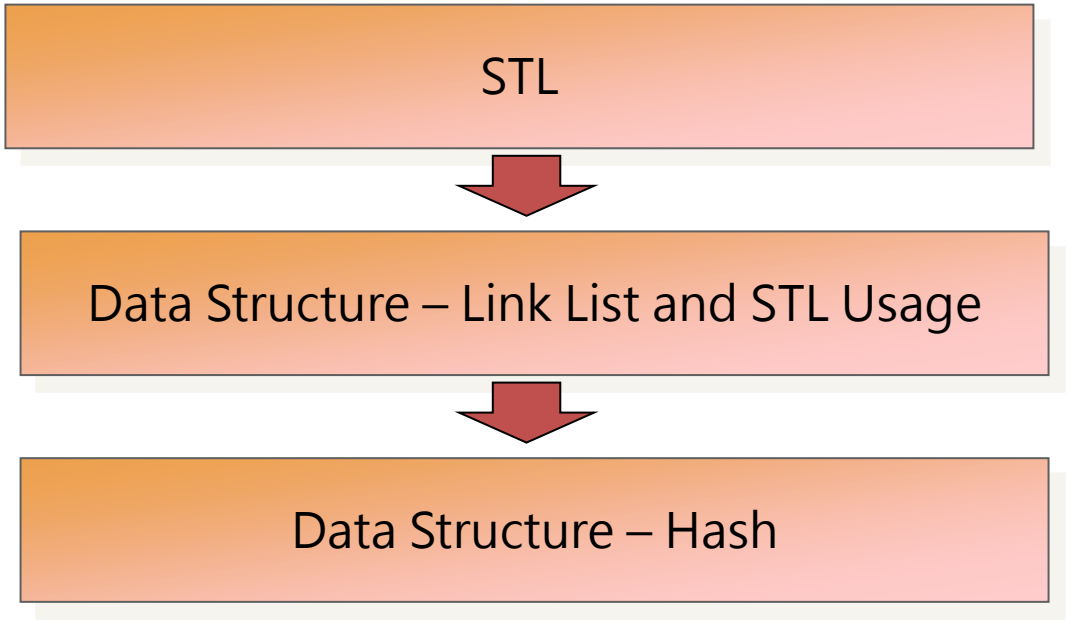
NCKU Programming Contest Training Course
Course 3
2013/01/02

Pin-Chieh Huang (free999)

Pinchieh.huang@gmail.com

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Outline


STL
Data Structure – Link List and STL Usage
Data Structure – Hash

Stack

- Stack
  - A stack is an ordered list in which insertions and deletions are made at one end called the top.
  - If we add the elements *A*, *B*, *C*, *D*, *E* to the stack, in that order, then *E* is the first element we delete from the stack
  - A stack is also known as a ***Last-In-First-Out*** (***LIFO***) list.

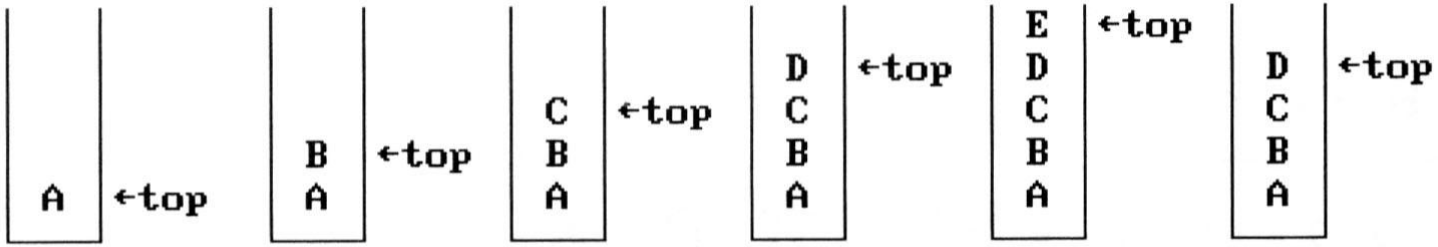
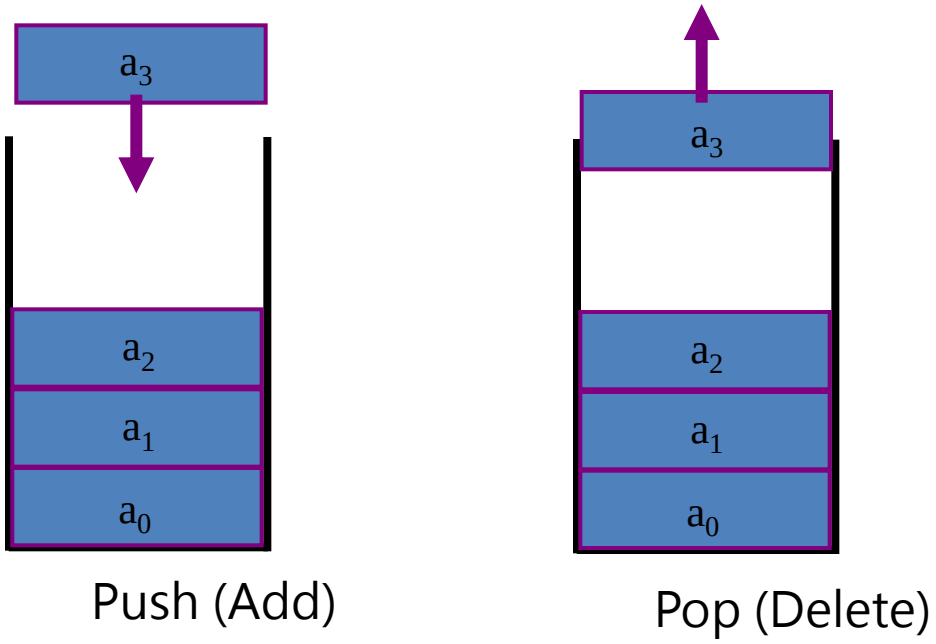


