Vectors in C++

Arrays but not really

Prepared by Usman Mohammed
Presented by Simeon Ramjit

What is a vector?

- Vectors are containers that can store **elements** (variables of one datatype) but can change their size dynamically (TK ,2016)
- Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterator
- Part of the Standard Template Library (STL). They are efficient and reliable
- Can store integers, floats and string elements
- Preferred over arrays when managing ever-changing elements

Why use a vector?

- Size of datasets can be unknown at runtime so having a dynamically sized variable caters to that
- Vectors are (usually) faster when having to traverse them for searches
- They come with modifiers, iterators, element accessors and more all prepackaged without you having to write additional code
- They are usually resized automatically based on the operation performed

Vectors – Getting Started

- Here's the basic syntax

```
std::vector <type> variable(elements)
```

- type - data type stored in a vector (e.g., <int>, <double> or <string>)
- variable - name that you choose for the data
- elements - the number of elements for the data (optional)
- Examples:

```
std::vector <int> myVector (5);
std::vector <std::string> names(1);
```

Vectors – Declaration and Initialization

- Multiple ways to initialize and create a vector
- Here we create a vector that automatically gets assigned 0 for all elements

```
#include<vector>  //Include vector library
#include<iostream>

int main() {
 std::vector <int> myVector(5);  //Declare a vector of size 5 and type int
 for (int x : myVector) {
  std::cout << x << std::endl; //Try to guess the output
 }
 return 0;
}
```

Vectors – Manual Population

- We could also manually populate the vector just like we would an array

```
#include<iostream>
#include<vector>
#include<string>

int main() {

    std::string arr[] = { "first", "sec", "third", "fourth" };
    // Vector with a string array
    std::vector <std::string> vec_of_str(std::begin(arr), std::end(arr));
    for (std::string str:vec_of_str)
        std::cout << str << std::endl;

    return 0;
}
```

Vectors – Iterators

- Allows for access of the data elements stored in vectors
- An object that functions *like* a <u>pointer</u>
- Iterators type in C++ : Input, Output, Forward, Bidirectional and <u>Random Access</u>
- Random access is the type vector supports

Vectors – Iterators*

- `vector::begin()` returns an iterator to point at the first element of a C++ vector.
- `vector::end()` returns an iterator to point at past-the-end element of a C++ vector.
- `vector::cbegin()` is similar to vector::begin(), but without the ability to modify the content.
- `vector::cend()` similar to vector::end() but can't modify the content.

*Definitions Sourced from BitDegree(2019).

Vectors – Iterators example

- Note the way that an iterator is used with a vector

```
vector_name.iterator();
```

- Ignore the use of 'push_back' for now

```
#include<iostream>
#include<vector>
int main() {
 std::vector <int> g1; // creating a vector
 for (int i = 1; i <= 5; i++)
   g1.push_back(i);
 std::cout << "Output of begin and end: ";
 for (auto i = g1.begin(); i != g1.end(); i++) //auto?
   std::cout << *i << " "; // Pointer dereference
 return 0;
}
```

Auto

- Now. What. Is. Auto. 🙃
- The keyword **`auto`** does type inference, so it deduces the type of variable that's on the right hand side of the expression
- Without **`auto`** we'd have to type **`std::vector<int>iterator`** and we don't have that time 🙅‍♀️
- Do not use auto when you're unsure of what the variable type is supposed to be. Only use it when the right side expression is immediately resolvable i.e. not a function call

Vectors - Iterators

- Remember pointers? Try to guess why the output of this program isn't consecutively increasing

```
#include<iostream>
#include<vector>
int main() {
  std::vector <int> g1; //creating a vector
  for (int i = 1; i <= 3; i++){
   g1.push_back(i);
   }
  std::cout << "\nOutput of beginning and end addresses: ";
  for (auto i = g1.begin(); i != g1.end(); i++) {
   std::cout << &*i << " "; //Difference is here
  }
  return 0;
}
```

Vectors – Modifiers

- These methods operate over the entire vector. They change the properties of the vector.
- Here are some examples, there a quite a few:
  - vector::push_back() - pushes elements from the back
  - vector::insert() - inserts new elements to a specified location
  - vector::pop_back() - removes elements from the back
  - vector::erase() - removes a range of elements from a specified location
  - vector::clear() - removes all elements
- More can be found at https://www.geeksforgeeks.org/vector-in-cpp-stl/

Vectors – Modifiers Example

- Let's initialize a vector using the `assign()` modifier

```
#include<iostream>
#include<vector>
int main() {

  //Declare vector
  std::vector<int> v;
  // fill the array with the value 10, five times
  v.assign(5, 10);
  std::cout << "The vector elements are: ";
  for (int i = 0; i < v.size(); i++)
    std::cout << v[i] << " ";

  return 0;
}
```

