Answer Sheet

Name (zh or en) Hsin-Po Wang Student ID B00201054

[0] B [1] I [2] L

（反正就像這樣啦，後面的題號不想做了）

1 Classes and Structures

[0] From the unfinished homework to the right, we can infer that (A) age is a private member (B) age is a private variable (C) too_old is a public method (D) time_flies is a public method (E) `Rabbit` is the name of the class (F) all of the above are correct

```
class Rabbit {
private:
    int age;
public:
    void time_flies() { age += 1; }
    void too_old() {
        // return ???;
    }
}
```

[1] `too_old` should return (G) `age` (H) `void` (I) `age < 3` (J) `age = 3` (K) `age >> 3`

[2] In the context of classes, a getter (L) gets the job done (M) gets the address of a method (N) gets the instances of a class (O) gets the value of a (private) member (P) gets the memory location of a variable

[3] Which option is wrong about setters? (Q) It can take one `float` to set an age (R) It can take two `float`s to set a complex number (S) It can be used to set the value of a public member (T) It can be used to set the value of a private member (U) For performance reasons, it should not validate the inputs

[4] In a class, which of the following is absolutely necessary? (V) public member (W) public method (X) private method (Y) private member (Z) None of the above is absolutely necessary

[5] What is/are POD (Plain Old Data)? (A) An `int`. (B) An array `float[100]`. (C) A `struct` of an `int` and `float[100]`. (D) A `struct` of two copies of the `struct` described in the previous option (E) Anything organized in a predictable way to use a predictable amount of memory is a POD.

2 Overloading

[6] Overloading the constructor of a class allows two of the following (F) construction of multiple instances in one go (G) construction of instances by different types of data (H) construction of instances by different amount of data (I) inheritance of the same number of destructors from the parent class

[7] Which two are correct about overloading `operator+` (J) For binary `+`, the first argument is sometimes called `self` or `left` (K) For binary `+`, the second argument is sometimes called `rhs` or `other` (L) You can only overload `operator+` in a way that `a + b` is equivalent to `b + a` (M) You can only overload `operator+` for numeric types like `float` or complex numbers, not strings

[8] Why, sometimes, it is `complex operator*(complex a)` instead of `a` and `b`? (N) It is in the class definition (O) It is implementing complex conjugate (P) It is overloading the dereferencing operator

[9] I hope that `Complexity C1 = 3; cout << C1 << endl;` will print the string `O(n^3)` to the screen.

```
class Complexity {
public:
    int degree;
    Complexity(int d) : degree(d) {}
    friend std::ostream& operator<<(std::ostream& oyster, const Complexity& c1) {



        return oyster;
    }
};
```

[10] Overload addition so that `O(n^1) + O(n^2) + O(n^3)` results in `O(n^3)`.

```
Complexity operator+(const Complexity& LHS, const Complexity& RHS) {



}
```

[11] Overload multiplication.

```
Complexity operator*(const Complexity& here, const Complexity& other) {



}
```

[12] Overload `operator<<` and `operator>>` such that `O(n^2) << O(n^4)` evaluates to true but `O(n^3) << O(n^3)` evaluates to false.

```



}
```

3 Polymorphism

[13] From the code to the right, `Rabbit R1; R1.eat();` results in (Q) `health` of `R1` increased by 1 (R) `health` of `R1` increased by 2 (S) `health` of `R1` increased by 3 (T) bad things happening

[14] `Animal *A2 = &R1; A2->eat()` results in (U) `health` of `R1` increased by 1 (V) `health` of `R1` increased by 2 (W) `health` of `R1` increased by 3 (X) bad things happening

[15] `Tiger *T3 = A2; T3->eat()` results in (Y) `health` of `R1` increased by 1 (Z) `health` of `R1` increased by 2 (A) `health` of `R1` increased by 3 (B) bad things happening

