Restaurant Wait Queue

CS 211 – Project 4

Restaurant Wait Queue

- No reservations taken
- Show up and give host/hostess name and party size
- When a table becomes available the group that has been waiting the longest (and that can all fit at the table) is seated

Restaurant Wait Queue

- One wrinkle, a party can “Call Ahead” and be added to the Wait List
- Once a Call Ahead party arrives, they must check in to tell the host/hostess that they have arrived

So NOW:

- When a table becomes available:
  - the group that has been waiting the longest AND
  - that can all fit at the table AND
  - is IN the restaurant)
- is seated (removed from the Wait Queue)

Restaurant Wait Queue

- Implement as a Linked List using the C Language
- Node should contain
  - Name (C style string)
  - Group Size (integer)
  - In-Restaurant Status (create Enumerated type – like Boolean)
  - Next Pointer

- Add new node to end of list when a group arrives or calls in
- Seat/Remove the node closest to the front that meets conditions

Restaurant Wait Queue Commands

- User Interface should be in base code file (except debug mode stuff)
- Add command: a <size> <name>
  - Add to list – group is in restaurant
  - Validate that name is unique (not already in wait queue)
- Call Ahead command: c <size> <name>
  - Add to list – group is NOT in restaurant
  - Validate that name is unique (not already in wait queue)
- Waiting Command: w <name>
  - Call Ahead List finally arrives in restaurant
  - Change value in groups node to indicate group is IN restaurant

Restaurant Wait Queue Commands

- Retrieve command: r <size>
  - Remove node from list that can fit at given size and is IN restaurant
  - Display name of group retrieved or message saying no one can fit
- List command: l <name>
  - List information for all groups ahead of the indicated group
- Display Command: d
  - Show information for all groups in the list
- Quit Command: q
- Help Command: ?
  - Both should already be programmed for you in base code

Development Requirements

- Must have (at least) 3 source code “.c” files
  - PLUS: header “.h” file and makefile
- Function decomposition is Specified
  - Which functions go into which source code files
- Names of Linked Lists Functions are Specified
  - You can create additional functions using whatever name you desire
- Often Program Specs that you MUST follow are given to you

Development Requirements: Source Code Files

The following functions from proj4base.c are to be in one source code file (these are the user interface functions):

- main()
- clearToEoln()
- getNextNWSChar()
- getPosInt()
- getName()
- printCommands()

Development Requirements: Source Code Files

The following functions from proj4base.c are to be in another source code file (these are the application functions that interact with the linked list functions): Note Parameters can be changed as needed!

- doAdd()
- doCallAhead()
- doWaiting()
- doRetrieve()
- doList()
- doDisplay()

Development Requirements: Source Code Files

The third source code file is to have the code that you are writing that will perform the linked list implementation.

The functions in this source code file will include the following functions **plus** any other you write to handle the linked list:

Given names **MUST** be used, you determine parameters

- addToList()
- doesNameExist()
- updateStatus()
- retrieveAndRemove()
- countGroupsAhead()
- displayGroupSizeAhead()
- displayListInformation()

Development Requirements

- Must have (at least) 3 source code “.c” files
- DON’T FORGET: header “.h” file and makefile
- You MUST follow header file and makefile design as done in LAB 7!
- Zip all files together and submit 1 .zip file to Blackboard

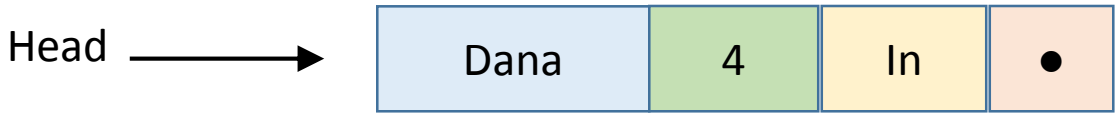
Running the Program

- Assume the user (i.e. restaurant host/hostess) knows:
  - how many tables exist
  - the size of each table
- Assume the user will enter commands when:
  - A group enters restaurant
  - A group calls the restaurant
  - A table becomes available for another group
  - A group wants to know how many people are ahead of them
  - etc

Running the Program

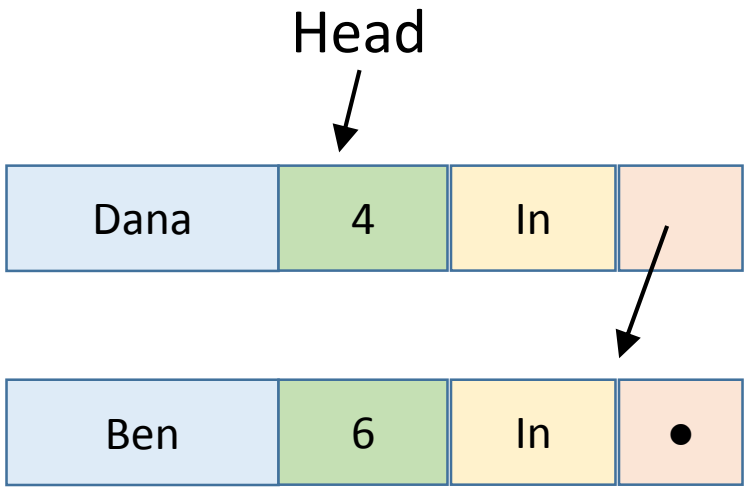
- Assume no one is waiting when the program is started
  - The wait queue (linked list) is empty
- Once a group can not be immediately seated, their name is added to the wait queue:
- Ex: Group of 4 with name of Dana enters restaurant
  - Command: a 4 Dana

