ITT DUBLIN
Institute of Technology Tallaght
Institiúid Teicneolaíochta Tamhlacht

Introduction to vectors

ITT DUBLIN
Institute of Technology Tallaght
Institiúid Teicneolaíochta Tamhlacht

Abstract Data Type

- A vector is an abstract data type that acts like a ‘smart array’
  - What is an abstract data type?
  - It is a well-encapsulated combination of a data structure and the operations that can be performed on that data structure ..
  - .. and well encapsulated means that even if the format of the data, or the implementation of the operations changes, code which uses the data structure (i,.e calls the operations) will continue to work as expected.
  - We will normally implement abstract data types as classes (well-encapsulated classes)
    - They are an ideal mechanism for encapsulating data
    - but it is perfectly possible to define a well-encapsulated data type without using classes

the vector data type

- Should be able to use it very like an array …
  - Access elements with square bracket notation as in an array
- But the vector will not have a fixed size
  - Can add extra elements to the end of the vector
  - Dynamically resize the vector whenever it runs out of space
  - Query the vector at any time to find out its current size
- As well as its size, the vector will also have a capacity
  - Which says how many more elements could be added before it will again be dynamically resized
  - Typically larger than the size, because resizing the vector is a complex task, so it make sense to reduce the number of times it has to be done by allocating capacity in largish chunks
- In order to provide this functionality, the vector will be implemented as a CLASS with appropriate data members and methods

The vector template class in C++

- C++ standard library provides a vector class
  - It is a 'template class' which means it can be used for a vector of any base type we choose
  - We will learn about template classes and how to write them next semester
  - There is a very rich collection of template classes in C++ called the standard template library (STL)
- For now we will look at how to use the vector class in the STL
  - As an exercise later we will try writing our own
  - It wont be as good as the one in the STL, but will be a useful learning exercise
- To declare a variable v for a vector of base type T

```
vector<T> v;
vector<int> w;       ←  w is a vector of base type int
vector<double> x;    ←  x is a vector of base type double
```

- vector<T> is a class name, so a declaration like those above cause the default constructor for the class to be called.
  - It will initialize the vector object as a vector that is empty (contains no elements) , although it could have a capacity

Accessing an element in a vector

- An individual item in a vector can be accessed just as in an array, using the `[]` operator.

  ```
  vector<int> w;
  …
  int a = w[4];
  ```

  - Using this method of access, it is still the programmers responsibility not to trample on, or access, memory outside the array bounds
  - It is left like this because of the efficiency trade-off (much faster access without bounds checking)

- An alternative access mechanism is provided by the `at()` method of the vector class.

  ```
  int b = w.at(4);
  ```

  - Using `at()` the array bounds will be checked at run time (exception thrown if there is a problem)

Some methods of the vector class

- Some methods (public member functions) of the vector class in the STL are
  - `push_back(element)` adds the element at the next available position in the vector
  - `size()` returns the size
  - `capacity()` returns the capacity

- As well as the default constructor for a vector, the starting size for the vector can be given when it is declared
  - `vector<double> v(20)` sets up a vector of 20 doubles, all initialised to zero
  - Of course the capacity of the vector may be greater than this

More methods:
Re-sizing a vector explicitly

- there is an operation to resize a vector …
  - **`resize(int sz)`**
  - if resizing to a larger size, the extra elements are initialized to zero
  - If resizing to a smaller size, the elements at the end of the vector are lost
- … and to resize the *capacity* of the vector

1. **`reserve(int cap)`**
   - To explicitly set the minimum capacity of the vector
   - If capacity is already greater than this, it will not be changed
   - This allows the programmer to make decisions based on the way they are intending to use the vector, to improve efficiency
   - The default behaviour if `reserve()` is not used, is always to double the present size of the vector whenever it runs out of space

Iterator over a collection

- A vector is just one of the collection types in the STL
  - Others are things like sets, lists ..
- There is an iterator type which will iterate over the elements of any type of collection, pointing to each one in turn
  - For a vector it is not so necessary, as we can iterate over the collection using a for-loop and index notation, just like we do for an array
  - But some of the other operations over a vector use an iterator as an argument, so we need to understand it.
- **`erase(iterator i)`** will erase the element that the iterator is pointing to, and 'close up the gap' in the vector.
  - This is likely to be a costly thing to do in terms of performance
  - Could we achieve the same effect any other way?
  - E.g. suppose the item to be removed is at the end of the vector?
- We can get at an iterator easily using the operator **`begin()`** which returns an iterator pointing to the beginning of the vector
  - We can then move the iterator along the vector by adding 1 to it for each move along
  - e.g to remove the element at index position 4 in the vector v

```
v.erase(v.begin() + 4)
```

- There is also an **`insert`** operation, which inserts an element at the iterator position, and moves everything along to make room for it

ITT DUBLIN
Institute of Technology Tallaght
Institiúid Teicneolaíochta Tamhlacht

Using the vector data type

To use the vector template class

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

<u>Exercise</u>

- Imagine you are writing a program to manage a shopping list. Each shopping list item is represented by a string. Your design requires a print function that prints out the contents of the shopping list.
- Using a vector to hold the shopping list items, write a print function to print out the contents of a vector of strings. Test your print function with a main program that does the following:
- Create an empty vector. Print it.
- Append the items, "eggs," "milk," "sugar," "chocolate," and "flour" to the list. Print it.
- Remove the last element from the vector. Print it.
- Append the item, "coffee" to the vector. Print it.
- Write a loop that searches for the item, "sugar" and replace it with "honey." Print the vector.
- Write a loop that searches for the item, "milk," and then remove it from the vector, as efficiently as possible. Print the vector.
- Search for the item, “chocolate" and find a way to remove it without changing the order of the rest of the vector. (This is not necessarily an efficient operation.) Print the vector one more time.

http://www.halpernwightsoftware.com/stdlib-scratch/exercises.html

Summary

- Abstract Data Type
  - a data type ‘wrapped up with’ the operations that can be performed on that data type.
- Vector data type is a smart array
  - an array together with the operations that can be performed on it.
- The vector template class defined in the STL
  - Some methods of this class
  - Iterators over this class