ALGORITHM AND PROGRAMMING

Standard Template Library (**STL**) in C++
20250109

OUTLINE

- Introduction to STL (Standard Template Library)
- Using built-in library
  - Stack
  - Queue
  - List
  - Vector
  - Hash map (unordered map)
- Examples

INTRODUCTION

The Standard Template Library (STL) in C++ provides a wide range of built-in data structures and algorithms.

Sequence containers:

array	Array class (class template)
vector	Vector (class template)
deque	Double ended queue (class template)
forward_list	Forward list (class template)
list	List (class template)

Container adaptors:

stack	LIFO stack (class template)
queue	FIFO queue (class template)
priority_queue	Priority queue (class template)

Unordered associative containers:

unordered_set	Unordered Set (class template)
unordered_multiset	Unordered Multiset (class template)
unordered_map	Unordered Map (class template)
unordered_multimap	Unordered Multimap (class template)

Associative containers:

set	Set (class template)
multiset	Multiple-key set (class template)
map	Map (class template)
multimap	Multiple-key map (class template)

STACK

- A **stack** is a **Last-In-First-Out (LIFO)** data structure. The last element inserted is the first to be removed.
- Header **#include <stack>**

Function	Description
push(data)	Adds an element to the top of the stack
pop( )	Removes the top element.
top( )	Returns the top element without removing it.
empty( )	Checks if the stack is empty
size( )	Returns the number of elements.

Example: Built-in **stack**

```
1   #include <iostream>
2   #include <stack>
3   using namespace std;
4
5    main() {
6       stack<int> s;
7       s.push(10);
8       s.push(20);
9       s.push(30);
10
11      cout<<"Size of the stack: " <<s.size()<<endl;
12      s.pop();
13      s.pop();
14      s.push(1);
15
16      cout<< "Size of the stack: " <<s.size()<<endl;
17      cout << "Top of stack: " << s.top()<<endl;
18      cout << "Is empty?: " << s.empty()<<endl;
19   }
```

What is the output of this program?

QUEUE

- **A queue** is a **First-In-First-Out (FIFO)** data structure. The first element inserted is the first to be removed.
- Header **#include <queue>**

Function	Description
push(data)	Add an element to the back of the queue
pop( )	Remove the front element.
front( )	Return the front element
back( )	Return the last element
empty( )	Check if the queue is empty
size( )	Return the number of elements.

Example: Built-in **queue**

```
1   #include <iostream>
2   #include <queue>
3   using namespace std;
4
5   main() {
6       queue<int> q;
7       q.push(1);
8       q.push(2);
9       q.push(3);
10
11
12      cout << "Front of queue: " << q.front()<< endl;
13      q.pop();
14      cout << "After pop, front: " << q.front()<< endl;
15      cout<<"Back of the queue: " << q.back()<<endl;
16      cout << "Size of queue: " << q.size()<< endl;
17      q.pop();
18      q.pop();
19      cout << "Size of queue: " << q.size()<< endl;
20      cout<<"Is queue empty? " << q.empty()<<endl;
21  }
```

What is the output of this program?

LIST

- **A list** is a doubly-linked list, suitable for frequent insertions and deletions.
- Header **#include <list>**

Function	Description
push_back(data)	Add an element to the back of the list
push_front( )	Add an element to the front of the list
pop_back( )	Remove element from the back
pop_front( )	Remove element from the front
size( )	Return the number of elements in the list
begin()	Iterator for traversing the list
end()	Iterator for traversing the list