Figure 3.1: Inserting and deleting elements in a stack

made by electron & free999

Stack

- Main Subroutine
  - Push
  - Pop
  - Top
  - Empty



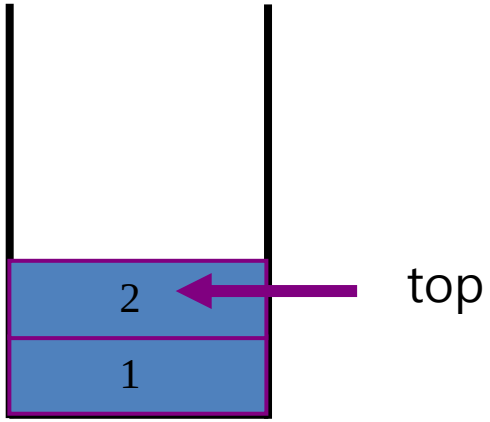
Push (Add)

Pop (Delete)

Stack

- Stack Usage in STL

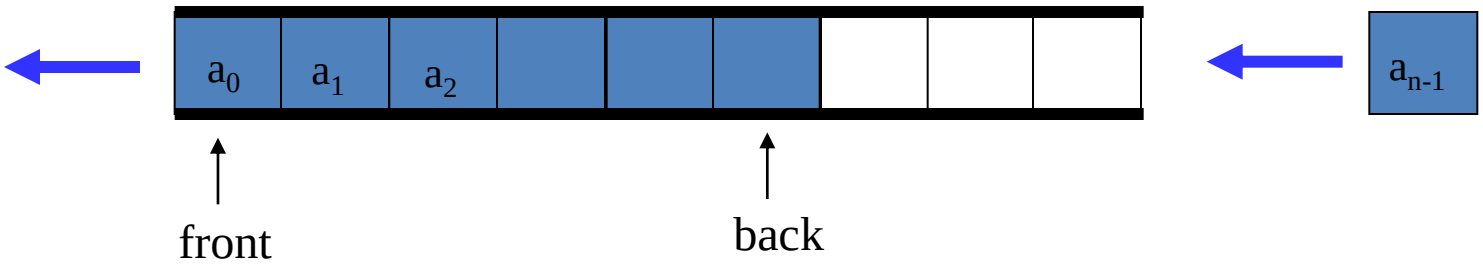
```
/* stack example */
#include <iostream>
#include <stack>
using namespace std;

int main()
{
        stack<int> stk;
        stk.push(1);
        stk.push(2);
        cout<<stk.top();   //2

        /* clear the stack */
        while(!stk.empty()) stk.pop();
}
```



