**Programming\_Abstractions\_in\_C++**

**[Eric\_Roberts]**

**Review Questions:**

**LEC1: Overview of C++**

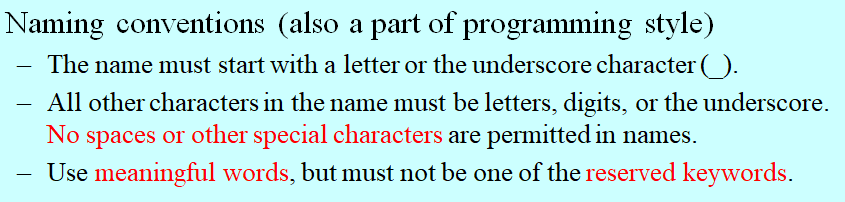
1. Source file
2. **//** and **/\* … \*/**
3. <>: Internal & C++ Standard library like <iostream> <vector>

“”: External library defined by users like “morsecode.h”.

1. const double CENTIMETERS\_PER\_INCH = 2.54;
2. main; return 0;
3. Line break, end current line and output a new line.
4. (int i = 0) 🡪 int(type) i(name) = 0(value);

scope: -2^31 ~ 2^31 - 1

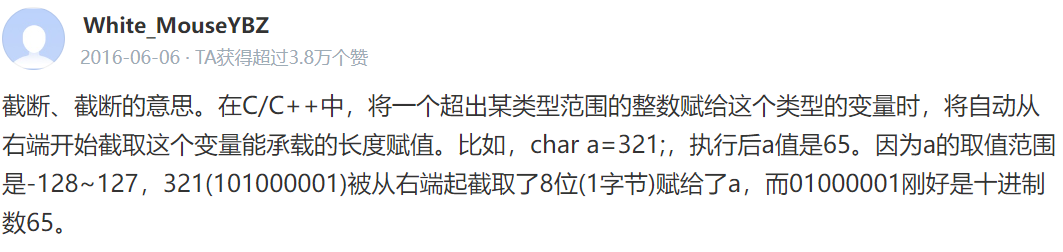
1. abcfhkl



1. domain of values & operations
2. domain of values & operations
3. America Standard Code for Information Interchange
4. true(1) false(0)
5. double x; cin >> x;
6. cout << “i=” << i << “ d = ” << d << “ c = ” << c << “ s = “ << s;
7. 5-int, 3-int, 3.8-double, 18.0-double, 4-int, 2-int
8. Unary minus: represent negative number, can work as the left value.

Subtraction: minus two numbers, an operation, requires two numbers to operate.

1. When a number exceeds the scope of its type, the number’s bit will be cut down for assignment.



1. Transform a type to another type

int i; double j = 9.9; i = int(j);

1. a-4, b-2, c-42, d-42
2. a = (x>y) ? x:y;
3. ++x: add x by 1 and return;

x++: return x first and then add x by 1;

1. Only calculate right-side value when necessary. E.g. if (1 == 1 || x = y): this will not judge whether x equals to y.
2. if (){} else{};

switch(){case 1:{break;} case 2:{break;} … default:{break;}}

while(){break;}

for (int i=0;i<n;i++){}

1. Select a case to execute. Break means exit current switch statement, or the program will execute the next case.
2. The value to stop iteration.
3. for (int i=1;i<=100;i++) for (int i=0;i<100;i+=7) for (int i=100;i>=0; i-=2)

**LEC2:** **Functions and Libraries**

1. Function: receive an input, give an output, part of the program.

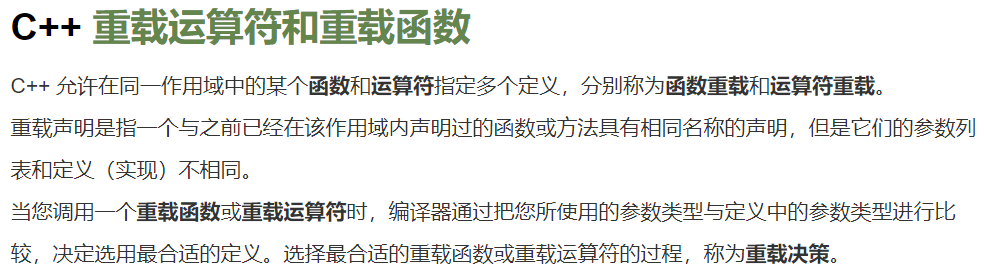
Program: a complete executing unit.

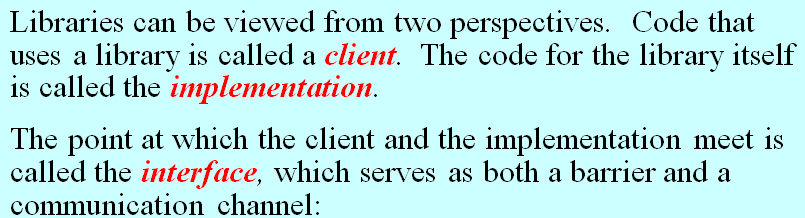
1. Call: Invoke the function

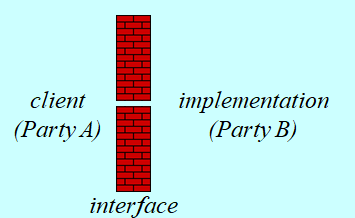
Argument: things pass to the function.

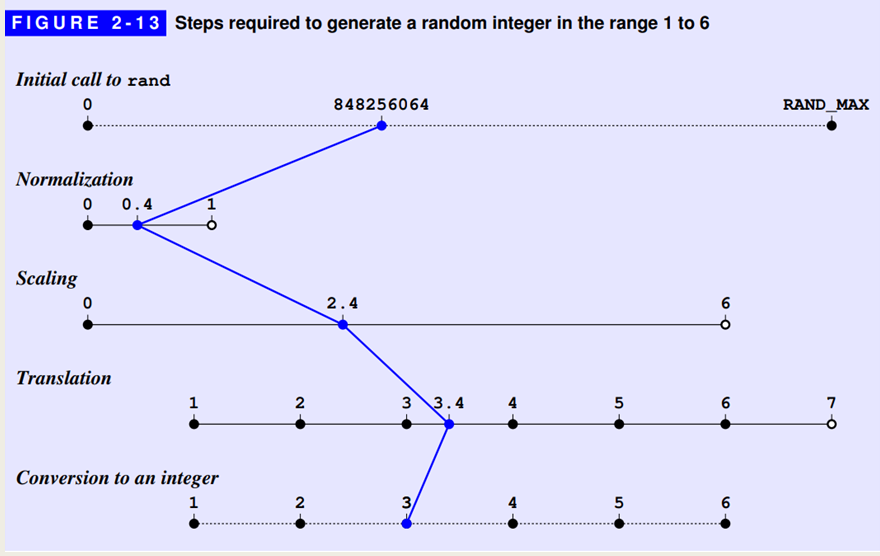
Return: function output

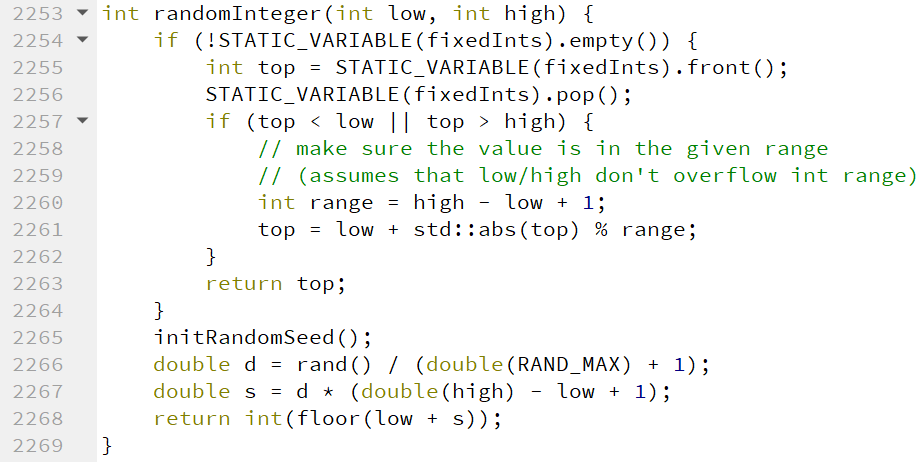
1. False. No need to write a prototype every time, main() also doesn’t need.
2. double sqrt(double i);
3. Yes. In if condition statement.
4. Function that returns a bool value.
5. Define multiple implementation for the same function name (but different function parameters) Use signatures to choose the most appropriate implementation to invoke.



