(14-2) Dynamic Data Structures II
H&K Chapter 13

Instructor – Beiyu Lin

CptS 121 (June 11th, 2019)

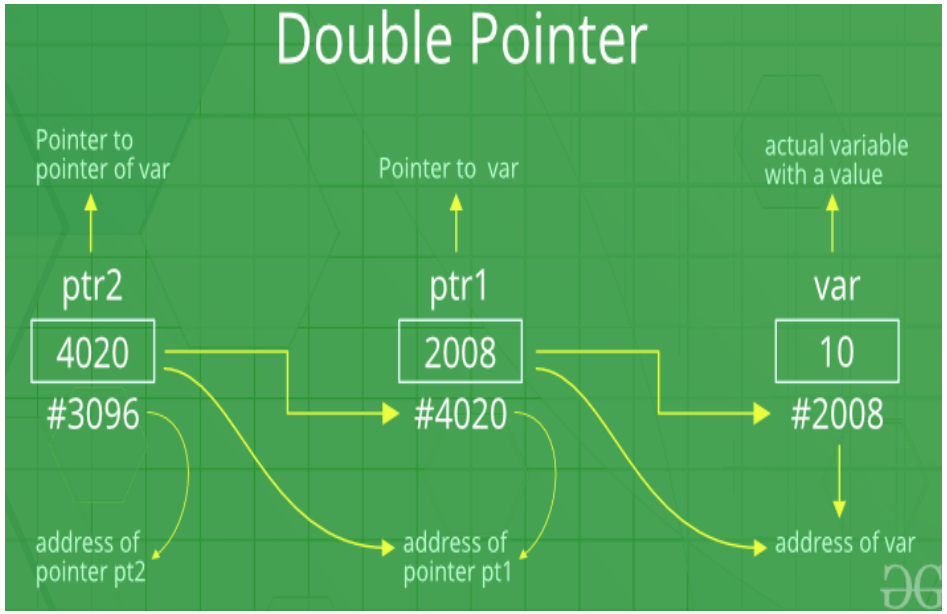
Washington State University

2D Pointers and Dynamic Allocation

```
int var = 10;
int *ptr1;
ptr1 = &var;
```

} **int *ptr1 = &var;**

```
int **ptr2;
ptr2 = &ptr1;
```




```
printf("var is %d \n", var);
printf("the value of ptr1 pointed to is %d \n" , *ptr1);
printf("the value of ptr2 pointed to is %d\n", **ptr2);
```

***ptr1 = 100;**

C. Hundhausen, A. O'Fallon, B. Lin

Graphs are from: https://www.geeksforgeeks.org/double-pointer-pointer-pointer-c/

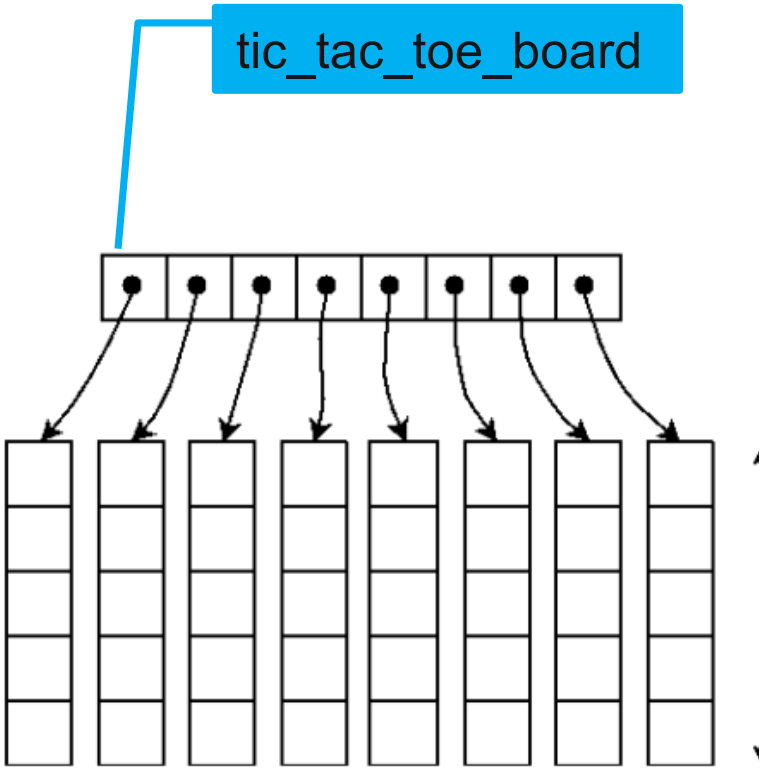
2D Pointers and Dynamic Allocation

```c
/* tic tac toe*/
num = 3;
char** tic_tac_toe_board = NULL;  // declare a two dimen
tic_tac_toe_board = malloc(num * sizeof(char*)); // allo
int t = 0, k = 0;

for (t = 0;t < num; t++)
{
    tic_tac_toe_board[t] = malloc(num * sizeof(char)); /
}

// think as regular two dimensional array.
for (t = 0; t < num; t++)
{
    for (k = 0; k < num; k++)
    {
        //tic_tac_toe_board[t][k] = 'X';
        *(*(tic_tac_toe_board+t)+k) = 'X';
    }
}

for (t = 0;t < num; t++)
{
    for (k = 0; k < num; k++)
    {
        printf("%c", tic_tac_toe_board[t][k]);
    }
    printf("\n");
}
```

