
FACE

Templates

Type-independent patterns that can work with multiple data types.

- Generic programming
- Code reusable

Function Templates

These define logic behind the algorithms that work for multiple data types.

Class Templates

These define generic class patterns into which specific data types can be plugged in to produce new classes.

FACE

Templates

- Can create a single function or a class to work with different data types using templates.

- Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs

FACE

Function Template

- Works in a similar to a normal function, with one key difference
- A single function template can work with different data types at once
- But a single normal function can only work with one set of data types.

FACE

Function and Function template

C++ routines work on specific types. We often need to write different routines to perform the same operation on different data types.

```
int maximum(int a, int b, int c)
 {
      int max = a;
      if (b > max) max = b;
      if (c > max) max = c;
      return max;
}
```

FACE

Function and Function template

```
float maximum(float a, float b,
float c)
 {
      float max = a;
      if (b > max) max = b;
      if (c > max) max = c;
      return max;
}
```

```
double maximum(double a, double
b, double c)
 {
      double max = a;
      if (b > max) max = b;
      if (c > max) max = c;
      return max;
}
```

- The logic is exactly the same, but the data type is different.
- Function templates allow the logic to be written once and used for all data types – generic function.

FACE

Function template

Generic function to find a maximum value

```
Template <class T>
T maximum(T a, T b, T c)
 {
      T max = a;
      if (b > max) max = b;
      if (c > max) max = c;
      return max;
}
```

FACE

Function Template

- Template function itself is incomplete because the compiler will need to know the actual type to generate code.

- So template program are often placed in .h or .hpp files to be included in program that uses the function.

- C++ compiler will then generate the real function based on the use of the function template.

FACE

Function Templates Usage

After a function template is included (or defined), the function can be used by passing parameters of real types.

```
Template <class T>
T maximum(T a, T b, T c)
   …
  int i1, i2, i3;
  …
  Int m = maximum(i1, i2, i3);
```

maximum(i1, i2, i3) will invoke the template function with T==int.
The function returns a value of int type.

FACE

Function Templates Usage

Each call to maximum() on a different data type forces the compiler to generate a different function using the template.

One copy of code for many types.

```
int i1, i2, i3;
  // invoke int version of maximum
  cout << "The maximum integer value is: "
     << maximum( i1, i2, i3 );
  // demonstrate maximum with double values
  double d1, d2, d3;
  // invoke double version of maximum
  cout << "The maximum double value is: "
     << maximum( d1, d2, d3 );
```

FACE

How Template work

```
template <typename T>
T mymax(T x, T y)
{
      return (x > y) ? X : y;
}

int main()
{
      cout<<myMax<int>(3, 7)<<endl;
      cout<<myMax<char>('g', 'e')<<endl;
      return 0;
}
```

Compiler internally generates and adds below code

```
int myMax(int x, int y)
{
      return (x > y) ? X : y;
}
```

Compiler internally generates and adds below code

```
char myMax(char x, char y)
{
      return (x > y) ? X : y;
}
```

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

int sum(int x, int y)
{
	return x + y;
}
float sum(float x, float y)
{
	return x + y;
}
double sum(double x, double y)
{
	return x + y;
}
int main()
{
	cout << sum(3,5) << endl;
	cout << sum(3.0, 5.2) << endl;
	cout << sum(3.24234, 5.24144);
	return 0;
}
```

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

template <typename T>

T sum(T x, T y)
{
	return x + y;
}

int main()
{
   cout<<sum(3,5)<< endl;
   cout<<sum(3.0, 5.2)<< endl;
   cout<<sum(3.224,5.241)<<endl;
   return 0;
}
```

Template increases the readability of the code

Example

```
template< class T >
void printArray( const T *array, const int count )
{
   for ( int i = 0; i < count; i++ )
      cout << array[ i ] << " "; cout << endl;
}
```

FACE

Example

```
template< class T >
void printArray( const T *array, const int count );

char  cc[100];
int      ii[100];
double dd[100];
……
printArray(cc, 100);
printArray(ii, 100);
printArray(dd, 100);
```

FACE

Example

```
template< class T >
void printArray( const T *array, const int count );

char  cc[100];
int      ii[100];
double dd[100];
myclass xx[100];  <- user defined type can also be used.
……
printArray(cc, 100);
printArray(ii, 100);
printArray(dd, 100);
printArray(xx, 100);
```

FACE

Use of template function

Can any user defined type be used with a template function?

- Not always, only the ones that support all operations used in the function.
- E.g. if myclass does not have overloaded << operator, the printarray template function will not work.

FACE

Class template

- So far the classes that we define use fix data types.
- Sometime is useful to allow storage in a class for different data types.
- See simplelist1 (a list of int type elements) example

  What if we want to make a simple list of double type?

  - ✓ Copy paste the whole file and replace int with double
  - ✓ Make use of typedef in C++, See simplelist2.

    Still need to change one line of code for a new type.

FACE

Class template

- So far the classes that we define use fix data types.
- Sometime is useful to allow storage in a class for different data types.
- See simplelist1 (a list of int type elements) example

  What if we want to make a simple list of double type?

  - ✓ Copy paste the whole file and replace int with double
  - ✓ Make use of typedef in C++, See simplelist2.

