Started on	Wednesday, 22 March 2023, 10:36 PM
State	Finished
Completed on	Wednesday, 22 March 2023, 10:53 PM
Time taken	17 mins 4 secs

Question **1**

Correct

Marked out of 3.00

Which of the following are motivations for C++ to introduce function templates?

Select one or more:

- [x] Because it is inconvenient for programmer to maintain many similar "versions" of a functions
- [ ] Because function templates make code look smarter
- [ ] Because function overloading cannot figure out which function to call
- [x] Because we want to define a single function that can reuse the same algorithm for different types

Question **2**

Correct

Marked out of 5.00

Identify C++ keywords in the following function template definition:

```
template <typename T>
T cube(T value) {
  return value*value*value;
}
```

`template`	Yes - this is a keyword ✔
`value`	No - this is not a keyword ✔
`return`	Yes - this is a keyword ✔
`cube`	No - this is not a keyword ✔
`T`	No - this is not a keyword ✔
`typename`	Yes - this is a keyword ✔

Question **3**

Correct

Marked out of 2.00

How many times is function template code compiled?

Select one or more:

- [x] Only once if the function template is never instantiated: to check for syntax errors in the definition of the function template
- [ ] Generic programming uses run-time polymorphism and therefore the code doesn't have to be compiled
- [x] Twice if the function template is instantiated: once to check for syntax errors in the definition of the function template itself and the second time to check for valid function calls in the instantiated function
- [ ] As many times as the function template is instantiated: if a function template is instantiated 5 times, the function template is compiled 5 times

Question **4**

Correct

Marked out of 4.00

Which of the following declaration statements declares a function template with a parameter of type *reference to an array of arbitrary size*?

Select one or more:

- `template <typename T, size_t N> void fun(T (&arr[N]));`
- `template <typename T> void fun(T (&arr)[]);`
- `template <typename T> void fun(T& arr);`
- `template <typename T, size_t N> void fun(T (&arr)[N]);` ✔
- `template <typename T, size_t N> void fun(T &arr[N]);`
- `template <typename T> void fun(T* arr);`
- `template <typename T, size_t N> void fun(T[N] arr);`

Question **5**

Correct

Marked out of 5.00

Assuming all necessary standard library headers are included, use the following code fragment:

```
template <typename T1, typename T2>
void function(T1, T2) { }

// in function main ...
std::string str {};
int x{};
double y{};

function(str, "");
function(3.5, x);
function(nullptr, &x);
function(y, 7);
function(&x, &x);
```

to select the functions that are instantiated.

Select one or more:

- `function<string, const char*>` ✔
- `function<std::nullptr_t, int*>` ✔
- `function<std::string, std::string>`
- `function<double, double>`
- `function<double, int>` ✔
- `function<float, float>`
- `function<void*, int*>`
- `function<int*, int*>` ✔
- `function<float, int>`

Question **6**

Correct

Marked out of 10.00

Use the following definition of function template **Max** to answer the subsequent questions.

```
template <typename T1, typename T2, typename T3>
T3 Max(T1 lhs, T2 rhs) {
  return lhs > rhs ? lhs : rhs;
}
```

The number of template type parameters in function template **Max** is:	3 ✔
The number of function parameters in function template **Max** is:	2 ✔
The number of template type arguments deduced in expression **Max(10,10.1)** is:	2 ✔
Will expression **Max(10,10.1)** compile?	The expression will not compile ✔
Will expression **Max<double>(10,10.1)** compile?	The expression will not compile ✔
Will expression **Max<double,double>(10,10.1)** compile?	The expression will not compile ✔
Will expression **Max<double,int>(10,10.1)** compile?	The expression will not compile ✔
Will expression **Max<double,double,int>(10,10.1)** compile?	The expression will compile ✔
Will expression **Max<int,int,int>(10,10.1)** compile?	The expression will compile ✔
Will expression **Max<>(10,10.1)** compile?	The expression will not compile ✔

