C++ overview

Can I use C?

Absolutely not. Don’t even think about it.

If you know C, you already know C++. You just need to learn some additional features of the language.

Can I use Java?

Yes, if you are very comfortable with the language. But this course will use C++.

So it is recommended that you get at least some familiarity with C++ so that you are able to follow the classes. When you implement the programs, you can do it in Java.

4 awesome features of C++ over C.

---

1. Passing by reference (so you don’t have to use pointers that much).
2. No malloc. Use new and delete. Syntax is much simpler.
3. String library, and
4. Of course, STL! Makes life so much easier. (Standard Template Library).

For all C++ related stuff, learncpp.com is your go to website.

Passing by reference.

See code in sublime text.

String Library

You can simply treat string as a data type like an int, float or char. You would have seen some properties of this data type during your typing speed calculator assignment.

string str = “coding batch”;

Rest of the code in sublime.

Standard Template Library (STL)

What is it? It is a set of features that make coding easier. It has several features, but we only need concern ourselves with 2:

1. **Containers** -> vector, list, queue, stack, set, map. Anything that contains some list of data (an array is also a container).
2. **Iterators** -> Pointers that can traverse the above containers and insert/delete/modify values.

Syntax is slightly different from regular C, but if you understand generics you will get used to it (remember I sent an email?).

Standard Template Library (STL)

Where to study from? Download this pdf and go through chapters 5, 6, 7 (best possible resource).

https://github.com/regstrtn/coding-batch-2020/blob/master/CS106L%20Standard%20C%2B%2B%20Programming%20Laboratory%20full%20course%20reader.pdf

Or watch this youtube video series.

https://www.youtube.com/watch?v=bADtYBxrM8I&list=PLk6CEY9XxSIA-xo3HRYC3M0Aitzdut7AA&index=3

STL Container: Vector

Vector: This is exactly like an array, but you don't have to declare the size and can add values dynamically.

Example usage:

```
vector<int> v; // No need to declare size.

for(int i=0;i<100;i++) {

    v.push_back(i);   // You can only add element at the end.

}

for(i=0;i<v.size();i++) {

    cout<<v[i]<<endl;

}   // Similarly, there are functions for deleting element, copying a vector etc.
```

STL Container: Deque

Deque: Same as vector, but one advantage:

1. Can add elements at the end as well as front.

Example usage:

```
deque<int> dq; // No need to declare size.
for(int i=0;i<100;i++) {
    dq.push_back(i);
    dq.push_front(i);
}  // array looks like 100,99,...,2,1,0,0,1,2,....,99,100
for(i=0;i<dq.size();i++) {
    cout<<dq[i]<<endl;
}
```

STL Container: Set

Set: Does exactly what it means. This data structure ensures that you have exactly one occurrence of an element in an array.

See code in sublime.

STL Container: Map

Map: This is exactly like the python dictionary, but you have to declare the data type of both key and value.

```
map<int, string> m;

m[0] = “zero”;

m[1] = “one”;

map<string, string> languages;

langagues[“UK”] = “English”;

languages[“France”] = “French”;
```

STL Container: Pair

This container allows you to store two disjoint data types into a single variable. Why study this? Because the map data structure uses this pair to store values inside it (and yes, for each STL container, you need to know details about internal implementation).

```
int main() {

    pair<int, string> p;

    p.first = 91;

    p.second = “India”;

    printPair(p);

}

void printPair(pair<int, string> p) {

    cout<<p.first<<” “<<p.second<<endl;

}
```

STL: Iterators

Iterators allow you to access the elements inside a container (vector, deque, set, map).

Actually **vector and deque allow random access** so you don't need to use an iterator for them. You can simply access elements using v[0], v[1] etc.

Set and Map do not allow random access, so only way to visit every element is to use an iterator. The syntax for iterators is a little tricky, but you just have to memorize it, there is no way around it.

Every iterator is declared as below.

```
ContainerType::iterator iter = containerVariableName.begin();
```

```
// Think of the iterator iter part as simply like int x.

// iter is just the name of the iterator, you can name it anything.
```

STL Iterator example: Set

Iterator syntax template:

```
ContainerType::iterator iter = containerVariableName.begin();
```

Example of a set<string> iterator:

```
set<string> s;

s.insert("France"); s.insert("UK");

set<string>::iterator it = s.begin();
```

ContainerType (notice that you also need to include <string> part.

This part is just like saying int x;

containerVariableName.begin()

What is s.begin()? .begin() returns a pointer, pointing to the first element of the container. (See example in sublime).

STL Iterator example: Set

Iterator syntax template:

```
    ContainerType::iterator iter = containerVariableName.begin();
```

Example of a set<string> iterator:

```
set<string> s;

s.insert(“France”); s.insert(“UK”);

set<string>::iterator it = s.begin();

for(;it!=s.end();it++) {

    cout<<*it<<endl;   // Notice the dereference operator.

                       // Why? Because iterator is a pointer.

} // Output: “France”, “UK”.
```

s.begin() points to the first element, but s.end() does not point to the last element.

STL Iterators: others

Similarly, you need to figure out how to traverse a map and other containers.

Go through the pdf that was mentioned earlier, it is an amazing book and will make you a better engineer overall.

Is this all?

No, we have barely scratched the surface of containers. You need to go through the pdf mentioned (or any other source on google), and study all the containers. For each container you need to know:

1. **Underlying implementation** of the container (array, linkedlist, tree).
2. **Operations**: insert, delete, search, size etc. Syntax of these operations. When does an operation throw error.
3. **Time complexity of these operations** in different containers (eg map vs unordered_map vs multimap, set vs unordered_set, vector vs deque).

What will we do in our next classes?

In our upcoming classes we will cover the following from scratch and in detail:

1. **Binary search algorithm** (the most important algorithm according to me. If you understand this algorithm, you will be able to code other things).
2. **Two pointer method** (very simple).
3. **Insertion sort.**
4. **Linked list:**
    a. Search and insert.
    b. Node deletion.
    c. Reversing a linked list
    d. Reversing a doubly linked list.