2-dim arrays

<type> <name>[m][n]

eg double a[3][4];

As with 1-dim arrays, confusion

declare int x[15]; //x an array of 15 ints

entry x[2] = 5; //assign 5 to the 3rd entry in x

SAMPLE PROGRAM read a 3x4 matrix
and a 4 vector, print them,
calculate their product + print.

```
void multiply( double a[3][4], double b[4],
                        double c[3])
{ int i,j; double sum;
  for (i=0; i<3; ++i)
  {sum=0;
    for(j=0; j<4; ++j)
    { sum += a[i][j] * b[j];}
    c[i] = sum;
  }
}
```

and

```
print_matrix ( double a[3][4])
{ int  i, j;
  for (i = 0;  i<3;  ++i)
  { for (j= 0; j<4;  ++j) ;
    { printf ("␣%8.3f", a[i][j]);
    }
    printf ("\n");
  }
}
```

customised

print_3vec, _4vec :

See notes.

```
int main()
{ ...
  scanf("%d %d", &ell, &m); // a: l x m
  for(i=0; i<ell; ++i)
  for(j=0; j<m; ++j)
  { scanf("%lf", &(a[i][j])); }

  scanf("%d", &n); // b, height n
  for(j=0; j<n; ++j)
  { scanf("%lf", &b[j])
  multiply(a, b, c);
  etcetera
}
```

Array arguments in routines

```
void multiply(double a[3][4],
              double b[4], double c[3]) {...}
main ...
    multiply(a, b, c);
```

The black c is a copy of the red c

YET result is put there. HOW?

answer: array values are addresses.

C is 'call by value',

arrays are passed 'by reference'

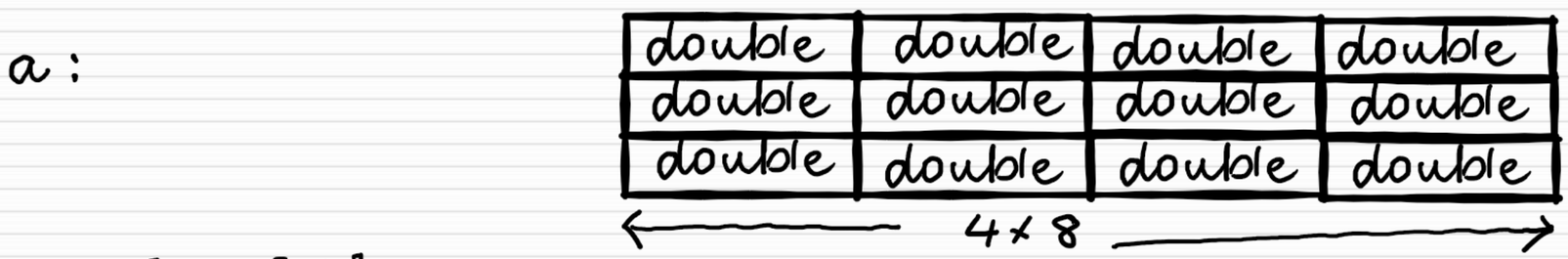
Similarly `scanf("%d", &n);`

int x[15]; double a[3][4]; STORAGE MAPPING

A (1234, say)

x[i] at A+4i

B (1294, say)



a[i][j] at

$$B + 4 \times 8 \times i + 8 \times j$$

x[i] at 1234 + 4i

x[14] at 1290

x[15] 1294 OOR

x[-4] at 1218 OOR

a[i][j] at 1294 + 32i + 8j

a[2][1] at 1294 + 64 + 8

= 1366 etcetera.

Exercise. Solve for i : x[i] has same address as a[2][1]

1234 + 4i = 1366 4i = 132 i = 33. (OOR)

The value of an array variable is the address of its first element.

To calculate the address of an array entry given index i you need only its starting address and the size of its entries.

Similarly a[i][j]

If x, a are arrays,

```
x = a;
```

is an error. x and a are addresses. It would be like

```
1234 = 1294;
```