Linked List should look as follows after proper code is run:



Running the Program

- Now, a group of 6 with name of Ben enters
- Command: a 6 Ben
  - What are the steps?
- Check that name Ben is not in list
- Create new node and add to end of list

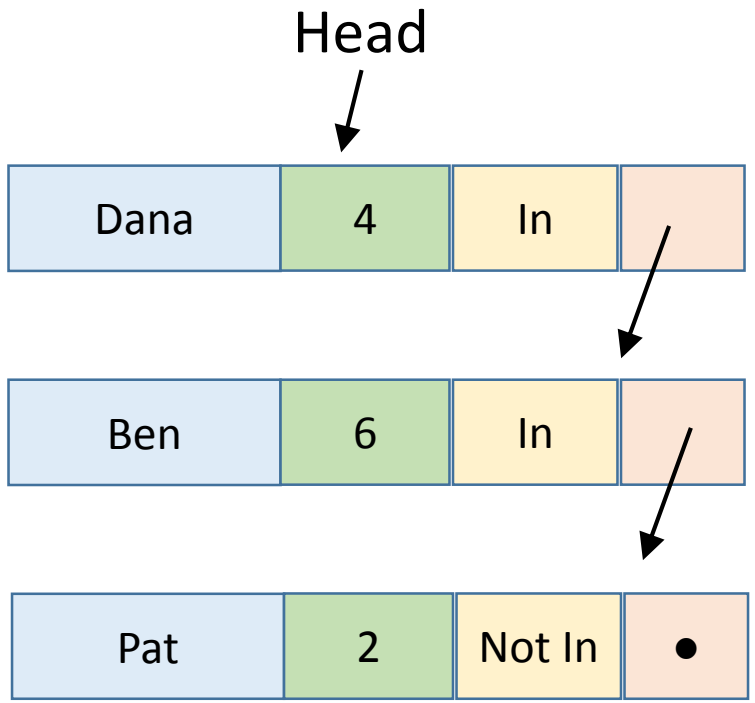


Running the Program

- Now, a group of 2 with name of Pat Calls Ahead
- Command: c 2 Pat

- Check that name Pat is not in list

- Create new node and add to end of list