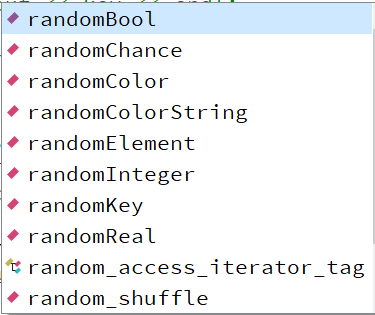
1. Write the variable’s initial value in the prototype / function name. int f(x, y = 0);
2. False. int f(x = 0, y) is false, it will cause ambiguity when invoking f(1);
3. The memory to store local variables (initialize in function).
4. The value of each argument is copied into the corresponding parameter variable when calling a function.
5. Local is its scope, the variable’s life cycle is inside the function.
6. It means call by reference [狗头]
7. Use &
8. 



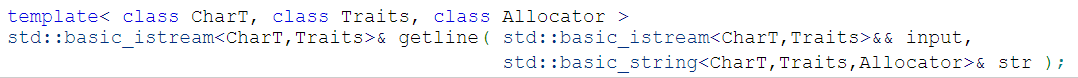
1. #include “mylib.h”
2. Use extern
3. Unified + Simple + Sufficient + Flexible + Stable
4. Maintain the same structure and effect. Making changes in the behavior of a library forces clients to change their programs, this reduces its utility.
5. False random, which means it’s not actually random but having certain rule.
6. Largest int (2^31 - 1)
7. 
8. randomInteger(1,100)
9. It depends on the implementation actually.



For Stanford Library’s case, it’s Yes, and -5, -4, -3, …, 3,4

1. No, d1 and d2 will have the same value.
2. true (if not set the seed according to current time)
3. Initial state of the random number, to generate random sequence.
4. Set the seed fixed.
5. 

**LEC3:** **Strings**

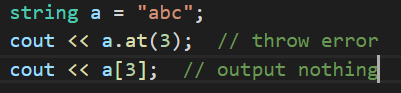
1. String is the combination of character (superset).
2. False (If there is a space, it will only read the part before the space)
3. 

Both istream and string.

1. Method: string.size(). Use a dot after the variable. Function: getline(cin, string); Use a parentheses
2. False. It should be str.length() (method instead of function)
3. s1. Method meaning: replace(pos, len, str) Start from index pos, replace string with length len with sting s2.

<http://www.cplusplus.com/reference/string/string/replace/>

1. Connect two strings.
2. Start from head, compare the ASCLL value of each character at the same position until difference. E.g. “abc” < “abd” (ASCLL value of ‘d’ is larger than ‘c’)
3. str[i] str.at(i) The latter one will check whether the index is out of boundary.
4. [] will not check effectiveness. .at() will throw out an exception of “out of range”

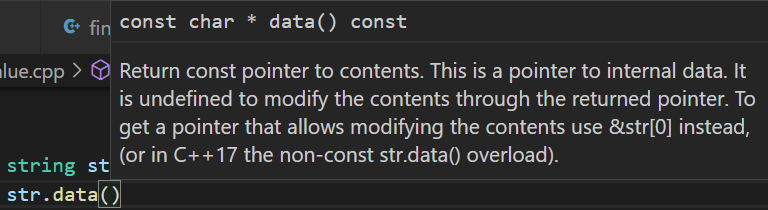


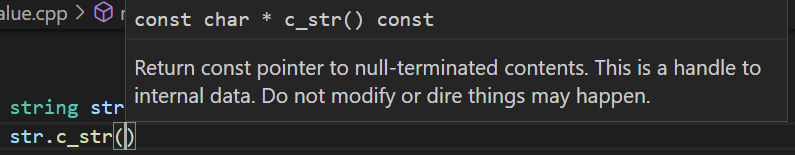
1. True
2. True
3. str.substr(pos, len). Start from index pos, return a cut string with length len.

Omit the second: Cut the string until the end. E.g. “abcde”.substr(2) = “cde”

1. string::npos <http://www.cplusplus.com/reference/string/string/find/>
2. repeat with 13
3. 

Start from index pos to find.

1. 5; 0; ‘C’; “ABCDE”; “a”; “ZCDE”; “ABC”; “E”; “DE”; “DE”
2. for (int i=0;i<str.length();i++){str[i]…} or for (char ch:string) {ch…}
3. for (int i=str.length()-1;i>=0;i--) {str[i]…}
4. Use + operator
5. False; True; False; 7; ‘A’; ‘a’
6. For the compatibility with C language.
7. 





**LEC4:** **Streams**

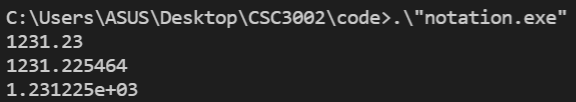
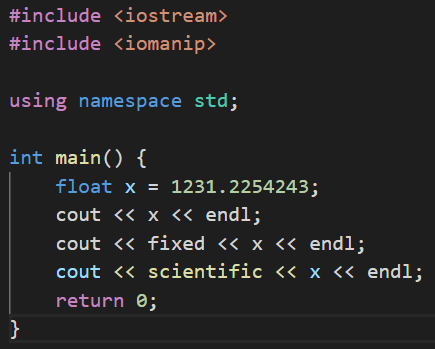
1. fstream, ifstream, ofstream
2. <<: insertion operator >>: extraction operator
3. << returns output stream (cout) >> returns input stream (cin)

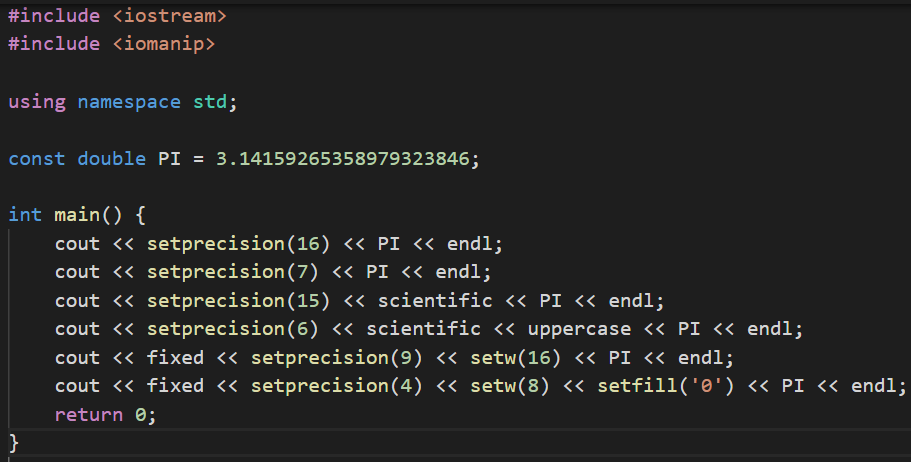
To support chain operations.

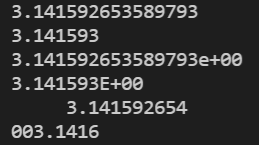
1. Operator to change format (skipws, noskipws, ws)
2. Transient: take effect once; Persistent: take effect forever
3. fixed: fixed-point notation.

scientific: scientific notation.

Default: Round the float number and output.



1. 



http://c.biancheng.net/view/275.html

1. For file I/O operations: read file, write file.
2. When using open, one should convert the string to C-style string first.

fstreamName.open(filename.c\_str());

1. Use fstreamName.fail()
2. fstreamName.get() == EOF or fstreamName.get() == fstreamName.eof();
3. There may be other characters like Chinese or Korean outside ASCII range (0~255).
4. Read the character back to the stream.

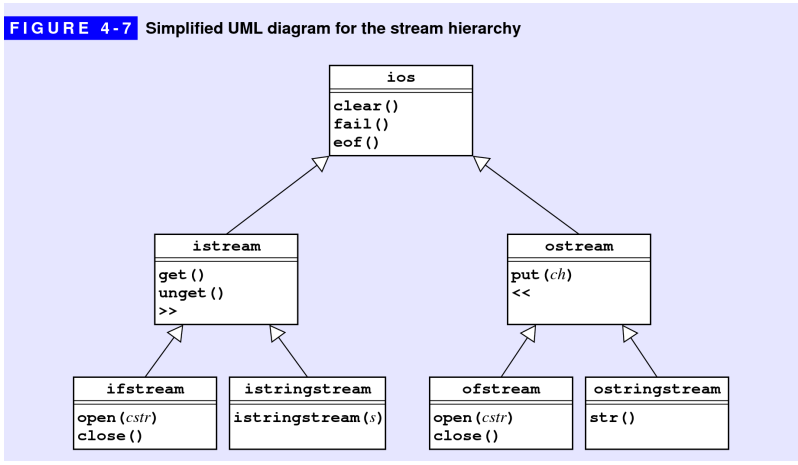
<https://blog.csdn.net/RuizeMa/article/details/6149662>

1. while (getline(cin, stringName)) {…}

Just judge whether the return value of getline() is false;

1. stringstream Stringstream deals with string (commonly used scenario: type conversion) Fstream deals with file read/write.
2. subclass: subclass; superclass: superclass; inheritance: inheritance.