made by electron & free999

Queue

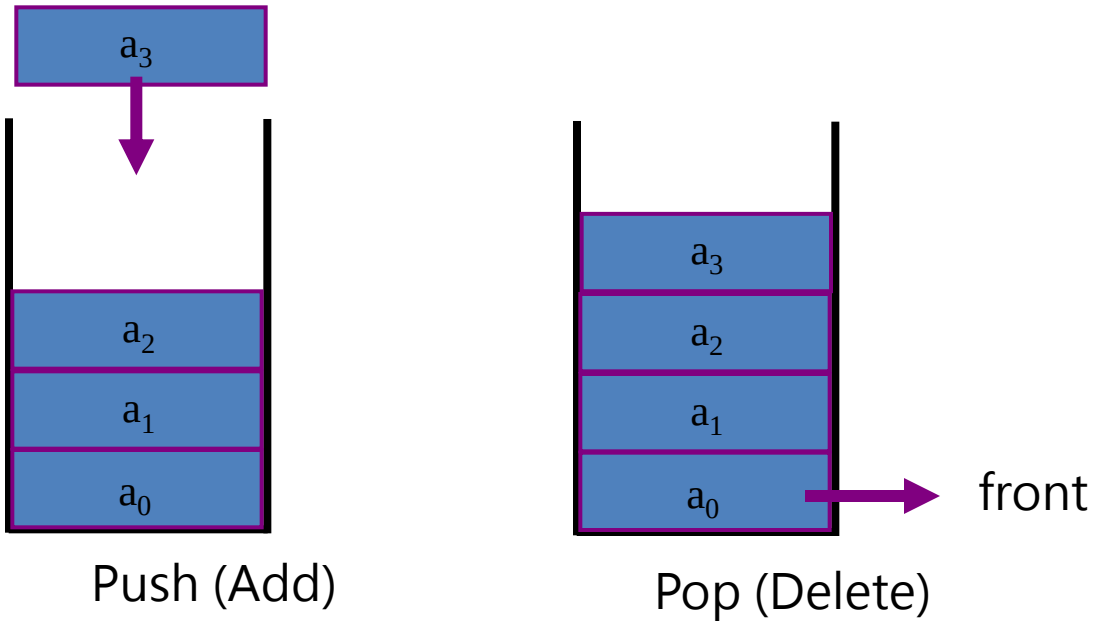
- Queue
  - A queue is an ordered list in which insertions and deletions are made at one end called the front
  - If we add the elements *A*, *B*, *C*, *D*, *E* to the stack, in that order, then *A* is the first element we delete from the queue
  - A stack is also known as a ***First-In-First-Out*** (***LIFO***) list.



made by electron & free999

Queue

- Main Subroutine
  - Push
  - Pop
  - Front
  - Empty

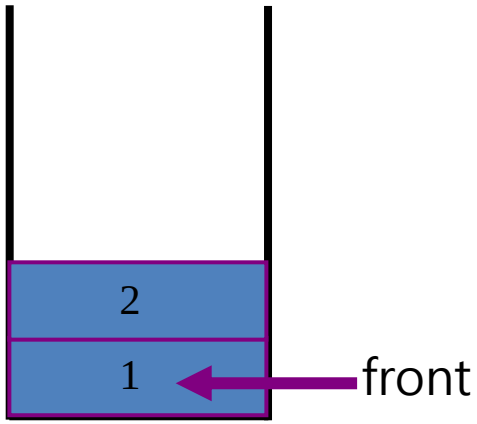


made by electron & free999

Queue

- Queue Usage in STL

```
/* stack example */
#include <iostream>
#include <queue>
using namespace std;

int main()
{
        queue<int> que;
        que.push(1);
        que.push(2);
        cout<<que.front();

        /* clear the stack */
        while(!que.empty()) que.pop();
}
```



made by electron & free999

Example 1

UVA-673 Parantheses

You are given a string consisting of parentheses () and []. A string of this type is said to be *correct*:

(a) if it is the empty string

(b) if A and B are correct, AB is correct,

(c) if A is correct, (A) and [A] is correct. Write a program that takes a sequence of strings of this type and check their correctness. Your program can assume that the maximum string length is 128.

