管理域

#A. Unit Test: STL and Templates

✓ Hydro ✗ ✓ zsc2003

客观题

NOTE:

该比赛已结束，您无法在比赛模式下递交该题目。您可以点击"在题库中打开"以普通模式查看和递交本题。

Unit Test: STL and Templates

Answer the following questions according to the C++17 standard.

For all the questions, suppose we have the following `using` declarations.

```
using std::vector;
using std::list;
using std::array;
using std::deque;
using std::forward_list;
using std::string;
```

Part 1: Basics

1. Suppose `a` is of type `C<T>`, where `C` $\in$ { `vector` , `list` , `forward_list` , `deque` } and `T` is some type that makes the type `C<T>` well-formed. Let `x` be an object of type `T`. For each of the following blanks, write the simplest solution. It may be either an expression or a statement.

(a) Suppose `C` $\in$ { `vector` , `list` , `deque` }. How do you append a copy of `x` to the end of `a`?

(b) Suppose `C` $\in$ { `vector` , `list` , `deque` }. How do you append `x` to the end of `a` through a **move**?

□ A. vector

□ B. array

□ C. deque

□ D. list

□ E. forward_list

(d) Suppose p, q, r are three objects such that T is constructible from them, i.e. the following code compiles and constructs an object of type T.

```
T object(p, q, r);
```

Suppose C is one of the containers on which push_front is supported. How do you **prepend** an object of type T constructed from (p, q, r) **to the beginning** of a without any copy or move of T?

(e) How do you empty a (i.e. remove all the elements)?

2. For each of the following containers, write the **iterator category** of its iterator. Answer in the form "WhatIterator".

(a) vector

(b) deque

(c) list

(d) array

(e) forward_list

3. Which of the following is/are correct?

□ A.

string and vector<char> are different.

□ B.

```
for (const auto &x : a)
  do_something(x);
```

is converted by the compiler to

```
for (std::size_t i = 0; i != a.size(); ++i) {
  const auto &x = a[i];
  do_something(x);
}
```

□ C.

Let `l` be of type `list<T>` for some type `T`. To traverse `l` using iterators, we can write

```
for (auto it = l.begin(); it < l.end(); ++it)
  do_something(*it);
```

□ D.

Let `v` be of type `vector<T>` for some type `T`. To make a copy of `v`, we **must** use a loop like this:

```
vector<T> w;
for (const auto &x : v)
  w.push_back(x);
```

□ E.

Let `v` be of type `vector<T>` for some type `T`. To move-construct a new object `w` from `v`, we **can** use a loop like this;

```
vector<T> w;
for (auto &&x : v)
  w.push_back(std::move(x));
```

4. Choose the best containers.

(a) We want to store a sequence of numbers $a_0, a_1, \cdots, a_n$, where n is determined at runtime. We need fast access of elements given their indexes.

○ A. `array`
○ B. `vector`
○ C. `list`
○ D. `set`

(b) We are writing a *Greedy Snake* game and want to store the body of a snake. The body of a snake can be represented by a sequence of tuples $(x_0, y_0, d_0), \cdots, (x_n, y_n, d_n)$, where (x_i, y_i) is the coordinate of the i-th part and $d_i \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ is the moving direction of the i-th part. The snake head is (x_0, y_0, d_0). Moving the snake one step forward consists of two operations:

■管理域ere is no food on (x^*, y^*), remove (x_n, y_n, d_n).

○ A. array

○ B. vector

○ C. deque

○ D. set

○ E. map

(c) We want to record the calls to malloc and free, as what we did in Homework 4. We need fast insertion, deletion and lookup of elements.

○ A. array

○ B. vector

○ C. list

○ D. set

(d) Given an article, we need fast query of the number of occurrences of some words.

○ A. array

○ B. list

○ C. set

○ D. map

5. For each of the following template function declarations and calls, write the deduction result for T and the type of the parameter x.

(a)

```
template <typename T>
void fun(T x);
int i = 42;
const auto &cri = i;
fun(cri);
```

T = ________ .

The type of the parameter x is ________ .

(b)

```
template <typename T>
void fun(std::vector<T> &x);
std::vector matrix(10, std::vector(10, 0.0));
fun(matrix);
```

The type of the parameter x is .

(c)

```
template <typename T>
void fun(T &&x)
std::string s;
fun(s);
```

T = .

The type of the parameter x is .

(d)

```
template <typename T>
void fun(T &&x);
std::string s, t;
fun(s + t);
```

T = .

The type of the parameter x is .

(e)

```
template <typename T>
void fun(T x);
int a[10];
fun(a);
```

T = .

The type of the parameter x is .

(f)

```
template <typename T>
void fun(T &x);
```

管理域 .

The type of the parameter `x` is .

Part 2: Dynarray 4.0

Read the code in `dynarray.hpp` and answer the following questions. For your convenience, we have marked the question numbers and the choices in the code.

