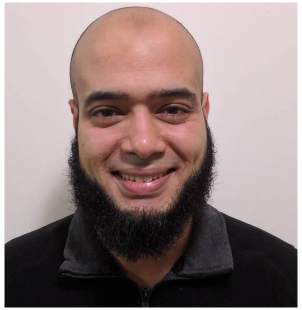
C++ Programming
Class Templates

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

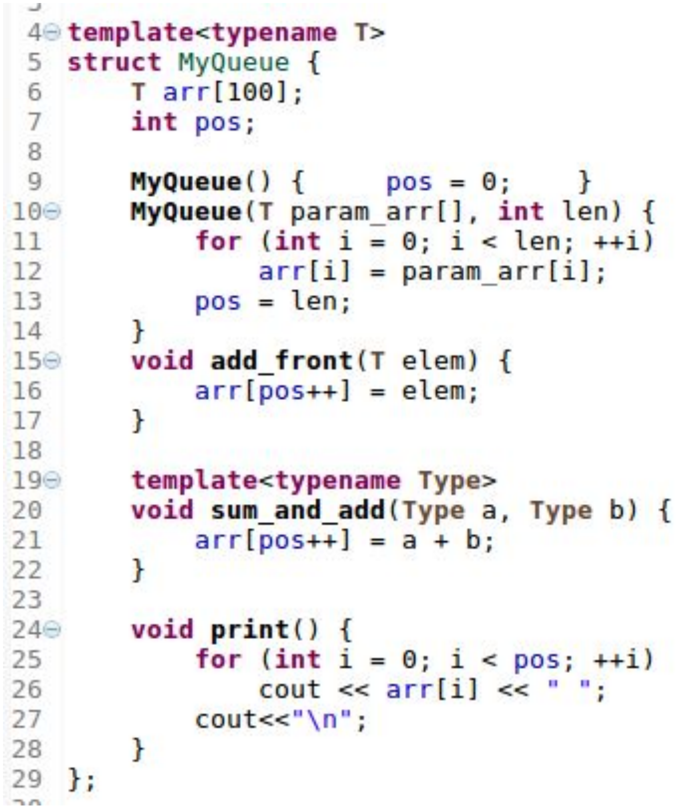
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



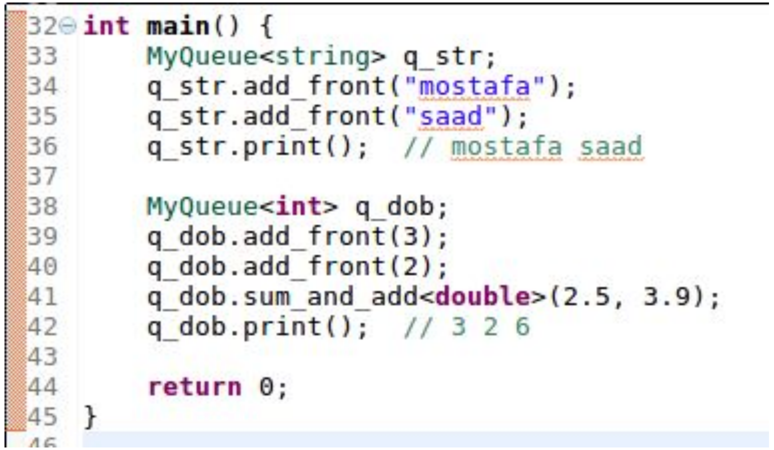
Class Templates

- Similar to functions, we can have struct (aka classes) to be independent of type
- Recall our Hospital Queue
  - What if I need queue of int and another of string?
  - Same struct code copy-paste!
- Class templates
  - Compiler generates several versions of the class template based on used types

