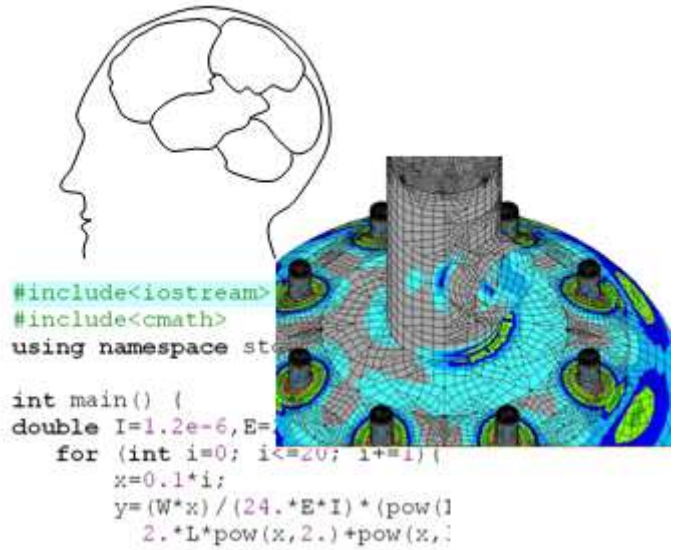
ME 110
Computation for
Mechanical
Engineering



Arrays

One-Dimensional

Content of this Week

This week we will study One-Dimensional Arrays:

- Concept
- Declaration, Initialisation, Assignment
- Processing Arrays
- Passing arrays to functions

Concept of Arrays

Up to now, a variable represents a *single value*; we call this a *scalar* variable. It has a type and a name; for example the declaration of a variable named **x** of type **double**:

```
double x;
```

An array variable (or a vector variable) can represent *more than one value*, but still with the *same name and same type*.

The above variable can be declared as an *array of 5 variables* as follows:

```
double x[5];
```

variable **x** can now store 5 values, all of type **double**.

Each *element* of the array of variables is accessed with an index:

x[i] with index **i = 0, 1, 2, 3, 4.**

in general for **n** elements the index range is **i = 0, 1, … , n-1.**

Accessing vectors is same, but their declaration is different (see later).

Array Declaration

The general form of the declaration of an array is:

```
type name[numberOfElements];
```

Examples

```
double mass[10];
```

The elements are:

```
mass[0]
mass[1]
mass[2]
mass[3]
mass[4]
mass[5]
mass[6]
mass[7]
mass[8]
mass[9]
```

```
int scores[3];
```

The elements are:

```
scores[0]
scores[1]
scores[2]
```

```
char status[2];
```

The elements are:

```
status[0]
status[1]
```

Note that in the declaration:

```
double mass[10];
```
Array size define at ***compile-time***

the number of elements, **10**, is defined using a *literal constant*.

Alternatively we can use a *named constant*;
for example the following is equivalent to the above declaration.

```
const int n=10;
double mass[n];
```
Array size defined at ***compile-time***

Note that "*Standard* C++" insists in this case that **n** is a constant and so forbids, for example, the following declarations:

```
int n;
cin >> n;
double mass[n];
```
Array size defined at ***run-time***

FORBIDDEN in standard C++!

or

```
int n=10;
double mass[n];
```

Your compiler might allow you to do this, but it is best to use only standard C++ features so that your program can be compiled on any platform that has a standard C++ compiler.

Declaration of an array containing 118 names.

```
string name[118];
```

The elements are:

name[0]	name[1]	name[2]	name[3]	name[4]	name[5]	name[6]
name[7]	name[8]	name[9]	name[10]	name[11]	name[12]	name[13]
name[14]	name[15]	name[16]	name[17]	name[18]	name[19]	name[20]
name[21]	name[22]	name[23]	name[24]	name[25]	name[26]	name[27]
name[28]	name[29]	name[30]	name[31]	name[32]	name[33]	name[34]
name[35]	name[36]	name[37]	name[38]	name[39]	name[40]	name[41]
name[42]	name[43]	name[44]	name[45]	name[46]	name[47]	name[48]
name[49]	name[50]	name[51]	name[52]	name[53]	name[54]	name[55]
name[56]	name[57]	name[58]	name[59]	name[60]	name[61]	name[62]
name[63]	name[64]	name[65]	name[66]	name[67]	name[68]	name[69]
name[70]	name[71]	name[72]	name[73]	name[74]	name[75]	name[76]
name[77]	name[78]	name[79]	name[80]	name[81]	name[82]	name[83]
name[84]	name[85]	name[86]	name[87]	name[88]	name[89]	name[90]
name[91]	name[92]	name[93]	name[94]	name[95]	name[96]	name[97]
name[98]	name[99]	name[100]	name[101]	name[102]	name[103]	name[104]
name[105]	name[106]	name[107]	name[108]	name[109]	name[110]	name[111]
name[112]	name[113]	name[114]	name[115]	name[116]	name[117]	

Clearly it is easier to declare one array (size 118) than it is to declare 118 scalars!