听君一席话，有如听君一席话 (Check PPT)

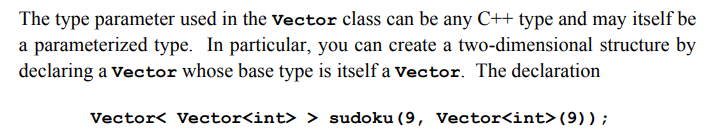
1. 

False. It is a superclass.

1. For more general purpose. E.g. deal with cin/cout, stringstream.
2. Can handle error checking. E.g. whether there is a space.
3. Google / Baidu.

**LEC5:** **Collections**

1. True. (If you have come to my tutorial, you gonna remember this question, type like **int / char / double** are defined in representation)
2. Simplicity, Flexibility, Security
3. Standard template library
4. #include “vector.h”
5. Dynamic size; Support insert/delete operations; More flexible initialization
6. Check whether the index is out of boundary
7. Classes that include a base-type specification (vector<**int**>)



1. vector<bool>
2. False. The default size is 0 but not 10
3. Vector<int> vec(20,0);
4. The member function: .size()
5. 0 to N-1 (The first argument to **insertAt** is an index number, and the new element

is inserted before that position)

0 to N-1 (Remove the element at the index)

1. The operator **[]**
2. To avoid copying a new vector/object and consume great storage and time
3. Grid<char> chessboard(8,8);

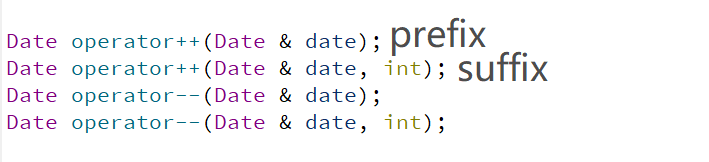
vector<vector<char>> chessboard(8, vector<char>(8))

1. chessboard[7][0] = ‘R’; chessboard[7][7] = ‘R’;
2. Last In First Out - Stack; First In First Out - Queue;
3. Push & Pop
4. Enqueue & Dequeue
5. Return the element at the top (the first one)
6. We can only simulate discrete time instead of continuous time (0, 1, 2, … minutes)
7. Key & Value
8. 
9. The operator **[]**
10. Lexicon class is space-efficient and convenient to iterate
11. .txt and .dat files
12. for (type var : collection) {}; foreach (type var in collection)
13. Destroy the implicit data structure. Destroy there definition and access rules of FIFO and LILO, can only access element from one side.

29. Vector: index order. Map & Set: no specific order, sometimes alphabetic order

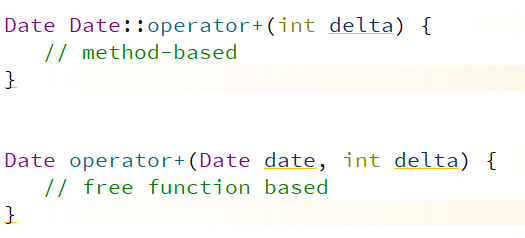
**LEC6:** **Class**

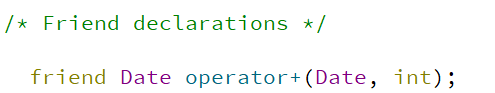
1. Find them in PPT
2. public: Accessible and modifiable by instance object. private: Not accessible by external instance but in the internal class definition. The difference is whether it is available to the clients or not.
3. False. Although this is the fundamental difference, but there are other differences like no inheritance for struct and the struct’s space is on stack and class on heap.
4. The dot operator ***.***
5. Class name() (e.g. BigInt() )\
6. 1, which is the class object itself: \*this (just like **self** in python), hidden
7. The function to access or set instance variables.
8. Impossible to modify the values of any instance (member) variables.
9. Use the double colon :: (e.g BigInt::) as the namespace
10. #include
11. int operator% (int x, int y)
12. prefix: first add/minus then return (++date). Suffix: first return then add/minus. add a “int” after operator, (date++)



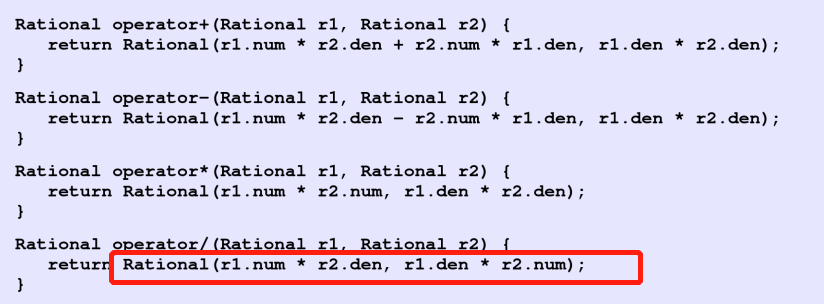
1. Stream type variable cannot copy by value. (If we do not return by reference, it’s return by value, the stream will be copied to another variable and return, but it’s forbidden)
2. False. It depends.
3. Method based: overload operators as a member function, has a default hidden argument: \*this, can access private member variables freely.

Free-function-based: Easy to read and understand. But no access to private variables unless using “friend” symbol.

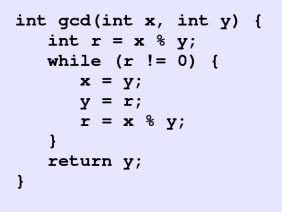
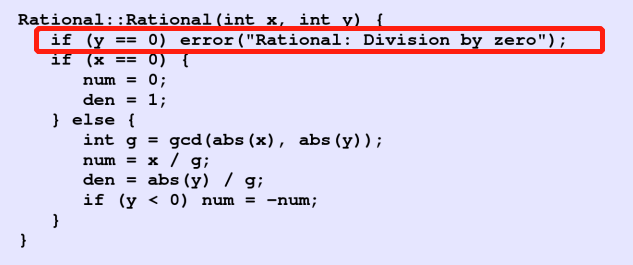




1. The method or another class can access the private variables of this class.
2. To iterate all directions. To define clockwise/counter-clockwise iteration.
3. Check PPT. (Think – decide private variables – design constructor – design operations/free functions – code, test)
4. A number that can be written as a fraction (a / b).
5. num and den are relatively prime, den > 0
6. Yes. Maybe, it depends on how divide & constructor & gcd() are implemented.



We don’t have explicit “/” here, so no division by zero error

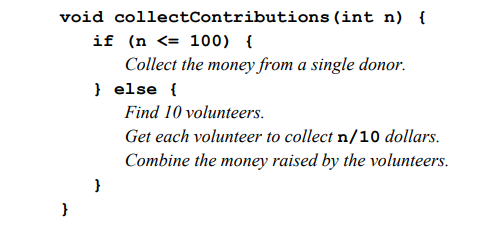
 

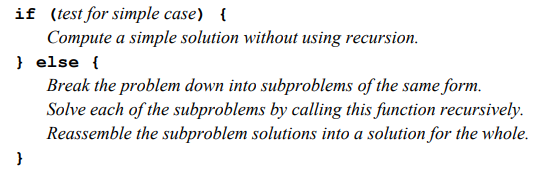
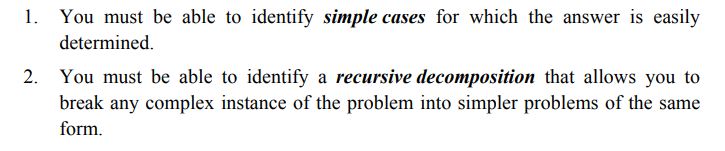
It will signal an error under current implementation

1. << does not access private variable but invoke public function .toString()
2. A Logical unit in a string
3. Iterate input, use scanner.nextToken() to get next token
4. TokenScanner scanner(); scanner.ignoreWhitespace();
5. Design method, set public variable/function, private variable/function reasonably.