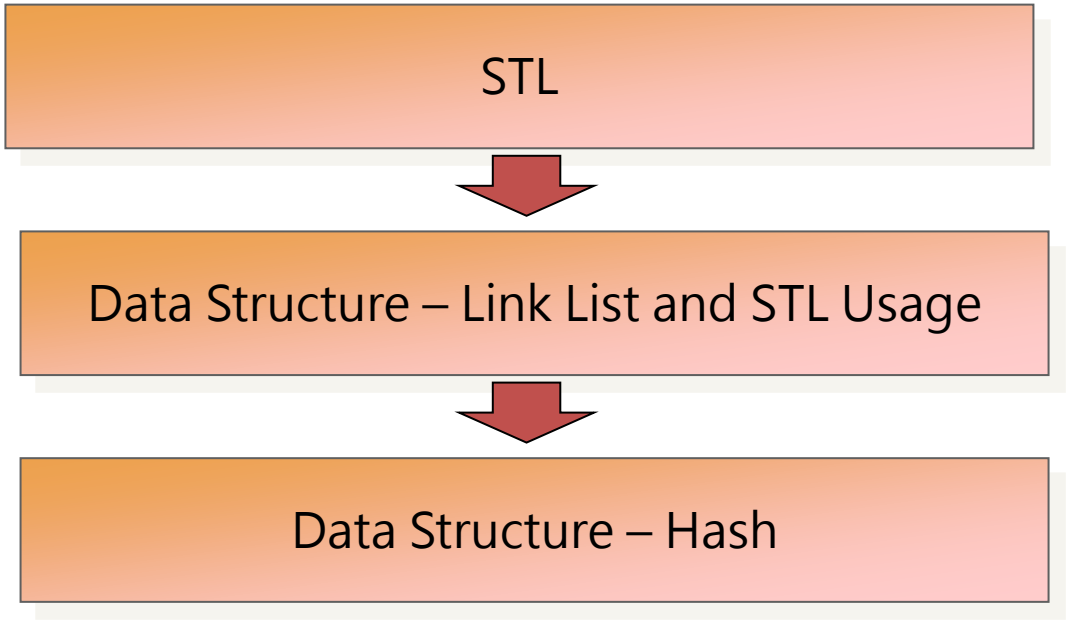
Example 1

- **Input**
  - The file contains a positive integer *n* and a sequence of *n* strings of parentheses () and [], one string a line.
- **Output**
  - A sequence of Yes or No on the output file.
- **Sample Input**

