



# Program Patterns: struct, Search Patterns (2)

---



# Array of Structures



# Array of Structures

- “data type” “array\_name” “[“max\_size”]”

```
struct EMPRECORD employee[250];
```

- Real power of structures
  - combined with looping, if-else (switch), functions
  - search, sort, compute, display,...

# 50 States of the United States

NOTE: ALASKA & HAWAII NOT TO SCALE

4





# Array of Structs



```
struct {  
    char name[20];  
    char abbr[2];  
    int  population;  
    float area;  
    double GDP;  
    char governor[20];  
} US_States[50];
```



## Using the Array of structs

---

- Find the state with the highest population
- Find 5 states with the highest GDP
- Find states with the same first character
- Sort states in alphabetical order of names
- ...



# Array of Structs

---

```
struct {  
    char name[20];  
    int age;  
    float salary;  
    char hobby[3][20];  
} employee[300];
```



# Table of Employees

name	age	salary	hobby
Kim	25	200	basketball
Lee	30	300	swimming
Park	23	250	music, soccer
Cho	40	500	sleeping
...			
Chung	28	900	game, cooking





## Exercise 1



- level 1 : static allocation
- level 2 : scanf allocation
- level 3 : dynamic allocation

```
struct EM* employee = (struct EM*) malloc(sizeof(struct EM) * max_length)
```

- print array of structs



## Using an Array of structs

---

- Find the 3 highest-paid employees
- Find employees who like basketball
- Find all employees named Kim
- Sort employees in the age order
- ...



# Exercises (Textbook Chapter 12)

---

- Programming Exercise 12.2 1
  - Define an array of 4 structs
    - struct MONTH\_DAYS

```
{  
    char month_name[10];  
    int  days;
```
  - Initialize the array (with your own data).
  - Print the name and days of each month.



# Programming Exercise

- 12.2 1. Using the following declaration

```
struct MonthDays
```

```
{
```

```
    char name[10];
```

```
    int days;
```

```
}
```

define an array of 12 structures of type MonthDays. Name the array **convert[]**, and initialize the array with the names of the 12 months in a year and the number of days in each month.

- Include the array in a program that displays the name and number of days in each month.



# Exercises



- Programming Exercise 12.2 3
  - Define an array of 6 structs for employees.
    - struct has 4 members: last **name** (char[20]), **ID** (int), **pay\_rate** (float), **hours\_worked** (float)
  - Initialize each struct with data (given in the book).
  - Calculate the total pay for each employee.
  - Print the name, ID, and total pay for each employee.

# Programming Exercise

- 12.2 3a. Declare a single structure type suitable for an employee record consisting of an integer identification **number**, a **last name** (consisting of a maximum of 20 characters), a **floating-point pay rate**, and a **floating-point number of hours** worked.
- 12.2 3b. Using the structure, write a C program that interactively accepts the following data into an array of six structures:

ID Number	Name	Pay Rate	Hours Worked
3462	Jones	4.62	40.0
6793	Robbins	5.83	38.5
6985	Smith	5.22	45.5
7834	Swain	6.89	40.0
8867	Timmins	6.43	35.5
9002	Williams	4.75	42.0

- Once the data have been entered, the program should create a payroll report listing each employee's name, number, and gross pay. Include the total gross pay of all employees at the end of the report.



## struct with a Pointer Member

---

```
struct {  
    char    name[20];  
    int     age;  
    float   GPA;  
    int     *grade_ptr;  
};
```



## Example: Defining a Data Node (1/2)

---

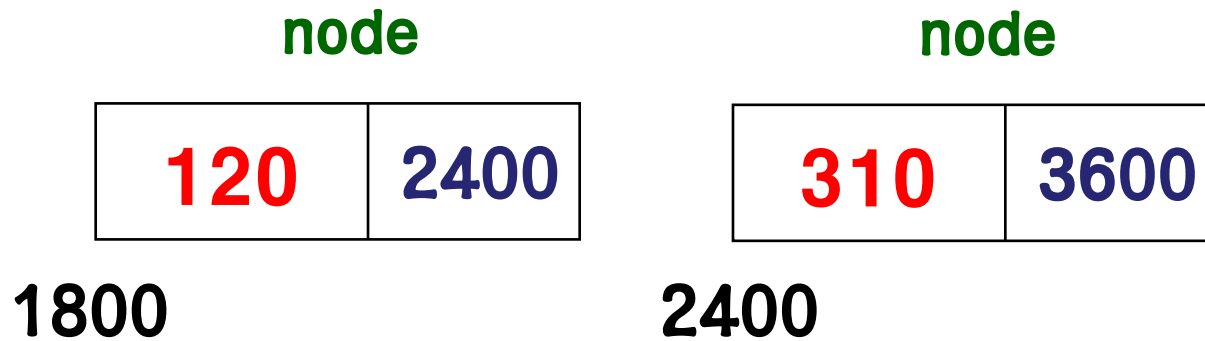
```
struct NODE {  
    int      key;  
    struct NODE *next;  
};
```