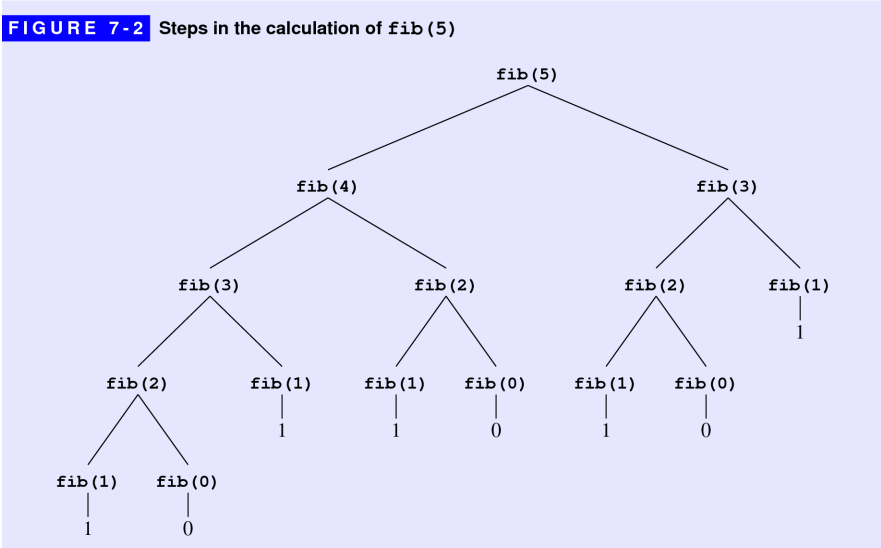
**LEC7:** **Recursion**

1. Invoke the function itself, iterate a certain collection/range. Yes
2. The function makes use of the same structure of the problem to solve larger problem with smaller parameters and the same interface, invoke itself.
3. n may be smaller than 100. All n <= 100 should belong to the basic case.



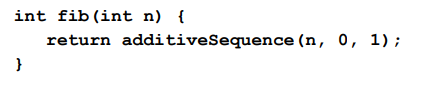
1. 
2. 
3. Divide the problem into a smaller problem and conquer
4. Assuming that any simpler recursive call will work correctly—is called the recursive leap of faith. Whenever you try to understand a recursive program, it is useful to put the underlying details aside and focus instead on a single level of the operation.
5. fib(3) = fib(2) + fib(1) = 1 + 1 = 2; Sketch out by yourself
6. Each element of a sequence is defined in terms of earlier elements

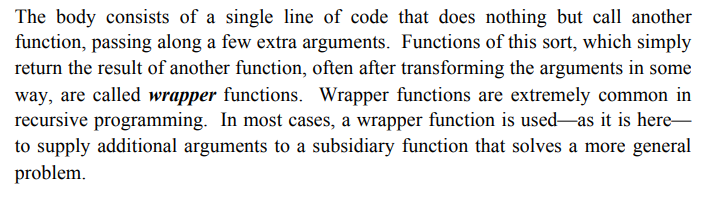


1. Rn = Rn-1 + Rn-2 – Rn-4
2. 

Call fib(1): fib(n-1) + fib(n-2)

1. A wrapper function invoke





1. Yes. Calculate one more term.
2. They are the base cases. No. When len = 1, the invoke term len-2 could be -1.
3. Check whether the middle term is palindrome after checking the characters on two sides.
4. You call me, I call you.
5. Mutual recursion will happen and never ends. No reduce, No simple case.
6. isEven(1) = isOdd(0) = isEven(-1) = …. Basic case is not complete.

**LEC8:** **Recursive Strategies**

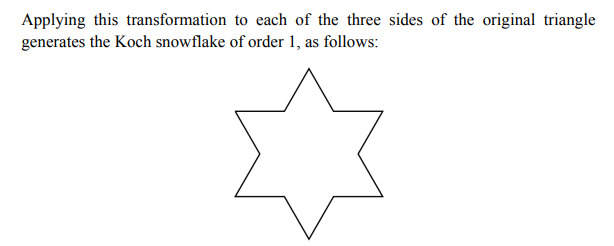
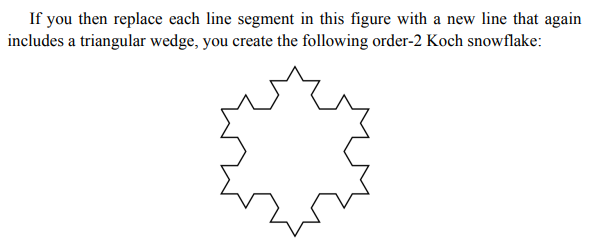
1. tower(n,A,B,C) = tower(n-1,A,C,B) + 1(A->C) + tower(n-1,B,A,C)

(Move n disks from A to C = Move the up n-1 disks from A to B + Move the lowest disk from A to C + Move the n-1 disks from B to C)

1. The b step will fail since larger disk cannot place upon smaller disk, the temporary spire is unavailable.
2. A->C A->C
3. Permutation 中文为排列
4. The first insight is to pick any character as the first one in the permutation, and perform full permutations on the remaining.

The second is to pick the first character out, perform full permutations on the remaining and insert the character to the n positions in the results.

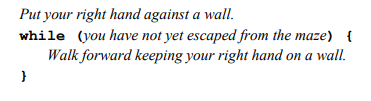
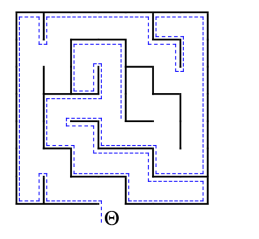
Both generate: **F(n) = n\*F(n-1) 🡪 O(N!)**

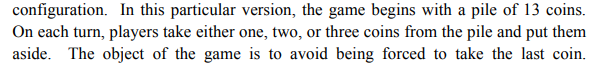
1. 4! = 24
2. Set the minimum area, if width \* height is smaller than the minimum area, terminate!
3. 
4. 

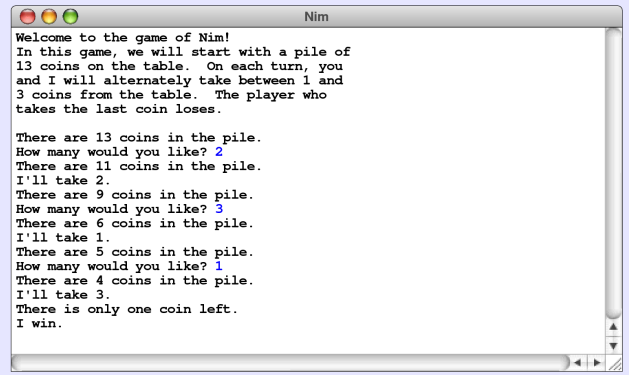
One-order segment has 12 lines, two-order segments has 12 \* 4 = 48 lines.

**LEC9:** **Backtracking Algorithms**

1. Iterative
2. Put your right hand on the wall and follow. Yes.

1. At least one sub-problem has a solution (all choice for the direction)
2. Walk out of maze (To the end) or no available direction to choose.
3. Avoid calculating the sub-problem repetitively. May cause infinite loop
4. If all directions in the for loop ends up with no solution, it should unmark and try other solution in the previous step. A successful solution may go through part of the failed trial path. Yes.
5. Judge whether the sub-problem has a solution and transfer the information to previous/next step. Also, to end the for loop.
6. Try all four directions and move by one step each time to check whether there is a solution starting from the current position.
7. 



Bad. Whatever you pick (say i), your opponent will pick (4 - i) coins, and finally leaves the last one for human player.

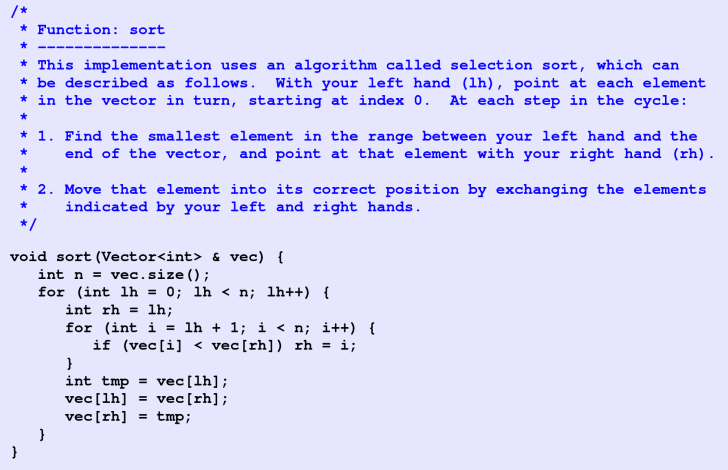
1. If nCoins is (4n + 1) then it’s bad, otherwise it’s good. (Can pick 1/2/3 each time)
2. Minimize the maximum of your opponents’ rating.

Consider your opponents, game theory

1. To apply in a more general sense. Have a grasp of the ideology.
2. Pass the information of individual step during the backtracking. Judge whether reaching the end and terminate.
3. 
4. To maximize your minimum score. The second position. -3.

