**Templates**

**Writer: Sayak Haldar**

**1.What is a template?**

a) A template is a formula for creating a generic class

b) A template is used to manipulate the class

c) A template is used for creating the attributes

d) None of the mentioned

Answer: a

**2. Pick out the correct statement about string template.**

a) It is used to replace a string

b) It is used to replace a string with another string at runtime

c) It is used to delete a string

d) None of the mentioned

**Answer: b**

Explanation: Every string template is used to replace the string with another string at runtime.

**3. How to declare a template?**

a) tem

b) temp

c) template<>

d) none of the mentioned

**Answer: c**

**4. What is the output of this program?**

#include <iostream>

using namespace std;

template <class T>

inline T square(T x)

{

T result;

result = x \* x;

return result;

};

template <>

string square<string>(string ss)

{

return (ss+ss);

};

int main()

{

int i = 4, ii;

string ww("A");

ii = square<int>(i);

cout << i << ii;

cout << square<string>(ww) << endl;

}

a) 416AA

b) 164AA

c) AA416

d) none of the mentioned

**Answer: a**

Explanation: In this program, We are using two template to calculate the square and to find the addition.

**Now, note that:** template function is overloaded

**Now,** template <>

string square<string>(string ss)

{

return (ss+ss);

};

This is called template specialization

**5. What is the output of this program?**

#include <iostream>

using namespace std;

template <typename T, typename U>

void squareAndPrint(T x, U y)

{

cout << x << x \* x << endl;

cout << y << " " << y \* y << endl;

};

int main()

{

int ii = 2;

float jj = 2.1;

squareAndPrint<int, float>(ii, jj);

}

a) 23

2.1 4.41

b) 24

2.1 4.41

c) 24

2.1 3.41

**Answer: b**

Explanation: In this multiple templated types, We are passing two values of different types and producing the result.

**6. What is the output of this program?**

#include <iostream>

#include <string>

using namespace std;

template<typename T>

void print\_mydata(T output)

{

cout << output << endl;

}

int main()

{

double d = 5.5;

string s("Hello World");

print\_mydata( d );

print\_mydata( s );

return 0;

}

a) 5.5

Hello World

b) 5.5

c) Hello World

d) None of the mentioned

**Answer) a)**

**7. How many types of templates are there in c++?**

**a) 1**

**b) 2**

**c) 3**

**d) 4**

**Answer: b**

Explanation: There are two types of templates. They are function template and class template.

**8. Which are done by compiler for templates?**

a) type-safe

b) portability

c) code elimination

d) all of the mentioned

**Answer: a**

Explanation: The compiler can determine at compile time whether the type associated with a template definition can perform all of the functions required by that template definition.

**9. What may be the name of the parameter that the template should take?**

a) same as template

b) same as class

c) same as function

d) none of the mentioned

**Answer: a**

**10. How many parameters are legal for non-type template?**

a) 1

b) 2

c) 3

d) 4

**Answer: d**

Explanation: The following are legal for non-type template parameters: integral or enumeration type, Pointer to object or pointer to function, Reference to object or reference to function, Pointer to member.

Now, first let me explain what is a non type template:

**template<class T,int size>**

**class array**

**{**

**T a[size];//automatic array initialization**

**//...**

**//...**

**};**

Now, the template supplies the size of array as an argument. This implies that the size of the array is known to the compiler at the compile time itself. The arguments must be specified whenever a template class is created. Now, to create an array using this templated class:

array<int,10> a1;

But, you cannot replace int size with T size.

Consider this full example taken from stackoverflow:

**#include<iostream>**

**using namespace std;**

**template <class T, T defaultValue>**

**class GenericClass**

**{**

**private:**

**T value;**

**public:**

**GenericClass()**

**{**

**value = defaultValue;**

**}**

**T returnVal()**

**{**

**return value;**

**}**

**};**

**int main()**

**{**

**GenericClass <int, 10> gcInteger;**

**GenericClass < float, 4.6f> gcFlaot;**

**cout << "\n sum of integer is "<<gcInteger.returnVal();**

**cout << "\n sum of float is "<<gcFlaot.returnVal();**

**return 0;**

**}**

It will generate the following error:

main.cpp: In function `int main()':

main.cpp:25: error: `float' is not a valid type for a template constant parameter

main.cpp:25: error: invalid type in declaration before ';' token

main.cpp:28: error: request for member `returnVal' in `gcFlaot',

which is of non-class type `int'

The reason is this:

The current C++ standard does not allow float (i.e. real number) or character string literals to be used as template non-type parameters. You can of course use the float and char \* types as normal arguments.

