**Quiz 6**

1. **Which of the following is true about templates.**

1) Template is a feature of C++ that allows us to write one code for different data types.  
  
2) We can write one function that can be used for all data types including user defined types. Like sort(), max(), min(), ..etc.  
  
3) We can write one class or struct that can be used for all data types including user defined types. Like Linked List, Stack, Queue ..etc.  
  
4) Template is an example of compile time polymorphism.

|  |  |
| --- | --- |
| **A** | 1 and 2 |
| **B** | 1, 2 and 3 |
| **C** | 1, 2 and 4 |
| **D** | All of the above |

1. **When are templates usually instantiated?**

|  |  |
| --- | --- |
| **A** | At runtime |
| **B** | At compile time |
| **C** | At link time |

1. **Predict output of the following program**

*#include <iostream>*

*using namespace std;*

*template <typename T>*

*void fun(const T&x)*

*{*

*static int count = 0;*

*cout << "x = " << x << " count = " << count << endl;*

*++count;*

*return;*

*}*

*int main()*

*{*

*fun<int> (1);*

*cout << endl;*

*fun<int>(1);*

*cout << endl;*

*fun<double>(1.1);*

*cout << endl;*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | x = 1 count = 0  x = 1 count = 1  x = 1.1 count = 0 |
| **B** | x = 1 count = 0  x = 1 count = 0  x = 1.1 count = 0 |
| **C** | x = 1 count = 0  x = 1 count = 1  x = 1.1 count = 2 |
| **D** | Compiler Error |

*#include <iostream>*

*using namespace std;*

*template <typename T>*

*T max(T x, T y)*

*{*

*return (x > y)? x : y;*

*}*

*int main()*

*{*

*cout << max(3, 7) << std::endl;*

*cout << max(3.0, 7.0) << std::endl;*

*cout << max(3, 7.0) << std::endl;*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | 7  7.0  7.0 |
| **B** | Compiler Error in all cout statements as data type is not specified |
| **C** | Compiler Error in last cout statement as call to max is ambiguous. |
| **D** | None of the above |

1. **Output of following program?**

*#include <iostream>*

*using namespace std;*

*template <class T>*

*class Test*

*{*

*private:*

*T val;*

*public:*

*static int count;*

*Test()  {   count++;   }*

*};*

*template<class T>*

*int Test<T>::count = 0;*

*int main()*

*{*

*Test<int> a;*

*Test<int> b;*

*Test<double> c;*

*cout << Test<int>::count   << endl;*

*cout << Test<double>::count << endl;*

*return 0;*

*}*

|  |  |
| --- | --- |
| **A** | 0  0 |
| **B** | 1  1 |
| **C** | 2  1 |
| **D** | 1  0 |

1. **Output of following program? Assume that the size of char is 1 byte and size of int is 4 bytes, and there is no alignment done by the compiler.**

*#include<iostream>*

*#include<stdlib.h>*

*using namespace std;*

*template<class T, class U>*

*class A  {*

*T x;*

*U y;*

*static int count;*

*};*

*int main()  {*

*A<char, char> a;*

*A<int, int> b;*

*cout << sizeof(a) << endl;*

*cout << sizeof(b) << endl;*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | 6  12 |
| **B** | 2  8 |
| **C** | Compiler Error: There can not be more than one template arguments. |
| **D** | 8  8 |

1. **Output of following program? Assume that the size of int is 4 bytes and size of double is 8 bytes, and there is no alignment done by the compiler.**

*#include<iostream>*

*#include<stdlib.h>*

*using namespace std;*

*template<class T, class U, class V=double>*

*class A  {*

*T x;*

*U y;*

*V z;*

*static int count;*

*};*

*int main()*

*{*

*A<int, int> a;*

*A<double, double> b;*

*cout << sizeof(a) << endl;*

*cout << sizeof(b) << endl;*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | 16  24 |
| **B** | 16  24 |
| **C** | 20  28 |
| **D** | Compiler Error: template parameters cannot have default values |

1. **Output of following program.**

|  |
| --- |
| *#include <iostream>*  *using namespace std;*    *template <class T, int max>*  *int arrMin(T arr[], int n)*  *{*  *int m = max;*  *for (int i = 0; i < n; i++)*  *if (arr[i] < m)*  *m = arr[i];*    *return m;*  *}*    *int main()*  *{*  *int arr1[]  = {10, 20, 15, 12};*  *int n1 = sizeof(arr1)/sizeof(arr1[0]);*    *char arr2[] = {1, 2, 3};*  *int n2 = sizeof(arr2)/sizeof(arr2[0]);*    *cout << arrMin<int, 10000>(arr1, n1) << endl;*  *cout << arrMin<char, 256>(arr2, n2);*  *return 0;*  *}* |

**Output:**

|  |  |
| --- | --- |
| **A** | Compiler error, template argument must be a data type. |
| **B** | 10  1 |
| **C** | 10000  256 |
| **D** | 1  1 |

1. **Output?**

|  |
| --- |
| *#include <iostream>*  *using namespace std;*    *template <int i>*  *void fun()*  *{*  *i = 20;*  *cout << i;*  *}*    *int main()*  *{*  *fun<10>();*  *return 0;*  *}* |

**Output:**

|  |  |
| --- | --- |
| **A** | 10 |
| **B** | 20 |
| **C** | Compiler Error |

1. **Output?**

|  |
| --- |
| *#include <iostream>*  *using namespace std;*    *template <class T>*  *T max (T &a, T &b)*  *{*  *return (a > b)? a : b;*  *}*    *template <>*  *int max <int> (int &a, int &b)*  *{*  *cout << "Called ";*  *return (a > b)? a : b;*  *}*    *int main ()*  *{*  *int a = 10, b = 20;*  *cout << max <int> (a, b);*  *}* |

**Output:**

|  |  |
| --- | --- |
| **A** | 20 |
| **B** | Called 20 |
| **C** | Compiler Error |

**Result:**

1. D

2. B.

3. A Compiler creates a new instance of a template function for every data type. So compiler creates two functions in the above example, one for int and other for double. Every instance has its own copy of static variable. The int instance of function is called twice, so count is incremented for the second call.

4. C

5. C here are two classes created by the template: Test and Test. Since count is a static member, every class has its own copy of it. Also, count gets incremented in constructor.

6. B Since count is static, it is not counted in sizeof.

7. A templates can also have default parameters. The rule is same all default values must be on the rightmost side. Since count is static, it is not counted in sizeof.

8. B

9. C Compiler error in line "i = 20;" Non-type parameters must be const, so they cannot be modified.

10. B Above program is an example of template specialization. Sometime we want a different behaviour of a function/class template for a particular data type. For this, we can create a specialized version for that particular data type.