

Object Oriented Programming C++

Chapter: Templates

Coding Questions and Solutions (Set 3)

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 and 2.

Part A: Function Template Programs

Q1. Write a function template to find the sum of all elements of an array.

Solution:

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

template <class T>
T sumArray(T a[], int n)
{
    T s = 0;
    for (int i = 0; i < n; i++)
        s += a[i];
    return s;
}

int main()
{
    int a[] = {1, 2, 3, 4, 5};
    double b[] = {1.5, 2.5, 3.0};

    cout << "Sum of int array: " << sumArray(a, 5) << endl;
    cout << "Sum of double array: " << sumArray(b, 3) << endl;
    return 0;
}
```

Output:

```
Sum of int array: 15
Sum of double array: 7
```

Q2. Write a function template power (base, exp) that returns base raised to the power exp.

Solution:

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

template <class T>
T power(T base, int exp)
{
    T result = 1;
    for (int i = 0; i < exp; i++)
        result = result * base;
    return result;
}
```

```
int main()
{
    cout << power(2, 5) << endl;
    cout << power(1.5, 2) << endl;
    cout << power(3, 0) << endl;
    return 0;
}
```

Output:

```
32
2.25
1
```

Q3. Write a function template to count how many times a given element occurs in an array.

Solution:

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

template <class T>
int countOccurrences(T a[], int n, T key)
{
    int count = 0;
    for (int i = 0; i < n; i++)
        if (a[i] == key)
            count++;
    return count;
}

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

    cout << countOccurrences(a, 5, 2) << endl;
    cout << countOccurrences(c, 3, 'a') << endl;
    return 0;
}
```

Output:

```
3
2
```

Q4. Write a function template to find the average of the elements of an array. The result must be a decimal value even for an int array.

Solution:

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

template <class T>
double average(T a[], int n)
{
    T s = 0;
    for (int i = 0; i < n; i++)
```

```

        s += a[i];
    return (double)s / n;
}

int main()
{
    int a[] = {10, 20, 30, 45};
    double b[] = {2.5, 3.5};

    cout << "Average: " << average(a, 4) << endl;
    cout << "Average: " << average(b, 2) << endl;
    return 0;
}

```

Output:

```

Average: 26.25
Average: 3

```

The return type is fixed as `double`, so the sum of `int` elements (105) is divided as a decimal value, $105 / 4 = 26.25$.

Q5. Write a function template `absolute()` that returns the absolute value of a number of any numeric type.

Solution:

```

#include <iostream>
using namespace std;

template <class T>
T absolute(T x)
{
    return (x < 0) ? -x : x;
}

int main()
{
    cout << absolute(-5) << endl;
    cout << absolute(-2.5) << endl;
    cout << absolute(7) << endl;
    return 0;
}

```

Output:

```

5
2.5
7

```

Q6. Write a function template `isEqual()` that checks whether two values are equal. Give an explicit specialization so that it also works correctly for C-style strings.

Solution:

```

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

```

```

template <class T>
bool isEqual(T a, T b)
{
    return a == b;
}

template <>
bool isEqual<const char *>(const char *a, const char *b)
{
    return strcmp(a, b) == 0;
}

int main()
{
    cout << boolalpha;
    cout << isEqual(10, 10) << endl;
    cout << isEqual(2.5, 3.5) << endl;
    cout << isEqual("hello", "hello") << endl;
    return 0;
}

```

Output:

```

true
false
true

```

For `const char *` the general template would compare the addresses of the strings, not their contents. The specialization uses `strcmp()` to compare the characters.

Q7. Write overloaded function templates `printValue()`: one that prints a value, and another that prints a value a given number of times on the same line.

Solution:

```

#include <iostream>
using namespace std;

template <class T>
void printValue(T x)
{
    cout << x << endl;
}

template <class T>
void printValue(T x, int times)
{
    for (int i = 0; i < times; i++)
        cout << x;
    cout << endl;
}

int main()
{
    printValue(5);
    printValue(2.5);
    printValue('*', 5);
    return 0;
}

```

Output:

```
5
2.5
*****
```

The two templates differ in the number of parameters, so the compiler selects the correct one from the number of arguments in the call.

Q8. Write a function template that prints the elements of any `vector`. Use it with `vector<int>` and `vector<string>`, and also sort the integer vector.

