



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Hermann Lenhart & Michael Blesel



Dr. Hermann-J. Lenhart

hermann.lenhart@informatik.uni-hamburg.de



Makefile

- Einführung
- Regeln
- Praxis mit Beispielen



Einführung Makefile I

make ist ein Tool, mit dem komplexe Programme auf einfache Weise kompiliert und ausgeführt werden können.

make liest eine Datei mit der Bezeichnung *Makefile* (oder *makefile*), in der die Abhängigkeiten des Übersetzungsprozesses von Programmen formalisiert erfasst sind.

D.h. der Ablauf von Quelldateien -> Objektdaten => Ergebnissen wird über das *Makefile* gesteuert.



Einführung Makefile II

Das *Makefile* wird aufgerufen durch den Befehl

> make

Dabei müssen Abhängigkeiten von Dateien berücksichtigt werden, die eine bestimmte Reihenfolge im Kompilieren und Linken erfordern.

Das Makefile kann darüber hinaus noch weitere Steuerfunktionen für das Programm übernehmen, z.B: Compilereinstellungen sowie die zusätzlichen Aufrufe von MPI oder OpenMP.



Makefile Regeln

Ein *Makefile* hat eine fest vorgeschriebene Struktur:

Target : Voraussetzung
 Kommando

z.B. für das „Hallo World“ Program:

```
run: hello.f90
    f95 -o hello.x hello.o
    ./hello.x
```



Makefile Regeln

Das wichtigste Merkmal vom *Makefile* **ist unsichtbar:**

$\mapsto$ *der Tab!*

Die korrekte Syntax lautet:

run: hello.f90

$\mapsto$ f95 -o hello.x hello.o

$\mapsto$ /hallo.x



Makefile Regeln

D.h. die korrekte Syntax für ein *Makefile* lautet:

Target : Vorraussetzung

⇒ *Kommando*

Der Tab kann nicht durch die entsprechende Anzahl von Leerzeichen ersetzt werden, was optisch erst mal gleich aussieht!

Dies führt häufig zu der Fehlermeldung: *** missing separator



Makefile Programmbeispiele

Beispiel Hallo World Programm

```
program main
    print *, 'Hallo World'
end program main
```

Kompilieren:

```
> f95 -o hallo.x hallo.f90
```

Ausführen:

```
> ./hallo.x
```



Makefile Programmbeispiel A

Beispiel Hallo World Programm

```
program main
    print *, 'Hallo World'
end program main
```

Kompilieren:

```
> f95 -o hallo.x hallo.f90
```

Ausführen:

```
> ./hallo.x
```

Makefile

```
run: hallo.f90
    f95 -o hallo.x hallo.f90
    ./hallo.x
```

Kompilieren:

```
> make
```



Makefile Programmbeispiel A

```
/home/lenhart/LEHRE/PPG-18/Help-Makefile/A-makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/A-makefiles$ ls -alF
total 2
drwxr-xr-x 2 lenhart wr  4 Apr 15 16:22 ./
drwxr-xr-x 9 lenhart wr  9 Apr 11 2018 ../
-rw-r--r-- 1 lenhart wr 56 Apr 12 2018 hallo.f90
-rw-r--r-- 1 lenhart wr 54 Apr 11 2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/A-makefiles$ make
f95 -o hallo.x hallo.f90
./hallo.x
Hallo World
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/A-makefiles$
```

Makefile

```
run: hallo.f90
    f95 -o hallo.x hallo.f90
    ./hallo.x
```

Kompilieren:

> make



Makefile Programmbeispiel A

```
/home/lenhart/LEHRE/PPG-18/Help-Makefile/A-makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/A-makefiles$ ls -alF
total 2
drwxr-xr-x 2 lenhart wr  4 Apr 15 16:22 ./
drwxr-xr-x 9 lenhart wr  9 Apr 11  2018 ../
-rw-r--r-- 1 lenhart wr 56 Apr 12  2018 hallo.f90
-rw-r--r-- 1 lenhart wr 54 Apr 11  2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/A-makefiles$ make
f95 -o hallo.x hallo.f90
./hallo.x
Hallo World
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/A-makefiles$ ls -alF
total 17
drwxr-xr-x 2 lenhart wr  5 Apr 15 16:22 ./
drwxr-xr-x 9 lenhart wr  9 Apr 11  2018 ../
-rw-r--r-- 1 lenhart wr  56 Apr 12  2018 hallo.f90
-rwx----- 1 lenhart wr 12728 Apr 15 16:22 hallo.x*
-rw-r--r-- 1 lenhart wr  54 Apr 11  2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/A-makefiles$
```

