Advanced features of C and Fortran

• Memory allocation
• Vector operations
• Derived types
• Build-in functions
Dynamical Memory Allocation

C:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *calloc(int num, int size);</td>
<td>This function allocates an array of num elements each of which size in bytes will be size.</td>
</tr>
<tr>
<td>void free(void *address);</td>
<td>This function release a block of memory block specified by address.</td>
</tr>
<tr>
<td>void *malloc(int num);</td>
<td>This function allocates an array of num bytes and leave them initialized.</td>
</tr>
<tr>
<td>void *realloc(void *address, int newsize);</td>
<td>This function re-allocates memory extending it upto newsize.</td>
</tr>
</tbody>
</table>

Fortran:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOCATE(A(N),B(M),...)</td>
<td>This function allocates arrays A, B, ....</td>
</tr>
<tr>
<td>DEALLOCATE(A,B)</td>
<td>This function release a block of memory</td>
</tr>
</tbody>
</table>
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char name[100];
    char *description;

    strcpy(name, "Zara Ali");

    /* allocate memory dynamically */
    description = malloc(30 * sizeof(char));
    if(description == NULL)
    {
        fprintf(stderr, "Error - unable to allocate required memory\n");
    }
    else
    {
        strcpy(description, "Zara ali a DPS student.");
    }

    /* suppose you want to store bigger description */
    description = realloc(description, 100 * sizeof(char));
    ....
    ....
```
Fortran:

```
ALLOCATION (Lst(Np),Label(Nmx:Nbx,Nmy:Nby,Nmz:Nbz))
Call Memory(Np+(Nbx-Nmx+1_8)*(Nby-Nmy+1_8)*(Nbz-Nmz+1_8))
ALLOCATION (Npzero(Nmx:Nbx,Nmy:Nby,Nmz:Nbz))
ALLOCATION (Pzero(Nmx:Nbx,Nmy:Nby,Nmz:Nbz))
Call Memory(2_8*(Nbx-Nmx+1_8)*(Nby-Nmy+1_8)*(Nbz-Nmz+1_8))

Call List
If(nHighRes >= 0)Then
  ALLOCATION(LstHigh(Np))
  Call Memory(INT(Np,8))
  NmxH = 0 ; NbxH = iSplit -1
  NmyH = 0 ; NbyH = iSplit -1
  NmzH = 0 ; NbzH = iSplit -1
  ALLOCATION(LabHigh(NmxH:NbxH,NmyH:NbyH,NmzH:NbzH,nHighRes))
  Call Memory((NbxH-NmxH+1_8)*(NbyH-NmyH+1_8)*(NbzH-NmzH+1_8)*nHighRes)
  Call ListHigh
EndIf
Call Memory(INT(Np,8))
ALLOCATION (dens(Np))
Call DensityAllParticles
  Call Memory(INT(Np,8))
  ALLOCATION (LmaxIndex(Np))
Call FindMaxima
If(nHighRes >= 0)Then
  Call Memory(INT(-Np -SIZE(LabHigh),8))
  DEALLOCATE(LabHigh,LstHigh)
```
Whole Arrays
A *whole array* is a named array; it is either a named constant or a variable. It is referenced by using the array name (without any subscripts).
If a whole array appears in a nonexecutable statement, the statement applies to the entire array. For example:

```
INTEGER, DIMENSION(2:11,3) :: L   ! Specifies the type and
   !   dimensions of array L
```

If a whole array appears in an executable statement, the statement applies to all of the elements in the array. For example:

```
L = 10             ! The value 10 is assigned to all the
   !   elements in array L
WRITE *, L         ! Prints all the elements in array L
```
Array Assignment Statements
Array assignment is permitted when the array expression on the right has the same shape as the array variable on the left, or the expression on the right is a scalar.
If the expression is a scalar, and the variable is an array, the scalar value is assigned to every element of the array.
If the expression is an array, the variable must also be an array. The array element values of the expression are assigned (element by element) to corresponding elements of the array variable. A *many-one array section* is a vector-valued subscript that has two or more elements with the same value. In intrinsic assignment, the variable cannot be a many-one array section because the result of the assignment is undefined.

Examples
In the following example, X and Y are arrays of the same shape:
\[ X = Y \]
The corresponding elements of Y are assigned to those of X element by element; the first element of Y is assigned to the first element of X, and so forth. The processor can perform the element-by-element assignment in any order.
The following example shows a scalar assigned to an array:
\[ B(C+1:N, C) = 0 \]
This sets the elements \( B(C+1,C), B(C+2,C), \ldots B(N,C) \) to zero.
The following example causes the values of the elements of array A to be reversed:
\[ \text{REAL } A(20) \ldots A(1:20) = A(20:1:-1) \]
You can use `typedef` to give a name to user defined data type.

For example you can use `typedef` with `structure` to define a new data type and then use that data type to define `structure` variables directly as follows:

```c
#include <stdio.h>
#include <string.h>

typedef struct Books
{
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
} Book;

int main()
{
    Book book;

    strcpy( book.title, "C Programming");
    strcpy( book.author, "Nuha Ali");
    strcpy( book.subject, "C Programming Tutorial");
    book.book_id = 6495407;

    printf("Book title : %s\n", book.title);
    printf("Book author : %s\n", book.author);
    printf("Book subject : %s\n", book.subject);
    printf("Book book_id : %d\n", book.book_id);

    return 0;
}
```
Define new data types: haloData, haloRecord, haloSnap

```
TYPE :: haloData
  Integer    :: num        ! halo number
  Real       :: x,y,z,vx,vy,vz,aM,Rm,rmsV,circV ! data for current halo
  Integer    :: sel        ! number of selected particles
  Integer*8  :: bound(Nbnd) ! list of bound particles
End TYPE haloData