Question **7**

Correct

Marked out of 10.00

Use the following definition of function template **Max** to answer the subsequent questions.

```
template <typename T1, typename T2, typename T3 = double>
T3 Max(T1 lhs, T2 rhs) {
  return lhs > rhs ? lhs : rhs;
}
```

The number of template type parameters in function template **Max** is:	3 ✔
The number of function parameters in function template **Max** is:	2 ✔
The number of template type arguments deduced in expression **Max(10,10.1)** is:	2 ✔
Will expression **Max(30.1,10.1)** compile?	The expression will compile ✔
Will expression **Max<double>(30.1,10.1)** compile?	The expression will compile ✔
Will expression **Max<double,double>(30.f,40.f)** compile?	The expression will compile ✔
Will expression **Max<double,int>(10,10.1)** compile?	The expression will compile ✔
Will expression **Max<double,double,int>(10,10.1)** compile?	The expression will compile ✔
Will expression **Max<int,int,int>(30.2,10.1)** compile?	The expression will compile ✔
Will expression **Max<>(30.1f,10.2)** compile?	The expression will compile ✔

Question **8**

Correct

Marked out of 12.00

Use the following function template declaration and variable definitions to deduce the template type parameter **T** for each of the subsequent expressions involving calls to function **foo**. If template type deduction fails, choose NC as your answer.

```
template<typename T>
void foo(T);

int       x{100};
const int cx{x};
int       &rx{x};
int const &crx{x};
int       *pi{&x};
int       a[10]{10};
int const ca[10]{10,20};
int       b[2][10]{};
```

foo(rx)	int	✔
foo(b)	int (*)[10]	✔
foo(crx)	int	✔
foo(ca)	int const*	✔
foo(pi)	int*	✔
foo(&x)	int*	✔
foo(x)	int	✔
foo(&pi)	int**	✔
foo(a)	int*	✔
foo(&cx)	int const*	✔
foo(cx)	int	✔
foo(5)	int	✔

Question **9**

Correct

Marked out of 12.00

Use the following function template declaration and variable definitions to deduce the template type parameter **T** for each of the subsequent expressions involving calls to function **foo**. If template type deduction fails, choose NC as your answer.

```
template<typename T>
void foo(T&);

int       x{100};
const int cx{x};
int       &rx{x};
int const &crx{x};
int       *pi{&x};
int       a[10]{10};
int const ca[10]{10,20};
int       b[2][10]{};
```

foo(x)	int	✔
foo(&cx)	NC	✔
foo(rx)	int	✔
foo(cx)	int const	✔
foo(a)	int [10]	✔
foo(crx)	int const	✔
foo(ca)	int const [10]	✔
foo(&pi)	NC	✔
foo(&x)	NC	✔
foo(pi)	int*	✔
foo(5)	NC	✔
foo(b)	int [2][10]	✔

Question **10**

Correct

Marked out of 5.00

Define a function template **xchange** that exchanges the two values passed as arguments. Your definition must satisfy the following use cases:

```
bool b1{true}, b2{false};
xchange(b1, b2);
// b1 is now false and b2 is true

std::string s1{"Singapore"}, s2{"Seattle"};
xchange(s1, s2);
// s1 is now "Seattle" and s2 is "Singapore"

std::pair<hlp2::Str, int> ns1{"Funky Kong", 7}, ns2{"Golden Peach", 42};
xchange(ns1, ns2);
// ns1 is now {"Golden Peach", 42} and ns2 is {"Funky Kong", 7}

std::vector<int> v1{1,2,3,4,5}, v2{-5,-4,-3,-2,-1,0,1,2,3,4,5};
xchange(v1, v2);
// v1 is now {-5,-4,-3,-2,-1,0,1,2,3,4,5} and v2 is {1,2,3,4,5}
```

No standard library headers are required and therefore none are included. Adding standard library headers will prevent your code from compiling!!!

```
// Standard library headers are not necessary and therefore none are included.
// Neither can you include any other header files!!!
// Define function template xchange ...
template<typename T1, typename T2>
void xchange(T1& t1, T2& t2) {
 T1 temp = t1;
 t1 = t2;
 t2 = temp;
}
```

