Compilation process:

source file →compiler → object file →
library file → *linker →executable file*

Variables:

Naming Convention: Begin with letters or underscore；Only letters, digits, underscore are allowed, no space or reserved words; case sensitive.
Name and type are fixed.
reserved words: explicit, extern, friend, goto, inline, mutable, new, protected, register, short, signed, sizeof, struct, this, throw, typedef, typeid, typename, union, using, volatil
Operators in C++ appear between two sub-expressions (=**operands**).
Operator that take two operands -> binary operator.
% applies **only** to integer operands.
type cast: convert one type to another, by using type name as a function.
运算优先级: unary – operators **first**.
shorthand assignment +=;
++x returns **incremented** value;
x++ returns original value, then ++
relational operators !=;
logical operators && (and), ||(or), !
3 * 6.0=18.0, int;
0<=x<=9 not legal(∵2 compares)

Function&Libraries

优点:reuse; abstraction (easier to read); decomposition; (top-down design)
parameters 形参; arguments 实参
Predicate functions : return bool
Procedure functions : return void
overloaded(∴多态)- Define several different functions with the same name. The pattern of arguments required for a particular function is called its *signature*.
Default values may appear only at the **end** of argument list.
client(code use library)
interface(info share by both)
implementation(code in lib)
```
#ifndef mylib_h #define _mylib_h
#endif
```
Interface Properties: unified (similar theme), simple (hide complexity), sufficient (meet demand), flexible (general enough), stable
The *interface* represents the information that the client and the implementation share.

String:

A sequence of characters(in C). 双引号包characters->C string, characters stored in an array of bytes, 终止 char 是 null byte \0 (其ASCII value=0). Index begins at 0.
`cin >> line;`
read a string up to next spaces(两端被空白 char 分隔), cannot read entire line.
`getline(cin, name);`
均 passed *by reference* stream vars 不能 copy.
#include <**string**>: C++ library; <**cstring**>: C string library used in C++; <**string.h**>: C string library used in C; **"string.h"**: C string library; **"cstring.h"**: error unless defined yourself.
C++ string: abstract data type(class&object)允许变 individual character
Receiver: object to which a message is sent.e.g.receiver.name(arguments);

C string literal vs C++ string

+ cannot be applied to string literals(C style).
`string str="nb"+", "+string("cp");`
//Flase, quoted strings are C strings
When compiler can determine that what you want is a C++ string , convert C string literals into C++ string object automatically. e.g.
`string str = "hello, world";`
`str[index]&str.at(index)`
difference: at 有 range-checking.
单引号：字符常量；双引号：字符串常量
Both C and C++ use ASCII as there encoding for character representation. The data type **char** therefore fits in a single **8-bit byte**.
`isdigit(7)` false; `isdigit('7')` true;
`toupper(7)` ->7
string::npos -> the end index
primitive string -> C++ `string(cstr)`
C++ string -> primitive `str.c_str()`

Stream

insertion operator << : takes an output stream on the left and an expression of any type on its right. Returns the output stream as its value. (<< could be chain infinite long)
extraction operator >> : stream >> variable (data from stream read into variable)
manipulators (<iomanip>) appear after cout << / cin >>

Text Files vs. Strings 1)Information stored in a file is permanent. The value of a string variable persists only as long as the variable does. 2)Files are usually read sequentially.
read data from a text file
1)construct ifstream: ifstream stream;
`stream.open(filename)`
2) call open (filename must be a **c string iteral**):
`open(filename.c_str())`
3)read data: get or getline
`.fail()` –> detect "open" success or not
`stream.get()` –> returns the next character value as an int, which is **EOF(-1)** at the end of the file
`stream.get(ch)` –> reads the next char into that variable,retrun **bool value**
`getline(stream, string)` – free function, both two parameters are taken by reference, sets **fail indicator** when read past end.
4)close file –> stream.close()
cin, cout, cerr – standard file streams
<< >> return the stream argument **by reference**
1)enable chaining behavior
2)avoid copy: because stream variables cannot be copied, the `ostream` argument must be passed *by reference*.