TYPE :: haloRecord
  Integer    :: nh         ! halo number in catalog
  Integer    :: Iso        ! index of isolation: 0 - distinct, /=0 pointer for parent
  Real       :: x,y,z,vx,vy,vz,aM,Rm,circV
End TYPE haloRecord

TYPE :: haloSnap
  TYPE(haloData) :: halo(Nmax) ! data for current halo
End TYPE haloSnap

Type(haloSnap) :: Snapshot(Nsnaps)
Type(haloRecord) :: Catalog(Nmax), Tree(Ncats,Lmax)
```

‘Catalog’ has Nmax elements each of type haloRecord
Advanced features of C and Fortran

```fortran
read(inc+100,iostat=iStat) j,x,y,z,vx,vy,vz,aM,Rm,rmsV,circV,nSelect,(nbd(k),k=1,nSelect)
  halo2%num = j
  halo2%x = x; halo2%y = y; halo2%z = z
  halo2%vx = vx; halo2%vy = vy; halo2%vz = vz
  halo2%aM = aM; halo2%Rm = Rm
  halo2%rmsV = rmsV; halo2%circV = circV
  halo2%sel = nSelect
  Do k =1,nSelect
    halo2%bound(k) = nbd(k)
  EndDo
  !write(*,'(2i8,3x,i6,10g12.4,3x,i3)')ic,i,Snapshot(ic)%halo(i)%num, &
  !     Snapshot(ic)%halo(i), &
  !     Snapshot(ic)%halo(i)%sel
  Else
    halo2%num = j
    halo2%x = x; halo2%y = y; halo2%z = z
    halo2%vx = vx; halo2%vy = vy; halo2%vz = vz
    halo2%aM = aM; halo2%Rm = Rm
    halo2%rmsV = rmsV; halo2%circV = circV
    halo2%sel = nSelect
    halo2%bound = 0
  EndIf
  Snapshot(icput)%halo(j) = halo2
```
## Useful Fortran build-in functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOT_PRODUCT</td>
<td>The dot product of two rank-one arrays (also called a vector multiply function)</td>
</tr>
<tr>
<td>MATMUL</td>
<td>The result of matrix multiplication (also called a matrix multiply function)</td>
</tr>
<tr>
<td>MAXLOC</td>
<td>The rank-one array that has the location of the maximum element in the argument array</td>
</tr>
<tr>
<td>MAXVAL</td>
<td>The maximum value of the elements in the argument array</td>
</tr>
<tr>
<td>SUM</td>
<td>The sum of the elements of the argument array</td>
</tr>
<tr>
<td>TRANSPOSE</td>
<td>The matrix transpose for the rank-two argument array</td>
</tr>
<tr>
<td>TRIM</td>
<td>The argument with trailing blanks removed</td>
</tr>
<tr>
<td>ADJUSTL</td>
<td>The specified string with leading blanks removed and placed at the end of the string</td>
</tr>
<tr>
<td>ADJUSTR</td>
<td>The specified string with trailing blanks removed and placed at the beginning of the string</td>
</tr>
</tbody>
</table>
Module Nbody ! Not the best code, but a very short one
integer, parameter :: Np =100
Real*8, DIMENSION(3,Np) :: X,V,G
Real*8, DIMENSION(Np) :: Mass
Real*8, Parameter :: eps =1.d-3, eps2=eps**2, dt =1.d-4, t_end =1.
Real*8 :: Ekin, Epot,t=0.
Contains
Subroutine Accel !------------------------------------------
   G = 0.
   Do i=1,Np
     Do j=1,Np
       G(:,i) =G(:,i) +Mass(j)*(X(:,j)-X(:,i))/sqrt( SUM( (X(:,j)-X(:,i))**2+eps2)**3 )
     EndDo
   EndDo
end Subroutine Accel
Subroutine Energy !------------------------------------------
   Epot =0.
   Ekin = 0.5*SUM(Mass*SUM(V**2,1))
   Do i=1,Np-1
     Do j=+1,Np
       Epot = Epot-Mass(i)*Mass(j)/sqrt( SUM( (X(:,j)-X(:,i))**2+eps2 ) )
     EndDo
   EndDo
write(*,'(4(2x,a,g12.5))') 't=',t,' Ekin=',Ekin,' Epot=',Epot,' Etot=',Ekin+Epot
end Subroutine Energy
end Module Nbody
Program MakeltSimple !------------------------------------------
Use Nbody
   Mass =1./Np
   Call random_number(X)
   Call random_number(V)
   X = 2.*X-1.; V = (V-0.5)/3.
   Do
      If(mod(INT(t/dt),100)==0)Call Energy
      Call Accel
      V = V+G*dt ; X = X+V*dt; t = t +dt
      If(t> t_end)exit
   EndDo
End Program MakeltSimple
Tricks and minor things with C and Fortran

- *If* is “cheap”
- Be careful with integers:
  \[ a = 1/3 \times b \]
  gives \( a = 0 \). You should write \( a = 1./3. \times b \)
- Do not write \( a**0.5 \) or \( a**0.5 \) or even \( a**2 \). Use \( \text{sqrt}(a) \) or \( a**2 \) instead – it is 100 times faster.
- Instead of \( a**0.25 \) use \( \text{sqrt}(\text{sqrt}(a)) \).
- Try to rewrite expressions to minimize the number of operations. Example: instead of
  \[ y = a + b \times x + c \times x**2 + d \times x**3 + e \times x**4 \]
  (10 multiplications and 4 additions) use:
  \[ y = a + x \times (b + x \times (c + x \times (d + x \times e))) \]
  (4 multiplications and 4 additions)