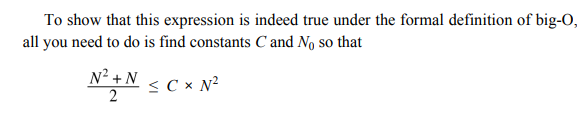
***I may jump some questions in following chapters and pick the fundamental ones.***

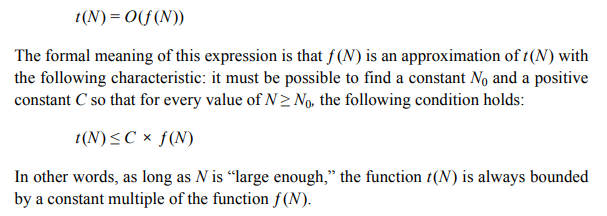
**LEC10:** **Algorithmic Analysis**

1. Iterative > recursive
2. Make the item in order
3. 

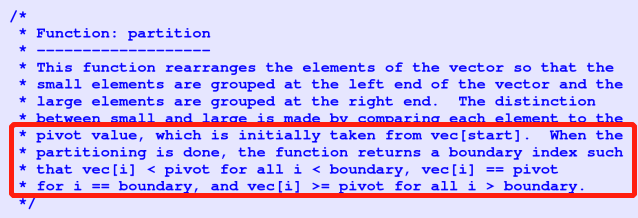
The first **lh** numbers is already in order, iterating them brings extra cost.

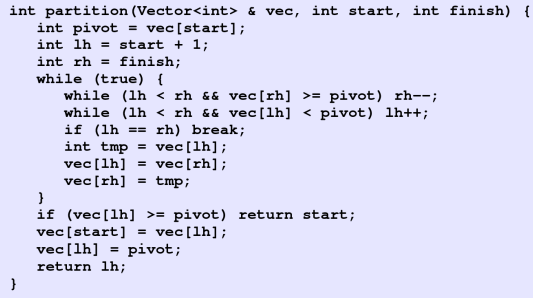
1. Selection sort is O(N²), 50 \* (1000/250)² = 800 ms
2. (N + 1) \* N / 2
3. /
4. T
5. /
6. Yes. It is just the same as O(N²)

Pick C = 2 or so

1. Yes. Selection sort f(n) = O(N²) <= O(N³) Only if it’s “smaller or equal than”.
2. No. O(NlogN) < O(N²)
3. O(N²)
4. O(1) (constant)
5. /
6. 



1. Merge compare the two arrays one by one and pick the smaller one out each time. All the numbers are iterated exactly once.
2. It doesn’t matter. The two lines just put the remaining numbers (the largest ones) at the end of the ordered array. Only one array has remains so pick any of the two is ok.
3. /
4. Can be expressed as a power of N
5. Whether the complexity is exponential
6. 



1. O(N²) (consider a completely ordered reversed array, each time you iterate the whole array but can only identify one position but not make a good partition) O(NlogN)
2. Base case 🡪 Iterative step
3. /

**LEC11:** **Pointers and Arrays**

1. Bit: The smallest memory unit, a single binary bit {0, 1}

Byte: The smallest transferable/addressable unit

Word：The biggest bytes a CPU can process each time. Depend on the system, might be 4bytes or 8 bytes

1. Binary digit
2. 2\*1024\*1024\*1024

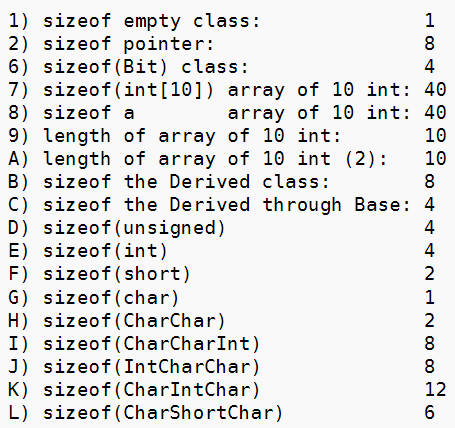
**[Here 2 GB means you have 2 \* 1024 \* 1024 available address in the system!!Each address can offer you 1 byte to store information]**

1. a.0x11 b.0x100 c. 0x6C1 d. 0xACE
2. a.23 b.100 c.204 d.4013
3. char: 1 double: 8
4. T
5. T
6. 1111111111111111111111111111001 (perform ~ then +1 正数二进制取反+1)

7 🡪 0111 🡪 (not) 1000 🡪 1001 (Negative number, add 1 in the front)

1. Stack frame + Heap + Static Data Region
2. Actual # of bytes required to store a value of type t/variable x. sizeof()

https://en.cppreference.com/w/cpp/language/sizeof



1. /
2. Left value, can place at the left hand side in the value assignment. A left value can store data and corresponds to an address in memory
3. A compact way to refer to large data structure; Convenient to dynamically allocate memory. Record the relationships of data items (linked list)
4. p1：Integer Pointer p2：Integer
5. \* 🡪 dereferencing & 🡪 referencing, obtain address
6. Pointer assignment: Assign an address of an item to the pointer, the pointer will store the address of that item and point to the new item now.

Value assignment: Assign the value to an item, its address in the memory is still the same.

1. Just remember that in stack frame the address grows from high value to low value
2. T
3. F (\* dereferencing is not always meaningful, when p is an int, it fails)
4. Pass the pointer as the argument
5. float realArray[100];

bool inUse[16];

string lines[1000];

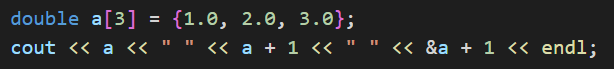
1. int squares[11]; for (int i=0; i<11; i++) squares[i] = I \* i;
2. Allocated size: the allocated size of the array during declaration

Effective size: How many elements in the array are actively in use (initialized)

1. Calculate j+3 🡪 Find the address of intArray (address of the first element) 🡪 Indexing, Add this address by 4\* (j + 3) (bytes, 4 since an integer occupies 4 bytes) 🡪 Access the value intArray[j+3] 🡪 Referencing, return the address of the value
2. array[2] is the third element of the array, it will return the value.

array + 2 will return the address of **array’s address + 2 \* size of an element**

**&array + 2 will return array’s address + 2 \* size of the array**

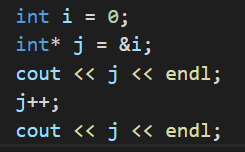




1. FF28

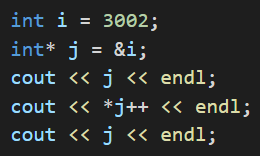
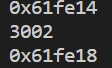
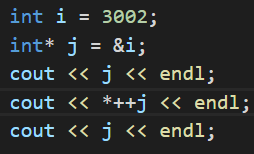
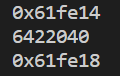
(FF00 + 5 \* 0x8 = FF00 + 0x28 = FF28, pay attention it is hexadecimal here)

1. F It will add the size of the type p is pointing to (int 🡪 4 bytes)

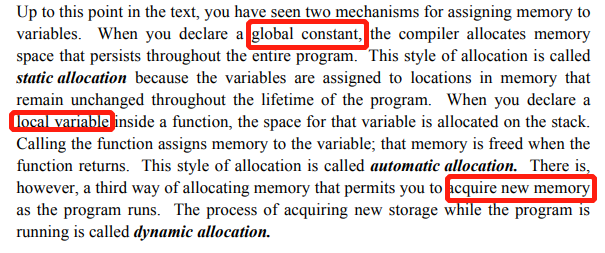
1. Dereference p, return \*p at first, and then increment p by the size of the type it points to.

If it’s \*++p, then p will increment first and then deference, return (cause a 乱码)

1. ++ is applied first. From right to left.

**LEC12: Dynamic Memory Management**

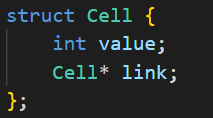
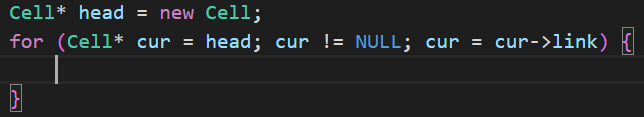
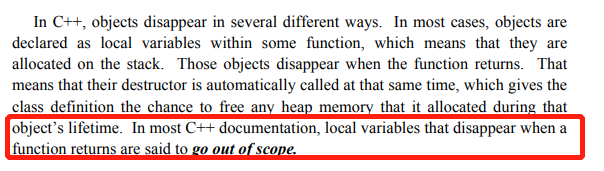
1. 
2. The pool of unallocated memory available to a program is called the heap. For dynamic allocation (new)
3. Maximize available space
4. bool\* bp;

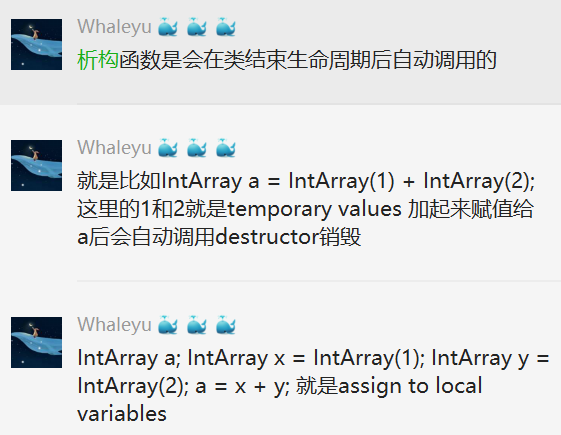
Point\* pp(3,4);

string\* name = new string[100];