Solution:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

template <class T>
void printVector(const vector<T> &v)
{
    for (size_t i = 0; i < v.size(); i++)
        cout << v[i] << " ";
    cout << endl;
}

int main()
{
    vector<int> a;
    a.push_back(3);
    a.push_back(1);
    a.push_back(2);

    vector<string> s;
    s.push_back("apple");
    s.push_back("banana");

    printVector(a);
    sort(a.begin(), a.end());
    printVector(a);
    printVector(s);
    return 0;
}
```

Output:

```
3 1 2
1 2 3
apple banana
```

`vector` and `sort` are themselves templates of the STL. Here a function template is written to work with any `vector<T>`.

Q9. Write a function template `findMax()` that finds the largest element of an array. Use it with an `int` array and with an array of `Student` objects (compare by marks).

Solution:

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

class Student
{
public:
    string name;
    int marks;
    Student(string n = "", int m = 0) { name = n; marks = m; }
    bool operator>(const Student &s) const { return marks > s.marks; }
};

template <class T>
T findMax(T arr[], int n)
{
    T m = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > m)
            m = arr[i];
    return m;
}

int main()
{
    int a[] = {4, 9, 2};
    cout << findMax(a, 3) << endl;

    Student s[3] = {Student("Ravi", 72), Student("Asha", 91),
Student("Kiran", 85)};
    Student top = findMax(s, 3);
    cout << top.name << " " << top.marks << endl;
    return 0;
}
```

Output:

```
9
Asha 91
```

The class `Student` overloads `operator>`, which the template uses, so the same template works for the user-defined type.

Part B: Class Template Programs

Q10. Write a class template `Array` that allocates memory dynamically, and has functions to set a value, display the array and find the maximum element.

Solution:

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

```

template <class T>
class Array
{
    T *ptr;
    int size;
public:
    Array(int n)
    {
        size = n;
        ptr = new T[n];
    }
    ~Array() { delete[] ptr; }

    void set(int i, T v) { ptr[i] = v; }

    void display()
    {
        for (int i = 0; i < size; i++)
            cout << ptr[i] << " ";
        cout << endl;
    }

    T maximum()
    {
        T m = ptr[0];
        for (int i = 1; i < size; i++)
            if (ptr[i] > m)
                m = ptr[i];
        return m;
    }
};

int main()
{
    Array<int> a(4);
    for (int i = 0; i < 4; i++)
        a.set(i, (i + 1) * 10);
    a.display();
    cout << "Max: " << a.maximum() << endl;

    Array<double> b(3);
    b.set(0, 1.5);
    b.set(1, 2.5);
    b.set(2, 0.5);
    b.display();
    cout << "Max: " << b.maximum() << endl;
    return 0;
}

```

Output:

```

10 20 30 40
Max: 40
1.5 2.5 0.5
Max: 2.5

```

Q11. Write a class template for a singly linked list with functions to insert at the end and display the list.

Solution:

```

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

template <class T>
struct Node
{
    T data;
    Node<T> *next;
    Node(T d) { data = d; next = NULL; }
};

template <class T>
class LinkedList
{
    Node<T> *head;
public:
    LinkedList() { head = NULL; }

    void insert(T x)
    {
        Node<T> *n = new Node<T>(x);
        if (head == NULL)
            head = n;
        else
        {
            Node<T> *t = head;
            while (t->next != NULL)
                t = t->next;
            t->next = n;
        }
    }

    void display()
    {
        Node<T> *t = head;
        while (t != NULL)
        {
            cout << t->data << " -> ";
            t = t->next;
        }
        cout << "NULL" << endl;
    }

    ~LinkedList()
    {
        while (head != NULL)
        {
            Node<T> *t = head;
            head = head->next;
            delete t;
        }
    }
};

int main()
{
    LinkedList<int> l1;
    l1.insert(10);
    l1.insert(20);
    l1.insert(30);
}

```

```

        l1.display();

        LinkedList<char> l2;
        l2.insert('a');
        l2.insert('b');
        l2.display();
        return 0;
    }

```

Output:

```

10 -> 20 -> 30 -> NULL
a -> b -> NULL

```

Q12. Write a class template `Pair` with a constructor and functions `getMax()` and `getMin()`. Define all member functions outside the class.

Solution:

```

#include <iostream>
using namespace std;

template <class T>
class Pair
{
    T a, b;
public:
    Pair(T x, T y);
    T getMax();
    T getMin();
};

template <class T>
Pair<T>::Pair(T x, T y)
{
    a = x;
    b = y;
}

template <class T>
T Pair<T>::getMax()
{
    return (a > b) ? a : b;
}

template <class T>
T Pair<T>::getMin()
{
    return (a < b) ? a : b;
}

int main()
{
    Pair<int> p1(15, 8);
    cout << "Max: " << p1.getMax() << " Min: " << p1.getMin() << endl;

    Pair<double> p2(2.5, 7.5);
    cout << "Max: " << p2.getMax() << " Min: " << p2.getMin() << endl;
    return 0;
}

```

```
}
```

Output:

```
Max: 15 Min: 8  
Max: 7.5 Min: 2.5
```

Q13. Write a class template `Vec` for a two-dimensional vector and overload the `+` operator. Use it with `int` and `double`.

