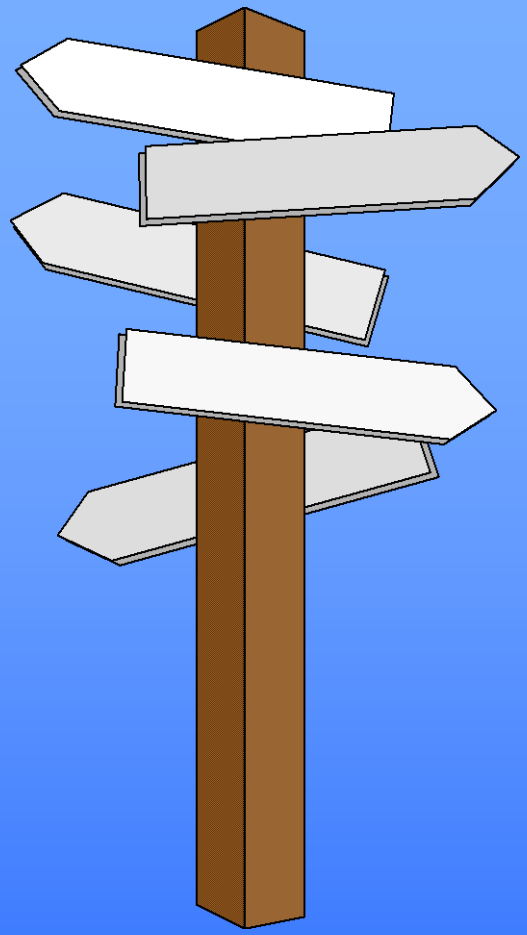


Chapter 8: Vectors

Mr. Dave Clausen

La Cañada High School

Objectives

- Understand the role of a vector as a fundamental data structure
- Declare and use a vector variable
- Write functions that process vectors
- Sort and search a vector

Variable Types

- So far we have been using 4 simple data types:
  - double, int, char, bool
- We are starting to create our own user defined types called “Structured Types”
  - Our first Structured type was the enumerated type.
  - Our second Structured type is called an “array” (in this case a vector, or safe array).

Vectors

- Arrays or vectors allow us to replace long lists of variables (that have different names) with one variable name (storing the same type of data).
  - For example, we can store several grades and perform calculations on them, without losing the actual grades.
- Vectors are data structures where each element is accessed using indexed positions.
  - Each index describes the relative position of an element in the vector.

Vectors

- Store several objects of the **same data type** in sequential order
- Locate objects with an integer index
- A data structure - can treat many elements (“things” – data entries) as one “thing”.

Declaring a Vector Variable

```
#include <vector>
int main()
{
vector<int> numbers;
```

vector type item type vector variable name

- Item type can be any data type.

- A vector has no cells for items by default;
- its length is 0 (zero)

Declaring Vectors

- When declaring a vector, we need to tell the computer:
  - the type of items to be stored (int, double, char, or string), called the base type
    - **these data elements must all be the same data type**
  - We don't have to declare the maximum number of items to be stored. A vector can be "resized" while the program is running.

Format for Declaring Vectors

- The general format is:

vector <component type> <variable name> (<integer value>);

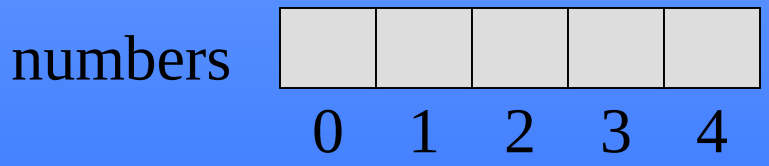
- **We must: #include <vector>**
- **start with: vector**
- **component type is any predefined, or user defined data type: this must be in angle brackets < >**
- **variable name is any valid identifier**
- **The integer value declares how many memory locations to reserve counting from *index numbers* 0 to N-1 in parentheses (where N represents how many elements you wish to use).**
- **vector uses dynamic memory, so you can ask the user how much memory to reserve.**

Specifying a Vector's Length

```
#include <vector>
int main()
{
  vector<int> numbers(5);
```

vector length

vector type item type vector variable name



- A vector's length can be specified when the variable is declared.
- Items in cells are undefined.

