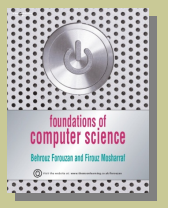


Data Structures

Instructor: Tsung-Che Chiang
tcchiang@ieee.org
Department of Computer Science and Information Engineering
National Taiwan Normal University

Introduction

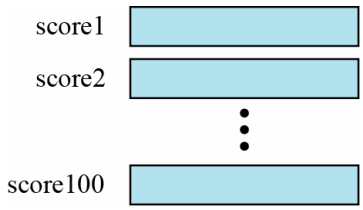
- A data structure uses a collection of related variables that can be accessed individually or as a whole.
- We discuss three data structures here:
  - Arrays
  - Records
  - Linked lists

Arrays

- Imagine that we have 100 scores.

  We need to read them, process them and print them. We must also keep these 100 scores in memory for the duration of the program. We can define a hundred variables, each with a different name.

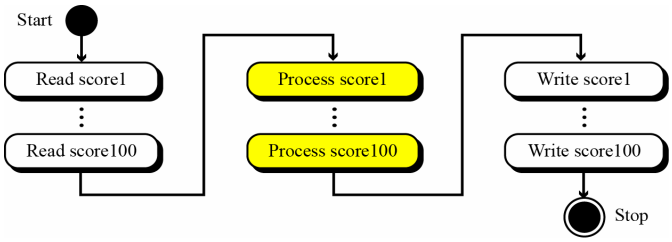
```
int main()
{
    int score1, score2, score3, score4, ...

    return 0;
}
```



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Arrays

- But having 100 different names creates other problems. We need 100 references to read them, 100 references to process them and 100 references to write them.

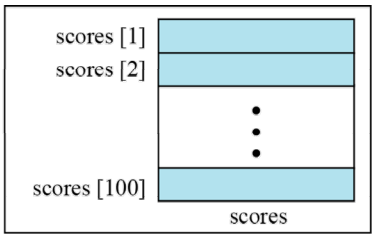



```
    int score1, score2, score3, score4, ...

    scanf("%d", &score1);
    scanf("%d", &score2);
    scanf("%d", &score3);
    scanf("%d", &score4);

    ...
```

Arrays

- An array is a sequenced collection of elements, normally of the same data type, although some programming languages accept arrays in which elements are of different types.
- We can refer to the elements in the array as the first element, the second element and so forth, until we get to the last element.

An array



Note.
In C/C++, the index starts from 0.
(But, do you know why?)

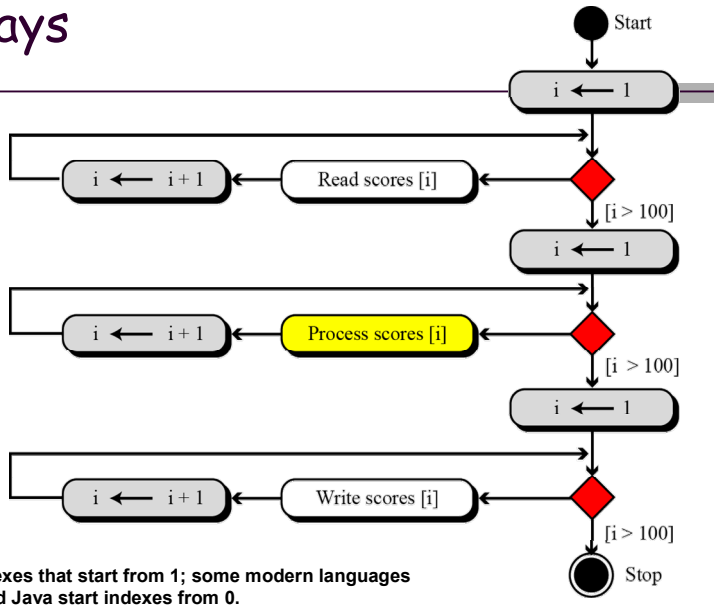
B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Arrays

- We can use loops to read, write, and process the elements in an array.
- Now it does not matter if there are 100, 1000 or 10,000 elements to be processed—loops make it easy to handle them all.

```
int main()
{
    const int DATA_NUM = 100;
    int score[DATA_NUM];

    for (int i=0; i<DATA_NUM; i+=1)
    {
        scanf("%d", &score[i]);
    }

    return 0;
}
```

Arrays



We have used indexes that start from 1; some modern languages such as C, C++ and Java start indexes from 0.

