An introduction to Fortran

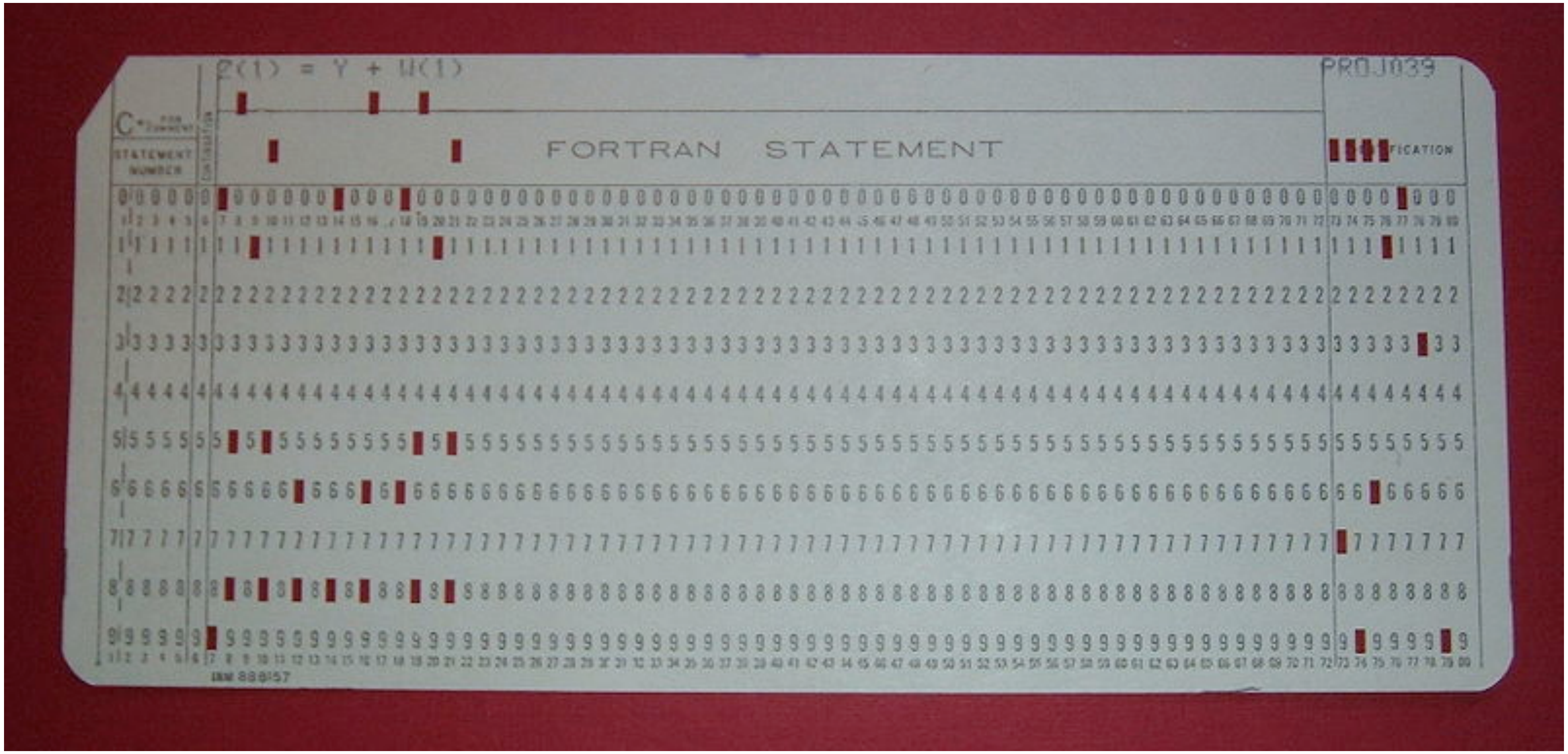
Daniel Price
School of Physics and Astronomy
Monash University
Melbourne, Australia