  3
  ([])
  (([()]))
  ([()[]()])()
- **Sample Output**

  Yes
  No
  Yes

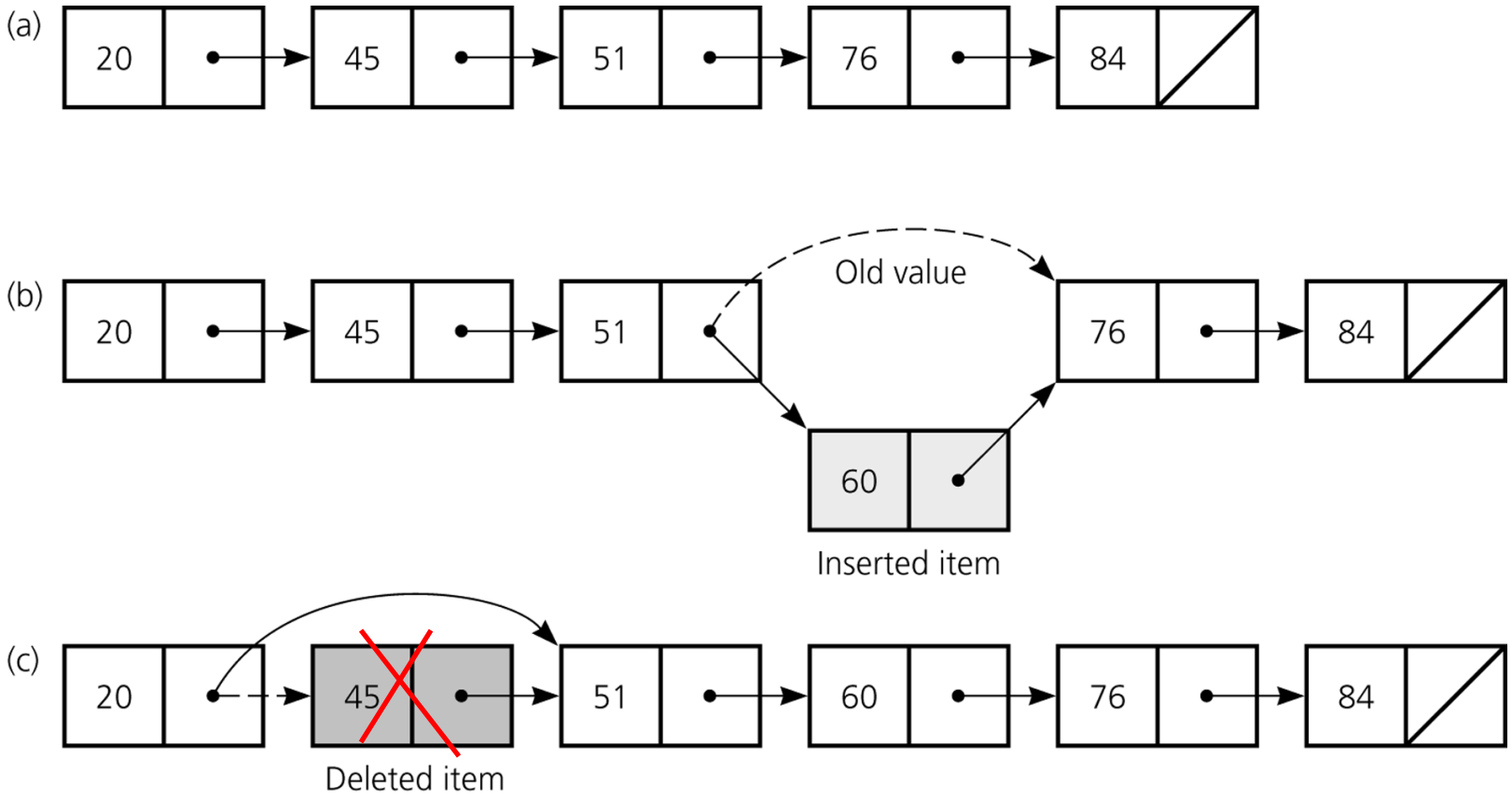
Outline


STL
Data Structure – Link List and STL Usage
Data Structure – Hash

Link List

- Array has the fixed size
  - Uneasy to dynamically update the data in the array
- Link List
  - Is able to dynamically update the data
  - Flexible size
  - Declaration mode
    - Use STL
    - Malloc, calloc, new [], …
  - Basic operation
    - Insertion
    - Deletion
    - Clear the link list

Link List



made by electron & free999

Link List

- Non-STL
  - C - malloc, calloc, free
  - C++ - new, delete
- Advantages
  - Fast Execution Time
  - More User Defined Operations
- Disadvantages
  - Code length is always long

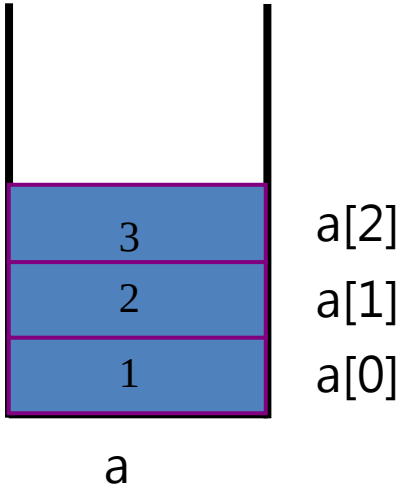
Link List

- STL-Based
  - C++ - vector, list
- Advantages
  - Code length is short
  - Easy to code
- Disadvantages
  - Long Execution time

vector

- Basic vector operation (#include <vector>)

```
vector<int>a;                       //declaration

a.push_back(1);                     //insertion
a.push_back(2);
a.push_back(3);

                                    //IO
for(int i=0; i<a.size(); i++)
        cout<<a[i]<<" ";            //1, 2, 3

vector<int>::iterator itr;
for(itr=in.begin(); itr!=in.end(); itr++)
        cout<<*itr<<" ";

a.clear();                          //clear the list
```



made by electron & free999

vector

- Sort a vector (#include <algorithm>)

```
vector<int>a;

a.push_back(3);
a.push_back(1);
a.push_back(5);                    //3, 1, 5

sort(a.begin(), a.end());          //1, 3, 5

Sort by decreasing order…
bool cmp (int a, int b)
{
        return a > b;
}
sort(a.begin(), a.end(), cmp);     //5, 3, 1
```

vector

- Multi-Dimension

```
vector<int>a[MAXN];
a[0].push_back(1), a[0].push_back(2);
a[1].push_back(5), a[1].push_back(4);
...
```

Ex: sort

```
sort(a[1].begin(), a[1].end());
```

Ex: clear

```
for(i=0; i<MAXN; i++)
    a[i].clear();
```

Ex: IO

```
for(i=0; i<a[1].size(); i++) cout<<a[1][i]<<" ";
```

vector

- Structure Based

```
struct STUDENT
{
    char name[10];
    int score;
};
vector<STUDENT>st;

STUDENT tmp;

//assign the data to tmp
Strcpy(tmp.name , "A");
tmp.score = 7;
...
st.push_back(tmp);
```

name	score
A st[0].name	7 st[0].score
G	3
D	2
C	1
B	5
E	4
F	6

~~tmp.name = "A"~~

made by electron & free999

vector

- Sort

```
//by lexical order of name
bool cmp(STUDENT a, STUDENT b)
{
    return strcmp(a.name, b.name)<0;
}
sort(st.begin(), st.end(), cmp);

struct STUDENT
{
    char name[10];
    int score;
    bool operator<(const STUDENT& t)const
    {  return strcmp(name, t.name)<0;    }
};
sort(st.begin(), st.end());
```

name	score
A	7
B	5
C	1
D	2
E	4
F	6
G	3

vector

- Sort

```
//by increasing order of score
bool cmp(STUDENT a, STUDENT b)
{
    return a.score < b.score;
}
sort(st.begin(), st.end(), cmp);
```

name	score
C	1
D	2
G	3
E	4
B	5
F	6
A	7

```
struct STUDENT
{
    char name[10];
    int score;
    bool operator<(const STUDENT& t)const
    {  return score < t.score;    }
};
sort(st.begin(), st.end());
```

vector

- 常用

#include <vector>

vector<int>v;

v.push_back()

v.size()

v.begin()

v.end()

v.clear()