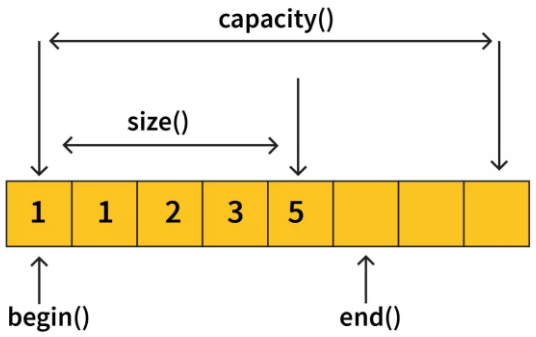
Example: Built-in list

```
#include <iostream>
#include <list>
using namespace std;

void displayList(list<int> mylist){
    for (int x : mylist) {
        cout << x << " ";
    }
    cout << endl;
}
void displayListV2(list<int> mylist){
    cout << "\Display elements using iterators: ";
    for (auto data = mylist.begin(); data != mylist.end(); data++) {
        cout << *data << " ";
    }
    cout << endl;
}

main() {
    list<int> mylist = {10, 90, 50};

    displayList(mylist);
    mylist.push_back(4);
    displayListV2(mylist);
    displayList(mylist);

    cout<<"\nFirst element: "<<*mylist.begin()<<endl;
    cout<<"Last element: "<<*mylist.end()<<endl;
    cout<<"Is list empty?: "<<mylist.empty()<<endl;
}
```

What is the output of this program?

VECTOR

- **A vector** is a dynamic array that can grow or shrink in size.
- Header **#include <vector>**



Function	Description
push_back(data)	Add an element to the end of the vector
pop_back( )	Remove the last element
size( )	Return the number of elements
begin( )	Iterator for traversing vector
end( )	Iterator for traversing vector
capacity( )	Current max size
Operator [ ]	Use bracket to access elements by index number

Example: Built-in **vector**

```cpp
#include <iostream>
#include <vector>
using namespace std;

void display(vector<string> v){
    for (int i = 0; i < v.size(); ++i) {
        cout << v[i] << " ";
    }
    cout<<endl;
}

main() {
    vector<string> v = {"Dara", "Sok", "Pisey"};

    display(v);
    cout<<"Size: "<<v.size()<<"\n\n";

    v.push_back("Panha");
    v.push_back("Sokha");
    display(v);
    cout<<"Size: "<<v.size()<<"\n\n";

    v.pop_back();
    display(v);
    cout<<"Size: "<<v.size()<<"\n\n";
}
```