Part I: Introduction to FORTRAN

A brief history of Fortran (and FORTRAN)

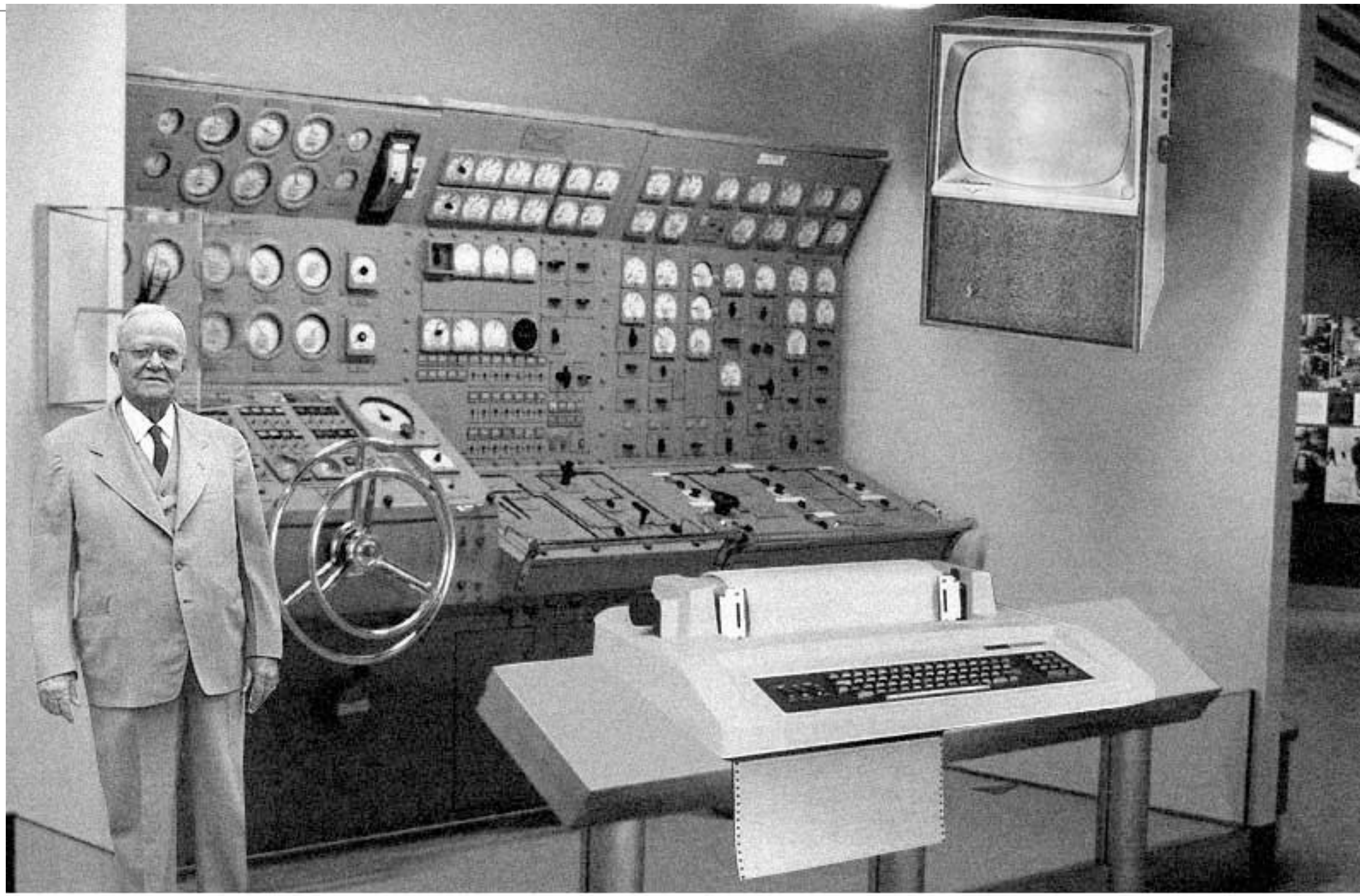
- developed in the 1950's by IBM
- FOR(mula) TRAN(slation): written for doing Maths!
- Prior to FORTRAN, most code was written in assembly language (i.e., machine specific)
- 1961: FORTRAN IV
- 1966: FORTRAN 66
- 1977: FORTRAN 77 standard (now known as FORTRAN).
- 1990: significant new standard, Fortran 90
- 1995: Minor update to Fortran 90
- 2003: Further updates (incl. interface with C)
- 2008: most recent standard, including generic types and co-arrays