Collection

优点: 1)Simplicity 2)Flexibility (free to change implementation) 3)Security
common properties:
1)represent abstract data types –> hide detail
2)declaration need type parameter(除 Lexicon)
3)declaration invoke constructor
4)= operator copy whole structure
5)pass *by reference*- avoid 4)
Vector – implementation is based on array – like list in python
Grid - `Grid<type>` grid(nrows, ncols);
Stack - FIFO, pop/push
Queue – LIFO, enqueue/dequeue (cannot loop through these two)
Map – `Map<key type, value type>`, include map and hash_map,像 py 的词典 (.get 却没 key, return default value of type)
Set – natural comparison order with its value, each value appears once
Lexion = set<string>: A set of words with no associated definitions. 1)space-efficient 2)can use contains prefix 3)fast&easy for iterators in alphabetical order

Class

Structures (C style)
```
struct typename {
    declarations of fields };
```
creates a *type*, not a *variable*. internal space of class called **field**; internal variable called **member**, select with **dot operator** (.)
Object-oriented languages:
表示 most data structures as objects
```
class typename {
    public: prototypes of public methods
    private: private instance variables;
prototypes of private methods }
```
getter: retrieve the values of instance variables
setter: set the values of instance variables
most class is designed immutable for instance after it has been created
constructors –same name as the class, *multiple constructors* when the constructors have different parameters sequences. The constructor that takes no arguments - ***default constructor***.
`class name::method name` let compiler know which class the method definition belongs.
-Public Section: available to clients of the class.
-Private: restricted to the implementation.
-Protected Section: available to subclasses but inaccessible to clients. (friend)
Overloading Operators (two ways to define)
–Defining an operator as a class method: the operator is part of the class and has *free access* to the *private* instance variables and methods.
–Defining an operator as a free function: produces code that is easier to read but means that the operator function must be designated as a **friend** to refer to the *private data*.
```
ostream & operator<<(ostream & os,
Point pt) {
   return os << pt.toString();}
```

Introduction Recursion

Recursive Paradigm
two things need to do:
1)simple cases – case that can be solved without recursion(move a disk)
2)recursive decomposition:
breaks each instance of the problem into simpler sub-problems 但是所有外部条件需和以前一样, which you can then solve by applying the method recursively.

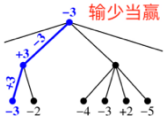
Pitfalls checklist: 1)begin by check for simple cases? 2)solved sim_cases correctly? 3)recursive decomposition simplify problem? 4)finally reach sim_cases? 5)recursive call's situation is the same as original? 6)solutions to the recursive sub-probs provide complete solution to original?
Mutual Recursion – two function recursively call each other, finally back to original function
str.substr(st_pos,len)
if len =0, return empty string;
if len 不填, return substr till the end

Recursion Strategy

inclusion-exclusion pattern
(Sub-sum question)
solution 中可能包含||不包含某一情况,故每一点都有两个子分支
simple cases: set.isempty()
recursive decomposition:
rest = set - set.frist();
return subsetSumExists(rest, target) ||
subsetSumExists(rest, target - element);
Permutation (of N object)
The set of all permutations of a string can be generated by taking every possible starting letter and then appending all possible permutations of the remaining N – 1 letters.
一个一个位置的排
take "A" first –> then consider permutation of "BCD". Or take B first

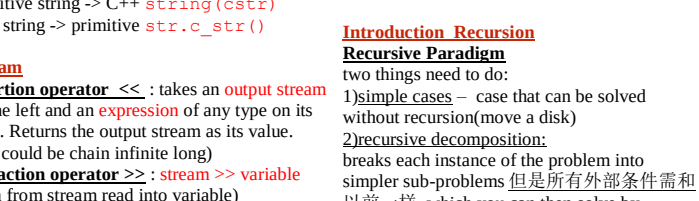
Backtracking algorithm:

(maze) At each square, first check if simple cases. If not, mark current square and solve the maze after 移一步 in each open direction. Solvable or moving to next direction. If all directions 走光->fail->pass back to a higher level of the recursive process.
simple cases: the square is outside the maze or marked
right-hand rule 若在开始位置周围有一个循环则会 endless circle around loop ∴通过 **mark** squares 解决(Unmark squares when backtrack->∴correct path is marked when reach the exit)
Maze's **simple case**: 1)object outside the maze; 2)no directions left to try.
enumerated type:定义一新 type where 可能值从 a set of possibilities 中选出
recursion & concurrency (same time) recursion 好处: simplify the bookkeeping. Each level of the recursive algorithm considers one choice point. The historical knowledge of what choices have already been tested and which ones remain for further exploration is maintained automatically in the execution stack.
```
int findGoodMove(intnCoins) {
   int limit = (nCoins <
MAX_MOVE) ? nCoins : MAX_MOVE;
   for (int nTaken = 1; nTaken <=
limit; nTaken++){
       if (isBadPosition(nCoins
- nTaken)) return nTaken;}
        return NO_GOOD_MOVE;}
bool isBadPosition(int nCoins) {
   if (nCoins == 1) return true;
   return findGoodMove(nCoins) ==
NO_GOOD_MOVE;}
```
Game tree ->
Minimax algorithm: choose the move that leaves your opponent with the worst possible best move.



