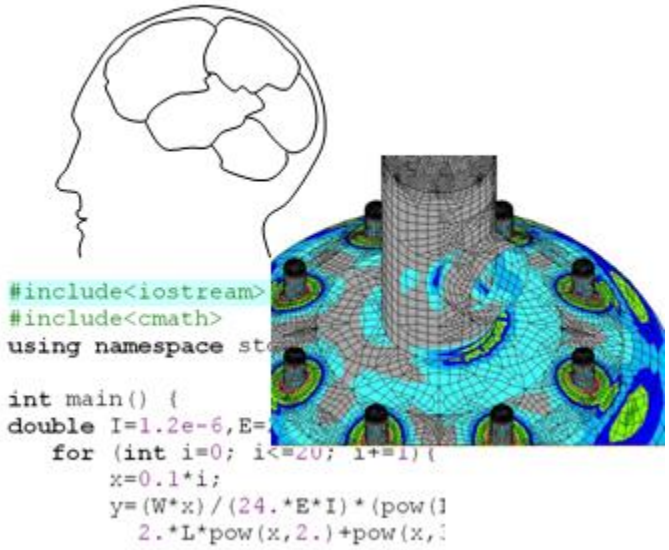
ME 110
Computation for
Mechanical
Engineering



Vectors

Basics

Content

This week we will study Vectors:

- Concept
- Declaration, Initialization, Assignment
- Dynamic Processing of Vectors

Concept of Vectors

Recall that for arrays:

- the size of an array cannot be defined at run-time
- the size of an array cannot be changed at run-time

Such objects are called *static*, they can only be defined at compile-time.

C++ provides the *vector* data class that enables the programmer to create *dynamic* arrays:

- the size of a vector can be defined at run-time
- the size of a vector may change at run-time

The vector data class provides many powerful methods for processing dynamic memory management.

Vector Declaration and Initialisation

First, to use the vector class the following header must be included:

```
#include <vector>
```

The general form of the declaration of a vector array is:

```
vector<type> name(numberOfElements);
```

optional

Examples

```
vector<double>  mass(6);
```

The elements are:

```
mass[0]
mass[1]
mass[2]
mass[3]
mass[4]
mass[5]
```

```
vector<int> scores;
```

This is an *empty* vector!
The ***numberOfElements*** is zero and so there are no elements.

Note that the indexing of the elements of vectors is the same as that of arrays.

Vector Initialisation

The general form of vector declaration:

```
vector<type> name(numberOfElements);
```

initialises all elements of the vector to zero.

Alternatively an initialiser can be given at declaration:

```
vector<type> name(numberOfElements, value);
```

initialises all elements of the vector to ***value***.

Examples

```
vector<double>  mass(6);
```

all elements of `mass` are initialised to 0.

```
vector<double>  mass(6,1.8);
```

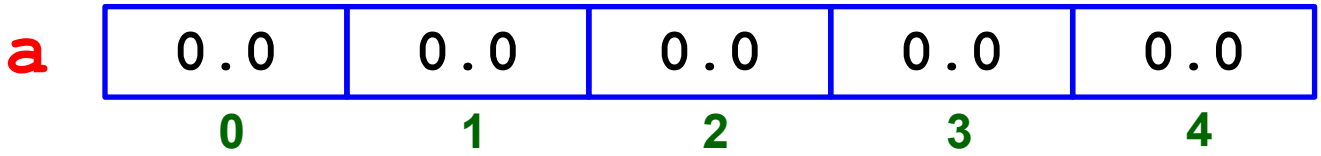
all elements of `mass` are initialised to `1.8`

Vector Assignment

Consider the vector declaration:

```
vector<double> a(5);
```

At this time, the elements are all automatically initialised to zero.



