International study centre

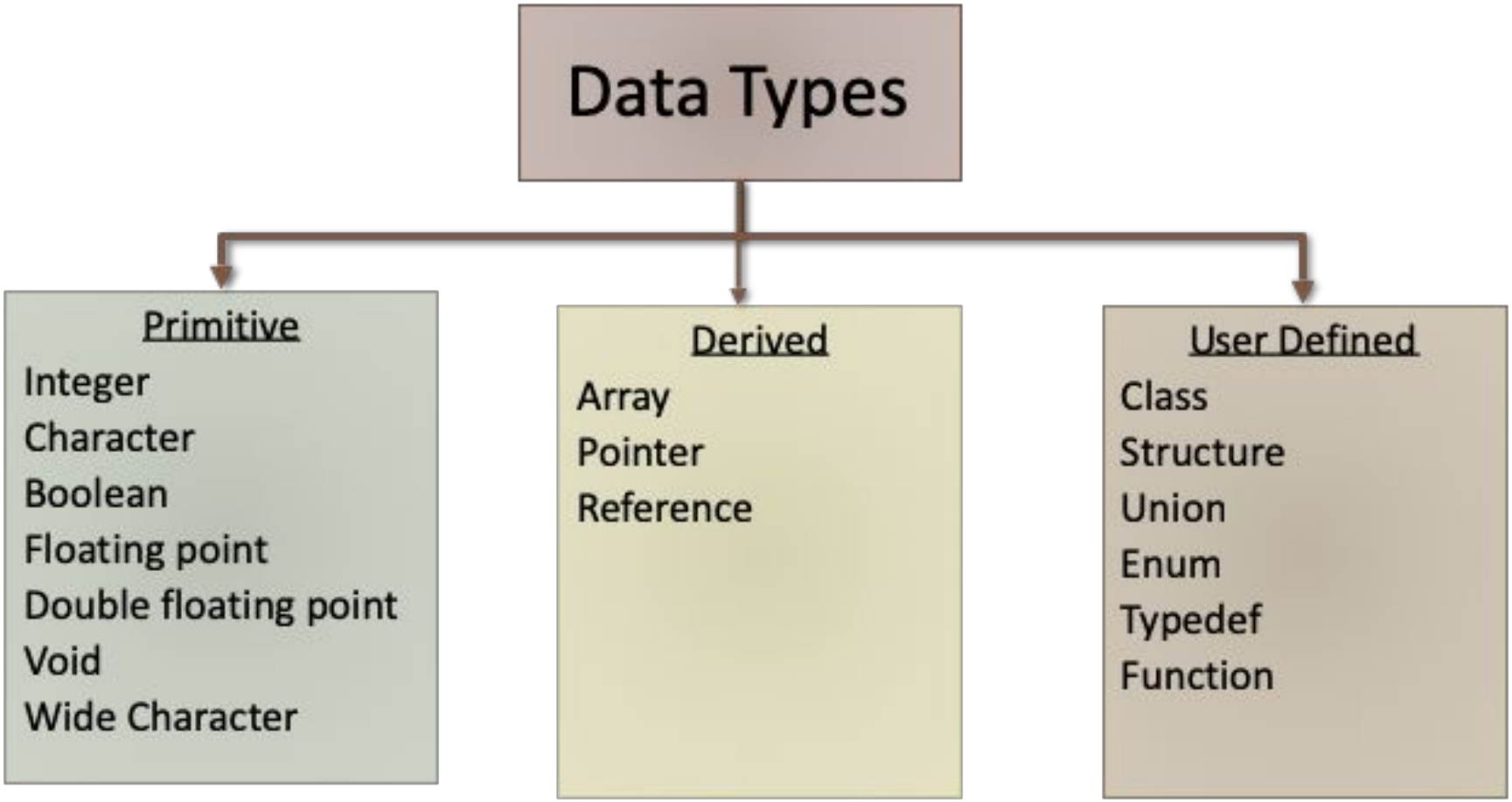
Data Structures and the C++ Standard Template Library