1. only for the last one: delete[] name;
2. cell: the basic element unit in the list, contains the data field and the link

link: represent the relationship between the cells

1. Use **NULL** Make the last element point to NULL
2. 
3. 
4. Program fails to free the heap memory allocated.
5. F (No garbage collection in C++ but in Java)
6. Release the memory occupied by the class. Avoid memory leak.
7. ~IntArray();
8. 
9. T (Destructor will automatically be invoked when a local variable / temporary value go out of scope)



1. Initialize a larger array with bigger size when expanding, and copy elements in the original array to the new.
2. Initial capacity; char\* array store data; capacity; count – current element number
3. Allocate a new array, copy original array to the new, delete original array
4. No, If the current size reaches capacity, every push method will require O(N) to copy and expand the array, causing the average complexity O(N);
5. Reclaimed: 回收. Whenever your program creates local variables in a function. When the function returns or the block ends (e.g. for {…}), go out of scope.
6. Whenever the program uses new. When the program uses delete.
7. To track what the program is doing and the address – value relationship.
8. Add &，write the original variable address in the square, add an arrow
9. \*this (The object / class itself, just like **self** in python)
10. To test a small part of the code. The test program that checks the correctness of that module in isolation from the rest of the code.
11. Shallow copy: Dynamic variables are copied as address not the data, the two items share one heap memory,

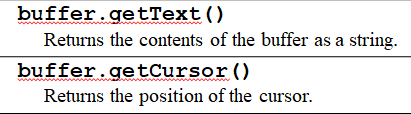
Deep copy: Dynamic variables are copied as data, Copy the heap memory at the same time

C++ uses **shallow copy** by default

1. Override the **=** operator and the copy constructor
2. The referenced variables/objects cannot be changed. (Will not change the value of that parameter)
3. The data fields of the class will not be changed. Classes that use the const specification for all appropriate parameters and methods.
4. You can use 等比数列求和公式 1 \* (1 – 0.5^k) / (1 – 0.5) = 2

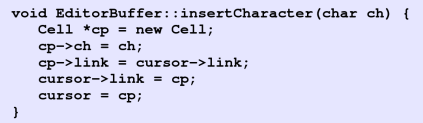
Or you can use mathematical induction to explain the sum is (1-1/2^k)

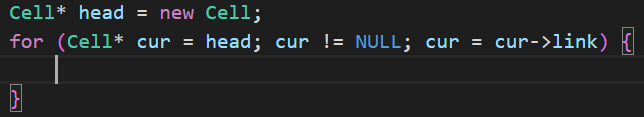
**LEC13: Efficiency and Representation**

1. F (Time + Space)
2. what you see is what you get
3. Use command line to edit texts
4. I: .insertCharacter(ch) J: .moveCursorToStart() E: .moveCursorToEnd() F: .moveCursorForward() B: .moveCursorBackward() D. .deleteCharacter()
5. 
6. I and D: move the elements of the inserted/deleted element afterwards by one step, causing many element movements.
7. Before (peek 🡪 end)：DCBA after：EFGHIJ 想象从中间掰开
8. Between the top elements of two (before & after) stacks.
9. J and E
10. Add a possible position for the cursor. If there are n cells, there will be (n+1) possible positions for the cursor.

(Point to Cell A then the cursor will be after Cell A

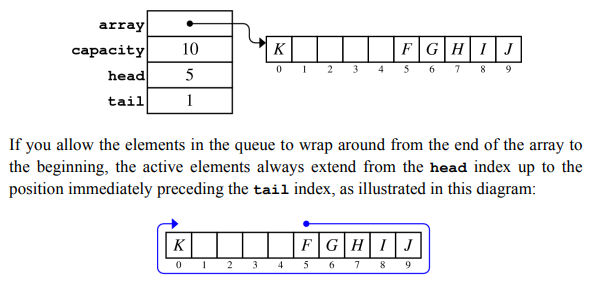
1. At the beginning. When there is no element, the cursor is at the beginning
2. Create a new cell 🡪 Assign value to the new cell 🡪 Let the new cell link to the next cell of the cursor 🡪 Let the cursor’s cell link to the new cell 🡪 Let the cursor point to the new cell



1. Start（dummy）—H—E—L—L—O The cursor points to the first L
2. Start（dummy）—H—E—L—X—L—O The cursor points to X
3. Go through the linked list
4. 
5. J and E. The linked list cannot access the previous cell and the last cell directly, must traverse the list.
6. Time-space tradeoff is the tradeoff between time and space三人行，必有三人
7. Double linked list, add a link to the previous cell
8. Time-space tradeoff, more space consumption. Buy a new computer maybe还

**LEC14: Linear Structures**

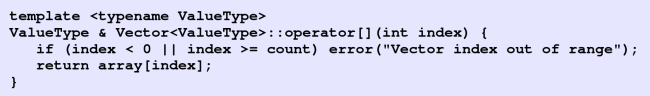
1. More generality, can accommodate different types. No overloading one by one
2. template <typename placeholder>
3. /
4. An integer array. An integer (store the size). Two integers representing the indices of head and tail.
5. A ring structure buffer. Use head and tail to mark the beginning and the ned. Applied in Array-based queue



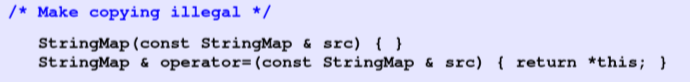
1. Whether head==tail

Compare the current size (tail – head + capacity) % capacity with capacity.

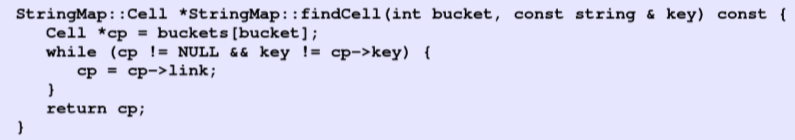
1. /
2. The technique of using remainders to reduce the result of a computation to a small, cyclical range of integers is an important mathematical technique called modular arithmetic. Let index inside the range 0 ~ capacity - 1
3. Tail may be smaller than head and generate a negative number (like the above image) Should be (tail – head + capacity) % capacity
4. /
5. Whether the head / tail pointer is NULL / see count
6. Typename & Classname::operator[](Typename);



**LEC15: Maps**

1. Order the Key. Binary search each time
2. Inserting an element in a vector costs O(N) to move elements right
3. Two-dimension
4. The first character represents NULL (special), the program will ignore
5. Bucket is the element of an array, representing one hashcode’s result. That array contains all the key-value pairs with a specific hash code.
6. Different keys generate the same hash code and fall into the same bucket.
7. 

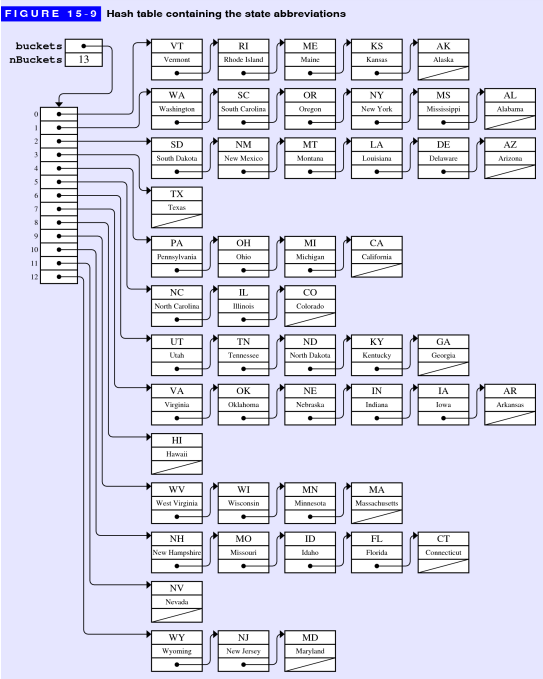
The default copy constructor will help you copy the local vector (not in heap)

1. 

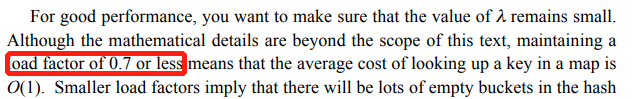
Given the index of the bucket and the key, find whether there is the key in the bucket.

1. The key has no relationship with the string str. You cannot access the key with the string next time.
2. Yes. But all keys with go to one bucket (need to deal with collision) and complexity becomes O(N)
3. AZ and DE

The first collision happens when two keys generate a same hash value and put into the same bucket. So we need to find which bucket obtains two elements the first. Following the alphabetic order (check from tail), AK 🡪 AL 🡪 AR 🡪 AZ 🡪 CA 🡪 CO 🡪 CT 🡪 DE (collision with AZ!)

