C++ slides - 9

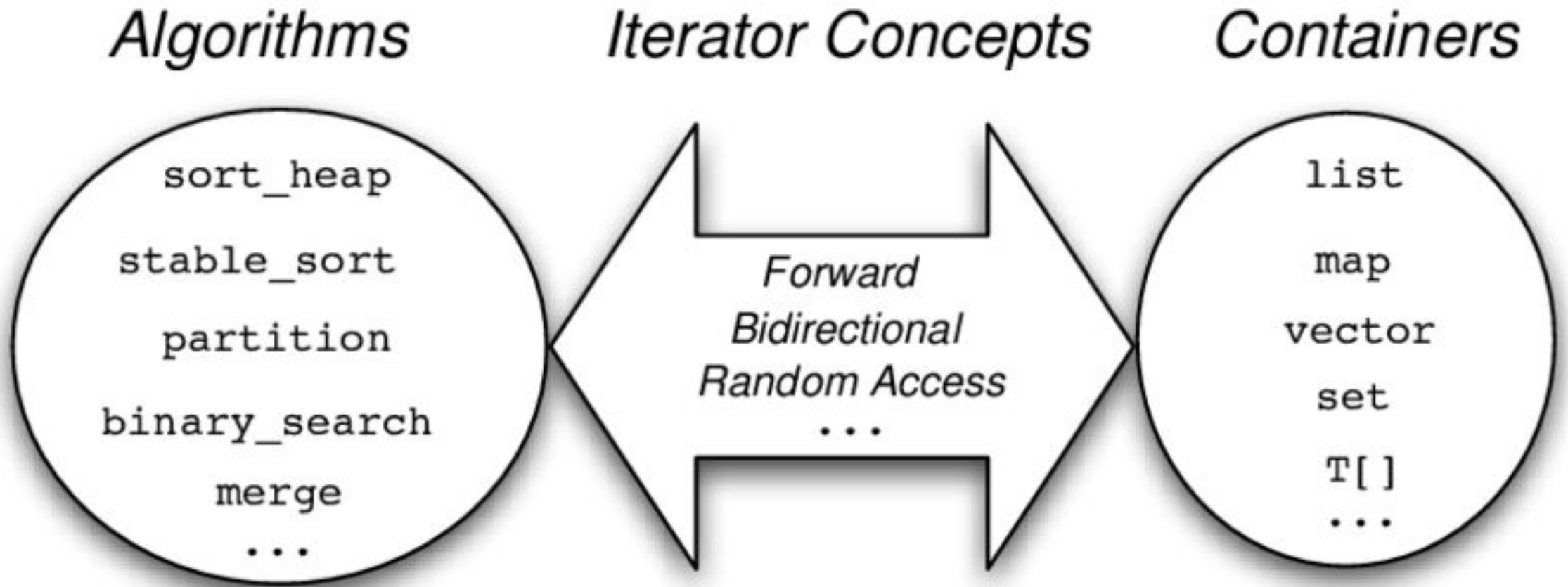
Standard Template Library: Fundamental idea about string, iterators, hashes and other types, The String and Vector classes vs C-style pointers

Standard Template Library (STL)