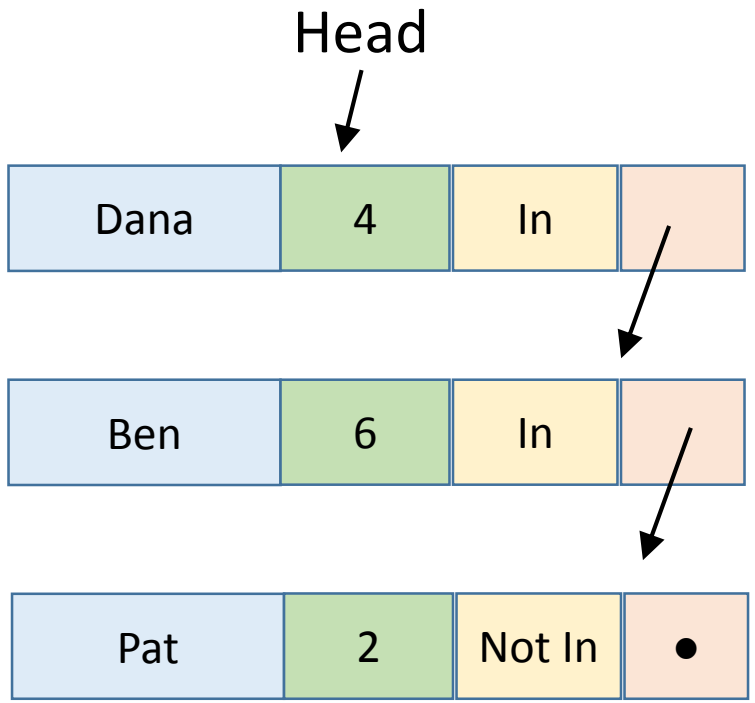


Running the Program

- Now, a group of 2 with name of Ben Enters,
- Command: a 2 Ben

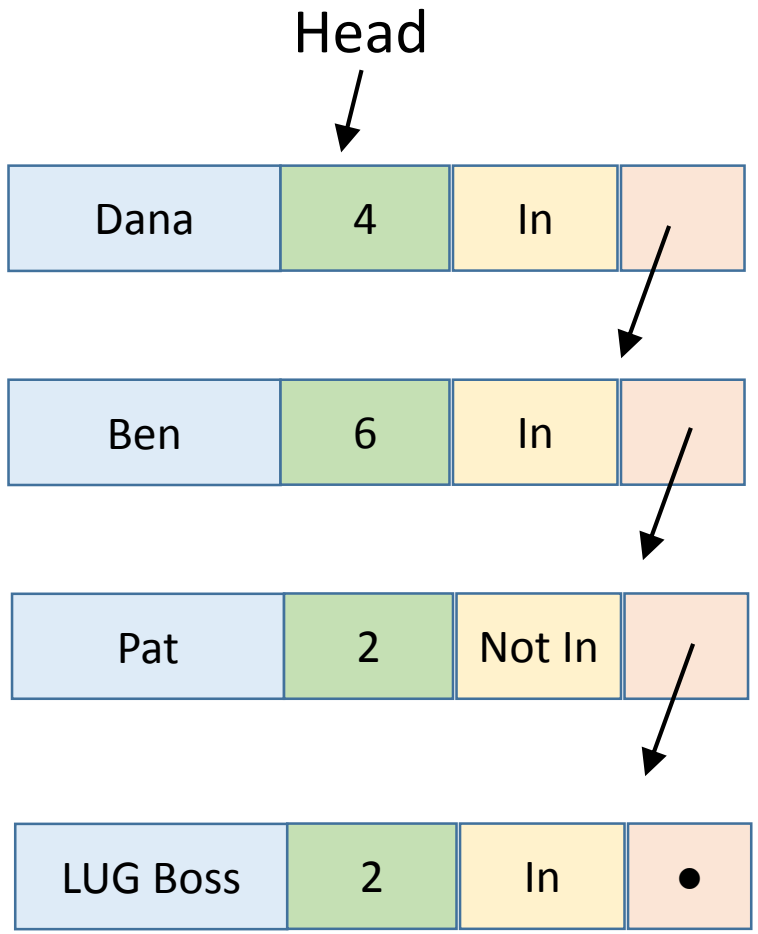
- Check that name Ben is not in list
  - The name already exists!

- Report that name already exists
- Do NOT change list



Running the Program

- Ben now gives name as “LUG Boss”,
- Command: a 2 LUG Boss
- Check that name “LUG Boss” is not in list
- Add new node to end of the list