- Name : John Alamina
- Email : john.alamina@hud.ac.uk

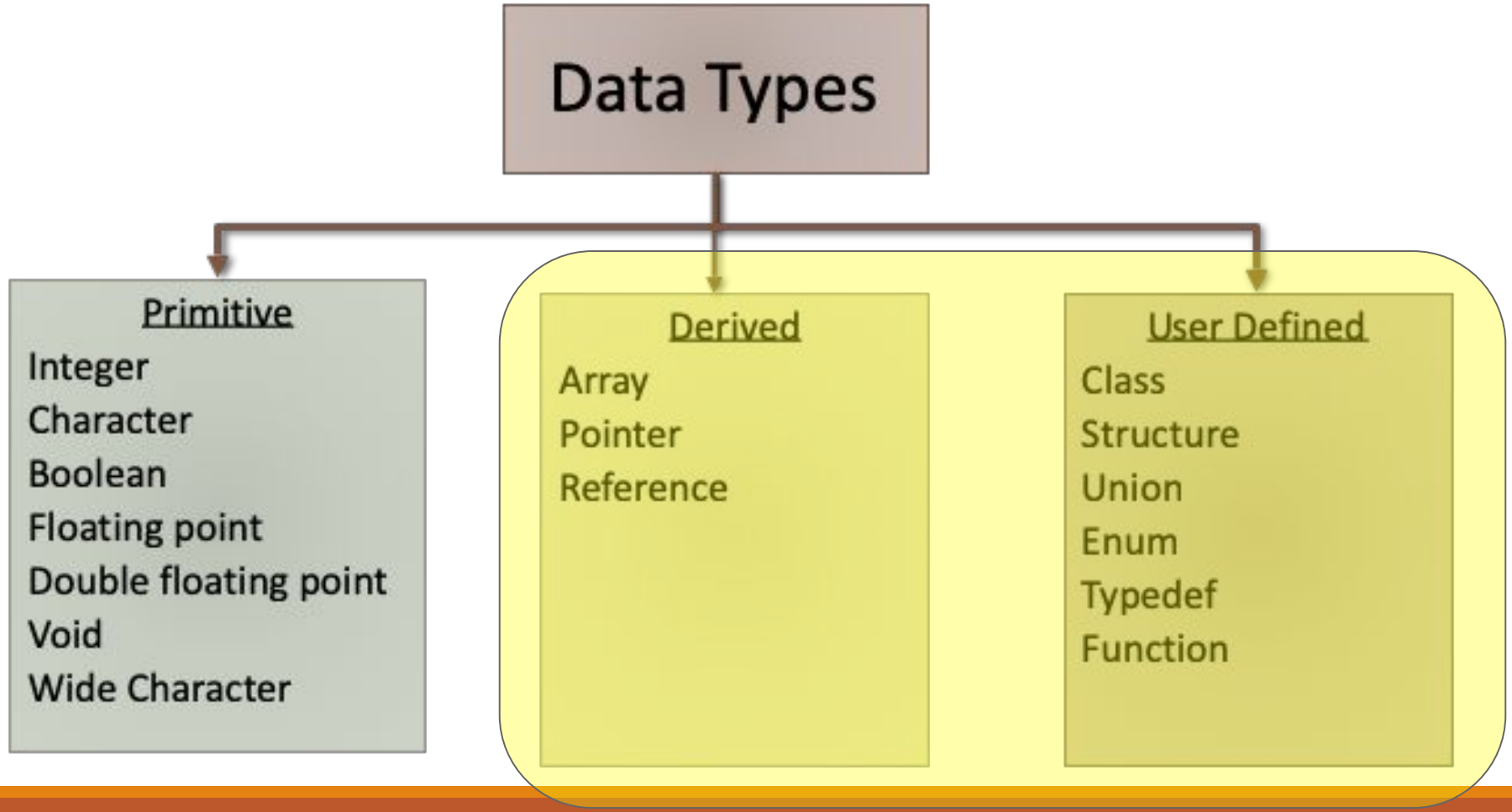
Contents.

- Introduction to Data Structures and Algorithms
- The Standard Template Library
- C/C++ strings
- STL array
- STL vector

Data Structures and algorithms



Data Structures and algorithms



Abstract Data types

We will study these

- **Linked List** – A set of nodes singly or doubly or circularly connected.
- **Records** – A generic term for Structs and classes without operations.
- **Stack**
- **Queues**

We won't study these

- **Heap**
- **Hash-table** – Efficient memory for fast lookup.
- **Map**
- **Trees** – A directed set of nodes connected in a hierarchical structure
- **Graphs** – Group of nodes connected in any particular manner.

