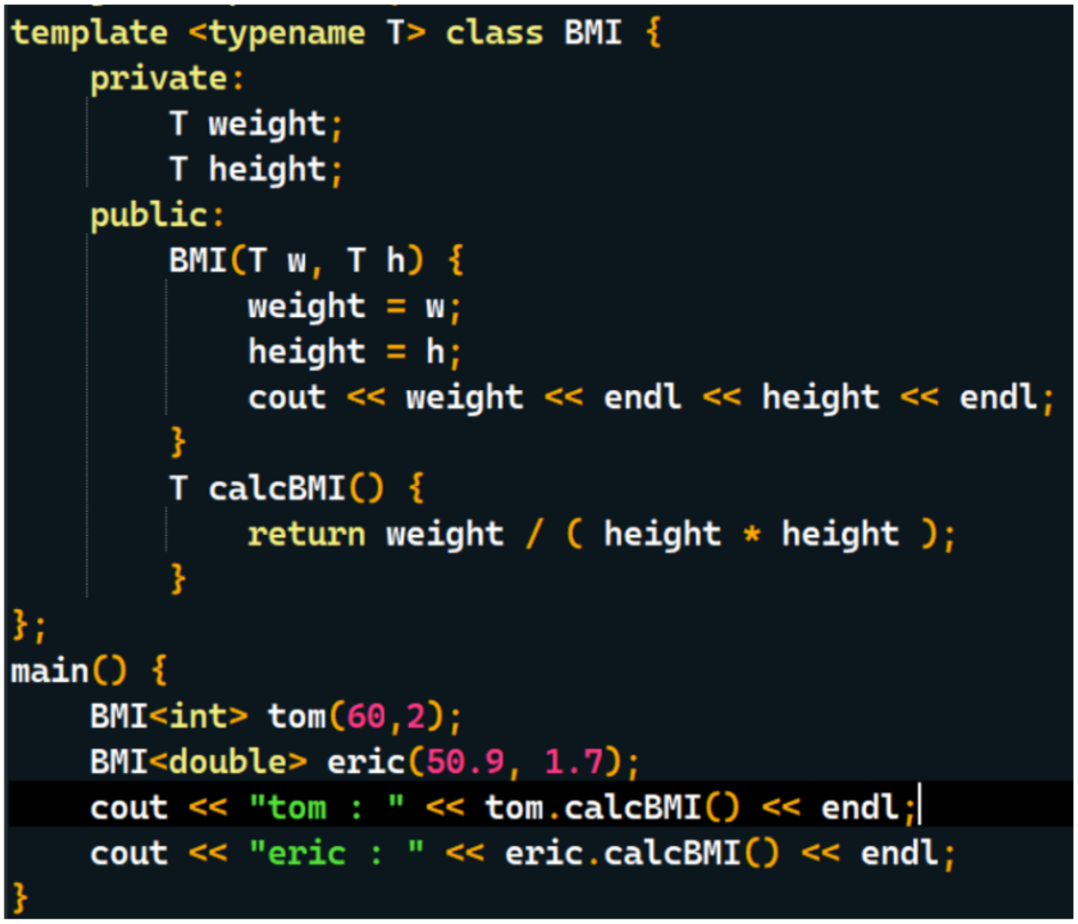
Summary

Sessions (30-12-2022)

- **Class Template :** A template is a simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. It allows you to define the generic classes and generic functions.

