Answered by 2015011308 唐适之

# Q&A List

The aim of the QA list is to ensure that the key knowledge in this course is delivered to you. We will summarize the good answers from our students, and compile them into a booklet for your future reference. Some questions are relatively easy to answer. You may just copy and paste from our slides. The underlined questions are extended and challenging, which need more background knowledge than what we have learned in class. Please try to find out the answers by yourself.

Assume the question No. is x and the student No. is y, then each student only needs to answer the following questions: {x | x % 4 == y % 4}.

Besides answering the required questions, each student is encouraged to propose other questions and the corresponding answers. Throughout this semester, students with enough GOOD new questions along with GOOD answers will be granted up to 5 bonus credits.

## Lecture 1

* Command-line argument（命令行参数）
  1. Suppose that you want to write an *A+B* program, which get A and B from command-line argument, how to complete the task? Please give an example of the source code, as well as how to compile and run the program.
  2. Please list（列举）the potential advantages and disadvantages of obtaining arguments by file reading vs. by command-line argument.
  3. How to input a command-line argument with spaces in it, e.g., how to input ‘Hello World’ as one argument?
  4. What is the meaning of *argc* and *argv*? What are their data types? If command “*a.exe 1 2 3”* is executed in the terminal (终端), what are the values of *argc* and *argv*?

*argc* means the number of command-line arguments passed in the program, including the program name itself as the 1st argument. *argv* is an array (or is a pointer that points to the beginning of that array) of pointers, elements of which points to a C-style string for each argument.

*argc* is of type *int*, while *argv* is of type *char\*\**.

If command “*a.exe 1 2 3”* is executed in the terminal, *argc* is 4, and *argv* is an array of length 4, in which *argv[0]*, *argv[1]*, *argv[2]* and *argv[3]* point to *“a.exe”*, *“1”*, *“2”*, *“3”* especially.

* Translation（翻译）from source code（源代码）to binary executable（二进制可执行文件）
  1. Please explain how interpreters（解释器）and compilers（编译器）work. What are the typical programming languages (典型编程语言) for each of the two translation methods? Please list the advantages and disadvantages of interpreters vs. compilers.
  2. GNU compiler g++ is universally used in both commercial and research projects. So you are supposed to know how to compile C++ source files into the binary file. State the basic usage (基本用法) to compile a source file. List useful compiling options (编译选项) as many as possible, and explain why and when we need these options.
  3. What is a library (库)? How to build a library using g++ compiler? How to use a library using g++ compiler? At what stage is a library loaded (载入)? Is it always during the linking stage (链接阶段)? Please explain the difference between static (静态) and dynamic (动态) libraries.
* Multi-file project
  1. Assume that there are *main.cpp*, *sum.cpp*, *product.cpp, and functions.h*, please give the command to compile them into *main.exe* with g++.

*g++ main.cpp sum.cpp product.cpp –o main.exe*

* 1. Please explain why multiple files are necessary for a project, especially the advantages of multi-file project compared with single-file project.
  2. What is separate compilation (分段编译)? Why is it better than direct compilation, i.e., to generate the executable directly from the source files? Please state the whole process of separate compilation.
  3. What should you do to use a global variable (全局变量) or function (函数) in several source files (源文件)? Please give as many ways as you can.
  4. Is it allowed to redefine a variable or function with the keyword **extern**? For example, “extern int a = 2;”. Why?

No. **extern** means to declare but not define a variable or function. We can use **extern** to re-declare a variable or a function, but not redefine it. In the case *“extern int a = 2;”*, variable *a* is given an initial value, so it is not only declared but defined, and therefore it is not allowed to redefine it.

We can have multiple declarations but only one definition of a variable or function, because during defining process, variable or function is assigned with memory address, and one variable or function cannot have more than one memory address.

* 1. Please explain the usage of header files (头文件). Is it allowed to define the function body (函数体) in the header file? Do we need to compile header files using g++? Why?
  2. Please explain the differences between ‘ “” ’ and ‘ <> ’ when including a header file in the source file.
* Make and makefile
  1. Please explain the dependency rules of *make*.
  2. How to enable separate compilation (分段编译) of source files using *make and makefile*? Assume that we have *main.cpp, sum.cpp, product.cpp, and functions.h,* please write the makefile to generate *main.exe* byseparate compilation.

