Vectors and Functions

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!!**
  - **EC:** smartPlaylist
- Grading Interviews after spring break
- Office hours session on Friday

Vectors as Function Parameters

Vectors as input parameters in functions

- How can we pass vectors as parameters to functions?
- … in the same way we pass arrays!
- But this time there are two cases:
  - we do not want to change the values in the vector
  - we do want to change the values in the vector

Vectors as input parameters in functions -- without changing the values

Example: Write a function to add up and return the sum of all the elements of an input vector of doubles.

```
double sum(vector<double> values)
{
  double total = 0;
  for (int i=0; i < values.size(); i++)
  {
     total += values[i];
  }
  return total;
}
```

- Note: this function **visits** each vector element but **does not** change them.

Vectors as input parameters in functions – and changing the values

Example: Write a function to multiply each element of an input vector of doubles by some factor.

```
void multiply(vector<double> values, double factor)
{
    for (int i=0; i < values.size(); i++)
    {
        values[i] = values[i] * factor;
    }
}
```

- Note: this function **visits** each vector element and **still does not** change them.

How do arrays work wrt functions?

- The key with arrays was that we **passed by reference**
  - the function would know where the array is **in memory** and modify it
  - so can we do the same with vectors?

```
void fillArray(int score[], int size)
{
    cout << "Enter 5 scores: \n";
    for(int i=0; i<5; i++)
    {
        cin >> score[0];
    }
}
```

```
void fillVector(vector<int> score)
{
    int input;
    cout << "Enter 5 scores: \n";
    for(int i=0; i<5; i++)
    {
        cin >> input;
        score.push_back(input);
    }
}
```

Vectors as input parameters in functions – and changing the values

Example: Write a function to multiply each element of an input vector of doubles by some factor.

```
vector<double> multiply(vector<double> values, double factor)
{
  vector<double> new_vec;
  for (int i=0; i < values.size(); i++)
  {
     new_vec.push_back(values[i] * factor);
  }
  return new_vec;
}
```

- Note: this function **returns a vector** of same size as the input vector (which is unchanged)

Warning – do not use this until project 3

(pass by reference)

Vectors as input parameters in functions – and changing the values – Pass by Reference

Example: Write a function to multiply each element of an input vector of doubles by some factor.

```
void multiply(vector<double>& values, double factor)
{
    for (int i=0; i < values.size(); i++)
    {
        values[i] = values[i] * factor;
    }
}
```

- Note: this function **visits** each vector element and **DOES** change them.


pass by reference
cup =
fillCup( )
pass by value
cup =
fillCup( )
www.penjee.com

Vector of Vectors

2D Vectors: a vector of vectors

- There are no 2D vectors, but if you want to store rows and columns, you can use a vector of vectors.

```
vector<vector<int>> counts;
//counts is a vector of rows. Each row is a vector<int>
```

- You need to initialize it, to make sure there are rows and columns for all the elements.

vector of vectors: advantages

The advantage over 2D arrays:

- vector row and column sizes don’t have to be fixed at compile time.

```
int COUNTRIES = . . .;
int MEDALS = . . .;
vector<vector<int>> counts;
for (int i = 0; i < COUNTRIES; i++)
{
     vector<int> row(MEDALS);
     counts.push_back(row);
}
```

vector of vectors

- You can access the vector `counts[i][j]` in the same way as 2D arrays.
- `counts[i]` denotes the ith row, and
- `counts[i][j]` is the value in the jth column of the ith row.

vector of vectors: Determining row/columns

- To find the number of rows and columns:

```
vector<vector<int>> values = . . .;

int rows = values.size();
int columns = values[0].size();
              values[1].size();
              values[2].size();
```

Arrays or vectors?

Short answer: Vectors are usually easier, and more flexible.

- Can grow/shrink as needed
- Don’t have to keep track of their size in a separate variable (vec.size())
- Pass-by-value

- But arrays are often **more efficient**. So beefier programs typically use arrays
- You still need to use arrays if you work with older programs or use C without the "++", such as in microcontroller applications.

Other functions for vectors

http://www.cplusplus.com/reference/vector/vector/