Make list : visited places

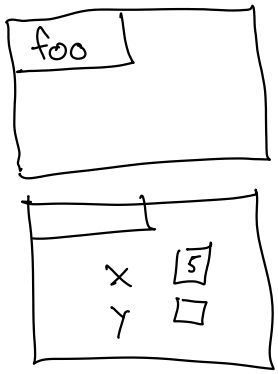
Pick direction ; add destination to visited

Never pick place we've been

[Only visited : pick at random] — goal : getting back to choice we haven't taken

Haven't Been	Have Been
front of the garage	front walk
sitting room	porch
dining roona	flower garden
	foyer

```
int x = 5;
int y = foo(x);
int z = 2;
```



ADT — Stack

push — add to top
pop — remove from top
peek — look at top
isEmpty
getSize

Lists are more powerful than Stacks and that's not always good

LinkedStack:

implementation of Stack
uses LinkedList

O(1) push: insertAtHead
O(1) pop: removeFromHead

ArrayStack:

uses ArrayList

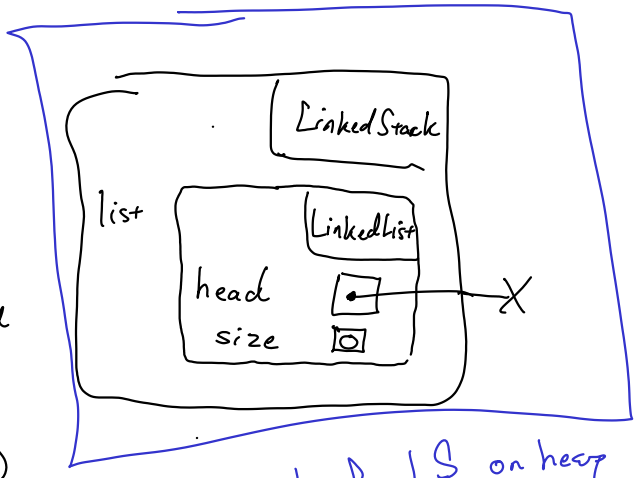
O(n) push: insertAtHead
O(n) pop: removeFromHead

amortized O(1) push: insertAtTail
O(1) pop: removeFromTail

```
template <typename T>
class LinkedStack : public Stack<T> {
    .
    .
    ;
private:
    LinkedList<T> listj
}
```

alt

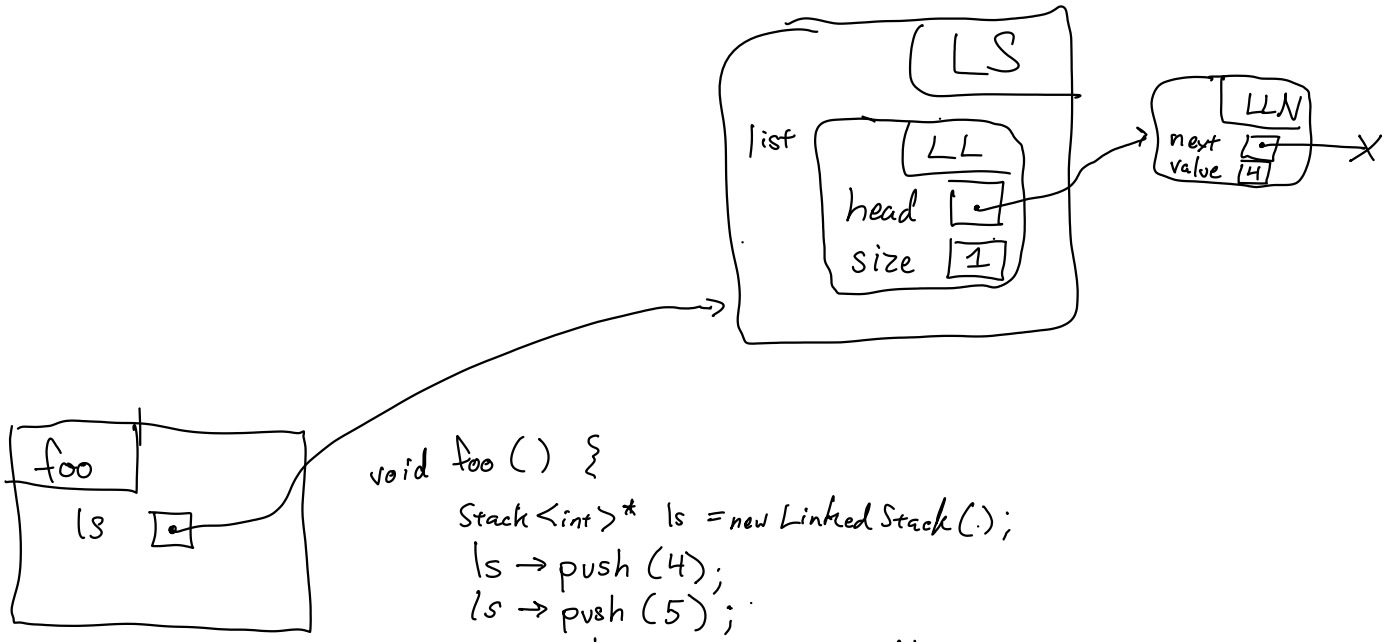
```
private:
    LinkedList<T>* listptrj
```



Example of LS on heap

Object Lifecycle

1. Creation
   a. Memory is allocated
   b. Initialized — inf: nothing
   c. Call constructor — object: created
2. Use
3. Destruction
   a. Call destructor
   b. Uninitialize (destruct object fields)
   c. Deallocate




```
void foo () {
    Stack<int>* ls = new LinkedStack();
    ls → push (4);
    ls → push (5);
⇒   cout << ls → pop() << endl;
```

Depth First Search (maze)

```
visited ← new VisitedGrid
frontier ← new Stack

while frontier is not empty:
    next ← frontier.pop();
    for each neighbor:
        if neighbor not visited:
            mark neighbor as visited
            frontier.push(neighbor)
```