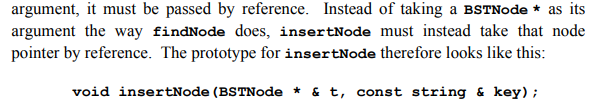


1. Raise bucket number (space), reduce the probability of collision and improve insert & find efficiency (time).
2. # of keys / # of buckets
3. 0.7

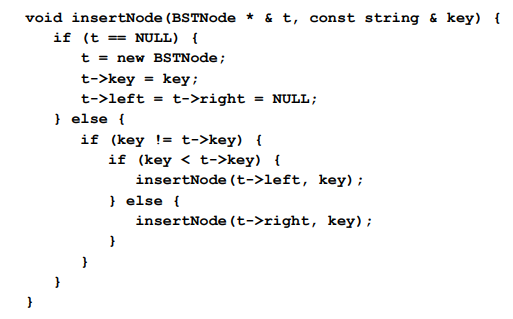


1. Change the number of buckets and push the elements inside the hash table again (with new bucket index maybe)
2. 
3. Use two-dimension look-up table

**LEC16: Trees**

1. Every node requires a collection to store the child nodes. No loop/circle formed.
2. Family tree, C++ class inheritance…
3. /
4. Root: Henry VII height=5
5. Node\*, the pointer
6. /
7. At most two child nodes for each node. Left child node is smaller than right.
8. 

Since the current node may be modified. (The root node changes)

1. /
2. /
3. /
4. Inorder Processing the current node between the two recursive calls represents an inorder traversal. No matter what order the nodes are inserted into the tree, in-order traversal will always generate the nodes in order (from small to large)
5. The height difference between any node’s two child trees is no larger than 1
6. a. yes b. no (For left child of the root node: left height is 2 and right height is 0; For right child of the root node: left 0, right 2) c. yes
7. F. They can still work correctly but with complexity O(N)
8. Difference between the left child tree and right child tree
9. A=0 E=2 O=1 I=0 Y=1 U=0
10. Not covered in this course, learn more about it in CSC3100 or CSC4120
11. T
12. /
13. 
14. Use the rightmost node in left child tree or leftmost node in right child tree to replace the removed node
15. 1521-2193-1604-3169-2708-1861

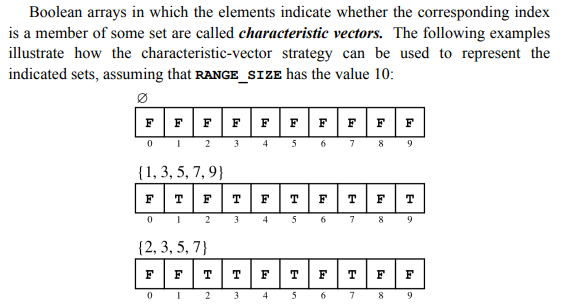
(insert 1521 as the last element and move up twice)

1. The representation is different, a heap is an array structure used to simulate the partially ordered trees (tree structure). But the mechanism is the same.

**LEC17: Sets**

1. T
2. F
3. Empty set, Integer set, Natural number set, Real number set
4. Belong to / not
5. {0,1,4,9,16,25,36,49,64,81,100}
6. {x | x=9 \* i, i∈N, 0 ≤ i ≤ 9}
7. ∪ ∩ -
8. {a, b, c, e} {a, c} {b} {b}
9. A proper subset does not include the set itself
10. {i | i∈N} {i² | i∈N}
11. /
12. （A∩B）∪（A∩C）∪（B∩C）-（A∩B∩C）

(A∩B∪C-（A∩C）∪（B∩C）) ∪ (A∩B∩C)

1. /
2. /
3. Map, each key element is unique and no repetition
4. 
5. No repeated element, the character must be in ASCII with length smaller than 128
6. F T F F T F F F F T
7. Number 0-9 isnum(char);
8. YES
9. a. 0000000001001001

b. 0100100011111111

c. 0100100010110110

d. 0000000000000000

e. 1011011110110110

f. 0100100000000000 (only x[i] = 1 y[i] = 0 can have bit 1)

g. 1011011100000000 (only x[i] = 0 y[i] = 0 can have bit 1)

h. 0000000000001111

i. 0100001001001000

j. 0000000001001000

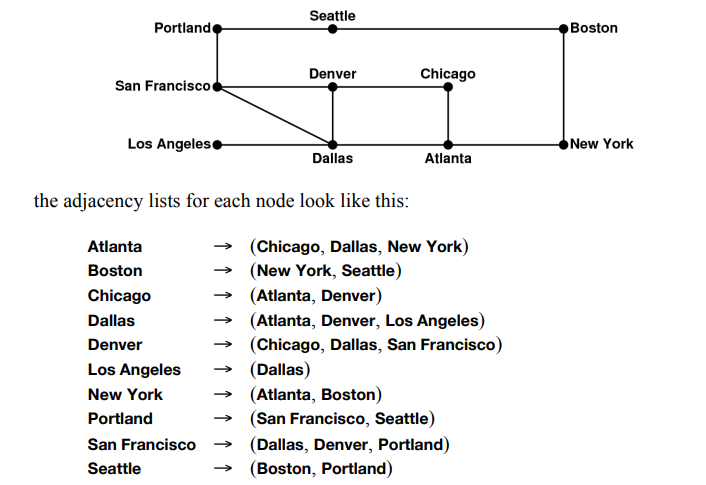
1. Octal: 44111 (Check 3 bits together) Hexadecimal：00FF (Check 4 bits together)
2. a. x & mask b. x |= mask c. x &= ~mask d. x ^= mask
3. unsigned(1) << k

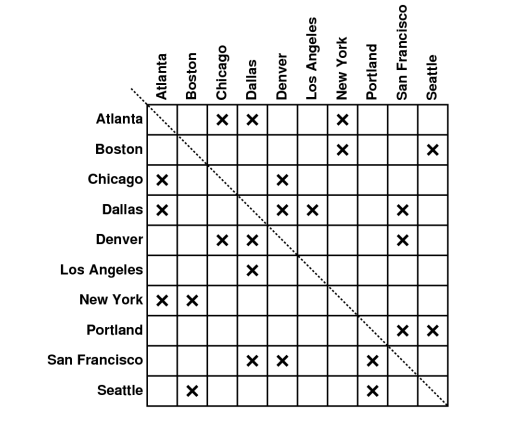
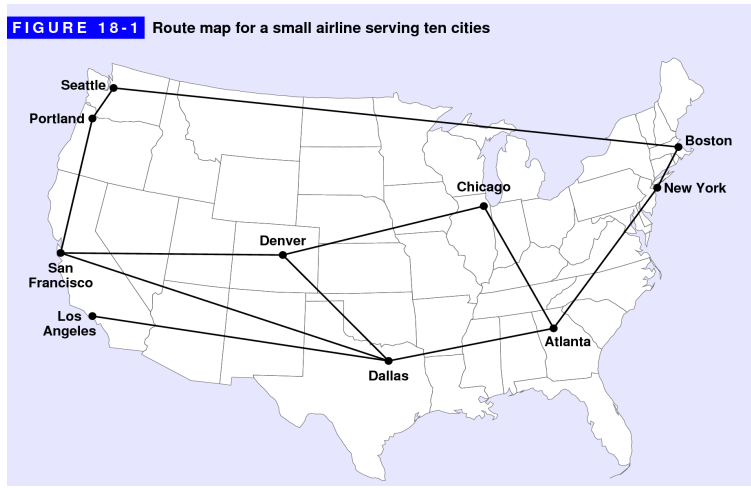
**LEC18: Graphs**

1. /
2. T
3. Whether an edge has direction
4. Add two edges between two nodes
5. /
6. Neighbor: How many nodes are one edge around the node

Degree: How many edges are connected with the node

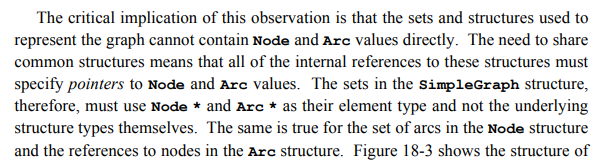
1. Whether starting from a node can reach any other node
2. T
3. Vertex, Edge
4. Define in this way:
5. Define in this way:

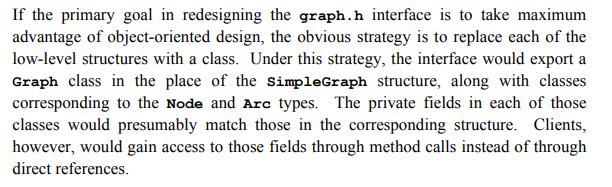


1. Define in this way: 
2. Density (# of edges / # of nodes)
3. Whether the graph is sparse (adjacency list) of dense (adjacency matrices)
4. There are different traversal order
5. To save space. Use a compact way to represent large data structure and easy to reference the neighboring nodes / arcs.
6. DFS & BFS
7. 

