Quiz

Question 1

In the standard C++ terminology, an abstract class has at least one pure virtual function. A pure abstract class has only abstract member functions and no data or concrete member functions.

```
class PureAbstractClass
{
public:
  virtual void AbstractMemberFunction() = 0;
  virtual void AnotherAbstractMemberFunction() = 0;
};
```

The keyword `virtual` need to be used to declare virtual function.

Question 2

The problems with the code is that:

- The `buf` attribute is public and any other program can access this data directly and can modify data.
- The operator `=` between 2 object is not implemented so that the `ex2` won't have data like `ex1`.

Solution:

- Make the `buf` is private and then create member function to`get()`, `set()` value.
- Overload the operator `=` to assign object to object.

Question 3

This code cannot work as expected, because the `Children` class only override `initialize` method belong to `Children`, the constructor of `Children` only call from `Parent`'s constructor so, the `initialize` method of `Parent` will be executed.

To solve this problem, we need to execute the `initalize()` again inside the constructor of `Children`:

```
class Children : public Parent
{
public:
    Children() : Parent() {initialize();}
    virtual void initialize() override { value = 5; }
};
```

The output of terminal now is `5`.

Question 4

The output of program is

```
test 2!
test 1!
```

The reason why is 2 `func` functions is overloaded each other, and depending on the input type, it will execute the different function with different signature.

The function with term `T &param` only allow a variable with decleration and the `param` can be modified.

On the other hand, the `T &&param` also called `rvalue reference`, a new feature in C++11, can allow refers to temporaries that are permitted to be modified after they are initialized. So this delceration can allow both `x` with type int and `20` value as input.

For that reason when input is `int x = 10` the compiler will use the `func(T &param)` and with input is only value `20` compiler will use the `func(T &&param )` to execute.

Question 5

Definition

- Static typing is a typing system, that a variable is bound to a datatype during compilation.
- Dynamic typing allow a variable to bound to data type at run time instead of compilation.

Advantages

Static typing

- It ensure data type safe, reduce the chances of runtime error while executing programs
- It's easy to debug and catch the error since the compiler knows the data type of variable
- Static typing allow compiler easier to optimze code and lead to faster execution time.

Dynamic typing

- It's flexibility because dynamic typing allow to change data type at runtime, so it's very good for adaptability case.
- It doesn't need type decleration during developing, so it help developer to code faster, finish feature faster.

Disadvantages

Static typing

- coding speed is quitely slow

Dynamic typing

- Error may appear at runtime and need to test carefully.
- The performance is not better than Static typing.

Question 6

The output of code after exection is:

```
Copy 1: [[100, 2, 300], [400, 5, 6], [7, 8, 9]]
Copy 2: [[1, 200, 3], [4, 5, 6], [7, 8, 9]]
Copy 3: [[100, 2, 300], [400, 5, 6], [7, 8, 9]]
Copy 4: [[100, 2, 300], [400, 5, 6], [7, 8, 9]]
Copy 5: [[1, 2, 3], [4, 500, 6], [7, 8, 9]]
```

So the method doing deep copy is:

- copy.deepcopy
- [row[:] for row in original_list]

This is because the new list created is only change when assign a value to it directly

The other method is shallow copy, because they share the same object, when change element in one of copy 1, 3,4, it also changes other list.

Question 7

The order of constructor is:

- b(2)
- a(1)
- c(3)
- d(4)

The order of destructor is revert, because the memory is allocated in stack memory:

- d(4)
- c(3)
- a(1)
- b(2)

Question 8

The output printed is:

```
SIZE : 8
```

To *pack* a struct is to place its members directly after each other in memory. It will pack the struct to boundary of 4-bytes. Since the char is 1 byte, short is 2 bytes and int is 4 bytes so the total sizeof Struct is 4+ 4 = 8bytes.

The `pack(push,4)` means it will push the current packing alignment value on the internal compiler stack, and sets the current packing alignment value to 4. The struct is delare after `push` so the struc will use boundary of 4-bytes.

After finish decleration of struct, we `pop` so that it will removes the record from the top of the internal compiler stack. If *n* isn't specified with **`pop`**, then the packing value associated with the resulting record on the top of the stack is the new packing alignment value. So now the current packing alignment is back to default - which is 8 for x86, ARM, and ARM64, default is 16 for x64 native and ARM64EC.

Question 9

The size of pointer will be different:

- 64 bit machine: sizeof(pointer) will be 64 bit length
- 32 bit machine: sizeof(pointer) will be 32 bit length

The max_size() member function of string, dequeue, ... of 32 bit machine and 64 bit machine will be different, because the max memory of 32 bit machine is 4GB but the 64 bit machine is much larger.

And the last one, in 32 bit system `alignof(double)` = 4, in 64 bit system the `alignof(double)` = 8

Question 10

Case 2 and case 3 run into error when executing program.

I change the `run.sh` to this so it can execute all 3 command. By using this method in Python, all main packages will be included in the path and the imports will operate accurately

```
python3 a.py # case1
python3 -m utils.dir1.b # case2
python3 -m dir2.c # case3
```

Question 11

Signals are software interrupts sent to a program to indicate that an important event has occurred.

SIGTERM is software termination signal (sent by kill by default). SIGTERM can also be referred as a soft kill because the process that receives the SIGTERM signal may choose to ignore it.

The bash script to produce SIGUSR1 will be implemented like below

```
#!/bin/bash

trap "echo 'Hello World'" SIGUSR1

echo $$

sleep 10
kill -SIGUSR1 $$
```

Question 12

`int *ptr[10]` This is an array of 10 int* pointers.

The below is example code

```
int main(void)
{
    int *ptr[10];
    int *a;
    int x = 10;
    a = &x;
    ptr[0] = a;
    std::cout << *ptr[0] << std::endl;
    return 0;
}
```

After executing, it print `10` the value of `x`.

`int (*ptr )[10]` is a pointer to an array of 10 ints.

The below is example code

```
int main(void)
{
    int (*ptr)[10];
    int a[10] = {0,1,2,3,4,5,6,7,8,9};
    ptr = &a;
    for(int i = 0;i<10;i++)
        std::cout << (*ptr)[i] << std::endl;
    return 0;
}
```

THe output terminal is

```
0
1
2
3
4
5
6
7
8
9
```