading (3)

- Met het programma `ldd` kunnen we zien wat voor dependencies een programma heeft en of deze op ons systeem worden gevonden.

```
$ ldd test
    linux-vdso.so.1 =>  (0x00007ffff7df7000)
    libmine.so => not found
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
(0x00007f9073483000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f9073863000)
$ export LD_LIBRARY_PATH="../lib"
$ ldd test
    linux-vdso.so.1 =>  (0x00007ffe4ff41000)
    libmine.so => ../lib/libmine.so (0x00007fa6f4d9a000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
(0x00007fa6f49bc000)
    /lib64/ld-linux-x86-64.so.2 (0x00007fa6f4f9e000)
```

- We gebruiken hier voor het gemak een "relatief" pad, maar geef in principe altijd "absolute" paden op in `LD_LIBRARY_PATH`!!

Dynamic library loading (4)

```
$ ldd /usr/bin/gimp
    linux-vdso.so.1 =>  (0x00007fffefbe4000)
    libgimpwidgets-2.0.so.0 => /usr/lib/libgimpwidgets-2.0.so.0 (0x00007f1785b82000)
    libgimpconfig-2.0.so.0 => /usr/lib/libgimpconfig-2.0.so.0 (0x00007f1785972000)
    libgimpmodule-2.0.so.0 => /usr/lib/libgimpmodule-2.0.so.0 (0x00007f178576c000)
    libgimpcolor-2.0.so.0 => /usr/lib/libgimpcolor-2.0.so.0 (0x00007f178555f000)
    libgimpthumb-2.0.so.0 => /usr/lib/libgimpthumb-2.0.so.0 (0x00007f1785356000)
    libgimpmath-2.0.so.0 => /usr/lib/libgimpmath-2.0.so.0 (0x00007f178514f000)
    libgimpbase-2.0.so.0 => /usr/lib/libgimpbase-2.0.so.0 (0x00007f1784f37000)
    libgtk-x11-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgtk-x11-2.0.so.0
(0x00007f17848fc000)
    libgdk-x11-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgdk-x11-2.0.so.0
(0x00007f1784649000)
    libpangocairo-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libpangocairo-1.0.so.0
(0x00007f178443d000)
    libgdk_pixbuf-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0
(0x00007f178421d000)
    libcairo.so.2 => /usr/lib/x86_64-linux-gnu/libcairo.so.2 (0x00007f1783f5e000)
    libgthread-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgthread-2.0.so.0
(0x00007f1783d5c000)
    libpangoft2-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libpangoft2-1.0.so.0
(0x00007f1783b32000)
    libpango-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libpango-1.0.so.0
(0x00007f17838e8000)
    libfontconfig.so.1 => /usr/lib/x86_64-linux-gnu/libfontconfig.so.1
(0x00007f17836b2000)
    libfreetype.so.6 => /usr/lib/x86_64-linux-gnu/libfreetype.so.6 (0x00007f1783415000)
    libdbus-glib-1.so.2 => /usr/lib/x86_64-linux-gnu/libdbus-glib-1.so.2
(0x00007f17831ee000)
    libdbus-1.so.3 => /lib/x86_64-linux-gnu/libdbus-1.so.3 (0x00007f1782faa000)
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f1782d8d000)
```