standard complexity classes: constant; logarithmic(logN); linear; NlogN; quadratic; cubic(e.g. matrix multiplication); exponential 2^N(e.g.Hanoi Tower)
Selection sort: 从左边每个位置开始，找从它到最后一个之间最小的数，换位
Merge sort: 1. Divide the vector into two halves. 2. Sort each of these smaller vectors recursively.3. Merge the two vectors back into the original one.
Quick sort: 保证每次 pivot 左边都比他小，右边比他大

Pointer&array



minimum addressing unit:bit (1 byte = 8 bit)

Sort	数据对象	稳定性	Time complexity Avg	worst
Bubble	array	✔	n^2	
Select	array	×	n^2	
	llist	✔		
Insert	Array llist	✔	n^2	
Heap	array	×	nlogn	
Merge	array	✔	nlogn	

(无序,有序)从无序区通过交换找出max元素放到有序区前端。
(有序,无序)在无序区里找一个min元素跟在有序区的后面。对array: 比较得多，换得少。
(有序,无序)把无序区的1st元素插入到有序区合适的位置。对array: 比较得少，换得多。
(最大堆,有序)从堆顶把根卸出来放在有序区之前，再恢复堆。
把数据分为两段,把这两段再分别递归排序,(直到每段只有一个)组合成原来的K=logN, O(merge) = N (N times merge, each is const
(小数,枢纽元,大数) A: random, W: sorted

The Allocation of Memory to Variables

- When you declare a variable in a program, C++ allocates space for that variable from one of several memory regions.
- One region of memory is reserved for variables that persist throughout the lifetime of the program, such as constants. This information is called ***static data***.
- Each time you call a method, C++ allocates a new block of memory called a ***stack frame*** to hold its local variables. These stack frames come from a region of memory called the ***stack***.
- It is also possible to allocate memory dynamically, as we will describe in Chapter 12. This space comes from a pool of memory called the ***heap***.
- In classical architectures, the stack and heap grow toward each other to *maximize the available space*.

Pointer: addresss in memory 优点: 1) allow you to refer to a large data structure in a compact way (*call by pointers*); 2) make it possible to reserve new memory during program execution (*dynamic allocation*); 3) can be used to record relationships among data items (*linked structures*).
value saved in that address could change, but address itself no change.
Lvalue: Expression that refers to an internal memory location capable of storing data.有地址,且地址不会变, can appear on both side. If cannot assign a anew value to it, it is not a lvalue.
Rvalue:无地址, can only appear on the right side.
Point Decalration:
***后接 pointer, (dereference) **return the value** of that pointer address
&后接 lvalue, (address of) **return the address** that store value 下例中：(*px = x)
`int x = 2.5; int* px = &x;`
-When use the same names for *parameters* and *instance variables*, must use the keyword **this** (defined as a pointer to the current object) to refer to the instance variable.
Call by reference(=pass address) sharing relationship between a parameter variable and 对应 argument in the caller. 允许被调用的函数能够访问调用函数中的变量。Eg.swap
优点: 1)pass info in both directions;2)eliminate the need to copy an argument(if a big object)
Return by reference 是被引用的变量的别名。
Array vs. Vector differences:
1)only operation is selection using [] 2)Array[] not check that the index is in range, (if out, you get random number) 3)length is fixed 4)array don't store length, need extra int value represent effective size –
`sizeof arr / sizeof arr [0]`
声明:`int list[3] = {1,2,3};`
`int *list = new char[5]`
–An array is represented internally as a *pointer points to its first element*. Passing an array as a parameter does not copy the elements. The function stores the *address* of the array in the caller. If the function changes the values of any elements of an array passed as a parameter, those changes will be visible to the caller.
array share its address in function parameter

```
double array[5];
double * dp = array; 那么
array[i] ≡ *(array+i) ≡ *(dp+i)
dp+i ≡ array+i ≡ &array[i]
```

