소프트웨어 공학

문자열(Strings)

- 문자들의 연속
- 각 프로그래밍 언어마다 다르게 표현
- C/C++:
  - NULL로 끝나는 배열: 문자열은 NULL 문자인 '\0' 로 끝남
  - 배열은 가능한 최대 문자열의 크기를 충분히 할당해야 함 (NULL 을 포함해서)
- Java:
  - 배열과 배열의 길이로 표현(Array plus length)

문자열 입력 : scanf(), gets()함수

- 함수 scanf()를 이용한 문자열 입력
  - %s로 입력 받기

```
char str[30];
scanf("%s", str);
```

- 함수 gets()를 이용한 문자열 입력
  - 문자열을 입력 받을 적절한 문자 배열 이용
  - 반환 주소 값을 이용하면 문자별로 참조가 가능

```
char line[81];
char *ptr;
//gets(line);
*ptr = gets(line);
```

문자열에 대한 연산

- 두 문자열을 연결 – strcat(), strncat()
- 문자열 비교하기 – strcmp(), strncmp()
- 문자열 복사 – strcpy()
- 문자열의 길이 확인 – strlen()
- 문자열 뒤집기
- 문자열에서 특정 문자 찾기 – strchr()
- 문자열에서 특정 문자열 찾기 – strstr()
  - 문자열 일치 문제(또는 문자열 찾기 문제)

C String Library

- ctype.h

함수	설명
int isalnum (int c);	c가 알파벳 또는 숫자이면 0이 아닌 값을 반환한다.
int isalpha (int c);	c가 알파벳이면 0이 아닌 값을 반환한다.
int isdigit (int c);	c가 숫자이면 0이 아닌 값을 반환한다.
int isgraph (int c);	c가 그래픽 문자이면 0이 아닌 값을 반환한다.
int islower (int c);	c가 소문자이면 0이 아닌 값을 반환한다.
int isprint (int c);	c가 출력할 수 있는 문자이면 0이 아닌 값을 반환한다.
int ispunct (int c);	c가 구두점 문자이면 0이 아닌 값을 반환한다.
int isspace (int c);	c가 공백 문자이면 0이 아닌 값을 반환한다.
int isupper (int c);	c가 대문자이면 0이 아닌 값을 반환한다.
int tolower (int c);	c를 소문자로 변환한다.
int toupper (int c);	c를 대문자로 변환한다.

C String Library

- string.h

함수	설명
char * **strcat** (char * dst, const char * src);	src를 dst뒤에 붙인다.
char * **strncat** (char * dst, char * src, size_t n);	src에서 dst뒤에 처음 n개의 문자들을 붙인다.
int **strcmp**(const char* s1, const char* s2)	s1과 s2를 비교한다. 같으면 0 반환
int **strncmp** (const char * s1, const char * s2, size_t n);	s1의 처음 n개의 문자를 s2의 처음 n개의 문자와 비교한다.
char * **strcpy(char** * dst,const char * src);	src를 dst에 복사한다.
char * **strncpy** (char * dst, const char * src, size_t n);	src에서 dst으로 처음 n개의 문자들을 복사한다.
size_t **strlen** (const char * str);	str의 길이를 반환한다.
char * **strstr** (const char * s1, const char * s2);	s1에서 s2를 검색하여 가장 먼저 나타나는 곳의 위치를 반환한다.
char * **strtok** (char * s1,const char * s2);	s1을 s2의 문자들로 분리한다.

함수 strcmp()

- 함수원형
  - 두 문자열을 비교하는 함수

```
int    strcmp(const  char *, const char *);
```

- 비교방법
  - 전달인자인 두 문자열을 사전(lexicographically) 상의 순서로 비교
  - 앞과 뒤의 문자열을 사전 순서로 비교하여 앞 문자열이 먼저 나오면 음수, 뒤 문자열이 먼저 나오면 양수, 같으면 0을 반환

```
printf("%d", strcmp("a", "ab")); //음수 출력
printf("%d", strcmp("ab", "a")); //양수 출력
printf("%d", strcmp("ab", "ab")); //0 출력
```

  - 비교 기준은 아스키 코드 값
  - 두 문자가 같다면 계속 다음 문자를 비교하여, 다른 문자에서 앞 문자가 작으면 음수, 뒤 문자가 작으면 양수, 같으면 0을 반환
  - 대문자가 소문자보다 아스키 코드 값이 작으므로
    - strcmp("java", "javA")는 양수를 반환

함수 strcat()

- 함수 원형

```
char *  strcat(char *, const char *);
```

  - 앞 문자열의 마지막 NULL 문자에서부터 뒤 문자열의 NULL 문자까지 연결하여, 앞의 문자열 주소를 반환

```
char stmt[25] = "Java is o-o ";
char lang[] = "language.";
printf("%s\n", strcat(stmt, lang));
```

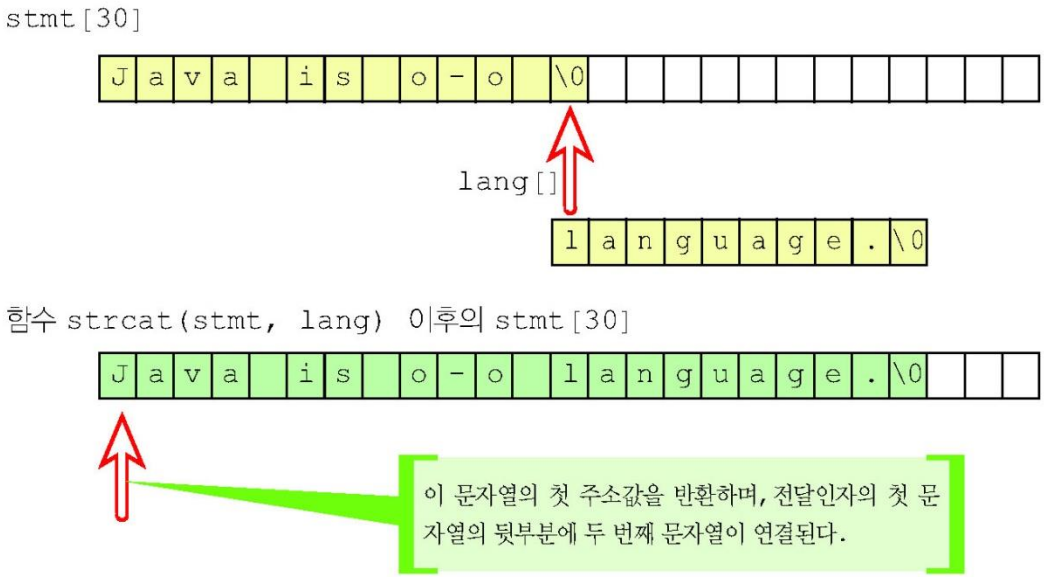


그림 15.19 함수 strcat( )의 문자열 연결방법

함수 strcpy()