Examples of vector declarations

```
vector <int> hours;      //Size is zero
vector <int> integers (100, -1);  //Size = 100
                                  //Every value = -1
cout<<"Enter the number of stocks to track: ";
cin>>num_stocks;
vector <double> stock_prices (num_stocks );

#include <vector>
const int MAX_LIST_SIZE = 20;
vector <string> words (MAX_LIST_SIZE );
```

Vector Indices and limits

- Individual items, elements, or components in the vector are referred to using an “index” or “subscript”.
  - **This index must be in the range of zero to N-1**
- Range Bound Error
  - If you use an index value less than zero or greater than N-1
  - Some C++ compilers will not stop the program, nor will there be an error message.
- **Or an assertion in vector will halt the program with an error message**.

Vectors and indices

- If we define the vector:

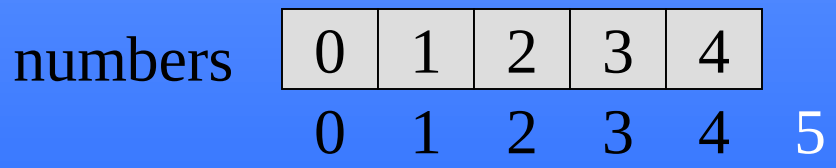
  vector <int> list (5);

  - we could visualize it as follows:
  - note that we start counting the indices from 0 (zero) to N-1 (one less element than declared)

Range Bound Errors

```
int main ()
{
vector<int> numbers(5);


for (int i = 0; i <= numbers.size(); i++)
   numbers[i] = i;
```



The program will halt with a run-time error when **`numbers[5]`** is referenced

Vector Elements

- The contents of each “box” is called an “element” or “component” of the vector, and may store one int, double, char, bool, or string value (whatever was defined as the base type or component type).
- When the Vector is declared, each element in the vector is in an “unpredictable state” until you assign values to each element.

Initializing A Vector

- Since each element in a vector is undefined, it is considered good programming to “initialize” each element of each vector that you declare.
- This means assigning a value to each element that you would not expect to see as a typical value in that vector.

Initializing Values

- For example:
  - A grade of -99 is rarely seen in the normal context of grades, so assigning every element equal to -99 (or another negative number) is a good choice.
  - If you were storing characters in a vector, you may want to initialize each element to contain one space. ‘ ‘, or the null character ‘\0’, which contains nothing.
  - A vector of strings could be initialized to the empty string, “”.

Specifying a Fill Value

```
#include <vector>
int main()
{
  vector<int> numbers(5, 1);
```

vector fill value

vector length

vector type

item type

vector variable name

numbers	1	1	1	1	1
	0	1	2	3	4

A default value for each item can be specified when a vector is declared.

Initializing A Vector

- Vectors are easily initialized when they are declared using the “fill value”.

```
cout<<“Enter the number of elements for the vector: “;
cin>>vector_size;
cout<<“Enter the initializing value for the vector: “;
cin>>initial_value;

vector <int> list (vector_size, initial_value);
```

A safe array class (Vectors)

- The safe array class should include the following features:
  - It should support the subscript or index operation.
  - The user can specify the size of the “vector”.
  - The “array” should be easily initialized with values set to a “fill value”.
  - Vectors should support the assignment statement to copy the contents of one vector into another.
  - Vectors should be able to adjust their size during assignment operations.