**11. What is a function template?**

a) creating a function without having to specify the exact type

b) creating a function with having a exact type

c) all of the mentioned

d) none of the mentioned

Answer: a

**12. Which is used to describe the function using placeholder types?**

a) template parameters

b) template type parameters

c) template type

d) none of the mentioned

**Answer: b**

**13. Pick out the correct statement.**

a) you only need to write one function, and it will work with many different types

b) it will take a long time to execute

c) duplicate code is increased

d) none of the mentioned

**Answer: a**

Explanation: Because of template type parameters, It will work with many types and saves a lot of time.

**14. What is the output of this program?**

#include <iostream>

using namespace std;

template<typename type>

type Max(type Var1, type Var2)

{

return Var1 > Var2 ? Var1:Var2;

}

int main()

{

int p;

p = Max(100, 200);

cout << p << endl;

return 0;

}

a) 100

b) 200

c) 300

d) 100200

**Answer: b**

Explanation: In this program, We are returning the maximum value by using function template.

**15. What is the output of this program?**

#include <iostream>

using namespace std;

template<typename type>

class Test

{

public:

Test()

{

};

~Test()

{

};

type Funct1(type Var1)

{

return Var1;

}

type Funct2(type Var2)

{

return Var2;

}

};

int main()

{

Test<int> Var1;

Test<float> Var2;

cout << Var1.Funct1(200) << endl;

cout << Var2.Funct2(3.123) << endl;

return 0;

}

a) 200

3.123

b) 3.123

200

c) 200

d) 3.123

**Answer: a**

**Explanation: In this program, We are passing the values and getting it back from template. And we are using the constructor and destructor for the function template.**

**16. What is the output of this program?**

#include <iostream>

using namespace std;

template<typename type>

class TestVirt

{

public:

virtual type TestFunct(type Var1)

{

return Var1 \* 2;

}

};

int main()

{

TestVirt<int> Var1;

cout << Var1.TestFunct(100) << endl;

return 0;

}

a) 100

b) 200

c) 50

d) none of the mentioned

**Answer: b**

Explanation: In this program, We are using class to pass the value and then we are manipulating it.

200

**17. What is the output of this program?**

#include <iostream>

using namespace std;

template<typename T>

inline T square(T x)

{

T result;

result = x \* x;

return result;

};

int main()

{

int i, ii;

float x, xx;

double y, yy;

i = 2;

x = 2.2;

y = 2.2;

ii = square(i);

cout << i << " " << ii << endl;

yy = square(y);

cout << y << " " << yy << endl;

}

a) 2 4

2.2 4.84

b) 2 4

c) error

d) runtime error

**Answer)a)**

**18. What is the output of this program?**

#include <iostream>

using namespace std;

template<typename T>

void loopIt(T x)

{

int count = 3;

T val[count];

for (int ii=0; ii < count; ii++)

{

val[ii] = x++;

cout << val[ii] << endl;

}

};

int main()

{

float xx = 2.1;

loopIt(xx);

}

a) 2.1

b) 3.1

c) 3.2

d) 2.1

3.1

4.1

**Answer) d)**

**19. What can be passed by non-type template parameters during compile time?**

a) int

b) float

c) constant expression

d) none of the mentioned

**Answer: c**

Explanation: Non-type template parameters provide the ability to pass a constant expression at compile time. The constant expression may also be an address of a function, object or static class member.

**20. From where does the template class derived?**

a) regular non-templated C++ class

b) templated class

c) regular non-templated C++ class or templated class

d) none of the mentioned

**Answer: c**

**21. What is meant by template parameter?**

a) It can be used to pass a type as argument

b) It can be used to evaluate a type

c) It can of no return type

d) None of the mentioned

Answer: a

**Explanation:** A template parameter is a special kind of parameter that can be used to pass a type as argument.

**22. Which keyword can be used in template?**

a) class

b) typename

c) both class & typename

d) function

Answer: c

**23. What is the validity of template parameters?**

a) inside that block only

b) inside the class

c) whole program

d) any of the mentioned

View Answer

**Answer: a**

What do I mean by this? You will know in next example.

