LAB 10

Template

2024/5/13

The usefulness of function template(1/2)

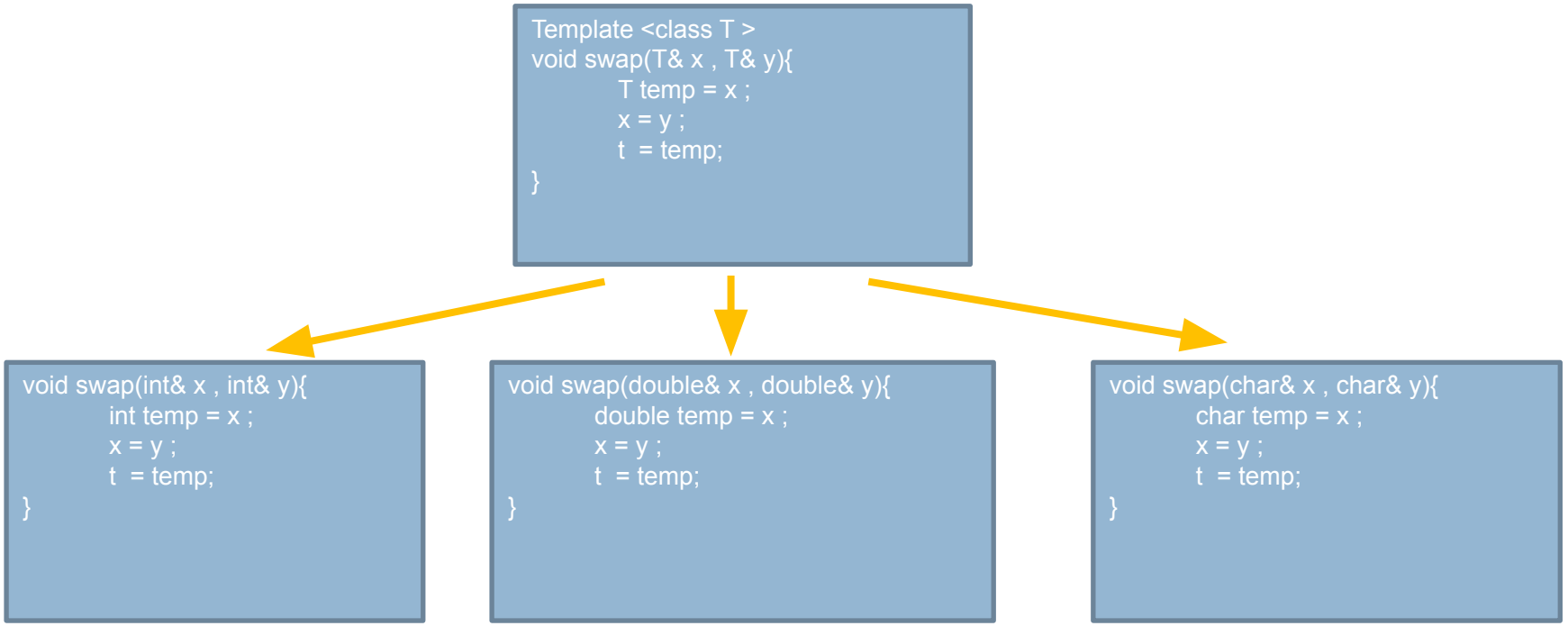
- C++ allows multiple overloading functions – but need define individually

```
void swap ( int& x ,int& y){          int temp=x;        x=y;  y=temp; }
void swap ( double& x, double& y ){    double temp=x ;x=y,   y=temp; }
void swap (char& x, char& y){     char temp=x;     x=y;  y=temp; }
```

- We want one function to do the things above

```
void swap ( vartype& x ,vartype& y){
        vartype temp=x;
        x=y;
        y=temp;
}
```

The usefulness of function template(2/2)



Creating function template (1/2)

```
template < class T >
void swap( T& x , T& y){
        T temp = x ;
        x = y ;
        t  = temp;
}
```

- The key class T does not necessarily mean that T stands for a programmer-created class . it can be int, char, ….
- Many newer compilers allow you to replace class with typename in the template definition

Creating function template (2/2)

- Function template: functions that use variable types
  - Outline a group of functions that only differ in datatypes of parameters used
- In a function template , at least one argument is generic
- A function template will generate one or more template functions
  - When calling a function template, compiler generates code for different functions as it needs

Overloading function template

- Can overload function templates only when each version takes a

different arguments list allow to distinguish

```
template < class T>
T findMax( T x, T y ){
T max ;
if(y > x)
        max = y ;
else
        max = x ;
return max;
}
```

```
template < class T>
T findMax( T x, T y, T z ){
T max = x ;
if(y > max)
        max = y ;
if(z > max )
        max = z ;
return max;
}
```