[16] From the code to the right, `Tiger T4; T4.eat();` results in (C) `health` of `T4` increased by 1 (D) `health` of `T4` increased by 2 (E) `health` of `T4` increased by 3 (F) bad things happening

```
class Animal {
 public:
    int health;
    virtual void eat() {
        health += 1;
    };
};
class Rabbit : public Animal {
public:
    void eat() {
        health += 2;
    };
};
class Tiger : public Animal {
public:
    void eat(Rabbit R1) {
        R1.health = 0;
        health += 3;
    };
};
```

[17] Which of the following statement(s) about polymorphism is true? (G) Polymorphism requires all derived classes to implement the same methods. (H) Polymorphism is only applicable to classes that do not inherit any method. (I) Polymorphism allows objects of different classes to be treated as objects of a common base class. (J) Polymorphism is a feature that allows a function to be defined multiple times with different signatures.

[18] Given `Animal *A5` and `Animal *A6`, how do I know if `A5` can eat `A6`? Specify which class/method you want to modify and show me an example of how to test in an `if`.

```
//Proposed change:



//How to test:
```

4 Template

[19] I want to sort **std::pair<int, float>** co-lexicographically.

```
std::vector<std::pair<int, float>> vee = {{1, 3.14}, {2, 2.72}, {3, 1.41}};
std::sort(vee.begin(), vee.end(), [](const auto& a, const auto& b) {

    return

});
```

[20] I want to be able to add two stacks of type **std::stack<int>** such that the top of **a** is at the bottom of **b**.

```
std::stack<int> operator+(const std::stack<int>& a, const std::stack<int>& b) {









}
```

[21] I want to be able to add two stacks of type **std::stack<T1>** for general type **T1**.

```
std::stack<T1> operator+(const std::stack<T1>& a, const std::stack<T1>& b) {









}
```

[22] Select six that compile. (K)

```
int main() {
    struct abc {
        int a, b, c;
    };
    abc X1;
}
```

(L)
```
int main() {
    struct abc {
        int a, b, c;
    };
    struct abc X1;
}
```

(M)
```
int main() {
    struct abc {
        int a, b, c;
    };
    class abc X1;
}
```

(N)
```
int main() {
    class abc {
        int a, b, c;
    };
    abc X1;
}
```

(O)
```
int main() {
    class abc {
        int a, b, c;
    };
    struct abc X1;
}
```

(P)
```
int main() {
    class abc {
        int a, b, c;
    };
    class abc X1;
}
```

5 Threads

[23] A thread is (Q) a process (R) a social media (S) a physical CPU core (T) something like a process but with shared memory

[24] Which one is correct? (U) More threads always mean faster execution (V) Fewer threads always mean faster execution (W) The number of threads should always equal to the number of CPU cores (X) The number of threads should be determined by benchmarking my machine (Y) The number of threads should be determined by benchmarking the target machines

[25] Which of the following for-loop benefits the least from parallelization?

(Z)
```
for (int i = 0; i < 10000; i++) {
     A[i] = B[i] + C[i];
}
```

(A)
```
for (int i = 0; i < 10000; i++) {
     A[i] *= A[i];
}
```

(B)
```
for (int i = 0; i < 10000; i++) {
     A[i + 1] = sin(100 + A[i]);
}
```

(C)
```
for (int i = 0; i < 10000; i++) {
     A[i] *= cos(100 + A[i + 1]);
}
```

[26] Rank the following options from fast to slow. (D) Use one thread to handle `A[2*i]` and another thread to handle `A[2*i+1]` (E) Use one thread to handle `A[0..size/2]` and another thread to handle `A[size/2..size]` (F) Give every thread 1000 or so elements and give it more when it finishes

[27] By default, what happens if the main thread and a subordinate thread try to write to the same memory location? (G) Immediate segfault (H) Nothing special will happen (I) The memory is locked and subordinate thread waits thread 1 (J) The memory is locked but it is unclear which thread will wait

6 Optimization

[28] From now on, I want to write `a << 2` whenever I mean `a * 4` (K) because bit-shifting is faster than multiplication (L) because bit-shifting is slower than multiplication (M) because bit-shifting is more readable than multiplication (N) because bit-shifting is more confusing than multiplication (O) except that I should not do that