**24.What is the output of this program?**

#include <iostream>

using namespace std;

template <class T, int N>

class mysequence

{

T memblock [N];

public:

void setmember (int x, T value);

T getmember (int x);

};

template <class T, int N>

void mysequence<T,N> :: setmember (int x, T value)

{

memblock[x] = value;

}

template <class T, int N>

T mysequence<T,N> :: getmember (int x)

{

return memblock[x];

}

int main ()

{

mysequence <int, 5> myints;

mysequence <double, 5> myfloats;

myints.setmember (0, 100);

myfloats.setmember (3, 3.1416);

cout << myints.getmember(0) << '\n';

cout << myfloats.getmember(3) << '\n';

return 0;

}

a) 100

b) 3.1416

c) 100

3.1416

d) none of the mentioned

**Answer: c**

Explanation: In this program, We are printing the integer in the first function and float in the second function.

Now, let’s explain that fact that a template parameter’s validity is inside the block.

#include <iostream>

using namespace std;

template <class T, int N>

class mysequence

{

T memblock [N];

public:

void setmember (int x, T value);

T getmember (int x);

};

void mysequence<T,N> :: setmember (int x, T value)

{

memblock[x] = value;

}

T mysequence<T,N> :: getmember (int x)

{

return memblock[x];

}

int main ()

{

mysequence <int, 5> myints;

mysequence <double, 5> myfloats;

myints.setmember (0, 100);

myfloats.setmember (3, 3.1416);

cout << myints.getmember(0) << '\n';

cout << myfloats.getmember(3) << '\n';

return 0;

}

It will give you compilation error.

**template1.cpp:11: error: ‘T’ was not declared in this scope**

**template1.cpp:11: error: ‘N’ was not declared in this scope**

**template1.cpp:11: error: template argument 1 is invalid**

**template1.cpp:11: error: template argument 2 is invalid**

**template1.cpp:11: error: ‘T’ has not been declared**

**template1.cpp: In function ‘void setmember(int, int)’:**

**template1.cpp:13: error: ‘memblock’ was not declared in this scope**

**template1.cpp: At global scope:**

**template1.cpp:15: error: ‘T’ does not name a type**

Because, the template types defined by

template <class T, int N>

Was only valid for class declaration block.

**25. What is the output of this program?**

#include <iostream>

using namespace std;

template <class T>

T max (T& a, T& b)

{

return (a>b?a:b);

}

int main ()

{

int i = 5, j = 6, k;

long l = 10, m = 5, n;

k = max(i, j);

n = max(l, m);

cout << k << endl;

cout << n << endl;

return 0;

}

a) 6

b) 6

10

c) 5

10

d) 6

5

**Answer: b**

Explanation: In this program, We are using the ternary operator on the template function.

**26. What is the output of this program?**

#include <iostream>

using namespace std;

template <class type>

class Test

{

public:

Test()

{

};

~Test()

{

};

type Funct1(type Var1)

{

return Var1;

}

type Funct2(type Var2)

{

return Var2;

}

};

int main()

{

Test<int> Var1;

Test<double> Var2;

cout << Var1.Funct1(200);

cout << Var2.Funct2(3.123);

return 0;

}

a) 100

b) 200

c) 3.123

d) 2003.123

View Answer

**Answer: d**

Explanation: In this program, We are passing the value and returning it from template.

**27. What is the output of this program?**

#include <iostream>

using namespace std;

template <typename T, int count>

void loopIt(T x)

{

T val[count];

for(int ii = 0; ii < count; ii++)

{

val[ii] = x++;

cout << val[ii] << endl;

}

};

int main()

{

float xx = 2.1;

loopIt<float, 3>(xx);

}

a) 2.1

b) 3.1

c) 4.1

d) 2.1

3.1

4.1

Answer: d

Explanation: In this program, We are using the non-type template parameter to increment the value in the function template.

**28. Why we use :: template-template parameter?**

a) binding

b) rebinding

c) both binding & rebinding

d) none of the mentioned

**Answer: c**

Explanation: It is used to adapt a policy into binary ones.

**29. Which parameter is legal for non-type template?**

a) pointer to member

b) object

c) class

d) none of the mentioned

**Answer: a**

Explanation: The following are legal for non-type template parameters:integral or enumeration type, Pointer to object or pointer to function, Reference to object or reference to function, Pointer to member.

**30. Which of the things does not require instantiation?**

a) functions

b) non virtual member function

c) member class

d) all of the mentioned

**Answer: d**

Explanation: The compiler does not generate definitions for functions, non virtual member functions, class or member class because it does not require instantiation.