- 함수원형

```
char *  strcpy(char *, const char *);
char *  strncpy(char *, const char *, size_t n);
```

  - 앞 문자열에 처음에 뒤 문자열을 복사하여 그 복사된 문자열을 반환
  - 앞 문자열에 처음에 뒤 문자열을 NULL 문자까지, 최대 n개를 복사하여 그 복사된 문자열을 반환

```
char stmt[30] = "Java is object oriented ";
printf("%s\n", strcpy(stmt, "C"));
```

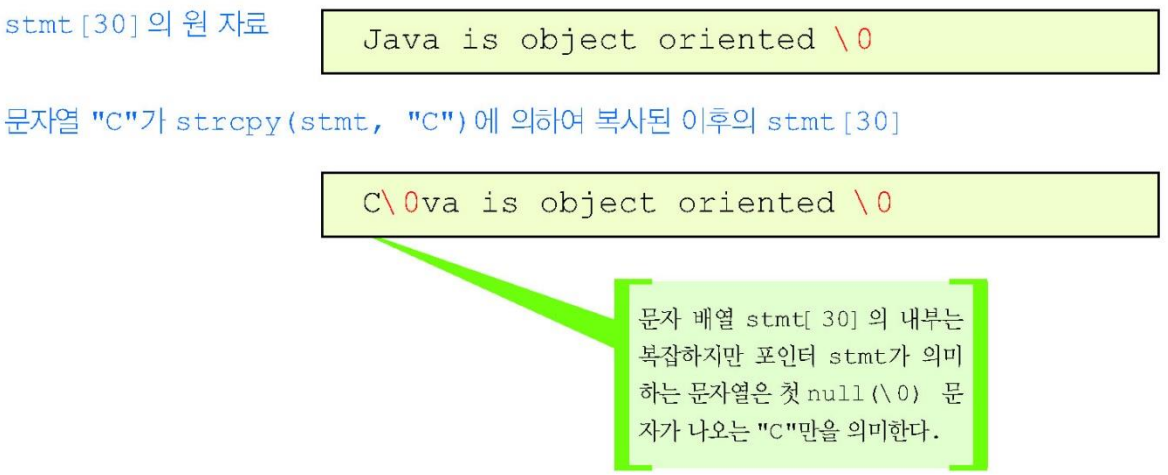


그림 15.20 함수 strcpy( )에 위한 문자열 배열의 변화

함수 strtok()

- 함수원형

```
char *  strtok(char *, const char *);
```

  - 앞 문자열에서 뒤 문자열에 포함되어 있는 구분자를 기준으로 토큰을 추출

- 사용방법
  - 다음 문자열에서 구분자를 공백 문자, 쉼표(,)로 토큰을 분리
    - "C, C++ language are best!"

```
char *ptoken, *delimiter = " ,";
char str1[] = "C, C++ language are best!";
ptoken = strtok(str1, delimiter);
while ( ptoken != NULL )
{
        printf("%s\n", ptoken);
        ptoken = strtok(NULL, delimiter);
}
```

    - 분리되는 토큰 단어는 “C”, “C++”, “language”, “are”, “best!” 5개

C++ Output : `cout` Object

- Use **<<** to send information to cout

```
#include <iostream>
 using namespace std;

 cout << "Hello, there!";
```

- Can use **<<** to send several items to cout

```
 cout << "Hello, " << "there!" << endl;
```

Or

```
 cout << "Hello, ";
 cout << "there!" << endl;
```

C++ Input : Character

```
char ch;

cin >> ch;      // Reads in any non-blank char

cin.get(ch);    // Reads in any char

cin.ignore()    // Skips over next char in
                // the input buffer
```

C++ Input : String

Reading in a string object

```
string str;

cin >> str;          // Reads in a string
                     // with no blanks

getline(cin, str); // Reads in a string
                     // that may contain
                     // blanks
```

C++ String Library

- C++ 은 c-style strings, string class를 제공.

```
string::size()
string::empty()
string::append(s)
string::erase(n, m)   // n번째부터 m개의 문자 삭제
string::insert(size_type n, const string&s) // n번째부터 문자열 s 삽입
string::find(s)
string::substr(s)
string::rfind(const string& s, size_type n) // n번째부터 앞쪽으로 s 문자열 검색
string::compare(s)
```

C++ String Operators

= Assigns a value to a string

```
string words;
words = "Tasty ";
```

+ Joins two strings together

```
string s1 = "hot", s2 = "dog";
string food = s1 + s2; // food = "hotdog"
```

+= Concatenates a string onto the end of another one

```
words += food; // words now = "Tasty hotdog"
```

Overloaded `string` Operators

```
string word1, phrase;
string word2 = " Dog";
cin >> word1; // user enters "Hot"
              // word1 has "Hot"
phrase = word1 + word2; // phrase has
                        // "Hot Dog"
phrase += " on a bun";
for (int i = 0; i < 16; i++)
   cout << phrase[i];  // displays
                       // "Hot Dog on a bun"
```

Conversion to C-strings

- `data()` and `c_str()` both return the C-string equivalent of a **`string`** object
- Useful in using a string object with a function that is expecting a C-string

```
char greeting[20] = "Have a ";
 string str("nice day");
 strcat(greeting, str.data());
```

Modification of `string` objects

- **`str.append(string s)`**

  appends contents of **`s`** to end of **`str`**

- Convert constructor for **`string`** allows a C-string to be passed in place of **`s`**

```
string str("Have a ");
str.append("nice day");
```

- **`append`** is overloaded for flexibility

Modification of `string` objects

- **`str.insert(int pos, string s)`**

  inserts **`s`** at position **`pos`** in **`str`**

- Convert constructor for **`string`** allows a C-string to be passed in place of **`s`**

```
string str("Have a day");
str.insert(7, "nice ");
```

- **`insert`** is overloaded for flexibility

예제 : Push Pop

- 문제
  - PUSH, POP 단어를 포함하는 임의의 문장에서 PUSH와 POP의 개수를 각각 구하는 프로그램을 작성하시오.
- 입력
  - PUSH 혹은 POP을 포함하는 임의의 문자열이 한 줄로 입력된다.
- 출력
  - PUSH와 POP의 개수를 출력한다.