[29] Why is it important to time your programs? (P) To compare different algorithms (Q) To identify performance bottlenecks (R) To ensure it runs within acceptable limits (S) To avoid wasting time on premature optimization (T) All of the above are correct

[30] What are possible reasons that timing a program multiple times results in less and less time? (U) The CPU predicts branches better (V) Intermediate results are cached (W) The OS purges the cache used by other programs (X) The CPU is getting warmer and electrons move faster

[31] Why `for(auto i : v)` is preferred over the traditional loop? (Y) It runs significantly faster (Z) It avoids off-by-one errors (A) It avoids writing `i` three times

7 Culture

[32] C++ Core Guidelines advises this because (choose two) (B) it includes a header file if it is actually used (C) it prevents multiple inclusions of the same file (D) it includes the header file only if it has been included (E) it lets other header files to test if this one is included (F) it tests if the compiler is capable of conditional compilation

```
// file foobar.h:
#ifndef LIBRARY_FOOBAR_H
#define LIBRARY_FOOBAR_H
// ... declarations ...
#endif // LIBRARY_FOOBAR_H
```

[33] `// file foobar.h:` (G) is needed at the beginning of every header file (H) is there to hint the reader about the file name (I) remains because the developer forgot to remove comments before git-committing

[34] `// LIBRARY_FOOBAR_H` (J) is needed to close an `ifndef` (K) is there to hint the end of the include guard (L) remains because the developer forgot to remove comments before git-committing

[35] Google C++ Style Guide is against exceptions because (choose all that apply) (M) the company is made of exceptional engineers (N) catching all types of exceptions can be tedious (O) an exception can penetrate multiple layers of function calls (P) most modern languages are already equipped with exceptions so C++ does not have to

[36] The type of a lambda expression is declared as `auto` because (sort the reasons from good to bad) (Q) we could not care less (R) the type can be very complicated and distracting (S) it is faster than explicitly specifying the type (T) `auto` is the only way to declare a lambda expression (U) the type can be automatically inferred by the compiler

[37] In a function definition, local static variable is preferred over global variable (V) to avoid global warning (W) to force allocation in SSD instead of RAM (X) to avoid accident access by other function (Y) to ensure the variable is initialized only once

```
int random101(int min, int max) {
    static int seed = 12;
    seed = (seed * 12) % 101;
    return min + seed % (max - min + 1);
}
```

[38] In a function definition, local static variable is preferred over volatile variable (Z) to avoid global warning (A) to force allocation in SSD instead of RAM (B) to avoid accidental access by other functions (C) to ensure that the variable is initialized only once

[39] At the very beginning of the main function, `cout << random101(0, 10) + random101(0, 10);` results in .....

8 Correct options

are made into hyperlinks that lead to this page.

9 Lab questions - Fourier

[40] [41] [42] Write a function `CosineTransform(int n, double* A, double* B)` that computes the cosine transform of `A` of length `n`. The cosine transform is used in jpg files and is defined as

$$B[k] = \sum_{j=0}^{n-1} A[j] \cos\Big(\frac{jk\pi}{n}\Big)$$

where $k = 0, 1, \ldots, n-1$.

[43] [44] [45] Write a template function `template<typename CC> void FourierTransform(int n, CC *A, CC *B)` That implements the Fourier transform of `A` of length `n`. Here, `CC` is a class that acts like complex numbers and I will provide the following:

- `const CC PI; // 3.14...`
- `const CC II; // sqrt(-1)`
- `CC operator+(const CC& a, const CC& b)`
- `CC operator+(const CC& a, const int& b)`
- `CC operator+(const int& a, const CC& b)`
- `CC operator-(const CC& a, const CC& b)`
- `CC operator-(const CC& a, const int& b)`
- `CC operator-(const int& a, const CC& b)`
- `CC operator*(const CC& a, const CC& b)`
- `CC operator*(const CC& a, const int& b)`
- `CC operator*(const int& a, const CC& b)`
- `CC operator/(const CC& a, const CC& b)`
- `CC operator/(const CC& a, const int& b)`
- `CC operator/(const int& a, const CC& b)`
- `CC exp(CC a)`
- `CC Csqrt(int a)`
- `ostream& operator<<(ostream& beaver, const CC& c)`

The Fourier transform is used everywhere and defined as

$$B[k] = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} A[j] \exp\Big(-\frac{2\pi ijk}{n}\Big)$$

where $k = 0, 1, \ldots, n-1$.

