1. namespace:: / classFurtherDefine:: / object. / pointerToObject->  
   class constructor, public private, inheritance :  
   for while function can ignore ; but class and commands cannot
2. namespace std: cout, cin
3. for(......;i++) == for(......;++i)
4. array:
   1. int A[2][3] = {{1,2,3},{4,5,6}};
   2. char C[5] = {‘a’,’b’,’c’,’d’,’e’}; | but char C[5]=”abcde” is not available
5. return space byte:
   1. sizeof: int4 / char1 / NULL4 / charArrayX / stringDirectX+1 / stringVariable8 / intArrayX\*4
   2. stringVariable.size(): X
6. int\* pointer; \*a value; int& alias; &a address
   1. int\* ptr = &a;
   2. int\* ptr = new int; int\* ptr = new int(5); delete ptr
   3. int\* ptr = new int[5]; delete [] ptr;
   4. int A[3]={1,2,3} | int\* ptr = A | but int\* ptr = {1,2,3} is not available
7. int in function:
   1. CBV: f(int x) | f(x)
   2. CBA: f(int\* x) | f(&x)
   3. CBR: f(int& x) | f(x)
8. array in function:
   1. A; A==&A==&A[0]; Therefore, must input address and size
   2. CBA: define function f(int\* A)==f(int A[]) | call function f(A)
9. struct, struct-constructor:
   1. definition: typedef struct{int leg; int eye;}Animal;
   2. instantiation object: Animal obj = {4,2} | Animal obj = {.leg=4, .eye=2}
   3. instantiation pointer: Animal\* ptr = malloc(1\*sizeof(Animal));
10. (C++) function overloading: parallel | function overriding: overwrite
11. constructor vs destructor:
    1. definition:
       1. constructor: Predefine class attributes that different from the input, then assign the values. e.g. int x, y; ListNode(int cx, int cy):x(cx),y(cy){}
       2. ~ListNode(){}
    2. instantiation object:
       1. ListNode obj1 = {1,2};
       2. ListNode obj2 = {.x=1, .y=2};
       3. ListNode obj3 = ListNode(1,2);
       4. ListNode obj4(1,2);
    3. instantiation pointer:
       1. ListNode\* ptr = new ListNode(1,2);
    4. constructor have parameters but destructor havn’t
    5. constructor can be overloaded but destructor can’t
    6. both of them cannot return
    7. If no constructor, instantiation needs () as well. i.e. Animal obj = Animal();
12. Operator overloading:
    1. this: self pointer
    2. myClass operator+(myClass& obj)
13. Encapsulation | access specifier:
    1. public: everyone
    2. private: class itself and friend class | friend void f(myClass& obj)
    3. protected: similar to private but allow son and friend’s son to access
14. Inheritance: based class - Animal | derived class - Dog
    1. public inheritance | class Dog: public Animal  
       public-public, protected-protected
    2. protected inheritance | class Dog: protected Animal  
       public-protected, protected-protected
    3. (default) private inheritance | class Dog: private Animal  
       public-private, protected-private
    4. constructor and destructor are not inherited. However, they will be called.  
       i.e. AnimalConstruct->DogConstruct->DogDeconstruct->AnimalDeconstruct  
       Dog(int y) : Animal (y) {cout<<"Dog"<<endl;}
15. Polymorphism:
    1. Dog, Cat inherit from Animal but override a method in different ways
    2. Virtual function (for overridden) in Animal: virtual void f(){}; It can also be implemented. Hence it is just reminder;
    3. Pure virtual function (for mandatory overridden) in Animal: virtual void f()=0;  
       Must be implemented in Dog and Animal cannot be instantiated. i.e. Virtual class
16. Template:
    1. function: template<class T,class U> T sum(T a, T b){return a+b;}
    2. class: template<class T> class myClass{}; myClass<T> obj = myClass<T>();
    3. template overloading: template<> class myClass<char>{};
17. Exception:
    1. try{ throw 99; } catch(int ball){} | catch(...) catch all types of error
18. File:
    1. #include <fstream>
19. Include:
    1. header (.h): interface. guardInclude and namespace-class-attrMethod only
    2. source (.cpp): program without main. include “xxx.h” and implement  
       e.g. void myClass::f(){}
    3. main (.cpp):
       1. include “xxx.h”
       2. Compile g++ source.cpp main.app
       3. a.exe
20. pointer vs iterator:
    1. usages are almost the same. But pointer is for array, iterator is for STL.
    2. For ordered data: both \*(ptr+i) and \*(it+i) are i^th element
    3. iterator begin 第0元素 | end 超過最後一元素
21. string:
    1. string s = “apple”; s.substr(1,2); string s(5,‘0’)
    2. for(auto); for(int);
    3. s.find()
    4. s.push\_back(‘a’)
    5. s.size()
    6. stoi(), stol, stoll() | to\_string()
22. unordered\_map (hashtable) vs map (hashtable+BST) vs multimap (BST)
    1. unordered\_map<int,int> H = {{“apple”,20},{“banana”,10}};
    2. for(auto it=H.begin();it!=H.end();++it) cout << it->first << " " << it->second;  
       auto 為 unordered\_map<int,int>::iterator 縮寫
    3. H.find("apple")==H.end() 即 not in
    4. H[“apple”]=30 | H.erase(“apple”)  
       erase可放元素 可放迭代 可刪除沒有的元素 若於multimap放元素則會全部刪除  
       若要刪除一個 auto it=H.find(); H.erase(it)
    5. H.size()
23. unordered\_set (hashtable) vs set (hashtable+BST) vs multiset (BST)
    1. unordered\_set<int> S = {1,2,3,4,5};
    2. for(auto it=S.begin();it!=S.end();++it) cout << \*it;
    3. S.find(1)==S.end() 即 not in
    4. S.insert(6) | S.erase(5)
    5. S.size()
    6. ordered set: min=\*S.begin(), max=\*S.rbegin()
24. vector
    1. vector<int> V = {1,2,3}; vector<int> V(100,0);
    2. for(int i=0;i<V.size();i++) cout << V[i];  
       for(auto it=V.begin();it!=V.end();++it) cout << \*it;
    3. find, count, min, max -> see algorithm
    4. V.push\_back(100) O(1) | V.insert(V.begin()+i,100) 插入後變第i元素 O(n)  
       V.pop\_back() O(1) | V.erase(V.begin()+4) 刪除第i元素 O(n) erase只可放迭代
    5. V.size()
25. algorithm in vector<int> V = {1,2,3};
    1. find(V.begin(), V.end(), item) != vec.end()
    2. auto it = find(V.begin(), V.end(), item);  
       int index = (it==V.end())? -1 : it-V.begin();
    3. count(V.begin(), V.end(), item)
    4. \*min\_element(V.begin(), V.end()) | \*max\_element(V.begin(), V.end())
    5. sort(V.begin(), V.end(), [](int a, int b){return abs(a)<abs(b)};
    6. Heap
       1. make\_heap(V.begin(), V.end());
       2. pop\_heap(V.begin(),V.end()); V.pop\_back();
       3. H.push\_back(9); push\_heap(V.begin(),V.end());

--------------------------------------------------

class C{

public:

int x, y;

C(int ix, int iy):x(ix),y(iy){}

};

int main(){

C obj = C(3,5);

cout << obj.x << obj.y << endl;

C\* ptr = new C(3,5);

cout << ptr->x << ptr->y << endl;

}