## Example: Defining a Data Node (2/2)

```
struct NODE {  
    int          key;  
    struct NODE  *next;  
};
```





## Defining a Data Node Array

```
struct NODE {  
    int key;  
    struct NODE *next;  
} node[3];
```

node[0]

100	2400
-----	------

1800

node[1]

250	3600
-----	------

2400



# Creating (Filling) a Data Node Array

```
node[0].key = 100;  
node[1].key = 250;  
node[2].key = 467;  
node[0].next = node[1].next = node[2].next = NULL;
```

node[0]

100	null
-----	------

node[1]

250	null
-----	------

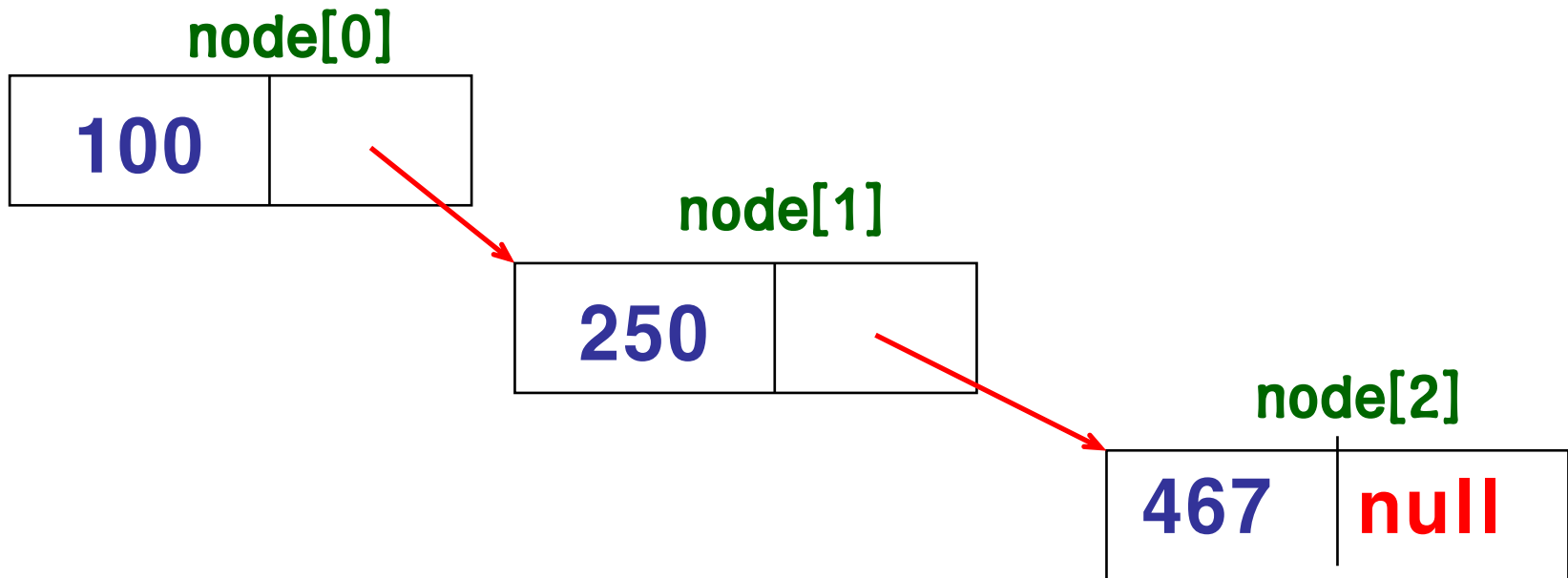
node[2]

467	null
-----	------

# Linking the Data Nodes

(linked list data structure -- later)

```
node[0].next = &node[1];  
node[1].next = &node[2];
```

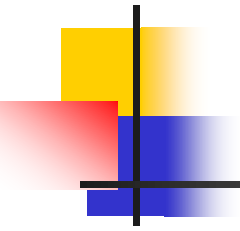




## Exercise

---

- Define an array of 5 structs, and store any string as a key in each node. Then link the five nodes.



