C++: Vector intro, Sort

BTW

- Make sure you are keeping up with the content on the course website

Outline

- Vector
- Sort
- Find

Vector

- Part of standard library <vector>
- Container that holds a collection of values
- Type of object it holds specified in <>
- Grows as needed
- Allows easy access to individual values

```
struct Student_info {
    string name;
    double midterm, final;
};  // note the semicolon--it's required
```

```
//students holds a collection of
//student_info objects
vector<Student_info> students;
```

Vector - adding

push_back(element)

Adds a new element to the end of the vector

Makes COPY of element it adds (so can reuse element)

```
struct Student_info {
    string name;
    double midterm, final;
};  // note the semicolon--it's required
```

```
record.name = "Oliver";
record.midterm = 50;
record.final = 100;
students.push_back(record);
```

Vector - allocated in contiguous memory (you can use [] for random access)

- Show contiguous memory on board
- Show how indexing works

```
18 int main() {
19     vector<ms> mv;
20
21     ms var1;
22     for (int i=0;i<4;i++){
23         var1.v1=i;
24         mv.push_back(var1);
25     }
26
27     //set and hit breakpoint at return 0
28     //in expressions view
29     //sizeof(ms)  => returns 4
30     //&mv[0] => gets address of 1st element in vector
31     //&mv[1] => 4 away from 0
32     //&mv[2] => 4 away from 1
33     return 0;
34 }
```

- Demo 4_vector_prove_contiguous.git

Vector - Miscellaneous

- V.push_back(element) adds element to the back of the vector
- V.pop_back() removes the last element of the vector
- v.begin() returns iterator (a “pointer”) to first value in v
- v.end() returns iterator (a “pointer”) to last+1 value in v (1 past end)
- v[i] returns value stored at i. You must make sure this value exists else undefined behavior.
  - BTW this container[i] random access syntax works on contiguous memory containers only (std::string, std::array, std::vector). It does **not** work on non-contiguous memory containers (std::list and most other containers).
- v.size() Returns number of elements in v.

Vector - Miscellaneous

- v.empty() checks whether v is empty (boolean)
- v.reserve() reserve storage, use this if you know approximately how big your vector will grow
- v.capacity() how many elements can be held in current storage

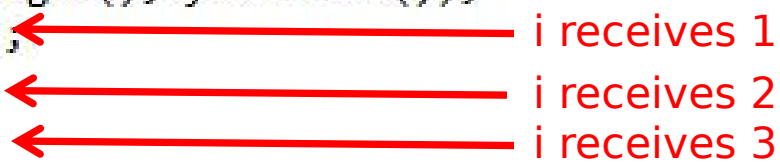
- v.clear() clears the contents
- v.erase() erase an element (returns iterator to next element, probably causes reallocation which is slow)
- v.size() Returns number of elements in v.

Sorting – simple

- Part of standard library <algorithm>
- Simple sorting – use if type container holds lends itself to comparison using < (int, double, string etc)
- Rearranges the container though, if need the original, make a copy

```
vector<int> myVect;
myVect.push_back(2);
myVect.push_back(1);
myVect.push_back(3);
int i = myVect[0];
i=myVect[1];
i=myVect[2];

sort(myVect.begin(),myVect.end());
i = myVect[0];
i=myVect[1];
i=myVect[2];
```



Sorting – complicated

- What if type is a struct that does not respond to <
- Sort takes a third parameter, a compare function

```
//used by sort algorithm
bool compareName(const Student_info& x, const Student_info& y)
{
    return x.name < y.name;
}
bool compareMidterm(const Student_info& x, const Student_info& y)
{
    return x.midterm < y.midterm;
}
bool compareFinal(const Student_info& x, const Student_info& y)
{
    return x.final < y.final;
}
//sort by name
sort(students.begin(), students.end(), compareName);
//sort by Midterm
sort(students.begin(), students.end(), compareMidterm);
//sort by Final
sort(students.begin(), students.end(), compareFinal);
```

See 'Vectors and Sorting' project

Vector – Finding stuff- brute force

```
vector<Student_info> students;

Student_info record;

record.name = "Oliver";
record.midterm = 50;
record.final = 100;
students.push_back(record);

for (Student_info &s : students) {
    if (s.name == "Oliver")
        std::cout << "found " << s.name << std::endl;
}
```

Vector – find_if
uses syntax we have not covered yet.

```
struct findByName
{
    std::string name;
    findByName(std::string name) : name(name) {}
    bool operator () (const Student_info& m) const
    {
        return m.name == name;
    }
};
```

default constructor with initializer list
Note you set the item you are looking for
And the operator compares what is passed in to it

```
vector<Student_info> students;

Student_info record;

record.name = "Oliver";
record.midterm = 50;
record.final = 100;
students.push_back(record);
vector<Student_info>::iterator itr;
itr = std::find_if(students.begin(), students.end(), findByName("Oliver"));

if (itr == students.end())
    std::cout << "did not find " << itr->name << std::endl;
else
    std::cout << "found " <<itr->name<< std::endl;
```

Find_if calls the () in findByName struct on every element in students, if it gets a hit it returns an iterator to it, otherwise it returns an iterator to one past the end (we will do iterators in a bit)

Summary

- Vectors and sorting
- Make sure value exists before dereference (use size())