

Object Oriented Programming C++

Chapter: Templates

Coding Questions and Solutions (Set 4)

Contents: Part A: Function Template Programs. Part B: Class Template Programs. Part C: Predict the Output. Part D: Find the Error. Every program is complete, with its output. These problems are different from Sets 1, 2 and 3.

Part A: Function Template Programs

Q1. Write a function template `cube()` that returns the cube of a number of any numeric type.

Solution:

```
#include <iostream>
using namespace std;

template <class T>
T cube(T x)
{
    return x * x * x;
}

int main()
{
    cout << cube(3) << endl;
    cout << cube(1.5) << endl;
    cout << cube(-2) << endl;
    return 0;
}
```

Output:

```
27
3.375
-8
```

Q2. Write a function template `limitValue(v, lo, hi)` that keeps a value within a given range.

Solution:

```
#include <iostream>
using namespace std;

template <class T>
T limitValue(T v, T lo, T hi)
{
    if (v < lo) return lo;
    if (v > hi) return hi;
    return v;
}

int main()
{
    cout << limitValue(15, 0, 10) << endl;
}
```

```
    cout << limitValue(-3, 0, 10) << endl;
    cout << limitValue(5.5, 1.0, 9.0) << endl;
    return 0;
}
```

Output:

```
10
0
5.5
```

If the value is below the lower limit the lower limit is returned, if it is above the upper limit the upper limit is returned, otherwise the value itself is returned.

Q3. Write a function template to copy the elements of one array into another array.

Solution:

```
#include <iostream>
using namespace std;

template <class T>
void copyArray(T src[], T dest[], int n)
{
    for (int i = 0; i < n; i++)
        dest[i] = src[i];
}

int main()
{
    int a[] = {10, 20, 30, 40};
    int b[4];
    copyArray(a, b, 4);
    for (int i = 0; i < 4; i++)
        cout << b[i] << " ";
    cout << endl;

    char c[] = {'x', 'y', 'z'};
    char d[3];
    copyArray(c, d, 3);
    for (int i = 0; i < 3; i++)
        cout << d[i] << " ";
    cout << endl;
    return 0;
}
```

Output:

```
10 20 30 40
x y z
```

Q4. Write a function template to find the second largest element of an array (all elements are distinct).

Solution:

```
#include <iostream>
using namespace std;
```

```

template <class T>
T secondLargest(T a[], int n)
{
    T first = a[0], second = a[0];
    for (int i = 1; i < n; i++)
    {
        if (a[i] > first)
        {
            second = first;
            first = a[i];
        }
        else if (a[i] > second || second == first)
            second = a[i];
    }
    return second;
}

int main()
{
    int a[] = {12, 45, 7, 33};
    double b[] = {2.5, 9.5, 4.5};

    cout << secondLargest(a, 4) << endl;
    cout << secondLargest(b, 3) << endl;
    return 0;
}

```

Output:

```

33
4.5

```

Q5. Write a function template that checks whether an array is a palindrome (reads the same from both ends).

Solution:

```

#include <iostream>
using namespace std;

template <class T>
bool isPalindrome(T a[], int n)
{
    for (int i = 0; i < n / 2; i++)
        if (a[i] != a[n - 1 - i])
            return false;
    return true;
}

int main()
{
    int a[] = {1, 2, 3, 2, 1};
    char c[] = {'a', 'b', 'c'};
    double d[] = {1.5, 2.5, 1.5};

    cout << boolalpha;
    cout << isPalindrome(a, 5) << endl;
    cout << isPalindrome(c, 3) << endl;
    cout << isPalindrome(d, 3) << endl;
    return 0;
}

```

```
}
```

Output:

```
true  
false  
true
```

Q6. Write a function template `convert<R>(x)` in which the return type is given explicitly by the caller, to convert a value from one type to another.

Solution:

```
#include <iostream>  
using namespace std;  
  
template <class R, class T>  
R convert(T x)  
{  
    return (R)x;  
}  
  
int main()  
{  
    cout << convert<int>(3.7) << endl;  
    cout << convert<double>(5) << endl;  
    cout << convert<char>(66) << endl;  
    return 0;  
}
```

Output:

```
3  
5  
B
```

The return type `R` cannot be deduced from the arguments, so it is written explicitly in angle brackets as the first template argument. The second type `T` is still deduced from the argument. Converting `3.7` to `int` drops the decimal part, and `66` converted to `char` is the character `'B'`.

Q7. Write a recursive function template to find the factorial of a number.

Solution:

```
#include <iostream>  
using namespace std;  
  
template <class T>  
T factorial(T n)  
{  
    if (n <= 1)  
        return 1;  
    return n * factorial(n - 1);  
}  
  
int main()  
{  
    cout << factorial(5) << endl;  
}
```

```
    cout << factorial(5.0) << endl;
    cout << factorial(10L) << endl;
    return 0;
}
```

Output:

```
120
120
3628800
```

The same template is instantiated for `int`, `double` and `long`. Each version calls itself recursively with the same type.