Punch cards



http://en.wikipedia.org/wiki/Fortran

The future?



Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use and only

unfortunately a hoax: http://www.snopes.com/inboxer/hoaxes/computer.asp#photo

When should *you* use Fortran?

- Fairly low level, compiled language. So not like matlab “solve ODE”, more like basic Maths, x = y + z; z = sin(x), etc.

- Used commonly for numerical work, e.g. solving ODEs, PDEs. Not for things like writing computer operating systems (C) or scripting (python/perl/unix shell).

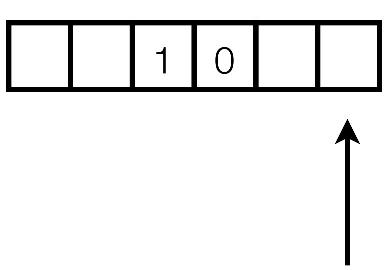
- Modern Fortran is a fully object-oriented language, similar to C++, but designed for solving mathematical problems.

Hello world in FORTRAN

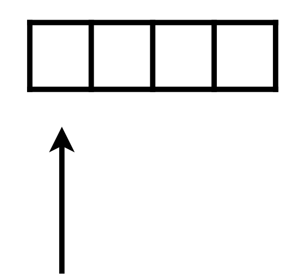
```
program helloworld
implicit none

print*,'hello world'

end
```

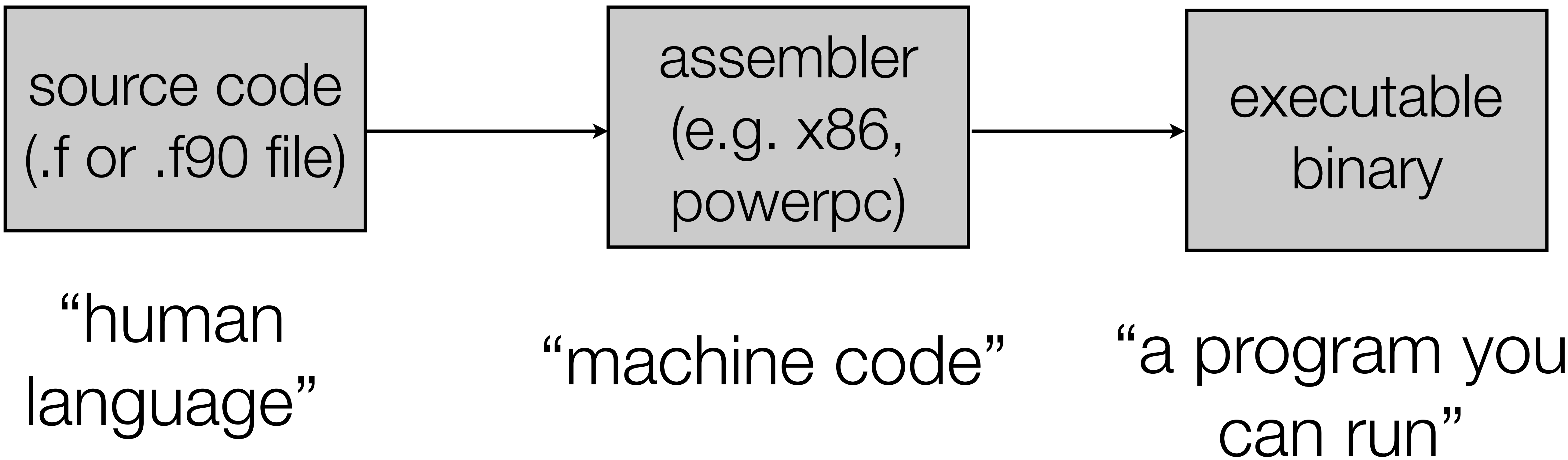


start typing in column 6 (cf. punchcard)



