

Computers, Programming and Fortran

Dragos B. Chirila

15 December 2009

Simple computer model

- CPU (Central Processing Unit)
- RAM (Random Access Memory)
- HDD (Hard Drive)
- Peripheral Devices
 - Input: Keyboard, Mouse, etc.
 - Output: Graphics card, Monitor, Printer, Sound Card, etc.

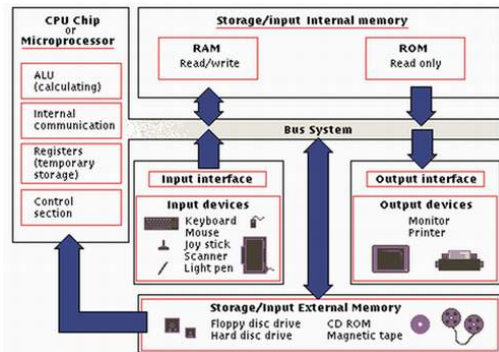


Figure: Schematic of a PC
(<http://www.ajmalbeig.addr.com>)

Life of a CPU

- Fetch instruction
- Fetch arguments
- Compute result
- Store result
- Fetch next instruction...

Uninteresting life, but evolving at a tremendous pace ($\sim 10^9$ operations per second !)

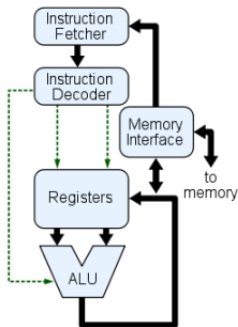


Figure: Life of the CPU
(<http://openbookproject.net>)

- computers only understand binary code (tedious and very verbose for us)

$$(123)_{10} = (1111011)_2$$

- “compressed” using base-16 (hex) → assembler language

$$(123)_{10} = (7B)_{16}$$

- code very difficult to maintain
- higher-level languages appeared
 - *compiled languages*: Fortran, C, C++, Java, Pascal, etc.
 - *interpreted*: Matlab, R, etc.
- convert high-level commands (e.g. `pressure = 100000.0`) to what computers understand (**binary**)
- we only discuss a specific compiled language (Fortran), but *you should also become familiar with an interpreted language*

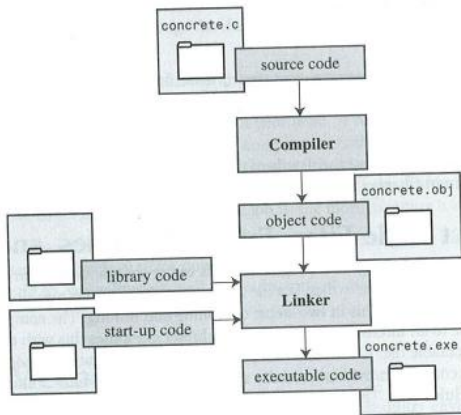


Figure: Compiler and linker (S. Prata, "C Primer Plus")

- **Clarify** the problem to be solved, program input and output
- **Type program** in text editor (not word processor!)
 - Linux: vim/emacs/joe/nano/pico etc.
 - Windows: notepad/vim/SciTE etc.
- **Compile** source code → *object code* (almost there)
- **Call linker**, often automatically called by compiler (adds start-up code and any external libraries used)
- **Execute** program
- **Check** results (hopefully everything is fine)

Errors/Problems will occur for more complex programs, be prepared to repeat steps (programming is more like gardening than construction)

- first high-level language
- continually revised/improved
- flagship of high-performance computing (for the difficult problems)
- recent iterations have most (and some) features of other modern languages
- easy to learn. . .

- you will eventually create a tree of subroutines/functions which solve pieces of the problem
- the tree can be sparse (e.g. only the main program unit) for simple problems
- ... or dense (10s-100s of subroutines - typical climate model)
- (except program-flow elements) instructions can be:
 - declarations (e.g. assign some space for my temperature data)
 - computations (e.g. divide momentum by density)
 - function/subroutine calls (e.g. call the function which averages salinity over a region)

Check your hand-outs for details

Write a program to count up to some value (countMax) and display a message for each counter increment.

```
program simple_do
  implicit none
  integer :: counter, countMax=5

  do counter=1,countMax
    write(*,*) "Counter is", counter, "and counting."
  enddo
end program simple_do
```