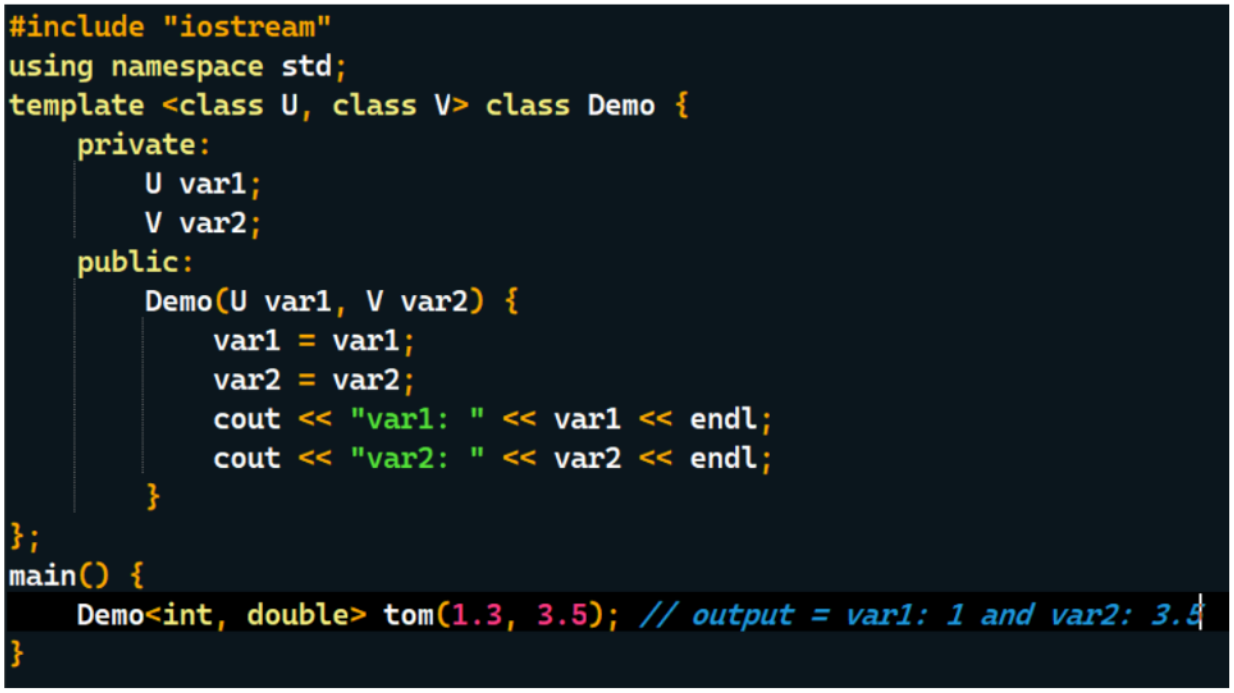
  Class Templates like function templates, class templates are useful when a class defines something that is independent of the data type.



```cpp
template <typename T> class BMI {
    private:
        T weight;
        T height;
    public:
        BMI(T w, T h) {
            weight = w;
            height = h;
            cout << weight << endl << height << endl;
        }
        T calcBMI() {
            return weight / ( height * height );
        }
};
main() {
    BMI<int> tom(60,2);
    BMI<double> eric(50.9, 1.7);
    cout << "tom : " << tom.calcBMI() << endl;
    cout << "eric : " << eric.calcBMI() << endl;
}
```

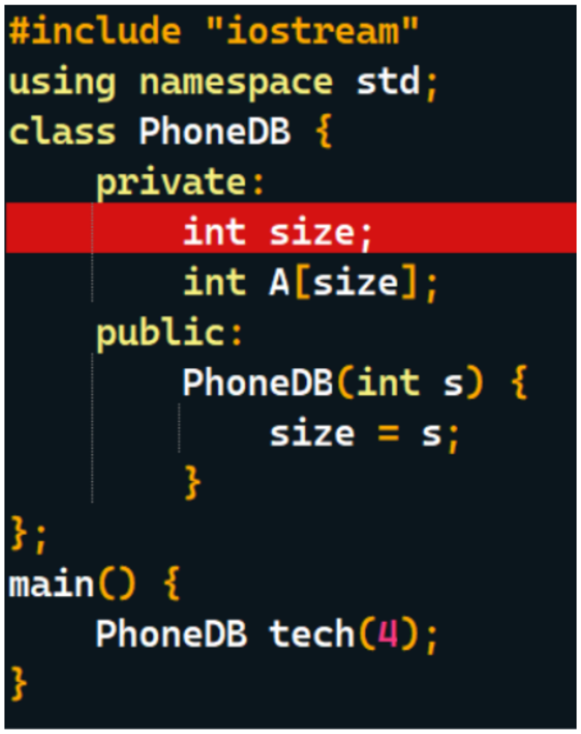
- In above example, as we can see we create a typename as a "T" for template and provide that "T" to as datatype. So when we create an object that time we can decide what data type we want to give in class blueprint means wherever we have “T” that will be replaced by the provided data type.

- **More than one argument to templates** :