Makefile

```
run: hallo.f90
    f95 -o hallo.x hallo.f90
    ./hallo.x
```

Kompilieren:

> make



Makefile Programmbeispiel B

Beispiel Hallo World Programm

```
program main
    print *, 'Hallo World'
end program main
```

Kompilieren:

```
> f95 -o hallo.x hallo.f90
```

Ausführen:

```
> ./hallo.x
```

Makefile

```
hallo.x: hallo.f90
    f95 -o hallo.x hallo.f90

run: hallo.x
    ./hallo.x
```

Kompilieren:

```
> make
```

Ausführen:

```
> make run
```



Makefile Programmbeispiel B

```
/home/lenhart/LEHRE/PPG-18/Help-Makefile/B-makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/B-makefile$ ls -alF
total 2
drwxr-xr-x 2 lenhart wr    4 Apr 15 16:44 ./
drwxr-xr-x 9 lenhart wr    9 Apr 11  2018 ../
-rw-r--r-- 1 lenhart wr   56 Apr 11  2018 hallo.f90
-rw-r--r-- 1 lenhart wr  126 Apr 11  2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/B-makefile$ make
f95 -c hallo.f90
f95 -o hallo.x hallo.o
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/B-makefile$ ls -alF
total 24
drwxr-xr-x 2 lenhart wr    6 Apr 15 16:45 ./
drwxr-xr-x 9 lenhart wr    9 Apr 11  2018 ../
-rw-r--r-- 1 lenhart wr   56 Apr 11  2018 hallo.f90
-rw----- 1 lenhart wr  2224 Apr 15 16:45 hallo.o
-rwx----- 1 lenhart wr 12728 Apr 15 16:45 hallo.x*
-rw-r--r-- 1 lenhart wr   126 Apr 11  2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/B-makefile$ make run
./hallo.x
Hallo World
```

Makefile

```
hallo.x: hallo.f90
        f95 -o hallo.x hallo.f90
```

```
run: hallo.x
        ./hallo.x
```

Kompilieren:

> make

Ausführen:

> make run



Makefile Programmbeispiel C

Beispiel Hallo World Programm

```
program main
    print *, 'Hallo World'
end program main
```

Kompilieren:

```
> f95 -o hallo.x hallo.f90
```

Ausführen:

```
> ./hallo.x
```

Makefile

```
hallo.x: hallo.o
    f95 -o hallo.x hallo.o

hallo.o: hallo.f90
    f95 -c hallo.f90

run: hallo.x
    ./hallo.x

clean:
    rm *.o *.x
```

Kompilieren:

```
> make
```

Ausführen:

```
> make run
```

Aufräumen

```
> make clean
```



Praxis Makefile

Der Gebrauch vom *Makefile* erleichtert die Überarbeitung von Programmen, indem nur die aktuellen Änderungen neu kompiliert werden.

=> Bei größeren Projekten spart dies Zeit beim Testen der Programme.

Erfolgt in einem umfangreichen Programmcode eine Änderung nur in einer Datei, so kann durch den Aufruf von **make** gezielt dieses überarbeitete Teil in ein neues lauffähiges Programm eingebunden werden.

Dies geschieht indem das Programm „make“ die Zeitstempel der Dateien auswertet.