v.erase()　v.erase(v.begin()+i)

v　　i = 2

1	2	6	3	5	3	1	4	7	8

vector

- 常用

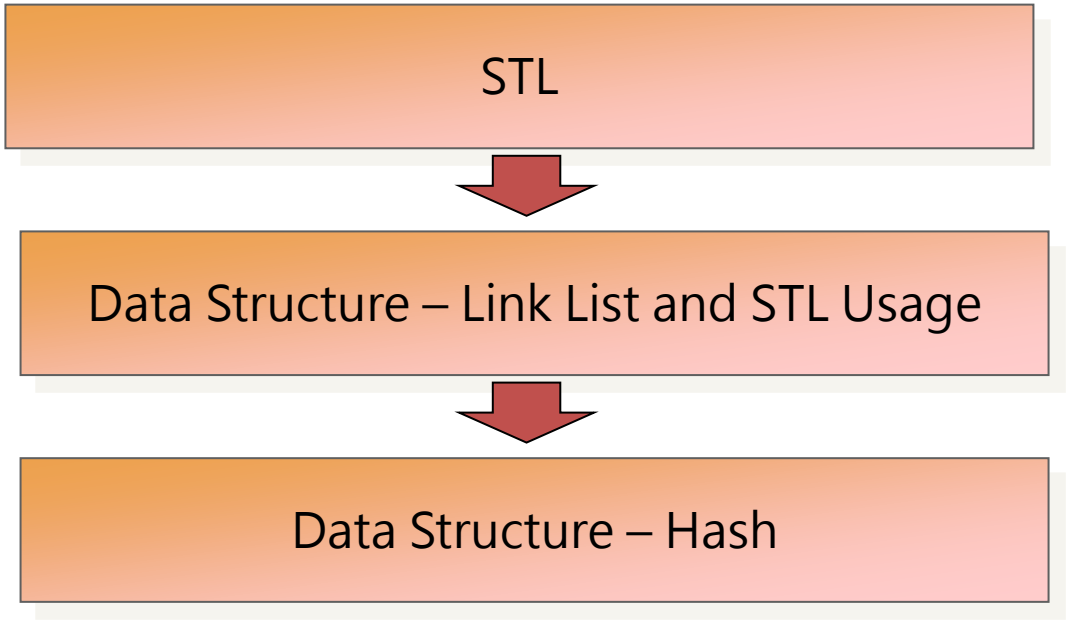
```
#include <vector>
vector<int>v;
v.push_back()
v.size()
v.begin()
v.end()
v.clear()
```

v.erase() : v.erase(v.begin()+i)

v　　i = 2

1	2	3	5	3	1	4	7	8

Outline


STL
Data Structure – Link List and STL Usage
Data Structure – Hash

Hash

- Why?
  - N number with numbered by 1~N
    - easy to declare an array
  - N number with range 1~2147483647
    - uneasy to declare an array

By STL map

- map (allows direct address)
  - red-black tree based hash map

```
#include <map>
//declaration
map<int, int>QQ;
QQ[1234567] = 123456;
QQ[8] = -1;
QQ[-3] = 1;
printf("%d\n", QQ[8]);              //-1
printf("%d\n", QQ[123456]);         //0
printf("%d\n", QQ[-3]);     //1

//clear
QQ.clear();
```

Example - 2

POJ 2503 Babel-fish

Description

You have just moved from Waterloo to a big city. The people here speak an incomprehensible dialect of a foreign language. Fortunately, you have a dictionary to help you understand them.

Input

Input consists of up to 100,000 dictionary entries, followed by a blank line, followed by a message of up to 100,000 words. Each dictionary entry is a line containing an English word, followed by a space and a foreign language word. No foreign word appears more than once in the dictionary. The message is a sequence of words in the foreign language, one word on each line. Each word in the input is a sequence of at most 10 lowercase letters.

Output

Output is the message translated to English, one word per line. Foreign words not in the dictionary should be translated as "eh".

Example - 2

POJ 2503 Babel-fish

Sample Input

dog ogday
cat atcay
pig igpay
froot ootfray
loops oopslay

atcay
ittenkay
oopslay

Sample Output

cat
eh
loops

Take a Break

made by electron & free999

By STL set

- set (not allows direct address)
  - red-black tree based hash set

```
#include <set>
//declaration
set<int>my;
my.insert(1); my.insert(5); my.insert(3);

//IO
set<info>::iterator itr;
for(itr=my.begin(); itr!=my.end(); itr++)
    cout<<*itr<<" "<<endl; //1, 3, 5

//clear
my.clear();
```

Example - 3

UVA - 10815 Andy’ s Dictionary

Andy, 8, has a dream - he wants to produce his very own dictionary. This is not an easy task for him, as the number of words that he knows is, well, not quite enough. Instead of thinking up all the words himself, he has a briliant idea. From his bookshelf he would pick one of his favourite story books, from which he would copy out all the distinct words. By arranging the words in alphabetical order, he is done! Of course, it is a really time-consuming job, and this is where a computer program is helpful.