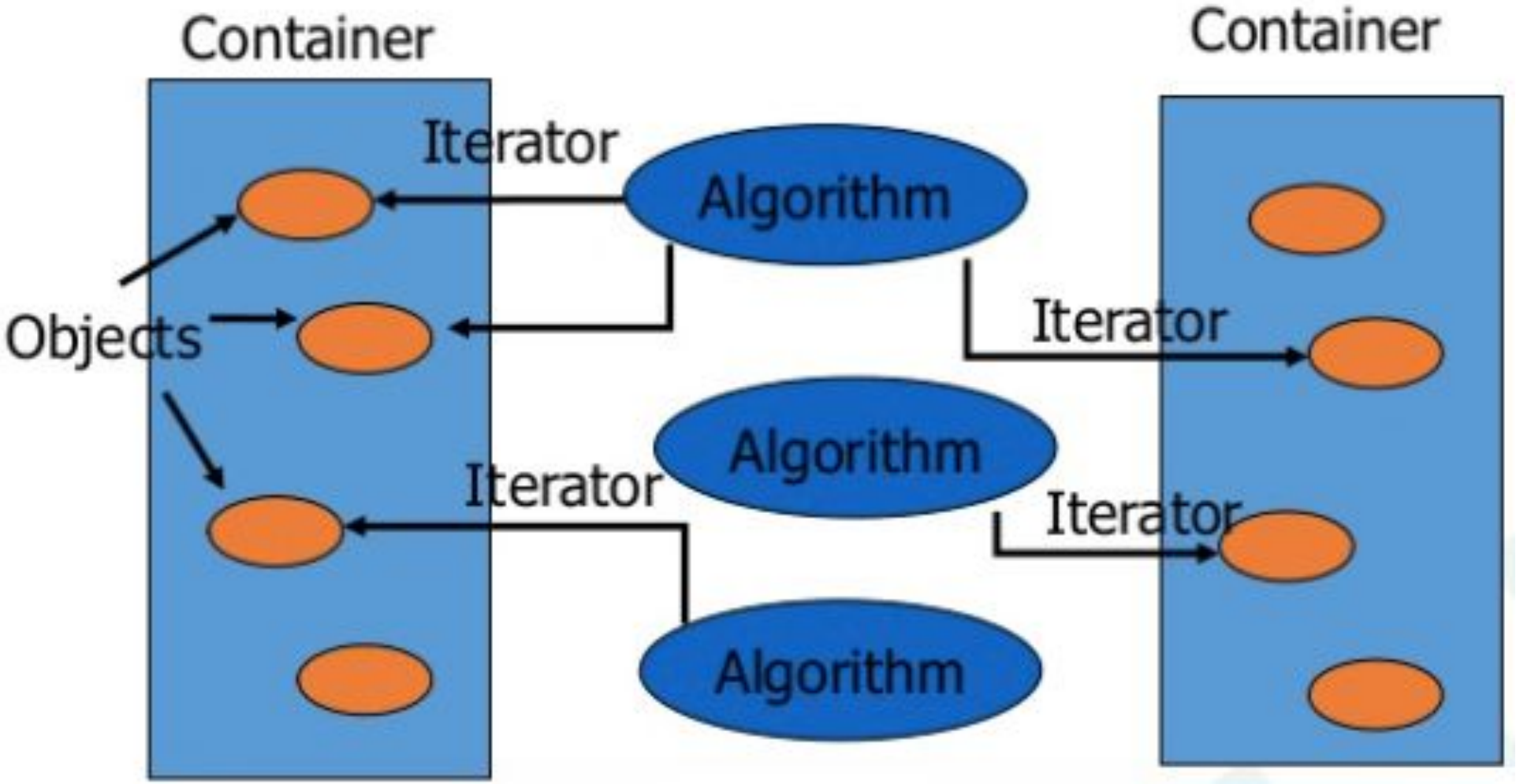
- STL provide general-purpose classes & functions to implement commonly used algorithms and data structures like vectors, lists, queues, and stacks.

STL has 3 components

- Containers – are the objects to organise/store various types of data such as vector, list
- Algorithms – process data in containers e.g. sort()
- Iterator – generalised concept of pointers to point an element in a container e.g. forward, bidirectional etc.

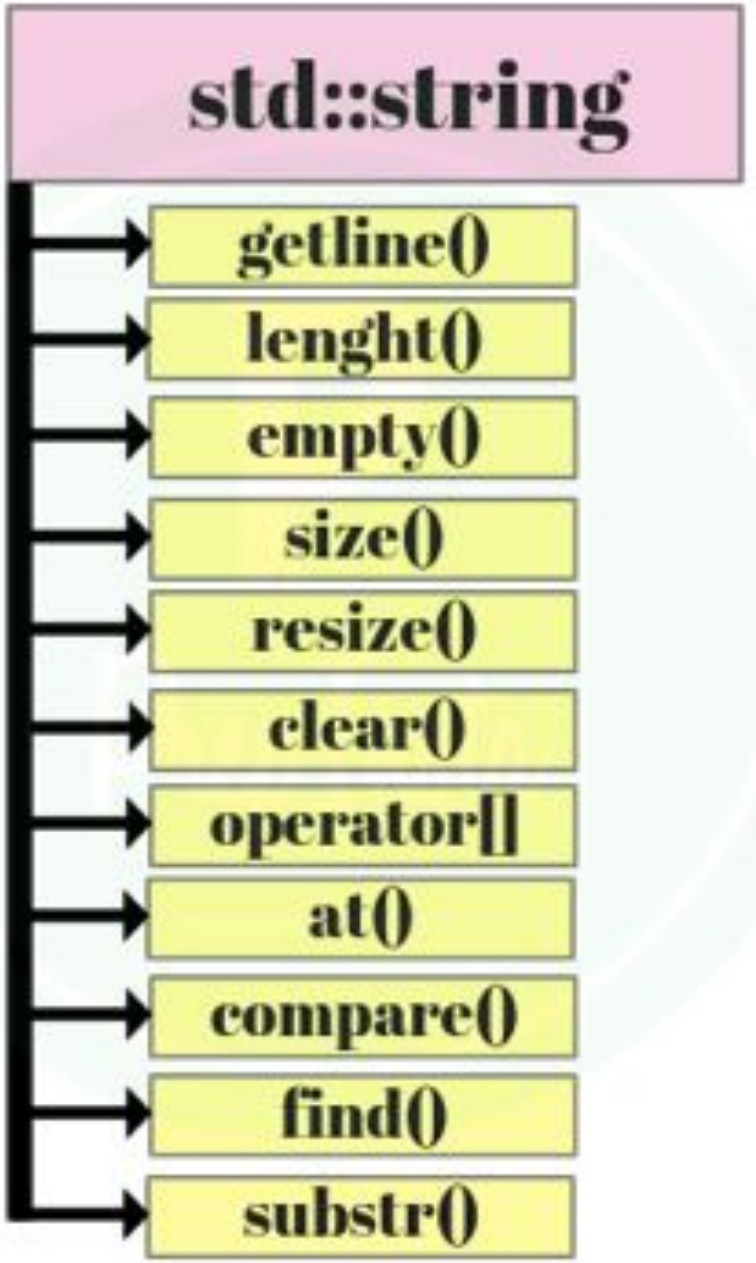

Algorithms
Iterator Concepts
Containers
sort_heap
stable_sort
partition
binary_search
merge
...
Forward
Bidirectional
Random Access
...
list
map
vector
set
T[]
...