++p 先自增再赋值，p++ 先赋值再自增.

```
int arr[] = {1, 2, 3, 4};
int * p = arr;
int a = *p++;
//a = *(p++); a = *p; p = p + 1;
int b = *++p;
//b = *(++p); p = p + 1; b = *p;
output:a = 1, b = 3
```

Dynamic Memory Management

Dynamic Allocation:
new operator allocate memory on heap
two type of allocation:

1)a single value: `int* ip = new int;`
2) an array of values: `new type[size];`
e.g. `int arr = new int[10];`
delete operator -> free memory
delete ip / delete[] array;
Destructors: ~Class name,
free allocation memory
-stack variable, auto call destructor when function *return*
-heap variable, must call **delete** operator explicitly
Memory leak: fail to free the heap memory you allocate and no longer needed.
Unit Testing: Use `assert` macro from the `<cassert>` library. If true, continue.
Copy: Assignment Operator & copy constructor

```
const type & type::operator=(const type & rhs)
type::type(const type & rhs)
```

Shallow Copy–allocates new fields for the object and copies information from original. Dynamic array is copied as *address*, not data.
Deep Copy–also copies contents of the dynamic array and creates two independent structures.
Copy constructor

```
type::type(const type & ms){
deepcopy(ms); };
void CharStack::deepCopy(const
CharStack & src) {
   array = new char[src.count];
   for (int i = 0; i < src.count;
i++) {array[i] = src.array[i];}
   count = src.count;
   capacity = src.capacity;}
```

capacity

Constant

Constant definitions. Adding the keyword `const` to a variable definition tells the compiler to disallow subsequent assignments to that variable, thereby making it constant.
`const double PI = 3.14159265358979323846;`

Constant call by reference. Adding `const` to the declaration of a reference parameter signifies that the function will not change the value of that parameter. This guarantee allows the compiler to share the contents of a data structure without allowing methods to change it.
`void deepCopy(const CharStack & src);`

Constant methods. Adding `const` after the parameter list of a method guarantees that the method will not change the object.
`int CharStack::size() const`

Efficiency:
(1)Array model: simplest, allocate the array storage dynamically and to expand the array when full. 区分 allocated size(=**capacity**)和 effective size(=**count**).
实例变量 size(length), capacity, cursor position(from 0 to =length); *array
(2)two-Stack model: characters before cursor store in stack called `before` stack and chars after the cursor stored in `after` stack. *The ones close to cursor are near the top of stacks.*
实例变量 CharStack before; CharStack after;
(3)Linked List model: allocate an extra cell (*dummy cell*) at the beginning. NULL marks the end. Represent the position of the cursor by pointing to the cell before the insertion point.
实例变量 Cell*start, *cursor
Time-Spae Tradeoff: Using a doubly list [全都 O(1)], effectively doubles the space cost.
改进: reduce the overhead by storing several characters in a single cell.

Buffer operations	array	stack	list
moveCursorForward	O(1)	O(1)	O(1)
moveCursorBackward	O(1)	O(1)	O(N)
moveCursorToStart()	O(1)	O(N)	O(1)
moveCursorToEnd()	O(1)	O(N)	O(N)
insertCharacter(ch)	O(N)	O(1)	O(1)
deleteCharacter()	O(N)	O(1)	O(1)

Linear Structure
`template <typename ValueType>`
Polymorphism: 将相同代码用于多种数据类型

	基于数组	基于链表
instance variables	ValueType *array; int capacity; int count;	Cell *list; (指向表头) int count;
构造函数	array = new ValueType[capacity]; count = 0;	list = NULL; count = 0;

Array-Based Queue: ring buffer enqueue-tail:(tail+1)%capacity; dequeue-head: (head+1)%capacity; full-size()==capacity-1;emtpy- head==tail; size:(tail + capacity - head) %capacity
Linked-List Queue: no dummy cell is required.
Vector: Redefining **operator[]**

```
template <typename ValueType>
ValueType&Vector<ValueType>::operator[]
(int index) {
    if (index < 0 || index >= count)
{Error("Vector selection index out of
range");}
    return elements[index];}
```

Return by reference: return value is shared with the caller rather than copied. Result into an *lvalue*. never return by reference with a value live in the stack frame of the current method.
`ValueType & Vector<ValueType>::
operator[](int index);`