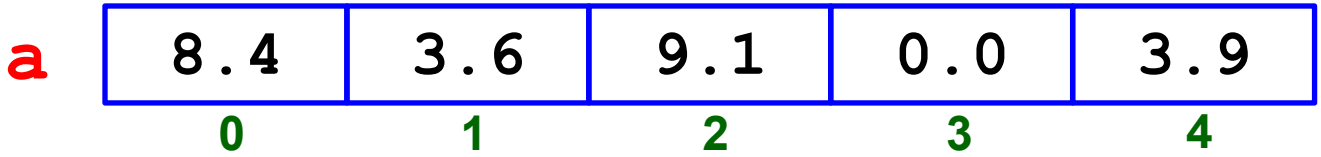
Elements of a vector array can be *assigned* (at any time) as follows:

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

The green values are the array indices.

Note that vector assignment is performed in the same way as array assignment.

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



Note that the value of element `a[3]` is still 0.0

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>
#include <vector>
using namespace std;

int main () {

  vector<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;

}
```

You could also use normal arrays:

replace `vector<double> a(5);`
with `double a[5];`

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
```

Dynamic Processing of Vectors

So far, vectors look very much like static arrays.

However, vectors can also be defined and processed dynamically; a vector is therefore a type of *dynamic array*.

There are many powerful methods available for dynamic processing of vectors; we will look at just five of them:

`name.size();`	returns the size of vector `name`
`name.push_back(x);`	adds an element with value `x` to the end of the vector (increasing the vector size by one).
`name.pop_back();`	removes an element from the end of the vector (decreasing the size by one).
`name.clear();`	removes all elements from the vector (leaving a vector of size zero)
`name.resize(s);`	resizes the vector to size `s`

Using the `.size()` method

The **`.size()`** method provides a simple and consistent way to loop over all elements in a vector without the need to keep track of the vector's size:

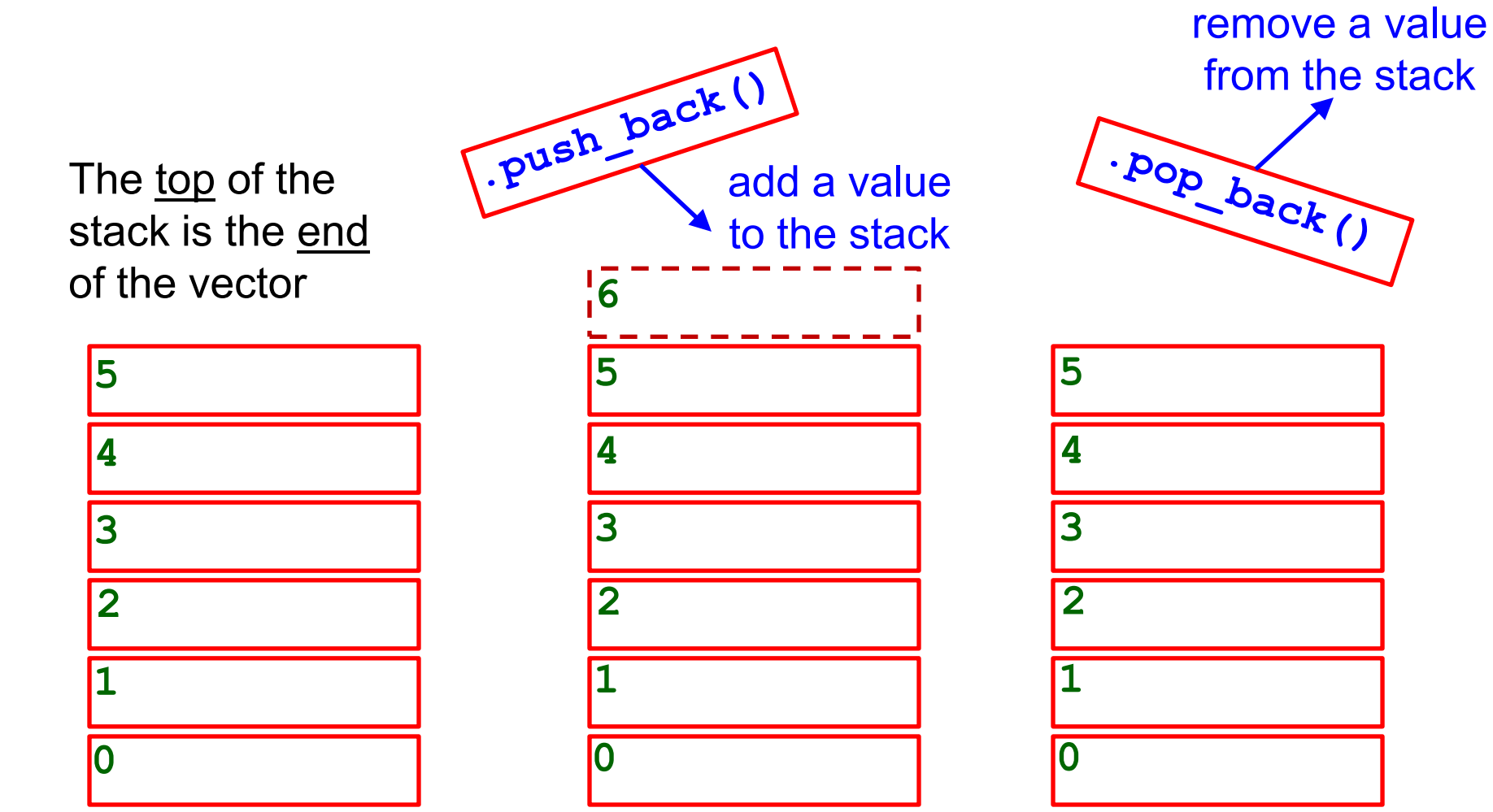
```
vector<double> mass(5);

for (unsigned int i=0; i<mass.size(); i++) {
  mass[i] = i*i;
}
```

