Scripting and automation of existing crystallographic software

Santosh Panjikar
Australian Synchrotron

Automation in crystallography

- Automation …
  makes straightforward cases accessible to wider group
  make difficult cases more flexible for expert
  can speed up the process
  can help reduce errors

- Automation also allows you to
  Try more possibilities
  Estimate uncertainties

Requirements for automation of structure determination by X-ray crystallography

- Software carrying out individual steps
- Seamless connections between steps
- A way to decide what is good Strategies for structure determination and decision-making

Objective

- How to run existing software from a script
- How to manage input and output
- How to bridge between existing programs.
- Examples and a discussion

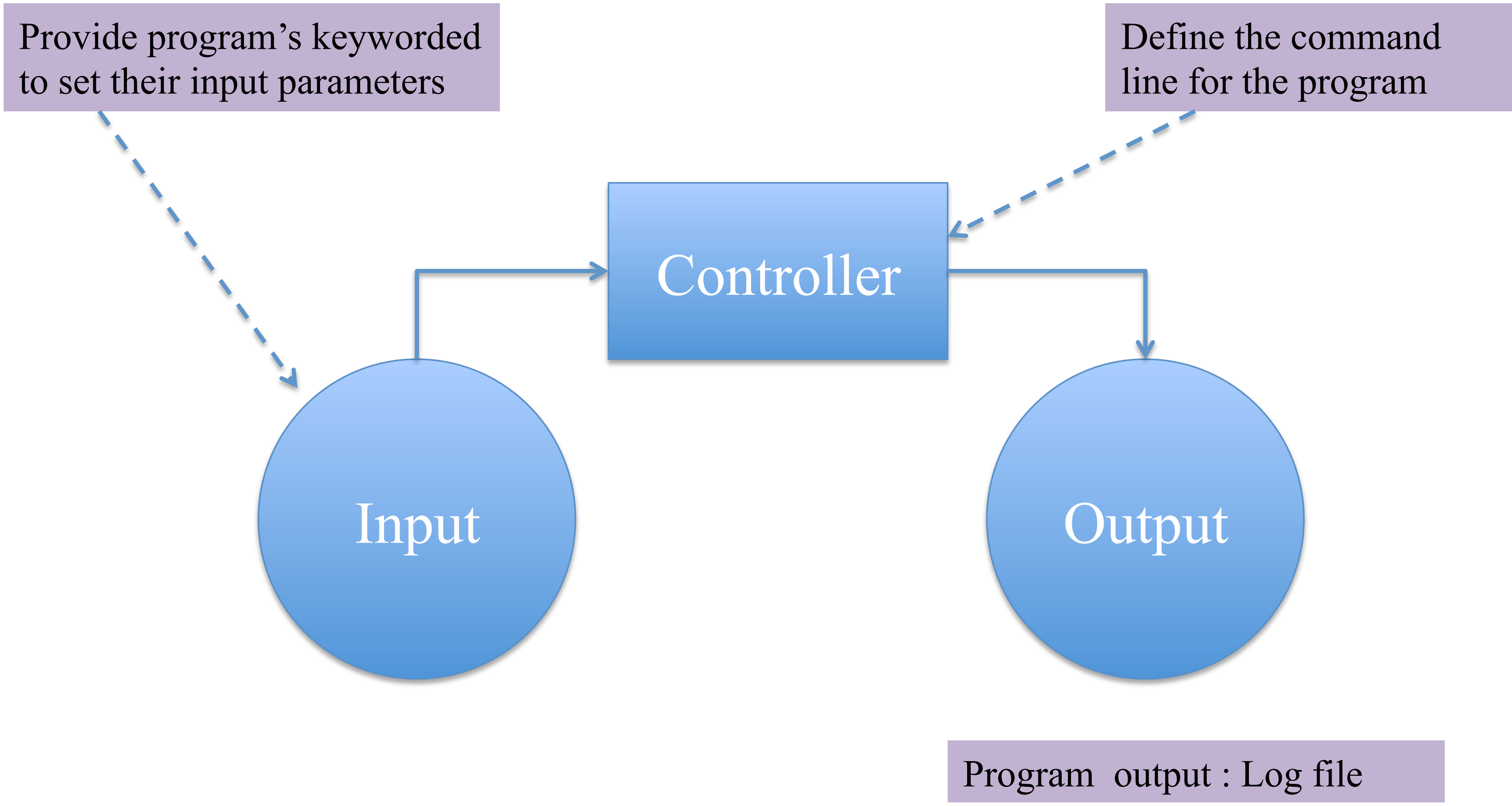
Crystallographic Program

- Usually written in a Fortran or C
- Requires Command line argument
- Requires key parameters to run the program

