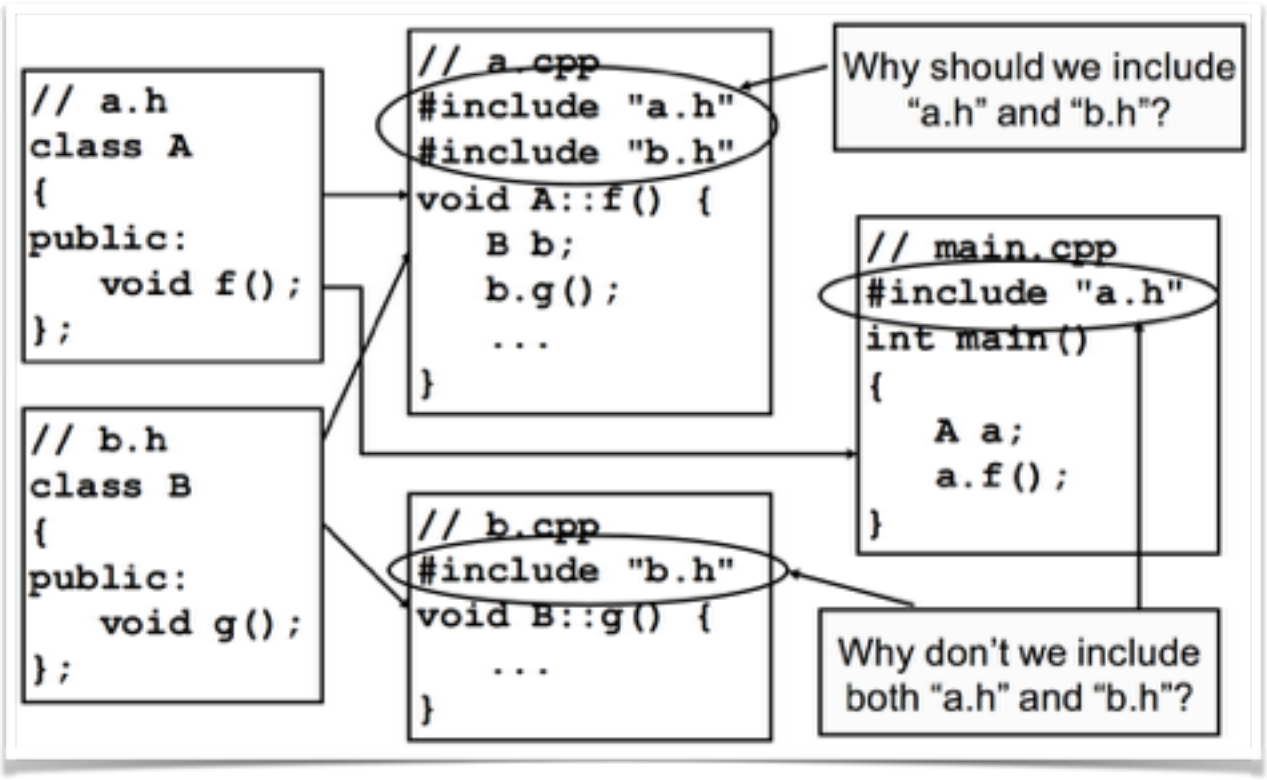

Practice #2 (for lecture note #2)

C++ Review Part II:

More on Functions, Variables, Classes

Wednesday, 2018-09-26

Notes and Advices:

1. Submission of the practice is NOT mandatory. The main purpose of these problems is for you to practice and get refreshed on the contents of the lecture note for the coming class meeting. Practice of the problems before class is HIGHLY recommended.
2. However, in case you need to "bargain" your term grade while you are below the next/higher letter grade for a small enough margin, you can talk to me if you have been consistently submitting your practices in time.
3. Note that your submission of the practice will NOT be graded or checked (by me or TA). Again, this is for your practice only.
4. The submissions on NTU CEIBA have strict deadline. Usually it is before midnight of the class meeting.
5. Practice problems will be, hopefully, uploaded to CEIBA no later than the previous weekend of the class.