stop typing in column 72

What a compiler does (I)



gfortran -o myprog helloworld.f

to run:

./myprog

What a compiler does (II):

```
program helloworld
implicit none

print*,'hello world'

end
```

```
gfortran -S hello.f90
```

```
        .const
LC0:
        .ascii "hello.f90\0"
LC1:
        .ascii "hello"
        .text
.globl _MAIN__
_MAIN__:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $372, %esp
        call    ___i686.get_pc_thunk.bx
"L00000000001$pb":
        leal    _options.0.1494-"L00000000001$pb"(%ebx), %eax
        movl    %eax, 4(%esp)
        movl    $8, (%esp)
        call    L__gfortran_set_options$stub
        leal    LC0-"L00000000001$pb"(%ebx), %eax
        movl    %eax, -340(%ebp)
        movl    $4, -336(%ebp)
        movl    $128, -348(%ebp)
        movl    $6, -344(%ebp)
        leal    -348(%ebp), %eax
        movl    %eax, (%esp)
        call    L__gfortran_st_write$stub
        movl    $5, 8(%esp)
        leal    LC1-"L00000000001$pb"(%ebx), %eax
        movl    %eax, 4(%esp)
        leal    -348(%ebp), %eax
        movl    %eax, (%esp)
        call    L__gfortran_transfer_character$stub
```

Fortran variable types

```
program variables
implicit none
logical ihavebrain

ihavebrain = .true.
inum = 1
rnum = 1
dnum = 1.0d0

print*,'vars=',ihavebrain,inum,rnum,dnum

end
```

The evils of implicit types

- Implicitly in FORTRAN, undeclared variables starting with a-h and o-h are of type real, and i-n are of type integer.

God is real unless declared integer

Fortran variable types (well written)

```
program variables
implicit none
logical ihavebrain
integer inum
real rnum
double precision dnum

ihavebrain = .true.  ! check if we have a brain
inum = 1             ! number of brain cells
rnum = 1.0           ! fraction of brain cells used
dnum = 0.5d0         ! fraction working now

print*,'vars=',ihavebrain,inum,rnum,dnum

end program variables
```

A bad FORTRAN example (Why you should ALWAYS use "implicit none")

- what does this code do?

```
     program badfort

     do 30 i=1.20
        print*,i
  30 continue

     end
```

```
program badfort
implicit none
integer i

do i=1.20
   print*,i
enddo

end
```

Some basic good practice

- always use “implicit none” to avoid silly mistakes
- add comments to your code as much as possible. These are for YOU so you remember what you did/what you were thinking at the time.
- try to avoid writing the same bit of code more than once: cut and paste is convenient but deadly whilst writing programs! Use a short subroutine or function instead.

Basic maths operations

```
program basicmaths
implicit none
real a,b,c,d,e

a = 1.
b = 2.
c = a + b
d = a*b
e = sqrt(b)

print*,’a=’,a,’ b=’,b,’ c=’,c,’ d=’,d,’ e = ’,e

end program basicmaths
```

Basic maths operations (in double precision)

```
program basicmathsdbl
implicit none
double precision a,b,c,d

a = 1.0d0
b = 2.0d0
c = a + b
d = a*b
e = sqrt(b)

print*,'a=',a,' b=',b,' c=',c,' d=',d,' e = ',e

end program basicmathsdbl
```