Makefile Programmbeispiel C

```
/home/lenhart/LEHRE/PPG-18/Help-Makefile/C-makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ ls -alF
total 2
drwxr-xr-x 2 lenhart wr    4 Apr 15 16:53 ./
drwxr-xr-x 9 lenhart wr    9 Apr 11 2018 ../
-rw-r--r-- 1 lenhart wr   56 Apr 11 2018 hallo.f90
-rw-r--r-- 1 lenhart wr  126 Apr 11 2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ make
f95 -c hallo.f90
f95 -o hallo.x hallo.o
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ make run
./hallo.x
Hallo World
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ ls -alF
total 24
drwxr-xr-x 2 lenhart wr    6 Apr 15 16:53 ./
drwxr-xr-x 9 lenhart wr    9 Apr 11 2018 ../
-rw-r--r-- 1 lenhart wr   56 Apr 11 2018 hallo.f90
-rw----- 1 lenhart wr  224 Apr 15 16:53 hallo.o
-rwx----- 1 lenhart wr 1272 Apr 15 16:53 hallo.x*
-rw-r--r-- 1 lenhart wr   126 Apr 11 2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ make
make: 'hallo.x' is up to date.
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$
```

Makefile

```
hallo.x: hallo.o
    f95 -o hallo.x hallo.o

hallo.o: hallo.f90
    f95 -c hallo.f90

run: hallo.x
    ./hallo.x

clean:
    rm *.o *.x
```

Kompilieren:

> make

Ausführen:

> make run

Aufräumen

> make clean



```
/home/lenhart/LEHRE/PPG-18/Help-Makefile/C-makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ ls -alF
total 2
drwxr-xr-x 2 lenhart wr    4 Apr 15 16:53 ./
drwxr-xr-x 9 lenhart wr    9 Apr 11 2018 ../
-rw-r--r-- 1 lenhart wr   56 Apr 11 2018 hallo.f90
-rw-r--r-- 1 lenhart wr  126 Apr 11 2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ make
f95 -c hallo.f90
f95 -o hallo.x hallo.o
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ make run
./hallo.x
Hallo World
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ ls -alF
total 24
drwxr-xr-x 2 lenhart wr    6 Apr 15 16:53 ./
drwxr-xr-x 9 lenhart wr    9 Apr 11 2018 ../
-rw-r--r-- 1 lenhart wr   56 Apr 11 2018 hallo.f90
-rw----- 1 lenhart wr  2224 Apr 15 16:53 hallo.o
-rwx----- 1 lenhart wr 12728 Apr 15 16:53 hallo.x*
-rw-r--r-- 1 lenhart wr   126 Apr 11 2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ make
make: 'hallo.x' is up to date.
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ touch hallo.f90
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ ls -alF
total 24
drwxr-xr-x 2 lenhart wr    6 Apr 15 16:53 ./
drwxr-xr-x 9 lenhart wr    9 Apr 11 2018 ../
-rw-r--r-- 1 lenhart wr   56 Apr 15 16:57 hallo.f90
-rw----- 1 lenhart wr  2224 Apr 15 16:53 hallo.o
-rwx----- 1 lenhart wr 12728 Apr 15 16:53 hallo.x*
-rw-r--r-- 1 lenhart wr   126 Apr 11 2018 Makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ make
f95 -c hallo.f90
f95 -o hallo.x hallo.o
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$ make run
./hallo.x
Hallo World
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/C-makefile$
```

Makefile

```
hallo.x: hallo.o
        f95 -o hallo.x hallo.o
```

```
hallo.o: hallo.f90
        f95 -c hallo.f90
```

```
run: hallo.x
        ./hallo.x
```

```
clean:
        rm *.o *.x
```

Kompilieren:

> make

Ausführen:

> make run

Aufräumen

> make clean



Makefile Programmbeispiel D

Komplexes „Hallo World“ Programm

main.f90

```
program main
  print *, 'noch in main'
  call hello
end program main
```

sr_hallo.f90

```
subroutine hello
  use mo_hello
  implicit none

  hello_string = 'Hello World from subroutine'
  write(*,*) hello_string
end subroutine hello
```

mo_hallo.f90

```
module mo_hello
  character(len=30) :: hello_string
end module
```