Map
vector based: private:
Implementation Strategies (**put get**)
1) linear search: O(N) - both **put**/**get**
2) binary search: (*vector sorted by key*) O(logN) for **get;** O(N) for **put**(需移动其之后的 element)
3) Lookup Table(grid): O(1) for both **put**/**get** at a cost in memory space
hashing: use the key to figure out where to look – hash code is specific for string characters
same string, same hash code
collisions: different strings have same hash code. Different keys fall into same bucket.
Hash Functions
1) inexpensive to compute
2) distribute keys as uniformly as possible, minimize collisions
- The correctness of the algorithm is not affected by the collision rate.
Bucket Hashing: use *hash code* for each key to select an index into an array.
bucket – one element of the array (linked list)
convert the hash code into a bucket index:
`index = hashCode(key) % nBuckets`
load factor - # of keys / #of buckets: closer to 1, less collision, achieve O(1) 越小越好
`struct Cell { string key; string value; Cell *link; };`
instance variable: `Cell **buckets;` buckets 每个元素指向键值对*链表*中 first element 的指针，buckets 是指向每个单元指针的指针。

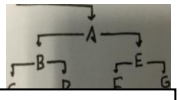
Tree
Tree as a recursive structure: a pointer to a node; a node is a structure that contains some number of trees.
BST:1) each node has two subtrees 2) key values are unique 3)根节点的值大于左子树所有节点的值、小于右子树所有节点的值
insertNode: argument pass *by reference*, because Node need to be able change value.
recursive strategy in BST:
1)compare root </=/> target ? 2)recursively call

```
struct TreeNode {
   string key;
   Vector<TreeNode *> children;};
Class TreeNode {
Private:
   *TreeNode root;}
```

BST to L/R substree

```
Node *findNode(Node *t, string key) {
   if (t == NULL) return NULL;
   if (key == t->key) return t;
   if (key < t->key) {
      return findNode(t->left, key);
   } else {
      return findNode(t->right, key);}}
```

删除节点: replace with the rightmost node in left subtree or the leftmost node in right subtree.一个 node 是 NULL, replace with non-NULL.
Traversal Strategies:
preorder: A B C D E F G
inorder: C B D A F E G
postorder: C D B F G E A




```
void preorderTraversal(Node *t) {
   if (t != null) {\\cout 位置的改变
      cout << t->key << endl;
      preorderTraversal(t->left);
      preorderTraversal(t->right);}}
```

A binary tree is ***balanced*** if the height of its left and right subtrees differ by at most one and if both of those subtrees are themselves balanced.
Priority Queues: elements dequeued on priority. Array, linked list:O(N), 其他 algorithm: O(log N)
partially ordered tree: complete, (i.e., completely balanced, each level of the tree is filled as far to the left as possible); root node has higher priority than the root of either of its subtrees.

Set
Implementing Sets
1)Hash tables: average O(1) for adding a new element, primary disadvantage is that do not support ordered iteration.
2)Balanced binary trees: offer *O*(log*N*) performance on the fundamental operations, but do make it possible to write an ordered iterator.
high-level operations – union, intersection, difference, subset, and equality
bitwise operator
(~)NOT[*complement*], (|) OR[*assemble*], (&) AND[*masking*], (^) XOR[*flip*],
(x << n) left shift, 0 replace, for (x >> n)
1) x is a signed type, arithmetic shift, leading bit in x won't change, if x lead by 1, fill in 1's
2) x is unsigned type, logical shift, missing digits are always filled with 0s

Graphs
graph consists of a **set** of nodes and arcs

```
struct SimpleGraph{
    Set<Node *> nodes;
    Set<Arc *> arcs;
    Map<string,Node *> nodemap; }
struct Node{string name;
Set<Arc*> arcs;}
struct Arc{Node*start;
Node*finish; double cost;}
```

DFS Using Recursion

```
void depthFirstSearch(Node *node) {
   Set<Node *> visited;
   visitUsingDFS(node, visited);}
void visitUsingDFS(Node *node,
Set<Node *> & visited) {
   if (visited.contains(node)) return;
   visit(node); // 或者别的操作
   visited.add(node);
   for (Arc *arc : node->arcs)
{visitUsingDFS(arc->finish,visited);}}
```