B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Arrays

- Compare the number of instructions we need to write to handle 100 scores without and with the use of an array.
  - Assume that processing each score needs only one instruction.
- Without the use of an array:
  - 100 instructions for reading, 100 instructions for writing, and 100 instructions for processing. ⇒ 300 instructions.
- With the use of an array:
  - 4 instructions in each loop, 3 loops ⇒ 12 instructions

Arrays

- The number of cycles (fetch, decode, and execute phases) the computer needs to perform is not reduced if we use an array.
- The number of cycles is actually increased, because we have the extra overhead of initializing, incrementing and testing the value of the index.

```
6
7     for (int i=0; i<DATA_NUM; i+=1)
8     {
9         scanf("%d", &score[i]);
10    }
11
```

- But our concern is not the number of cycles: it is the number of lines we need to write the program.

Arrays

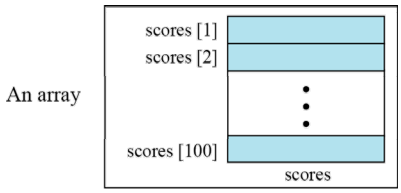
- In computer science, one of the big issues is the reusability of programs.
- Assume we have written two programs to process the scores without and with the use of an array. If the number of scores changes from 100 to 1000, how many changes do we need to make in each program?
  - In the first program we need to add 3 × 900 = 2700 instructions.
  - In the second program, we only need to change three conditions (I > 100 to I > 1000).

If you define a named constant, you only have to modify one place.

Arrays

- Array name vs. element name

```
int main()
{
    int score[100];

    score
    &score
    score[0]
    &score[0]

    score+1
    &score+1
    score[0]+1
    &score[0]+1

    return 0;
}
```

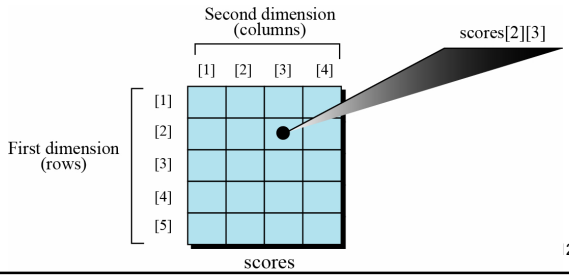


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Make sure that you know the meaning of all expressions on the left-hand side.

Arrays

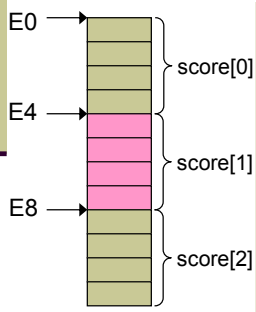
- Multidimensional arrays
  - The arrays discussed so far are known as one-dimensional arrays because the data is organized linearly in only one direction.
  - Many applications require that data be stored in more than one dimension.



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Arrays

- Memory layout
  - The indices in a one-dimensional array directly define the relative positions of the element in actual memory.

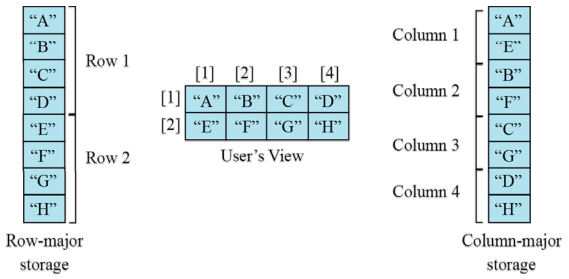



```
5  {
6      int score[100];
7
8      printf("score: %p\n"
9             "&score[0]: %p\n"
10            "&score[1]: %p\n"
11            "&score[2]: %p\n"
12            , score, &score[0], &score[1], &score[2]);
13
14     system("pause");
15     return 0;
16 }
17
18
19
20
```

```
D:\Teaching Course\_Introduction to Compu
score: 0022FDE0
&score[0]: 0022FDE0
&score[1]: 0022FDE4
&score[2]: 0022FDE8
請按任意鍵繼續 . . .
```


Arrays

- Memory layout
  - The following figure shows a two-dimensional array and how it is stored in memory using row-major or column-major storage.
    - Row-major storage is more common.



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Arrays

- We have stored the two-dimensional array `students` in memory.
  - The array is 100 × 4 (100 rows and 4 columns).
  - Assuming that the element `student[1][1]` is stored in the memory location with address 1000.
  - Each element occupies only one memory location
  - The computer uses row-major storage.

