Vectors

Chapter 11

For Next Time

- Read Chapter 11

Scalar Variables

- Variables we have seen and used so far can hold only one value at a time
  - We call these scalars
- Sometimes we need to store multiple data elements under one name
- C++ **`vector`** objects allow us to store a collection of data
  - Arrays are their more primitive counterparts

Declaring Vectors

- `#include <vector>`
- Full name: `std::vector`
  - `using namespace std;` takes care of this
- Vectors store homogeneous data
- Specify type of elements within angle brackets, e.g.:
  - `vector<int> list;`
  - `vector<double> list;`
  - `vector< vector<int> > list;`

Operations on Objects

- Also called
  - Methods
  - Member functions
- To call a method:
  - *object_name* **.** *method_name* **(***parameter_list***)**
  - The dot operator associates an object with a method to invoke on behalf of the object

Some Vector Methods

- **`push_back`**—inserts a new element onto the back of a vector
- **`pop_back`**—removes the last element from a vector
- **`operator[]`**—returns the value stored at a given position within the vector
- **`size`**—returns the number of values currently stored in the vector
- **`clear`**—makes the vector empty

Building a Vector

Building a Vector

```
vector<int> list;   // Declare list to be a vector
```

list

Building a Vector

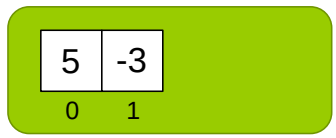
```
vector<int> list;   // Declare list to be a vector
list.push_back(5);  // Add 5 to the end of list
```

list

Building a Vector

```
vector<int> list;   // Declare list to be a vector
list.push_back(5);  // Add 5 to the end of list
list.push_back(-3); // Add -3 to the end of the list
```

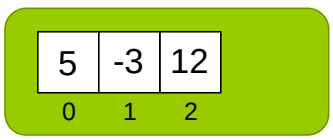
list



Building a Vector

```
vector<int> list;   // Declare list to be a vector
list.push_back(5);  // Add 5 to the end of list
list.push_back(-3); // Add -3 to the end of the list
list.push_back(12); // Add 12 to the end of list
```

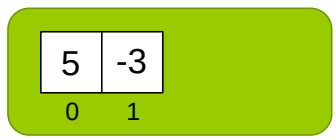
list



Building a Vector

```
vector<int> list;   // Declare list to be a vector
list.push_back(5);  // Add 5 to the end of list
list.push_back(-3); // Add -3 to the end of the list
list.push_back(12); // Add 12 to the end of list
List.pop_back();    // Remove last item pushed
```

list



Accessing an Element

- `operator[]` method
- Beginning index is zero
- `list[2]` is the third element
- Display all the elements in vector `list`:

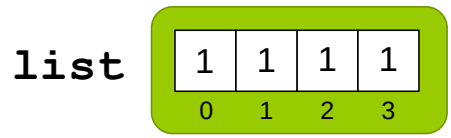
```
for ( unsigned i = 0; i < list.size(); i++ )
    cout << list[i] << endl;
```

- Do not attempt to access an element at an index less than zero or greater or equal to the vector's size!

Building a Fixed-size Vector

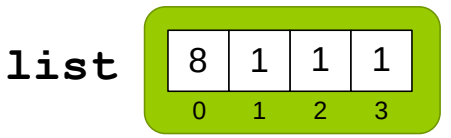
Building a Fixed-size Vector

```
vector<int> list(4, 1);   // list holds four 1s
```



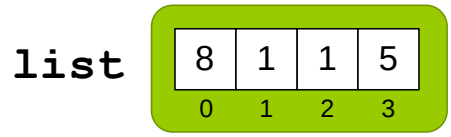
Building a Fixed-size Vector

```
vector<int> list(4, 1);   // list holds four 1s
list[0] = 8;    // Change item at index 0 to be 8
```



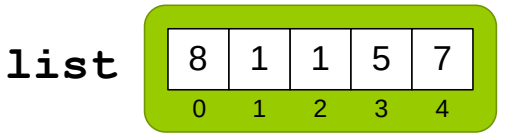
Building a Fixed-size Vector

```
vector<int> list(4, 1);   // list holds four 1s
list[0] = 8;    // Change item at index 0 to be 8
list[3] = 5;    // Change item at index 3 to be 5
```



Building a Fixed-size Vector

```
vector<int> list(4, 1);   // list holds four 1s
list[0] = 8;    // Change item at index 0 to be 8
list[3] = 5;    // Change item at index 3 to be 5
list.push_back(7); // Add 7 to the end of list
```



Vector Assignment

- One vector can be assigned to another vector
- All the elements are copied from one vector to the another

```
vector<int> v1, v2;
v1.push_back(5);  v1.push_back(3);
v2 = v1;
```

Vectors and Functions

- Vectors can be passed to functions
- Vectors can be returned from functions
- For efficiency, avoid passing vectors by value
  - Pass by reference, but use `const`
- For efficiency, avoid returning vectors from functions
  - Pass by reference, and do **not** use `const`

Multidimensional Vectors

- Vectors can hold other vectors
- A vector of scalar elements is a one-dimensional vector (1-D vector)
- A vector that holds 1-D vectors is a two-dimensional vector (2-D vector)
  - Table = all the contained vectors the same length
- Higher-dimensional vectors (3-D+) are possible but rarely used

Sorting

- A common task: arrange elements of a vector into ascending (or descending) order
- Many different sorting algorithms exist
  - Simple (slower) vs. complex (faster) sorts
- **Selection sort** is a simple to understand, simple to implement sorting algorithm
- **Quicksort** is the fastest known general-purpose sorting algorithm

Selection Sort

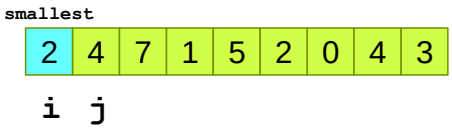
2	4	7	1	5	2	0	4	3

Selection Sort

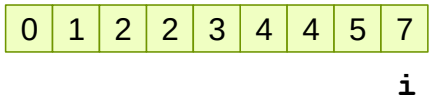
2	4	7	1	5	2	0	4	3

`i  j`

Selection Sort



Selection Sort



Search

- Linear search
  - Start at first element
  - Examine all the elements in turn until you find the one you seek
- Binary search
  - Elements must be in sorted order
  - Check middle element
    - If it is the sought element, you are done
    - If the middle element is larger than the sought element, search the front half of the vector
    - Otherwise, search the back half of the vector
  - Much faster than linear search in general

Next . . .

Other Standard C++ Objects