Problems // "(pn)" refers to the page #n in the lecture note

1. (p7) Define a function, say "`void f(int a)`"
   - Write the code in the following order:
     i. The function prototype (i.e. forward declaration)
     ii. `main()`, and call `f()` inside main()
     iii. Definition of function `f()`.
   - Make sure the code can be compiled. Then remove the function prototype. Compile again. What error do you see?
   - Put the function prototype back. Define the default argument in both function prototype and function definition. Compile again. What error do you see?
   - Change the values of the default arguments. Can the compilation error be fixed?

2. (p16) Define two variables as follows
   ```
   const int a = 10;
   int b = 10;
   ```
   Add the following codes, and compile the program to see it there is any error. If no error, execute to see the results.
   - `a = 20;`
   - `a = 10;`
   - `a = b;`
   - `int& c = a; c = 20; cout << a;`
   - `const int& c = b; c = 20;`
   - `const int& c = b; b = 20; cout << c;`

3. (p24) Define a `class T` with a data member "`int _d`", a non-const method `f()` in which `_d` is incremented by 10, and a const method `p()` that prints out the value of `_d`. Declare a const object of `T`

as "`const T a(10)`", in which `_d` is initialized to 10. Add the following codes, and compile the program to see it there is any error. If no error, execute to see the results.

- `a.f();`
- `T(a).f().p(); a.p();`
- `const_cast<T *>(&a)->f()->p(); a.p();`

4. (p35) Declare a `char` array and a `void*` as:

```
char c[33] = "0123456789abcdefghijklmnopqrstu";
void *p = c;
```

   - Define "`char *p1`", "`short *p2`", and "`int *p3`" and initialize then to `p`.
   - Print out `p1, p2, p3` and `(p1+1), (p2+1)` and `(p3+1)`. See how they diff.
   - Define "`short *q = p2+1`". Try to define "`int *s`" whose is equal to `q`. Note `s` is now NOT multiple of `sizeof(int)`.
   - Do "`*s = 0`". Print out `p1, p1+2, p1+4, p1+6` to see what are affected.
   - Note that in the above practices, you may encounter type-casting errors. Try to make use of "`void *`".

5. (p53) Define two header files as:

   [a.h]

```
class A { };
```

   [b.h]

```
#include "a.h"
class B { A _a; };
```

   Define p5.cpp as:

```
#include "a.h"
#include "b.h"
int main() { A a; B b; }
```

   - Any compilation error? Why? How to fix it?

6. (p64) Define a class N as:

   [a.h]

```
class N {
   void *_p;
public:
   N(void*p): _p(p) {}
};
```

   - Instantiate two objects `n1, n2` of `class N` with some pointers.
   - Define a function "`void setMark()`" that uses the LSB of `N::_p` to record the object is "marked".
   - Define a function "`bool checkMark() const`" that check whether this object is marked.
   - Define a function "`void* getPtr() const`" that returns a valid pointer for `N::_p` (that is, a pointer address without the mark bit).

- Use `n1, n2` to play around the above functions.

7. (pp 85-86)
   - Define a `class N` and `N_` as:

```
class N {
   N_  *_n;
public:
   N(): _n(0) {}
   void gen();
   void statistics() const;
};
class N_ {
   friend class N;
   size_t    _d[1 << 17];  // 1MB
   unsigned  _refCnt;
   N         _child1;
   N         _child2;
   N_(): _refCnt(0) {}
};
```

   - Define a global variable:

```
N_* nList[1 << MAX_DEPTH] = {0};
```

   - In main(), do:

```
srandom(getpid());
N root;
root.gen();
root.statistics();
```

   - The behavior of `N::gen()` is as follows:
     - Assert `_n == 0`.
     - Generate a random number "`i`" in [0, 1 << MAX_DEPTH)
     - If (`nList[i] == 0`)
       - Create a new `N_*` and assign it to `_n`.
       - Increase its `_refCnt` and assign it to `nList[i]`.
       - For each of its children, recursive call `_child.gen()`.
     - Else // if (nList[i] != 0)
       - Let _n = nList[i].
       - Increase its _refCnt.
   - The behavior of `N::statistics()` is as follows:
     - Define `maxRef` to be the maximum number of `_refCnf` for all the nodes in `nList[]`.
     - For `i = 0 to maxRef`, print out the number of nodes in `nList[]` whose `_refCnf == i`.
     - The sample output is as:

```
Ref[0] = 52830 (20.2%)
```

```
Ref[1] = 85249 (32.5%)
Ref[2] = 67460 (25.7%)
...
```