- 입출력 예

<입력예>

```
PUSH POP POPOPOP PUSH PUSH
```

<출력예>

```
3 1
```

strtok() 함수 이용

```
void eg1()
{
    char line[81], *ptoken;
    int cpush = 0, cpop = 0;

    gets(line);
    ptoken = strtok(line, " ");
    while(ptoken != NULL) {
        if (strcmp(ptoken, "PUSH") == 0 ) cpush++;
        if (strcmp(ptoken, "POP") == 0 ) cpop++;
        ptoken = strtok(NULL, " ");
    }
    printf("%d %d₩n", cpush, cpop);
}
```

sscanf() 함수 이용

```
void eg2()
{
    char line[81], word[30], *ptr;
    int cpush = 0, cpop = 0;

    gets(line);
    ptr = line;
    while( sscanf(ptr, "%s", word) != EOF ) {
        if (strcmp(word, "PUSH") == 0 ) cpush++;
        if (strcmp(word, "POP") == 0 ) cpop++;
        ptr += strlen(word);
    }
    printf("%d %d₩n", cpush, cpop);
}
```

‘\n’ 검사하기

```
void eg3()
{
    char ch, word[30];
    int cpush = 0, cpop = 0;

    while( 1 ) {
        scanf("%s", word);
        if (strcmp(word, "PUSH") == 0 ) cpush++;
        if (strcmp(word, "POP") == 0 ) cpop++;
        ch = getchar();
        if ( ch == '₩n' )
            break;
    }
    printf("%d %d₩n", cpush, cpop);
}
```

실습1 : Max와 Min

- 문제
  - N (2 ≤ N ≤ 2,000)개의 정수로 이루어진 수열에서 가장 큰 수와 가장 작은 수를 구하는 프로그램을 작성하시오.
- 입력
  - 첫 줄에는 수열의 길이 N이 입력되고, 그 다음에 N개의 정수가 입력된다. 꼭 한 줄에 입력될 필요는 없다.
- 출력
  - 가장 큰 수와 가장 작은 수를 출력한다.

- 입출력 예

<입력예>

```
6
4 7 2 9
10 1
```

<출력예>

```
10 1
```

실습2 : Histogram

- 문제
  - N (2 ≤ N ≤ 2,000)개의 정수로 이루어진 수열에서 끝자리가 '0'인 수의 개수, 끝자리가 '1'인 수의 개수 등을 각각 구하는 프로그램을 작성하시오.
- 입력
  - 첫 줄에는 수열의 길이 N이 입력되고, 그 다음에 N개의 정수가 입력된다. 꼭 한 줄에 입력될 필요는 없다.
- 출력
  - 끝자리가 '0' ~ '9'에 해당하는 숫자의 개수를 출력한다.

- 입출력 예

<입력 예>

```
10
4 7 2 9 1009
10 1 1003 200499
```

<출력 예>

```
0 1
1 1
2 1
3 1
4 2
5 0
6 0
7 1
8 0
9 3
```

실습3 : 가장 깊은 괄호 깊이 구하기

- 문제
  - 괄호가 포함된 임의의 수식에서 가장 깊은 괄호쌍의 깊이를 구하는 프로그램을 작성하시오. 예를 들어, (a + (b*c)+(d -e)*(10*(1-x)))가 입력으로 들어오면 "1-x"를 포함하고 있는 괄호가 3개의 괄호쌍에 묶여 있으므로 가장 깊은 괄호 깊이는 3이 된다.
- 입력
  - 괄호를 포함하는 한 줄의 수식을 문자열로 입력한다.
- 출력
  - 가장 깊은 괄호 깊이를 출력한다.

- 입출력 예

<입력예>

```
(a + (b*c)+(d -e)*(10*(1-x)))
```

<출력예>

```
3
```

실습 4 : 영어 단어 수 구하기

- 문제
  - 영어 알파벳 대소문자와 스페이스로 구성된 영어 문장에서 단어의 개수를 구하는 프로그램을 작성하시오.
- 입력
  - 영어 알파벳과 스페이스로 이루어진 문자열을 한 줄에 입력한다.
- 출력
  - 단어 수를 출력한다.

- 입출력 예

<입력예>

```
I am a boy
```

<출력예>

```
4
```

실습 5 : 최대 연속 공통 부분문자열 구하기

- 문제
  - 두 개의 문자열 A, B가 주어졌을 때 A와 B의 연속되는 공통의 부분 문자열 중 가장 긴 것을 구하여 출력하시오.
- 입력
  - 첫째 줄에는 A의 문자열이 입력된다. 둘째 줄에는 B의 문자열이 입력된다. 각 문자열은 각각 20자를 넘지 않는다. 또한 입력문자열은 소문자로만 구성되어 있다.
- 출력
  - 최대의 길이를 갖는 연속 공통 문자열을 출력한다.
- 입출력 예