Class Templates



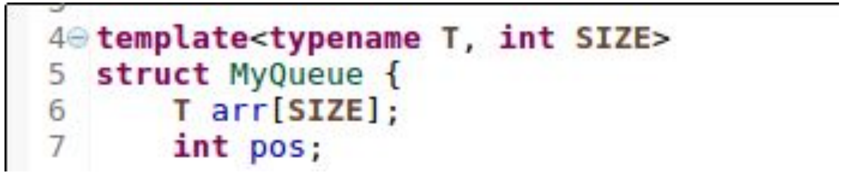
```
template<typename T>
struct MyQueue {
    T arr[100];
    int pos;

    MyQueue() {     pos = 0;    }
    MyQueue(T param_arr[], int len) {
        for (int i = 0; i < len; ++i)
            arr[i] = param_arr[i];
        pos = len;
    }
    void add_front(T elem) {
        arr[pos++] = elem;
    }

    template<typename Type>
    void sum_and_add(Type a, Type b) {
        arr[pos++] = a + b;
    }

    void print() {
        for (int i = 0; i < pos; ++i)
            cout << arr[i] << " ";
        cout<<"\n";
    }
};
```



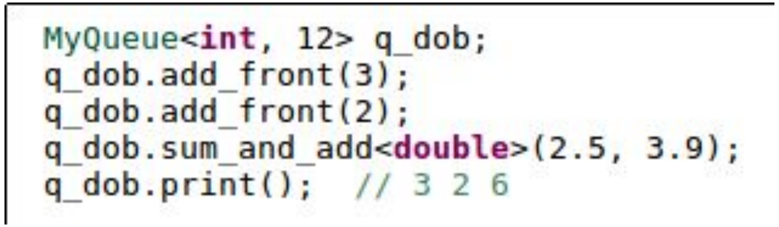
```
int main() {
    MyQueue<string> q_str;
    q_str.add_front("mostafa");
    q_str.add_front("saad");
    q_str.print();  // mostafa saad

    MyQueue<int> q_dob;
    q_dob.add_front(3);
    q_dob.add_front(2);
    q_dob.sum_and_add<double>(2.5, 3.9);
    q_dob.print();  // 3 2 6

    return 0;
}
```

Non-type **parameters** for templates

- The array size was fixed. Can we pass the array size?
  - Yes. Compiler is generating in compile time!
  - But it MUST be const value (e.g. you don't read from a user)
- Let's pass the SIZE parameter
  - Even can put a default value!
- Typical usage: Constants and arrays sizes

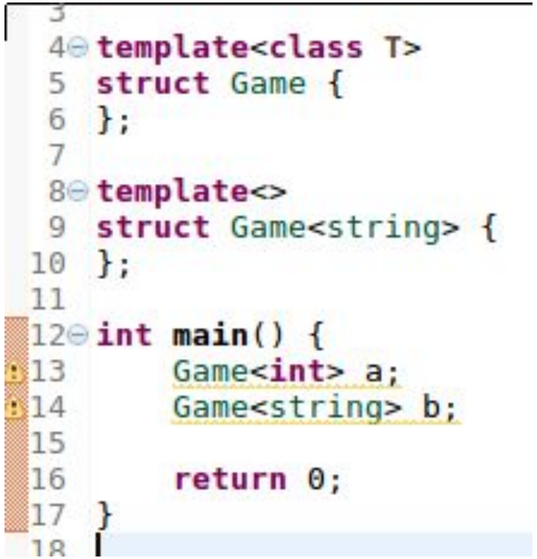


```
template<typename T, int SIZE>
struct MyQueue {
    T arr[SIZE];
    int pos;
```



```
MyQueue<int, 12> q_dob;
q_dob.add_front(3);
q_dob.add_front(2);
q_dob.sum_and_add<double>(2.5, 3.9);
q_dob.print();  // 3 2 6
```

Class Template specialization



```
template<class T>
struct Game {
};

template<>
struct Game<string> {
};

int main() {
    Game<int> a;
    Game<string> b;

    return 0;
}
```

Overloading Vs Template

- Templates: **identical syntax** for different data types
- Function overloading is **identical function** name + different parameters + different function behaviour

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."