Array variables

- To this point, none of the programs have saved the data generated (for the most part)
- Tabulate f(x)

```
DO i = 1, n+1
   x = a + (i-1)h   } compute x, f then do something
   f = ~~~~         } with them, but overwrite them
   }                  in the next pass through the loop.
ENDDO
```

- Very often, the values of x, f need to be used in several different contexts.
- We need a way to save the values like x and f so we can re-use them instead of having to regenerate them.
- The answer to this is array variables (or subscripted variables).
- Instead of

```
DO i = 1, n+1
   x = a + (i-1)*h
   f = ~~~~
ENDDO
```

⟹ we have

```
DO i = 1, n+1
   x(i) = a + (i-1)*h
   f(i) = ~~~~
ENDDO.
```

↑

The variables x and f now contain a whole set of numbers. Each individual number is called an <u>element</u> of the array.

- For example

```
DO i = 1,5
   ia(i) = i**2
ENDDO
```

ia(1) = 1^2 = 1
ia(2) = 2^2 = 4
ia(3) = 3^2 = 9
ia(4) = 4^2 = 16
ia(5) = 5^2 = 25

SYNTAX, DECLARATIONS AND RULES

- Array variables can be <u>static</u> or <u>dynamic</u>

  <u>static</u> - size of array is fixed during compilation and can't change

  <u>dynamic</u> - size of array can change while the program is running.

  We will use static arrays at first since these are easier to use.

- You declare an array variable like any other variable, except a <u>size</u> must be given

```
INTEGER :: ia(5)
REAL(Kind=dp) :: xa(10)
```

→ Declares an integer array variable ia with 5 elements. These are numbered

ia(1), ia(2), ia(3), ia(4), ia(5)

ia(3) = Element number 3.

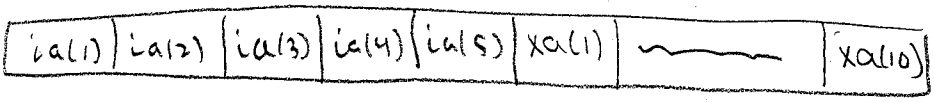
=

Declares a double precision variable xa with 10 elements. These are numbered

xa(1), xa(2), ~~~~, xa(9), xa(10)

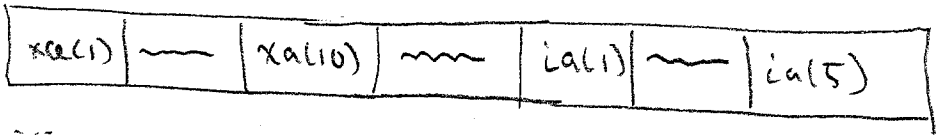
xa(7) = Element number 7

• Storage. The elements of an array must be stored in consecutive memory locations (the program does this for you).



(*)

or



or ....

You don't have much control over where variables are stored in memory.

• Suppose the arrays wind up being stored as in (*) and you do

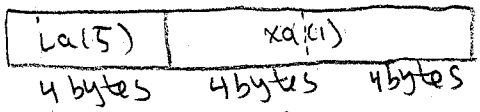
```
DO i = 1, 6
   ia(i) = i**2
ENDDO
```

- This is an <u>array bounds error</u>. This occurs whenever you do one of the following

  * assign array(m) = ~~~~ and m is greater than the declared size for array

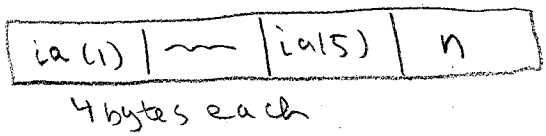
  * set something = something using array(m) and m is greater than the declared size of array

- what happens when you do this? It depends on the compiler.



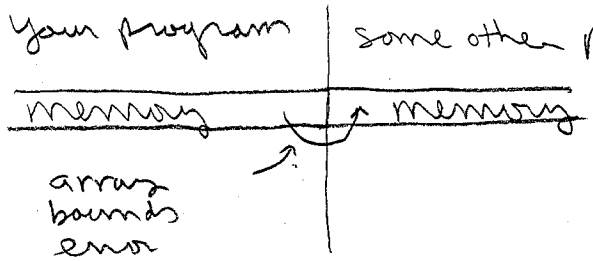
← it is possible that setting ia(6) = 36 will clobber the first bits of xa(1)

- suppose the memory map looked like



if you assign ia(6) = 36, this changes the value of n from 5 to 36.

- array bounds errors are bad and cause unpredictable behavior



big problem in early windows versions

- we will force gfortran to turn on array bounds checking. This will cause the program to halt with a critical error if an array bound error is encountered.

- NOTE: array bounds checking slows down the program. Normally you would design your program with this turned on, then turn it off once you are certain that array errors can't occur.

=

- one advantage of array variables is that it allows decoupling of processes. Instead of having to do everything in one loop, you can separate these.

- Revisit the trap rule program.