<입력예>

```
photography
autography
```

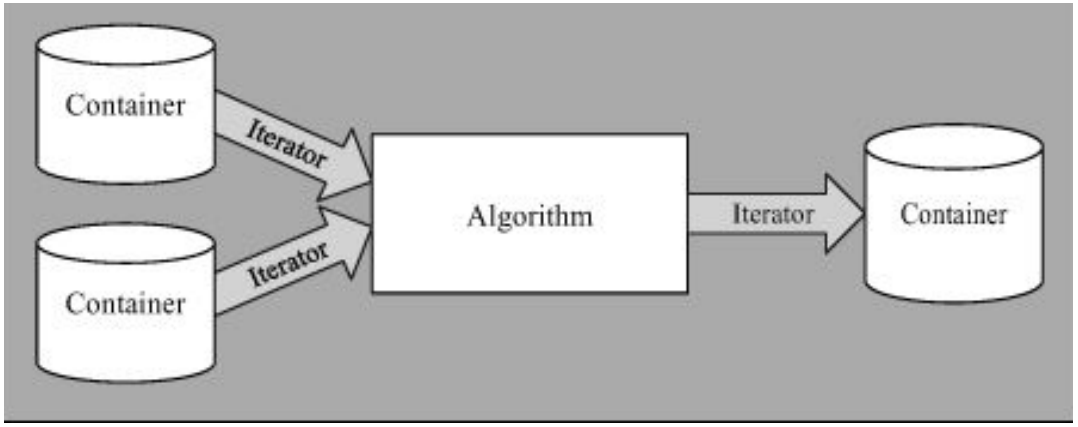
<출력예>

```
tography
```

STL

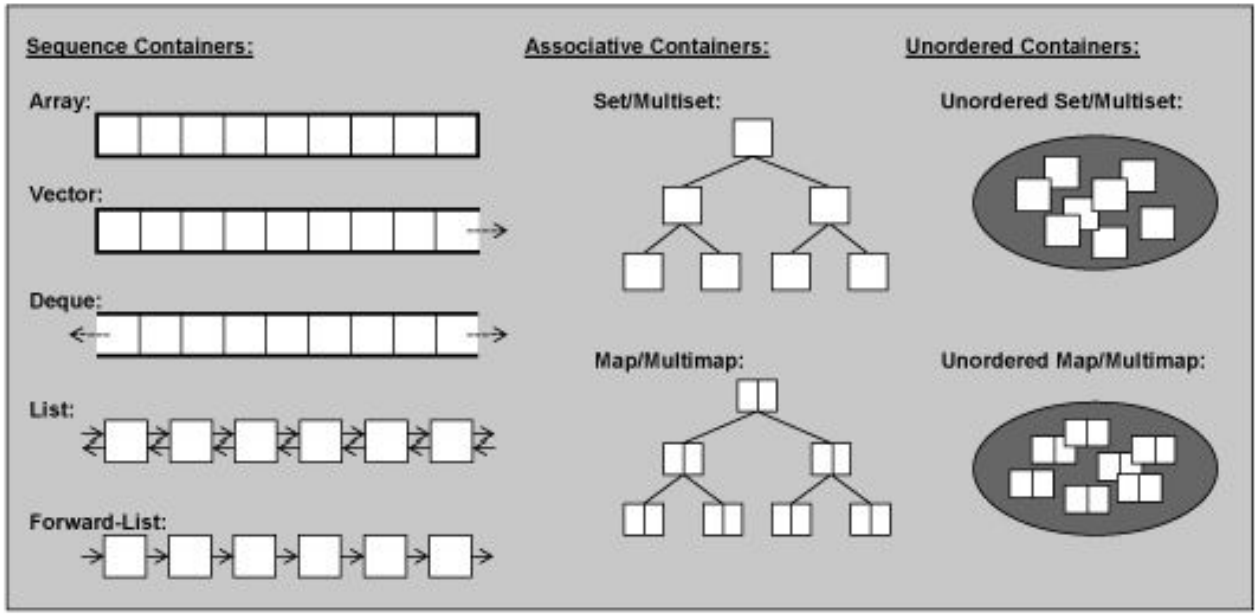
Standard Template Library

- Two important types of data structures in the STL:
  - containers: classes that store data and impose some organization on it
  - iterators: like pointers; mechanisms for accessing elements in a container



Containers

- Two types of container classes in STL:
  - sequence containers: organize and access data sequentially. **`vector`**, **`dequeue`**, and **`list`**
  - associative containers: use keys to allow data for quick access. **`set`**, **`multiset`**, **`map`**, and **`multimap`**



Container Objects 생성

- For example, to create a list of **`int`**, write

```
list<int> mylist;
```

- To create a vector of **`string`** objects, write

```
vector<string> myvector;
```

Iterators

- Generalization of pointers, used to access information in containers
- Four types:
  - forward (uses **++**)
  - bidirectional (uses **++** and **--**)
  - random-access
  - input (can be used with **cin** and **istream** objects)
  - output (can be used with **cout** and **ostream** objects)

Algorithms

- STL contains algorithms implemented as function templates to perform operations on containers.
- Requires **`algorithm`** header file
- Collection of algorithms includes

`binary_search`	**`count`**
`for_each`	**`find`**
`find_if`	**`max_element`**
`min_element`	**`random_shuffle`**
`sort`	and others

Containers and Iterators

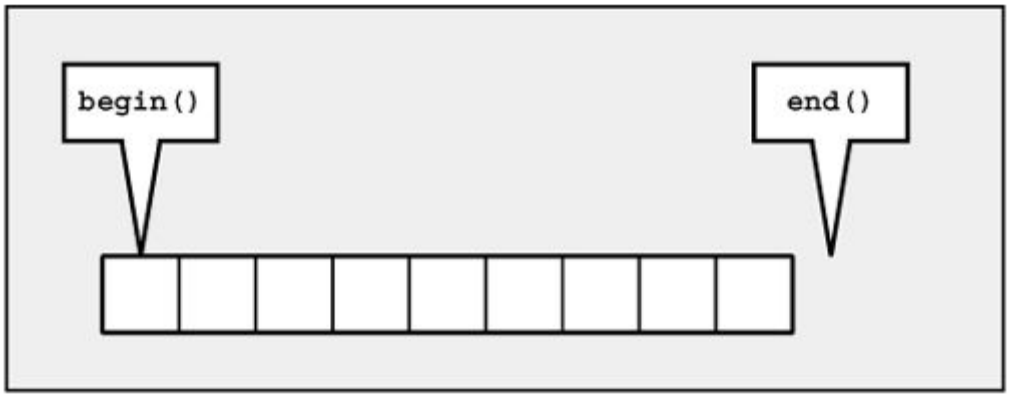
- Each container class defines an iterator type, used to access its contents
- The type of an iterator is determined by the type of the container:

```
list<int>::iterator x;
list<string>::iterator y;
```