Array Initialisation

An array can be *initialised* with an initialiser list, like this:

```
double a[5] = {8.4, 3.6, 9.1, 4.7, 3.9};
```

If the array has more elements than values listed, then the remaining elements are initialized to zero. Hence,

```
double a[5] = {8.4, 3.6};
```

is equivalent to

```
double a[5] = {8.4, 3.6, 0.0, 0.0, 0.0};
```

The use of the size specifier (***numberOfElements***) may be omitted; so for example,

```
double a[] = {8.4, 3.6, 9.1, 4.7, 3.9};
```

is equivalent to the declaration,

```
double a[5] = {8.4, 3.6, 9.1, 4.7, 3.9};
```

Program illustrating array initialisation.

```
#include <iostream>
using namespace std;

int main () {

  const int n = 5;

  double a[ ] = {8.4, 3.6, 9.1, 4.7, 3.9};
  double b[n] = {4.0, 2.0};
  double c[n] = {0.0};

  for (int i=0; i<n; i++)
    cout << "a[" << i << "] = " << a[i] << ", "
         << "b[" << i << "] = " << b[i] << ", "
         << "c[" << i << "] = " << c[i] << endl;

  return 0;
}
```

Output

```
a[0] = 8.4, b[0] = 4, c[0] = 0
a[1] = 3.6, b[1] = 2, c[1] = 0
a[2] = 9.1, b[2] = 0, c[2] = 0
a[3] = 4.7, b[3] = 0, c[3] = 0
a[4] = 3.9, b[4] = 0, c[4] = 0
```

Declaration and initialization of an array containing names of 118 atoms
"Hydrogen" to "Ununoctium": http://en.wikipedia.org/wiki/List_of_elements

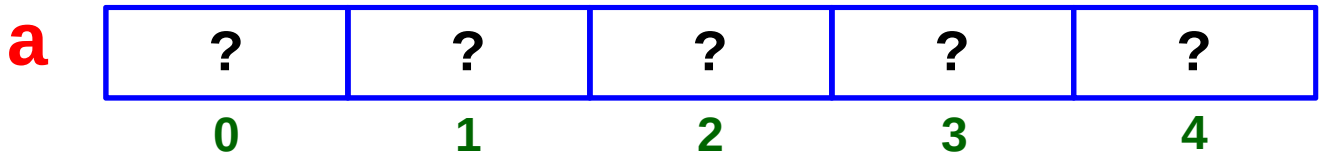
```
string name[] = {"Hydrogen","Helium","Lithium","Beryllium","Boron",
"Carbon","Nitrogen","Oxygen","Fluorine","Neon","Sodium","Magnesium",
"Aluminium","Silicon","Phosphorus","Sulphur","Chlorine","Argon",
"Potassium","Calcium","Scandium","Titanium","Vanadium","Chromium",
"Manganese","Iron","Cobalt","Nickel","Copper","Zinc","Gallium",
"Germanium","Arsenic","Selenium","Bromine","Krypton","Rubidium",
"Strontium","Yttrium","Zirconium","Niobium","Molybdenum","Technetium",
"Ruthenium","Rhodium","Palladium","Silver","Cadmium","Indium","Tin",
"Antimony","Tellurium","Iodine","Xenon","Caesium","Barium",
"Lanthanum","Cerium","Praseodymium","Neodymium","Promethium",
"Samarium","Europium","Gadolinium","Terbium","Dysprosium","Holmium",
"Erbium","Thulium","Ytterbium","Lutetium","Hafnium","Tantalum",
"Tungsten","Rhenium","Osmium","Iridium","Platinum","Gold","Mercury",
"Thallium","Lead","Bismuth","Polonium","Astatine","Radon","Francium",
"Radium","Actinium","Thorium","Protactinium","Uranium","Neptunium",
"Plutonium","Americium","Curium","Berkelium","Californium",
"Einsteinium","Fermium","Mendelevium","Nobelium","Lawrencium",
"Rutherfordium","Dubnium","Seaborgium","Bohrium","Hassium",
"Meitnerium","Darmstadtium","Roentgenium","Copernicium","Ununtrium",
"Ununquadium","Ununpentium","Ununhexium","Ununseptium","Ununoctium"};
```

You might wish to assign these values by reading them from a file. (see next lecture).

