ECE 220 Computer Systems & Programming

Lecture 20 – Problem Solving with Linked List

ECE ILLINOIS

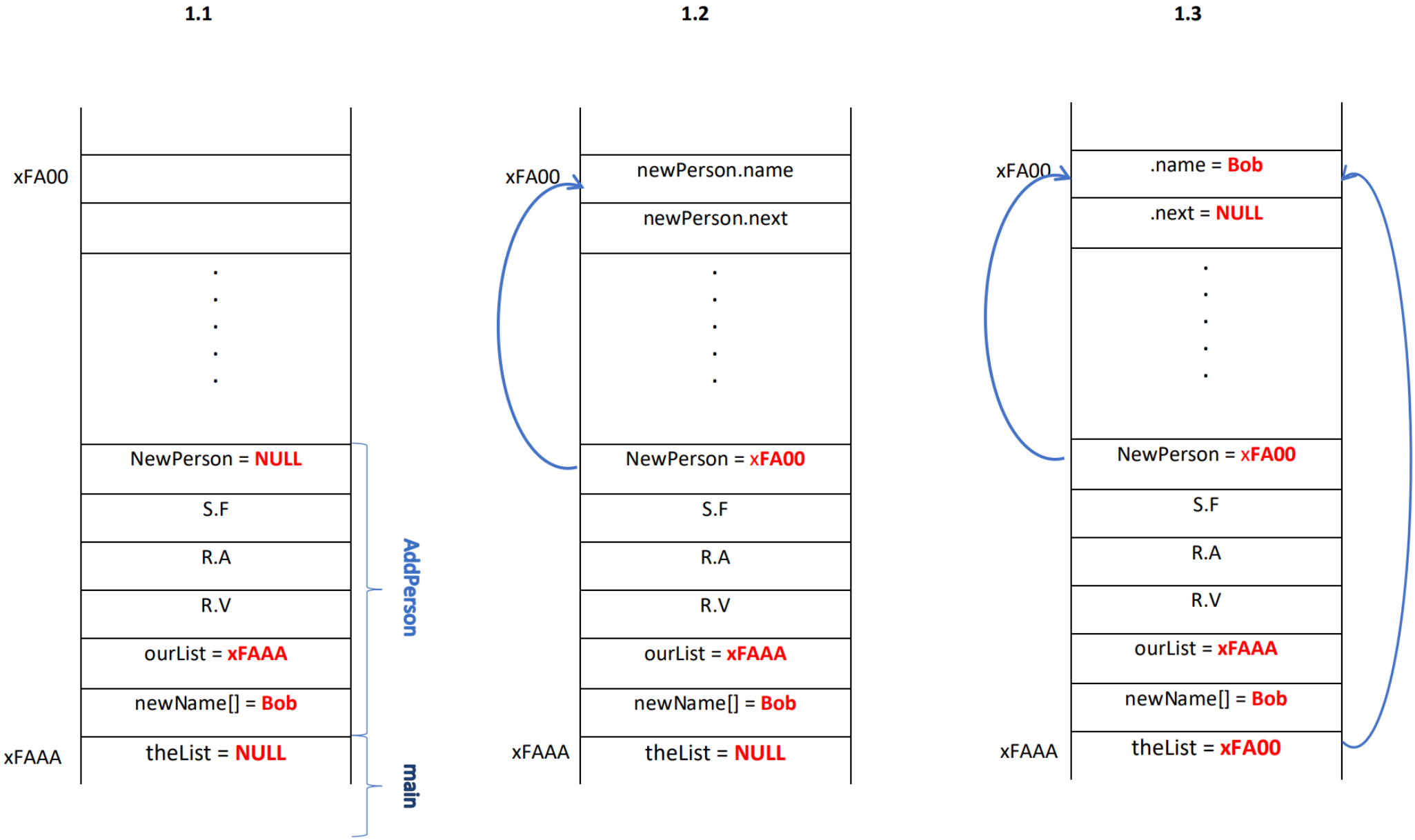
ILLINOIS

Review: Linklist and its runtime stack

```
typedef struct person_node Person;
struct person_node
{
   char name[20];
   Person *next;
};
```