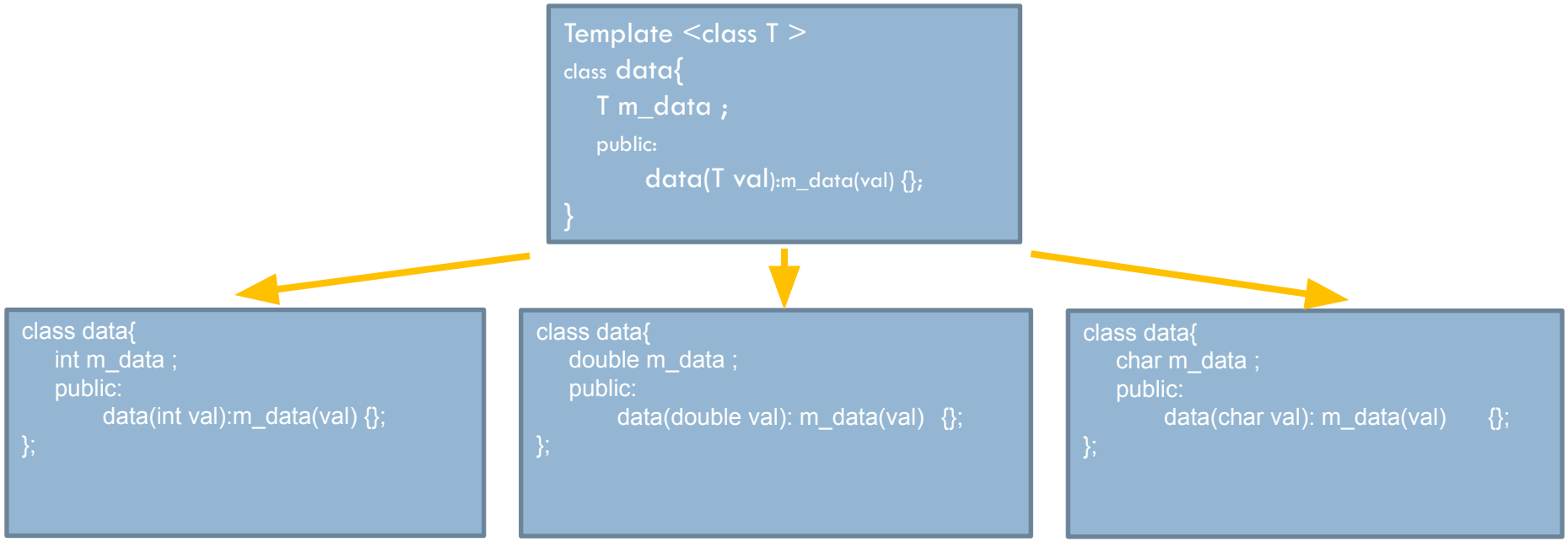
More than one type

```
template < class T, class U >
void compare( T v1, U v2){
        if(v1 > v2 )
                cout<< v1 << " >" << v2  << endl;
        else
                cout<< v1 <<" <= " << v2  << endl;
        ....
}
```

- You should be aware of comparison of different type, the datatype created by programmer should have its overloading function

The usefulness of class template(1/2)

- If we want to create a class, which can store some data, but we are not sure what type it is until the program compiles .



The usefulness of class template(2/2)

- A class template define a family of class :
  - Serve as a class outline to generate many classes
  - Specific classes are generated during compile time
- Class template promote code reusability
  - Reduce program develop time
  - In a class template , at least one argument is genetic

Creating class template

```
template <class T >
class data{
    T m_data ;
    public:
     data(T val):m_data(val)
{};
     void ShowData(){

cout<<m_data<<endl;
     }
}
```

```
int main(){
    data<int> d1(5) ;
    data<char> d2('D') ;

    data d3(5) ;      compile error,
without argument
    data<int> d4 ;      compile error,
no match constructor

    d1.ShowData();      d2.ShowData();
}
```

Template parameters(1/5)

- 3 forms of template parameters
  - type parameter
  - non-type parameter
  - template parameter

Template parameters(2/5)

- Type parameter

```
template <class T, class U, class P>
class c1{
        ……..
};
```

```
int main(){
c1<int,int,double>    x1; //T=int,   U=int,        P=double
c1<char,double,int>x2; //T=char, U=double, P=int

c1<int,int>x3; compile error, wrong number of argument
…….
}
```

Template parameters(3/5)

- Legal non-type parameter
  - integral types: int, char, bool
  - enumeration type
  - reference to object or function
  - pointer to object, function, member
- illegal non-type parameter
  - float, double
  - user-define class type
  - type void

```
template <data d1, double d2, float f>
class c1{
        ……..
};
```

⇒ illegal

Template parameters(4/5)