Running program

```
#first way of getting the data and running
echo “first line of input” > instruct.txt
echo “second line of input” >> instruct.txt
echo “third line of input” >> instruct.txt
program_name < instruct.txt

#Second way of getting the data and running
Program_name << EOF
first line of input
second line of input
third line of input
EOF
```

Necessary components for running crystallographic program


Provide program's keyworded to set their input parameters
Define the command line for the program
Controller
Input
Output
Program output : Log file

Necessary components for running crystallographic program

- Define the command line for the program
- Write a command script
- Execute the command to run program

Command line arguments/file connection

- input and output data files are connected as specified by command line arguments, given after the name of the program to be invoked
- parameters and option specifications are read on the standard input stream

```
<program name> [ <logical name> <file name> ] ...

fft hklin native-refmac5.mtz mapout 2Fo-Fc.map << eof
Key input -1
Key input-2
………….
eof
```

Keyworded input

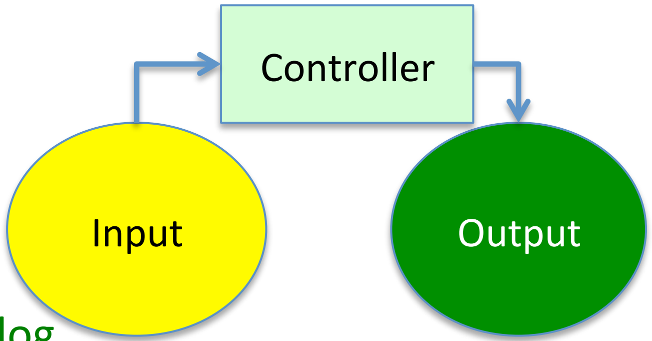
- Most programs take 'keyworded' input to set their parameters.
  keyword= argument_parameter or
  keyword argument _parameter or
  keyword, argument _parameter
  (The detail of the input expected can be found in the documentation for each program).

- Only the first four characters of keywords are significant (although you are recommended to use complete keywords) and they are case-insensitive.

- Records may be continued across line breaks using &, - or \ as the last non-blank, non-comment character on the line to be continued.

- Text following a non-quoted ! or # is treated as a comment and ignored. A continuation character may precede the comment;

An example



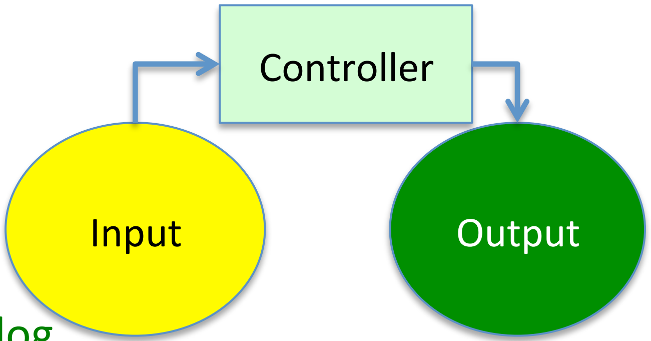
```
truncate HKLIN junk1.mtz HKLOUT junk2.mtz << eof > truncate.log
TITLE
LABOUT F=FP SIGF=SIGFP DANO=DANO SIGDANO=SIGDANO –
          F(+)=F(+) SIGF(+)=SIGF(+) F(-)=F(-) SIGF(-)=SIGF(-)
NOHARVEST
RANGES 60
RESOLUTION 100 2.5
RSCALE 5.5 2.5
NRESIDUE 300
PLOT on
HEADER history
ANOMALOUS yes
TRUNCATE yes
SYMMETRY P212121
CELL 30.00 40.00 50.00 90.00 90.00 90.00
END
eof
```

