Vectors

Jordi Cortadella
Department of Computer Science
Vectors

• A **vector** is a data structure that groups values of the same type under the same name.

• Declaration:  
  ```
  vector<type> name(n);
  ```

• A vector contains $n$ elements of the same type ($n$ can be any expression).

• **name[i]** refers to the $i$-th element of the vector ($i$ can also be any expression)

• Note: use  `#include<vector>` in the program
• Declaration of an empty vector:
  \texttt{vector\langle type\rangle a;}

• Declaration of a vector with an initial size (100 elements):
  \texttt{vector\langle type\rangle a(100);} 

• Declaration of a vector of 30 integers in which all elements have an initial value (7):
  \texttt{vector\langle int\rangle a(30, 7);} 

• Adding an element at the tail of the vector:
  \texttt{a.push\_back(x);} 

• Removing an element from the tail of the vector:
  \texttt{a.pop\_back();} 

• Reading the size of a vector:
  \texttt{a.size()}
push_back and pop_back operations

```cpp
vector<int> a;  // a.size()=0

a.push_back(3);
a.push_back(5);
a.push_back(8);

// a = [3, 5, 8]; a.size()=3

a.pop_back();

// a = [3, 5]; a.size()=2
```
Example: counting the last element

• Write a program that reads a sequence of numbers and prints how many times the last element is repeated in the sequence.

Input:   3 8 7 5 7 2 4 7 9 8 1 3 4 7
Output:  7 appears 4 times

• Rationale: there is no way to count the number of occurrences of a number without knowing the number.

• Strategy: store the sequence somewhere (in a vector).
// Input: a sequence of integer numbers.
// Output: A message indicating the number of occurrences of
//         the last number has been printed.
int main() {
    // Store the sequence in a vector
    vector<int> S;
    int n;
    while (cin >> n) S.push_back(n); // Adds the element at the tail

    if (S.size() == 0) { // Special case (empty sequence)
        cout << "The sequence is empty" << endl;
        return 0;
    }

    int last = S[S.size() - 1];
    int count = 0;
    for (int i = 0; i < S.size(); ++i) {
        if (S[i] == last) ++count;
    }
    cout << last << " appears " << count << " times " << endl;
}
Vectors: be careful

• A reference to a vector may not always exist. Example, if i=25 and x has 10 elements, then x[i] does not exist. A runtime error might be produced.

• Aliases: x[i] and x[j] refer to the same element if i and j have the same value:

```cpp
i = 4;
j = 3;
A[i] = 5;
A[j + 1] = 6;
cout << A[i] << endl; // Prints 6
```
Vectors: be careful

```cpp
vector<int> x(5);
x[0] = 0;
x[1] = 0;
x[2] = 0;
x[3] = 0;
x[4] = 0;

x[x[0]] = 1;
cout << x[x[0]] << endl; // What does it print?
```

Prints 0
Min value of a vector

// Pre: A is a non-empty vector.
// Returns the min value of the vector.

int minimum(const vector<int>& A) {
    int n = A.size();
    int m = A[0]; // visits A[0]
    // loop to visit A[1..n-1]
    for (int i = 1; i < n; ++i) {
        if (A[i] < m) m = A[i];
    }
    return m;
}
Constant parameters and variables

• A call-by-value parameter requires a copy of the parameter (inefficient for large vectors).

• Call-by-reference is more efficient, but the callee may unexpectedly modify the parameter.

• `const` parameters can be passed by reference and be protected from any modification.  
  – They cannot be written inside the function or passed to another function as a non-`const` parameter.

• Variables can also be declared as constant (only writable during declaration).
const double Pi = 3.14159; // Constant variable

void g(vector<int>& V) {
    ...
    V[i] = V[i - 1] + 1; // Allowed (V is not const)
    ...
}

int f(const vector<int>& A) {
    ...
    A[i] = A[i - 1] + 1; // Illegal (A is const)
g(A); // Illegal (parameter of g is not const)
Pi = 3.14; // Illegal (Pi is const)
    ...
}
Average value of a vector

• Given a non-empty vector, return the average value of the elements in the vector.

    // Pre: a non-empty vector A.
    // Returns the average value of the elements in A.

    double average(const vector<int>& A) {
        int n = A.size();
        int sum = 0;
        for (int i = 0; i < n; ++i) {
            sum = sum + A[i];
        }
        // Be careful: enforce a “double” result
        return double(sum)/n;
    }
Reversing a vector

• Design a procedure that reverses the contents of a vector:

• Invariant:
void reverse(vector<int>& A) {
    int last = A.size() - 1;
    // Calculate the last location to reverse
    int middle = A.size()/2 - 1;

    // Reverse one half with the other half
    for (int i = 0; i <= middle; ++i) {
        int z = A[i];
        A[i] = A[last - i];
        A[last - i] = z;
    }
}
Reversing a vector (another version)

// Post: A contains the reversed contents
//       of the input vector

#include <vector>

void reverse(vector<int>& A) {
    int i = 0;
    int last = A.size() - 1;
    // Inv: The elements in A[0...i-1] have been
    //      reversed with the elements in
    //      A[last+1...A.size()-1]
    while (i < last) {
        int z = A[i];
        A[i] = A[last];
        A[last] = z;
        i = i + 1;
        last = last - 1;
    }
}

Introduction to Programming  © Dept. CS, UPC 15
The largest null segment of a vector

• A null segment is a compact sub-vector in which the sum of all the elements is zero.

• Let us consider vectors sorted in increasing order.

-9 -7 -6 -4 -3 -1 3 5 6 8 9

Null segment

-9 -7 -6 -4 -3 -1 3 5 6 8 9

Largest null segment

-9 -7 -6 -4 -3 -1 3 5 6 8 9
The largest null segment of a vector

• Observations:
  – If a null segment contains non-zero elements, then it must contain positive and negative elements.
  
  – Let us consider a segment of a vector. If the sum of the elements is positive, then the largest positive value cannot belong to any null segment included in the segment.
  
  – The same is true for negative numbers.
The largest null segment of a vector

- Invariant:

  - The largest null segment is included in the [left...right] segment
  - sum contains the sum of the elements in the [left...right] segment

**Observation**: the search will finish when sum = 0. If the segment becomes empty (no elements) the sum will become 0.
The largest null segment of a vector

// Pre: A is sorted in increasing order.
// Post: <left,right> contain the indices of the
// largest null segment. In the case of an empty
// null segment, left > right.

void largest_null_segment (const vector<int>& A,
                          int& left, int& right)
{
    left = 0;
    right = A.size()-1;
    int sum = sum_vector(A); // Calculates the sum of A
    while (sum != 0) {
        if (sum > 0) {
            sum = sum - A[right];
            right = right - 1;
        } else {
            sum = sum - A[left];
            left = left + 1;
        }
    }
    // sum = 0 and the largest segment is A[left...right]
}
**typedef**

- **Typedef** declarations create *synonyms* for existing types:

```cpp
// Declaration of the type
typedef vector<double> listTemperatures;

// Declaration of a variable
listTemperatures MyTemp;

// The parameter of a function
double maxTemp(const listTemperatures& L) {
    ...
    
    }
```
Summary

• Vectors are data structures that store values of the same type.

• Unlike sequences, elements can be read and written at any time.

• Passing vectors as parameters:
  – Try to avoid passing by value (inefficient).
  – Pass by constant reference whenever possible.