CSO-Recitation 06

CSCI-UA 0201-007

R06: Assessment 04 & Strings & Linked list

Today’s Topics

- Assessment 04
- Strings
- Linked list
- Lab2 has bonus

Strings

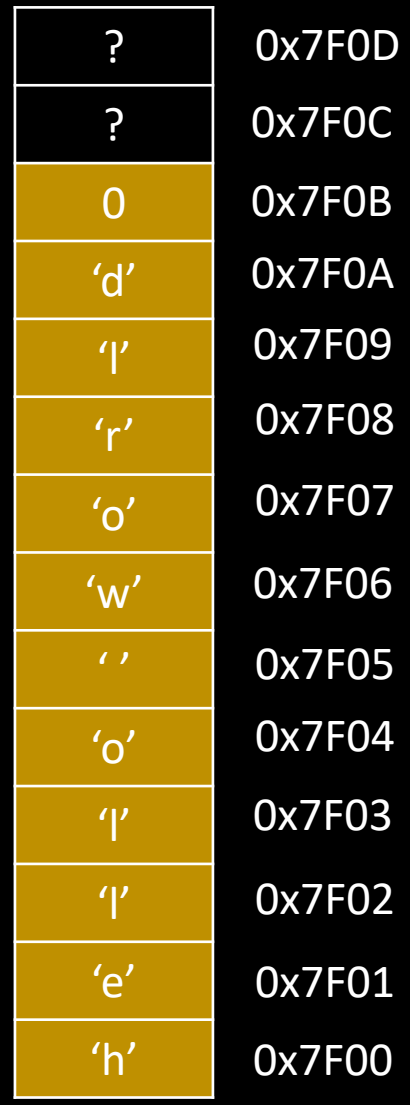
Arrays of chars

What are strings?

- They are arrays of the type *char*, which is typically one byte
- Char literals are in single quotes ‘ ’
- String literals are in double quotes “ “
- Unlike other arrays, strings have a way of knowing the length even at runtime
  - Strings are stored with the last byte set to 0 (or ‘\0’)
    - C strings are called “null terminated”
    - So you can find the length by looping over the string, keeping a counter, and stopping when you find a char equal to zero
  - There is also a standard library function for this, *strlen*

Defining a string

- char *arr = “hello world”;
- char arr[11] = “hello world”;
- The literal “hello world” includes the null-terminator.



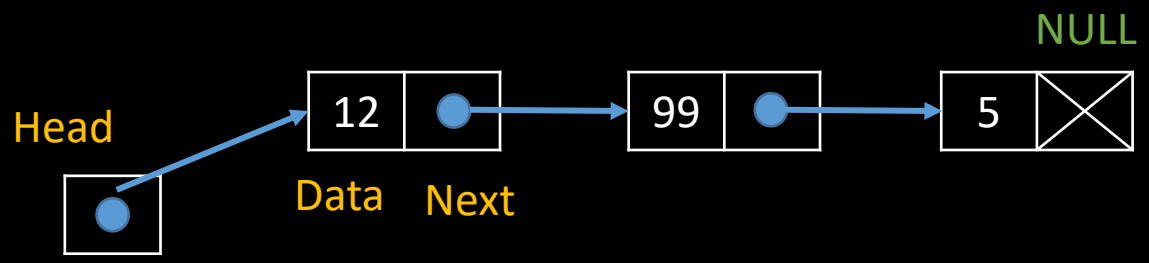
Array of pointers: argv

- argv is an array of strings (pointers to char)
  - the strings are your arguments
  - argv[0] is the name of the executable file
- argv has argc many elements

Linked list

A linear data structure

Why linked list?



- Like arrays, Linked List is a linear data structure.
- Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.
- Arrays have limitations:
  - The size of the arrays are fixed
  - Inserting (Deleting) a new element in an array of elements is expensive
    - because the room has to be created for the new elements and existing elements have to be shifted.

Advantages and Drawbacks

- Advantages over arrays:
  - Dynamic size
  - Ease of insertion/deletion
- Drawbacks:
  - Random access is not allowed
    - We have to access elements sequentially starting from the first node. (Traverse)
  - Extra memory space for a pointer is required with each element of the list.
  - Not cache friendly
    - Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Linked list

- A linked list is represented by a pointer to the first node of the linked list
  - It is called the *head*
  - If the linked list is empty, then the value of the head is NULL
- Each node in a list consists of at least two parts:
  - data
  - Pointer (or Reference) to the next node
- In the case of the last node in the list,
  - the next field contains NULL - it is set as a null pointer.
- In C, we can represent a node using struct
  - nodes are defined as (e.g.) node using *typedef*
  - *node *head*

Initialize the linked list

- The list is initialized by creating a *node *head* which is set to NULL
- The variable *head* is now a pointer to NULL, but as nodes are added to the list, *head* will be set to point to the first node
- In this way, *head* becomes the access point for sequential access to the list.

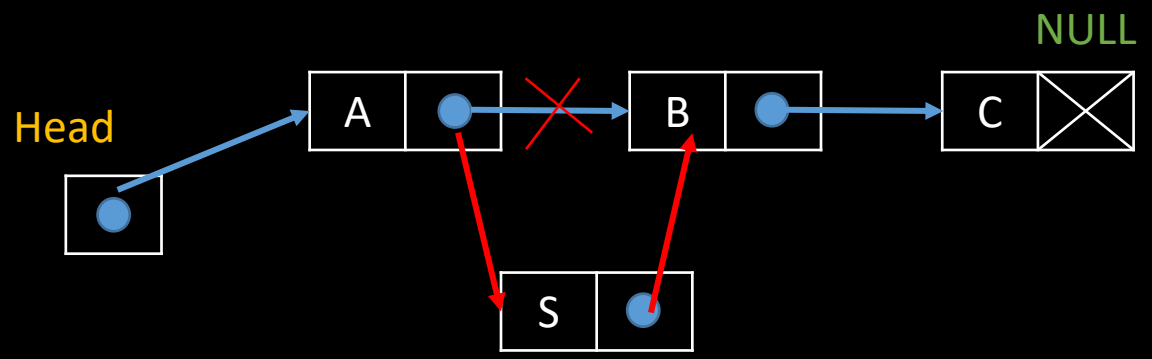
Linked list

- Linked list insertion
- Linked list Deletion
- Search an element in a linked list
  - Iterative and Recursive
- Traverse a linked list
- Find length of a linked list
- …

Linked list

- In class, we pass the header pointer,
  - ask it to return a new head
  - the caller is responsible for updating it itself
- In lab-2, we pass a pointer to pointer parameter (pointer to the head pointer),
  - to allow changing the head pointer directly instead of returning the new one
  - note that there’s no return value; It’s not needed.

Inserting a node



- How can we insert a node in a sorted linked list?
  - Insert a node at the front of the linked list
    - insert_front
  - Insert a node after a given node
    - Think: how can I know my S should be inserted between A and B?
    - If by comparing A and S I know S should be at the position after A, then how can I know S should be after B or between A and B?
  - Insert a node at the end of the linked list

Dynamic memory allocation

- Each time you need to manually allocate data, use *malloc*
  - void *malloc(size_t size);
- If you need to manually de-allocate
  - void free(void *ptr);

More on linked list

- Implement a hash table
  - see clear instructions on our website lab-2 page
- A hash table is an array of linked lists with a hash function
  - A hash function basically just takes things and puts them in different “buckets” (hash table’s array of entries)
  - Each “bucket" just points to a linked list here