

Arrays and Vectors

Victor Eijkhout, Harika Gurram,
Je'aime Powell, Charley Dey

Fall 2018

General note about syntax

Many of the examples in this lecture need the compiler option `-std=c++11`. This works for both compilers, so:

```
// for Intel:  
icpc -std=c++11 yourprogram.cxx  
// for gcc:  
g++ -std=c++11 yourprogram.cxx
```

Static arrays

Array creation

```
{  
    int numbers[] = {5,4,3,2,1};  
    cout << numbers[3] << endl;  
}  
  
{  
    int numbers[5]{5,4,3,2,1};  
    numbers[3] = 21;  
    cout << numbers[3] << endl;  
}
```

Range over elements

You can write a *range-based for* loop, which considers the elements as a collection.

```
for ( float e : array )  
    // statement about element with value e  
for ( auto e : array )  
    // same, with type deduced by compiler
```

Code:

```
int numbers[] = {1,4,2,6,5};  
int tmp_max = numbers[0];  
for (auto v : numbers)  
    if (v>tmp_max)  
        tmp_max = v;  
cout << "Max: " << tmp_max << " (should be 6)" << endl;
```

Output:

Max: 6 (should be 6)

Indexing the elements

You can write an *indexed for* loop, which uses an index variable that ranges from the first to the last element.

```
for (int i= /* from first to last index */ )  
    // statement about index i
```

Example: find the maximum element and where it occurs.

Code:

```
int numbers[] = {1,4,2,6,5};  
int tmp_idx = 0;  
int tmp_max = numbers[tmp_idx];  
for (int i=0; i<5; i++) {  
    int v = numbers[i];  
    if (v>tmp_max) {  
        tmp_max = v; tmp_idx = i;  
    }  
}  
cout << "Max: " << tmp_max  
    << " at index: " << tmp_idx << endl;
```

Output:

Max: 6 at index: 3

Exercise 1

Code:

```
{
  int numbers[] = {1,4,2,6,5};
  int tmp_max = numbers[0];
  for (auto v : numbers)
    if (v>tmp_max)
      tmp_max = v;
  cout << "Max: " << tmp_max << " (should be 6)" << endl;
}
```

Output:

Max: 6 at index: 3

Is the student's code correct for arrays of length one and zero?

Exercise 2

Code:

```
{
  int numbers[] = {1,4,2,6,5};
  int tmp_max = numbers[0];
  for (auto v : numbers)
    if (v>tmp_max)
      tmp_max = v;
  cout << "Max: " << tmp_max << " (should be 6)" << endl;
}
```

Output:

Max: 6 at index: 3

Is the student's code correct for arrays of length one and zero?

Vectors

Vector definition

```
#include <vector>
using std::vector;

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

where

- vector is a keyword,
- type (in angle brackets) is any elementary type or class name,
- name is up to you, and
- size is the (initial size of the array). This is an integer, or more precisely, a `size_t` parameter.
- value is the uniform initial value of all elements.

Accessing vector elements

In a number of ways, vector behaves like an array:

```
vector<double> x(5, 0.1 );  
x[1] = 3.14;  
cout << x[2];
```

Ranging over a vector

```
for ( auto e : my_vector)  
    cout << e;
```

Note that `e` is a copy of the array element:

Code:

```
vector<float> myvector  
    = {1.1, 2.2, 3.3};  
for ( auto e : myvector )  
    e *= 2;  
cout << myvector[2] << endl;
```

Output:

3.3

Ranging over a vector by reference

To set array elements, make e a reference:

```
for ( auto &e : my_vector)  
    e = ....
```

Code:

```
vector<float> myvector  
    = {1.1, 2.2, 3.3};  
for ( auto &e : myvector )  
    e *= 2;  
cout << myvector[2] << endl;
```

Output:

6.6

Vector initialization

You can initialize a vector with much the same syntax as an array:

```
vector<int> odd_array{1,3,5,7,9};  
vector<int> even_array = {0,2,4,6,8};
```

(This syntax requires compilation with the `-std=c++11` option.)

Vector initialization'

There is a syntax for initializing a vector with a constant:

```
vector<float> x(25,3.15);
```

which gives a vector of size 25, with all elements initialized to 3.15.

Vector indexing

Your choice: fast but unsafe, or slower but safe

```
vector<double> x(5);  
x[5] = 1.; // will probably work  
x.at(5) = 1.; // runtime error!
```


Vector copy

Vectors can be copied just like other datatypes:

Code:

```
vector<float> v(5,0), vcopy;  
v[2] = 3.5;  
vcopy = v;  
cout << vcopy[2] << endl;
```

Output:

```
./vectorcopy  
3.5
```

Vector methods

- Get elements with `ar[3]` (zero-based indexing).
(for C programmers: this is not dereferencing, this uses an operator method)
- Get elements, including bound checking, with `ar.at(3)`.
- Size: `ar.size()`.
- Other functions: `front`, `back`.

Dynamic extension

Extend with `push_back`:

Code:

Output:

```
./vectorend  
6  
35
```

also `pop_back`, `insert`, `erase`.
Flexibility comes with a price.

Multi-dimensional vectors

Multi-dimensional is harder with vectors:

```
vector<float> row(20);  
vector<vector<float>> rows(10,row);
```

Vector of vectors.

Dynamic behaviour

Dynamic size extending

```
vector<int> iarray;
```

creates a vector of size zero. You can then

```
iarray.push_back(5);  
iarray.push_back(32);  
iarray.push_back(4);
```

Vector extension

You can push elements into a vector:

```
vector<int> flex;  
/* ... */  
for (int i=0; i<LENGTH; i++)  
    flex.push_back(i);
```

If you allocate the vector statically, you can assign with at:

```
vector<int> stat(LENGTH);  
/* ... */  
for (int i=0; i<LENGTH; i++)  
    stat.at(i) = i;
```

Vector extension

With subscript:

```
vector<int> stat(LENGTH);  
/* ... */  
for (int i=0; i<LENGTH; i++)  
    stat[i] = i;
```

You can also use new to allocate (see section ??):

```
int *stat = new int[LENGTH];  
/* ... */  
for (int i=0; i<LENGTH; i++)  
    stat[i] = i;
```


Timing

Flexible time: 2.445
Static at time: 1.177
Static assign time: 0.334
Static assign time to new: 0.467

Vectors and functions

Vector as function return

You can have a vector as return type of a function:

Code:

```
vector<int> make_vector(int n) {  
    vector<int> x(n);  
    x[0] = n;  
    return x;  
}  
  
/* ... */  
vector<int> x1 = make_vector(10); // "auto" also possible!  
cout << "x1 size: " << x1.size() << endl;  
cout << "zero element check: " << x1[0] << endl;
```

Output:

```
./vectorreturn  
x1 size: 10  
zero element check: 10
```

Vector as function argument

You can pass a vector to a function:

```
void print0( vector<double> v ) {  
    cout << v[0] << endl;  
};
```

Vectors, like any argument, are passed by value, so the vector is actually copied into the function.

Vector pass by value example

Code:

```
void set0
( vector<float> v,float x )
{
    v[0] = x;
}
/* ... */
vector<float> v(1);
v[0] = 3.5;
set0(v,4.6);
cout << v[0] << endl;
```

Output:

```
./vectorpassnot
3.5
```

Vector pass by reference

If you want to alter the vector, you have to pass by reference:

Code:

```
void set0
( vector<float> &v,float x )
{
    v[0] = x;
}
/* ... */
vector<float> v(1);
v[0] = 3.5;
set0(v,4.6);
cout << v[0] << endl;
```

Output:

```
./vectorpassref
4.6
```

(hints for the next exercise)

```
// high up in your code:  
#include <random>  
using namespace std;  
  
// in your main or function:  
float r = 1.*rand()/RAND_MAX;  
// gives random between 0 and 1
```

Exercise 3

You need to pass the array length as a separate parameter.

Vectors in classes

Can you make a class around a vector?

Vector needs to be created with the object, so you can not have the size in the class definition

```
class witharray {  
private:  
    vector<int> the_array( ???? );  
public:  
    witharray( int n ) {  
        thearray( ???? n ???? );  
    }  
}
```

Create and assign

The following mechanism works:

```
class witharray {  
private:  
    vector<int> the_array;  
public:  
    witharray( int n ) {  
        thearray = vector<int>(n);  
    }  
}
```

Matrix class

```
class matrix {  
private:  
    int rows,cols;  
    vector<vector<double>> elements;  
public:  
    matrix(int m,int n) {  
        rows = m; cols = n;  
        elements =  
            vector<vector<double>>(m,vector<double>(n));  
    }  
    void set(int i,int j,double v) {  
        elements.at(i).at(j) = v;  
    };  
    double get(int i,int j) {  
        return elements.at(i).at(j);  
    };  
};
```

Matrix class'

Better idea:

```
elements = vector<double>(rows*cols);  
...  
void get(int i,int j) {  
    return elements.at(i*cols+j);  
}
```

Exercise 4

Add methods such as transpose, scale to your matrix class.
Implement matrix-matrix multiplication.

Pascal's triangle

Pascal's triangle contains binomial coefficients:

```
Row   1:      1
Row   2:    1  1
Row   3:  1  2  1
Row   4:    1  3  3  1
Row   5:      1  4  6  4  1
Row   6:        1  5 10 10  5  1
Row   7:          1  6 15 20 15  6  1
Row   8:            1  7 21 35 35 21  7  1
Row   9:              1  8 28 56 70 56 28  8  1
Row  10:                1  9 36 84 126 126 84 36  9  1
```

where

$$p_{rc} = \binom{r}{c} = \frac{r!}{c!(r-c)!}.$$

The coefficients can easily be computed from the recurrence

$$p_{rc} = \begin{cases} 1 & c \equiv 1 \vee c \equiv r \\ p_{r-1,c-1} + p_{r-1,c} & \end{cases}$$