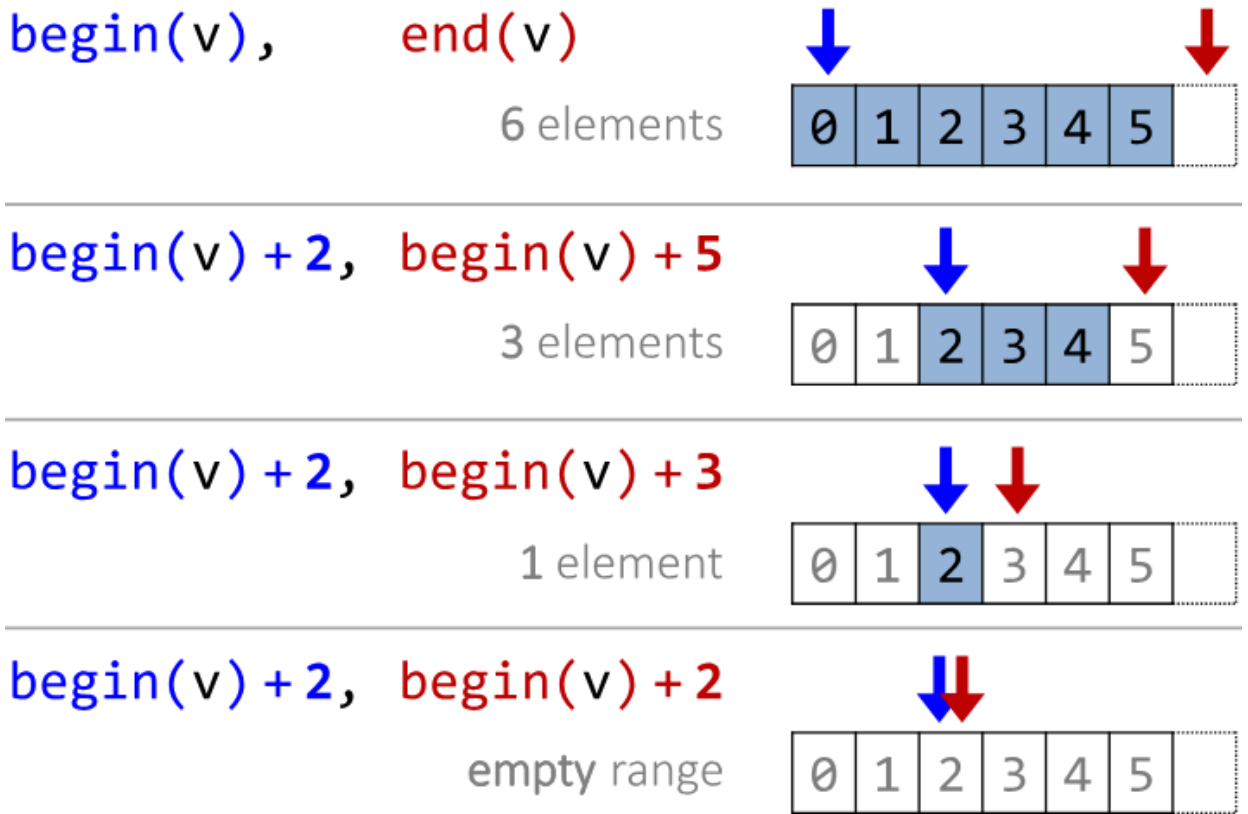
What is the output of this program?

```cpp
#include <iostream>
#include <vector>
using namespace std;

void display(vector<string> v){
    for (int i = 0; i < v.size(); ++i) {
        cout << v[i] << " ";
    }
    cout<<endl;
}

void displayV2(vector<string> v){
    for(string data : v){
        cout<<data<<" ";
    }
    cout<<endl;
}

main() {
    vector<string> v = {"Dara", "Sok", "Pisey"};

    display(v);
    cout<<"Size: "<<v.size()<<"\n\n";

    v.push_back("Panha");
    v.push_back("Sokha");
    display(v);
    cout<<"Size: "<<v.size()<<"\n\n";

    v.pop_back();
    displayV2(v);
    cout<<"Size: "<<v.size()<<"\n\n";
}
```

What is the output of this program?

ANOTHER EXAMPLE OF VECTOR

```
std::vector<int> v {0, 1, 2, 3, 4, 5};
```



VECTOR OF VECTOR

How to create a 2D vector?

vector<**vector**<int>> data;

```
1   #include<iostream>
2   #include<vector>
3   using namespace std;
4   main(){
5       vector<vector<int>> matrix;
6
7       matrix.push_back({10, 20, 30});  // Adds a row
8       matrix.push_back({90, 80, 70});  // Adds another row
9       matrix.push_back({2, 4, 6});  // Adds another row
10
11      for(int row=0; row < matrix.size(); row++){
12          for(int column=0; column < matrix.size(); column++){
13              cout<<matrix[row][column] <<"\t";
14          }
15          cout<<endl;
16      }
17  }
```