Array Assignment

Consider again the array a containing 5 elements.
The declaration (*without* initialisation) is: `double a[5];`

At this time, the elements have *unpredictable values!*



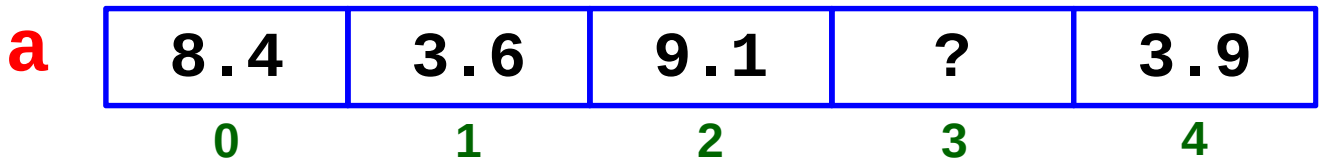
Elements of the array can be *assigned* (at any time) as follows:

```
a[0] = 8.4;
a[1] = 3.6;
a[2] = 9.1;
a[4] = 3.9;
```

The green values are the array indices.

Remember that you need to assign values before you use the array!

The values of (some of) the elements are now defined:



Note that element `a[3]` is still not defined!

Assignment can be performed directly from input:

This program declares an array, assigns the elements with values input from the keyboard, and then outputs the values in reverse order.

```
#include <iostream>
using namespace std;

int main () {
  double a[5];
  cout << "Input 5 real numbers:" << endl;
  for(int i=0; i<5; i++) cin >> a[i];
  cout << "In reverse order: " << endl;
  for(int i=4; i>=0; i--) cout << a[i] << " ";
  cout << endl;
  return 0;
}
```

Output

```
Input 5 real numbers:
1.2 3.5 -0.4 10.2 7.1
In reverse order:
7.1 10.2 -0.4 3.5 1.2
```

Processing Arrays

Elements of an array can be processed in the same way as scalar variables.

```
#include <iostream>
using namespace std;

int main () {

  const int n = 5;
  double a[n] = {1.7, 4.1, 5.6, 3.4, 3.1};

  double s1=0.0, s2=0.0;
  for (int i=0; i<n; i++) {
    s1 = s1 + a[i];
    s2 = s2 + a[i]*a[i];
  }

  cout << "The sum is " << s1 << endl;
  cout << "The sum of the squares is "
       << s2 << endl;

}
```

$$s_1 = \sum_i a_i$$

$$s_2 = \sum_i a_i^2$$

Output

```
The sum is 17.9
The sum of the squares is 72.23
```

Passing Arrays to the Functions

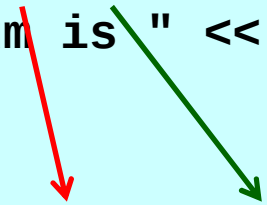
This works in a similar way as passing scalar variables to functions.

```
#include <iostream>
using namespace std;

double sum(double [], int);

int main () {
  double a[5] = {1.7, 4.1, 5.6, 3.4, 3.1};
  double s = sum(a, 5);
  cout << "The sum is " << s << endl;
  return 0;
}

double sum(double x[], int n) {
  double t = 0.0;
  for (int i=0; i<n; i++)
    t = t + x[i];
  return t;
}
```



In the prototype, `[]` indicates that an array is to be passed by referenced.

The function needs to know the size of the array, this is passed to parameter `n`.

Output

```
The sum is 17.9
```

Example: Max function to determine the maximum value

```
#include <iostream>
using namespace std;
int max(int x[],int n){
int m=x[0];
  for(int i=1;i<n;i++)if(m<x[i])m=x[i];
  return m;
}
int main () {
    const int k=10;
    int a[k];
    for(int i=0;i<k;i++){
      cout<<"input the element of:"<<i<<endl;
      cin>>a[i];
    }
    cout<<"maximum elemet is:"<<max(a,k);
  system("pause");return 0;
}
```

Example: Write a program to input 10 double values and output the mean and standard deviation of them.

Hint:

Mean value, $$av = (1/n) * \sum_{i=1,n} a_i$$

Standard deviation, $$sd = \sqrt{\frac{\sum_{i=1}^{n} (a_i - av)^2}{n-1}}$$

Solution

```
#include <iostream>
#include <cmath>
using namespace std;
int main () {
    const int n=10;
    double a[n],s=0,sd=0,av;
    for(int i=0;i<n;i++){
      cout<<"input the element of:"<<i<<endl;
      cin>>a[i];
      s+=a[i];
    }
    av=s/n;
    for(int j=0;j<n;j++)sd+=pow((a[j]-av),2);
    cout<<"standard deviation is:"<<sqrt(sd/(n-1.))<<endl;
  system("pause");return 0;
}
```