Q8. Write a function template `applyToAll()` that applies a given function to every element of an array. Use another function template `doubleIt()` as the function.

Solution:

```
#include <iostream>
using namespace std;

template <class T>
T doubleIt(T x)
{
    return x * 2;
}

template <class T>
void applyToAll(T a[], int n, T (*f)(T))
{
    for (int i = 0; i < n; i++)
        a[i] = f(a[i]);
}

int main()
{
    int a[] = {1, 2, 3};
    applyToAll(a, 3, doubleIt<int>);
    for (int i = 0; i < 3; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

Output:

```
2 4 6
```

The address of a particular instance, `doubleIt<int>`, is passed as a function pointer.

Part B: Class Template Programs

Q9. Write a class template `Rectangle` with functions to find the area and the perimeter. Use it with `int` and `double`.

Solution:

```

#include <iostream>
using namespace std;

template <class T>
class Rectangle
{
    T length, width;
public:
    Rectangle(T l, T w) { length = l; width = w; }
    T area() { return length * width; }
    T perimeter() { return 2 * (length + width); }
};

int main()
{
    Rectangle<int> r1(4, 5);
    cout << "Area: " << r1.area() << " Perimeter: " << r1.perimeter() <<
endl;

    Rectangle<double> r2(2.5, 4.0);
    cout << "Area: " << r2.area() << " Perimeter: " << r2.perimeter() <<
endl;
    return 0;
}

```

Output:

```

Area: 20 Perimeter: 18
Area: 10 Perimeter: 13

```

Q10. Write a class template `Matrix` for a 2 x 2 matrix with functions to set values, add two matrices and display the result.

Solution:

```

#include <iostream>
using namespace std;

template <class T>
class Matrix
{
    T m[2][2];
public:
    void input(T a, T b, T c, T d)
    {
        m[0][0] = a; m[0][1] = b;
        m[1][0] = c; m[1][1] = d;
    }

    Matrix add(Matrix o)
    {
        Matrix r;
        for (int i = 0; i < 2; i++)
            for (int j = 0; j < 2; j++)
                r.m[i][j] = m[i][j] + o.m[i][j];
        return r;
    }

    void display()
    {

```

```

        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
                cout << m[i][j] << " ";
            cout << endl;
        }
    };

int main()
{
    Matrix<int> A, B;
    A.input(1, 2, 3, 4);
    B.input(5, 6, 7, 8);
    Matrix<int> C = A.add(B);
    C.display();
    return 0;
}

```

Output:

```

6 8
10 12

```

Q11. Write a class template `MinMax` that keeps track of the smallest and largest values added to it.

Solution:

```

#include <iostream>
using namespace std;

template <class T>
class MinMax
{
    T mn, mx;
    bool empty;
public:
    MinMax() { empty = true; }

    void add(T x)
    {
        if (empty)
        {
            mn = mx = x;
            empty = false;
        }
        else
        {
            if (x < mn) mn = x;
            if (x > mx) mx = x;
        }
    }

    T getMin() { return mn; }
    T getMax() { return mx; }
};

int main()
{

```

```

MinMax<int> a;
a.add(5); a.add(2); a.add(9); a.add(4);
cout << "Min: " << a.getMin() << " Max: " << a.getMax() << endl;

MinMax<double> b;
b.add(3.5); b.add(1.5);
cout << "Min: " << b.getMin() << " Max: " << b.getMax() << endl;
return 0;
}

```

Output:

```

Min: 2 Max: 9
Min: 1.5 Max: 3.5

```

Q12. Write a class `Student` as a class template in which the type of the marks is generic (for example `int` or `float`).

Solution:

```

#include <iostream>
#include <string>
using namespace std;

template <class T>
class Student
{
    string name;
    T marks;
public:
    Student(string n, T m) { name = n; marks = m; }
    void show() { cout << name << ": " << marks << endl; }
};

int main()
{
    Student<int> s1("Ravi", 85);
    Student<float> s2("Asha", 89.5f);
    s1.show();
    s2.show();
    return 0;
}

```

Output:

```

Ravi: 85
Asha: 89.5

```

Q13. Write a class template `Point` and overload the `<<` operator (as a friend) to print its objects.

Solution:

```

#include <iostream>
using namespace std;

template <class T>
class Point
{
    T x, y;
public:

```

```

    Point(T a, T b) { x = a; y = b; }

    friend ostream &operator<<(ostream &o, const Point &p)
    {
        o << "(" << p.x << ", " << p.y << ")";
        return o;
    }
};

int main()
{
    Point<int> p1(2, 3);
    Point<double> p2(1.5, 2.5);
    cout << p1 << endl;
    cout << p2 << endl;
    return 0;
}

```

Output:

```

(2, 3)
(1.5, 2.5)

```

Q14. Write a normal (non-template) class `Display` that contains a member function template `show()`, which can display a value of any type.

Solution:

```

#include <iostream>
using namespace std;

class Display
{
public:
    template <class T>
    void show(T x)
    {
        cout << "Value: " << x << endl;
    }
};

int main()
{
    Display d;
    d.show(10);
    d.show(2.5);
    d.show("Hi");
    return 0;
}

```

Output:

```

Value: 10
Value: 2.5
Value: Hi

```

Only the member function is a template, not the whole class. The class is an ordinary class, so it is used without angle brackets.

Q15. Write a class template `SafeArray` with a non-type size parameter and an overloaded `[]` operator that checks the index.

Solution:

```
#include <iostream>
using namespace std;

template <class T, int N>
class SafeArray
{
    T a[N];
public:
    SafeArray()
    {
        for (int i = 0; i < N; i++)
            a[i] = T();
    }

    T &operator[](int i)
    {
        if (i < 0 || i >= N)
        {
            cout << "Index out of range" << endl;
            return a[0];
        }
        return a[i];
    }
};

int main()
{
    SafeArray<int, 3> s;
    s[0] = 10;
    s[1] = 20;
    s[2] = 30;
    cout << s[1] << endl;
    cout << s[5] << endl;
    return 0;
}
```

Output:

```
20
Index out of range
10
```

For the invalid index 5 the message is printed first and the first element (10) is returned and displayed.

Q16. Write a Stack class template of capacity 2 that uses exception handling to report overflow and underflow.

Solution:

```
#include <iostream>
using namespace std;

template <class T>
class Stack
```

```

{
    T arr[2];
    int top;
public:
    Stack() { top = -1; }

    void push(T x)
    {
        if (top == 1)
            throw "Stack Overflow";
        arr[++top] = x;
    }

    T pop()
    {
        if (top == -1)
            throw "Stack Underflow";
        return arr[top--];
    }
};

int main()
{
    Stack<int> s;
    try
    {
        s.push(1);
        s.push(2);
        s.push(3);
    }
    catch (const char *msg)
    {
        cout << "Error: " << msg << endl;
    }

    try
    {
        cout << s.pop() << endl;
        cout << s.pop() << endl;
        cout << s.pop() << endl;
    }
    catch (const char *msg)
    {
        cout << "Error: " << msg << endl;
    }
    return 0;
}

```

Output:

```

Error: Stack Overflow
2
1
Error: Stack Underflow

```

Part C: Predict the Output

Q17. Predict the output of the following program.

Solution:

```

#include <iostream>
using namespace std;

template <class T>
class Test
{
public:
    static int n;
    Test() { n++; }
};

template <class T>
int Test<T>::n = 0;

int main()
{
    Test<int> a, b;
    Test<char> c;
    Test<int> d;
    cout << Test<int>::n << " " << Test<char>::n << " " << Test<float>::n
<< endl;
    return 0;
}

```

Output:

```
3 1 0
```

Explanation: Test<int>, Test<char> and Test<float> are three different classes, each with its own static variable n. Three objects of Test<int> were created (a, b, d), one of Test<char> (c) and none of Test<float>.

Q18. Predict the output of the following program (assume int is 4 bytes).

Solution:

```

#include <iostream>
using namespace std;

template <class T>
void show(T x)
{
    cout << sizeof(T) << " ";
}

int main()
{
    show(1);
    show(1.5);
    show('a');
    cout << endl;
    return 0;
}

```

Output:

```
4 8 1
```

Explanation: The size of `T` depends on the type deduced from the argument: `int` (4 bytes), `double` (8 bytes) and `char` (1 byte).

Q19. Predict the output of the following program.

Solution:

```
#include <iostream>
using namespace std;

template <class T, int N = 3>
class Arr
{
public:
    int size() { return N; }
};

int main()
{
    Arr<int> a;
    Arr<char, 10> b;
    cout << a.size() << " " << b.size() << endl;
    return 0;
}
```

Output:

```
3 10
```

Explanation: For `Arr<int>` the default value of `N` (3) is used. For `Arr<char, 10>` the value 10 is given explicitly.

Part D: Find the Error

Q20. Find the error in the following code and correct it.

```
template <class T>
void show(T a, int b) { cout << "First" << endl; }

template <class T>
void show(int a, T b) { cout << "Second" << endl; }

int main()
{
    show(1, 2);
    return 0;
}
```

Error: For the call `show(1, 2)` both templates match equally well (with `T` as `int`), so the compiler cannot decide which one to use. This is an **ambiguous call**.

Correction: remove one of the templates, or add an ordinary function for this exact case, which is always preferred over templates.

```
void show(int a, int b) { cout << "Ordinary" << endl; }
```

Q21. The following program gives a linker error. Find the reason and correct it.

```
// maths.h
template <class T>
T add(T a, T b);

// maths.cpp
#include "maths.h"
template <class T>
T add(T a, T b) { return a + b; }

// main.cpp
#include <iostream>
#include "maths.h"
int main()
{
    std::cout << add(2, 3);
    return 0;
}
```

Error: The template is only declared in the header, and its definition is placed in `maths.cpp`. While compiling `main.cpp` the compiler cannot see the body of the template, so it cannot instantiate `add<int>`, and the linker reports an undefined reference.

Correction: put the complete definition of the template in the header file.

```
// maths.h
template <class T>
T add(T a, T b)
{
    return a + b;
}
```