`x` is an iterator for a container of type `list<int>`

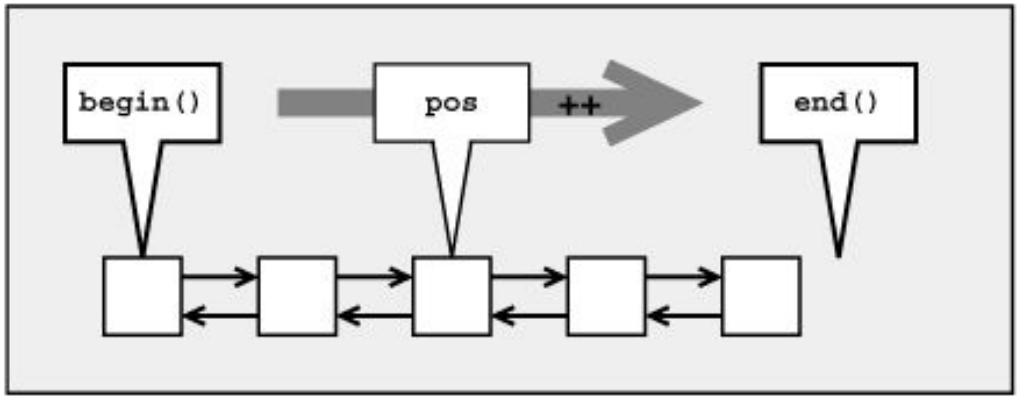
Containers and Iterators

- Each container class defines functions that return iterators:
  - **`begin()`**: returns iterator to item at start
  - **`end()`**: returns iterator denoting end of container



Containers and Iterators

- Iterators support pointer-like operations:
  - `*iter` is the item it points to: this dereferences the iterator
  - `iter++` advances to the next item in the container
  - `iter--` backs up in the container
  - `iter += 2 // impossible`
- The `end()` iterator points to past the end: it should never be dereferenced



Container 내용 검색

Given a vector

```
vector<int> v;
for  (int k=1; k<= 5; k++)
   v.push_back(k*k);
```

Traverse it using iterators:

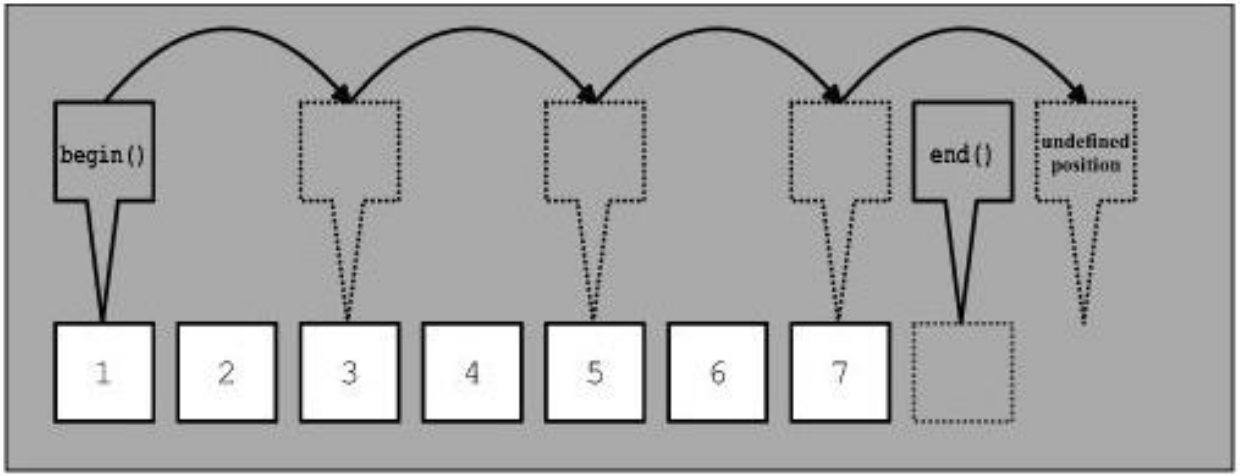
```
vector<int>::iterator iter = v.begin();
while (iter != v.end())
  { cout << *iter << " "; iter++}
```

Prints 1 4 9 16 25

Iterator 예제

```
for (pos = coll.begin(); pos < coll.end(); pos += 2) {
   cout << *pos << ' ';
}
```

- **pos += 2 불가능 (오류)**
- **pos < coll.end() 불가능**



STL 알고리즘

- Many STL algorithms manipulate portions of STL containers specified by a begin and end iterator
- **`max_element(iter1, iter2)`**
  - **`max_element(list.begin(), list.end());`**
- **`min_element(iter1, iter2)`** is similar to above
- **`random_shuffle(iter1, iter2)`** randomly reorders the portion of the container in the given range
- **`sort(iter1, iter2)`** sorts the portion of the container specified by the given range
  - **`Vector`** 등에 사용가능, **`list`**, **`set`**, 등에는 불가능
  - **`List`**는 자체 **`list.sort()`** 사용

random_shuffle(), erase()

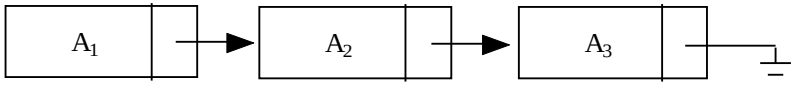
```
int main()
{
    vector<int> vec;
    for (int k = 1; k <= 5; k++)
      vec.push_back(k*k);

    random_shuffle(vec.begin(),vec.end());

    vector<int>::iterator p = vec.begin();
    while (p != vec.end())
    {
        cout << *p << "  ";
        p++;
    }

    vec.erase(vec.begin()+1); // erase second one

    return 0;
}
```

리스트(List)

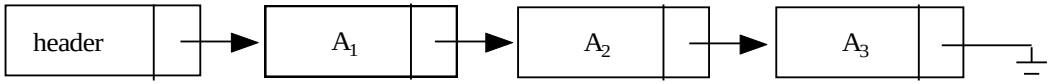
- 사용 가능한 연산(operation):
  - insert: 데이터를 추가
  - remove: 특정 데이터 값을 삭제
  - erase : 특정 iterator의 위치를 삭제
  - sort : 데이터를 정렬함
- 각 데이터는 고정된 위치를 가짐
- 두 가지 구현 방법:
  - 배열 기반 리스트(Array-based list)
  - 연결 리스트(Linked list)

리스트(List)

