Vectors: A Song of Push and Pop

Lab 12: https://maryash.github.io/135/labs/lab_12.html

- Sadab Hafiz

Shortcomings of C++ Arrays

Some shortcomings of C++ Static Arrays:

- the size of the array needs to be known before declaration
- once initialized, items cannot exceed the size constraint
- arrays can't be returned by value from a function

In lab 9, we briefly discussed dynamic arrays. Using dynamic memory allocation, a C++ programmer can make their own array-like data structure class that can change in size depending on the number of items in the array. An example of such a dynamic array is **Vector**.

Introduction to Vector

Vectors are essentially dynamic arrays that can resize automatically at runtime. In addition, vectors can be returned from functions.

To use vectors, include the following: `#include <vector>`

Vector initialization requires the datatype. For example, here’s how to initialize an int vector called vec: `vector<int> vec;`

Unlike arrays, we don’t have to declare the size during initialization.

Adding new items to a Vector

```
#include <vector>
using namespace std;
int main(){
      vector<int> v;                                     // initialize empty vector
      v.push_back(10);                                   // add 10 to the back of the vector
      v.push_back(20);                                   // add 20 to the back of the vector
      v.push_back(30);                                   // add 30 to the back of the vector
      // v now contains elements [10, 20, 30]
      vector<int> new_v {10, 20, 30};                    // list initialize a vector
      // new_v also contains elements [10, 20, 30]
}
```

Accessing elements in a vector

By value: Use [ ] similar to arrays. For example: vec[0] would return item in 0 index of the vector vec as the datatype of the vector.

By reference: Use at() function to get an element by reference. For example: vec.at(0) to get item in 0 index.

First item: front() function returns the first element by reference

Last item: back() function returns the last element by reference

Size vs Capacity

The size of a vector is the number of elements in the vector. The capacity of a vector is the storage space currently allocated to the vector.

size( ) : returns the number of items in the vector
capacity( ) : returns the space allocated to the vector
empty( ) : returns true if the vector is empty(size is 0)

The capacity changes as we keep adding new items to the vector. To test this, create a few vectors, fill them with elements, and check the difference between their size and capacity as the number of elements increase.

Mutators (setters or modifiers)

push_back(n)	:	adds element n at the back (end) of the vector
pop_back( )	:	removes the last element in the vector
clear( )	:	removes everything from the vector

There are other functions within the vector class. You can learn more about those in the official documentation of vector.

As stated before, vectors are dynamic arrays. You can implement your own vector class using dynamic arrays.

Pairwise Sum

Implement the following function:

vector<int> sumPairWise(const vector<int> &v1, const vector<int> &v2);
Returns a vector of integers whose elements are the pairwise sum of the elements from the two vectors passed as arguments. If a vector has a smaller size than the other, consider extra entries from the shorter vectors as 0.

```
int main(){

    vector<int> v1{1,2,3};                         // initialize v1 as [1, 2, 3]

    vector<int> v2{4,5};                           // initialize v2 as [4, 5]

    sumPairWise(v1, v2);                           // returns [5, 7, 3]

}
```

Pseudocode

```
// everything in green is pseudocode

vector<int> sumPairWise(const vector<int> &v1, const vector<int> &v2) {

    . . . create two empty vectors small and large

    if (size of v1 is less than size of v2){

        . . . set small equal to v1 and large equal to v2

    }

    else {

        . . . set small equal to v2 and large equal to v1

    }

    for (i starts at 0 and goes upto the size of small) {

        large[i] = large[i] + small[i];

    }

    return large;

}
```