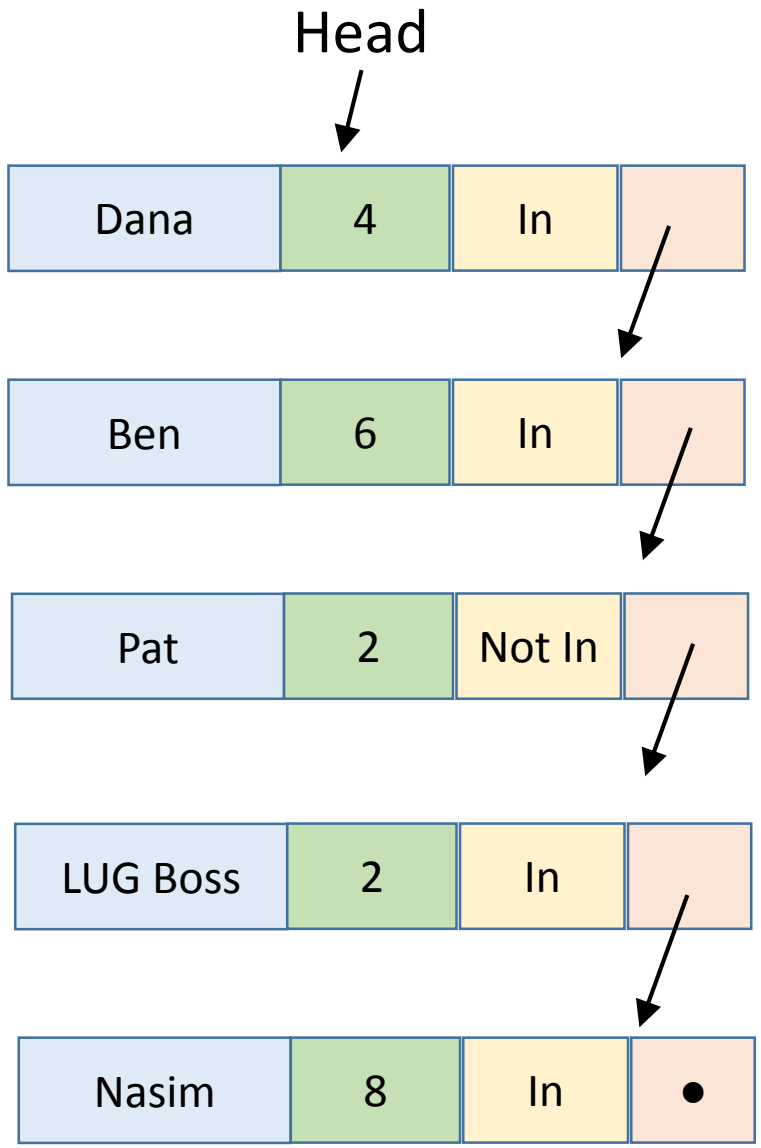


Running the Program

- Now, a group of 8 with name of Nasim enters
- Command: a 8 Nasim

- Check that name Nasim is not in list

- Add new node to end of the list



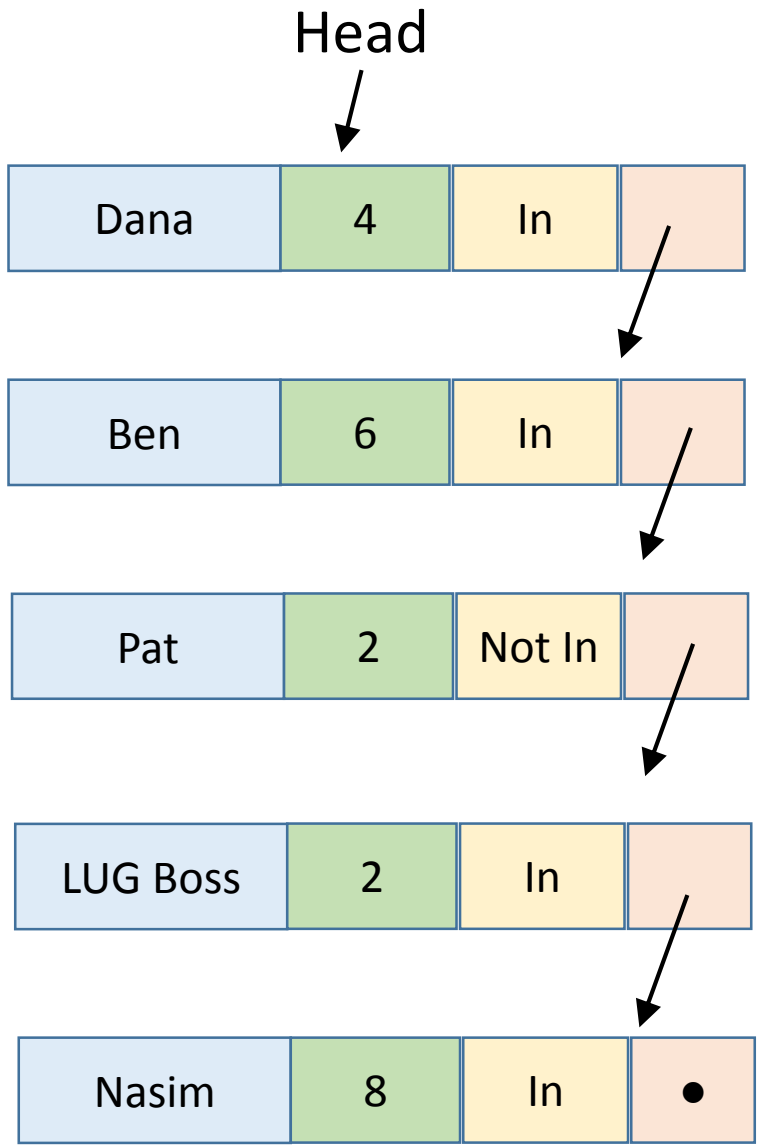
Running the Program

- LUG Boss become impatient and wants to know how many people are ahead
- Command: l LUG Boss

