CSC 222: Computer Programming II

Spring 2004

- vectors
  - \<vector\> library, templated
  - advantages over arrays:
- const-reference parameters
- growable vectors (push_back & size)

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)
```
3

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?

4

Reverse array   vs.   Reverse vector

```cpp
#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;
}
```

```cpp
#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;
}
```

stops processing when array is full   simply resizes vector when full, goes on

---

bounds checking & parameter passing

unlike arrays, vector access are checked for OOB

```cpp
int nums[100];

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

RESULT: ???

```cpp
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

```cpp
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]

```cpp
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]

Example: grade cutoffs

suppose you are a teacher trying to determine grade cutoffs
- read students grades from a file (exact number is not known ahead of time)
- once grades are stored, repeatedly:
  - prompt teacher for cutoff value
  - traverse grades and count how many are ≥ cutoff

note: since might want to try out different cutoffs, need to store grades
- in general, avoid reading from a file over and over again

we could store the grades in an array, but this implies
- would have to give a MAX size for the array, possibly waste space

better choice would be a vector
- start with an INITIAL size, but resize if run out of space

Grades code

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

const int INITIAL_SIZE = 20;

void OpenFile(ifstream & myin);
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;

  while (cutoff != -1) {
    cout << "There are "
      << CountAbove(grades, numGrades, cutoff)
      << " grades at or above " << cutoff
      << endl;

    cout << "Enter another (or -1 to quit): ";
    cin >> cutoff;
  }

  return 0;
}
```

```cpp
void OpenFile(ifstream & myin)
// Results: myin is opened to user's file
{
    string filename;
    cout << "Enter the grades file name: ";
    cin >> filename;

    myin.open( filename.c_str() );

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

void ReadGrades(vector<int> & grades,
                int & numGrades)
// Results: reads grades and stores in vector
//          numGrades is set to the # of grades
{
    ifstream myin;
    OpenFile(myin);

    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)

- but copying can waste time and space

```
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;
}
```

alternative is to pass by-const-reference

- by-reference avoids making a copy, just passes a pointer – EFFICIENT
- const ensures that the vector cannot be changed – SAFE

```
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;
}
```

use const-reference when passing large structures that don't change!

9

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", const-reference is efficient

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)

10

ReadGrades (resizable vector vs. growable vector)

```cpp
void ReadGrades(vector<int> & grades,
                int & numGrades)
// Results: reads grades and stores in vector
//          numGrades is set to the # of grades
{
    ifstream myin;
    OpenFile(myin);

    numGrades = 0;
    int grade;
    while (myin >> grade) {
        if (numGrades == grades.size()) {
            grades.resize(2*grades.size());
        }
        grades[numGrades] = grade;
        numGrades++;
    }
    myin.close();
}
```

```cpp
void ReadGrades(vector<int> & grades)
// Results: reads grades and stores in vector
{
    ifstream myin;
    OpenFile(myin);

    int grade;
    while (myin >> grade) {
        grades.push_back(grade);
    }
    myin.close();
}
```

when create vector, must give an initial size
```cpp
vector<int> grades(INITIAL_SIZE);
```

when create vector, initially empty
```cpp
vector<int> grades;
```

requires separate counter, still wasteful
note: `numGrades != grades.size()`

don't need counter, size() is exact
but more resizing?

11

Improved version

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

void OpenFile(ifstream & myin);
void ReadGrades(vector<int> & grades);
int CountAbove(const vector<int> & grades,
               int cutoff);

int main()
{
  vector<int> grades;
  ReadGrades(grades);

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

  while (cutoff != -1) {
    cout << "There are "
         << CountAbove(grades, cutoff)
         << " grades at or above " << cutoff
         << endl;

    cout << "Enter another (or -1 to quit): ";
    cin >> cutoff;
  }

  return 0;
}
```

```cpp
void OpenFile(ifstream & myin)
// Results: myin is opened to user's file
{
    // SAME AS BEFORE
}

void ReadGrades(vector<int> & grades)
// Results: reads grades and stores in vector
{
    ifstream myin;
    OpenFile(myin);

    int grade;
    while (myin >> grade) {
        grades.push_back(grade);
    }
    myin.close();
}

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

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

    return numAbove;
}
```

12