You are asked to write a program that lists all the different words in the input text. In this problem, a word is defined as a consecutive sequence of alphabets, in upper and/or lower case. Words with only one letter are also to be considered. Furthermore, your program must be CaSe InSeNsItIvE. For example, words like "Apple", "apple" or "APPLE" must be considered the same.

- **Input**
  - The input file is a text with no more than 5000 lines. An input line has at most 200 characters. Input is terminated by EOF.
- **Output**
  - Your output should give a list of different words that appears in the input text, one in a line. The words should all be in lower case, sorted in alphabetical order. You can be sure that he number of distinct words in the text does not exceed 5000.

Example - 3

- Sample Input
  - Adventures in Disneyland Two blondes were going to Disneyland when they came to a fork in the road. The sign read: "Disneyland Left." So they went home.
- Sample Output

```
a
adventures
blondes
came
disneyland
fork
going
home
in
left
read
road
sign
so
the
...
```

Hash

- Hash
  - A Static Table with Fixed Size
  - hash represent the procedure that using a function f(x) to transform a key word x into a new address

made by electron & free999

Hash

- Bucket (桶):The Size of f(x) for the Hash Table
- Slot (槽): The Size of each Bucket
- Collision (碰撞): x_1 != x_2, while $f(x_1)=f(x_2)$
- Overflow (溢位): The Size of Bucket is **Full**

Hash

- Execution Time of Hash Technique
  - Depend on the Defined Hash Function
  - A **hash function** is any **well-defined procedure** or **mathematical function** which converts a large, possibly variable-sized amount of data into a small datum, usually a single integer that may serve as an index to an array. The values returned by a hash function are called **hash values**, **hash codes**, **hash sums**, or simply **hashes**.

Hash Function 1

- Example of Hash Function (Mid-Square)
  - 將識別字轉成一個數值,再求它的平方值,然後再取其中間的幾個位數作為桶的位址
  - 由於平方值的中間幾個位數通常和識別字的所有字元有關,所以會有較高的機率產生不同的位址
    - EX.
    - CFGA→3671(轉成數值)→13476241(取平方值)→762(取中間三位數)

Hash Function 2

- Example of Hash Function (Folding)

- Shift folding
- Ex.

  key word X

  X=167329421598 12

  pick up for each three characters

P1	167
P2	329
P3	421
P4	598
P5	12

summation 1527

Hash Function 3

- Example of Hash Function (Boundary at the Folding)

P1	167
P2	~~329~~ ←reverse 923
P3	421
P4	~~598~~ ←reverse 895
P5	12

summation 2418

Hash Function 4

- Example of Hash Function (Division)
  - Divide the key word by a value M and use the modulo value as the address

$$f(x) = x\%M$$

M是除數,求出的餘數介於0至M-1之間.