**31.What is meant by template specialization?**

a) It will have certain data types to be fixed

b) It will make certain data types to be dynamic

c) Certain data types are invalid

d) None of the mentioned

**Answer: a**

Explanation: In the template specialization, it will make the template to be specific for some data types.

**32. Which is similar to template specialization?**

a) template

b) function overloading

c) function template overloading

d) none of the mentioned

**Answer: c**

**33. Which is called on allocating the memory for array of objects?**

a) destructor

b) constructor

c) method

d) none of the mentioned

**Answer: b**

Explanation: When you allocate memory for an array of objects, the default constructor must be called to construct each object. If no default constructor exists, you’re stuck needing a list of pointers to objects.

**34. What is the output of this program?**

#include <iostream>

using namespace std;

template <class T>

inline T square(T x)

{

T result;

result = x \* x;

return result;

};

template <>

string square<string>(string ss)

{

return (ss+ss);

};

int main()

{

int i = 2, ii;

string ww("A");

ii = square<int>(i);

cout << i << ": " << ii;

cout << square<string>(ww) << ":" << endl;

}

a) 2:4AA

b) 2:4

c) AA

d) 2:4A

**Answer: a**

Explanation: Template specialization is used when a different and specific implementation is to be used for a specific data type. In this program, We are using integer and character.

**35. What is the output of this program?**

#include <iostream>

using namespace std;

template <typename T = float, int count = 3>

T multIt(T x)

{

for(int ii = 0; ii < count; ii++)

{

x = x \* x;

}

return x;

};

int main()

{

float xx = 2.1;

cout << xx << ": " << multIt<>(xx) << endl;

}

a) 2.1

b) 378.228

c) 2.1: 378.228

d) None of the mentioned

**Answer) c)2.1: 378.228**

**template <typename T = float, int count = 3>**

Now, here we are using a template argument feature named default template arguments.

Now, you have to compile like: c++ -std=c++0x template2.cpp

Because,

default template arguments may not be used in function templates without -std=c++0x or -std=gnu++0x

**36. What is the output of this program?**

#include <iostream>

using namespace std;

template <class T>

class XYZ

{

public:

void putPri();

static T ipub;

private:

static T ipri;

};

template <class T>

void XYZ<T>::putPri()

{

cout << ipri++ << endl;

}

template <class T> T XYZ<T>::ipub = 1;

template <class T> T XYZ<T>::ipri = 1.2;

int main()

{

XYZ<int> a;

XYZ<float> b;

a.putPri();

cout << a.ipub << endl;

b.putPri();

}

a) 1

b) 1.2

c) 1

1.2

d) 1

1

1.2

**37. What is the output of this program?**

#include <iostream>

#include <string>

#include <cstring>

using namespace std;

template <class type>

type MyMax(const type Var1, const type Var2)

{

cout << "no specialization";

return Var1 < Var2 ? Var2 : Var1;

}

template <>

const char \*MyMax(const char \*Var1, const char \*Var2)

{

return (strcmp(Var1, Var2)<0) ? Var2 : Var1;

}

int main()

{

string Str1 = "class", Str2 = "template";

const char \*Var3 = "class";

const char \*Var4 = "template";

const char \*q = MyMax(Var3, Var4);

cout << q << endl;

return 0;

}

a) template

b) class

c) no specialization

d) none of the mentioned

**38. What is the output of this program?**

#include <iostream>

using namespace std;

template<class T = float, int i = 5> class A

{

public:

A();

int value;

};

template<> class A<>

{

public: A();

};

template<> class A<double, 10>

{

public: A();

};

template<class T, int i> A<T, i>::A() : value(i)

{

cout << value;

}

A<>::A()

{

cout << "default";

}

A<double, 10>::A()

{

cout << "10" << endl;

}

int main()

{

A<int, 6> x;

A<> y;

A<double, 10> z;

}

a) 6

b) 10

c) 6default10

d) None of the mentioned

**39. How many types of specialization are there in c++?**

a) 1

b) 2

c) 3

d) 4

**Answer: b**

Explanation: There are two types specialization. They are full specialization and partial specialization.