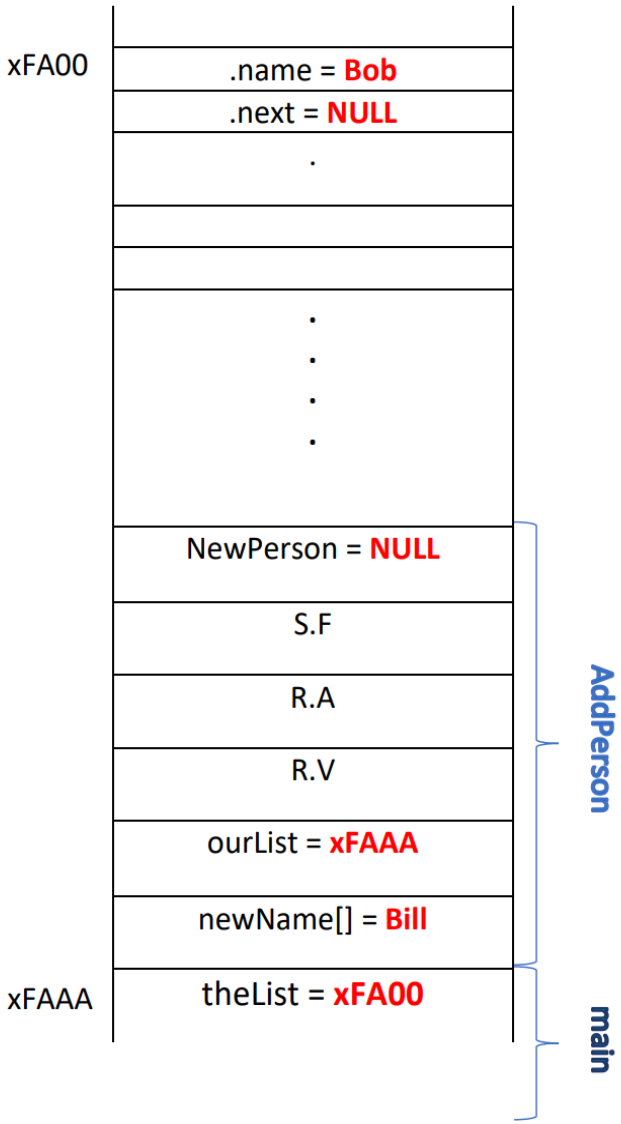
```
int main()
{
   Person *theList = NULL;
1  AddPerson(&theList, “Bob");
2  AddPerson(&theList, "Bill");
}
```

```
/* add to the linked list */
int AddPerson(Person **ourList, char name[])
{
   Person *newPerson = NULL;                    → 1

   newPerson = (Person *)malloc(sizeof(Person));
   if (newPerson == NULL)
      return 0;                                 → 2

   strcpy(newPerson->name, name);
   newPerson->next = *ourList;
                                                → 3
   *ourList = newPerson;

   return 1;
}
```

