
EEE146
VECTORS

C++ vectors

*** Static Arrays (SA):**

- the size of SA cannot be defined at run-time
- the size of SA cannot be changed at run-time

*** Dynamic Arrays (DA):**

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

*** Vectors:**

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 Initialization

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);
```

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 size 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 Initialization

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 0.0

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

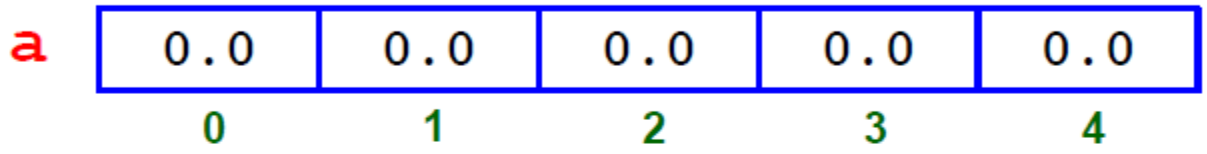
all elements of `mass` are initialised to 1.8

Vector Assignment

Consider the vector declaration:

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

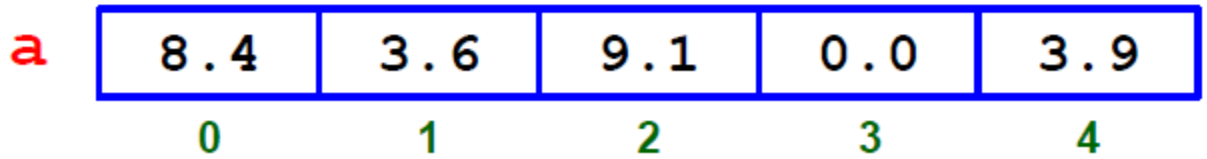
At this time, the elements are all automatically initialised to zero.



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;
```

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



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

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

int main () {

  int n;
  cout << " Input n: "; cin >> n;
  vector<double> a(n);

  cout << "Input " << n << " real numbers:" << endl;
  for(int i=0; i<n; i++)
      cin >> a[i];

  cout << "In reverse order: " << endl;
  for(int i=n-1; i>=0; i--)
      cout << a[i] << " ";

}
```

You could also use DA arrays:

replace `vector<double> a(n);`
with `double *a = new double [n];`

```
Input n: 5
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 Vectors

Vectors can be processed in the same way as arrays.

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

int main () {

  int n = 5;
  vector<double> a(n);
  a[0]=1.7; a[1]=4.1; a[2]=5.6; a[3]=3.4; a[4]=3.1;

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

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

}
```

$$s_2 = \sum_i a_i^2$$

Note that we can define the size of the vector at run-time!

Output:

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

Dynamic Processing of Vectors

There are many powerful methods available for dynamic processing of vectors; we will see the following.

name.size();	Returns the size of vector.
name.push_back(*x*);	Adds value *x* to the end of the vector.(increasing size by one)
name.pop_back();	Removes a value from the end of the vector.(decreasing the size by one)
name.clear();	Removes all values from the vector. (leaving a vector of size zero)

Dynamic Processing of Vectors

name.resize(s);	Resizes the vector to size s.
name.begin();	Returns an iterator to the first element.
name.end();	Returns an iterator to the element following the last element.
name.insert();	Inserts element.
name.erase();	Deletes element.
name.empty();	Returns true if the size is zero. Returns false otherwise.

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:

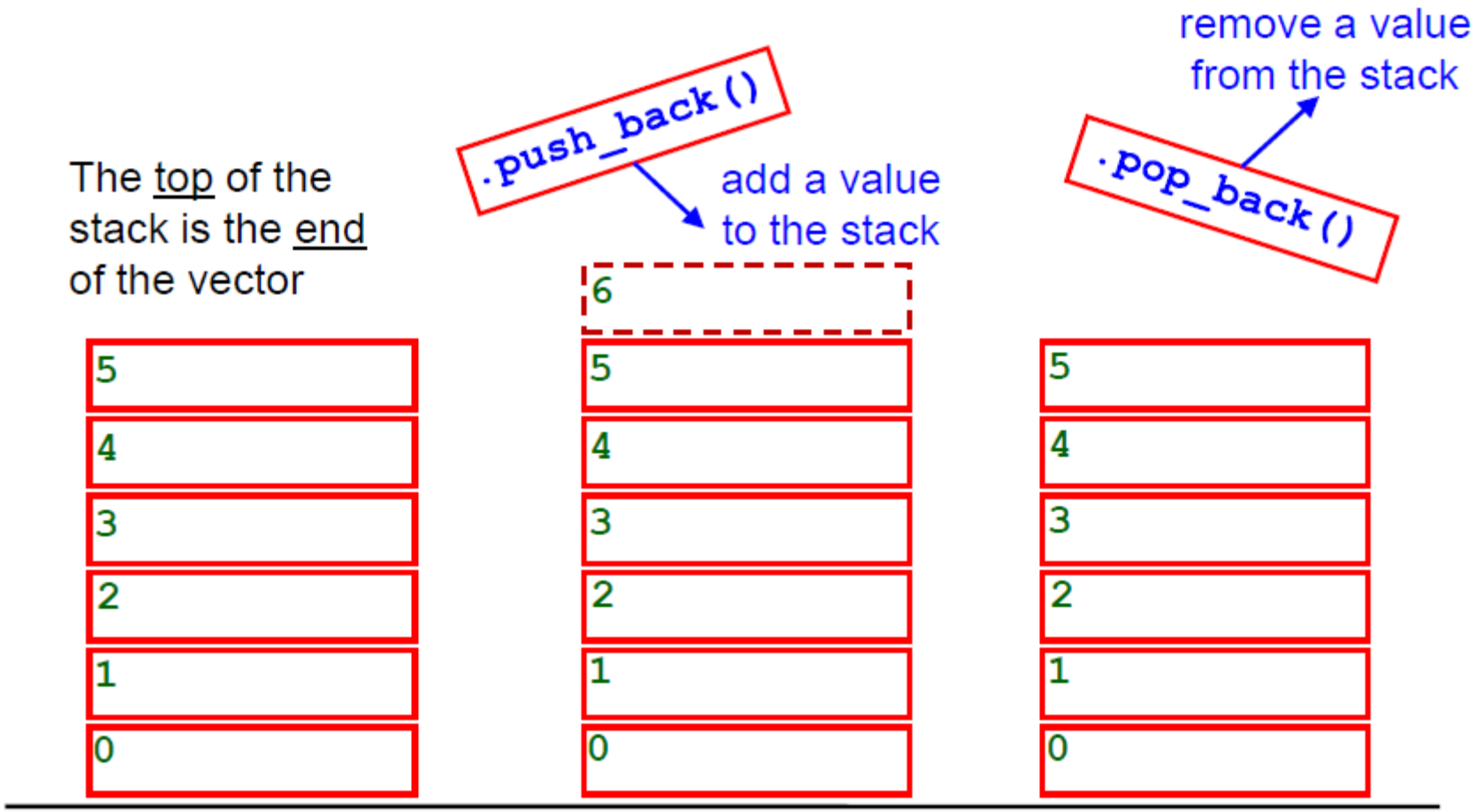
```
vector<double> mass(5);