Algorithms use iterators to interact with containers



C++ string (as an STL) vs char array

- A string is an inbuilt C++ class that contains a char array but automatically manages it using built in functions



Basic string program

```
#include <string>
#include<iostream>
using namespace std;
int main(){
string greeting ="Hello";
cout<<greeting;
}
```

String concatenation using ‘+’

```
string firstName = "John";

string lastName = "Doe";

string fullName = firstName + " " + lastName;
```

Some basic string functions

```
string txt = ”aBcDeF";

cout << txt.length();

cout<< txt[0];  //for the first letter
```

Some basic string functions

```
getline(cin,txt);  //for including blank spaces

txt.substr(1, 3);   // substring BcD (from 1 till 3 char)

str1.compare(str2);  // compare string str1 with str2
```

Hashing for faster access & efficient storage

- Hashing maps given *value* to a particular *key*. This helps in faster access.

- Let H(x) maps the *value* at *key* **x%10** in an Array. Then *values* {11,12,13,14,15} will be stored at {1,2,3,4,5} *keys*.

- Hashing is one way encryption which means you cannot go from *key* to *value*. (1 to 11 for example)

Header file for hashing programs

- To run the following hash programs, you need #include<bits/stdc++>
- DevCpp may not have this file so either you can download and include it using #include”bits/stdc++”
- OR use an online compiler https://www.onlinegdb.com/online_c++_compiler

```
// Character hashing


#include<iostream>
using namespace std;
int main(){
    hash <char> hash_character;  //syntax for char type
    cout << "Hash key is: " << hash_character('a');
}
```

Output

Hash key is: 97

Looks like hash key is getting calculated same as ASCII value of the character (for char case)

// Integer hashing

```
#include<iostream>
using namespace std;
int main(){
    hash <int> hash_int;
    cout << "Hash key is: " << hash_int(597)<<endl;
    cout << "Hash key is: " << hash_int(-597)<<endl;
}
```

Output

Hash key is: 597

Hash key is: 18446744073709551019

Looks like hash key for positive integer is same as its value and for negative it is a complicated code (may be t’s compliment, not sure)

```
// String hashing

#include<iostream>
using namespace std;
int main(){
    hash <string> hash_string;
    cout << "Hash key is: " << hash_string("a")<<endl;
    cout << "Hash key is: " << hash_string("apple")<<endl;
}
```

Output

Hash key is: 4993892634952068459

Hash key is: 1377659774762457284 8

For both “a” and “apple” the hash values are complicated codes calculated internally.

Vector (another STL)

- Vector is a dynamic array with a size flexibility and direct access to any element.

```
// Simple vector to insert int values
#include<vector>
#include<iostream>
using namespace std;
int main()  {
vector<int> v1;
v1.push_back(11);
v1.push_back(22);
vector<int>::iterator i;
for(i=v1.begin();i!=v1.end();++i)
  cout << *i << endl;
}
```

Output

11

22

#include<algorithm> for vector functions

By including *algorithm* header file, many inbuilt functions can be executed on vector such as:

- sort (values.begin(), values.end());
- reverse (values.begin(), values.end());
- random_shuffle (values.begin(), values.end());
- count (values.begin(), values.end(), 0); //count zeros

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main()  {
vector<int> v1;
v1.push_back(11);  v1.push_back(33); v1.push_back(22);
sort (v1.begin(), v1.end());
vector<int>::iterator i;
for(i=v1.begin();i!=v1.end();++i)
  cout<<*i<<endl;
cout<< "Max = "<<*max_element (v1.begin(), v1.end());
}
```

Output

11

22

33

Max = 33

String and Vector classes vs C-style pointers

(1) String constant of C++

In C++ one must use *const* for int

const char* str = ”This is a const string in C++";

Otherwise warning message -

main.cpp: In function ‘int main()’: main.cpp:4:13: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings] char* str = "This is a const string in C++";

(2) Cannot modify a string

```
 int main() {
    char* str = "Hello";
//warning – it should be const char *
    str[1] = 'o';
}
```

Output - Segmentation fault

Using char[] modification is possible

```
int main() {
  char str[] = "Hello";
    str[1] = 'o';
    cout << str << endl;
}
// Output will be Hollo
```

(3) char* is faster

- string will likely have more overhead