Linked List (I)

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline


Outline

Definition

- We have learned array and sequential mapping (e.g., polynomial ADT).
  - Successive nodes of the data objects are stored in a fixed distance.

Definition

- We have learned array and sequential mapping (e.g., polynomial ADT).
  - Successive nodes of the data objects are stored in a fixed distance.
- **Issue:** When a sequential mapping is used for ordered lists:
  - no more available storage
  - waste of storage

Definition

- We have learned array and sequential mapping (e.g., polynomial ADT).
  - Successive nodes of the data objects are stored in a fixed distance.
- **Issue:** When a sequential mapping is used for ordered lists:
  - no more available storage
  - waste of storage
  - Excessive data movement is required for deletions and insertions.

Example

Alan	Bill	Carter	David	Elvis	Frank	

- Insert “Charlie” after Carter.

Example

Alan	Bill	Carter	David	Elvis	Frank	

- Insert "Charlie" after Carter.

Alan	Bill	Carter	Charlie	David	Elvis	Frank

Example

Alan	Bill	Carter	David	Elvis	Frank	

- Insert "Charlie" after Carter.

Alan	Bill	Carter	Charlie	David	Elvis	Frank

Three elements are moved.

Example

Alan	Bill	Carter	Charlie	David	Elvis	Frank

- Delete "Carter" after Bill.

Example

Alan	Bill	Carter	Charlie	David	Elvis	Frank

- Delete "Carter" after Bill.

Alan	Bill	Charlie	David	Elvis	Frank	

Example

Alan	Bill	Carter	Charlie	David	Elvis	Frank

- Delete "Carter" after Bill.

Alan	Bill	Charlie	David	Elvis	Frank	

Four elements are moved.

Solution: linked presentation

- A linked list is comprised of nodes; each node has zero or more data fields and one or more link or pointer fields.
  - The nodes may be placed anywhere in memory.
  - The address of the next (or another) node in the list.

Singly Linked List

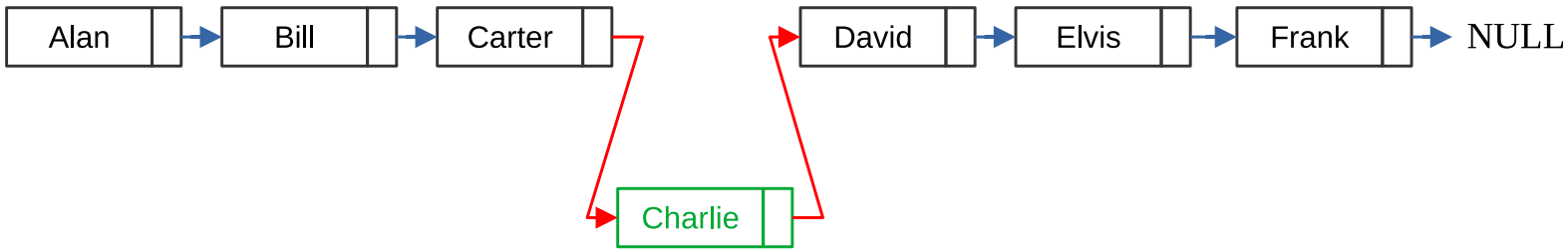
- In a singly linked list, each node has pointer field.
- A singly linked list in which **the last node has a null link** is called a chain.

Singly Linked List

- In a singly linked list, each node has **exactly one** pointer field.
- A singly linked list in which **the last node has a null link** is called a chain.