Output:

```
Counter is          1 and counting.
Counter is          2 and counting.
Counter is          3 and counting.
Counter is          4 and counting.
Counter is          5 and counting.
```

Wait! When we are at the last value, we are not 'still counting'. An IF-condition fixes that.

```
program simple_do
  implicit none
  integer :: counter, countMax=5

  do counter=1,countMax
    if(counter < countMax) then
      write(*,*) "Counter is", counter, "and counting."
    else
      write(*,*) "Counter is", counter, "and not counting anymore."
    endif
  enddo
end program simple_do
```

Output:

```
Counter is          1 and counting.
Counter is          2 and counting.
Counter is          3 and counting.
Counter is          4 and counting.
Counter is          5 and not counting anymore.
```

Example 1: Matrix Multiplication

```
program matmultiply
  implicit none
  integer, parameter :: N=5      ! matrices will be N x N
  integer, dimension(N,N) :: A,B,C
  integer :: i,j,k

  ! Put some values in matrices
  do i=1,N
    do j=1,N
      A(i,j) = i; B(i,j) = j; C(i,j) = 0
    enddo
  enddo

  ! Compute result matrix
  do i=1,N
    do j=1,N
      do k=1,N
        C(i,j) = C(i,j)+A(i,k)*B(k,j)
      enddo
    enddo
  enddo

  write(*,*) "Result:"
  call printSqrMatrix(C,N)
  C = MATMUL(A,B) ! Compute using intrinsic
  write(*,*) "Result of INTRINSIC FUNCTION:"
  call printSqrMatrix(C,N)

contains

  subroutine printSqrMatrix(mat,matSize)
    implicit none
    integer, dimension(:,,:), intent(IN) :: mat
    integer, intent(IN) :: matSize
    integer :: p
    ! Print the result nicely
    do p=1,matSize
      write(*,*) mat(p,:)
    enddo
  end subroutine printSqrMatrix
end program matmultiply
```

Example 2: Gradient of a scalar field

$$f(x, y) = - \left(\cos(x)^2 + \cos(y)^2 \right)^2$$

```
! Filename: gradient.f90
! Purpose: Compute the gradient of the function
! f(x,y) = -(cos(x)**2+cos(y)**2)**2
program gradient
  implicit none
  real, dimension(:, :), allocatable :: A
  real, dimension(:, :, :), allocatable :: gradA
  real :: dx = 0.1, x, y
  integer :: i, j, N=10

  ! Allocate memory
  allocate(A(-(N+1):N+1, -(N+1):N+1), gradA(-N:N, -N:N, 1:2))

  ! Initialize matrix
  do i=-(N+1), (N+1)
    do j=-(N+1), (N+1)
      x = real(i)*dx; y = real(j)*dx
      A(i, j) = -(cos(x)**2+cos(y)**2)**2
    enddo
  enddo

  ! Open file for writing
  open(UNIT=20, FILE="gradient.dat")
  ! Compute gradient & write to file
  do i=-N, N
    do j=-N, N
      x = real(i)*dx; y = real(j)*dx
      gradA(i, j, 1) = (A(i+1, j)-A(i-1, j))/(2.0*dx)
      gradA(i, j, 2) = (A(i, j+1)-A(i, j-1))/(2.0*dx)
      write(20, *) x, y, gradA(i, j, :), A(i, j)
    enddo
  enddo
  close(20) ! Close file
end program gradient
```

Example 2: Gradient of a scalar field (cont'd)