Click this link to download `dynarray.hpp`. If you encounter any problems, go to Piazza resources to download it.

6. Which of the following is/are correct?

☐ A. The assignment operator is both a copy assignment operator and a move assignment operator.

☐ B. In the assignment operator, the type of the parameter `other` is `Dynarray<T>`.

☐ C. Note that the `cbegin()` function calls the `begin()` function. If `a` is of type `Dynarray<double>`, `a.cbegin()` will call the non-`const` version of the `begin()` function.

☐ D. Suppose `a` is an object of type `Dynarray<double>`. If the non-`const` version of `begin()` is not present, `a.begin()` does not compile.

7. Read the following code.

```
#include "dynarray.hpp"

struct X {
  int id;
};

std::ostream &operator<<(std::ostream &os, const X &x) {
  return os << "X(" << x.id << ')';
}

int main() {
  Dynarray<X> a(5);
  for (auto i = 0; i != 5; ++i)
    a[i].id = i * i;
  std::cout << a << std::endl;
}
```

Which of the following is/are correct?

☐ A.

Note that the `operator<` for `Dynarray<T>` relies on the `operator<` for `T`. Since `X` does not have an overload for `operator<`, `X` cannot be the element type of `Dynarray` and the code above does not

The code above compiles and prints `[0, 1, 4, 9, 16]`.

□ C.

The code above compiles and prints `[X(0), X(1), X(4), X(9), X(16)]`.

□ D.

The code above does not compile if the default constructor of `X` is deleted, i.e.

```
// In class X
X() = delete;
```

□ E.

The following code does not compile if `Y` does not have a default constructor.

```
Dynarray<Y> a;
```

8. Which of the following is/are correct?

□ A.

`Dynarray<T>::iterator` is `DynarrayIterator<T>`.

□ B.

`Dynarray<T>::const_iterator` is `DynarrayIterator<T, true>`.

□ C.

The iterator of `Dynarray<T>` is a RandomAccessIterator.

□ D.

Since `DynarrayIterator<T, C>` has a pointer member `m_current`, it should have a destructor defined as this:

```
// In class DynarrayIterator<T, C>
~DynarrayIterator() {
  delete[] m_current;
}
```

□ E.

`DynarrayIterator` is neither copyable nor movable, because it does not have any copy or move operations defined.

9. Note that the `static_assert` at the beginning of `Dynarray` ensures that `T` must be a non-`const` type. Which of the following is/are correct?

□ A.

`Dynarray<int>::iterator::value_type` is `int`, and `Dynarray<int>::const_iterator::value_type` is `const int`, because `const_iterator` is the iterator with "low-level `const`ness".

Dynarray<int>::iterator::reference is int &, and

Dynarray<int>::const_iterator::reference is const int &.

□ C.

If T is a non-const type, the indirection operator (operator*) for DynarrayIterator does not compile, because it is a const member function.

□ D.

The subscript operator of DynarrayIterator should have const and non-const overloads like this:

```
// In class DynarrayIterator<T, C>
T &operator[](difference_type n) {
  return m_current[n];
}
const T &operator[](difference_type n) const {
  return m_current[n];
}
```

10. A template function declared like this

```
template <typename T,
          typename = std::enable_if_t<some_condition>>
RetType funcName(...);
```

can only be called if the condition some_condition evaluates to true. For example, the following function foo can only be called with an integral argument.

```
template <typename T,
          typename = std::enable_if_t<std::is_integral_v<T>>>
void foo(T x) {
  std::cout << x << std::endl;
}
foo(42); // compiles
foo('a'); // compiles
foo(3.14); // Error
foo("hello"); // Error
```

All these metafunctions (std::enable_if, std::is_integral, std::is_same, ...) are defined in <type_traits> and you can find the references here.

Which of the following is/are correct?

□ A.

The constructor marked 10/A can only be used to construct a const_iterator from an iterator, which is adding the low-level constness.

□ B.

The following constructor

can be called only when the arguments are ForwardIterators.

□ C.

The constructor marked `10/C` can only be called when the arguments passed in are at least ForwardIterators.

□ D.

The following code compiles and prints `[goodbye, cs100]`. (Note that we have provided a deduction guide, marked `10/D`.)

```
std::list<std::string> ls{"goodbye", "cs100"};
Dynarray ds(ls.begin(), ls.end());
std::cout << ds << std::endl;
```

递交

101 102 103 104 105
201 202 203 204 205
3 401 402 403 404
5011 5012 5021 5022 5031
5032 5041 5042 5051 5052
5061 5062 6 7 8
9 10

编辑

文件

Homework 8

✓ 已认领

查看作业

管理域

编辑作业

导出所有代码

所有递交

帮助

Hydro zsc2003

状态
已结束

题目
3

开始时间
2023-5-23 0:00

截止时间
2023-6-13 5:00

可延期
0 小时