Continuation of line

Argument_parameter

End of file

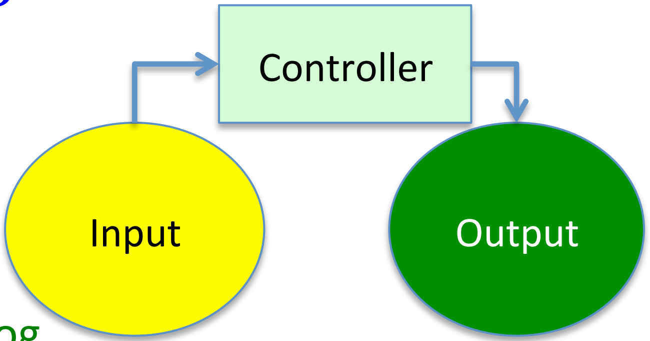
An example



```
truncate HKLIN junk1.mtz HKLOUT junk2.mtz << eof > truncate.log
TITLE
LABOUT F=FP SIGF=SIGFP DANO=DANO SIGDANO=SIGDANO –
             F(+)=F(+) SIGF(+)=SIGF(+) F(-)=F(-) SIGF(-)=SIGF(-)
NOHARVEST
RANGES 60
RESOLUTION 100 2.5
RSCALE 5.5 2.5
NRESIDUE 300
PLOT on
HEADER history
ANOMALOUS yes
TRUNCATE yes
SYMMETRY P212121
CELL 30.00 40.00 50.00 90.00 90.00 90.00
END
eof
```

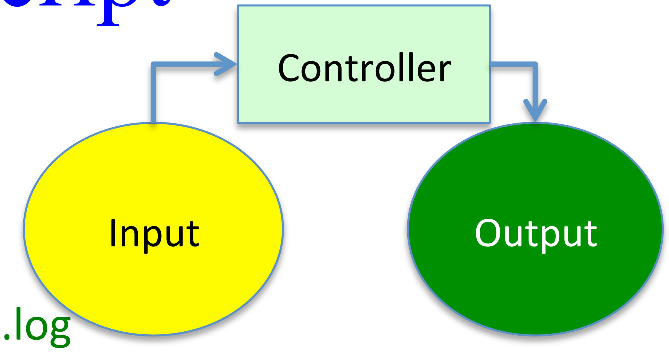
Setting up variables

```
set highres = 2.5
set residue = 300
set cell = “30.00 40.00 50.00 90.00 90.00 90.00”
set spacegroup = P212121

truncate HKLIN junk1.mtz HKLOUT junk2.mtz << eof > truncate.log
TITLE
LABOUT F=FP SIGF=SIGFP DANO=DANO SIGDANO=SIGDANO –
            F(+)=F(+) SIGF(+)=SIGF(+) F(-)=F(-) SIGF(-)=SIGF(-)
NOHARVEST
RANGES 60
RESOLUTION 100 ${highres}
RSCALE 5.5 ${highres}
NRESIDUE $residue
PLOT on
HEADER history
ANOMALOUS yes
TRUNCATE yes
SYMMETRY $spacegroup
CELL $unitcell
END
eof
```



