

UEE 1303(1068): Object-Oriented Programming

Lab #12: Template

In this laboratory session you will learn:

- how to use function template and class template

Lab 12-1: Function Template

- ✓ A function template defines a function that takes type parameters. Please execute lab12-1. Here is an example to maintain memory allocation for different types.

```
#include <iostream>
#include <cassert>
using namespace std;
template <class T>
T *new1D(int n, T k)
{
    T *vec = new T [n];
    for (int i = 0; i < n; i++)
        vec[i] = k;
    return vec;
}
template <class T>
void delete1D(T *vec)
{
    assert(vec != NULL);
    delete [] vec;
}
template <class T>
void display1D(T *vec, int n)
{
    for (int i = 0; i < n; i++)
        cout << vec[i] << " ";
    cout << endl;
}
int main()
```

```
{  
    int *ivec = new1D<int>(10,1);  
    display1D<int>(ivec,10);  
    delete1D<int>(ivec);  
    double *dvec = new1D<double>(10,3.2);  
    display1D<double>(dvec,10);  
    delete1D<double>(dvec);  
    return 0;  
}
```

Lab 12-2: Function Template: Specialization

- ✓ In program lab12-1, you can maintain a specific version of `display1D()` for `double`. Please add this specialization of `display1D<T>` to lab12-1 and execute the program again.

```
template <>  
void display1D(double *vec, int n)  
{  
    cout << fixed << setprecision(2);  
    for (int i = 0; i < n; i++)  
        cout << vec[i] << " ";  
    cout << endl;  
}
```

Lab 12-3: Class Template

- ✓ You can also define a class template by adding prefix `template<class T>`.

```
// lab12-3.cpp  
#include <iostream>  
#include <iomanip>  
using namespace std;  
template <class T>  
class Point2D  
{  
    private:  
        T x;  
        T y;  
    public:  
        Point2D(): x(T(0)), y(T(0)){}  
}
```

```
    Point2D(T a, T b): x(a), y(b){}
    void display() const;
};
template <class T>
void Point2D<T>::display() const
{
    cout << x << " " << y << endl;
}
int main()
{
    Point2D<int> p1;
    p1.display();
    Point2D<double> p2(1.9, 3.4);
    p2.display();
    return 0;
}
```

Lab 12-4: Class Template: Specialization

- ✓ Here define a specialization of the template class Point2D<T> when its elements are complex number.

```
// lab12-4.cpp
#include <iostream>
using namespace std;
class Complex
{
private:
    double real;
    double image;
public:
    Complex(const double a, const double b)
        : real(a), image(b){}
    Complex(const Complex &c)
        : real(c.real), image(c.image){}
    void display() const
    {
        cout << real << " " << image << endl;
    }
}
```

```
};  
// template Point2D defined in lab12-3  
  
template <>  
class Point2D<Complex>  
{  
    private:  
        Complex x;  
        Complex y;  
    public:  
        Point2D(const Complex &a, const Complex &b)  
            :x(a),y(b){}  
        void display() const;  
};  
void Point2D<Complex>::display() const  
{  
    x.display();  
    y.display();  
}  
int main()  
{  
    Complex c1(1.9,3.4);  
    Complex c2(2.0,1.3);  
    Point2D<Complex> pc(c1,c2);  
    pc.display();  
    return 0;  
}
```

- If we do not define a specialization of the template class Point2D<T> when its elements are complex number, there is a compiler error in this example. Please fix the compiler error to produce the same output.

Exercise 12-1 (Statistical Analysis)

- ✓ Please finish the undefined function template in ex12-1.

```
class Point2D  
{  
    private:  
        int x;
```

```
    int y;
public:
    // add any member if necessary
};

template<class T>
void analysis(int n, int k = 0)
{
    T *vec = new1D<T> (n, k);
    rand1D<T>(vec,n);
    // for int 1~10, for double 0.00~10.00, for char a~z,
    // for Point2D x: 0~9 y:0~9
    display1D<T>(vec,n);
    sort1D<T>(vec,n);
    display1D<T>(vec,n);
}

int main()
{
    int n;
    cout << "Enter n: ";
    cin >> n;
    srand(1);
    analysis<int>(n);
    analysis<double>(n);
    analysis<char>(n);
    analysis<Point2D>(n);
    return 0;
}
```

- ✓ The output of the program should like as,

```
Enter n: 8
2 8 5 1 10 5 9 9
1 2 5 5 8 9 9 10
9.62 4.64 7.05 1.45 2.81 8.27 9.61 4.91
1.45 2.81 4.64 4.91 7.05 8.27 9.61 9.62
f i r c v s c x
c c f i r s v x
(2, 2) (1, 6) (8, 5) (7, 6) (1, 8) (9, 2) (7, 9) (5, 4)
```

(1, 6) (1, 8) (2, 2) (5, 4) (7, 6) (7, 9) (8, 5) (9, 2)

- Hint: use x coordinate first to compare Point2D and then compare y coordinate if x coordinates for two Point2Ds are the same.

Exercise 12-2 (Vector)

- ✓ Please finish the undefined function template in ex12-2. The main function is like as follows.

```
int main()
{
    int n;
    cout << "Enter n: ";
    cin >> n;
    Vector<double> dvec(n,1);
    double *b = new double[n];
    for (int i = 0; i < n; i++)
        b[i] = i;
    Vector<double> dvec2(n,b);
    cout << "dvec = ";
    dvec.display();
    cout << "dvec2 = ";
    dvec2.display();
    dvec2 += dvec;
    cout << "new dvec = ";
    dvec2.display();
    double c = dot(dvec, dvec2);
    cout << "dot(dvec, dvec2) = " << c << endl << endl;

    srand(1);
    Point2D *v = new Point2D[n];
    rand1D<Point2D>(v, n); //0~9
    Vector<Point2D> vp1(n,1);
    Vector<Point2D> vp2(n,v);
    cout << "vp1 = ";
    vp1.display();
    cout << "vp2 = ";
    vp2.display();
    vp2 += vp1;
```

```
    cout << "new vp2 = ";  
    vp2.display();  
    Point2D d = dot(vp1, vp2);  
    cout << "dot(vp1, vp2) = " << d << endl;  
    return 0;  
}
```

- ✓ The results are,

```
Enter n: 3  
dvec = 1 1 1  
dvec2 = 0 1 2  
new dec2 = 1 2 3  
dot(dvec, dvec2) = 6  
vp1 = (1,1) (1,1) (1,1)  
vp2 = (1,7) (4,0) (9,4)  
new vp2 = (2,8) (5,1) (10,5)  
dot(vp1, vp2) = (17,14)
```

- Hint: dot operation for two vector is defined as $\sum_{i=0}^{n-1} v_1(i) \times v_2(i)$, and multiplication for two Point2Ds is written as `Point2D(p1.x*p2.x, p1.y*p2.y);`
- ✓ Please declare class Point2D in Point2D.h and define its functionality in Point2D.cpp.
- ✓ Please declare template class Vector in Vector.h and define its functionality in Vector.cpp.

```
template <class T>  
class Vector  
{  
    private:  
        int len;  
        T* vec;  
    public:  
        // add any member if necessary  
        template<class S>  
        friend S dot (const Vector<S> &, const Vector<S> &);  
};
```