ITE1015 Creative Software Design

Fall 2021 Final Exam

ID	NAME

Notice

1. Please check if you received all 9 pages (5 problems) of the test material excluding the title page.
2. Write down your student ID and name on the test material and answer sheets.
3. Write down page numbers on your answer sheets.
4. Write down your answers using a black or blue ballpoint pen
5. You can use any C++ features and STL unless they are not prohibited in the problem description.
6. You must include all the necessary headers to get full credits.
7. You can leave from 1 hour after the start of the test.

Blank

1. Answer the following multiple-choice questions (10 points, 2 points for each correct answer,)

(1) Which of the following is true about exception handling in C++?

(a) There is a standard exception class in STL. (b) All exceptions are unchecked in C++, i.e., compiler doesn't check if the exceptions are caught or not. (c) In C++, a function `A()` cannot catch the exceptions thrown by its callee function `B()` (i.e., `A()` calls `B()`).

① (a) and (b)

② (a) and (c)

③ (b) and (c)

④ (a), (b), and (c)

(2) Which of the following operators cannot be overloaded?

① `*` (Pointer dereference operator)

② `&` (Address-of operator)

③ `.` (Member Access or Dot operator)

④ `()` (Function call operator)

(3) Which of the followings is/are automatically added to every class, if we do not write our own?

① Copy constructor

② Assignment operator

③ A constructor without any parameter

④ All of the above

(4) Which of the following is true about templates?

(a) Template is a feature of C++ that allows us to write one code for different data types.
(b) We can write one function that can be used for all data types including user defined types.
(c) We can write one class or struct that can be used for all data types including user defined types.
(d) Template is an example of runtime polymorphism.

① (a) and (b)
② (a), (b), and (c)
③ (a), (b), and (d)
④ (a), (b), (c), and (d)

(5) Which of the following is true about smart pointers?

(a) `std::unique_ptr` and `std::shared_ptr` are introduced since C++11
(b) You cannot copy one `std::unique_ptr` object into another
(c) If you want to store smart pointers in STL containers, you should use `std::shared_ptr`.
(d) Using `std::shared_ptr` guarantees no memory leak

① (a) and (b)
② (a), (b), and (c)
③ (a), (b), and (d)
④ (a), (b), (c), and (d)

2. Write down the expected results of the following C++ programs. If it has a compile error or a runtime error, write down "compiler error" or "runtime error" with a brief explanation as an answer, respectively. (20 points)

(1) (5 points)

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

int main()
{
    try{
       throw 10;
    }
    catch (...) {
        cout << "default exception\n";
    }
    catch (int param) {
        cout << "int exception\n";
    }

    return 0;
}
```

(2) (5 points)

```
#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;
}
```

(3) (5 points)

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

class Base
{
public:
    int fun()  { cout << "Base::fun() called"; }
    int fun(int i)  { cout << "Base::fun(int i) called"; }
};

class Derived: public Base
{
public:
    int fun() {  cout << "Derived::fun() called"; }
};

int main()
{
    Derived d;
    d.fun(5);
    return 0;
}
```

(4) (5 points)

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

class A
{
  int x;
public:
  void setX(int i) {x = i;}
  void print() { cout << x; }
};

class B:  public A
{
public:
  B()  { setX(10); }
};

class C:  public A
{
public:
  C()  { setX(20); }
};

class D: public B, public C {
};

int main()
{
    D d;
    d.print();
    return 0;
}
```

3. Write functions `sum_odd_n()`, `sum_even_n()`, and `sum_prime_n()` that take a `const std::vector<int>&` as the first parameter and an integer value `n` as the second parameter, and return the sum of first `n` odd numbers, first `n` even numbers, and first `n` prime numbers in the parameter, respectively. Your code should follow the three restrictions. (20 points)
    A. Restriction 1: You are allowed to use only `std::for_each()` and cannot use any loop (i.e., `for`, `while`, and `goto`) other than `std::for_each()`.
        i. You can find a code example that shows how to use `std::for_each()`
    B. Restriction 2: You cannot use any conditional statements (i.e., `if`, `?:`) in the body of the three functions.
    C. Restriction 3: You can freely use loops and conditional statements in the body of lambda functions.

```
#include <vector>
#include <algorithm>
#include <iostream>

int main()
{
    std::vector<int> nums{3, 4, 2, 8, 15};

    std::for_each(nums.begin(), nums.end(), [](int &n){
        if (n % 2 == 0){
            std::cout << n << " ";
        }});

    std::cout << "\n";
}
```

Execution results:

4 2 8

4. Write a C++ function (or function template) that merges two STL containers into one STL container. (10 points)

   A. Its name should be `mergeContainers()` and it takes two containers as parameters

      i. We call the first and second arguments as `container1` and `container2`, respectively

   B. It should support merge of the following STL containers: `std::vector`, `std::list`, `std::deque`, `std::set`, and `std::map`

      i. Element type of the containers can be arbitrary

   C. After `mergeContainers()` returns, `container1` should contain all the elements of the passed parameters, `container1` and `container2`

   D. If two different elements in `container1` and `container2` have the same key, you should discard the elements in `container2`

   E. The order of elements after merge does not matter

5. Write down your own vector class template called `MyVector<T>` following the description. You can assume that there is no exceptional other than the explicitly described ones (e.g., full of buffer). (40 points)

(1) Define `Container<T>` class template that presents an interface for a container. `Container<T>` should have the following member functions as pure virtual functions. (10 points)

- `unsigned int capacity()`
    A. returns the number of elements that can be held in currently allocated buffer
- `unsigned int size()`
    A. returns the number of elements
- `bool empty()`
    A. checks whether the container is empty. It returns `true` if no element is store and `false`, otherwise
- `void clear()`
    A. clears the contents
- `void insert(unsigned int pos, const T& value)`
    A. inserts a new element `value` at the position `pos`. All the elements located at `pos`, `pos+1`, ..., `pos+n` should be moved to `pos+1`, `pos+2`, ..., `pos+n+1`, respectively. If the buffer is full, you need to increase the size of the buffer.
- `void erase(unsigned int pos)`
    A. removes an element at the position pos. All the elements located at pos+1, pos+2, ..., pos+n should be moved to pos, pos+1, ..., pos+n-1, respectively

(2) Define `MyVector<T>` class template that implements a `std::vector`-like container. (20 points)

- `MyVector<T>` should inherit from `Container<T>`.

- `MyVector<T>` has a constructor with no argument, a copy constructor, a move constructor, a copy assignment constructor, a move assignment constructor, and a destructor.
- The constructor with no argument constructs an object with a buffer whose capacity is 0
- You cannot use any STL container as a buffer

(3) Implement `MyVector<T>`'s own member functions. (10 points)

- `void push_back()(const T& value)`
  A. adds an element `value` to the end. If the buffer is full, you need to increase the size of the buffer.
- `void pop_back()`
  A. removes the last element
- `T& operator[](unsigned int pos)`
  A. returns the element at the position `pos`