Passing values to the script

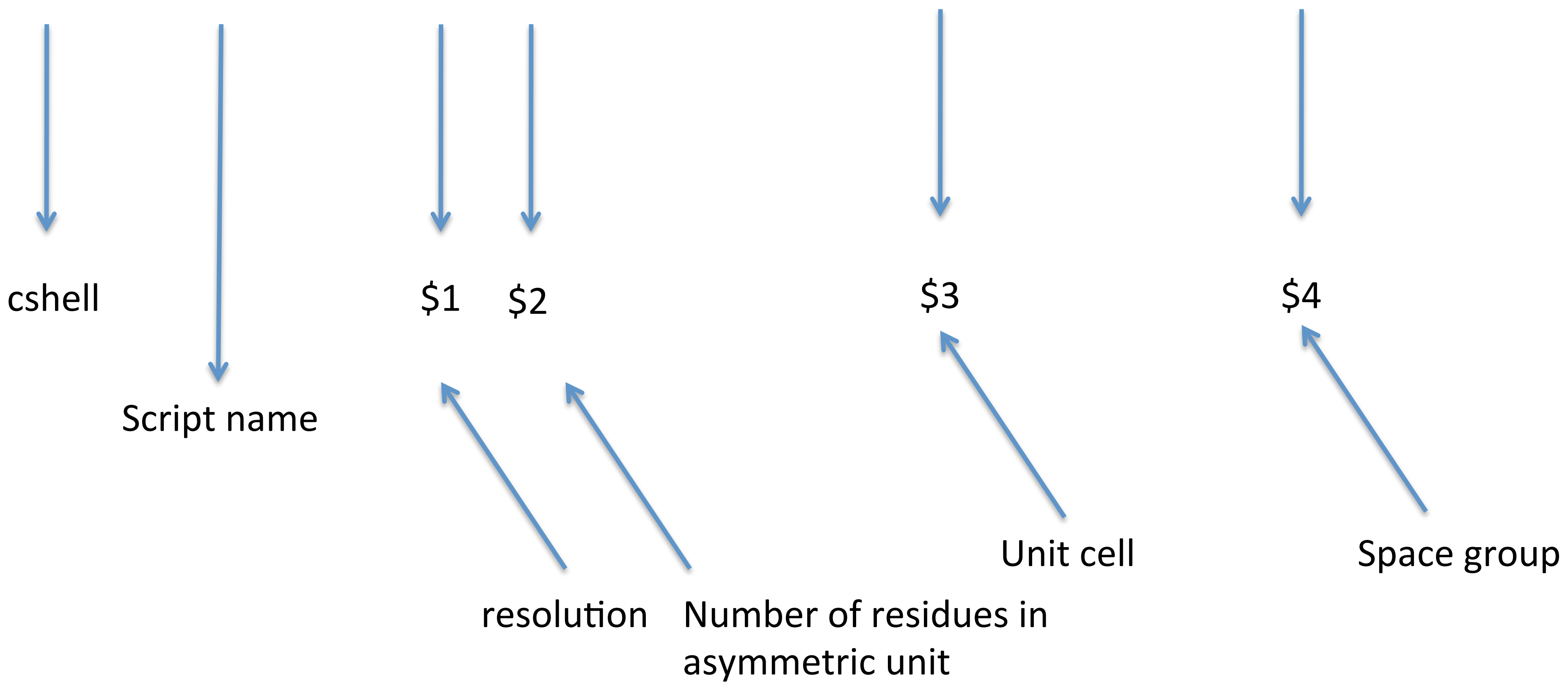
```
set highres = $1
set residue = $2
set cell = $3
set spacegroup = $4

truncate HKLIN junk1.mtz HKLOUT junk2.mtz << eof > truncate.log
TITLE
LABOUT F=FP SIGF=SIGFP DANO=DANO SIGDANO=SIGDANO –
            F(+)=F(+) SIGF(+)=SIGF(+) F(-)=F(-) SIGF(-)=SIGF(-)
NOHARVEST
RANGES 60
RESOLUTION 100 ${highres}
RSCALE 5.5 ${highres}
NRESIDUE $residue
PLOT on
HEADER history
ANOMALOUS yes
TRUNCATE yes
SYMMETRY $spacegroup
CELL $unitcell
END
eof
```



- Save the script as a file called "truncate.com"
- make it executable (chmod +x truncate.com)
- csh truncate com 2.5 300 "30.00 40.00 50.00 90.00 90.00 90.00" P212121

Passing values to the script

```
csh truncate.com 2.5 300 “30.00 40.00 50.00 90.00 90.00 90.00” P212121
```



Scripting

Scripting is a way of telling the computer what to do. However, computer can only understand commands to do things if you tell the exactly what to do in a specific code or language.

Scripting Language

A **scripting language** is a programming language that supports the writing of **scripts**, programs written for a software environment that automate the execution of tasks which could alternatively be executed one-by-one by a human operator.

Scripting language

- **Cshell**
- Bash
- Perl
- Java
- Tclsh

- Python
- …….
- ……….

Shell scripts

- The first scripting languages date back to the 1960s. The language was referred to as "job control languages". They were just simple sets of commands, executed to save the human operator the need to enter all of them manually. These files soon developed into "shell scripts". Shell scripts are a collection of commands for the shell, also known as the command line of an operating system.

- Shell scripts are typically used for file manipulations, program execution and text printing.

Writing scripts

- Use editor to write script

  Emacs, vi, nedit, gedit, pico and nano
- Scripts need to be written in as “plain text” (ASCII text)

Writing scripts

"Hello World" shell script

```
#!/bin/csh –f
#
#This is a comment
#
echo "hello world"
```

Save the shell script as "**hello_world.csh**"
In order to make to runnable or executable

```
chmod +x hello_world.csh
```