BFS Using a Queue

```
void breadthFirstSearch(Node *node) {
   Set<Node *> visited;
   Queue<Node *> queue;
   queue.enqueue(node);
   while (!queue.isEmpty()) {
      node = queue.dequeue();
      if (!visited.contains(node)) {
visit(node);visited.add(node);
      foreach (Arc *arc in node->arcs)
{queue.enqueue(arc->finish); }}}}
```

Dijkstra's Algorithm
–implement with priority queues.
–keep track of all nodes to which the total distance has already been fixed. Distances are fixed whenever you dequeue a path from the priority queue.
Kruskal's Algorithm 最小生成树
1)Start with a new empty graph with the same nodes as the original one but an empty set of arcs. 2)Sort all the arcs in the graph in order of increasing cost.3)Go through the arcs in order and add each one to the new graph if the endpoints of that arc are not already connected by a path.

Inheretence
`class` *subclass* `: public` *superclass* `{};`
Abstract Classes: *based on behavior rather than representation.* Never created on its own but serves as a superclass for *contrete classes*. Can't declare an object of abstract classes because some of the methods are not implemented. But can declare a pointer of an abstract, which can be pointed to a concrete subclass.
`Employee *a`; **a** can assign to subclass
Virtual Methods: methods implemented by a concrete subclass is indicated by including **= 0** before the semicolon on the prototype line, to mark the definition of a ***pure virtual method***. virtual in every subclass, no need to use the keyword virtual in the subclasses.
no virtual->compiler call method based on object declaration but not construct
Calling Superclass Constructors
default: call constructor with no argument
Sclicing: throw away any fields in the assigned object that don't fit into the super class.
allow to assign an object to a variable of its superclass type : A a = B();
BUT subclass object is assigned more space than objects of its superclass.
solution: use opinter A *a = B();
Multiple Inheritance
"deadly diamond of death"
If there is a method in A that B and C have overridden, and D does not override it, then which version of the method does D inherit?

Iterators
Index-based loop

```
for (int i = 0; i < str.length(); i++)
{... Body of loop that manipulates str[i] ...}
```

Range-based `for` loop

```
for (string word : lexicon)
{... Body of loop involving word ...}
```

C++ compiler translates it using ***iterators***

```
for (ctype::iterator it =
c.begin();it != c.end(); it++)
{... Body of loop involving *it ...}
```

.begin() returns an iterator at begin of collection
.end() just past the final element.
Iterator Orders: *arrays*-index order; *grids*-row-major; *map/set*-standard comparison(key); *lexicon*-alphabetical; *hashmap/hashset*-random
Function pointer
double *g(); Declare **g** as a function returning a pointer to a double
double (*fn)(double); Declares **fn** as pointer to function take/return a double
Callback Functions

`double x;`	*Declares* `x` *as a* `double`.
`double list[n];`	*Declares* `list` *as an array of* `n` `doubles`.
`double *px;`	*Declares* `px` *as a pointer to a* `double`.
`double **ppx;`	*Declares* `ppx` *as a pointer to a pointer to a* `double`.
`double f(double);`	*Declares* `f` *as a function taking and returning a* `double`.
`double *g(double);`	*Declares* `g` *as a function taking a* `double` *and returning a pointer to a* `double`.
`double (*fn)(double);`	*Declares* `fn` *as a pointer to a function taking and returning a* `double`.

回调函数就是一个通过函数指针调用的函数
pass functions as parameters to other functions

```
void plot( double (*fn)(double) );
```

Mapping Functions-is callback function

```
template <typename ValueType>
void mapAll(void (*fn) (ValueType) );
```

Passing Data to Mapping Functions
-Passing an additional argument, then included in the set of arguments to the callback function.
-Passing a function object. A ***function object*** is any object that overloads the function-call operator, which is designated as **`operator()`**.
iterators as parameters: `v.begin(), v.end()`

```
class ListKLetterWords {
public:
   ListKLetterWords(int k) {
      this->k = k;}
   int operator()(string word) {
      if (word.length() == k) {
         cout << word << endl;}}
private:
   int k; \\Length of desired words };
void listWordsOfLengthK(const Lexicon
& lex, int k) {
   lex.mapAll(ListKLetterWords(k));}
void Lexicon::mapAll(void
(*fn)(std::string)) const {
    for (std::string word :
m_allWords) {fn(word);}}
```

Multi-paradigm programming
C++ supports *procedural* and *object-oriented* paradigms naturally. Supports the *functional* paradigm through the <functional> interface. Supports many other paradigms through various external libraries.