# M2\_ShortAnswers

#### Lishuo Pan

### T1.

We want to divide the whole program into several smaller program:

1. A whole program can be too complex to read or modify. If we divide it into different part it can be easier to work with
2. this dividing can make the cooperation between team mates working on the same program easier.

why do they use distinctive filename extensions such as .cpp and .h:

This can divide the files by different jobs. .h files include the declaration of the variables, structures, class, inline functions, function prototypes and #define const … while the .cpp files contain the function codes and main function which invokes other functions

### T2.

a pointer is \*p, it points to the address of a variable of a function. I can be reassigned to different address during the run time.

a reference r is the representation of a variable. Once it is assigned we can not reassign is to another variable, we use the &r to represent the address of the variable itself.

one more thing: when we pass a actual variable to the function in the C++, we are pass-by-reference. Which means we are passing the variable itself. While in the C, we can only hack this with pass-by-pointer in order to change the variable.

### T3.

const is a keyword we use in the declaration to say that the value of a variable will not change in the run time of the function.

Here is some use case of the const:

1. we can use the const to initialize some variables we don not want to change by accident in the run time of the program. e.g. const int month = 12;
2. we can declaration a pointer to be a const pointer e.g. int \* const p = &r; to make this point only point to the address of the r, and will not change in the run time of the function
3. we can declaration a pointer to a const. e.g. const int \* p = &r; In this way. We can not change the value within the &r. That is to say we can not change the value of r, even if it is not a const.

### T4.

the function overloading is the same as the function polymorphism.

In short, the function overloading occurs when more than one functions have the same name, but with the different type or number of input parameters.

The key is these functions have the different function signature which depends on the type or number of input parameters. C++ enables you to define two functions by the same name, provided that the functions have different signatures.

As for when to use them: When we want use the same function name with the different functions or different type of input, to make things clearer for the user, we should use the function overloading.

### T5.

* using declaration make one name available at one time; the using directive make all the names in one namespace available
* Using a using directive to import all the names from a namespace wholesale is not the same as using multiple using declarations. using directive will not conflict with the declaration of a variable if the name of the variable is the same. using declaration will conflict with a variable declaration if the name of the variable is the same.
* using declaration is safer: it shows exactly what names you are making available. And if the name conflicts with a local name, the compiler lets you know. You will not be warned if you accidentally overwrite you namespace variable if you use using directive.