Makefile Programmbeispiel D

Komplexes „Hallo World“ Programm

main.f90

```
program main
  print *, 'noch in main'
  call hello
end program main
```

sr_hello.f90

```
subroutine hello
  use mo_hello
  implicit none

  hello_string = 'Hello World from subroutine'
  write(*,*) hello_string
end subroutine hello
```

mo_hello.f90

```
module mo_hello
  character(len=30) :: hello_string
end module
```

Makefile

```
main.x: mo_hello.o sr_hello.o main.o
    f95 -o main.x main.o mo_hello.o sr_hello.o

main.o: sr_hello.o main.f90
    f95 -c main.f90

sr_hello.o: mo_hello.o sr_hello.f90
    f95 -c sr_hello.f90

mo_hello.o: mo_hello.f90
    f95 -c mo_hello.f90

run: main.x
    ./main.x

clean:
    rm -f a.out main.x *.o *.mod
```



Makefile Programmbeispiel D

```
/home/lenhart/LEHRE/PPG-18/Help-Makefile/D-makefile
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/D-makefiles$ ls -alF
total 17
drwxr-xr-x 2 lenhart wr   6 Apr 15 17:03 ./
drwxr-xr-x 9 lenhart wr   9 Apr 11 2018 ../
-rw-r--r-- 1 lenhart wr  67 Apr 11 2018 main.f90
-rw-r--r-- 1 lenhart wr 293 Apr 11 2018 Makefile
-rw-r--r-- 1 lenhart wr  62 Apr 11 2018 mo_hello.f90
-rw-r--r-- 1 lenhart wr 139 Apr 11 2018 sr_hello.f90
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/D-makefiles$ make
f95 -c mo_hello.f90
f95 -c sr_hello.f90
f95 -c main.f90
f95 -o main.x main.o mo_hello.o sr_hello.o
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/D-makefiles$ make run
./main.x
noch in main
Hello World from subroutine
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/D-makefiles$ ls -alF
total 76
drwxr-xr-x 2 lenhart wr   11 Apr 15 17:04 ./
drwxr-xr-x 9 lenhart wr   9 Apr 11 2018 ../
-rw-r--r-- 1 lenhart wr   67 Apr 11 2018 main.f90
-rw-r--r-- 1 lenhart wr 2296 Apr 15 17:04 main.o
-rwx-r--r-- 1 lenhart wr 12880 Apr 15 17:04 main.x*
-rw-r--r-- 1 lenhart wr  293 Apr 11 2018 Makefile
-rw-r--r-- 1 lenhart wr   62 Apr 11 2018 mo_hello.f90
-rw-r--r-- 1 lenhart wr 251 Apr 15 17:04 mo_hello.mod
-rw-r--r-- 1 lenhart wr  992 Apr 15 17:04 mo_hello.o
-rw-r--r-- 1 lenhart wr  139 Apr 11 2018 sr_hello.f90
-rw-r--r-- 1 lenhart wr 2264 Apr 15 17:04 sr_hello.o
lenhart@cluster:~/LEHRE/PPG-18/Help-Makefile/D-makefiles$
```

Makefile

```
main.x: mo_hello.o sr_hello.o main.o
       f95 -o main.x main.o mo_hello.o sr_hello.o

main.o: sr_hello.o main.f90
       f95 -c main.f90

sr_hello.o: mo_hello.o sr_hello.f90
          f95 -c sr_hello.f90

mo_hello.o: mo_hello.f90
          f95 -c mo_hello.f90

run: main.x
    ./main.x

clean:
    rm -f a.out main.x *.o *.mod
```



Praxis Makefile

Wichtig dabei ist dass alle Abhängigkeiten (Dependencies)
im *Makefile* abgebildet wurden.

Dazu wird ein „Dendency Tree“ oder „Abhängigkeitsbaum“ ausgewertet.