Arrays

```
program array1
implicit none
real rnum(3)

rnum(1) = 1.0
rnum(2) = 2.0
rnum(3) = 3.0

print*,'rnum=',rnum

end program array1
```

Arrays II

```
program array2
implicit none
real rnum(3,2)

rnum(1,1) = 1.0
rnum(2,1) = 2.0
rnum(3,1) = 3.0
rnum(1,2) = 4.0
rnum(2,2) = 5.0
rnum(3,2) = 6.0

print*,'rnum=',rnum

end program array2
```

Logical constructs: if-then-else

```
program ifanimal
implicit none
logical :: isacow,hastwohorns
integer, parameter :: nhorns = 2

isacow = .true.
if (isacow) then   ! check if our animal is a cow
   print*,' my animal is a cow...'
   if (nhorns.eq.2) print*,' ...with two horns'
else
   print*,' my animal is not a cow'
endif

end program ifanimal
```

Logical constructs: if-then-elseif

```
isacow = .false.
isadog = .true.
!
!--here we check the type of animal
!   (and the number of horns if it is a cow)
!
if (isacow) then   ! check if our animal is a cow
   print*,' my animal is a cow...'
   if (nhorns.eq.2) print*,' ...with two horns'
elseif (isadog) then   ! or if it is a dog
   print*,' my animal is a dog. Woof.'
else
   print*,' my animal is not a cow or a dog'
endif
```

Fortran loops

```
program loop
implicit none
integer :: i

do i=1,10
    write(*,”(a,i2)”) ‘ number ‘,i
enddo

end program loop
```

```
program loop
implicit none
integer :: i

i = 0
do while (i.lt.10)
    i = i + 1
    write(*,”(a,i2)”) ‘ number ‘,i
enddo

end program loop
```

Formatted print

```
print*,’x=’,x
print “(f6.3)”, x
print “(a,2x,f6.3)”, ’x = ’,x
print “(’ x= ’,f6.3)”,x
    print 10,x
10 format(‘x = ‘,f6.3)
```

Fortran loops: advanced

```
program loop
integer :: i

loop1: do i=1,10
     write(*,"(a,i2)") ' number ',i
     if (i.eq.5) exit loop1
enddo loop1

end program loop
```

Reading and writing to/from the terminal

```
program hello
character(len=20) :: name

print “(’---’,2x,a,2x,’---’)”,’welcome to the hello program’

print*,’ please enter your name’
read(*,*) name

write(*,*) ’hello ’,name
write(6,*) ’I like the name ’//trim(name)
write(*,”(a)”) ’I once had a friend called ’//trim(name)

end program hello
```

Writing to a file

```
program nametofile
character(len=20) :: name
integer :: npets

print*,' please enter your name'
read(*,*) name
print*,' how many pets do you have?'
read(*,*) npets

open(unit=1,file='myname.txt',status='replace')
write(1,*) name
write(1,*) npets
close(unit=1)

end program nametofile
```

Opening a file and reading content

```
program namefromfile
character(len=20) :: name

open(unit=3,file='myname.txt',status='old')
read(3,*) name
read(3,*) npets
close(unit=3)

write(*,*) 'hello ',name
write(*,*) 'I see you have ',npets,' pets'

end program namefromfile
```

Subroutines

```
program callsub
implicit none
real :: x1,y1,z1

x1 = 3.
y1 = 4.
call mysub(x1,y1,z1)
print*,'z1= ',z1

contains

subroutine mysub(x,y,z)
implicit none
real, intent(in) :: x,y
real, intent(out) :: z

z = sqrt(x**2 + y**2)

end subroutine mysub

end program callsub
```

Functions

```
program callfunc
implicit none
real :: x1,y1,z1
real :: zfunc

x1 = 3.
y1 = 4.
z1 = zfunc(x1,y1)
print*,'z1= ',z1

end program callfunc

function zfunc(x,y)
implicit none
real, intent(in) :: x,y
real :: zfunc

zfunc = sqrt(x**2 + y**2)

end function zfunc
```

Part II: A simple FORTRAN primer...