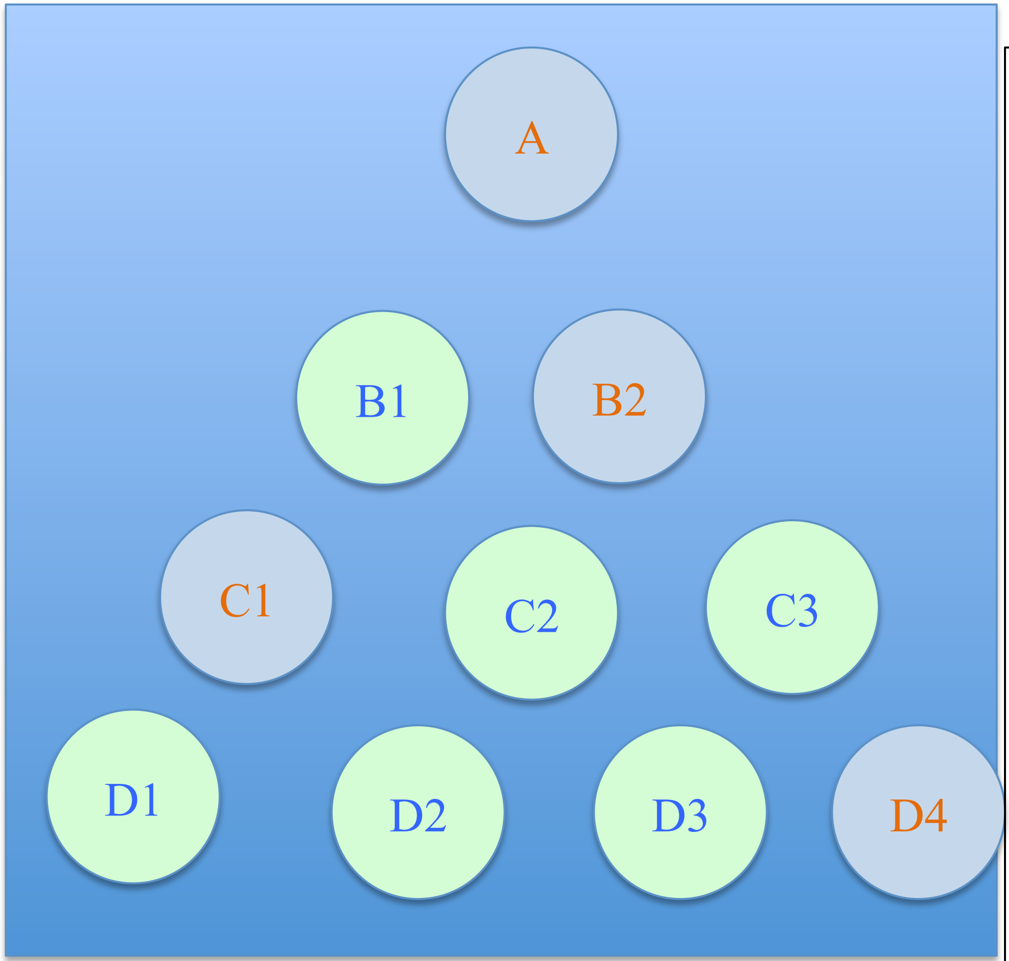
Simple C shell syntax for making decision

```
if (expression) then
...............
endif
```

```
while (expression) then
...............
end
```

```
foreach varname  list
...............
end
# List are enclosed with
   parantheses: (a b c d e f)
```

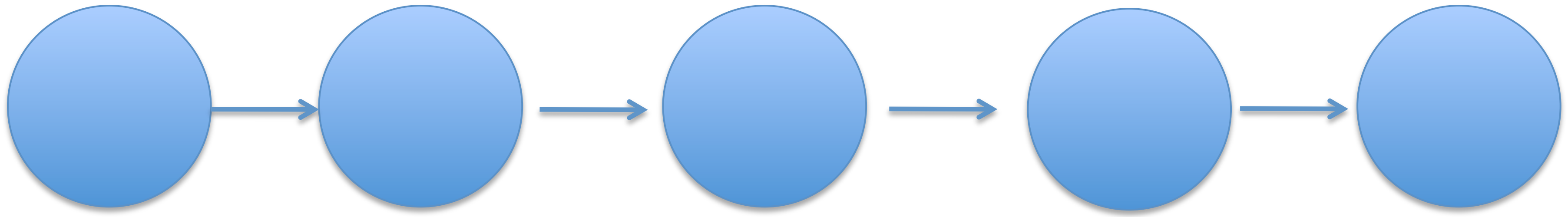
Simple C shell syntax for moving from one part to other part of the script