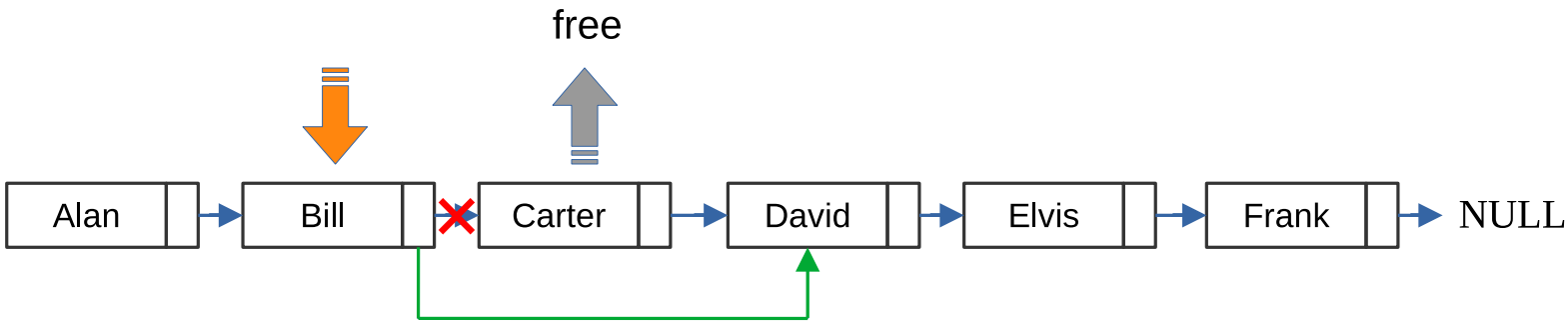

Functions of Linked Lists (1/2)

- Insert ("Charlie") after "Carter".
  1. Get an unused node *a* and set the data field of *a* to "Charlie".
  2. Set the link field of *a* to the node after "Carter", which contains "David".
  3. Set the link field of the node containing "Carter" to *a*.



Functions of Linked Lists (2/2)

- Delete the node containing "Carter".
  1. Find the node *a* that immediately precedes the node containing "Carter".
  2. Set the link of *a* to point to "Carter"'s link.
     - ⋆ We don't need to move any data.
     - ⋆ If possible, free the memory space of node containing "Carter".



Outline

Pointers

- C provides extensive supports for pointers.

> &: address operator
>
> *: dereferencing (indirect) operator

```
int i, *pi; // i:integer variable; pi: a pointer to an integer.
pi = &i;    // pi gets the address of i.
i = 10; // assign the value 10 to i
*pi = 20; // assign the value 20 to i
if (pi == NULL) ... // or if  (!pi); test if the pointer is null.
```


Dynamically Allocated Storage

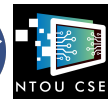
- C provides a mechanism, called heap, for allocating storage at run-time.
  - `malloc` or `calloc`: dynamic memory allocation.
  - `free`: free the memory previously (dynamically) allocated.

```
int i, *pi;
float f, *pf;
pi = (int *) malloc(sizeof(int));
pf = (float *)malloc(sizeof(float));
*pi = 1024; *pf = 3.14;
free(pi);
free(pf);
```


Dynamically Allocated Storage

- How about C++?
  - `new` or `calloc`: dynamic memory allocation.
  - `delete`: free the memory previously (dynamically) allocated.

```
int i, *pi;
float f, *pf;
pi = new int;
pf = new float;
*pi = 1024; *pf = 3.14;
delete pi;
delete pf;
```

Using struct and typedef

```c
struct employee {
      char name[4];
      struct employee *link;
};
typedef struct employee human; //usage: human h1, h2;
typedef struct employee *hPointer; // usage: hPointer link;
```

Self-Referential Structure

```
struct Node {
    int data;
    Node *link;
};
```


Self-Referential Structure

- C allows us to create a pointer to a type that does not yet exist.

```
typedef struct listNode *listPointer; // listNode is still unknown!

struct listNode {
    char data[4];
    listPointer link;
};
```


More functions for linked lists

- To create a new empty list:
  - `listPointer first = NULL;`
- To test for an empty set:
  - `#define IS_EMPTY (first) (!(first))`
- To obtain a new node:
  - `first = (listPointer) malloc(sizeof(*first));`
-

Outline

TBD


Discussions