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

Instructor – Beiyu Lin
CptS 121 (June 10th, 2019)
Washington State University

Dynamic Data Structures Revisited

```c
int num = 10;

/* dynamically allocation some memory for a variable */

int * nump = malloc(sizeof(int));
```

```c
int num = 10;

/* dynamically allocation some memory for an array*/
int* arr = calloc(num, sizeof(*arr));
//Do whatever you need to do with arr
int i = 0;
for (i = 0; i < 5; i++)
{
   arr[i] = i;
   printf("arr value is %d\n", arr[i]);
}

free(arr);
```

Dynamic Data Structures Revisited

- Recall dynamic data structures expand and contract at program runtime
- We generally use `malloc()` to allocate one or more blocks of memory and `free()` to de-allocate blocks of memory

C. Hundhausen, A. O'Fallon

Why Should we Allocate Memory Dynamically?

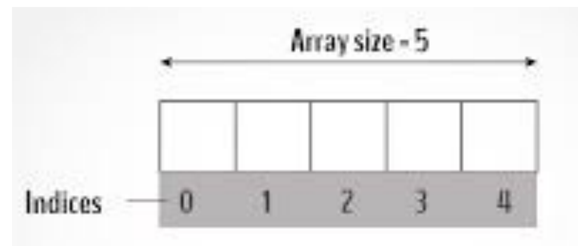
- To allocate exact number of bytes required by our program at any particular point during execution of the program
- To eliminate allocating unnecessarily large amounts of unused memory (i.e. like with an array[MAX_SIZE])

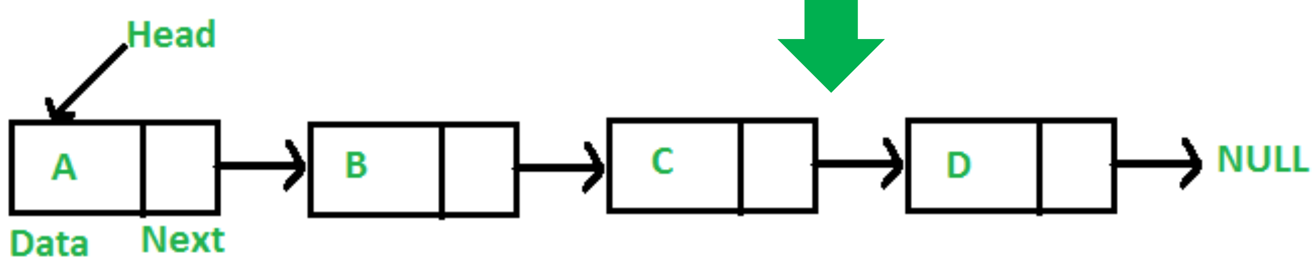
C. Hundhausen, A. O'Fallon

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;
```

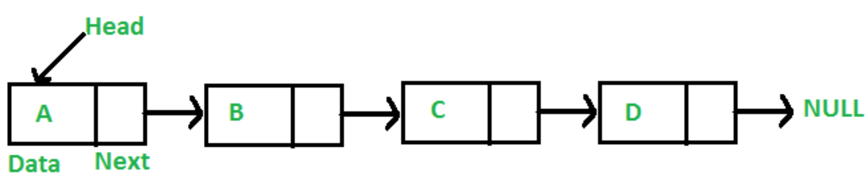


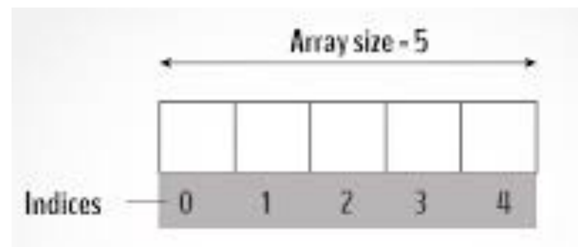


Graphs are from:
https://www.programiz.com/c-programming/c-arrays
https://www.geeksforgeeks.org/data-structures/linked-list/

Grocery Store List Design

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	

Let us create a simple linked list with 3 nodes (live coding).

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

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

Applying Dynamic Memory to an Example – Grocery Store List

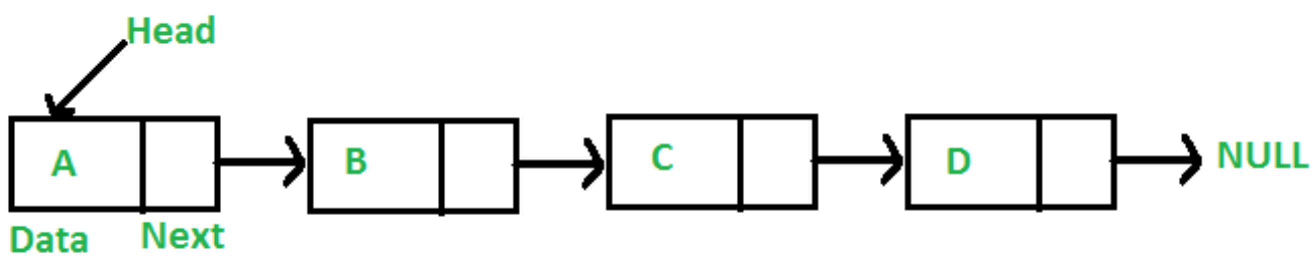
- Let's say we want to build a program that keeps track of our list of grocery store items
- The program must allow the user to add and remove items from the list while shopping
- Items may only be added and removed from the front of the list

C. Hundhausen, A. O'Fallon

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;
```



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

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;
}
```


C. Hundhausen, A. O'Fallon

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

C. Hundhausen, A. O'Fallon

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