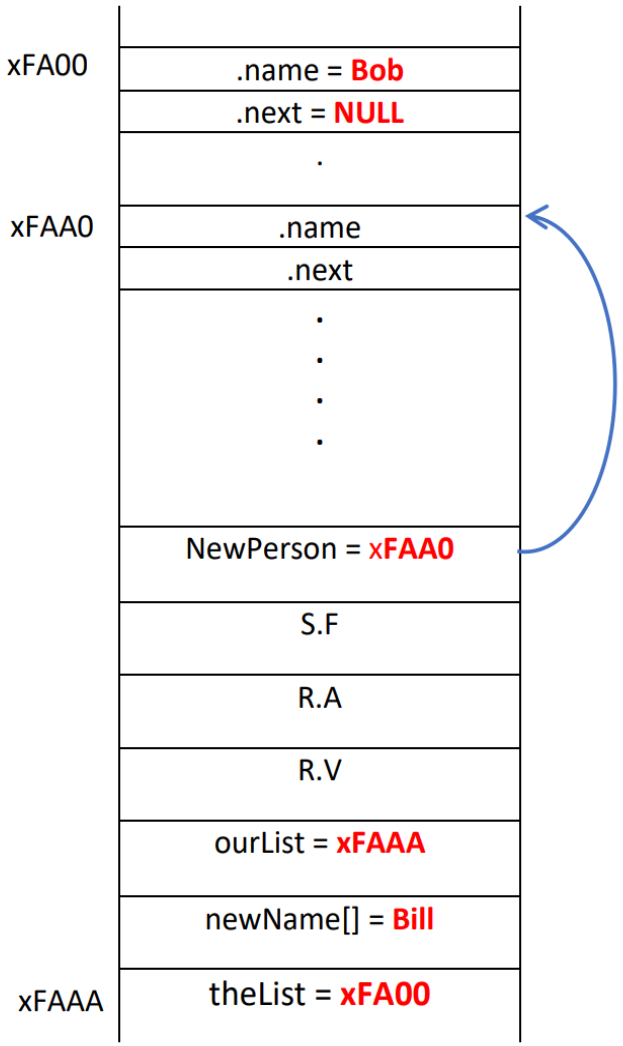


Runtime stack - **AddPerson(&theList, "Bob");**

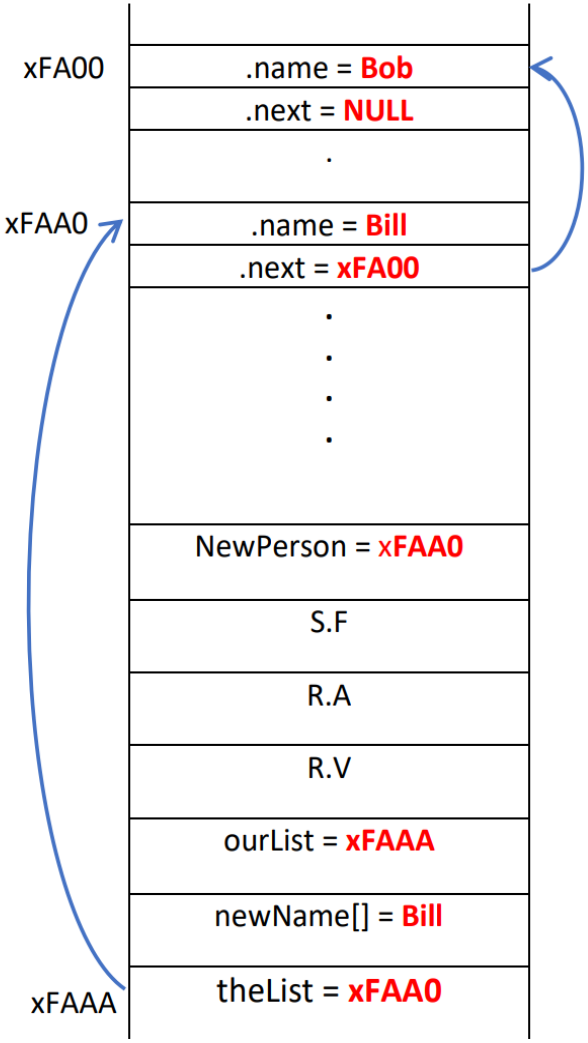
2.1



2.2



2.3

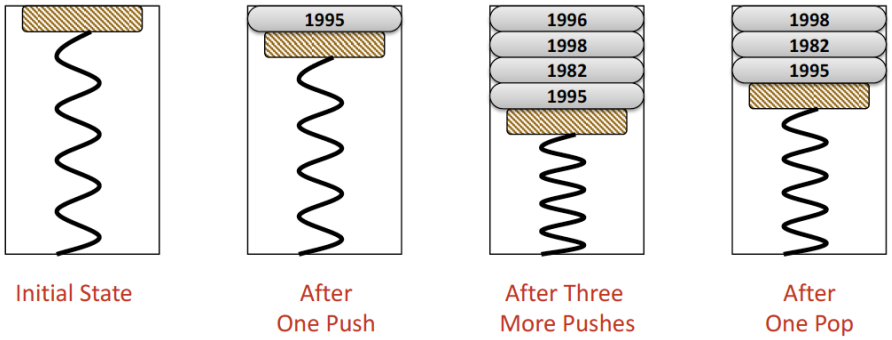


Runtime stack - **AddPerson(&theList, "Bill");**

Stack data types

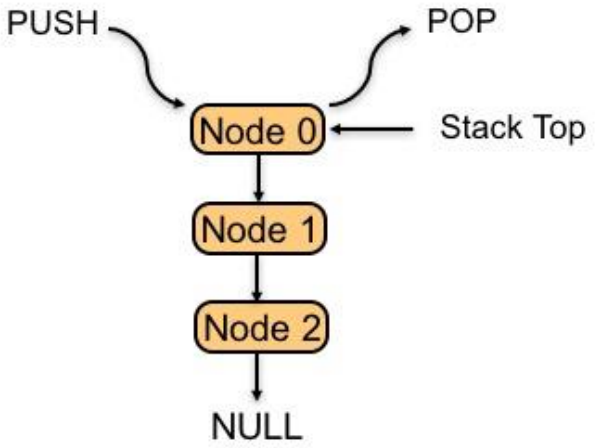
Stack

- First item in is the last item out - _____
- Two operations for data movement: _____ & _____