Praxis Makefile: Darstellung Abhängigkeiten

Einfache Abhängigkeit der Programmteile untereinander

main.f90



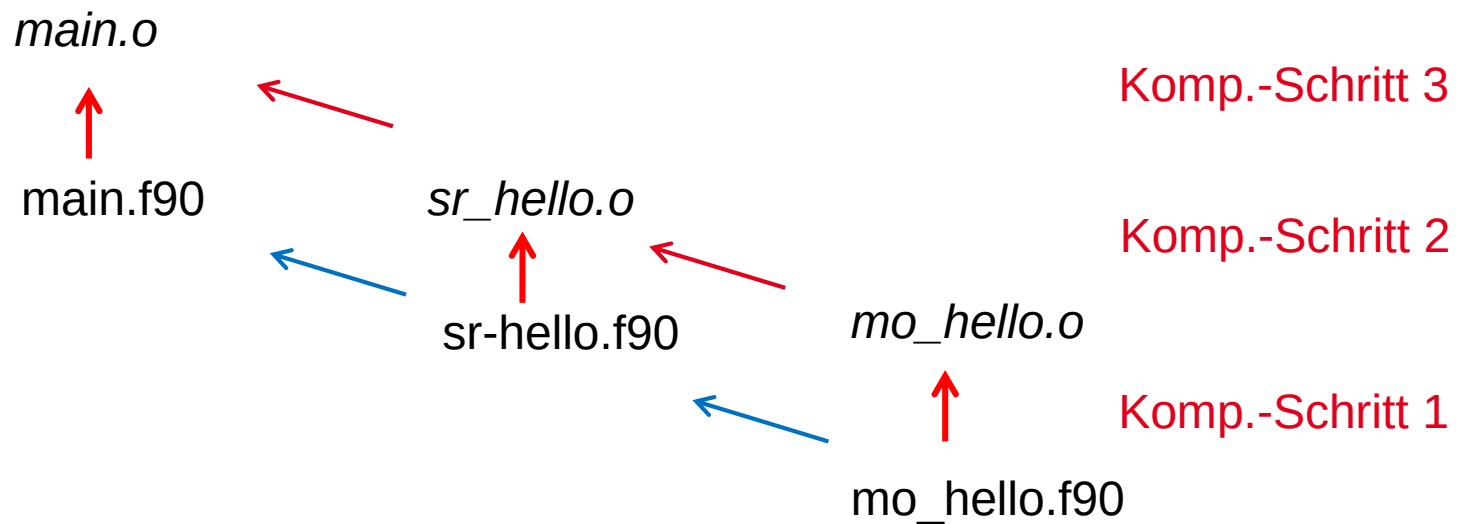
sr-hello.f90



mo_hello.f90



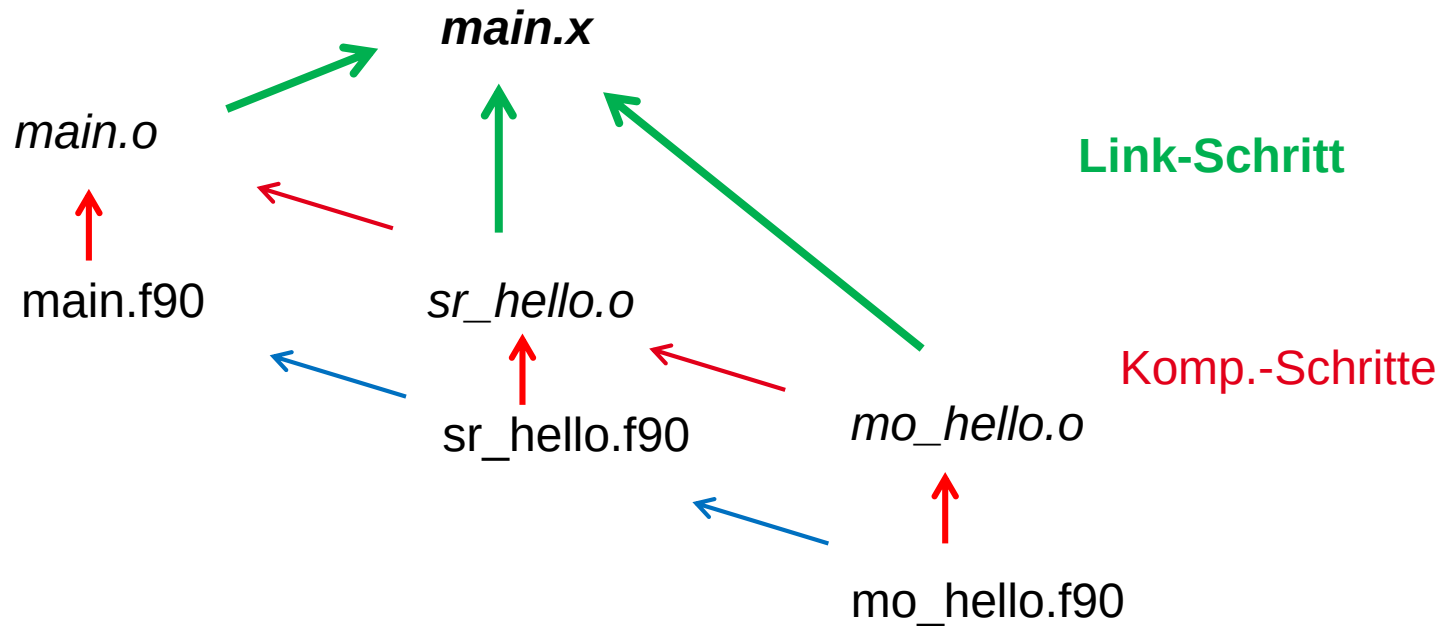
Praxis Makefile: Darstellung Abhängigkeiten