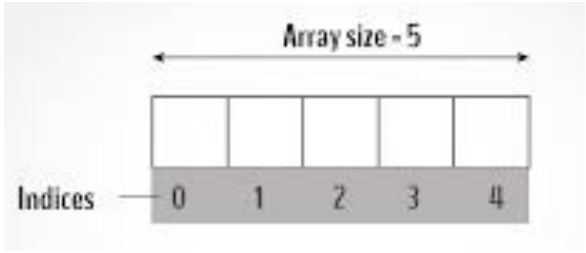


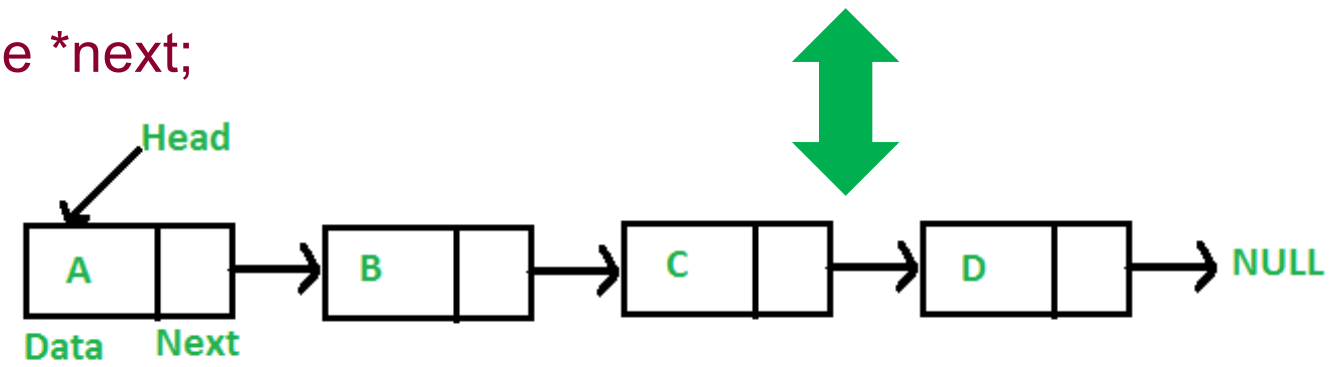
Graphs are from: https://www.techiedelight.com/dynamically-allocate-memory-for-2d-array/

Introduction of Linked List

- Let's define each item as part of a "node"
- A "node" is defined as follows:

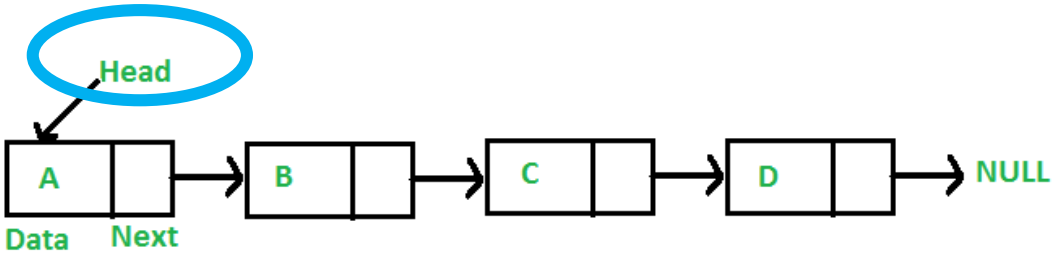
```
typedef struct node
{
    int * data;
    struct node *next;
} Node;
```





Graphs are from:
https://www.programiz.com/c-programming/c-arrays
https://www.geeksforgeeks.org/data-structures/linked-list/

Introduction of Linked List



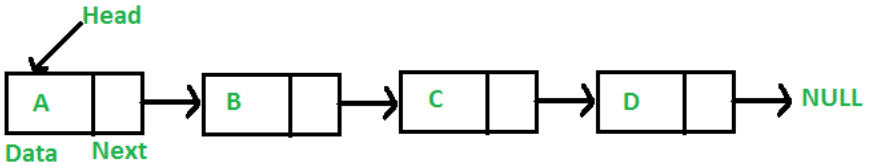
A linked list is represented by a pointer to the first node of the linked list. The first node is called head. If the linked list is empty, then value of head is NULL.

