

CSC 221: Computer Programming I

Fall 2001

- vectors
 - <vector> library, templated
 - advantages over arrays:
 - ✓ size can be determined at run-time
 - ✓ resizable
 - ✓ out-of-bounds checking
 - ✓ parameter passing is normal
 - ✓ can easily add to the end

array disadvantages

- size must be determined at compile-time
 - can't prompt the user for the size, then create it
 - means that you often waste lots of space
 - if run out of space, too bad!
- no bounds checking on array accesses
 - can lead to unexpected results
- parameter passing is very strange
 - when pass by value, function can still change the contents of the array

through a class definition, C++ provides for a "better array"

- a vector is an array wrapped up in a class, with all the ugly details hidden
 - provides access to an indexable collection of data elements (all of same type)
 - in addition, provides other useful features not found in arrays
- ➔ all of the above disadvantages of arrays are fixed using vectors!

vectors

the vector class is defined in the `<vector>` library

- when you declare a vector, you must specify the type and a size
 - the type is specified inside `< >` (vector is known as a *templated class*)
 - the size does *not* need to be known at compile-time

```
#include <vector> // loads definition of the vector class
using namespace std;

vector<int> nums(100); // declares a vector of 100 ints
// equiv. to int nums[100];

for (int i = 0; i < 100; i++) { // vector elements are accessible via
    nums[i] = 0; // an index (same as with arrays)
}

vector<string> words(10); // declares a vector of 10 strings
// equiv. to string words[10];

int numGrades;
cout << "How many grades are there? ";
cin >> numGrades;

vector<double> grades(numGrades); // declares a vector of numGrades
// doubles (can't be done with arrays)
```

size & resize

two useful member functions of the vector class are `size` & `resize`

- `size` will return the amount of space allocated for the vector

```
vector<int> nums(100);

cout << nums.size() << endl; // displays 100
```

- `resize` expands (or shrinks) the vector, retaining the data

```
nums.resize(200); // makes space for 100 more ints,
// nums[0]..nums[99] is unchanged
```

→ can read in and store an arbitrary number of values

in pseudo-code:

```
while (READ INPUT OK) {
    if (VECTOR IS FULL) {
        RESIZE THE VECTOR TO MAKE ROOM;
    }

    ADD INPUT TO VECTOR;
    INCREMENT ELEMENT COUNT;
}
```

how big do you resize?

- by 1?
- by a constant amount?
- double it?

reverse array

vs.

reverse vector

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

const int MAX_SIZE = 100;

int main()
{
    string words[MAX_SIZE];
    int numWords = 0;

    string input;
    while (numWords < MAX_SIZE && cin >> input) {
        words[numWords] = input;
        numWords++;
    }

    for (int i = numWords-1; i >= 0; i--) {
        cout << words[i] << endl;
    }

    return 0;
}
```

stops processing when array is full

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

const int INITIAL_SIZE = 10;

int main()
{
    vector<string> words(INITIAL_SIZE);
    int numWords = 0;

    string input;
    while (cin >> input) {
        if (numWords == words.size()) {
            words.resize(2*words.size());
        }
        words[numWords] = input;
        numWords++;
    }

    for (int i = numWords-1; i >= 0; i--) {
        cout << words[i] << endl;
    }

    return 0;
}
```

simply resizes vector when full, goes on

bounds checking & parameter passing

unlike arrays, vector access are checked for OOB

```
int nums[100];
for (int i=0; i<=100; i++) {
    nums[i] = 0;
}
```

RESULT: ???

```
vector<int> nums(100);
for (int i=0; i<=100; i++) {
    nums[i] = 0;
}
```

RESULT: run-time error

unlike arrays, vector parameters behave as any other type

```
void foo(vector<int> nums)
{
    nums[0] = 999;
}

vector<int> numbers(10);
foo(numbers);
```

RESULT: nums is a copy of the vector,
no change to numbers[0]

```
void foo(vector<int> & nums)
{
    nums[0] = 999;
}

vector<int> numbers(10);
foo(numbers);
```

RESULT: nums is an alias for the numbers vector
simultaneously changes numbers[0]

grade cutoff example

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

const int INITIAL_SIZE = 20;

void ReadGrades(vector<int> & grades,
                int & numGrades);
int CountAbove(vector<int> grades,
               int numGrades, int cutoff);

int main()
{
    vector<int> grades(INITIAL_SIZE);
    int numGrades;
    ReadGrades(grades, numGrades);

    int cutoff;
    cout << "Enter the desired grade cutoff: ";
    cin >> cutoff;

    cout << "There are "
         << CountAbove(grades, numGrades, cutoff)
         << " grades above " << cutoff << endl;

    return 0;
}

////////////////////////////////////
```

```
void ReadGrades(vector<int> & grades,
                int & numGrades)
// Results: reads grades and stores in vector
//          numGrades is set to the # of grades
{
    string filename;
    cout << "Enter the grades file name: ";
    cin >> filename;

    ifstream myin;
    myin.open( filename.c_str() );

    while (!myin) {
        cout << "File not found. Try again: ";
        cin >> filename;
        myopen.clear();
        myin.open( filename.c_str() );
    }

    numGrades = 0;

    int grade;
    while (myin >> grade) {
        if (numGrades == grades.size()) {
            grades.resize(2*grades.size());
        }
        grades[numGrades] = grade;
        numGrades++;
    }
    myin.close();
}
```

grade cutoff example (cont.)

can pass a vector by-value, and function will make a copy of the entire structure (SAFE)

```
int CountAbove(vector<int> grades,
               int numGrades, int cutoff)
// Assumes: grades contains numGrades grades
// Returns: number of grades >= cutoff
{
    int numAbove = 0;

    for(int i = 0; i < numGrades; i++) {
        if (grades[i] >= cutoff) {
            numAbove++;
        }
    }

    return numAbove;
}
```

copying a vector can be time and space intensive – alternative is to pass by-const-reference (const for safety, reference for efficiency)

```
int CountAbove(const vector<int> & grades,
               int numGrades, int cutoff)
// Assumes: grades contains numGrades grades
// Returns: number of grades >= cutoff
{
    int numAbove = 0;

    for(int i = 0; i < numGrades; i++) {
        if (grades[i] >= cutoff) {
            numAbove++;
        }
    }

    return numAbove;
}
```

vector summary

vectors are more flexible, easier, and safer than arrays

- size can be determined during run-time, can be resized
- bounds checking is automatically performed
- parameter passing is "normal"

but wait, there's more!

- a common use of arrays is reading in values and storing them in order
- vectors provide member functions specially designed for such tasks

- can declare an empty vector (size 0)
- then, can add new values to the end using the `push_back` member function
 - ✓ does not require remembering where the end is
 - ✓ does not require resizing to make room, automatically done
 - ✓ does not waste any space, no separate counter (size == # of elements)

resizable vector

vs.

growable vector

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

const int INITIAL_SIZE = 10;

int main()
{
    vector<string> words(INITIAL_SIZE);
    int numWords = 0;

    string input;
    while (cin >> input) {
        if (numWords == words.size()) {
            words.resize(2*words.size());
        }
        words[numWords] = input;
        numWords++;
    }

    for (int i = numWords-1; i >= 0; i--) {
        cout << words[i] << endl;
    }

    return 0;
}
```

requires separate counter, can still waste

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

const int INITIAL_SIZE = 10;

int main()
{
    vector<string> words; // default size 0

    string input;
    while (cin >> input) {
        words.push_back(input);
    }

    for (int i = words.size()-1; i >= 0; i--) {
        cout << words[i] << endl;
    }

    return 0;
}
```

final size is exact (but lots of resizing)