Vectors – Modifiers Example

- And another one. `push_back()`

```
#include<iostream>
#include<vector>
int main() {
  //Declare vector
  std::vector<int> v;
  //Push a value into the vector from the end
  for (int i = 1; i <= 5; i++)
    v.push_back(i);
  std::cout << "The vector elements are: ";
  for (int i = 0; i < v.size(); i++)
    std::cout << v[i] << " ";
  return 0;
}
```

Vectors – Modifiers Example

- Let's look at how to erase or remove an element – **pop_back()**

```
#include<iostream>
#include<vector>
int main() {
    std::vector<int> v;
    for (int i = 1; i <= 5; i++)
        v.push_back(i); // 1 2 3 4 5
    // removes last element
    v.pop_back();
    // prints the vector
    cout << "The vector elements are: ";
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " "; //1 2 3 4
    return 0;
}
```

Vectors – Modifiers Example

- Let's look at how to erase or remove an element – `erase()`

```
#include<iostream>
#include<vector>
int main() {
    std::vector<int> v;
    for (int i = 1; i <= 5; i++)
        v.push_back(i);
    // removes the first element
    v.erase(v.begin());
    cout << "The first element is: " << v[0];
    return 0;
}
```

- What if we try `v.erase(v.begin(), v.begin()+3);`

Vectors – Modifiers Example

- Let's look at how to erase or remove an element – `erase()`

```
#include<iostream>
#include<vector>
int main() {
    std::vector<int> v;
    for (int i = 1; i <= 5; i++)
        v.push_back(i);
    // removes the first element
    v.erase(v.begin());
    cout << "The first element is: " << v[0];
    return 0;
}
```

- What if we try `v.erase(v.begin(), v.begin()+3);`

Vectors – Capacity

- These methods give us information about the size properties of the vector
- Capacity methods include:
  - `size()` – Returns the number of elements in the vector.
  - `max_size()` – Returns the maximum number of elements that the vector can hold.
  - `capacity()` – Returns the size of the storage space currently allocated to the vector expressed as number of elements.
  - `resize(n)` – Resizes the container so that it contains 'n' elements.
  - `empty()` – Returns whether the container is empty.
- Syntax is what we're accustomed to so far

```
vector_name.method();
```

Vectors – Capacity example

- Let's get some information about this vector

```
#include<iostream>
#include<vector>
int main() {
    std::vector<int> v;

    for (int i = 1; i <= 5; i++)
    v.push_back(i);

    cout << "Size : " << v.size();
    cout << "\nCapacity : " << v.capacity();
    cout << "\nMax_Size : " << v.max_size();
    return 0;
}
```

Vectors – Accessing elements

- Elements are accessed the same way in which arrays are accessed

```
#include<iostream>
#include<vector>
int main() {
 std::vector<int> v;
 for (int i = 0; i < 5; i++) {
   v.push_back(i);
 }
 for (int i = 0; i < v.size(); i++){
  std::cout << "v[" << i << "] " << v[i] << std::endl;
 }
 return 0;
}
```

- **Remember** that vectors can be automatically resized (except when removing an element) so elements won't always have the same position

Pro-Tips

- VS users – install the copyashtml extension to copy and paste code into Word without losing formatting
- Always do research on your algorithm before you start to code, there may be a library that can simplify your workflow
- Learn keyboard shortcuts, using your mouse takes time
- Consider opening up an account on Stack Overflow and Electrical Stack Exchange – Minimum Reproducible Example

More Vector Methods

- `vector::at()`
- `vector::back()`
- `vector::clear()`
- `vector::operator=`
- `vector::operator[]`
- `vector::front()`
- `vector::reserve()`
- `vector::resize()`
- `vector::shrink_to_fit()`
- `vector::data()`
- `vector::emplace_back()`
- `vector::emplace()`

Questions?

simeon.ramjit@sta.uwi.edu

github.com/simeon9696/programmingworkshop

References

- BitDegree. "The Basics of C Vector Explained With Examples." BitDegree. BitDegree, September 3, 2019. https://www.bitdegree.org/learn/c-plus-plus-vector.

- Tk. "C Vector: A Pretty Simple Guide." Medium. The Renaissance Developer, July 8, 2019. https://medium.com/the-renaissance-developer/c-standard-template-library-stl-vector-a-pretty-simple-guide-d2b64184d50b.

- "Vector in C STL." GeeksforGeeks, September 4, 2019. https://www.geeksforgeeks.org/vector-in-cpp-stl/.