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 3

* Scope

1. What does the scope (作用域) mean? State the scopes of *a*, *b*, *c* and *d*.

int a;

int f ()

{

static int c;

int b;

{int c; }

}

int main ()

{

f ();

}

1. State the scopes of the objects of struct O.

struct O

{

int i;

};

O f (O o)

{

return o;

}

const O& g (O o)

{

return std::move(o);

}

O h (const O& o)

{

return o;

}

* Constructor and destructor

1. Shall we declare destructors as public members? And why? What about constructors?
2. When will copy constructors and move constructors be called?

When constructing an object with L-value references, it will call the copy constructor. One can achieve this by explicitly passing a value (T for type T), or an L-value reference (T& for type T) to the constructor, or using implicit or explicit type converting.

When constructing an object with R-value references, it will call the move constructor. One can achieve this by explicitly passing an R-value reference (T&& for type T) to the constructor. “std::move” is used to convert an L-value reference to an R-value reference.

1. What are constructors and destructors for? Try to write an example of a class with constructors and the destructor.
2. Can you call constructors or the destructor explicitly? Should we do that?
3. State which constructors are called in the following statements.

constructor ();

constructor (int);

constructor a;

constructor b(5);

constructor \*c = new constructor;

constructor \*d = new constructor (5);

constructor e[5] = {};

constructor f[5] = {(1), (2)};

1. How to define an object on stack? And how to define an object on heap? What will happen if you define a big object on stack whose size is greater than the system’s limit?

There are 4 ways to define an object on stack.

1. Use temporary variables. For example, (for type T, the same below)

**{**

T a**;**

**}**

1. Create an object but not assign it to a variable. For example,

a **=** T**().**f**();**

1. Copy to a parameter. For example,

**void** f**(**T a**);**

// b is an object of T

f**(**b**);**

1. Copy to the returned value. For example,

T f**() {** **return** a**; }** // a is an object of T

There are 2 ways to define an object on heap:

1. Use malloc. For example,

T **\***p **=** malloc**(sizeof(**T**) \*** num**);**

1. Use new. For example,

T **\***p **=** **new** T**[**num**];**

If defining an object on stack which is bigger than limit, it will cause stack overflow, and will lead to a segmentation fault.

1. Do you need to free the memory taken by stack objects? And do you need to free the memory taken by heap objects? What will you do to ensure that no memory leaks (内存泄漏) occur?
2. Are destructors always necessary? Please explain why.
3. Is the code below correct? If there are errors, please point them out and then state how to fix the problems.

struct O

{

int i;

O (int i) {}

};

int main ()

{

O\* o = new O[5];

}

## Lecture 4

* Function overloading

1. Why do we need overloaded functions? What kinds of functions can be overloaded?
2. Please explain how the compiler distinguish between two functions with the same name.

* Default argument

1. Where should default arguments be provided? In declarations or in definitions?
2. Which arguments in the function’s argument list can have default values? Please explain how the grammar avoids the ambiguity (歧义).

The tailing arguments can have default values. When there are multiple variables which have default values, the left most variables have the priority to get the value passed in, so there is no ambiguity.

1. How to add an argument to an existing function without affecting the existing calls to the function (原有的函数调用)? Please give a simple example.
2. Does the following codes work? Why?

struct O

{

int i;

O (int i = 0) {}

O (int i, int j = 0) {}

};

int main ()

{

O o(6);

}

* Macro

1. Please state the technology of header guarding by macros “#ifndef xxx … #define xxx … #endif”. Please give an example of compiling error without header guarding.
2. Why are we supposed to avoid macros for defining numbers? What grammars can be used instead?

There are 2 disadvantages:

1. Macros can be redefined. One may pollute a defined macro by accidents.
2. Macros may lead to a result you don’t expect. For example,

**#define a 1+2**

a **==** **3**

a **\*** **2** **==** **5**

One can use const variables instead.

1. In the following example, please explain why we have to remove the commented line for avoiding the compile error. And what is the output?

enum smallenum: std::int16\_t {A, B, C };

enum color {red, yellow, green = 20, blue};

enum class altitude: char{ high='h',low='l',};

enum {d,e,f = e + 2};

int f1 (int a) {}

int f2 (char a) {}

int main ()

{

std::cout << sizeof(smallenum) << " " << A << " " << C << std::endl;

std::cout << sizeof(red) << std::endl;

std::cout << sizeof(d) << " " << d << std::endl;

f1 (d);

f1 (color::red);

//f2 (altitude::high);

f1 (A);

}

* Constants

1. Why can we define a constant variable in the header file without causing compiling errors? What should you do to use a constant variable defined in another source file?
2. Please explain the difference between the following statements.

const int\* a = s;

int const\* b = s;

int\* const c = s;

1. How to initialize a constant member variable?
2. Use the “:” expression in constructor. For example, (C is the class and a is the constant variable, same below)

C**::**C**()**

**:** a**(0)**

**{}**

1. Since C++11, we can define constant variables inline if the definition won’t call another function. For example,

**const int** a **=** **0;**

If the definition calls another function but its result can be determined in compile-time, we can use constexpr. For example,

constexpr **int** a **=** f**();**

1. How to convert a constant variable into a non-constant one?
2. Where should the keyword *const* be placed to declare or define a constant member function? Please give an example.
3. Please state the restrictions of constant member functions. Can we avoid it (i.e., access the members in other ways)?