Stack can be implemented as a linked list in which adding and removing elements occurs at the top of the list (LIFO)

Functions to add and remove elements from a stack: push and pop



Stack (code)

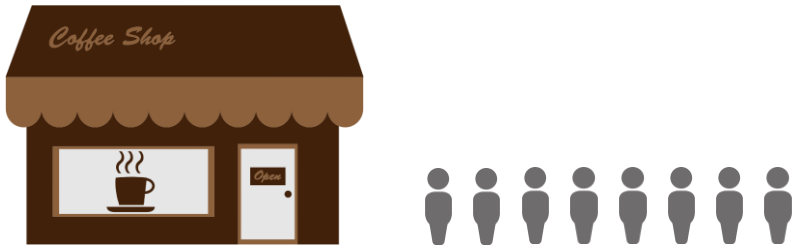
```
int main(int argc, char *argv[])
{
    item *inventory = NULL;

    PrintList(inventory);

    /* stack */
    Push(&inventory, "part3", 10, 1.0f);
    Push(&inventory, "part1", 5, 1.0f);
    Push(&inventory, "part2", 5, 1.0f);
    PrintList(inventory);
    Pop(&inventory);
    PrintLis

    FreeList

    return 0
}
```

```
Printing linked list:
Item #0 @ 0x19190d0: part2 5 1.000000 0x1919070
Item #1 @ 0x1919070: part1 5 1.000000 0x1919010
Item #2 @ 0x1919010: part3 10 1.000000 (nil)

Printing linked list:
Item #0 @ 0x1919070: part1 5 1.000000 0x1919010
Item #1 @ 0x1919010: part3 10 1.000000 (nil)
```

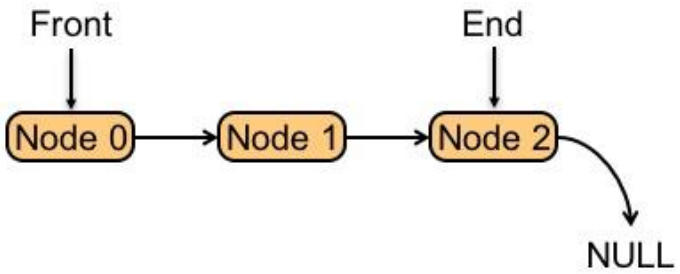
Queue data types

Queue

- First item in is the first item out - _____
- Two operations for data movement: _____ & _____



Queue is a linked list in which adding a new element occurs at the end of the list and removing an element occurs at the start of the list.
Functions to add and remove elements: enqueue and dequeue



Queue (code)

```
int main(int argc, char *argv[])
{
    item *inventory = NULL;

    PrintList(inventory);

    /* queue */
    enqueue(&inventory, "part3", 10, 1.0f);
    enqueue(&inventory, "part2", 5, 1.0f);
    enqueue(&inventory, "part1", 5, 1.0f);
    PrintList(inventory);
    dequeue(&inventory);
    PrintList(inventory);

    FreeList(&inv

    return 0;
}
```

```
Printing linked list:
Item #0 @ 0x1d9b010: part3 10 1.000000 0x1d9b070
Item #1 @ 0x1d9b070: part2 5 1.000000 0x1d9b0d0
Item #2 @ 0x1d9b0d0: part1 5 1.000000 (nil)

Printing linked list:
Item #0 @ 0x1d9b070: part2 5 1.000000 0x1d9b0d0
Item #1 @ 0x1d9b0d0: part1 5 1.000000 (nil)
```

Queue implementation using a linked list

```
int enqueue(item **head, char *name, int q, float cost)
{
    item *new_item;

    if (           ); //check if the *head is NULL
    {
        //allocate heap for the new data item
        // if fails return 0;


        //fill-up the member element and
        //update the new_item->next and the current head


        return 1;
    }

    return ???? //(recursively calling the enqueue!!);
    //remember head should remain same only head->next
    //is updated with the new element
```