Hash Function 4

- push **345, 728, 251, 490, 15** into 12 buckets, M is 12, **f(x)=x%12**

  f(345)=9

  f(728)=8

  f(251)=11

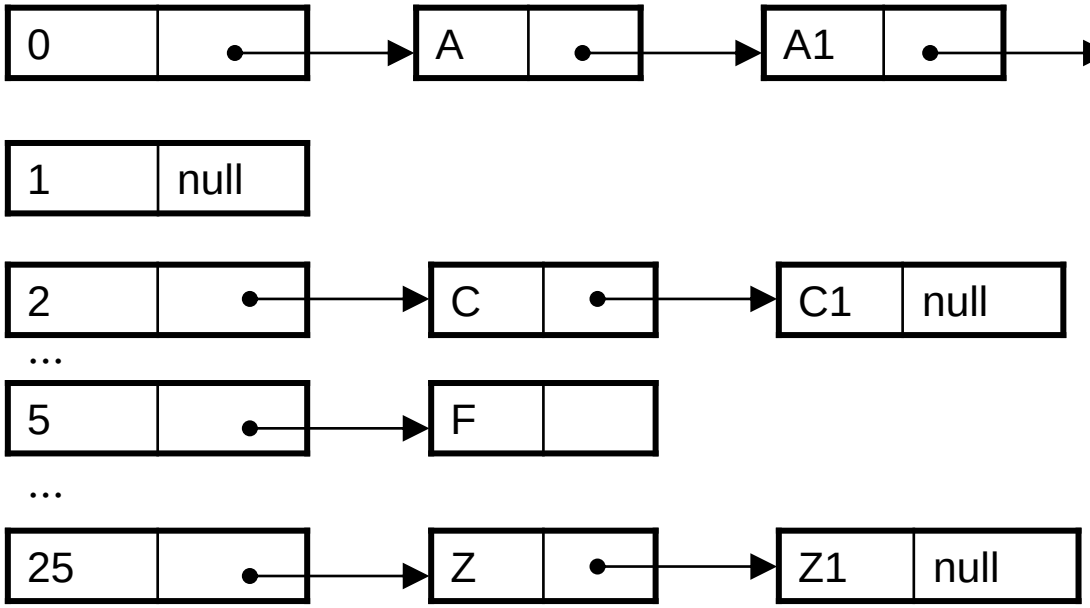
  f(490)=10

  f(15)=3

	X		X
0		6	
1		7	
2		8	728
3	15	9	345
4		10	490
5		11	251

Overflow Handling

- Overflow Handling
  - the size of bucket is overflow due to multiple key words that access the same bucket
  - use the link list to store multiple key words (vector is powerful)



key words:
A, A1, A2, C, C1, F, Z, Z1

made by electron & free999

Example - 4

Snow Snowflakes (POJ 3349)

Problem Description

You may have heard that no two snowflakes are alike. Your task is to write a program to determine whether this is really true. Your program will read information about a collection of snowflakes, and search for a pair that may be identical. Each snowflake has six arms. For each snowflake, your program will be provided with a measurement of the length of each of the six arms. Any pair of snowflakes which have the same lengths of corresponding arms should be flagged by your program as possibly identical.

Example - 4

Snow Snowflakes (POJ 3349)

I/O Description

Input

The first line of input will contain a single integer n, $0 < n \leq 100000$, the number of snowflakes to follow. This will be followed by n lines, each describing a snowflake. Each snowflake will be described by a line containing six integers (each integer is at least 0 and less than 10000000), the lengths of the arms of the snow ake. The lengths of the arms will be given in order around the snowflake (either clockwise or counterclockwise), but they may begin with any of the six arms. For example, the same snowflake could be described as 1 2 3 4 5 6 or 4 3 2 1 6 5.

Output

If all of the snowflakes are distinct, your program should print the message:
No two snowflakes are alike.
If there is a pair of possibly identical snow akes, your program should print the message:
Twin snowflakes found.

Example - 4

Snow Snowflakes (POJ 3349)

Sample I/O

Input

2
1 2 3 4 5 6
4 3 2 1 6 5

Output

Twin snowflakes found.

made by electron & free999

Example - 5

Gold Balance Lineup (POJ 3274)

Problem Description

Farmer John's *N* cows (1 ≤ *N* ≤ 100,000) share many similarities. In fact, FJ has been able to narrow down the list of features shared by his cows to a list of only *K* different features (1 ≤ *K* ≤ 30). For example, cows exhibiting feature #1 might have spots, cows exhibiting feature #2 might prefer C to Pascal, and so on.

FJ has even devised a concise way to describe each cow in terms of its "feature ID", a single K-bit integer whose binary representation tells us the set of features exhibited by the cow. As an example, suppose a cow has feature ID = 13. Since 13 written in binary is 1101, this means our cow exhibits features 1, 3, and 4 (reading right to left), but not feature 2. More generally, we find a 1 in the 2^(*i*-1) place if a cow exhibits feature *i*.

Always the sensitive fellow, FJ lined up cows 1..*N* in a long row and noticed that certain ranges of cows are somewhat "balanced" in terms of the features the exhibit. A contiguous range of cows *i*..*j* is balanced if each of the *K* possible features is exhibited by the same number of cows in the range. FJ is curious as to the size of the largest balanced range of cows. See if you can determine it.

Example - 5

Gold Balance Lineup (POJ 3274)

I/O Description

Input

Line 1: Two space-separated integers, *N* and *K*.
Lines 2..*N*+1: Line *i*+1 contains a single *K*-bit integer specifying the features present in cow *i*. The least-significant bit of this integer is 1 if the cow exhibits feature #1, and the most-significant bit is 1 if the cow exhibits feature #*K*.

Output

Line 1: A single integer giving the size of the largest contiguous balanced group of cows.

Example - 5

Gold Balance Lineup (POJ 3274)

Sample I/O

Input

```
7 3
7
6
7
2
1
4
2
```

Output

```
4
```

Homework 3

Total 25 Problems

- uva (10)
  - 673, 311, 834, 846, 10020, 10050, 10249, 10678, 11417, 11489
- zoj2 (7)
  - d003, d004, d008, d009, d011, d017, d077
- pku (8)
  - 1936, 2109, 2388, 2503, 2586, 3080, 3349, 3274

Thank for Your Attention

made by electron & free999