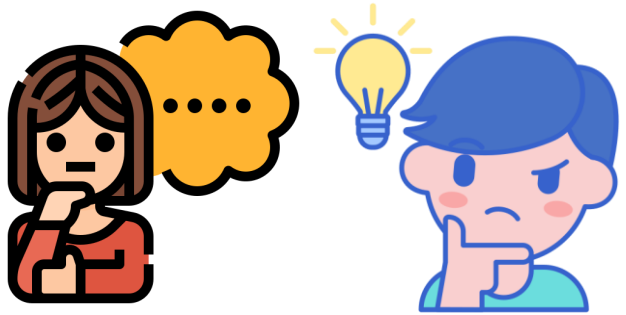
When row and column are the same size

What is the output of this program?

```
#include<iostream>
#include<vector>
using namespace std;
main(){
    vector<vector<int>> matrix;

    matrix.push_back({10, 20, 30});  // Adds a row
    matrix.push_back({90, 80, 70, 60});  // Adds another row
    matrix.push_back({2, 4, 6, 8, 10});  // Adds another row
    matrix.push_back({0, 5});  // Adds another row

    for(int row=0; row < matrix.size(); row++){
        for(int column=0; column < matrix[row].size(); column++){
            cout<<matrix[row][column] <<"\t";
        }
        cout<<endl;
    }
}
```

When each row has different number of columns

What is the output of this program?



Resizing a vector

```cpp
#include<iostream>
#include<vector>
using namespace std;

void display(vector<vector<int>> matrix){
    for(int row=0; row < matrix.size(); row++){
        for(int column=0; column < matrix[row].size(); column++){
            cout<<matrix[row][column] <<"\t";
        }
        cout<<endl;
    }
}

main(){
    vector<vector<int>> matrix;

    matrix.push_back({10, 20, 30});  // Adds a row
    matrix.push_back({90, 80, 70, 60});  // Adds another row
    matrix.push_back({2, 4, 6, 8, 10});  // Adds another row
    matrix.push_back({0, 5});  // Adds another row
    display(matrix);

    cout<<endl<<endl;
    matrix.push_back({100, 200, 300});
    matrix[0].resize(6);
    matrix[0][2] = 100;
    matrix[0][3] = 500;
    display(matrix);
}
```

What is the output of this program?

VECTOR OF LIST

vector<list<int>> vlist;

```
vlist.push_back({1, 2, 3});   // Adds a list {1, 2, 3}
vlist.push_back({4, 5});      // Adds another list {4, 5}
```

```
vlist[0].push_back(42);       // Adds 42 to the end of the first list
vlist[1].remove(5);           // Removes all occurrences of 5 from the second list
```

```cpp
#include <iostream>
#include <vector>
#include <list>
using namespace std;

main() {
    vector<list<int>> vlist;

    vlist.push_back({1, 2, 3});
    vlist.push_back({4, 5, 6});

        // Add an element to the first list
    vlist[0].push_back(42);

        // Display
    for (int i = 0; i < vlist.size(); i++) {
        cout << "List " << i << ": ";
        for (int data : vlist[i]) {
            cout << data << " ";
        }
        cout << "\n";
    }
}
```

D:\Algo2024-25\STL\vectorSTL-2D-VecOfList.exe

```
List 0: 1 2 3 42
List 1: 4 5 6
```

HASH MAP (UNORDERED MAP)

- **An unordered_map** is a hash table that provides fast access to key-value pairs.
- Header **#include <unordered_map>**

Function	Description
insert( )	Adds a key-value pair
erase( )	Removes a key-value pair
find( )	Checks if a key exists
size( )	Return the number of elements
Operator [ ]	Inserts or accesses elements by key.

Example: Built-in **hash map**

What is the output of this program?

```
#include <iostream>
#include <unordered_map>
using namespace std;

void displayHashMap(unordered_map<string, string> umap){
    cout<<"Term\tDefinition"<<endl;
    cout<<"________________________"<<endl;
    for (auto data : umap) {
        cout << data.first << " \t" << data.second << endl;
    }
    cout<<endl;
}
main() {
    unordered_map<string, string> umap;
    umap["book"] = "study materials";
    umap["car"] = "vehicle with 4 wheels";
    umap["apple"] = "fruit. we can eat";

    displayHashMap(umap);

    string keys[] = {"book", "car", "apple"};
    for(string key : keys){
        cout<< umap[key]<<"\n";
    }

    umap.erase("car");
    cout<<"\nSize of hashmap now: "<<umap.size();
}
```

Q&A