- However user types name wrong:
- Typed command: l lugboss

- Name of “lugboss” is not in list

- Report error that name doesn’t exist

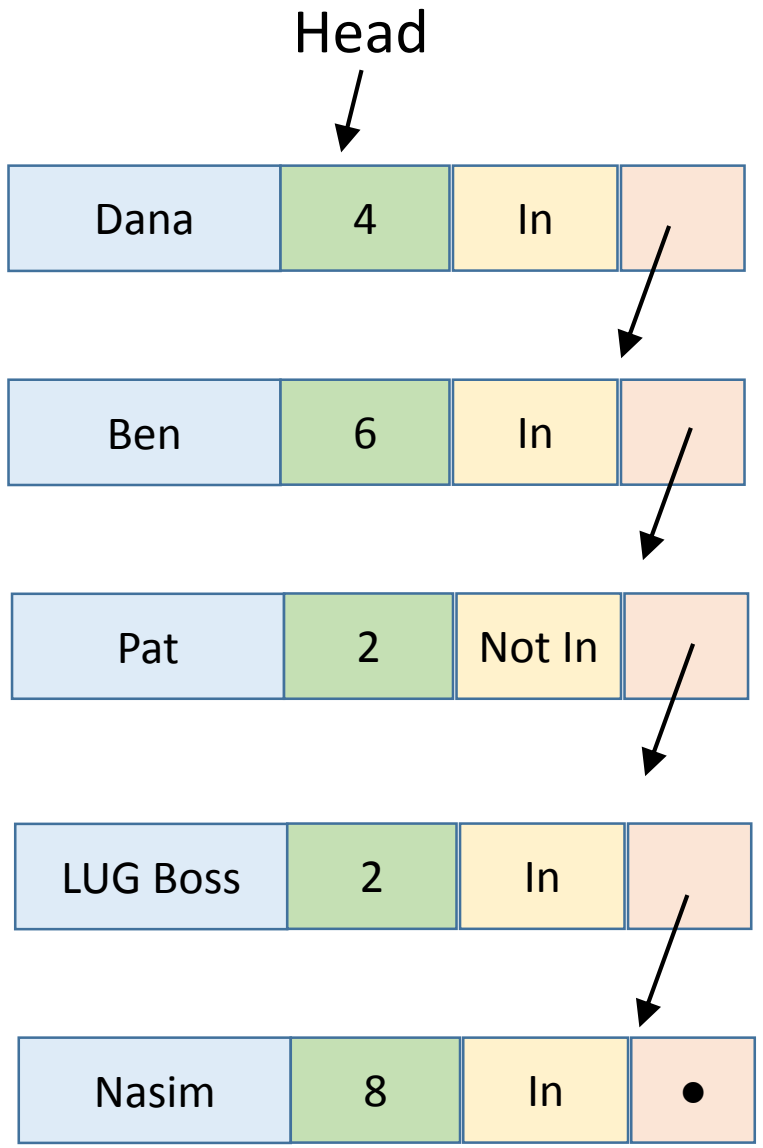


Running the Program

- LUG Boss become impatient and wants to know how many people are ahead
- Command: l LUG Boss

- Now command is typed correctly, result:

Three groups waiting before you. Size of these groups are: 4, 6, 2



Running the Program

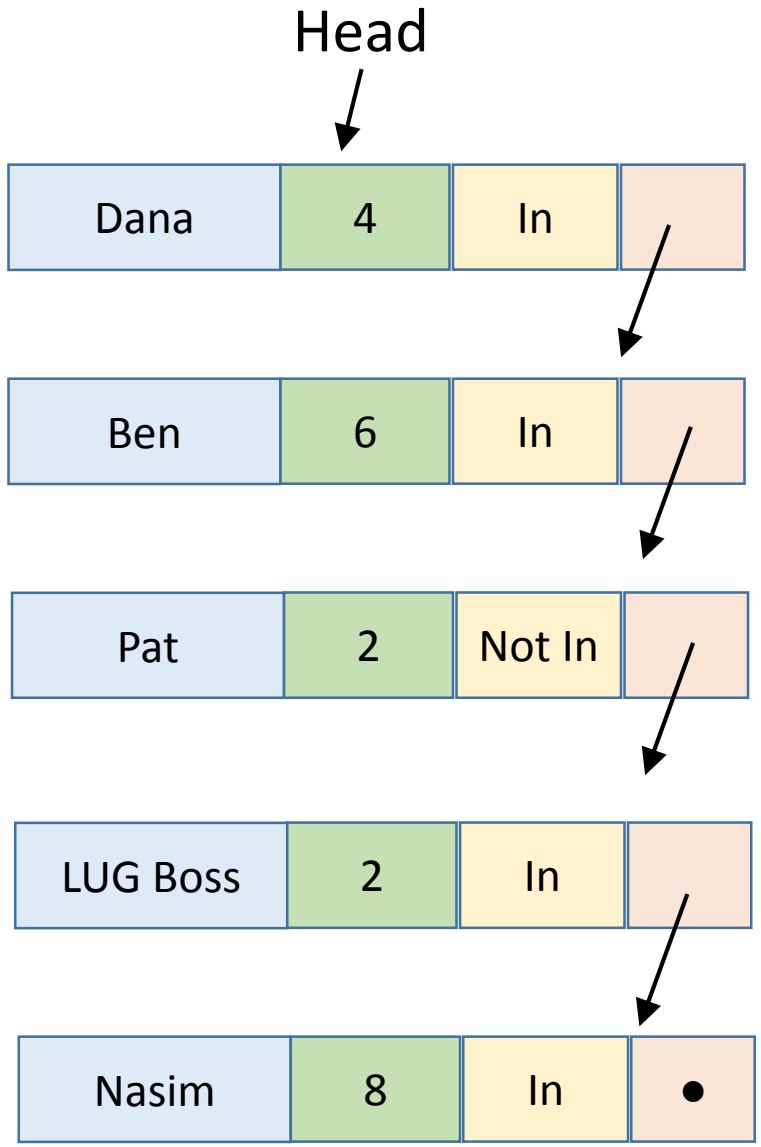
- Table of size 4 becomes available
- Command: r 4