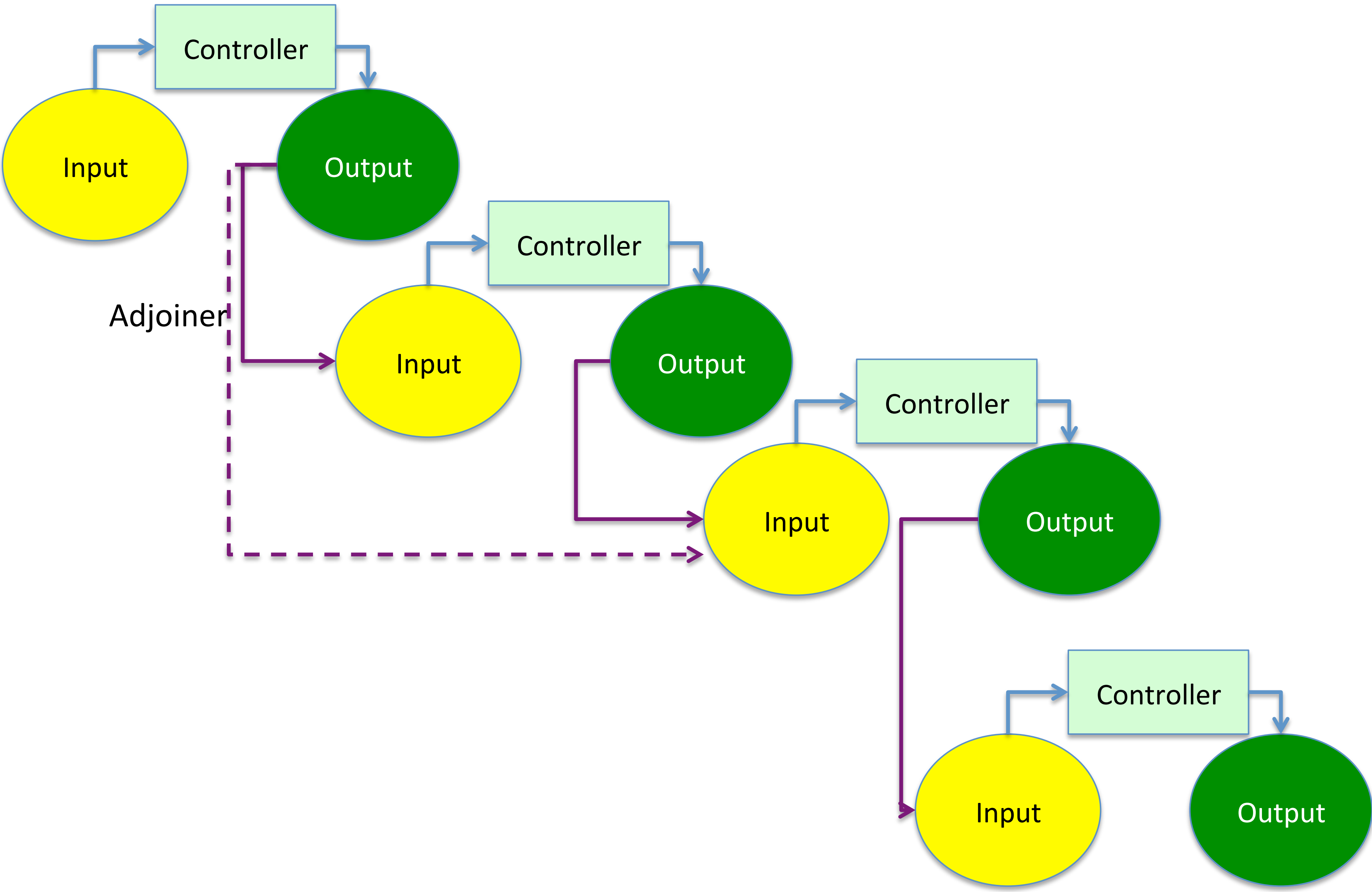
```
# part of script -A
goto B2

D4:
#program script-D4
Exit

C1:
#program script –C1
#logic
goto D4

B2:
#program script-B2
#logic
goto C1
```

Extending the script

- Prepare script for each program
- Determine number of parameters for individual program those change
- Set variable for each changing parameter
- Run the program
- Evaluate the output
- Some parameter values can be extracted for the next program from output of the previous program and pass to the next program in the script



Passing variable parameters and input files to next program


Controller
Input
Output
Adjoiner
Controller
Input
Output
Controller
Input
Output
Controller
Input
Output

A simple example on linking crystallographic software

- We will choose, SHELXC, SHELXD and SHELXE for solving crystal structure from intensity data for phasing method SAD, 2W-MAD and 3W-MAD.
- For this we need to understand what are the input parameters for individual program for various phasing method.
- To run SHELX program: its logical flow is
  SHELXC → SHELXD → SHELXE
- The flow needs to be prepared for each phasing method.