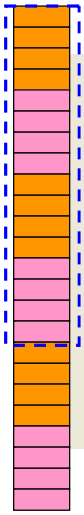
Show the address of the element `students[5][3]`.

x: start address
y: target address

$$y = x + \text{Cols} \times (i - 1) + (j - 1) \quad \Rightarrow 1018$$

Arrays

score[0]



```
int main()
{
    int score[100][4];

    printf("score: %p\n"
           "&score[0][0]: %p\n"
           "&score[0][1]: %p\n"
           "&score[1][0]: %p\n"
           , score, &score[0][0], &score[0][1], &score[1][0]);

    system("pause");
    return 0;
}
```

```
D:\Teaching Course\_Introduction to Computer Science\8_I
score: 0022F930
&score[0][0]: 0022F930
&score[0][1]: 0022F934
&score[1][0]: 0022F940
請按任意鍵繼續 . . .
```

Make sure that you know how to pass 1-D and 2-D arrays to functions in C/C++.

Arrays

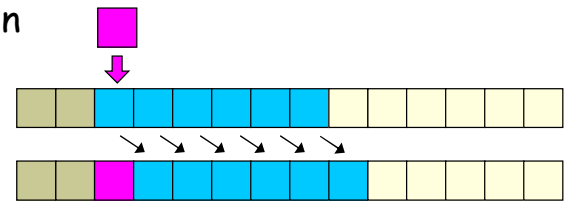
- Operations on array
  - There are some operations that we can define on an array as a data structure.
  - The common operations on arrays as structures are
    - searching
    - insertion
    - deletion
    - retrieval
    - traversal

Many move operations are required and thus inefficient.

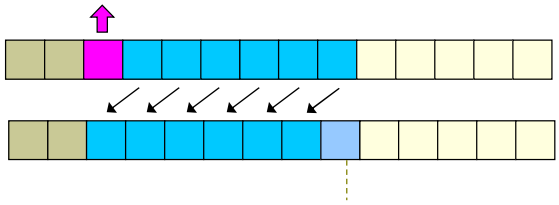
Very efficient due to its ability of random access.

Arrays

- Insertion



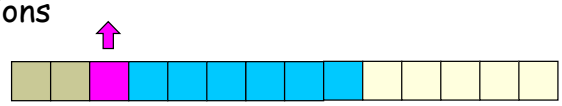
- Deletion



In fact, the data is often left untouched here.

Arrays

- Delete efficiently but cause cost for other operations



We do not move the succeeding elements. We just mark it as deleted.

Cost:

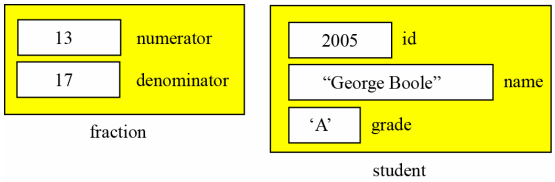
- When doing search, we should check if the element is marked by 'D'.
- We cannot do random access anymore.

Arrays

- Applications
  - If we have a list in which a lot of insertions and deletions are expected after the original list has been created, we should not use an array.
  - An array is more suitable when the number of deletions and insertions is small, but a lot of searching and retrieval activities are expected.

Records

- A record is a collection of related elements, possibly of different types, having a single name.
- Each element in a record is called a field.
  - A field is the smallest element of named data that has meaning. It has a type and exists in memory.
  - Fields can be assigned values, which in turn can be accessed for selection or manipulation.
  - A field differs from a variable primarily in that it is part of a record.



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Records

```
 2 int main()
 3 {
 4     int mary_score[3], john_score[3];
 5     int mary_age, john_age;
 6     char mary_name[20], john_name[20];
 7     double mary_avg_score, john_avg_score;
 8
 9     return 0;
10 }
```

```
44
45 void process(const char name[],
46                int age,
47                const int scores[],
48                double avg_score)
49 {
50 }
51
```

Without records

```
13
14 struct TStudent
15 {
16     int score[3], age;
17     char name[20];
18     double avg_score;
19 };
20
21 int main()
22 {
23     TStudent mary, john;
24
25     return 0;
26 }
```

```
53 void process(TStudent &stud)
54 {
55 }
56
```

With records

Records

- Record name

  Just like in an array, we have two types of identifier in a record:
  - the name of the record
  - the name of each individual field inside the record