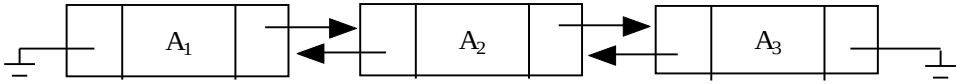
Linked list:



Linked list with a header:



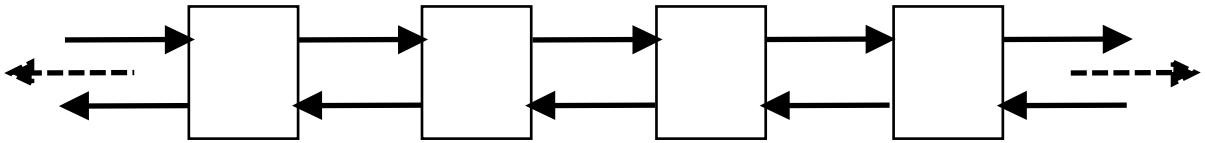
Doubly linked list:



C++의 리스트(List)

- 이중(doubly) 연결 리스트의 형태로 구현
  - 리스트의 각 노드(node)는 자체적인 메모리 공간에 자신의 앞과 뒤의 노드에 대한 링크를 가짐
  - 단점: 임의 접근(random access)를 제공할 수 없음
    일반적으로 임의의 노드에 접근하기 위해서는
    직선적 시간(linear time)이 걸림
    - 따라서 [ ] 연산자를 지원하지 않음
  - 장점: 어떠한 위치에서든 특정 노드의 삽입 또는 삭제가 빠름

- http://www.cplusplus.com/reference/stl/list/



정렬(Sorting)

- 컴퓨터 과학에서 가장 기본적인 알고리즘
  - 내부 정렬(Internal sorting):
    - 비교적 적은 양의 데이터를 정렬
    - 메인 메모리 내에서 수행 가능
  - 외부 정렬(External sorting):
    - 대용량의 데이터를 정렬
    - 메인 메모리 내에서 수행할 수 없음
    - 보조 기억장치 사용(디스크나 테이프 등)

정렬 알고리즘의 복잡도(Complexity)

- 알고리즘의 복잡도(Complexity)
  - 복잡도 계산
    - 최악의 경우(worst case)
    - 최고의 경우(best case)
    - 평균적인 경우(average case)
- 시간 복잡도(Time complexity)
  - 키(Key)만을 비교하면 최소한 $\Omega(n \log n)$의 비교가 필요
- 공간 복잡도(Space complexity)
  - 메모리 사용(Memory usage)을 의미
  - “in place” 알고리즘
    - O(1) 또는 O(log n)과 같은 알고리즘
    - 정렬하는 데이터 이외의 추가적인 메모리를 사용하지 않음
  - “not in place”(or “out of place”) 알고리즘
    - 정렬 중에 정렬하는 데이터 이외의 추가적인 메모리가 필요함

정렬 알고리즘의 안정성(Stability)

- 안정성(Stability)
  - 안정적인 정렬 알고리즘(stable)
    - 같은 키(i.e. values)를 가진 아이템의 순서를 정렬 후에도 정렬 전과 같게 유지
  - 불안정적인 정렬 알고리즘(Unstable)
    - 같은 키(i.e. values)를 가진 아이템의 순서가 정렬 후에는 정렬 전과 달라질 수 있음
    - 일부 수정하여 안정성을 가지게 구현 가능

단순한 정렬 알고리즘

- 거품 정렬(Bubble sort 또는 Sinking sort)
- 선택 정렬(Selection sort)
- 삽입 정렬(Insertion sort)
  - Best case: $O(N)$
  - Worst case: $O(N^2)$
  - Average case: $\Theta(N^2)$
- 모두 $O(N^2)$의 시간 복잡도를 가짐

거품 정렬(Bubble sort)

- 가장 간단하지만 비효율적인 알고리즘
- 인접한 두 요소들을 비교해 정렬이 되어 있지 않을 경우에 서로 위치를 바꾸는 방식
- 속도는 $O(N^2)$

Pseudo code:

```
BubbleSort(A)
  for i := length[A]-1 에서 1 까지
    for j := 0 부터 i-1 까지
      if (A[j] > A[j+1]) {    // 내림차순일 때는 ">" 를 "<" 로 바꾸면 됨
        temp := A[j]
        A[j] := A[j+1]
        A[j+1] := temp
      }
```

실습 6. 거품 정렬 구현하기

- 아래를 참조하여 거품 정렬 함수를 구현하여 보자.

```
BubbleSort(A)
  for i := length[A]-1 에서 1 까지
    for j := 0 부터 i-1 까지
      if (A[j] > A[j+1]) {   // 내림차순일 때는 ">" 를 "<" 로 바꾸면 됨
        temp := A[j]
        A[j] := A[j+1]
        A[j+1] := temp
      }
```

좀 더 복잡한 정렬 알고리즘

- 병합 정렬(Merge sort)
  - 하나의 리스트를 두 개의 같은 크기의 리스트로 분할한 뒤 각각의 리스트를 정렬하여 합쳐서 전체를 정렬하는 방법
  - 분할 정복(divide and conquer)의 좋은 예
  - 안정적(stable)
  - 외부 정렬 알고리즘의 토대
- 힙 정렬(Heap sort)
  - 리스트를 이진 트리로 작성하고 이에 기초하여 정렬
  - 자료구조 힙(heap) 을 사용
  - 불안정적(unstable)
- 실행시간: O(NlogN)

퀵 정렬(Quicksort)

- 일반적인 경우 가장 빠른 정렬 알고리즘
  - 분할 정복 재귀호출 알고리즘
- 평균 실행 시간 : O(NlogN)
- 최악의 실행시간: $O(N^2)$
- 불안정적인 알고리즘
- 배열 S를 Quicksort :
  - 만약 S 의 원소의 수가 0 또는 1이면 반복을 멈춤
  - S의 임의의 원소 v를 선택. 이것을 피벗(*pivot*)이라고 부름
  - $S-\{v\}$ 를 두 개의 그룹으로 나눔
    - $S_1 = \{x \in S-\{v\} | x \leq v\}$,
    - $S_2 = \{x \in S-\{v\} | x \geq v\}$
  - {quicksort (S_1) 그 다음에 v 그 다음에 quicksort (S_2)}를 반환