# Structures and Functions



## Function Call with a struct as an Argument

---

- Call by Value

- pass a copy of a structure
- pass a copy of a member of a structure
- Function may return a structure or a member of a structure.

- Call by Reference

- Explicitly pass the address of a structure. (unlike passing an array)
- Function may make changes to the structure.



## Function Call with a struct as an Argument

---

- Define struct data type globally
  - for it to be known to the function being called





# Call by Value: (single struct)

```
struct EMPRECORD {  
    char name[20];  
    int age;  
    float salary;  
    char hobby[3][20];  
} employee, newemp;
```

...

```
new_emp = update_records (employee); /* function call */
```

-----

```
struct EMPRECORD  update_records /* function definition */  
    (struct EMPRECORD emp) {  
        emp.age = 25;  
        return emp;  
    }
```



# Call by Reference: (single struct)

```
struct EMPRECORD {  
    char name[20];  
    int  age;  
    float salary;  
    char hobby[3][20];  
} employee;
```

...

```
update_records (&employee);    /* function call */
```

```
-----  
void  update_records    /* function definition */  
    (struct EMPRECORD *emp) {  
        (*emp).age = 25;  
    }
```



# Struct and Pointer

---

- `(*emp).age` vs. `*emp.age`
  - different
- `(*emp).age` vs. `emp -> age`
  - same



# Call by Reference: (array of structs)

---

```
struct EMPRECORD {  
    char name[20];  
    int  age;  
    float salary;  
    char hobby[3][20];  
} employee[300];
```

...

```
update_records (employee); /* function call */
```

-----

```
void  update_records /* function definition */  
    (struct EMPRECORD emp[]) {  
        (emp[100]).age = 25;  
    }
```



# Homework

---

- Chapter 12 Programming Exercises
  - 12.2 #4 (5 points)
  - 12.3 #1, 3, 4 (total 15 points)



# Programming Exercises

- 12.2 4a Declare a single structure type suitable for a car record consisting of an integer car identification number, an integer value for the miles driven by the car, and an integer value for the number of gallons used by each car.
- 12.2 4b Using the structure, write a C program that interactively accepts the following data into an array of five structures:

<b>Car Number</b>	<b>Miles Driven</b>	<b>Gallons Used</b>
<b>25</b>	<b>1,450</b>	<b>62</b>
<b>36</b>	<b>3,240</b>	<b>136</b>
<b>44</b>	<b>1,792</b>	<b>76</b>
<b>52</b>	<b>2,360</b>	<b>105</b>
<b>68</b>	<b>2,114</b>	<b>67</b>

- Once the data have been entered, the program should create a report listing each car number and the miles per gallon achieved by the car. At the end of the report, include the average miles per gallon achieved by the five cars.



# Programming Exercises

- 12.3 1. Write a C function named **Days()** that determines the number of days from the date 1/1/2000 for any date passed as a structure. Use the following Date structure:

```
struct Date
{
    int month;
    int day;
    int year;
}
```

In writing the **Days()** function, assume that all years have 360 days and each month has 30 days. The function should return the number of days for any date structure passed to it.

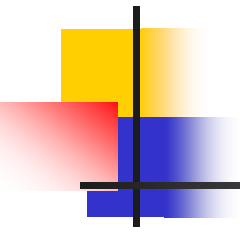


# Programming Exercises

---

- 12.3 3. Rewrite the `Days()` function so that it directly accesses a `Date` structure, as opposed to receiving a copy of the structure.
- 12.3 4a Write a C function named `recent()` that returns the later date of any two dates passed to it. For example, if the dates 10/9/2001 and 11/3/2001 are passed to `recent()`, the second date would be returned.
- 12.3 4b Include the `recent()` function in a complete program. Store the data structure returned by `recent()` in a separate date structure and display the member values of the returned date.





# Program Patterns



# Programming Patterns

---

- Computers are used for
  - computation and
  - data processing
- Computers were created to “computing”.
- But today computers are used predominantly for “data processing”.