**Full Specialization example:**

template<class T>

class Array { /\*...\*/ };

template<class T>

void sort(Array<T>& v); // primary template

template<>

void sort(Array<int>&); // specialization for T = int

**//Now, the last function where sort is specialized for only type int and it is for full specialization.**

**Partial specialization example:**

template<class T1, class T2, int I>class A {}; // primary template

 template<class T, int I>class A<T, T\*, I> {}; // #1: partial specialization where T2 is a pointer to T1

 template<class T, class T2, int I>class A<T\*, T2, I> {}; // #2: partial specialization where T1 is a pointer

 template<class T>class A<int, T\*, 5> {}; // #3: partial specialization where T1 is int, I is 5,

// and T2 is a pointer

 template<class X, class T, int I>class A<X, T\*, I> {}; // #4: partial specialization where T2 is a pointer

**40. What is other name of full specialization?**

a) explicit specialization

b) implicit specialization

c) function overloading template

d) none of the mentioned

**Answer: a**

template<class T>

class Array { /\*...\*/ };

template<class T>

void sort(Array<T>& v); // primary template

template<>

void sort(Array<int>&); // specialization for T = int

**//Now, the last function where sort is specialized for only type int and it is for full specialization.**

**(Now, note the following thing about specialization:**

How does template specialization work?

When we write any template based function or class, compiler creates a copy of that function/class whenever compiler sees that being used for a new data type or new set of data types(in case of multiple template arguments).

If a specialized version is present, compiler first checks with the specialized version and then the main template. Compiler first checks with the most specialized version by matching the passed parameter with the data type(s) specified in a specialized version.

)

**41. Which is dependant on template parameter?**

a) base class

b) abstract class

c) method

d) none of the mentioned

**Answer: a**

Explanation: None

**42. Which value is placed in the base class?**

a) derived values

b) default type values

c) both default type & derived values

d) none of the mentioned

**Answer: b**

Explanation: We can place the default type values in a base class and overriding some of them through derivation.

**43. How many bits of memory needed for internal representation of class?**

a) 1

b) 2

c) 4

d) no memory needed

**Answer: d**

Explanation: classes that contain only type members, nonvirtual function members, and static data members do not require memory at run time.

**44. What is the output of this program?**

#include <iostream>

using namespace std;

template<typename T>class clsTemplate

{

public:

T value;

clsTemplate(T i)

{

this->value = i;

}

void test()

{

cout << value << endl;

}

};

class clsChild : public clsTemplate<char>

{

public:

clsChild(): clsTemplate<char>( 0 )

{

}

clsChild(char c): clsTemplate<char>( c )

{

}

void test2()

{

test();

}

};

int main()

{

clsTemplate <int> a( 42 );

clsChild b( 'A' );

a.test();

b.test();

return 0;

}

a) 42

b) A

c) 42

A

d) A

42

**Answer: c**

Explanation: In this program, We are passing the values by using the template inheritance and printing it.

42

A

**45. What is the output of this program?**

#include <iostream>

using namespace std;

template <class T>

class A

{

public:

A(int a): x(a) {}

protected:

int x;

};

template <class T>

class B: public A<char>

{

public:

B(): A<char>::A(100)

{

cout << x \* 2 << endl;

}

};

int main()

{

B<char> test;

return 0;

}

a) 100

b) 200

c) error

d) runtime error

**Answer)Explanation:** In this program, We are passing the values and manipulating it by using the template inheritance.

**46. What is the output of this program?**

#include <iostream>

using namespace std;

template <class type>

class Test

{

public:

Test();

~Test();

type Data(type);

};

template <class type>

type Test<type>::Data(type Var0)

{

return Var0;

}

template <class type>

Test<type>::Test()

{

}

template <class type>

Test<type>::~Test()

{

}

int main(void)

{

Test<char> Var3;

cout << Var3.Data('K') << endl;

return 0;

}

a) k

b) l

c) error

d) runtime error

**Answer) Answer: a**

Explanation: In this program, We are passing the values and printing it by using template inheritance.

Explanation: In this program, We are printing the order of execution of constructor and destructor in the class.

Note: How the class is defined.

**47. How many kinds of entities are directly parameterized in c++?**

a) 1

b) 2

c) 3

d) 4

View Answer

Answer: c

Explanation: C++ allows us to parameterize directly three kinds of entities through templates: types, constants, and templates.

**48. How many kinds of parameters are there in C++?**

a) 1

b) 2

c) 3

d) None of the mentioned

**Answer: c**

Explanation: There are three kinds of parameters are there in C++. They are type, non-type, template.