- Program finds first group of size 4 or less that is “In” the restaurant

- Reports that Dana Party is to be seated

- Removes that node from the list

List Before the command



Running the Program

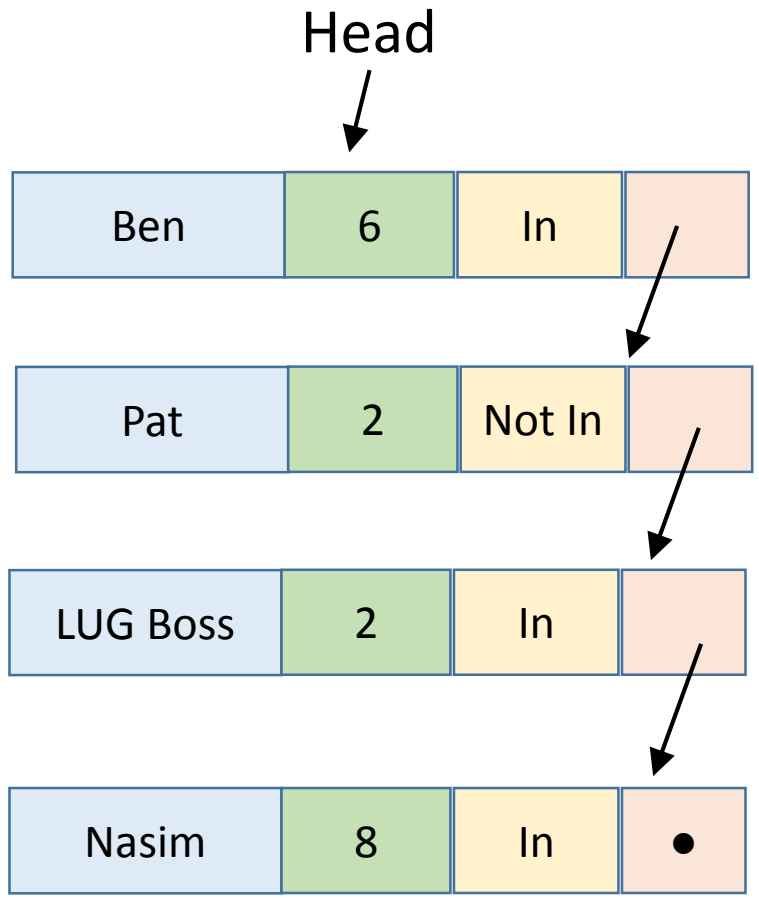
- Table of size 4 becomes available
- Command: r 4

- Program finds first group of size 4 or less that is “In” the restaurant

- Reports that Dana Party is to be seated

- Removes that node from the list

List After the command



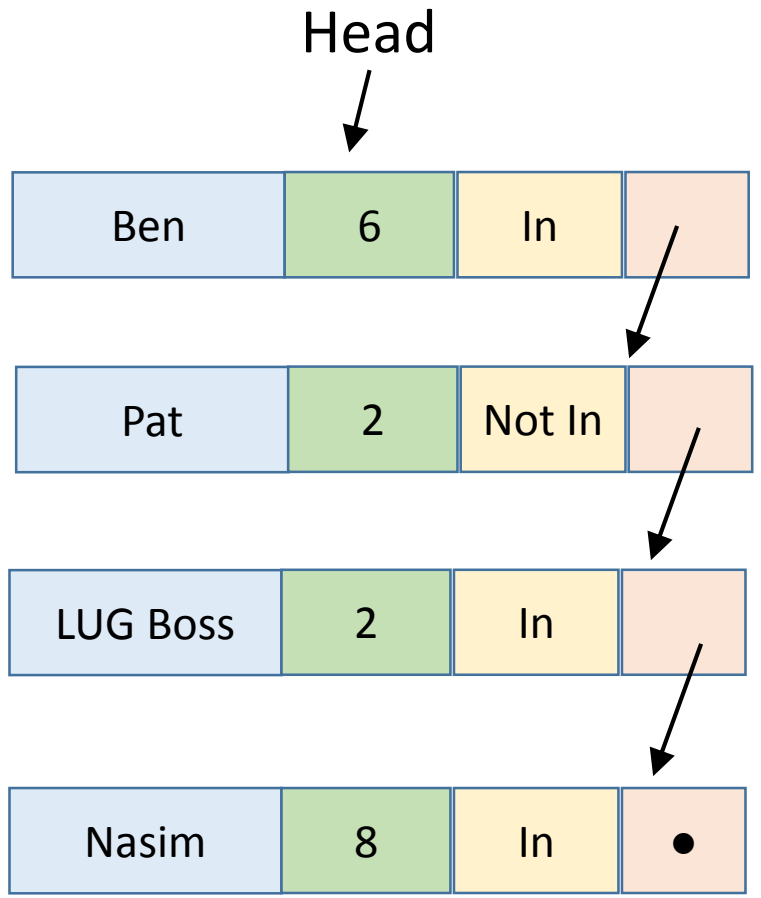
Running the Program

- Table of size 2 becomes available
- Command: r 2

- Program finds first group of size 2 or less that is “In” the restaurant
  - Ben Party is too large
  - Pat Party is NOT in the restaurant