*main.exe : main.o sum.o product.o*

*g++ main.o sum.o product.o -o main.exe*

*main.o : main.cpp functions.h*

*g++ -c main.cpp -o main.o*

*sum.o : sum.cpp functions.h*

*g++ -c sum.cpp -o sum.o*

*product.o : product.cpp functions.h*

*g++ -c product.cpp -o product.o*

* 1. On linux, how to generate a binary file with a makefile? How to use the library file to generate an executable file with makefile? Please give a simple example on Linux.
  2. Please explain more options for make, which are the command-line arguments of *make*.
  3. Please list useful advanced grammars (高级语法) of makefile as many as you can. It is suggested to explain the usage of these grammars. For example, it is convenient to automatically detect all the source files in a folder (自动检测文件夹下的所有文件) and automatically compile them all. How does *makefile* support this?

## Lecture 2

* Abstraction

1. Generally speaking, what are the two major parts of a class? Please define a class that describes a *computer*, and then give an example of how to instantiate an object and access its members.
2. Where should we place the class definition, in the header file or the source file? Please try to explain why header files are necessary.
3. Please try to explain why the designers want to organize data and functions into classes? What are the advantages and disadvantages?

* Access control

1. Please try to use access control to enable information hiding of your class definition for *computer* in Problem No.1.

class computer

**{**

private**:**

memory\_t memory**;**

cpu\_t cpu**;**

disk\_t disk**;**

keyboard\_t keyboard**;**

screen\_t screen**;**

// ...

**void** access\_memory**(**mem\_address\_t address**);**

**void** access\_disk**(**disk\_address\_t address**);**

// ...

public**:**

**void** power\_on**();**

**void** power\_off**();**

**void** keyboard\_input**(**key\_t key**);**

// ...

**};**

1. Please try to explain the benefits of access control.
2. Please tell the difference between keywords ***public***, ***private*** and ***protected***. We know that access control works for objects of the class. Does access control work in member functions? I.e., is it possible that one member function cannot access another member?
3. Please try to explain why it is suggested to define a member function outside class definition.
4. Please explain the differences between keyword ***struct*** and ***class***.

In C++ (not in C), the only difference between these two is that, default access control in **struct** is *public*, while default access control in **class** is *private*.

* Friend

1. What should you do to permit a non-member function to access private or protected members? What about permitting another class to access them? Please give an example.

* Inline

1. Please try to define some inline functions in Problems No. 1 with keyword ***inline***. It is suggested to add a header file.
2. What are the advantages of inline functions? When and where should we use keyword ***inline***?
3. Please try to explain the implementation of inline functions (内联函数的实现) in compiler, and explain the reason for the advantages of inline functions.

The compiler inserts the instructions directly into where it is called, and stores parameters, returned value and local variables directly into the stack layer use by the function that calls the inline function.

However, the compiler stores instructions for a non-inline function in a separate area, which is pointed by a point where the function is called. The parameters, returned value and local variables are stored in a newly created stack layer.

Calling a small inline function is faster for the following reasons. Firstly, it is no need to create a new stack layer to stores the extra data. Secondly, the instructions are stored together with the instructions outside the function, so the CPU is more easily to cache the instructions from memory.

1. What will happen if we abuse (滥用) keyword ***inline***?

* Header guarding

1. Please give an example of header guarding using preprocessor directives (预编译命令) “#ifndef … #define …#endif”, and try to explain the possible compiling errors without them.

* This pointer

1. What does ***this*** mean? How to avoid the name conflicts (名字冲突) between member and non-member variables with ***this***? How to return current objects in member functions?

* Memory allocation

1. What are the differences between ***new/delete*** and ***malloc/free***?
2. **new/delete** will automatically calculate the needed memory length by the data type, while **malloc/free** programmers have to calculate it themselves.
3. **new/delete** will call the corresponding constructor of the objects that use the newly allocated memory, but **malloc/free** just allocates the memory and does nothing else.
4. Why does C++ bring in ***new/delete*** to replace ***malloc/free*?**

* Incomplete class

1. Please list the situations where we need to use incomplete type, i.e., forward declaration (前向声明).