Part III: Advanced Fortran (Fortran 90)

Fortran 90

- files end in .f90
- lines can be longer than 72 characters, do not have to start in column 6
- powerful array notation a = b + c where a, b and c are arrays
- new intrinsic functions e.g., dot_product, trim, matmul
- modules: all subroutines should go in a module that is “used” by the calling routine - allows interfaces to be checked. Modules also replace weird things like COMMON blocks.
- dynamic memory allocation (allocatable arrays) and pointers
- derived data types
- recursive subroutines and functions

Fortran 95

- very minor update to Fortran 90
- where/elsewhere statement
- forall

Fortran 2003

- interoperability with C
- intrinsic functions for getting command line arguments, environment variables etc. (previously these had been compiler extensions)
- Fortran 2003 is fully object oriented.

Fortran 2008

- Co-array fortran for parallel computing

f90 vs f77

```
program xdoty
implicit none
real x(3),y(3),xdoty

x(1) = 1.
x(2) = 1.
x(3) = 1.
y(1) = 0.
y(2) = 0.
y(3) = 3.

xdoty = x(1)*y(1) + x(2)*y(2) + x(3)*y(3)
print*,' xdoty = ',xdoty

end
```

```
program xdoty
 implicit none
 real, dimension(3) :: x,y
 real :: xdoty

 x(:) = 1.
 y(1:2) = 0.
 y(3) = 3.

 xdoty = dot_product(x,y)
 print*,' xdoty = ',xdoty

end program xdoty
```

Logical constructs: select case (Fortran 90)

```
program animalsounds
implicit none
character(len=20) :: myanimal
character*20 :: youranimal

myanimal = ’himalayan yak’
write(*,*,ADVANCE=’NO’) ’my animal says ’

select case(trim(myanimal))
case(’cow’)
     write(*,*) ’moo’
case(’zebra’,’donkey’,’mutated horse’)
     write(*,*) ’a kind of donkey-like braying’
case default
     write(*,*) ’an unspecified non-human sound’
end select

end program animalsounds
```

Modules

```
module circles
 implicit none
 real, parameter :: pi = 3.1415926536

 public :: area
 private

contains
!
!  a function to calculate the area
!
 real function area(r)
  implicit none
  real, intent(in) :: r

  area = pi*r**2    ! area of a circle

 end function area

end module circles
```

Using the module

```
program getarea
 use circles
 implicit none
 logical :: bored

 print*,' pi = ',pi

 bored = .false.
 do while (.not.bored)
    print*,' enter r'
    read*,r
    if (r < 0) then
       bored = .true.
    else
       print*,' the area is ',area(r)
    endif
 enddo

end program getarea
```

```
program getarea
 use circles, only:area
 implicit none
 logical :: bored

 bored = .false.
 do while (.not.bored)
    print*,' enter r'
    read*,r
    if (r < 0) then
       bored = .true.
    else
       print*,' the area is ',area(r)
    endif
 enddo

end program getarea
```

Compiling multiple files

```
gfortran -o myprog myprog.f90 mysub.f90
```

Compiling multiple files (in steps)

```
gfortran -o mysub.o -c mysub.f90
gfortran -o myprog.o -c myprog.f90
gfortran -o myprog mysub.o myprog.o
```

Makefiles

- easy way to compile a program consisting of multiple source files
- just type “make” instead of having to remember all the separate commands
- we will type a simple example together

Fortran 90 Exercise

- write a subroutine that solves (returns all the real roots of) a cubic equation using the exact solution for a cubic.
- put this in a module
- use this module in a program that reads the coefficients for the cubic as input from the user, calls the subroutine you wrote to solve it, and checks the answer.
- use the prompting module provided in the fortran_examples directory to interface with the user
- write a Makefile that will compile your program with the cubic module and the program in different files.

Advanced Fortran 90

- write a second version of your cubic solver subroutine that works on double precision input