Jedes neu generierte File entspricht einer *Dependency*, d.h. ein Schritt im *Makefile*.



Praxis Makefile: Darstellung Abhängigkeitsbaum (Dependency tree)



Der Link-Schritt entspricht ebenfalls einer *Dependency* im *Makefile*.

Praxis Makefile: Darstellung Abhängigkeitsbaum (Dependency tree)

Makefile

```
main.x: mo_hello.o sr_hello.o main.o
    f95 -o main.x main.o mo_hello.o sr_hello.o

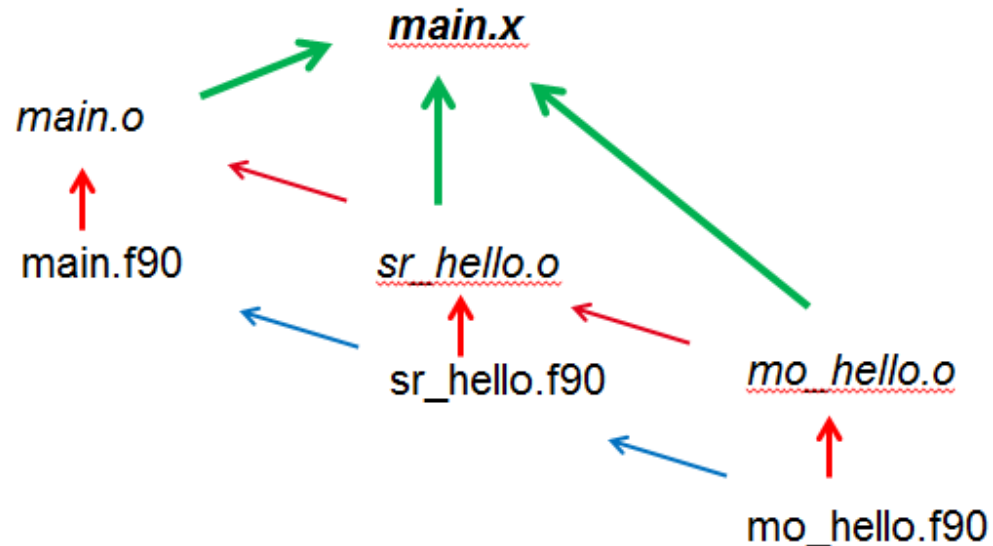
main.o: sr_hello.o main.f90
    f95 -c main.f90

sr_hello.o: mo_hello.o sr_hello.f90
    f95 -c sr_hello.f90

mo_hello.o: mo_hello.f90
    f95 -c mo_hello.f90

run: main.x
    ./main.x

clean:
    rm -f a.out main.x *.o *.mod
```