SHELXC, SHELXD and SHELXE

- SHELXC prepares input for SHELXD and SHELXE

  Files generated by SHELXC are with prefix:

  .hkl, _fa.hkl and _fa.ins
- SHELXD uses _fa.hkl (anomalous difference or FA) and _fa.ins (a instruction file) and produces

  _fa.res (fractional heavy atom co-ordinate) and _fa.pdb (Cartesian heavy atom co-ordinate)
- SHELXE uses .hkl, _fa.hkl, _fa.ins and _fa.res

SHELXC

SAD

```
shelxc $PROJECT << eof
SAD  $4
CELL $unitcell
SPAG $SPAG
FIND $HATOMS
NTRY 100
eof
```

2W-MAD

```
shelxc $PROJECT <<
eof
PEAK  $4
INFL   $5
CELL $unitcell
SPAG $SPAG
FIND $HATOMS
NTRY 100
eof
```

3W-MAD

```
shelxc $PROJECT <<
eof
PEAK   $4
INFL     $5
HREM  $6
CELL $unitcell
SPAG $SPAG
FIND $HATOMS
NTRY 100
eof
```

Common Keyword for each phasing protocol are CELL, SPAG, FIND and NTRY. Hence: we will set the parameters value for each keyword and input for the keword SAD, PEAK, INFL, HREM requires intensity data, we will take input from command line of the script.

```
set PROJECT =   my  # define this name as your choice
set unitcell =       # this can be extracted from intensity file (third line of scalepack format)
set SPAG   =         # this is keyword for space group , needs to be given
set HATOMS =    # this is keyword for number of heavy atoms to search, needs to be given
```

SHELXC

SAD

```
shelxc $PROJECT << eof
SAD  $4
CELL $unitcell
SPAG $SPAG
FIND $HATOMS
NTRY 100
eof
```

2W-MAD

```
shelxc $PROJECT <<
eof
PEAK  $4
INFL   $5
CELL $unitcell
SPAG $SPAG
FIND $HATOMS
NTRY 100
eof
```

3W-MAD

```
shelxc $PROJECT <<
eof
PEAK   $4
INFL     $5
HREM  $6
CELL $unitcell
SPAG $SPAG
FIND $HATOMS
NTRY 100
eof
```

```
set PROJECT =   my  # define this name as your choice
set method =     $1  # choose SAD, 2W-MAD or 3W-MAD
set SPAG   =        $2  # this is keyword for space group , needs to be given from command line
set HATOMS =   $3  # this is keyword for no. of heavy atoms to search, needs to be given
set unitcell =     `head -3 $4 | tail -1 | awk '{ print $1, $2, $3, $4, $5, $6}'`
                    # this can be extracted from intensity file (third line of scalepack format)
```