Comparing an Array with a Vector

- Here is a sample program using arrays:

  ARRAYAVDev.CPP

- Here is a sample program using vector:

  VECTORAVDev.CPP

  - The biggest difference between the two programs is that “vectorav.cpp” allows the user to declare how many elements they want to use in the vector.
  - The other difference is to include:
    - #include <vector>

The size function

- The vector class has a member function to indicate the “size” of the vector.
  - This indicates the length (size) that the vector was declared, not necessarily the number of valid elements that are stored in the vector.

physicalSize = list .size( );

//Since ***size*** is a member function of the vector class, we use the dot notation.

The Resize Function

vector includes a function to “resize” the vector while the program is “running”.

```
// description:  resizes the vector to newSize elements
// precondition: the current capacity of vector is size( );
//newSize >= 0
// postcondition: the current capacity of vector is newSize; for
//each k
//              such that 0 <= k <= min(size, newSize), vector[k]
//              is a copy of the original; other elements of vector are
//              initialized using the 0-argument itemType constructor
//              Note: if newSize < length, elements may be lost
```

Example of Resize

```
const int SIZE = 5;
vector <int> scores(SIZE);
cout<<scores.size( );
//Ask the user for the new size of the vector
cout<<“Enter the number of elements for the vector: “;
cin>>new_size;
//Resize the vector, if it is not too small
if (new_size > scores.size( ) )
  scores.resize(new_size);
//Remember: if the new_size was smaller than the original
    vector length, data can be lost.
```

Structured Types

- A vector is a structured type. Vectors can vary in size, therefore, few commands can be used on the entire vector.
- You cannot use “cin” or “cout” with the entire vector, only with individual elements in the vector.

Operations on Data In A Vector

- A definite or indefinite loop is used to perform many operations with a vector, including:
  - Entering the data
  - Printing the data
  - Searching the vector for a key value
  - Sorting the data in the vector
  - Performing calculations on the values in the vector.

Entering Data into a Vector

```
for(int index=0; index < grade.size( ); ++index)
{
    cout<<"Enter Grade # "<< index <<" : ";
   cin>>grade [ index ];
}
```

- An indefinite loop would be much better if you don't want to fill all the elements of the vector.

0	1	2	3	4
95	87	73	99	85

Displaying the contents of a Vector

```
void Print_Vector (const vector<double> &grade)
{
for(int index=0; index <  grade.size( ); ++index)
        cout<<“grade # “<<index<<“ is: “
   <<grade[index] <<endl;.
}
//Use constant reference parameters when vectors are
// used but not changed in a function.
```

Displaying the contents of a Vector using rows & columns

```
for(int index=0; index < grade.size( ); ++index)
{
    cout<<setw(5) <<grade[index] ;
     if (index % 10 = = 0)
        cout<<endl;
}  //Students: Modify this to fix the logic error here.
```

This should display10 grades across each row before sending a carriage return (endl) for the next row.

Vector Algorithms: Average of a Vector

- To find the average of all the elements in a vector:

```
sum = 0;
for(int index=0; index <  numscores; ++index)
   sum =sum + scores [index];
assert(numscores !=0);
average = double(sum) / double (numscores);
//don't forget explicit type conversion
//numscores is the logical size of the vector.
```

Vector Algorithms: Maximum score of a Vector

- To find the maximum score of all the elements in an vector:

```
max = scores[0];
for(int index=0; index < scores.size( ); ++index)
   if (scores[index] > max)
         max = scores[index];
```

Vector Algorithms: Finding the Index of the Maximum score of a Vector

- To find the index (position) of the maximum score of all the elements in a vector:

```
index_max = 0;
for(int index=0; index < scores.size( ); ++index)
   if (scores[index] > scores[index_max])
         index_max = index;
```

Vector Parameters and Functions

```
//Function: Get_Data
//Get scores from the user at the keyboard until
//the SENTINEL is entered.
//
//Inputs: list
//Outputs: a vector of integers representing the list and logical_size
//representing the vectors logical size

void Get_Data (vector <int> &list , int & logical_size);
//Use reference parameters, &, when passing vectors.
```

Vectors and Functions

- **Vectors need to be passed as reference parameters to functions.**
- Be aware that any changes to the vector in a function, change the actual vector, not a copy of the vector as is done with value parameters.
- The vector is not actually passed to the function, only the address of the vector, as is the case with reference parameters.
- You should pass the actual number of valid elements in the vector to a function, but you do not need to send its size. We can find the physical size of the vector using the size function.