Makefile Programmbeispiel E

FC=f95

Referenzierung über Variable; hier den Compiler f95

```
main.x: mo_hello.o sr_hello.o main.o
    $(FC) -o main.x main.o mo_hello.o sr_hello.o
```

```
main.o: sr_hello.o main.f90
    $(FC) -c main.f90
```

```
sr_hello.o: mo_hello.o sr_hello.f90
    $(FC) -c sr_hello.f90
```

```
mo_hello.o: mo_hello.f90
    $(FC) -c mo_hello.f90
```

```
run: main.x
    ./main.x
```

```
clean:
    rm -f a.out main.x *.o *.mod
```

Makefile Programmbeispiel F

Referenzierung über Variable;

```
EXECUTABLE = hello.x

#-----
# set default compiler and other
#-----
FC = f95
FFLAGS = -fbounds-check -Wall -Wtabs

#-----
# list of allowed suffixes (first cleared)
#-----
.SUFFIXES:
.SUFFIXES: .f .f90 .o

#-----
# general compilation rules
#-----
.f.o:
    $(FC) -c $(FFLAGS) $<
.f90 .o:
    $(FC) -c $(FFLAGS) $<
```

z.B. weitere Nutzung für Compilerflags & mehr!

```
#-----
# targets
#-----
.PHONY: run clean

$(EXECUTABLE): sr_hello.o main.o
    $(FC) -o $(EXECUTABLE) main.o .....

run: $(EXECUTABLE)
    ./$(EXECUTABLE)

clean:
    rm -f a.out $(EXECUTABLE) *.o *.mod

#-----
# dependencies
#-----
sr_hello.o: mo_hello.o
main.o: sr_hello.o
```



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG



Praxis Makefile: Compiler Switches zur Programmsteuerung

```
.SUFFIXES:
.SUFFIXES: $F $O

#-----
#      compiler switches for program control
#-----
ifeq ($(FORTRAN_COMPILER),IBM)          # for xlf90_r (blizzard)
  DEFINES +=-WF,-DNETCDF
  # DEFINES +=-WF,-DdebugMK,
  DEFINES +=-WF,-Dold_input
  DEFINES +=-WF,-Dold_warmstart
#  DEFINES +=-WF,-Dconvert_warmstart # convert old warmstart to new warmstart
  DEFINES +=-WF,-Dold_restoring
  DEFINES +=-WF,-Drestoring
  DEFINES +=-WF,-Drivers
  DEFINES +=-WF,-Dmeteo
  DEFINES +=-WF,-Dchemie
  DEFINES +=-WF,-Dsediment
  DEFINES +=-WF,-Dbiogeo
#  DEFINES +=-WF,-Deco9
#  DEFINES +=-WF,-Dd093          # compile code according to run D093
#  DEFINES +=-WF,-DNCEP -WF,-DCalcAv_ECO
else # any other compilers are easy to handle ... ?
  DEFINES += -DNETCDF
  # DEFINES += -DdebugMK
  DEFINES += -Dold_input
  DEFINES += -Dold_warmstart
#  DEFINES += -Dconvert_warmstart # convert old warmstart to new warmstart
  DEFINES += -Dold_restoring
  DEFINES += -Drestoring
  DEFINES += -Dd093          # compile code according to run D093
  DEFINES += -Drivers
  DEFINES += -Dmeteo
  DEFINES += -Dchemie
  DEFINES += -Dsediment
  DEFINES += -Dbiogeo
  DEFINES += -DMAIN_SINGLE_STEP
#  DEFINES += -DDEBUG_UK
#  DEFINES += -Deco9
#  DEFINES += -DNCEP -DCalcAv_ECO
endif
#-----
```

```
integer, intent(inout) :: ierr
?
! ?LOCAL VARIABLES:
!
!-----

ierr=0

! initialize modules
#ifdef sediment
  call init_sediment(ierr); if(ierr/=0)return
#endif

#ifdef biogeo
  call init_biogeo(ierr); if(ierr/=0)return
  call update_biogeo(ierr); if(ierr/=0)return
#endif

#ifdef chemie
  call init_chemie(ierr); if(ierr/=0)return
#endif
#ifdef biogeo
  ! import DIC and ALK from statevariable to chemistry module
  ! call update_chemie('put',ierr,st(:, :, : , idic), st(:, :, : , ialk))
#else
  ! transfer DIC and ALK from chemistry module to statevariable
  call update_chemie('get',ierr,st(:, :, : , idic), st(:, :, : , ialk))
#endif
! initialize derived constituents
call do_chemie(1,ierr,st(:, :, : , idic),st(:, :, : , ialk))
if(ierr/=0)return
#endif

call init_output(ierr); if(ierr/=0)return
if(ierr/=0) stop 'init_main #90'

! print*, 'main#82:ch',ch(75,1,32,itdic),ch(75,1,33,itdic)
! print*, 'main#83:st',st(75,1,32,idic),st(75,1,33,idic)
! print*, 'main#83:st',st(69,1,3,ip1c),st(69,1,3,ip2c)
! print*, 'main#83:st',st(jj,kk,ii,in3n),sst(jj,kk,ii,in3n)

end subroutine init_main

!-----
!
! ?INTERFACE:
  subroutine do_main(td,ierr)
  .
```