# Data Processing Patterns

---

- There are several frequently used patterns in the design and development of data processing software.
- In this course, we will learn study and think about six of them.
  - Data Search
  - Data Update/Change
  - Data Copying & Moving
  - Data Derivation
  - Data Transformation
  - Data Reorganization



# Data Search

---

- search for max/min
- element count
- Boolean predicate-based search
- string match
  - exact match
  - partial match
  - approximate match



# Data Update

---

- insert
- update/replace
- delete
- append, concatenate
- filling in missing data
  
- multiple qualifying data
- constraints on data update
  - unique, null, data typing, value range
  - aggregate



# Data Copying & Moving

---

- data copying/replication
- data subsetting
- data appending
- data moving
- data compaction



# Data Derivation

---

- data aggregation
  - total, average
- data versioning
- data differential
- data sampling
- data lineage (data lifecycle)



# Data Transformation

---

- data conversion (low level)
  - integer to string, string to integer
  - integer/float to categorical data
  - date/time, money, measurement units checksum computation
- data compression and decompression
- data encryption and decryption





# Data Reorganization

---

- sorting
- grouping
- vertical decomposition
- horizontal decomposition
- recomposition
- data structure change



# Data Processing Patterns

---

- Data search
- Data update
- Data copying & moving
- Data derivation
- Data transformation
- Data reorganization



# Data Search

---

- search for max/min
- element count
- Boolean predicate-based search
- string match
  - exact match
  - partial match
  - approximate match



# Data Search Patterns

---

- Predicate-based Search
- Exact match
- Partial match

# Search for Exact Match

- Given a string, and a search string, determine if the string contains the search string.
- string
  - Is your name Bob?
  - No, my name is Rob.
  - Hello, Rob.
- search string: "name"

found!

I	s		y	o	u	r		n	a	m	e		B	o	b	?	
---	---	--	---	---	---	---	--	---	---	---	---	--	---	---	---	---	--

found!

N	o	,		m	y		n	a	m	e		i	s		R	o	b	.
---	---	---	--	---	---	--	---	---	---	---	--	---	---	--	---	---	---	---

not found

H	e	l	l	o	,		R	o	b	.
---	---	---	---	---	---	--	---	---	---	---



## Example 1: Problem

---

- Given a string, find all sub-strings that match a search string.
- **string**
  - "A thief named **hong gil dong** lived with friends named **hong gil don** and **hong gil ja** in a village named **hong gil dong village**."
- **search string**
  - "**hong gil dong**"



# (Reminder) Software Development Steps

---

- Follow the steps of program development
  - Step 1: Understand the problem
  - Step 2: Outline a solution (including logic sketch)
  - Step 3: Form a program structure
  - Step 4: Write a pseudo code (including logic sketch)
  - Step 5: Write the program
  - Step 6: Inspect the program
  - Step 7: Compile the program
  - Step 8: Test the program
  - Step 9: Document the source code
  - Step 10: Maintain the source and test cases



# Step 1: Understand the problem

---

- Make up a search string
  - "hong gil dong"
- Think about various match cases
  - no match, partial match, exact match
- Make up an example string (with various cases)
  - " A thief named **hong gil dong** lived with friends named **hong gil don** and **hong gil ja** and **hhong gil dong** and **kong gil dong** and **honggil dong** and **hong gil donggg** in a village named **hong gil dong village**."





## Step 2: Outline a Solution

---

- Loop through the characters in the string
  - outline of the match logic
    - find a character that matches the first character of the search string
      - If found, use a loop to find out if the characters that start from the first character exactly match the search string
        - If a match is found, increment count by 1



## Using a “Sliding Window”

---

A thief named hong gil dong lived with friends

**hong gil dong** ← search string

named hhhong gil dong and hong gil don in  
a village named hong gil dong village.



## Using a “Sliding Window” (cont’d)

---

A thief named hong gil dong lived with friends  
named hhhong gil dong and hong gil don in  
hong gil dong  
a village named hong gil dong village.



## Using a “Sliding Window” (cont’d)

---

A thief named hong gil dong lived with friends  
named hhhong gil dong and hong gil don in  
hong gil dong  
hong gil dong  
a village named hong gil dong village.