Dynamic library loading (5)

```
    libgegl-0.0.so.0 => /usr/lib/libgegl-0.0.so.0 (0x00007f1782b3c000)
    libgio-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgio-2.0.so.0 (0x00007f17827ed000)
    libbabl-0.0.so.0 => /usr/lib/libbabl-0.0.so.0 (0x00007f17825bd000)
    libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f17822c0000)
    libgobject-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0
(0x00007f1782071000)
    libglib-2.0.so.0 => /lib/x86_64-linux-gnu/libglib-2.0.so.0 (0x00007f1781d7c000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f17819bd000)
    libgmodule-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgmodule-2.0.so.0
(0x00007f17817b9000)
    libX11.so.6 => /usr/lib/x86_64-linux-gnu/libX11.so.6 (0x00007f1781483000)
    libXfixes.so.3 => /usr/lib/x86_64-linux-gnu/libXfixes.so.3 (0x00007f178127d000)
    libatk-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libatk-1.0.so.0 (0x00007f178105b000)
    libXext.so.6 => /usr/lib/x86_64-linux-gnu/libXext.so.6 (0x00007f1780e49000)
    libXrender.so.1 => /usr/lib/x86_64-linux-gnu/libXrender.so.1 (0x00007f1780c3f000)
    libXinerama.so.1 => /usr/lib/x86_64-linux-gnu/libXinerama.so.1 (0x00007f1780a3c000)
    libXi.so.6 => /usr/lib/x86_64-linux-gnu/libXi.so.6 (0x00007f178082b000)
    libXrandr.so.2 => /usr/lib/x86_64-linux-gnu/libXrandr.so.2 (0x00007f1780623000)
    libXcursor.so.1 => /usr/lib/x86_64-linux-gnu/libXcursor.so.1 (0x00007f1780419000)
    libXcomposite.so.1 => /usr/lib/x86_64-linux-gnu/libXcomposite.so.1
(0x00007f1780215000)
    libXdamage.so.1 => /usr/lib/x86_64-linux-gnu/libXdamage.so.1 (0x00007f1780012000)
    libpixman-1.so.0 => /usr/lib/x86_64-linux-gnu/libpixman-1.so.0 (0x00007f177fd7a000)
    libpng12.so.0 => /lib/x86_64-linux-gnu/libpng12.so.0 (0x00007f177fb52000)
```

Dynamic library loading (6)

```
    libxcb-shm.so.0 => /usr/lib/x86_64-linux-gnu/libxcb-shm.so.0
(0x00007f177f94f000)
    libxcb-render.so.0 => /usr/lib/x86_64-linux-gnu/libxcb-render.so.0
(0x00007f177f744000)
    libxcb.so.1 => /usr/lib/x86_64-linux-gnu/libxcb.so.1
(0x00007f177f526000)
    libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f177f30f000)
    libexpat.so.1 => /lib/x86_64-linux-gnu/libexpat.so.1
(0x00007f177f0e4000)
    librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007f177eedc000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f1785ecd000)
    libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1
(0x00007f177ecbc000)
    libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2
(0x00007f177eaa0000)
    libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f177e89c000)
    libffi.so.6 => /usr/lib/x86_64-linux-gnu/libffi.so.6
(0x00007f177e693000)
    libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3
(0x00007f177e456000)
    libXau.so.6 => /usr/lib/x86_64-linux-gnu/libXau.so.6
(0x00007f177e252000)
    libXdmcp.so.6 => /usr/lib/x86_64-linux-gnu/libXdmcp.so.6
(0x00007f177e04c000)
```

make & Makefiles

- Make wordt voornamelijk gebruikt om het compileren van software te automatiseren.
- Eigenlijk: het genereren van bestanden gebaseerd op andere bestanden.
- Een bestand wordt alleen gegenereerd als dat nodig is:
  - Het bestand bestaat nog niet, of
  - het gegenereerde bestand is ouder dan de bestanden waarop het is gebaseerd.

make rules

- Een Makefile bestaat uit "rules".
- Deze hebben de volgende vorm:

```
target:    dependencies
           een of meer commando's
```

- Belangrijk! Gebruik tabs en geen spaties.

```
test:      test.c test.h
           gcc -Wall -o test test.c
```

- Om deze regel uit te voeren:

  `make test` (`make <target>`).

Speciale targets

- Vaak wordt er een target genaamd "all" toegevoegd bovenaan de makefile.
- Dit is dan het "default" target.
- Hier wordt aangegeven welke targets moeten worden gegenereerd als je make aanroept zonder argumenten.

```
all:    test

test:   test.c test.h
        gcc -Wall -o test test.c
```

Automatische variabelen

- Er zijn een aantal speciale variabelen die je in make rules kunt gebruiken:
  - `$@` bevat naam van het target.
  - `$<` bevat naam van de eerste dependency.
  - `$^` bevat naam van alle dependencies.