Question **11**

Correct

Marked out of 5.00

Define a function template **foo** that satisfies the following use cases:

```
std::cout << foo<5>(10) << "\n";       // writes 50 to output stream
std::cout << foo<-1>(5) << "\n";       // writes -5 to output stream
std::cout << foo<4>(-10.4) << "\n";    // writes -41.6 to output stream
std::cout << foo<-4>(-11.23f) << "\n"; // writes 44.92 to output stream
unsigned short ush{3};
std::cout << foo<4>(ush) << "\n";      // writes 12 to output stream
long lo{-3};
std::cout << foo<-3>(lo) << "\n";      // writes 9 to output stream
std::cout << foo<-6>(3.14L) << "\n";   // writes -18.84 to output stream
std::cout << foo<3>(std::string("a")) << "\n"; // compile-time error
```

No standard library headers are required and therefore none are included. Adding standard library headers will prevent your code from compiling!!!

```
// Standard library headers are not necessary and therefore none are included.
// Neither can you include any other header files!!!
// Define function template foo that satisfies the use cases ...
template <int N, typename T>
T foo(T value) {
 if constexpr (N == 0) {
 return 0;
 } else if constexpr (N > 0) {
 return N * value;
 } else {
 return N * (value);
 }
}
```

Question **12**

Correct

Marked out of 5.00

Define a function template **Compare** to compare two values [of the same type], and indicate whether the first is less than, equal to, or greater than the second. Your definition must not perform any copies and must satisfy the following use cases:

```
std::cout << Compare(7, 42) << "\n";       // prints -1 to output stream
std::cout << Compare(-7.1, -42.2) << "\n"; // prints 1 to output stream
std::cout << Compare(42.f, 42.f) << "\n";  // prints 0 to output stream

std::string const sin{"Singapore"}, sea{"Seattle"};
std::cout << Compare(sin, sea) << "\n";   // prints 1 to output stream

hlp2::Str const a{"A"}, z{"Z"};
std::cout << Compare(a, z) << "\n";       // prints -1 to output stream

std::vector<int> const v1{1, 2, 3, 4}, v2{5, 6, 7};
std::cout << Compare(v1, v2) << "\n";    // prints -1 to output stream

std::forward_list<std::string> const f1{"1", "2", "3"}, f2{"1", "2", "3"};
std::cout << Compare(f1, f2) << "\n";    // prints 0 to output stream
```

No standard library headers are required and therefore none are included. Adding standard library headers will prevent your code from compiling!!!

```
// Standard library headers are not required and therefore none is included.
// Adding standard library header files will prevent your code from compiling!!!
// Define function template Compare that satisfies the use cases ...
template <typename T>
int Compare(T const &a, T const &b) {
 if (a < b)
 return -1;
 else if (b < a)
 return 1;
 else
 return 0;
}
```

Question **13**

Correct

Marked out of 5.00

Define a function template **Length** that returns the length of a raw array. Your definition must not perform any copies and must satisfy the following use cases:

```
int ai[40];
char cstr[1024];
double ad[16];
std::string astr[5];

std::cout << Length(ai) << "\n";             // prints 40 to output stream
std::cout << Length(cstr) << "\n";           // prints 1024 to output stream
std::cout << Length(ad) << "\n";             // prints 16 to output stream
std::cout << Length("Hello World") << "\n"; // prints 12 to output stream
std::cout << Length(astr) << "\n";           // prints 5 to output stream
```

Standard library header **<cstddef>** [for **size_t**] is included. Adding other standard library headers will prevent your code from compiling!!!

```
// Standard library header <cstddef> is included.
// Adding other standard library header files will prevent your code from compiling!!!
// Define function template Length that satisfies the use cases ...
template <typename T, std::size_t N>
constexpr std::size_t Length(T(&)[N]) {
 return N;
}
template <std::size_t N>
constexpr std::size_t Length(const char(&)[N]) {
 return N;
}
```