Quicksort

- divide the array into two partitions and then sort each partition recursively

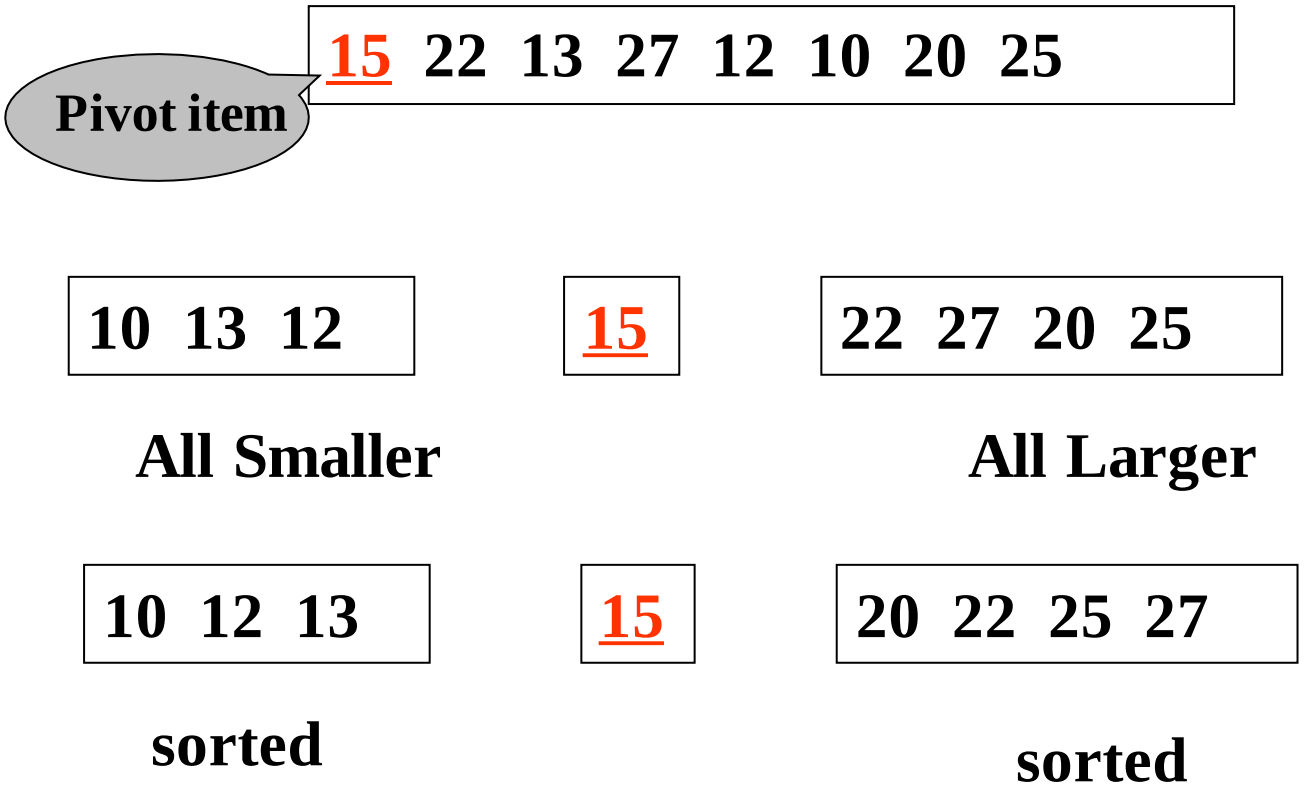


Illustration of Quicksort

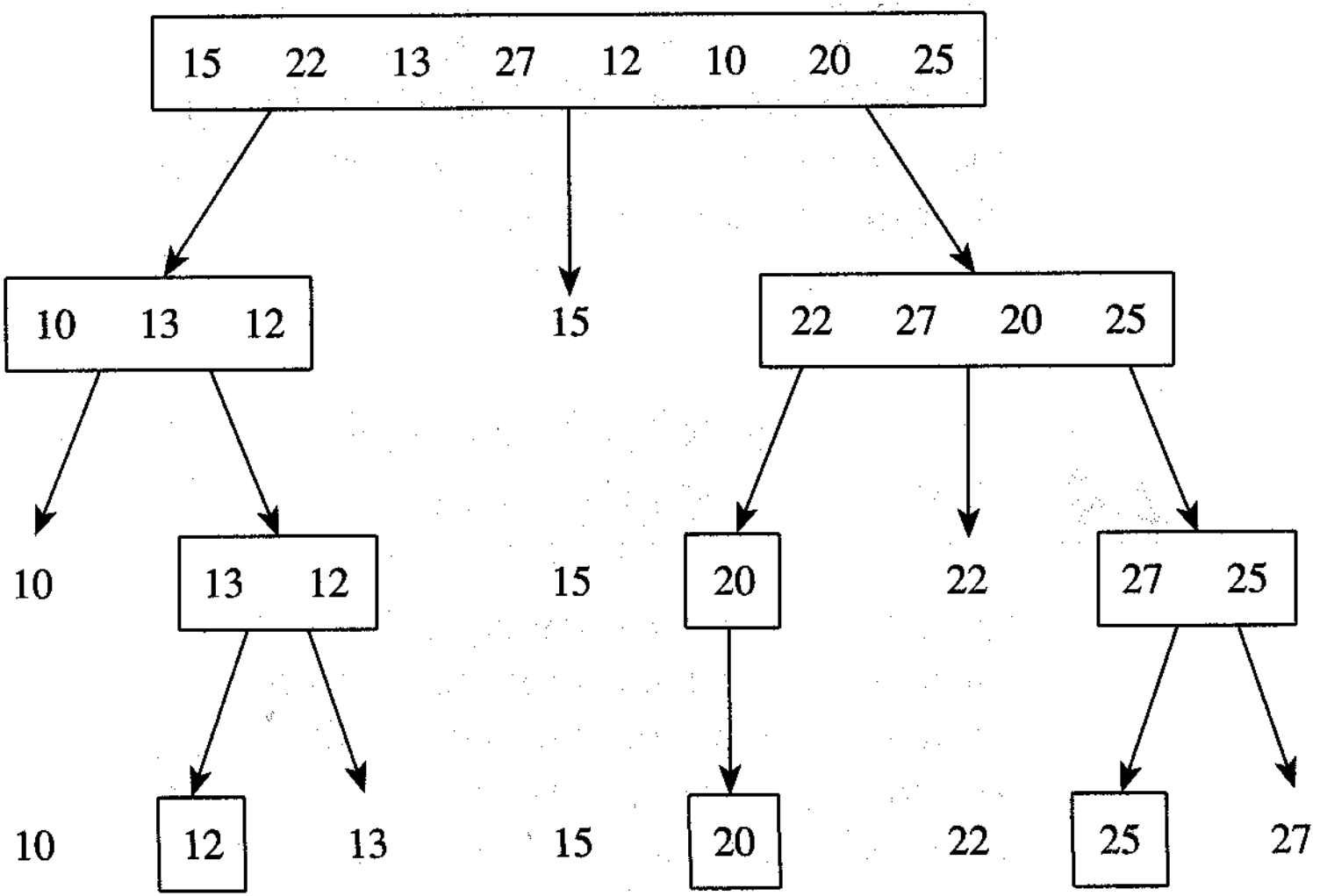


Figure 2.3 The steps done by a human when sorting with Quicksort. The subarrays are enclosed in rectangles whereas the pivot points are free.

Quicksort

Algorithm *Quicksort*

Problem : sort n keys in non-decreasing sequence

Inputs : positive integer n, array of keys S indexed from 1 to n

Outputs : the array S containing the keys in non-decreasing order

```
void quicksort (int low, int high)
{
        int pivotpoint;

        if ( high > low) {
                partition(low, high, pivotpoint);
                quicksort(low, pivotpoint-1);
                quicksort(pivotpoint+1,high);
        }
}
```

Partition

Algorithm *Partition*

```
void partition (int low, int high, int& pivotpoint) {
        int i, j;
        keytype pivotitem;

        pivotitem = S[low];
        j = low;
        for (i = low+1; i <= high; i++)
            if ( S[i] < pivotitem) {
                    j++;
                    exchange S[i] and S[j];
            }
        pivotpoint = j;
        exchange S[low] and S[pivotpoint];
}
```

j : 작은값 마지막

pv	작은값	큰값	미검사 항목

i : 작은값인 경우

Pivot보다 작은값을 왼쪽에 유지하며
작은 값이 나오면 작은 값 영역에 넣음

실습 7. Quick Sort 구현하기

- 아래를 참조하여 Quick Sort 함수를 구현하여 보자.

```
void quicksort (int low, int high)
{
    int pivotpoint;

    if ( high > low) {
        partition(low, high, pivotpoint);
        quicksort(low, pivotpoint-1);
        quicksort(pivotpoint+1,high);
    }
}
```

```
void partition (int low, int high, int& pivotpoint)
{
    int i, j;
    keytype pivotitem; // int pivotitem;

    pivotitem = S[low];
    j = low;
    for (i = low+1; i <= high; i++)
        if ( S[i] < pivotitem) {
            j++;
            exchange S[i] and S[j]; // swap(S[i],S[j]);
        }
    pivotpoint = j;
    exchange S[low] and S[pivotpoint];
}
```

정렬 라이브러리(Sorting Library)

- C에서 정렬과 검색

• 헤더 : stdlib.h

• 정렬(퀵 정렬):

