Problem 1
Before:

* A template function defines a "prototype" function. During compilation, the compiler will generate the matching functions base on the function call argument type and the template (argument deduction).
* The size of the compiled program is the same as you write each of the function separately.

Answer:

* x, y: 3, 3.2; z:3 conversion; w: no default conversion; m: no matching function

After:

* Return type doesn't matter. conversion.
* Template data type for at least one argument. Can't have only the return type as the template data type.

Problem 2

* explicit function specializations first then argument deduction.

Problem 3

* Once the matched function is generated, all function calls has to perform correctly.

```
bool lessThan(const Animal& a, const Animal& b)
{
    return (a.getWeight() < b.getWeight());
}
bool operator<(const Animal& a, const Animal& b)
{
    return (a.getWeight() < b.getWeight());
}
bool Animal::operator<(const Animal& a)const
{
    return m_weight < a.m_weight;
}
```

* Why getWeight has to be const?
* Why pass by const reference?
* Why const function

```
Stack/Queue/Vector/List/Map/Set

    list<int> li;
    for (list<int>::iterator it = li.begin(); it != li.end(); it++)
    {
        cout<<*it<<endl;
    }
list<obj*>  (*it)->
void func(const list<int> &l)
list<int>::const_iterator
```

```
for (list<int>::iterator it = li.begin(); it != li.end();)//no it++
{
    if (*it > 2)
        it = li.erase(it);
    else
        it++;
}
```

construction order
construct base class -> initialize data member(1, initialization list? 2. default constructor?) -> go to the body <- destructor
A(10), A() ,B(), A(5), B(5),C(), ~C(), ~(B),~(A), ~(B), ~(A), ~(A)

The destruction of data members is of the reverse order of construction(http://stackoverflow.com/questions/2254263/order-of-member-constructor-and-destructor-calls). I said during the discussion that m_b1 is destructed first. That is wrong. m_b2 is destructed first and then m_b1, because we constructed m_b1 first and then m_b2. With that in mind. The output should always be symmetric.