Note that the **`.size()`** method returns an **`unsigned int`** and so the counter **`i`** is also defined as type **`unsigned int`**.

In time, you will discover more uses for this method...

Using the `.push_back()` and `.pop_back()` methods

A vector can be considered as a ***stack*** of values.



Using the .push_back() method

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

int main () {

  vector<double> x(3, 8.3);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

  x.push_back(5.9);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

}
```

```
The size is 3
The content is: 8.3 8.3 8.3
The size is 4
The content is: 8.3 8.3 8.3 5.9
```

Using the .pop_back() method

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

int main () {

  vector<double> x(3, 8.3);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

  x.pop_back();

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

}
```

```
The size is 3
The content is: 8.3 8.3 8.3
The size is 2
The content is: 8.3 8.3
```

Using the `.clear()` method

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

int main () {

  vector<double> x(3, 8.3);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

  x.clear();

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

}
```

```
The size is 3
The content is: 8.3 8.3 8.3
The size is 0
The content is:
```

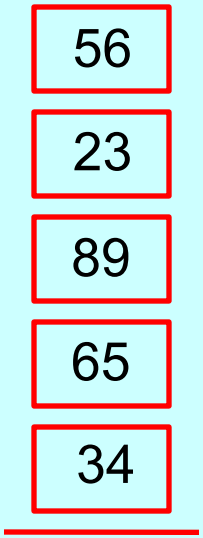
Using the `.resize()` method

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

int main () {

  vector<double> x(3, 8.3);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

  x.resize(5);

  cout << "The size is " << x.size() << endl;
  cout << "The content is: ";
  for (unsigned int i=0; i<x.size(); i++) cout << x[i] << " ";
  cout << endl;

}
```

```
The size is 3
The content is: 8.3 8.3 8.3
The size is 5
The content is: 8.3 8.3 8.3 0.0 0.0
```

This program builds a vector from values input from the keyboard. The size of the vector (initially zero) increases until a zero is input.

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

int main() {

  vector<int> iv;

  int n;
  while(true) {
    cout << "Input: ";
    cin >> n;
    if (n==0) break;
    iv.push_back(n);
  }

  cout << "iv is:" << endl;
  for(unsigned int i=0; i<iv.size(); i++)
    cout << "iv[" << i << "] = " << iv[i] << endl;

}
```



Output

```
Input: 34
Input: 65
Input: 89
Input: 23
Input: 56
Input: 0
iv is:
iv[0] = 34
iv[1] = 65
iv[2] = 89
iv[3] = 23
iv[4] = 56
```