STL Containers

- Pair
- Vector
- List
- Dequeue
- Queue
- Priority Queue
- Stack
- Set
- Multiset
- Map
- Multimap
- Heap using STL C++

STL Containers

- Pair
- Vector
- List
- Dequeue
- Queue
- Priority Queue
- Stack
- Set
- Multiset
- Map
- Multimap
- Heap using STL C++

STL Container Groups

Sequence Containers

1. Array
2. List
3. Dequeue
4. Vector

Ordered Containers

1. Map
2. Set
3. Multimap
4. Multiset

Unordered Containers

1. Unordered_map
2. Unordered set
3. Unordered_multimap
4. Unordered multiset

Adaptive Containers

1. Stack
2. Queue
3. Priority queue

STL Container Groups

Sequence Containers

1. Array
2. List
3. Dequeue
4. Vector

Ordered Containers

1. Map
2. Set
3. Multimap
4. Multiset

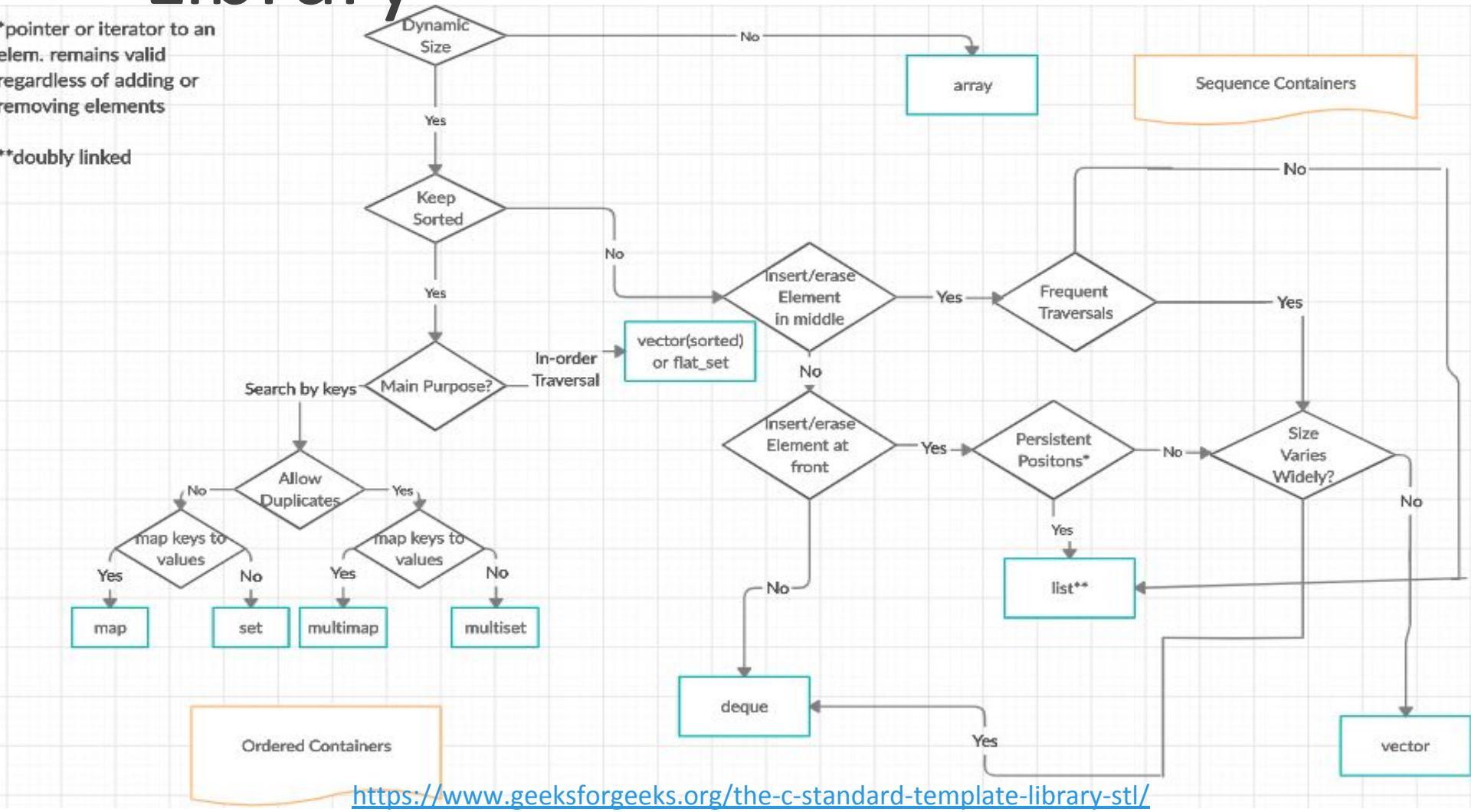
Unordered Containers

1. Unordered_map
2. Unordered set
3. Unordered_multimap
4. Unordered multiset

Adaptive Containers

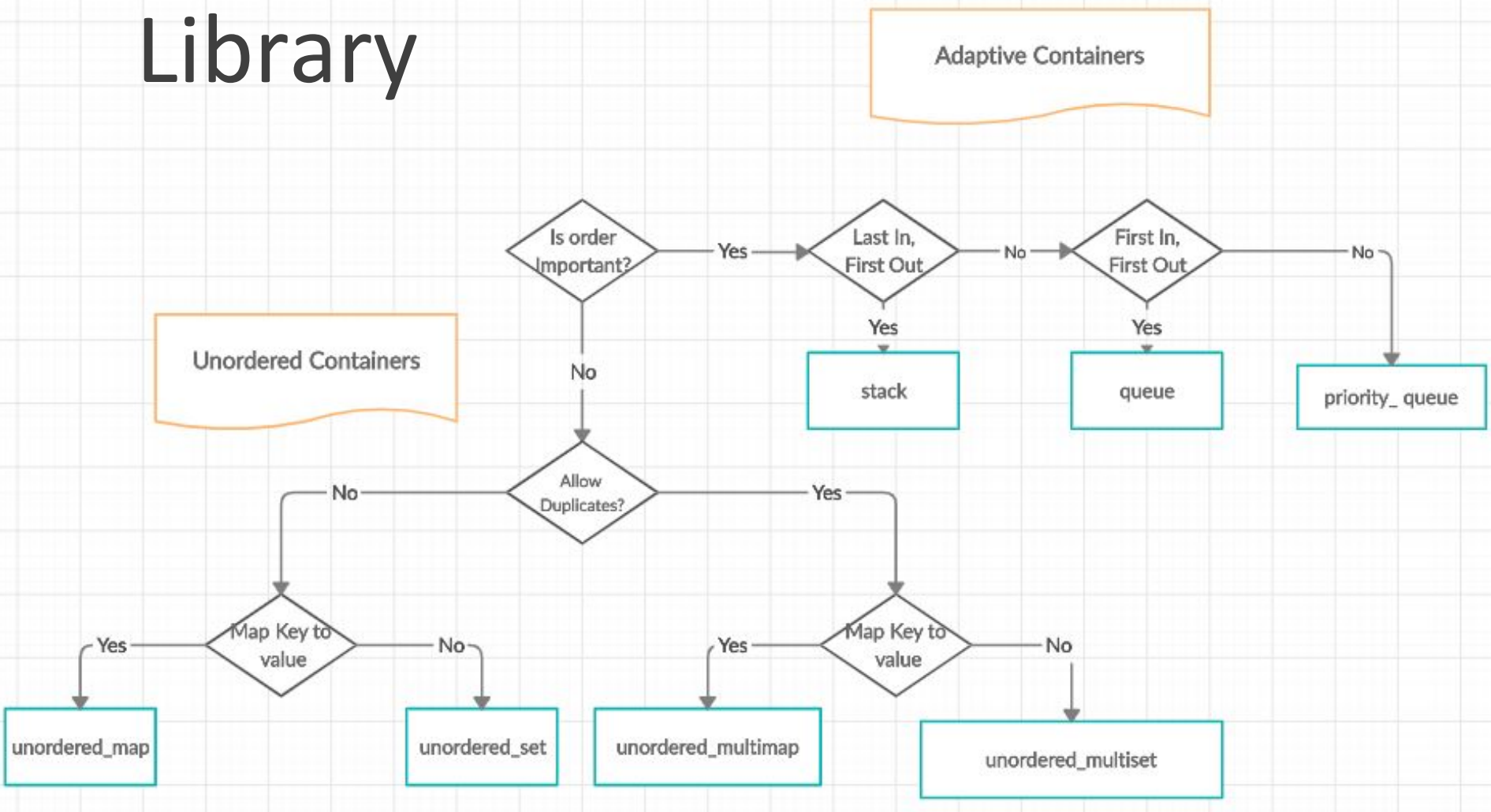
1. Stack
2. Queue
3. Priority queue

The Standard Template Library



https://www.geeksforgeeks.org/the-c-standard-template-library-stl/

The Standard Template Library



https://www.geeksforgeeks.org/the-c-standard-template-library-stl/

C-Style strings

- Declare string e.g `char string[50];`
- String literal e.g. `"This is a static string"`
- String initialisation e.g. `char str[50]="This is a static string";`
- Dynamic memory declaration `char* arr=new char[50];`
- Dynamic memory cleanup `delete [] arr;`