- Most programming languages use a period (.) to separate the name of the structure (record) from the name of its components (fields).

```
13
14 struct TStudent
15 {
16     int score[3], age;
17     char name[20];
18     double avg_score;
19 };
20
21 int main()
22 {
23     TStudent mary, john;
24
25     mary.age = 10;
26     mary.score[0] = 99;
27
28     return 0;
29 }
```

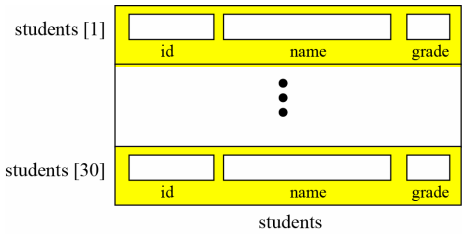

Records

- Arrays vs. Records
  - An array defines a collection of elements (of the same type), while a record defines the identifiable parts of an element.
  - For example, an array can define a class of students (40 students), but a record defines different attributes of a student, such as id, name or grade.

Records

- Array of records
  - For example, in a class of 30 students, we can have an array of 30 records, each record representing a student.

```
struct TStudent
{
    int id, grade;
    char name[20];
};

int main()
{
    TStudent stu[30];

    stu[0].id = 498470001;

    return 0;
}
```



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Records

- Using a loop to read the data into an array of records

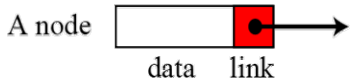
Algorithm 11.2 Part of the pseudocode to read student records

```
i ← 1
while (i < 31)
{
    read (students [i]).id
    read (students [i]).name
    read (students [i]).grade
    i ← i + 1
}
```

B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Linked Lists

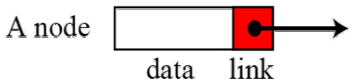
- A linked list is a collection of data in which each element contains the location of the next element.
- The elements in a linked list are traditionally called node.
- Each node contains two parts: data and link.
  - The data hold the value information.
  - The link is used to chain the data together, and contains a pointer (an address) that identifies the next element in the list.



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Linked Lists

```
struct TData
{
    int id, grade;
    char name[20];
};

struct TNode
{
    TData data;
    TNode *link;
};

int main()
{

    TNode node;

    node.data.id = 498470001;
    node.link = 0; // null

    return 0;
}
```



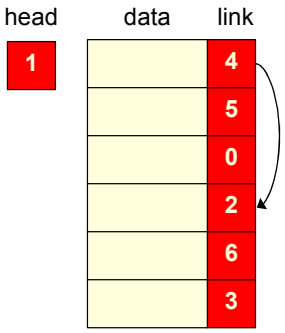
B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

In linked lists, nodes are usually created through dynamic memory allocation.

(`malloc()` in C or `new` in C++)

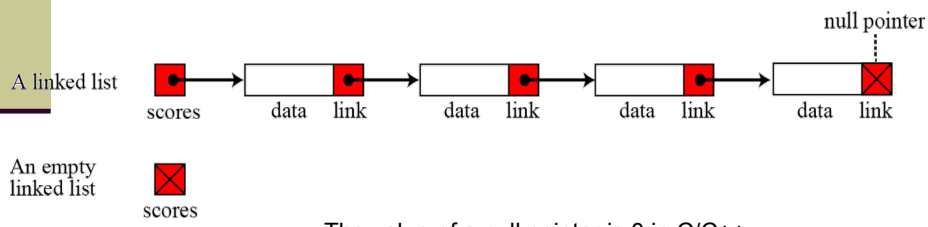
Linked Lists

- It does not mean that there is only one way to implement the linked list.
- We can use array to implement the linked list, but this is less common.



Linked Lists

- A pointer variable identifies the first element in the list. The name of the list is the same as the name of this pointer variable.
- We define an empty linked list to be only a null pointer.



The value of a null pointer is 0 in C/C++.