Solution:

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

template <class T>
class Vec
{
    T x, y;
public:
    Vec(T a = 0, T b = 0) { x = a; y = b; }

    Vec operator+(Vec v)
    {
        return Vec(x + v.x, y + v.y);
    }

    void show()
    {
        cout << "(" << x << ", " << y << ")" << endl;
    }
};

int main()
{
    Vec<int> a(1, 2), b(3, 4);
    (a + b).show();

    Vec<double> c(1.5, 2.5), d(0.5, 0.5);
    (c + d).show();
    return 0;
}
```

Output:

```
(4, 6)
(2, 3)
```

For the double vectors, the results 2.0 and 3.0 are printed by `cout` as 2 and 3.

Q14. Write a class template `Buffer` with a non-type parameter `N` for its capacity. Show what happens when more than `N` items are added.

Solution:

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

template <class T, int N>
```

```

class Buffer
{
    T data[N];
    int count;
public:
    Buffer() { count = 0; }

    bool add(T x)
    {
        if (count == N)
        {
            cout << "Buffer full" << endl;
            return false;
        }
        data[count++] = x;
        return true;
    }

    int size() { return count; }
    int capacity() { return N; }
};

int main()
{
    Buffer<int, 3> b;
    b.add(1);
    b.add(2);
    b.add(3);
    b.add(4);
    cout << b.size() << "/" << b.capacity() << endl;
    return 0;
}

```

Output:

```

Buffer full
3/3

```

Q15. Write a class template `Info` that gives the name of a data type. Use specialization to return correct names for `int` and `double`, and a default name for all other types.

Solution:

```

#include <iostream>
using namespace std;

template <class T>
class Info
{
public:
    static const char *name() { return "unknown"; }
};

template <>
class Info<int>
{
public:
    static const char *name() { return "int"; }
};

```

```

template <>
class Info<double>
{
public:
    static const char *name() { return "double"; }
};

int main()
{
    cout << Info<int>::name() << endl;
    cout << Info<double>::name() << endl;
    cout << Info<char>::name() << endl;
    return 0;
}

```

Output:

```

int
double
unknown

```

Q16. Write a program to reverse a string using a Stack class template.

Solution:

```

#include <iostream>
using namespace std;
#define SIZE 20

template <class T>
class Stack
{
    T arr[SIZE];
    int top;
public:
    Stack() { top = -1; }
    void push(T x) { arr[++top] = x; }
    T pop() { return arr[top--]; }
    bool isEmpty() { return top == -1; }
};

int main()
{
    char str[] = "HELLO";
    Stack<char> s;

    for (int i = 0; str[i] != '\0'; i++)
        s.push(str[i]);

    cout << "Reversed: ";
    while (!s.isEmpty())
        cout << s.pop();
    cout << endl;
    return 0;
}

```

Output:

```

Reversed: OLLEH

```

A stack removes items in last-in first-out order, so the characters come out in reverse order.

Part C: Predict the Output

Q17. Predict the output of the following program.

Solution:

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

template <class T>
void f(T x) { cout << "T" << endl; }

template <>
void f<int>(int x) { cout << "int" << endl; }

void f(double x) { cout << "double" << endl; }

int main()
{
    f(1);
    f(2.5);
    f('a');
    f(3.5f);
    return 0;
}
```

Output:

```
int
double
T
T
```

Explanation: `f(1)` deduces `T` as `int`, so the explicit specialization runs. `f(2.5)` matches the ordinary function `f(double)` exactly, and an ordinary function is preferred over a template. `f('a')` and `f(3.5f)` match the template exactly (`char` and `float`), whereas the ordinary function would need a conversion, so the template is chosen.

Q18. Predict the output of the following program.

Solution:

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

template <class T>
T add(T a, T b)
{
    return a + b;
}

int main()
{
    cout << add(1, 2);
    cout << add(1.5, 2.5);
}
```

```
    cout << add<int>(1.9, 2.9);  
    cout << endl;  
    return 0;  
}
```

Output:

```
343
```

Explanation: add(1, 2) gives 3. add(1.5, 2.5) gives 4.0, printed as 4. In add<int>(1.9, 2.9) the type is forced to int, so the arguments become 1 and 2 (the decimal part is dropped) and the result is 3. The three values are printed together as 343.

Part D: Find the Error

Q19. Find the error in the following code and correct it.

```
template <class T>  
class Sample  
{  
    T v;  
public:  
    T get();  
};  
  
T Sample<T>::get()  
{  
    return v;  
}
```

Error: The member function is defined outside the class without the template header, so T is not known.

Correction:

```
template <class T>  
T Sample<T>::get()  
{  
    return v;  
}
```

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

```
template <class T, int N>  
class Array  
{  
    T a[N];  
};  
  
int main()  
{  
    int n = 5;  
    Array<int, n> x;  
    return 0;  
}
```

Error: A non-type template argument must be a constant expression known at compile time. The variable `n` is an ordinary variable, so it cannot be used.

Correction: make `n` a constant, or write the value directly.

```
const int n = 5;  
Array<int, n> x;      // or  Array<int, 5> x;
```