[46] [47] [48] Write a template function `template<typename CC> void FourierTransform2D(int m, int n, CC** A, CC** B)` that computes the 2D Fourier transform of `A` of size `m x n`. The 2D Fourier transform is formally defined as

$$B[u][v] = \frac{1}{\sqrt{mn}} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} A[x][y] \exp\Big(-2\pi i \left(\frac{ux}{m} + \frac{vy}{n}\right)\Big)$$

where $u = 0, 1, \ldots, m-1$ and $v = 0, 1, \ldots, n-1$.

[49] [50] [51] (Probably too hard) Implement FFT.

10 Lab questions - Permutation

[52] [53] Implement a class `Permutation` with one member `std::vector<int> images` to store a permutation of finitely many integers. For `Permutation pmtt;` I want you to use `pmtt.images[i]` to store the image of `i` under the permutation, for `i` less than `pmtt.images.size`. I want that `pmtt(i)` returns `pmtt.images[i]` if `i` is less than `pmtt.images.size`, and returns `i` if `i` is $\geq$ `pmtt.size`.

Example testing code:

```
std::vector<int> vee = {2, 0, 3, 1};
Permutation pmtt(vee);
cout << pmtt(0) << pmtt(1) << pmtt(2) << pmtt(3) << pmtt(4); // 20314
```

You do not have to check if the input is a valid permutation. Proceed silently even if `vee[i] = vee[j]` for some `i` not equal to `j`. Proceed silently even if `vee[i]` is not between `0` and `vee.size() - 1`.

[54] [55] It is not hard to see that

```
Permutation
pee1(std::vector{1, 0}),
pee2(std::vector{1, 0, 2}),
pee3(std::vector{1, 0, 2, 3}),
pee4(std::vector{1, 0, 2, 3, 4});
```

are all the same permutation: all I want is to exchange `0` and `1`, and keeping the other numbers untouched. Implement `ostream& operator<<(ostream& sealion, const Permutation& pmtt)` to print the permutation. Implement `pmtt.reduce()` to return the shortest form of the permutation. Implement `pee == poo` to check if two permutations are the same.

Example testing code:

```
std::vector<int> vee = {1, 2, 0, 3, 4, 5, 6, 7, 8, 9};
Permutation pee(vee);
Permutation poo = pee.reduce();
cout << pee << endl; // Permutation({1,2,0,3,4,5,6,7,8,9})
cout << poo << endl; // Permutation({1,2,0})
cout << (pee == poo); // 1
```

[56] I want to be able to say `Permutation P3 = P1 * P2;` to get the composition of two permutations. Note that, because permutations are like functions, `P3(i)` would be `P1(P2(i))`, not `P2(P1(i))`.

Example testing code:

```
std::vector<int> vee = {0, 2, 1}; // exchanging 1 and 2
std::vector<int> uoo = {0, 1, 2, 3, 5, 4}; // exchanging 5 and 4
Permutation pee(vee), poo(uoo);
Permutation paa = pee * poo;
cout << paa; // Permutation({0,2,1,3,5,4})
```

[57] implement `Permutation operator~()` to return the inverse of the permutation. It is the permutation that undoes the original permutation That is, `pmtt * ~pmtt` and `~pmtt * pmtt` should be the identity permutation. The input permutation will always be a valid permutation, although it may not be the shortest form.