DFS: Atlanta 🡪 Chicago 🡪 Denver 🡪 Dallas 🡪 Los Angeles 🡪 San Francisco 🡪 Portland 🡪 Seattle 🡪 Boston 🡪 New York

BFS: Atlanta 🡪 Chicago 🡪 Dallas 🡪 New York 🡪 Denver 🡪 Los Angeles 🡪 San Francisco 🡪 Boston 🡪 Portland 🡪 Seattle

1. 



Also Check 18.5 explanation

1. Node must include a name and set<ArcType \*> arcs. ArcType must include a NodeType \* start and a NodeType \* finish.
2. Choose the most beneficial behavior in the (every) current step but may not be globally the most beneficial.
3. <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>

Learn more about it in CSC3100 or CSC4120

1. Check textbook
2. Check textbook

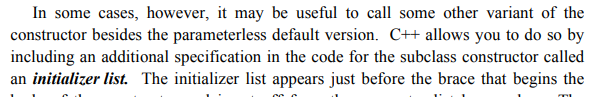
**LEC19: Inheritance**

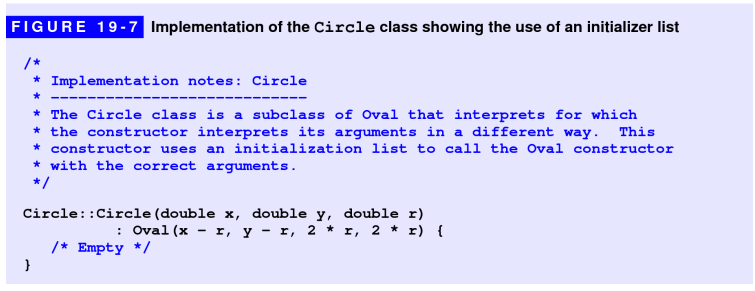
[使用一个基类类型的指针或者引用，来指向子类对象，进而调用由子类复写的个性化的虚函数，这是C++实现多态性的一个最经典的场景。]

1. class Sub: public Super{};
2. T 

(It can be non-template and no specification)

1. F (The parameter lists may be different)
2. Allow sub-class to override super-class method. Dynamic banding. When invoking methods, choose the pointer’s pointing type instead of pointer’s initialization type.
3. A pure virtual method will never be executed in the super-class but in sub-class. To guarantee security. No need to write **virtual** in the sub-class.
4. Add = 0 at the end
5. An abstract class cannot be instantiated (but can define its pointer) and can only serve as the parent class of other classes. Yes, can provide implementation or leave the method empty.
6. Assign value of sub-class to super-class. Slicing will throw off all fields in sub-class but not in the super-class.
7. Pointers. There may be slicing if storing values.
8. draw(gw)
9. Protected section is visible/available to sub-class but not to clients. Public section is visible to all. Private section is only visible to the class itself.
10. A different version of the superclass constructor.



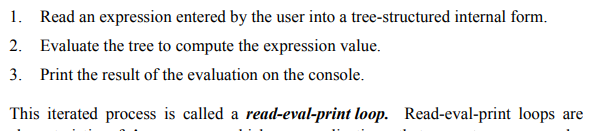
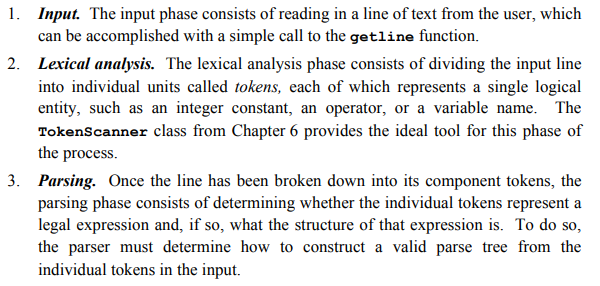
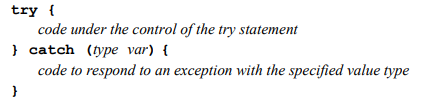


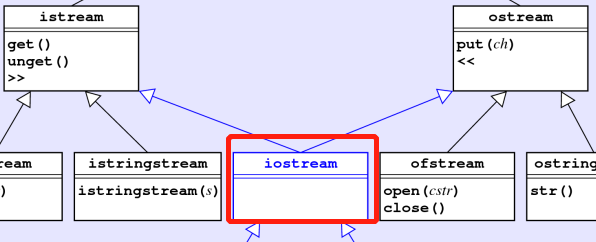
Use sub-class’s constructor to initialize super-class

Appear in sub-class’s constructor

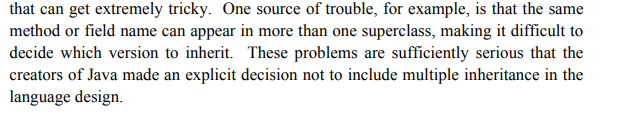
1. Interpreter: Execute the code line by line directly (Python).

Compiler: transform source code to machine code (binary form) and generate an exe file to execute (C++). Not suitable for cross-platform but has a much higher efficiency than Interpreter.

1. 
2. (No need to know)
3. An error
4. 
5. /
6. /
7. /
8. /
9. /
10. /
11. /
12. /
13. /
14. /
15. /
16. /
17. /
18. /
19. /
20. /
21. Classes can inherit behavior from more than one superclass



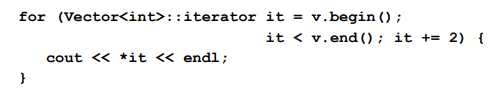
1. F



**LEC20: Strategies for Iteration**

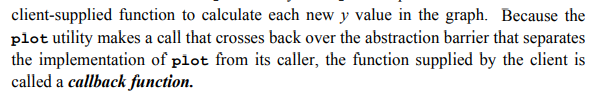
1. for (Set<int>::iterator it=primes.begin(); it!=primes.end(); it++)

{cout << \*it << endl;}



1. Return \*it first and then it++; Return the value that iterator points to. Then increment the iterator by 1, now it points to the next element of the collection.
2. T
3. F
4. The ForwardIterator class combines these capabilities and supports both reading and writing.
5. F
6. Every function is stored in memory and therefore has an address.
7. The first one is a function that receives a string and returns a char\*

The second one is a pointer to the function, where the function receives a string and returns a char.

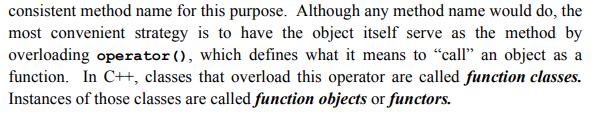
1. bool (\*fn) (int, int);
2. 

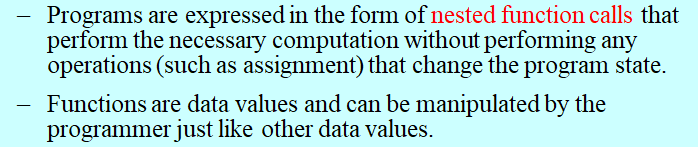
The function serves as the parameter of other functions

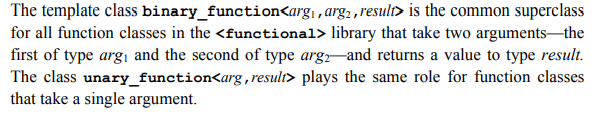
1. 
2. Function object: any object that overloads the function-call operator.

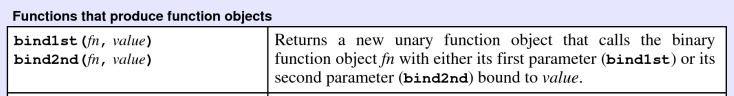
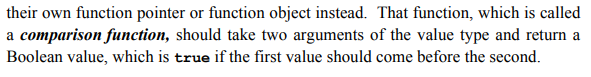
Function pointers: a pointer pointing to a function

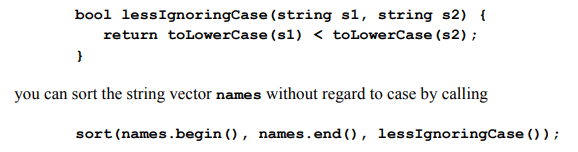
1. Operator()



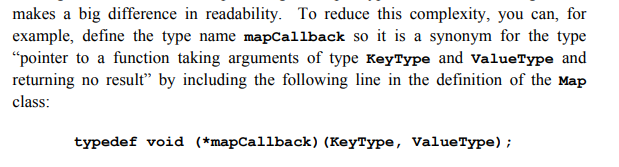
1.  two iterators.
2. 
3. F

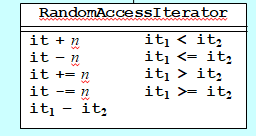


1. 
2. Imitate this: 
3. 



1. Define proc as a type “pointer to a function taking no argument and return no result”



1. 
2. F Iterator acts as pointers