- Reports that LUG Boss Party is to be seated
- Removes that node from the list

List Before the command



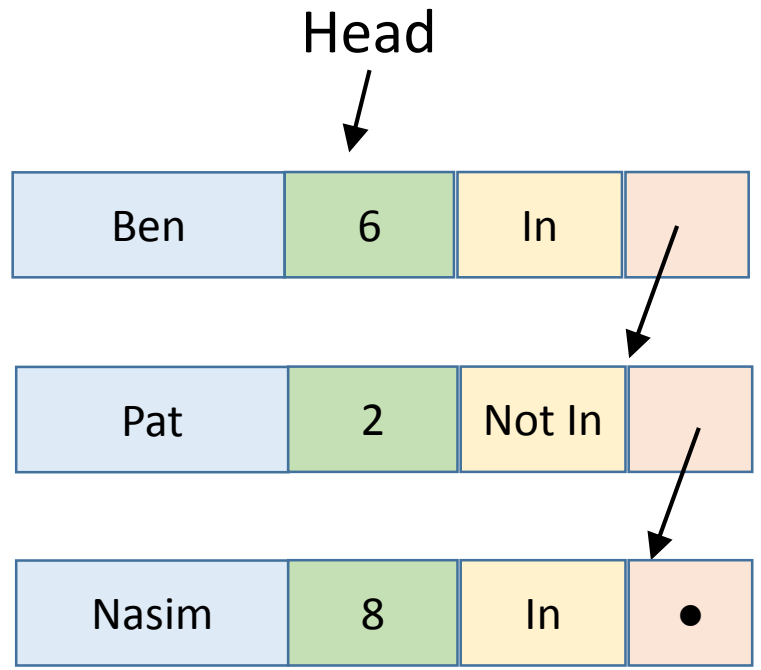
Running the Program

- Table of size 2 becomes available
- Command: r 2

- Program finds first group of size 2 or less that is “In” the restaurant
  - Ben Party is too large
  - Pat Party is NOT in the restaurant

- Reports that LUG Boss Party is to be seated
- Removes that node from the list

List After the command



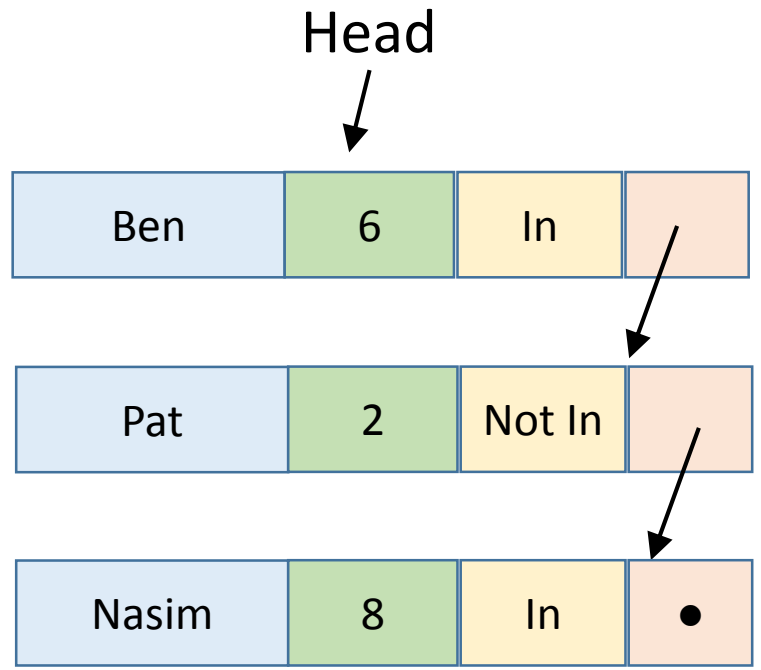
Running the Program

- Table of size 4 becomes available
- Command: r 4

- Program finds first group of size 4 or less that is “In” the restaurant
  - Ben Party is too large
  - Pat Party is NOT in the restaurant
  - Nasim party is too large

- Reports no one can be seated at this time
- List is not changed

List After the command

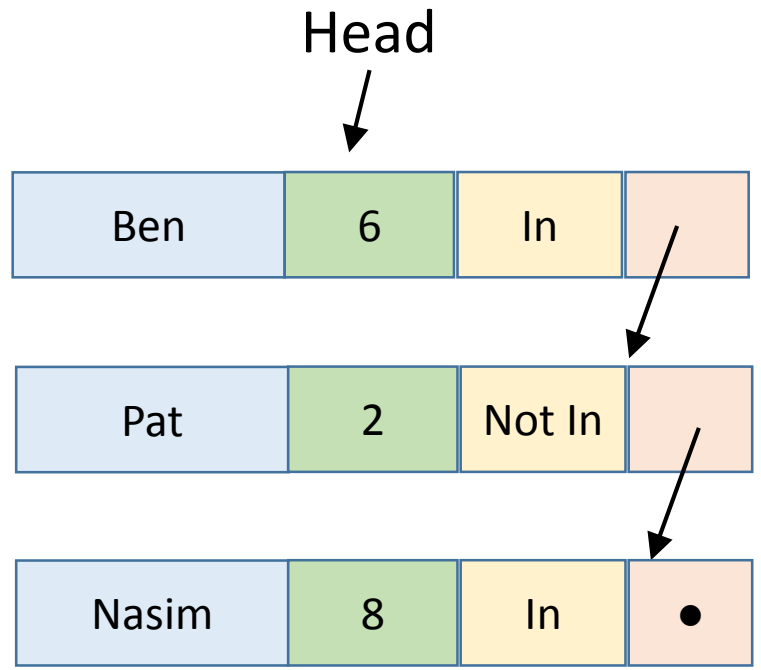


Running the Program

- Pat Party (finally) arrives
- Command: w Pat

- Program finds the Pat Party node
- Change “Not In” to “In”