Example testing code:

```
std::vector<int> vee = {1, 2, 0, 3, 4, 5, 6, 7, 8, 9};
Permutation pee(vee);
Permutation poo = ~pee;
```

```
cout << pee << endl; // Permutation({1,2,0,3,4,5,6,7,8,9})
cout << poo << endl; // Permutation({2,0,1})
cout << (pee * poo) // Permutation({})
cout << (pee * poo == Permutation({})); // 1
```

[58] Implement `Permutation Exchange(int a, int b)` to return the permutation that exchanges `a` and `b` and keeps the other numbers untouched.

[59] Implement `Permutation operator^(int k)` to return the `k`-th power of the permutation.

[60] Implement `pmtt.order()` that returns the least positive integer `k` such that there exists an `n` such that `pmtt^k == pmtt^(n+k)`. See also https://en.wikipedia.org/wiki/Cycle_detection

[61] [62] [63] (Perhaps too hard) Given two integers k and n, how many permutations in S_n have order k?

11 Lab questions - Runge–Kutta

Let `CC` be a class that acts like some continuous number systems. It could be real numbers, complex numbers, matrices with real or complex entries, etc. I will provide `CC +- CC` and `CC */ int`. I will also provide `cout << CC`.

Let `FF` be a class that acts like functions from `(double, CC)` to `CC`. That is, if `F` is of type `FF`, `t` is of type `double`, and `x` is of type `CC`, then `F(t, x)` is of type `CC`.

Sanity checks: We know exact solutions to the following IVPs:

- $y(t) := 101$ satisfies $\frac{d}{dt}y(t) = 0$ and $y(0) = 101$.
- $y(t) := t^2 - t + 2$ satisfies $\frac{d}{dt}y(t) = 2t - 1$ and $y(0) = 2$.
- $y(t) := 1/(1+t)$ satisfies $\frac{d}{dt}y(t) = -1/(1+t)^2$ and $y(0) = 1$.
- $y(t) := \exp(t)$ satisfies $\frac{d}{dt}y(t) = \exp(t)$ and $y(0) = 1$.
- $y(t) := \sin(t)$ satisfies $\frac{d}{dt}y(t) = \cos(t)$ and $y(0) = 0$.
- $y(t) := \exp(-t^2)$ satisfies $\frac{d}{dt}y(t) = -2t\exp(-t^2)$ and $y(0) = 1$.

[64] [65] Define `template<typename CC, typename CC> CC EulerSolver(FF F, CC y0, int n)` to solve the IVP in y

$$\frac{d}{dt}y(t) = F(t, y(t)), \qquad y(0) = y_0$$

using the Euler method and returns $y(1)$, which is of type `CC`. The Euler method computes

$$y(t + 1/n) \leftarrow y(t) + F(t, y(t))/n$$

iteratively. The idea is to believe that the first-order Taylor expansion

$$y(t + \varepsilon) \approx y(t) + \varepsilon\frac{d}{dt}y(t) = y(t) + \varepsilon F(t, y(t))$$

is good enough.

[66] [67] Define `template<typename CC, typename CC> CC MidpointSolver(FF f, CC y0, int n)` to solve the IVP and returns $y(t)$ by using the derivative at the midpoint:

$$y\Big(t + \frac{1}{n}\Big) \leftarrow y(t) + \frac{1}{n}F\Big(t + \frac{1}{2n}, y(t) + \frac{1}{2n}F(t, y(t))\Big)$$

That is, it first uses the Euler to estimate the derivative at the midpoint and uses that to estimate the value at the end of the interval.

[68] [69] Define `template<typename CC, typename CC> CC RK4Solver(FF f, CC y0, int n)` using

$$k_1 = F(t, y(t)), \quad k_2 = F\Big(t + \frac{1}{2n}, y(t) + \frac{k_1}{2n}\Big), \quad k_3 = F\Big(t + \frac{1}{2n}, y(t) + \frac{k_2}{2n}\Big), \quad k_4 = F\Big(t + \frac{1}{n}, y(t) + \frac{k_3}{n}\Big)$$

$$y\Big(t + \frac{1}{n}\Big) \leftarrow y(t) + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6n}$$