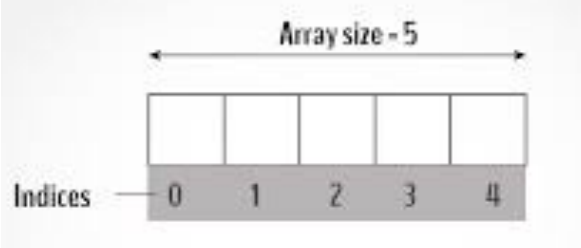
Each node in a list consists of at least two parts:
1) data
2) pointer to the next node

Graphs are from:
https://www.geeksforgeeks.org/linked-list-set-1-introduction/
https://www.geeksforgeeks.org/data-structures/linked-list/

Introduction of Linked List

Linked List v.s. Array





Linked List	Array
Not contiguously located	contiguous locations of elements
Dynamic size	can also be dynamic size
Ease of insertion/deletion	difficulty to insert/delete
Random access is not allowed	Random access allowed
Extra memory space for a pointer	

- Like arrays, Linked List is a linear data structure.
- Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers.

Graphs are from:
https://www.programiz.com/c-programming/c-arrays
https://www.geeksforgeeks.org/data-structures/linked-list/

Applying Dynamic Memory to an Example – Grocery Store List

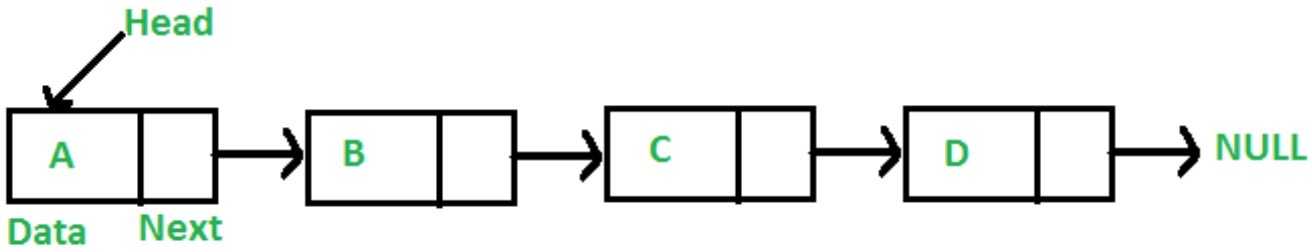
- Let's say we want to build a program that keeps track of our list of grocery store items
- The program must allow the user to add and remove items from the list while shopping
- Items may only be added and removed from the front of the list

Introduction of Linked List

- Let's define each item as part of a "node"
- A "node" is defined as follows:

```
typedef struct node
{
    char * grocery_item;
    struct node *next_ptr;
} Node;
```

char pointer to declare/define a string



Graphs are from:
https://www.programiz.com/c-programming/c-arrays

Grocery Store List Implementation (1)

- How do we allocate memory for a node?

```
Node * make_node (char * item)
{
        Node *mem_ptr = NULL;

        // No error checking for malloc ( ) is provided
        mem_ptr = (Node *) malloc (sizeof (Node));

        mem_ptr -> grocery_item = (char *) malloc (sizeof (char) * (strlen (item) + 1));
        strcpy (mem_ptr -> grocery_item, item);

        mem_ptr -> next_ptr = NULL;

        return mem_ptr;
}
```

Reflection on make_node ( ) (1)

- make_node ( ) required the use of malloc ( ) twice
  - Once to allocate memory for a Node, which consists of a pointer to a character (char *) and a pointer to another node (struct node *)
  - Another to allocate memory to store a copy of the grocery item string passed in as a parameter
    - In this case, since we did not define the grocery_item (in Node) as an array, but instead as a pointer, we needed to allocate enough memory to store a string

Reflection on make_node ( ) (2)

- make_node ( ) returns a pointer to a block of memory that is dynamically allocated; however the pointer is not placed into any “context” like a list yet

Grocery Store List Implementation (2)

- How do we insert a node into the beginning of a list?

```
void insert_at_front (Node **start_ptr, char *item)
{
        Node *mem_ptr = NULL;

        // Assuming enough memory is available
        mem_ptr = make_node (item);

        // Be sure not to lose the rest of the list!
        mem_ptr -> next_ptr = *start_ptr;
        *start_ptr = mem_ptr;
}
```

Reflection on insert_at_front ( )

- insert_at_front ( ) requires a Node ** parameter in order to retain changes made to the list
  - If only a Node * is passed in to the function then changes will not be retained - Why?
- In order to add nodes to a list, only the start of the list is required

Grocery Store List Implementation (3)

- How do we delete a node from the front of the list?
- How do we print a list?
  - Can you implement this function recursively?
- Try to implement these functions on your own…

References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8th Ed.)*, Addison-Wesley, 2016
- P.J. Deitel & H.M. Deitel, *C How to Program (7th Ed.)*, Pearson Education , Inc., 2013.

Collaborators

- Chris Hundhausen
- Andrew O'Fallon