Vectors

Due this week

- **Project 2**
  - Write solutions in VSCode and paste in Autograder, **Project 2 CodeRunner**.
  - Zip your .cpp files and submit on canvas **Project 2**. Check the due date! **No late submissions!!**
  - Grading Interviews: Week after Spring Break
- 3-2-1 on Friday
- No class on Friday

Vectors

Array: Drawbacks

The size of an array cannot be changed after it is created

- you need to know the size **before** you define an array
- any function that takes the array as an input needs the **capacity/size** too
- wouldn't it be nice if there were something we could ***dynamically reshape***?!

Vectors

Dynamic array

- Not fixed in size when created
  - member function: [vector].size()
- Doesn’t require an auxiliary variable to track the size
- Can keep adding things to it, taking things out
- Header file
  - #include<vector>

Defining vectors

- When you define a vector, you must specify the type of the elements in angle brackets:

```
vector<double> data;
```

- **Default:** vector is created empty
- Like a string is always initialized to be empty:

```
string yeet; // yeet = ""
```

Similarities to arrays

- Here, the data vector `(vector<double> data)` can only contain doubles, same way an array `(double array[10])` could only contain doubles

- Can specify initial size in parentheses:

```
vector<double> data(10);
```

- Access elements using brackets:

```
data[i] = 7.0;
```

Examples

`vector<int> numbers(10);`	A vector of 10 integers
`vector<string> names(3);`	A vector of 3 strings
`vector<double> values;`	A vector of size 0 (empty)
`vector<double> values();`	**ERROR:** do not use empty () to create a vector

Accessing elements in a vector

- You access elements in a vector the same way as in an array, using an index and brackets:

```
vector<double> values(10);
// display the fourth element
cout << values[3] << endl;
```

- But a common error is to attempt to access an element that is not there:

```
vector<double> values(2);
// display the fourth element
cout << values[3] << endl;
```

Using vectors

How can we visit every element in a vector?

- With arrays, we could do:

```
for (int i=0; i < 10; i++) {
    cout << values[i] << endl;
}
```

Using vectors

How can we visit every element in a vector?

- With vectors:

```
for (int i=0; i < values.size(); i++) {
    cout << values[i] << endl;
}
```

- But with vectors, we don’t know if 10 is still the current size or not
  - use the .size() member function -- returns the current size of the vector
  - all those looping algorithms for arrays work for vectors too! Just use [vector].size()

Manipulating elements in vectors

Think of the vector a stack of papers

- Starts out empty `vector<int> papers;`
- Then somebody (say, the number 3) arrives
  - they go to the “back” of the line `papers.push_back(3);`

Manipulating elements in vectors

Think of the vector a stack of papers

- Starts out empty

```
vector<int> papers;
```

- Then somebody (say, the number 3) arrives
  - they go to the "back" of the line

```
papers.push_back(3);
```

- Then the numbers 5, 1 and 8 arrive, in that order
  - they each go to the "back" of the line (or top of the stack)

```
papers.push_back(5);
papers.push_back(1);
papers.push_back(8);
```

- **Check:** What now should be the elements of papers? papers.size()? What order?

Manipulating elements in vectors

- We can also remove elements from the back: `.pop_back()`
  - removes the last element placed into the vector
- Starting with `papers = {3, 5, 1, 8}` …
- We pick up paper 8 off the stack

```
papers.pop_back();
```

  - .pop_back() doesn't need an argument!
  - Just removes the last element
  - (whatever is at the top of the stack)
  - LIFO method
- **Check:** What now should be the elements of papers? papers.size()? What order?

Manipulating elements in vectors

Example: We can fill vectors from user input.

```
vector<double> values;
double input;
while (cin >> input) {
   values.push_back(input);
}
```

Vectors: initialization

- We can also initialize vectors like we have initialized arrays:

```
vector<int> your_money = { 0, 18, 7, 43, 4 };
```

- … is equivalent to…

```
vector<int> your_money;
your_money.push_back(0);
your_money.push_back(18);
your_money.push_back(7);
your_money.push_back(43);
your_money.push_back(4);
```

Arrays

- If you have two arrays: `int your_money[5]={ 0, 18, 7, 43, 4 };`
  `int my_money[5];`
- And further, we want what is stored in your_money to become my_money

- With arrays, we can not simply do this: `my_money = your_money;`
- Instead, we must loop:

```
for (int i=0; i < 5; i++) {
      my_money[i] = your_money[i];
}
```

Vectors

- With vectors, we can simply do this: `my_money = your_money;`

Other functions

- [vector].size() – returns currents size of vector
- [vector].at(i) – returns element at i^{th} position
- [vector].push_back(element) – add element to the back of vector
- [vector].pop_back() – removes the last in vector
- [vector].front() – returns first element in vector
- [vector].back() – returns last element in vector
- [vector].empty() – returns true if no element in vector