    Still need to change one line of code for a new type.

FACE

Class template

- To make a class into a template, prefix the class definition with the syntax:
    - template< class T >
        - ✓ Here T is just a type parameter. Like a function parameter, it is a place holder.
        - ✓ When the class is instantiated, T is replaced by a real type.

FACE

Class template

- To access a member function, use the following syntax:
  - ✓ className< T >:: memberName.
  - ✓ SimpleList < T > :: SimpleList()

- Using the class template:
  - ✓ ClassName<real type> variable;
  - ✓ SimpleList < int > list1;

FACE

Syntax

```
template <class T>
class className
{
        ……….
        public:
            T var;
            T someOperation(T arg);
        ……….
};
```

FACE

```
1  #include <iostream>
2  using namespace std;
3
4  template <typename T>
5  class Array
6  {
7  private:
8        T *ptr;
9        int size;
10 public:
11       Array(T arr[], int s);
12       void print();
13 };
14
15 template <typename T>
16 Array<T>::Array(T arr[], int s)
17 {
18       ptr = new T[s];
19       size = s;
20       for(int i = 0; i < size; i++)
21             ptr[i] = arr[i];
22 }
23 template <typename T>
24 void Array<T>::print()
25 {
26       for(int i = 0; i < size; i++)
27             cout<<" "<<*(ptr + i);
28       cout<<endl;
29 }
30
31 int main()
32 {
33       int arr[5] = {1, 2, 3, 4, 5};
34       Array<int> a(arr, 5);
35       a.print();
36       return 0;
37 }
38
39
40
41
```

Question 1

Which are done by compiler for templates?

A) type-safe

B) portability

C) code elimination

D) all of the mentioned

Question 2

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

Question 3

Same template is used to generate many different instances, this can be done by

A) Functions

B) Template parameters

C) Operators

D) None of them

Question 4

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;
}
```

FACE

Question 4

A) 6
12

B) 2
8

C) 8
8

D) Compiler Error: There can not be more than one template arguments.

Question 5

Which parameter is legal for non-type template?

A) pointer to member

B) object

C) class

D) none of the mentioned

Question 6

Templates are abstract recipe for producing a concrete code, and it is used for

A) Producing functions

B) Producing classes

C) Nothing

D) Both A and B

Question 7

What is the syntax of class template?

A) template <paramaters> class declaration

B) Template <paramaters> class declaration

C) temp <paramaters> class declaration

D) temp <paramaters> class declaration

Question 8

What will be the output of the following C++ code?

```
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;
template<class T>
class A
{
   public:
      A(){
            cout<<"Created\n";
      }
```

```
~A(){
      cout<<"Destroyed\n";
      }
};

int main(int argc, char const
*argv[])
{
      A a;
      return 0;
}
```

Question 8

A) Created
Destroyed

B) Destroyed
Created

C) Compile-time error

D) Run-time error

Question 9

What will be the output of the following C++ code?

```
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;
template<class T>
class A
{
    public:
      A(){
            cout<<"Focus\n";
      }
```

```
~A(){
            cout<<"Academy\n";
      }
};

int main(int argc, char const
*argv[])
{
      A <int>a;
      return 0;
}
```

Question 9

A) Focus
Academy

B) Academy
Focus

C) Compile-time error

D) Run-time error

Question 10

What will be the output of the following C++ code?

```
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;
template<class T>
class A
{
    public:
      T func(T a, T b){
            return a/b;
      }
};
```

```
int main(int argc, char const
*argv[])
{
      A <int>a1;
      cout<<a1.func(3,2)<<endl;
      cout<<a1.func(3.0,2.0)<<en
dl;
      return 0;
}
```

Question 10

A) 1
1

B) 1
1.5

C) 1.5
1

D) 1.5
1.5

FACE

Question 11

What is the validity of template parameters?

A) inside that block only

B) inside the class

C) whole program

D) inside the main class

Question 12

What will be the output of the following C++ code?

```
#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;
    }
```

Question 12

A) 6

B) 6
10

C) 5
10

D) 5

FACE

Question 13

Why we use :: template-template parameter?

A) binding

B) rebinding

C) both binding & rebinding

D) reusing

Question 14

What will be the output of the following C++ code?

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

Question 14

A) 10

B) 20

C) Compiler Error

D) No Output

Question 15

What will be the output of the following C++ code?

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

template<int n> struct funStruct
{
    static const int val =
2*funStruct<n-1>::val;
};

template<> struct funStruct<0>
{
static const int val = 1 ;
};

int main()
{
    cout << funStruct<10>::val
<< endl;
    return 0;
}
```

Question 15

A) Compiler Error

B) 1024

C) 2

D) 1

Question 16

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) All the above

Question 17

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

Question 18

What will be the output of the following C++ code?

```
#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;
}
```

Question 18

A) 16
24

B) 8
16

C) 20
28

D) Compiler Error: template parameters cannot have default values.

Question 19

What will be the output of the following C++ code?

```
#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);
}
```

Question 19

A) 20

B) Called 20

C) Compiler Error

D) Nothing will be printed

Question 20

Which of the following is used for generic programming?

A) Virtual functions

B) Modules

C) Templates

D) Abstract Classes

THANK YOU

FACE