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
```

Using .begin(), .end(), .insert(), .erase()

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

void printVector(vector<int> v)
{
    cout << "myvector contains: ";
    for(int i=0;i< v.size();i++)
       cout << v[i] << "  ";
    cout <<endl;
}
int main()
{
  vector<int> myvector (3,100);

  myvector.insert ( myvector.begin() , 200 );//inserts 200 as the 0th element
  printVector(myvector);

  myvector.insert (myvector.begin(),2,300);//inserts two 300 as 0th and 1st element
  printVector(myvector);
```

Using .begin(), .end(), .insert(), .erase()

```
 vector<int> anothervector (2,400);//declares anothervector with two elements with
      values 400
myvector.insert(myvector.begin()+2,anothervector.begin(),anothervector.end());//inserts
      anothervector from beginning to the end to position 2
printVector(myvector);

 int myarray [] = { 501,502,503 };
 myvector.insert (myvector.begin(), myarray, myarray+3); //inserts first three elements of
      myarray to the beginning of myvector
 printVector(myvector);
 myvector.erase(myvector.begin()+4); //erases the element with index 4
 printVector(myvector);
 myvector.erase(myvector.begin()+3,myvector.end()-2);  //erases the range starting from
      3th(index) element, ending 2 elements before the end
   printVector(myvector);
   return 0;
}
```

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

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

int main() {

 int n;
 vector<int> iv;

 while(true) {
   cout << "Input an integer: ";
   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 an integer: 34
input an integer: 65
input an integer: 89
input an integer: 23
input an integer: 56
input an integer: 0
iv is:
    iv[0] = 34
    iv[1] = 65
    iv[2] = 89
    iv[3] = 23
    iv[4] = 56
```

Using vectors with functions

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

double max(vector<double> v){
  double eb = v[0];
  for(int i=0; i<v.size(); i++){
    if(v[i]>eb) eb = v[i];
  }
  return eb;
}
int main() {

 int n;
 cout << "Input n: ";
 cin >> n;
 vector<double> x(n);

 for(unsigned int i=0; i<x.size(); i++) cin >> x[i];

 cout << "maximum element is: " << max(x) << endl;
}
```

Output

```
Input n: 4
1.1
2.2
-4.3
0.4
maximum element is: 2.2
```

Homework

1. A vector is given as follows: B={3,-5,-2,4,-7,9,22,-8}.
   Write a program to remove the negative elements from the vector.

2. Write a program to do followings:

   a) Input n

   b) Input elements of an integer dynamic array of size n (use **new** operator)

   c) Sort the elements in increasing order and output the sorted values to the user screen.

Example output for n=5:

```
input n: 5
input elements: 5 -4 7 9 1
Sorting: -4  1  5  7  9
```

3. Write a program to find the mean, mode and median of an n-element integer vector. You must read elements of the vector from keyboard.

The median is the number in the middle and the mode is the most frequent number in a data set.

For example:
For the data set {3, 4, 4, 5, 6, 8, 8, 8, 10},
median = 6 and mod = 8.
For the data set {5, 5, 7, 9, 11, 11, 18, 18},
median = (9+11)/2 = 10 and mod = 18.

Mode of the set: {2, 2, 5, 9, 9, 9, 10, 10, 11 12 18} is 9. (unimodal data)
Mode of the set: {2, 3, 4, 4, 4, 5, 7, 7, 7, 9} is 4 and 7 (bimodal set of data)
Mode of the set: {1, 2, 3, 8, 9, 10, 12, 14, 18} is ? (data has no mode)