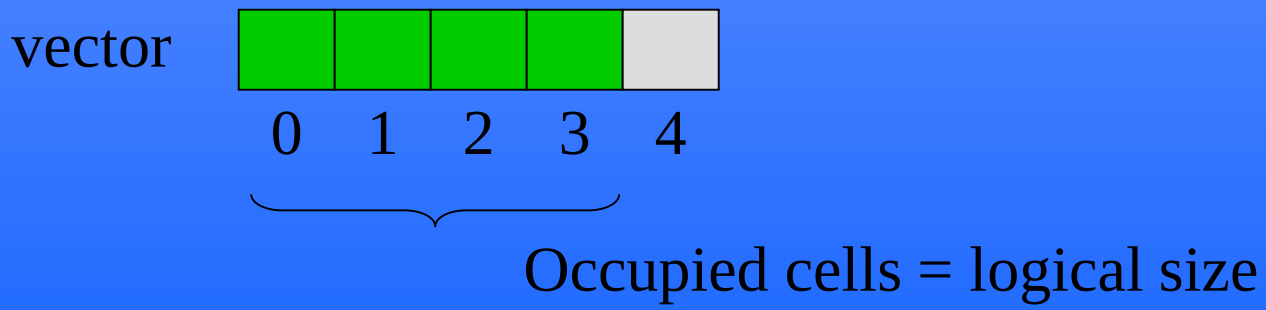
Vectors and Functions 2

- list.size( ); represents the **physical size** of the vector.
- The logical_size parameter in Get_Data function represents the **logical size** of the vector when the function returns the value.
- The **physical size** of an vector is the number of memory units **declared** for storing data items in the vector.
- The **logical size** of an vector represents the **actual** number of data items in the vector at any given time.
- Use **const** vector <basetype> &vector_name when you want to ensure that no changes occur in the vector. (i.e. Display_Output function)

Physical Size vs Logical Size

- When created, vectors have a fixed capacity (number of physical cells)

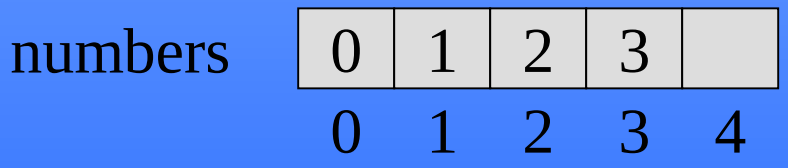
- The number of data items (logical size of the list) may not equal the capacity



Physical Size vs Logical Size 2

```
int main()
{
vector<int> numbers(5);

for (int i = 0; i < 4; i++)
   numbers[i] = i;
```



Physical size = number of cells declared in vector
Logical size = number of items currently stored and used by the program

Entering Data into a Vector Using Indefinite Loops

- A primed while loop will allow the user to enter data until they wish to quit by entering an input SENTINEL.
- This method also lets you calculate the logical length of the vector.

P434VECT.CPP

Example: Interactive Input

```
int main()
{
vector<int> v(10);              // Start with size of 10
int data;
int count = 0;

cout << "Enter a number or -999 to quit: ";
cin >> data;
while (data != -999)
{
   if (v.size() = = count)     // Expand by factor of 2 if no room
      v.resize(count * 2);
   v[count] = data;
   ++count;
   cout << "Enter a number or -999 to quit: ";
   cin >> data;
}

if (v.size() != count) // Shrink to number of items stored
   v.resize(count);
```

This keeps the physical size and the logical size the same.

Sorting & Searching Vectors

Sequential Search

Selection Sort

Bubble Sort.

Insertion Sort