B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

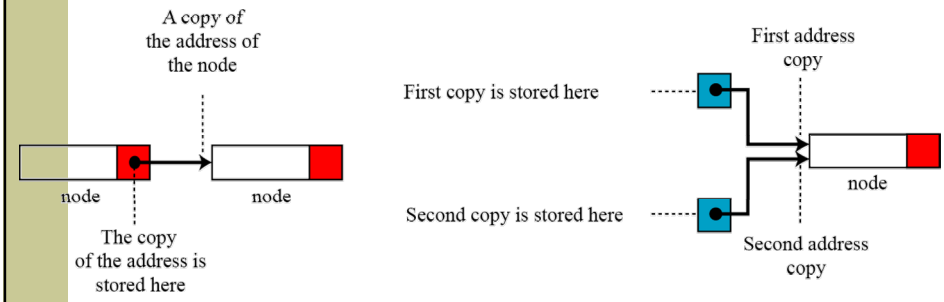
Linked Lists

```
4 struct TData
5 {
6     int id, grade;
7     char name[20];
8 };
9
10 struct TNode
11 {
12     TData data;
13     TNode *link;
14 };
```

```
16 int main()
17 {
18
19     TNode node1, node2;
20
21     node1.data.id = 498470001;
22     node1.link = &node2;
23
24     node2.data.id = 498470002;
25
26     printf("%d", (*node1.link).data.id); // print 498470002
27
28     system("pause");
29     return 0;
30 }
```

You can also write `node1.link->data.id`.

Linked Lists

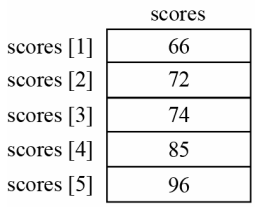


- The arrowhead represents the address of the node to which the arrow head is pointed.
- The solid circle shows where this copy of the address is stored.
- A copy of the address can be stored in more than one place.

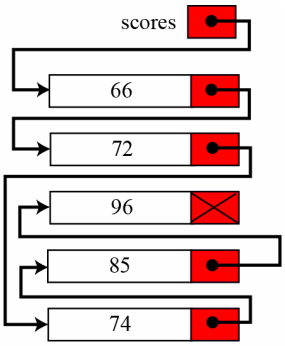
B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Linked Lists

- Arrays vs. Linked lists



a. Array representation



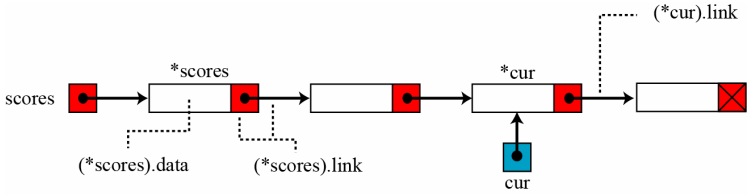
b. Linked list representation

Nodes are not guaranteed to be contiguous in memory.

B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Linked Lists

- List name vs. node name
  - The name of a linked list is the name of the head pointer that points to the first node of the list.
  - Nodes, on the other hand, do not have an explicit names in a linked list, just implicit ones.



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

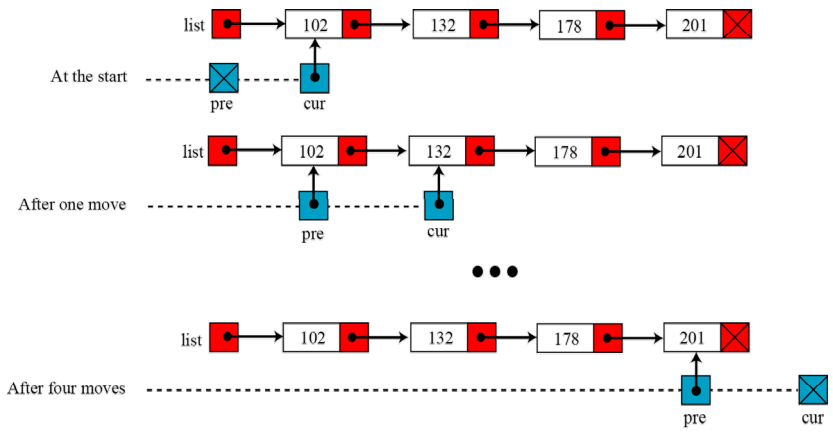
Operations on Linked Lists

Searching a linked list

- Since nodes in a linked list have no names, we use two pointers, `pre` (for previous) and `cur` (for current).
  - At the beginning of the search, the `pre` pointer is null and the `cur` pointer points to the first node.
  - The search algorithm moves the two pointers together towards the end of the list.