- non-type parameter

```
template <class T = int, int n = 10>
class c1{
        ……..
};
```

good

Note:
without default value of template argument, fill the argument completely when generating a template class .

```
Int main(){
C1< >          x1 ; OK
C1< , >        x2 ;  error, arg1、arg2 invalid
C1< ,30>       x3 ; error, arg1 invalid
C1<double>          x4 ;     OK, T = double, n = 10
C1<double,30>  x5 ;  OK
```

Template parameters(5/5)

- Template parameter

```
template<class A = int , int n = 10 >
class data{
        A m_data;
        public: //…..
                };
```

```
Int main(){

    data<int,10> d1;
    data<int,30> d2;
    data<double>d3;
    oop<data> x1(d1); OK
    oop<data> x2(d2); error, n=10 not match
    oop<data> x3(d3); error, int not match
    ……

}
```

```
template<   template<class A = int, int n = 10 > class V >
class oop{
        V<  > m_data ;
        public:
                oop(V<  > data):m_data(data){};
};
```

Inheritance in templates

```
template <class T>
class basic{
      T x ;
      public:
        basic(T val): x(val){};
        void ShowX(){cout <<
x;}
};
```

```
class derive1: public basic<double>{
      double y ;
      public:
          derive1(double a, double b):
                    basic<double>(a), y(b){}
          void ShowY(){cout<< y;}
};
```

Normal class

```
template <class T>
class derive2: public basic<T>{
      T y ;
      public:
          derive2(T a, T b):basic<T>(a), y(b){}
          void ShowY(){cout<< y;}
};
```

Class template

Inheritance in templates

```
int main(){
        basic<int>        obj1(100);
        derive1           obj2(100.1,100.1) ;
        derive2<char>     obj3('A','B');


        obj1.ShowX(); cout<<endl;
        obj2.ShowX(); cout<<"   "; obj2.ShowY();
cout<<endl;
        obj3.ShowX(); cout<<"   "; obj3.ShowY();
cout<<endl;
…
}
```

```
100
100.1   100.1
A   B
```

Lab Exercise – Shopping Cart

- Main

```
int main() {
    // Example usage
    Product<string> iPad("iPad pro 11 256GB", 34900, 2);
    Product<int> Phone(1257846, 23800, 3);
    Product<string> Ticket("TPE -> HND", 10500, 8);
    Product<int> Student(112511999, 123546, 1);

    // Shopping cart can hold up to ten products
    ShoppingCart<string, 10> cart;
    cart.addProduct(iPad);
    cart.addProduct(Product<string>(to_string(Phone.getName()), Phone.getPrice(), Phone.getQuantity())); // Convert Phone name to string
    cart.addProduct(Ticket);
    cart.addProduct(Product<string>(to_string(Student.getName()), Student.getPrice(), Student.getQuantity())); // Convert Student name to string
    cart.displayCartContents();

    return 0;
}
```

Lab Exercise – Shopping Cart

- Class Product
  - Every product with its Name(or Number), Price, and Quantity
  - Need to define the variables

```
// Template class for products
template <typename T>
class Product {
public:
    Product() : name(T()), price(0), quantity(0) {}

    Product(const T& name, int price, int quantity) : name(name), price(price), quantity(quantity) {}

    T getName() const {
        // Write your code

    }
    int getPrice() const {
        // Write your code

    }
    int getQuantity() const {
        // Write your code

    }

private:
    // Write your code
    // Define name, price, and quantity

};
```

Lab Exercise – Shopping Cart

- Class ShoppingCart
  - Calculate the price, and display the detail

```
// Template class for shopping cart
template <typename T, int MaxProducts>
class ShoppingCart {
public:
    ShoppingCart() : total(0), productCount(0) {
        // products array
        for (int i = 0; i < MaxProducts; ++i) {
            // Write your code

        }
    }

    // Calculate the total price
    void addProduct(const Product<T>& product) {
        // Write your code

    }

    // Display the cart contents
    void displayCartContents() {
        // Write your code
        // Make sure your output format same as OJ
        // Ex:Product Name or Number: iPad pro 11 256GB, Price: 34900, Quantity: 2

    }

private:
    int total;
    Product<T> products[MaxProducts];
    int productCount;
};
```

Execution result

- Output

```
Shopping Cart Contents:
Product Name or Number: iPad pro 11 256GB, Price: 34900, Quantity: 2
Product Name or Number: 1257846, Price: 23800, Quantity: 3
Product Name or Number: TPE -> HND, Price: 10500, Quantity: 8
Product Name or Number: 112511999, Price: 123546, Quantity: 1
Total Price: 348746
```

Submission

- You should exactly follow the output format
  - Compile

    g++ main.cpp -I . -o Lab10
  - Run

    ./Lab10
  - OJ

    /home/share/demo_OOP112_2 Lab 10