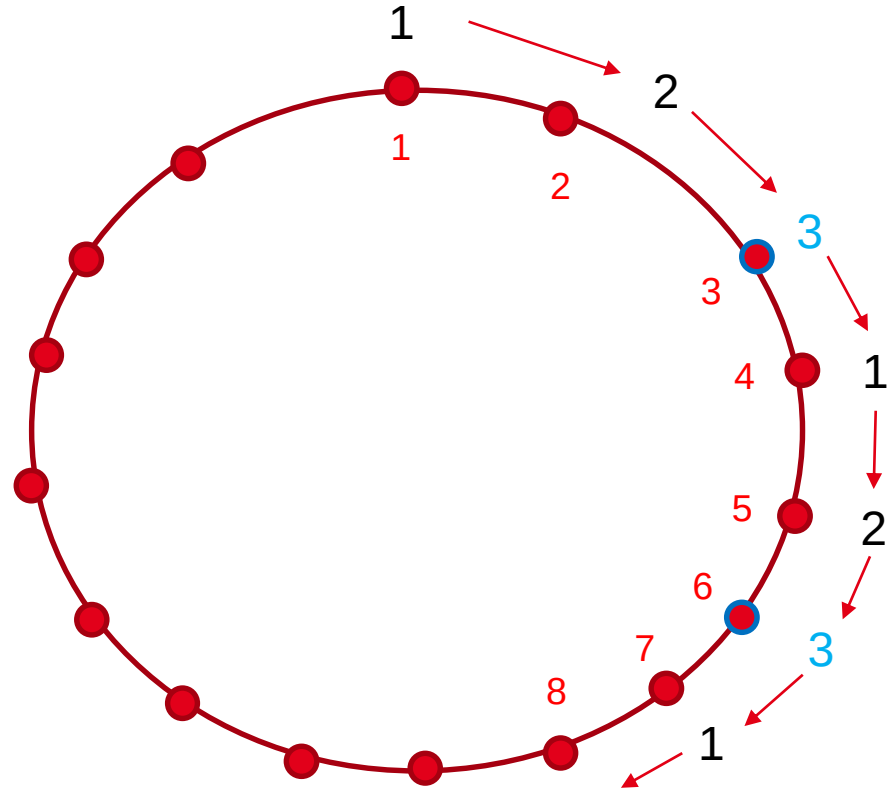
Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG



**Danke,
gibt es noch Fragen?**

Hinweis Aufgabe 1A



Hinweis Aufgabe 1B

```
program main

use mod_initializeField

use mod_lifecycle

use mod_glider

implicit none

logical, dimension (32,22) :: field

integer :: outputUnit = 6

open(outputUnit, encoding='UTF-8')

call createField(field)

call createFigures(field)

call developLife(field)

call printTwoDLogical(outputUnit,field)

end program main
```

mod_initializeField

```
subroutine createField
subroutine createFigures
```

mod_lifecycle

```
subroutine developLife
subroutine countNeighbors
```

mo_glider

```
....
subroutine printTwoDLogical
```



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG



**Danke,
das was.**