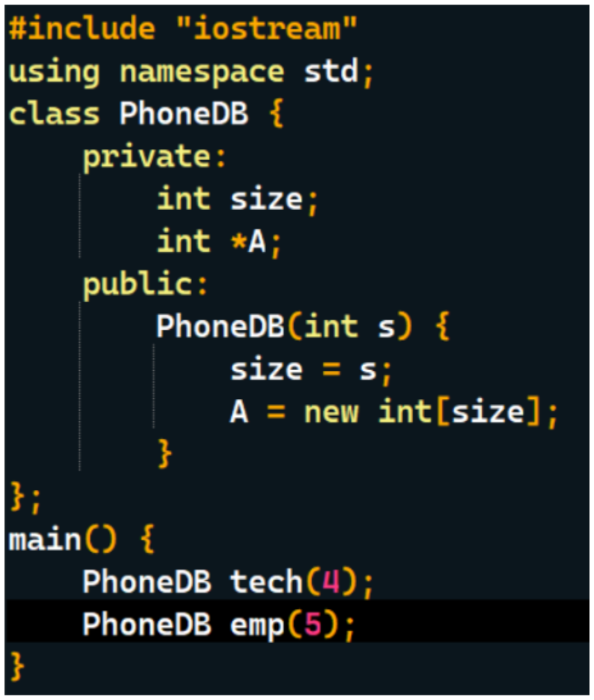
```
#include "iostream"
using namespace std;
template <class U, class V> class Demo {
    private:
        U var1;
        V var2;
    public:
        Demo(U var1, V var2) {
            var1 = var1;
            var2 = var2;
            cout << "var1: " << var1 << endl;
            cout << "var2: " << var2 << endl;
        }
};
main() {
    Demo<int, double> tom(1.3, 3.5); // output = var1: 1 and var2: 3.5
}
```

- In the below code we want an array therefore we define the array in class and size of array we will take at run time means at the time of object creation. But what we used in class that “size” variable takes space from stack and stack memory wants size at compile time only. Therefore the below code shows error.



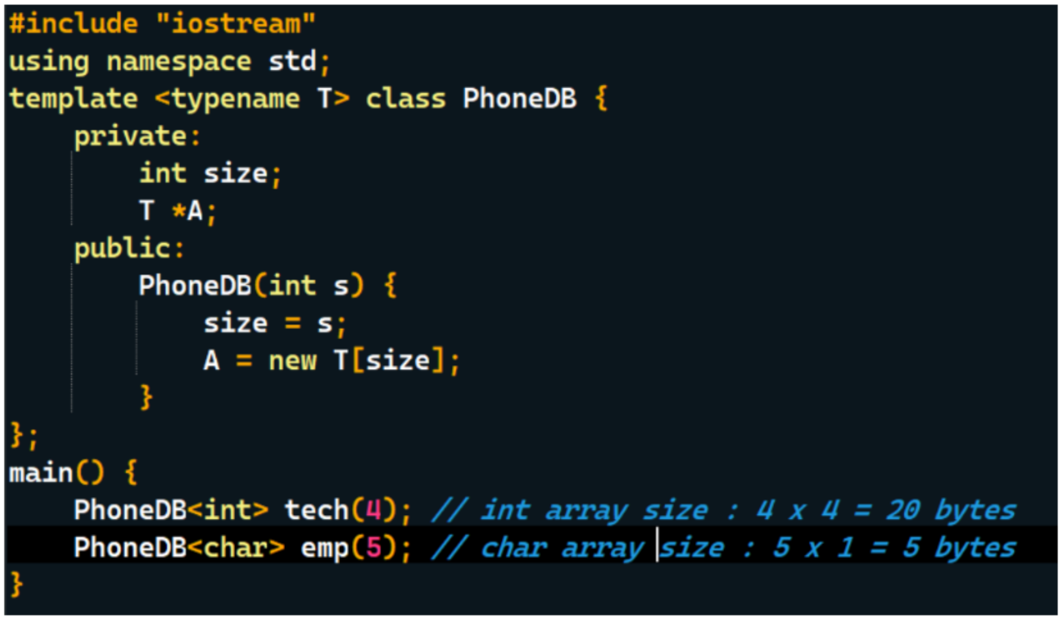
```
#include "iostream"
using namespace std;
class PhoneDB {
    private:
        int size;
        int A[size];
    public:
        PhoneDB(int s) {
            size = s;
        }
};
main() {
    PhoneDB tech(4);
}
```

- For solving the above problem we will take space from heap memory and create a dynamic array.



```
#include "iostream"
using namespace std;
class PhoneDB {
    private:
        int size;
        int *A;
    public:
        PhoneDB(int s) {
            size = s;
            A = new int[size];
        }
};
main() {
    PhoneDB tech(4);
    PhoneDB emp(5);
}
```

- If we want generic array means at the time of object creation if we want to provide what type of data we have to provide into an array then we can use template.



```
#include "iostream"
using namespace std;
template <typename T> class PhoneDB {
    private:
        int size;
        T *A;
    public:
        PhoneDB(int s) {
            size = s;
            A = new T[size];
        }
};
main() {
    PhoneDB<int> tech(4); // int array size : 4 x 4 = 20 bytes
    PhoneDB<char> emp(5); // char array size : 5 x 1 = 5 bytes
}
```