Operations on Linked Lists

Searching a linked list



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Operations on Linked Lists

Algorithm 11.3 Searching a linked list

Algorithm: **SearchLinkedList** (list, target)

Purpose: Search the list using two pointers: **pre** and **cur**

Pre: The linked list (head pointer) and target value

Post: None

Return: The position of **pre** and **cur** pointers and the value of the flag (*true* or *false*)

```
{
    pre ← null
    cur ← list
    while ( cur != null && target < (*cur).data )
    {
        pre ← cur
        cur ← (*cur).link
    }
    if (cur != null && (*cur).data = target)  flag ← true
    else  flag ← false
    return (cur, pre, flag)
}
```


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

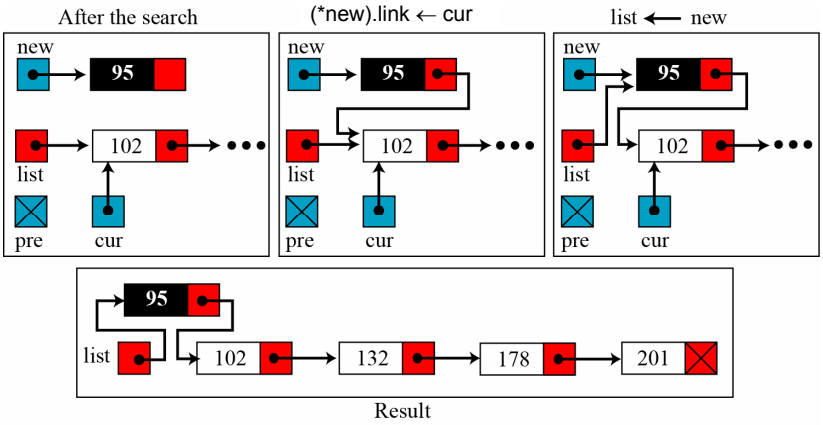
Operations on Linked Lists

Inserting a node

- Before insertion into a linked list, we first apply the searching algorithm.
  - Note that we do not allow data with duplicate values.
- Two cases can arise:
  - Insertion at the beginning of the list. (Inserting into an empty list.)
  - Insertion in the middle of the list. (Insertion at the end of the list.)

Operations on Linked Lists

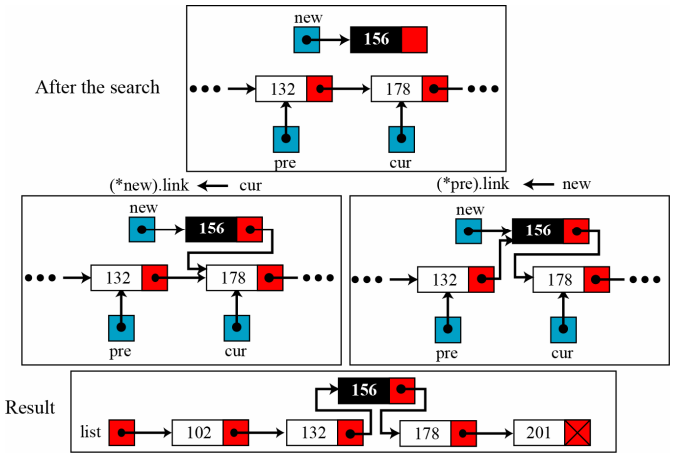
Inserting a node at the beginning



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Operations on Linked Lists

Inserting a node in the middle



B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Operations on Linked Lists

Algorithm 11.4 Inserting a node in a linked list

```
Algorithm: InsertLinkedList (list, target, new)
{
        searchlinkedlist (list, target, pre, cur, flag)
        // Given target and returning pre, cur, and flag

        if (pre = null)                          // Insertion at the beginning
        {
                (*new).link ← cur
                list ← new
                return list
        }

        (*new).link ← cur                        // Insertion in the middle
        (*pre).link ← new
        return list
}
```


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

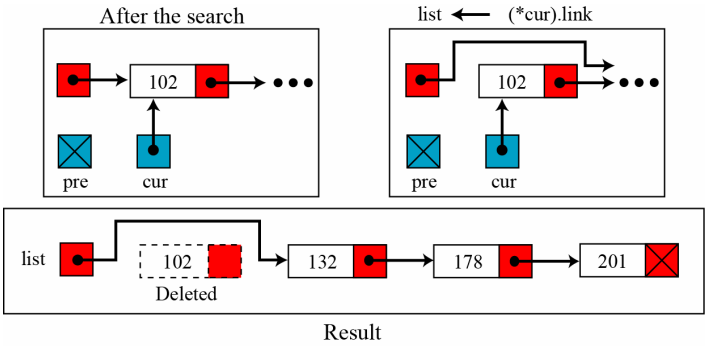
Operations on Linked Lists

Deleting a node

- Before deleting a node in a linked list, we apply the search algorithm.
  - If the flag returned from the search algorithm is true (the node is found), we can delete the node from the linked list.
- We also have two cases:
  - deleting the first node
  - deleting any other node.