## Using a “Sliding Window” (cont’d)

A thief named hong gil dong lived with friends  
named hhhong gil dong and hong gil don in  
hong gil dong  
hong gil dong  
hong gil dong  
a village named hong gil dong village.

# Match Logic

cursor



named hhhong gil dong and

hong gil dong

i



```
/* use a cursor to advance the string by one char */  
/* use i to advance the search string */  
check for “\0” in string
```

```
int cursor=0, i;  
if string[cursor] == search_string[i]  
    if i == strlen(search_string) { /* exact match */  
        match found  
        i=0} /* for next match */  
    else i++  
cursor++ /* move the cursor on the string */
```



## Step 3: Form a Program Structure

---

read string

read search-string

search for match /\* the heart of the program \*/

print count



## Step 4: Write a Program Outline

---

read string

read search-string

/\* search for match \*/

for loop (1 through string length)

if first character of search-string found

for loop (1 through length of search-string)

if substring matches the search-string, found

if found

increment count

print count /\* print number of sub-strings that match  
the search string \*/





# Search for Partial Match

---

- Search using wildcard \*
- search key: "Hong\*Dong"
- any string that starts with "Hong" and ends with "Dong"
- \* (wildcard) means any string of any length



## Example (1/2)

---

- string: "hello mister monkey"
- search string: "money"
- result: match not found
- search string: "mon\*ey"
- result: match found
- search string: "m\*y"
- result: match found



## Example (2/2)

---

- string: "my name is lee jongho"
  - search string: "lee \*ho"
  - result: match found
- 
- search string: "lee \*ha"
  - result: match not found



## Exercise 2: Problem

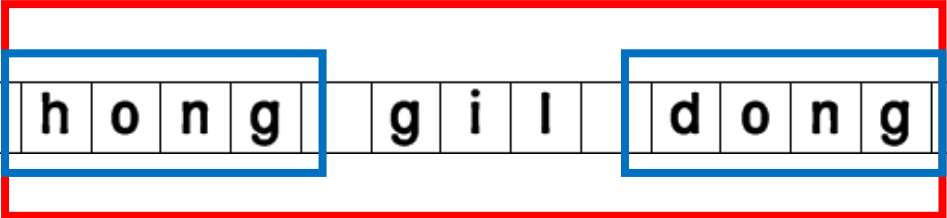
---

- Read a string, and store it.
- Read a search string.
- Determine the number of sub-strings that match the search string, allowing the use of a wildcard \*
- Assumption: Wildcard is used only in the middle of a string, and only once (e.g. hong\*dong)
  - \*dong (x), dong\* (x), \* (x), a\*b\*c (x)

# Step 1: Understand the problem

- Make up a search string: "hong\*dong"
- Think about various match cases.
- Make up an example string (with various cases).
  - "My name is hong gil dong. My brother is hong je dong. My sister is hong gilja, and her friend is hongdong."

My name is hong gil dong.



found!



## Step 2: Outline a Solution

---

- Divide the search string into two parts
  - hong\*dong → first part: hong, second part: dong
- Loop through the characters in the string
  - Logic
    - Find substring that matches the first part of the search string  
My name is hong gil dong.
    - If found, start from the end of the first part and find substring that matches the second part of the search string  
My name is hong gil dong.
    - If second part of the search string is found, increment count by 1 and go to the next part of the string



## Step 3: Form a Program Structure

---

read string

read search-string

divide the search string

search for match /\* heart of the program \*/

print count



## Step 4: Write a Program Outline

---

read string

read search-string

divide the search string

/\* search for match \*/

for loop

    find the first part of the search string

    if found

        for loop

            find the second part of the search string

            if found

                increment count and go to the next part of the string

print count   /\* print number of sub-strings that match  
                    the search string \*/

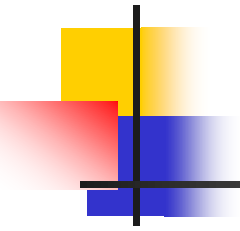




## Homework (30 points)

---

- Complete Examples 1 and 2 (Do steps 5-9)
- Notes
  - Be sure to document the code.
  - Run the program and screen capture the results.
  - Run the program with 5 different, carefully chosen test datasets (strings and search strings for each string).
  - \* Note: Write a function and call it 5 times, each time with a different string and corresponding search strings.



