Linked lists: Barebones Version

CS 115

Dr. Joseph Eremondi, adapted from Dr. Shakil Khan, Dr. Philip Fong, and Dr. Howard Hamilton

Last updated: March 21, 2025

Linked Lists

Motivation

- Dynamic data structure

Motivation

- Dynamic data structure
    - Alternative to arrays

Motivation

- Dynamic data structure
    - Alternative to arrays
    - Don’t want fixed length

Motivation

- Dynamic data structure
    - Alternative to arrays
    - Don’t want fixed length
        - Allocate one node at a time

Motivation

- Dynamic data structure
    - Alternative to arrays
    - Don’t want fixed length
        - Allocate one node at a time
    - Able to quickly insert

Motivation

- Dynamic data structure
  - Alternative to arrays
  - Don't want fixed length
    - Allocate one node at a time
  - Able to quickly insert
- Idea: dynamically allocate single node when requested

Motivation

- Dynamic data structure
  - Alternative to arrays
  - Don't want fixed length
    - Allocate one node at a time
  - Able to quickly insert
- Idea: dynamically allocate single node when requested
  - problem: then we will need an arbitrary number of static (i.e. allocated at compile-time) pointers

Potential Solution

- use 1 static pointer to dynamically allocate the first node

Potential Solution

- use 1 static pointer to dynamically allocate the first node
  - don't just allocate for the data

Potential Solution

- use 1 static pointer to dynamically allocate the first node
    - don't just allocate for the data
    - allocate enough space data and pointer to potential next node

Potential Solution

- use 1 static pointer to dynamically allocate the first node
    - don't just allocate for the data
    - allocate enough space data and pointer to potential next node
        - i.e. change the definition of a node

Potential Solution

- use 1 static pointer to dynamically allocate the first node
  - don't just allocate for the data
  - allocate enough space data and pointer to potential next node
    - i.e. change the definition of a node
    - if the next node does not exist, indicate with null

Potential Solution

- use 1 static pointer to dynamically allocate the first node
  - don't just allocate for the data
  - allocate enough space data and pointer to potential next node
    - i.e. change the definition of a node
    - if the next node does not exist, indicate with null
    - Linked data structure that can dynamically grow and shrink as needed

Node definition

Node definition

```
struct Node{
  int data;       // the actual data
  Node *next; // pointer to potential next node
};

// We only need 1 pointer to handle this linked list
// It points to the first node of the list

Node *head; // let's make it global for now
//(can be generalized if needed)
// e.g. as an object
// Initially, there are no nodes, i.e. the list is empty

head = nullptr;
```

Adding to end of the list

Adding to end of the list

```
void insert(int data){

  Node *temp = head;

  if(temp == nullptr){
    temp = new Node;
    temp->data = data;
    temp->next = nullptr;
    head = temp;
  }

  else{
    while(temp->next != nullptr)
      temp = temp->next;
    temp->next = new Node;
    temp = temp->next;
    temp->data = data;
    temp->next = nullptr;
  }
} // end insert
```

Adding in sorted order

Adding in sorted order

```
void insertS(int data){
  Node *curr = head;
  Node *prev = nullptr;

  while(curr!=nullptr && curr->data<data){
    prev = curr;
    curr = curr->next;
  }
  if(prev == nullptr){
    prev = new Node;
    prev->data = data;
    prev->next = curr;
    head = prev;
  }
  else{
    prev->next = new Node;
    prev = prev->next;
    prev->data = data;
    prev->next = curr;
  }
} // end insertS
```

Traversing the list

Traversing the list

```
bool isEmpty(){
  return head == nullptr;
}

void print(){
  Node *temp = head;
  while(temp != nullptr){
    cout << temp->data <<" ";
    temp = temp->next;
  }
}

int count() {
  Node *temp = head;
  int ctr=0;
  while(temp != nullptr){
    ctr++;
    temp = temp->next;
  }
  return ctr;
}
```

Deleting the entire list

Deleting the entire list

```
void deleteLinkedList(){

  Node *current = head, *previous = nullptr;

  while(current != nullptr){
    previous = current;
    current = current->next;
    delete previous;
  }

  head = nullptr;

}
```

Removing a node

Removing a node

```
void remove(int x){
  Node *prev = nullptr;
  Node *curr = head;

  if(isEmpty())
    return;
  while(curr != nullptr){
    if(curr->data == x)
      break;
    else{
      prev = curr;
      curr = curr->next;
    }
  } // end while
    // found: 1st node needs removing
  if(prev == nullptr){
    Node *temp = head;
    head = head->next;
    delete temp;
  }
```

Testing it all out

Testing it all out

```
int main() {

  insert(5);
  insert(6);
  insert(7);

  // try out every possibility
  remove(8); // 5, 7, empty list

  insert(3);

  print();

  cout << "\nCount=" << count(); cout << "\n";

  deleteLinkedList();

  cout << "\nCount=" << count(); cout << "\n";

  return 0;
}
```