$4, $5, $6 (intensity data) will be taken from script command line input

SHELXD and SHELXE

- To run SHELXD:

```
shelxd  my_fa
```

Program executable

First letters are project name and reads my_fa.hkl and my_fa.ins

- To run SHELXE:

```
shelxe my my_fa –s0.50  -m20 –a4 –q –t2
```

Program executable

First letters are project name and reads my.hkl

-s keyword for solvent content

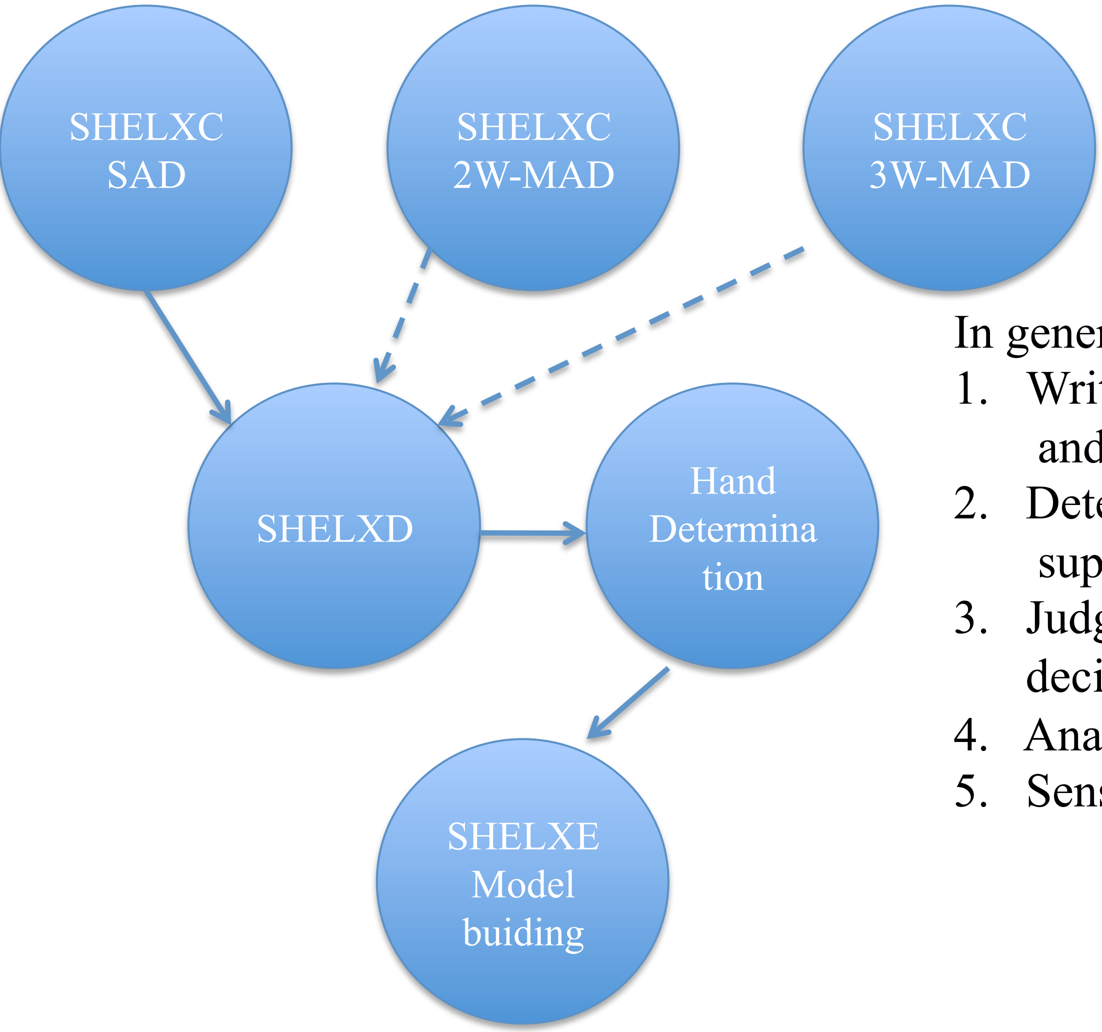
-m keyword for number of cycle

-a keyword for number of building cycle

Automation design for SHELXC/D/E for SAD/2W-MAD/3W-MAD

- SHELXD and SHELXE does not usually require change in the input parameters as the input is going to be similar for any phasing method we choose. [Though input parameters may be changed in difficult cases. Here we like to keep it simple]
- SHELXC inputs will be required to design for each phasing method and then we can pass it to the next step (SHELXD).
- We will need to make decision on the hand of heavy atom sites at the SHELXE step to ensure original or inverse hand is correct.
- Once correct hand is determined, we can pass it to SHELXE density modification and model building step.

Flow chart for the automated script



In general important consideration

1. Write individual script for each program and for each phasing method.
2. Determine keyword parameters to supply the script.
3. Judge which parameters you can make decision to go to the next step
4. Analysis of the output files
5. Sensible error handling message

Tutorial

- **A basic script and test datasets will be supplied to you that would contain the work flow for SAD/2W-MAD/3W-MAD datasets. It will use SHELXC/D/E as external program. We will go through the logic.**
- Your task

1. Run the script using any phasing method (SAD, 2W-MAD or 3W-MAD) and the provided datasets.
2. Extend the script in order to add 4W-MAD phasing protocol and add error handing message when correct number of datasets are not provided. Finally run the protocol.

If you still have time, try following:

3. Terminate SHELXD automatically as soon it finds solution.
4. If SHELXD fails to find solution, add resolution cut-off parameter at the SHELXC step or .ins files so that SHELXD takes further attempt to solve the substructure at lower resolution.

Thank You