# Search Patterns (continued)



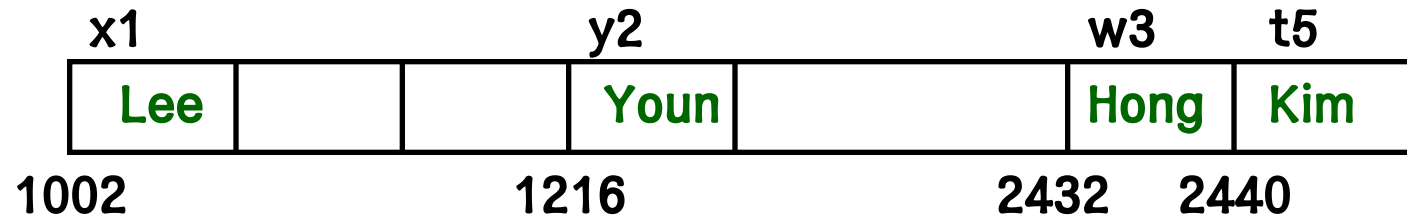
# Search Scope

---

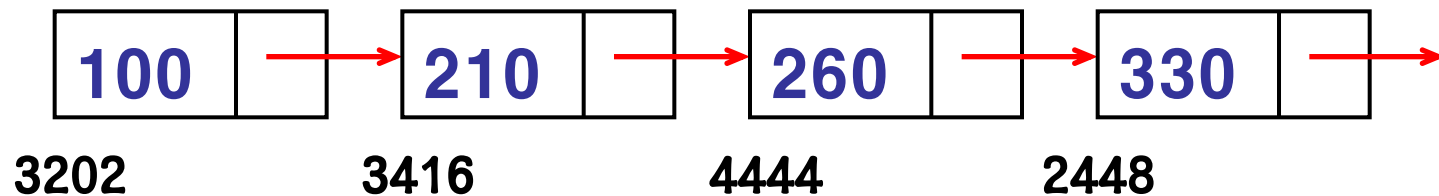
- Exhaustive search
  - look at all the data
- Index-based search
  - zero in on selected data

# Exhaustive Search

## Data in Array



## Data in Linked List





# Index-Based Search

---

## ■ Index

- Data is stored in some data structure
  - e.g, struct array, linked list, heap, tree,...
- Index is a separate data structure
- It maintains (key, key address) pairs for the data.
- It is useful to get direct access to the data using the key.

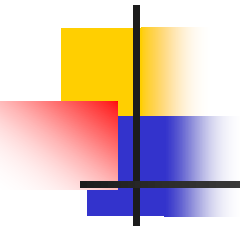


# Index (for Data Stored in an Array)

## Data in Array

x1		y2		w3	t5
Lee		Youn		Hong	Kim
1002		1216		2432	2440

	Key	Address
Index	Hong:	2432
	Kim:	2440
	Lee:	1002
	Youn:	1216



<b>100</b>	<b>Hong gil dong</b>	<b>22</b>	<b>night worker</b>
------------	----------------------	-----------	---------------------

2432



# Creating & Maintaining an Index

---

- Array of **struct** with 2 members
  - key and memory address of the key
  - (e.g.) **struct** {  
    **int** key;  
    **int** \*key\_ptr;  
} **index**[1000];
- Index needs to be changed when data is updated, inserted, or deleted.





## Lab (10 points): Index Search

- Write the following C program:
  - Store the dataset (on the next page) in a struct array.
    - Each line contains **name**, **age** and **hobby**
  - Create an index (struct array) by **name** in the order in the dataset (shown on the next page)
  - (\* In the index, the names are stored in a sort order; but for this Lab, there is no need to sort the names. \*)
  - For names (Lee and Park), do the following:
  - (\* You should write a function and call it twice, once for "Lee" and once for "Park".)
    - Search the index for a name, and find and print the (name) age and hobby in the dataset corresponding to the name.
  - (\* To search the index, a technique named hashing is used. But for this Lab, do a sequential search of the index. \*)



# Lab Dataset, and Index

dataset	array index		
	0	Kim	39
	Tennis		
	1	Ko	15
	Soccer		
index	2	Lee	17
	Soccer		
	3	Choi	21
	Tennis		
	index 4	Park	array index 10
index	Tennis		
	Kim	0	
	Ko	1	
	Lee	2	
	Choi	3	
	Park	4	



# End of Class

---