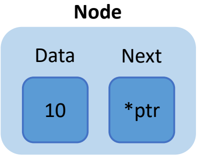
601.220 Intermediate Programming

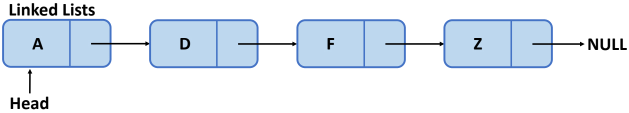
Linked lists

Linked List

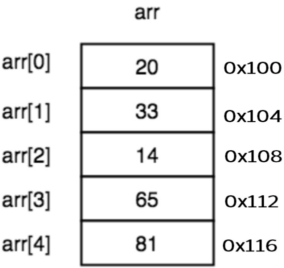
- linear data structure in which the elements, called *nodes*, are not stored at contiguous memory locations (in contrast with arrays)
- each node comprises two items - the data it stores and a pointer to the next node
- last node's next pointer points to NULL
- the entry point is called *head*
- the *head pointer* is not itself a node; it just holds the address of first node
- in an empty linked list, the *head pointer* points to NULL

Linked List

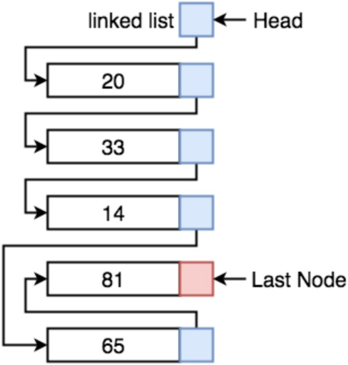




Linked List vs Array



arr		
arr[0]	20	0x100
arr[1]	33	0x104
arr[2]	14	0x108
arr[3]	65	0x112
arr[4]	81	0x116



Linked List vs Array

Array	**Linked List**
size of the array is fixed	sized of linked list is not fixed
occupies less memory for the same number of elements	requires more space because of “next”
accessing i’th value is fast using indices (simple arithmetic)	has to traverse the list from start
inserting new elements is expensive	after deciding where to add, is straightforward (no shifting)
no deleting without shifting items	deleting is easy (kind of)

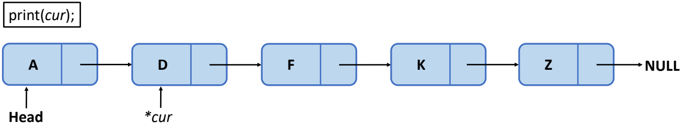
Node struct & create_node

```c
typedef struct node_ {
  char data;                // could be any type
  struct node_ * next;  // self-referential!
} Node;

Node * create_node(char ch) {
  Node * node = (Node *) malloc(sizeof(Node));
  assert(node);
  node->data = ch;
  node->next = NULL;
  return node;
}
```

List print function

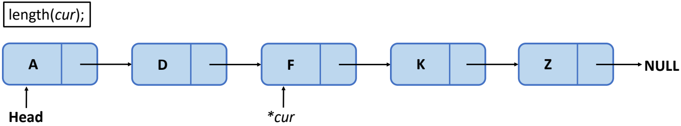
- print - output all data items in order from head to tail
  - `void print(const Node * head)`
  - use a Node pointer named cur to advance node by node through list, and each time cur encounters another node, output that node's data value
  - Analogy: using for loop to print all elements of an array



- What is the output of `print(cur)`?

List `length` function

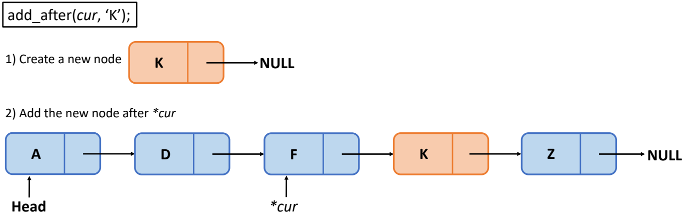
- length - reports number of items currently in list
  - `long length(const Node * head)`
  - use a Node pointer named `cur` to advance node by node through list, and increment a counter each time `cur` encounters another node



- What is the output of `length(cur)`?

List `add_after`

- add_after - insert new node with a given data value immediately after a given existing node
  - `void add_after(Node * node, char val)`
  - val parameter is data value to place in new node
  - node parameter holds address of existing node that new one should be placed right after
  - the new node needs to be dynamically allocated
  - additional statements are needed to adjust links appropriately so list stays connected



add_after(*cur*, ‘K’);

1) Create a new node

2) Add the new node after **cur*

quiz!

Consider the following program. What output is printed?

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node_ {
  char data;
  struct node_ *next;
} Node;

int main(void) {
  Node *a = malloc(sizeof(Node)),
       *b = malloc(sizeof(Node)),
       *n;
  a->data = 'A';
  b->data = 'B';
  a->next = b;
  b->next = a;
  for (n = a; n != NULL; n = n->next) {
    printf("%c ", n->data);
  }
  printf("\n");
  return 0;
}
```

What output is printed?

A. No output is printed

B. `A`

C. `A B`

D. `B A`

E. None of the above