Generieke rules

- Je wilt niet voor elk C-bestand een aparte make rule schrijven.
- Het is mogelijk om een generieke rule te schrijven die wordt gebruikt als er geen expliciete rule is gevonden.
- De automatische variabelen komen hier wel erg goed van pas.

```
%.o:    %.c
        gcc -Wall -g -c $<
```

Variabelen

- Zelf kun je ook variabelen introduceren.
- Vaak wordt dit gebruikt om de compiler flags te specificeren.
- Merk op dat de naam tussen haakjes moet staan bij gebruik van de variabele.

```
CFLAGS = -Wall -g

test:    test.c
         gcc $(CFLAGS) -o $@ $<
```

Een wat completer voorbeeld

```
CFLAGS = -Wall -g
OBJECTS = main.o feature1.o feature2.o

all:    test

test:   $(OBJECTS)
        gcc $(CFLAGS) -o $@ $^

%.o:    %.c
        gcc $(CFLAGS) -c $<
```

Handige make opties

- `-n` print commando's zonder uit te voeren (dry run).
- `-C <dir>` draai make in de opgegeven directory.
- `-j N` draai N jobs tegelijkertijd (nuttig op multi-core machines).

Huiswerk 1

- Huiswerk 1: deadline a.s. vrijdag.
- Mag worden gemaakt in tweetallen.
- Oefenen met zelf maken van shared libraries & schrijven Makefile.

Build systems

- Voor (zeer) grote projecten wordt het zelf schrijven van Makefiles te complex.
- Vooral wanneer je rekening gaat houden met verschillende systemen, library versies en locaties, enz.
- Schrijft iedereen dan zelf Makefiles?

Build systems (2)

- Nee, er bestaan weer tools om Makefiles automatisch te genereren:
  - cmake
  - autotools (autoconf, automake)
  - scons
  - qmake
  - etc.
- Hoe werken dit soort tools in het algemeen?

Build systems (3)

- Met een speciale syntax (afhankelijk van de tool) wordt een bestand geschreven waarin onder andere staat:
  - Nagaan of een geschikte compiler en andere tools zijn geinstalleerd en kunnen worden gevonden.
  - Welke externe dependencies er zijn, hoe deze moeten worden gedetecteerd, wat de minimale versie is en hoe hier tegen moet worden gelinkt.
  - Welke delen source code tot een shared object of executable moeten worden gecompileerd.
  - Hoe deze delen moeten worden gecompileerd (speciale compile flags, dependencies, etc.).

Build Systems (4)

- Waar en hoe de bestanden uiteindelijk moeten worden geinstalleerd. Dit geldt ook voor andere bestanden die deel uitmaken van het programma zoals bijvoorbeeld iconen en dergelijke.
- Enz.

- In feite een totaal overzicht hoe een compleet project moet worden gecompileerd.

Build systems (5)

- Het compileren van een project bestaat vaak uit de volgende stappen:
  - (1) "Configure" fase: detecteren external dependencies.
  - (2) Het compileren van de source code.
  - (3) Het installeren van de resulterende objecten.
- Gezien de tijd slaan we het bestanden van de verschillende build system "syntaxen" over.
- We kijken wel kort naar het gebruik van twee systemen.

autotools

- **GNU autotools**: zeer veel gebruikt voor projecten voor Linux systemen.
- Bestaat uit: automake, autoconf, aclocal, autoheader, ...
- Kenmerk van project dat autotools gebruikt: er is een `configure` script meegeleverd met de source code.
- Te volgen stappen:
  - (1) `./configure --prefix=/path/to/prefix` (standaard `/usr/local`).
  - (2) `make`
  - (3) `make install`

cmake

- Kenmerk van een project dat cmake gebruikt: er is een `CMakeLists.txt` bestand meegeleverd met de source code.
- Te volgen stappen:
  - (1) `cmake .` in de directory van `CMakeLists.txt`.
  - (Eventueel `cmake -DCMAKE_INSTALL_PREFIX=/path/to/prefix/ . `)
  - (2) `make`
  - (3) `make install`

Volgende week

- Relatie C & machine code.
- Pointer arithmetic.
- Stack frames & calling conventions.
- Geheugenallocatie in C.
- Debugging.