Linklist

Sorting Algorithm – Bubble Sort

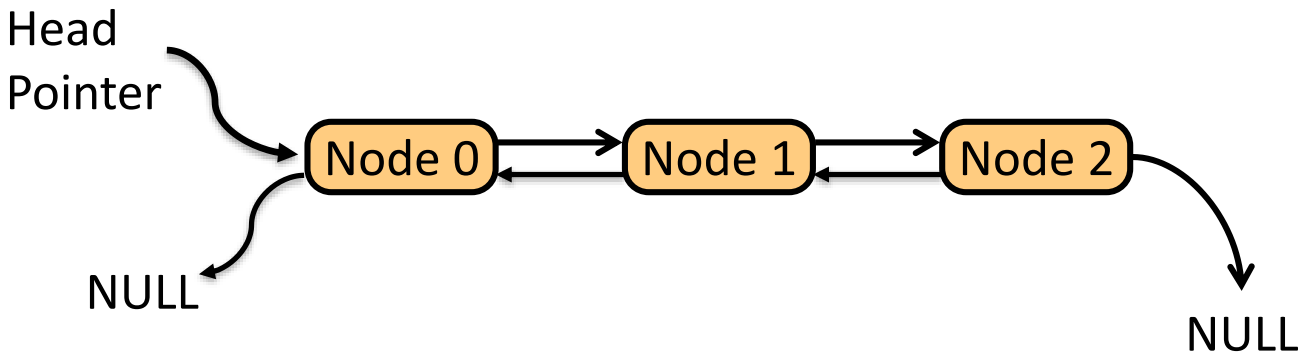
```c
int main(int argc, char *argv[])
{
    item *inventory = NULL;

    /* Insert/Sort example */
    InsertItem(&inventory, "paerC", 1, 1.0f);
    InsertItem(&inventory, "paerA", 1, 1.0f);
    InsertItem(&inventory, "paerD", 1, 1.0f);
    InsertItem(&inventory, "paerQ", 1, 1.0f);
    InsertItem(&inventory, "paerB", 1, 1.0f);
    PrintList(inventory);

    BubbleSort(&inventory);

    PrintList(inventory);

    /* using list for computation */

    FreeList(&inventory);

    return 0;
}
```

Result from Bubble Sort Algorithm:

```
Printing linked list:
Item #0 @ 0x1d7d070: paerA 1 1.000000 0x1d7d190
Item #1 @ 0x1d7d190: paerB 1 1.000000 0x1d7d010
Item #2 @ 0x1d7d010: paerC 1 1.000000 0x1d7d0d0
Item #3 @ 0x1d7d0d0: paerD 1 1.000000 0x1d7d130
Item #4 @ 0x1d7d130: paerQ 1 1.000000 (nil)

Printing linked list:
Item #0 @ 0x1d7d130: paerQ 1 1.000000 0x1d7d0d0
Item #1 @ 0x1d7d0d0: paerD 1 1.000000 0x1d7d010
Item #2 @ 0x1d7d010: paerC 1 1.000000 0x1d7d190
Item #3 @ 0x1d7d190: paerB 1 1.000000 0x1d7d070
Item #4 @ 0x1d7d070: paerA 1 1.000000 (nil)
```

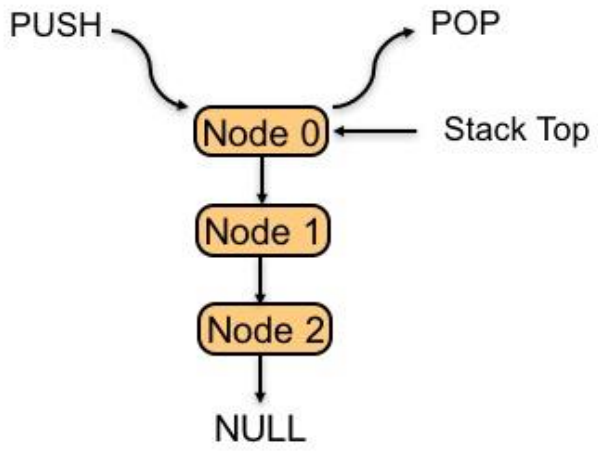
Doubly linked list

```
typedef struct studentStruct Record;
struct studentStruct
{
    char Name[100];
    int UIN;
    float GPA;
    Record *prev;
    Record *next;
};
```

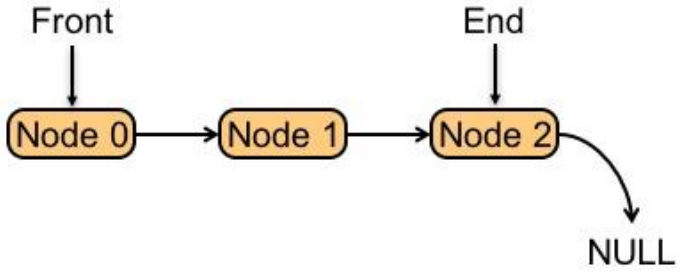


Implement abstract data types using linked list

- **Stack**



- **Queue**



- **Deque ("Deck", double-ended queue)**