Question **14**

Correct

Marked out of 2.00

Will the following code fragment compile? Assume all necessary standard library headers are included.

```
template <typename T> // declaration of class template wrapper
class wrapper;

void use_wrapper(wrapper<int> *ptr); // declaration of function use_wrapper

int main() {
  wrapper<int> a(42); // define a concrete wrapper
  use_wrapper(&a);    // use the concrete wrapper
}

template <typename T>
class wrapper {
  // assume member functions including a member function get
  // are defined here ...
};

// definition of function use_wrapper
void use_wrapper(wrapper<int> *ptr) {
  std::cout << ptr->get() << '\n';
}
```

Select one:

- True
- False ✔

Question **15**

Correct

Marked out of 5.00

Define a class template **C<T>** that defines a member function **add** that takes two references of type **T** and returns their sum. Your definition must satisfy the following use cases:

```
C<int> ci;
ci.add(41.3, 22.6); // evaluates to value 63 of type int
C<double> cd;
cd.add(41.3, 22.6); // evaluates to value 63.9 of type double

std::string const s1{"enjoy"}, s2{"ment"};
C<std::string> cs;
cs.add(s1, s2); // evaluates to std::string value encapsulating C-string "enjoyment"

std::complex<double> const c1{1.1,2.2}, c2{3.3,4.4};
C<std::complex<double>> ccd;
ccd.add(c1, c2); // evaluates to value (4.4,6.6) of type std::complex<double>
```

No standard library headers are required and therefore none are included. Adding standard library headers will prevent your code from compiling!!!

```
// Standard library headers are not necessary and therefore none are included.
// Neither can you include any other header files!!!
// Define class template C<T> ...
template <typename T>
class C{
    public:
    T add(const T &a, const T &b){
        return a + b;
    }
};
```

Question **16**

Correct

Marked out of 5.00

Define a class **C** that defines a member function template **add** that takes two references and returns the sum of the two parameters. Your definition must satisfy the following use cases:

```
C c;
c.add(41.3, 22.5);      // evaluates to value 63.8 of type double
c.add(41, 22);          // evaluates to value 63 of type int
c.add<int>(41.3, 22.5); // evaluates to value 63 of type int

std::string const s1{"enjoy"}, s2{"ment"};
c.add(s1, s2); // evaluates to value of type std::string that encapsulates C-string
"enjoyment"

std::complex<double> const c1{1.1,2.2}, c2{3.3,4.4};
c.add(c1, c2); // evaluates to value of type std::complex with value (4.4, 6.6)

std::vector<int> vi1{1,2,3}, vi2{11,22,33};
c.add(v1, v2); // compile-time error because std::vector<T> doesn't declare operator+
```

No standard library headers are required and therefore none are included. Adding standard library headers will prevent your code from compiling!!!

```
2  // Standard library headers are not necessary and therefore none are included.
3  // Neither can you include any other header files!!!
4  // Define class C ...
5  class C{
6      public:
7      template <typename T>
8      T add(const T &a, const T &b){
9          return a + b;
10     }
11 };
```

Question **17**

Correct

Marked out of 5.00

Define a class template **wrapper<T>** that encapsulates a value of type **T** and whose small interface is specified by the following use cases:

```
wrapper<double> a{42.1};
std::cout << a.get();     // prints 42.1 to output stream
std::cout << a.as<int>(); // prints 42 to output stream

wrapper<int> const b{42};
int ib = b.get();           // ib is initialized to 42
double db = b.as<double>(); // db is initialized to 42.0
```

No standard library headers are required and therefore none are included. Adding standard library headers will prevent your code from compiling!!!

```
2  // Standard library headers are not necessary and therefore none are included.
3  // Neither can you include any other header files!!!
4  // Define class template wrapper ...
5  template <typename T>
6  class wrapper {
7  public:
8      explicit wrapper(T const& value) : value_(value) {}
9
10     T const& get() const {
11         return value_;
12     }
13
14     template <typename U>
15     U as() const {
```

◄ Assignment 8: matrix class template using DRY principle

Jump to...

Lab: Doubly Linked List - Real-world Use ►