Operations on Linked Lists

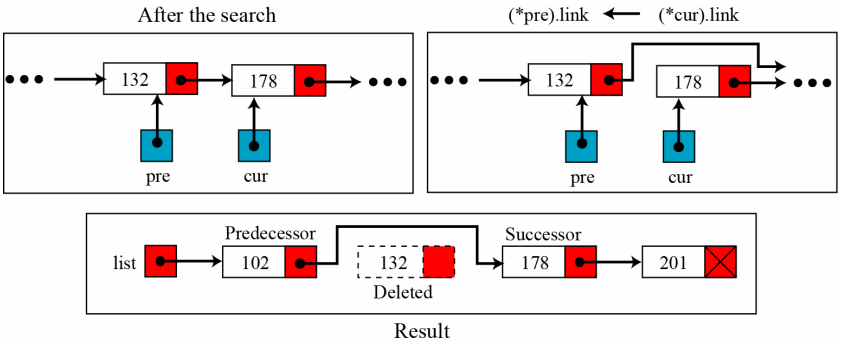
Deleting the first node



If you allocate the memory dynamically, remember to release the memory.
`free()` for `malloc()` and `delete` for `new`.

Operations on Linked Lists

Deleting other nodes



If you allocate the memory dynamically, remember to release the memory.
`free()` for `malloc()` and `delete` for `new`.

Operations on Linked Lists

Deleting other nodes

```
Return: The new linked list
{
        // Given target and returning pre, cur, and flag
        searchlinkedlist (list, target, pre, cur, flag)
        if (flag = false) return list      // The node to be deleted not found
        if (pre = null)                    // Deleting the first node
        {
                list← (*cur).link
                return list
        }
        (*pre).link ← (*cur).link          // Deleting other nodes
        return list
}
```


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

Operations on Linked Lists

Retrieving a node

- Retrieving means randomly accessing a node for the purpose of inspecting or copying the data contained in the node.
- Before retrieving, the linked list needs to be searched.

Operations on Linked Lists

Algorithm 11.6 Retrieving a node in a linked list

```
Algorithm: RetrieveLinkedList (list, target)
Purpose: Retrieves the data in a node after searching the list for the right node
Pre: The linked list (head pointer) and the target (data to be retrieved)
Post: None
Return: Return the data retrieved
{
        searchlinkedlist (list, target, pre, cur, flag)
        if (flag = false) return error     // The node not found
        return (*cur).data
}
```


B. Forouzan and F. Mosharraf, Foundations of Computer Science, 2nd ed., 2008.

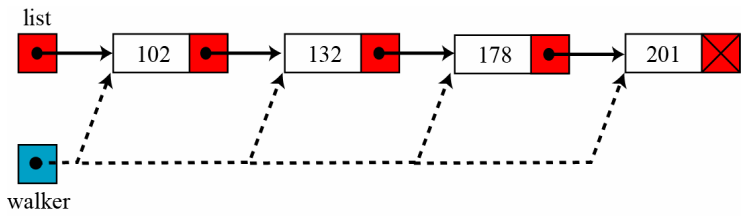
Operations on Linked Lists

Traversing a linked list

- To traverse the list, we need a "walking" pointer,
  - which is a pointer that moves from node to node as each element is processed.
- We start traversing by setting the walking pointer to the first node in the list.
- Then, using a loop, we continue until all of the data has been processed.
- Each iteration of the loop processes the current node, then advances the walking pointer to the next node.

Operations on Linked Lists

Traversing a linked list



Operations on Linked Lists

Traversing a linked list

```
Return: The list
{
        walker ← list
        while (walker ≠ null)
        {
                Process (*walker).data
                walker ← (*walker).link
        }
        return list
}
```

Linked Lists

- Applications
  - A linked list is a very efficient data structure for sorted list that will go through many insertions and deletions.
  - A linked list is a dynamic data structure in which the list can start with no nodes and then grow as new nodes are needed.
  - A node can be easily deleted without moving other nodes, as would be the case with an array.

Summary

- Array, record, and linked list
  - What are they?
  - How to do operations?
  - When will you use them?