C-Style string functions

1. int strcmp (const char *s1, const char *s2);//compares strings
2. char *strcat (char *dest, const char *src); //string concatenation
3. char *strcpy (char *dest, const char *src);//string copy
4. size_t strlen (const char *s);//string length
5. char *strncpy (char *dest, const char *src, size_t len);//safe copy

C++ Style strings

- Include string.h
- Declare string e.g std::string str ;
- String initialisation e.g. std::string str="This is a static string";
- Dynamic memory declaration std::string* str=new string;
- Dynamic memory cleanup delete str;

C++ Style string functions

1. `int compare( const basic_string& str ) const;`//compares strings
2. char *strcat (char *dest, const char *src); //string concatenation
3. `size type copy( CharT* dest, size_type count, size type pos = 0 ) const;`//string copy
4. `size type length() const;`;//string length
5. `substr( size_type pos = 0, size_type count = npos ) const;`//substring
6. getline(std::cin, name); //get line from console keyboard input
7. `size type find( const basic_string& str, size_type pos = 0 ) const;//find position of substring`

https://en.cppreference.com/w/cpp/string/basic_string
https://en.cppreference.com/w/cpp/string/basic_string/substr
https://en.cppreference.com/w/cpp/string/basic_string/copy
https://en.cppreference.com/w/cpp/string/basic_string/getline
https://en.cppreference.com/w/cpp/string/basic_string/operator%2B

STL Array

- **Declaration**

```
#include <array>
std::array<int, 3> myArray; // declare an integer array with length 3
```

- **Initialisation**

```
std::array<int, 5> myArray = { 9, 7, 5, 3, 1 }; // initializer list
std::array<int, 5> myArray2 { 9, 7, 5, 3, 1 }; // uniform
initialization
std::array<int, > myArray { 9, 7, 5, 3, 1 }; // illegal, array length
must be provided
std::array<int> myArray { 9, 7, 5, 3, 1 }; // illegal, array length
must be provided
std::array myArray { 9, 7, 5, 3, 1 };
myArray.at(1) = 6; // array element 1 valid, sets array element 1 to
value 6
myArray.at(9) = 10; // array element 9 is invalid, will throw an error
```

STL Array

- **Size and sorting**

```
std::array myArray { 9.0, 7.2, 5.4, 3.6, 1.8 };
std::cout << "length: " << myArray.size() << '\n';
```

-

```
#include <algorithm> // for std::sort
#include <array>
#include <iostream>
int main()
{
    std::array myArray { 7, 3, 1, 9, 5 };
    std::sort(myArray.begin(), myArray.end()); // sort the array forwards
//  std::sort(myArray.rbegin(), myArray.rend()); // sort the array backwards
    for (int element : myArray)
        std::cout << element << ' ';
    std::cout << '\n';
    return 0;
}
```

Array ADT

- Properties
  a. List of elements
  b. Array size
- Methods
  a. Sort( )
  b. writeAt(index,value) // i.e. arr[index]=value
  c. readAt(index) // i.e. arr[index]
  d. exists(value)
  e. indexOf(value)

Vector = Dynamically sized array

- Declaration

```
#include <vector>
// no need to specify length at initialization
std::vector<int> array;
std::vector<int> array2 = { 9, 7, 5, 3, 1 }; // use initializer list to initialize array
std::vector<int> array3 { 9, 7, 5, 3, 1 }; // use uniform initialization to initialize array (C++11 onward)
```

- Access

```
array[6] = 2; // no bounds checking
array.at(7) = 3; // does bounds checking
```

- Vectors remember their size - v.size()
- Vectors can be resized - v.resize()
- Vectors can be appended - v.push_back(elem)

https://www.geeksforgeeks.org/vector-in-cpp-stl/

Exercise

On the discussion space, come up with the abstract data type definition for the vector class that includes its properties and methods. Also comment on your classmate's definition.

Any Questions?