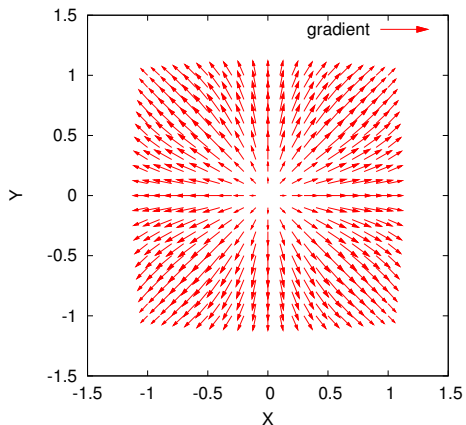


Figure: Gradient of $f(x,y) = -(\cos(x)^2 + \cos(y)^2)^2$

Example 3: Logistic Equation

Logistic ODE

$$\frac{dx}{dt} = ax(1-x)$$

```
program intLogisticEq
  implicit none
  integer :: i, iterMax=50
  double precision :: initial=0.9d0, sol1, sol2, a=-0.1d0, dt=3.0d0
  sol1 = initial ! Both methods are
  sol2 = initial ! initialized with same values
  ! Open output file for writing
  open(20,status='replace',file='LogisticEq.dat')
  do i=0,iterMax ! Time Loop. Real time is i(=no. of iteration)*dt(=timestep)
    write(20,'(4(f12.8))') dble(i)*dt, sol1, sol2, analyticSol(initial,a,i,dt)
    call euler(sol1,a,dt)
    call rk4(sol2,a,dt)
  enddo
  close(20) ! Close datafile
contains
  function f(x,a)
    ! Purpose: Evaluate the derivative. In this case, no time dependence is
    ! present.
    implicit none
    double precision, intent(in) :: x, a
    double precision :: f

    f = a*x*(1.0d0-x)
  end function f

! Program continued on next page...
```

Example 3: Logistic Equation (cont'd)

Logistic ODE

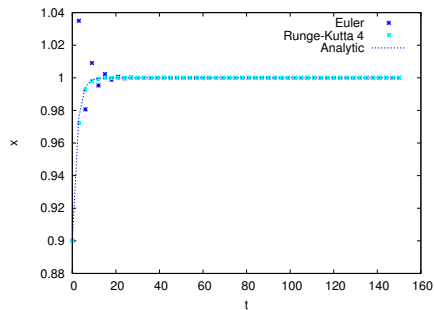
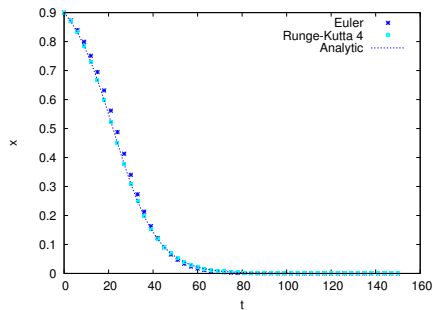
$$\frac{dx}{dt} = ax(1-x)$$

```
! Continuation...
function analyticSol(x0,a,i,dt)
  ! Purpose: evaluate the analytical solution of the ODE (which is known in th
is
  ! case), for evaluation of our numerical methods.
  implicit none
  integer, intent(in) :: i
  double precision, intent(in) :: x0,a,dt
  double precision :: analyticSol
  analyticSol = 1.0d0/(1.0d0+(1.0d0/x0-1.0d0)*exp(-a*i*dt))
end function analyticSol

subroutine euler(x,a,dt)
  ! Simple Euler-forward integration of the ODE
  implicit none
  double precision, intent(inout) :: x
  double precision, intent(in) :: a,dt
  x = x+f(x,a)*dt
end subroutine euler

subroutine rk4(x,a,dt)
  ! 4th-order accurate Runge-Kutta integration of the ODE
  implicit none
  double precision, intent(inout) :: x
  double precision, intent(in) :: a,dt
  double precision :: k1, k2, k3, k4
  k1 = dt*f(x,a) ! For time-dependent derivatives f,
k2 = dt*f(x+k1/2.0d0,a) ! the function call would also include
k3 = dt*f(x+k2/2.0d0,a) ! fractionary steps for time. But here
k4 = dt*f(x+k3,a) ! it is easier.
  x = x+k1/6.0d0+k2/3.0d0+k3/3.0d0+k4/6.0d0
end subroutine rk4
end program intLogisticEq
```

Example 3: Logistic Equation (results)



- **Linux:** use the package manager of your distribution to install gcc (and gfortran) and gnuplot
- **Windows:** see the provided guide

- 1 S.J. Chapman, *Fortran 90/95 for Scientists and Engineers*, 1998
- 2 M. Metcalf, J. Reid, *Fortran 90/95 explained*, 1999
- 3 B.D. Hahn, *Fortran 90 for Scientists & Engineers*, 1996
- 4 S. Prata, *C Primer Plus*, 2004

Thank You!