void qsort(void *base, size_t nmemb, size_t size, int (*compare) (const void *, const void *));

- base로부터 시작되는 크기가 size인 nmemb개의 원소들을 가진 배열을 compare 함수를 사용하여 정렬.

• 이진 검색(binary search):

void *bsearch(const void *key, const void *base, size_t nmemb, size_t size,
int (*compare)(const void *, const void *));

- base로부터 시작되는 크기가 size인 nmemb개의 원소들을 가진 배열을 compare 함수를 사용하여 key 타입의 키를 비교하여 검색.

qsort( ) 예제

```
#include <iostream>
#include <string.h>
#include <stdlib.h>
using namespace std;

// 정렬할 원소들의 크기를 비교하는 함수
static int cmpr(const void *a, const void *b) {
   int numa = *(int *)a;
   int numb = *(int *)b;

   if ( numa < numb) return -1;
   else if ( numa > numb ) return 1;
   else return 0;
}

// main에서 호출되는 정렬 함수
void sortintarr(void *array, unsigned n) {
    qsort(array, n, sizeof(int), cmpr);
}
```

```
int main(void) {
 int num, line[1024];
 int count = 0;
 // 정렬할 값들을 입력받음
 while(1) {
     scanf("%d", &num);
     if( num == 0 )
         break;
     line[count++] = num;
 }
sortintarr(line, count); // 정렬
// 정렬한 값들을 출력
 for(int i = 0; i < count; i++)
    printf("%d ", line[i]);

 return 0;
}
```

C++에서 정렬과 검색

- STL은 정렬,검색 등등의 함수를 가짐

```
void sort(RandomAccessIterator bg, RandomAccessIterator end);

void sort(RandomAccessIterator bg, RandomAccessIterator end,
          BinaryPredicate op);
```

• 안정적인 정렬 함수를 지원

```
void stable_sort(RandomAccessIterator bg, RandomAccessIterator end);
void stable_sort(RandomAccessIterator bg, RandomAccessIterator end,
                 BinaryPredicate op);
```

Java에서 정렬과 검색

- java.util.Arrays

static void sort(Object [] a)

static void sort(Object [] a, Comparator c)

static int binarysearch(Object [] a, Object key)

static int binarysearch(Object [] a, Object key, Comparator c)

java.util.Arrays의 정렬 함수들은 모두 안정적인 함수

실습 8. 통계 프로그램

- 책에서 쓰이는 단어들 중 어떤 단어가 가장 많이 사용되는지 알 수 있도록 통계 프로그램을 만들어야 한다. “quit” 이 나올 때까지 모든 단어를 알파벳 순서로 출력하는 프로그램을 작성한다.
- 입력예

```
twas brilling and the slithy toves
did gyre and gimble in the wabe
```

- 출력예

```
and 2
brilling 1
did 1
gimble 1
gyre 1
in 1
slithy 1
the 2
toves 1
twas 1
wabe 1
```