List Before the command

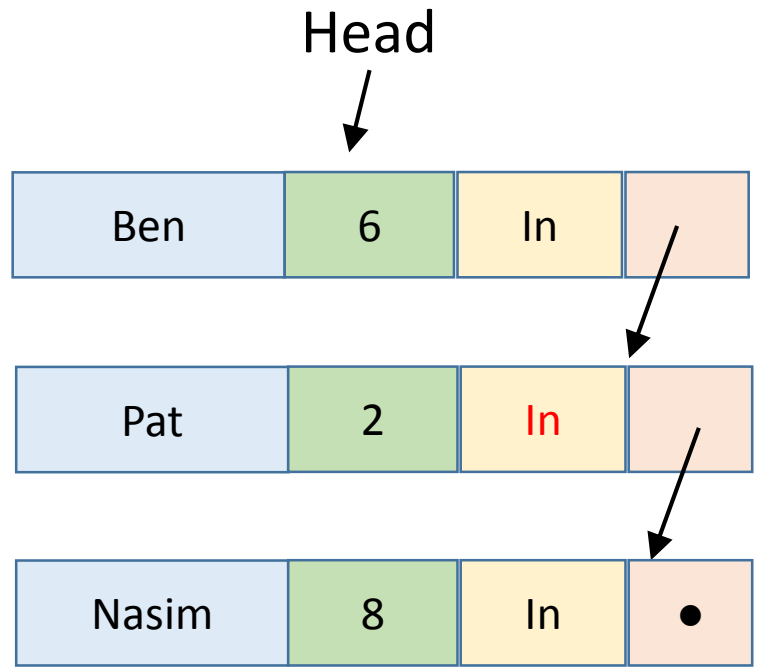


Running the Program

- Pat Party (finally) arrives
- Command: w Pat

- Program finds the Pat Party node
- Change “Not In” to “In”

List After the command



Running the Program

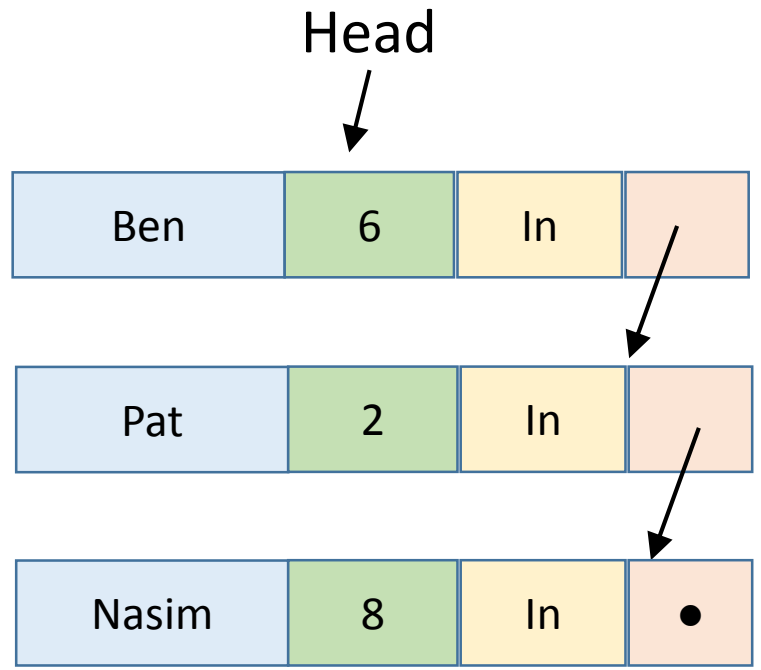
- Table of size 6 becomes available
- Command: r 6

- Program finds first group of size 6 or less that is “In” the restaurant

- Reports that Ben Party is to be seated

- Removes that node from the list

List Before the command



Running the Program

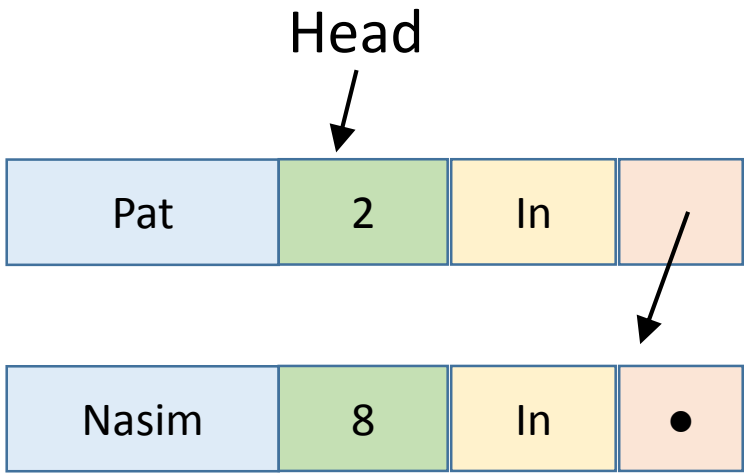
- Table of size 6 becomes available
- Command: r 6

- Program finds first group of size 6 or less that is “In” the restaurant

- Reports that Ben Party is to be seated

- Removes that node from the list

List After the command



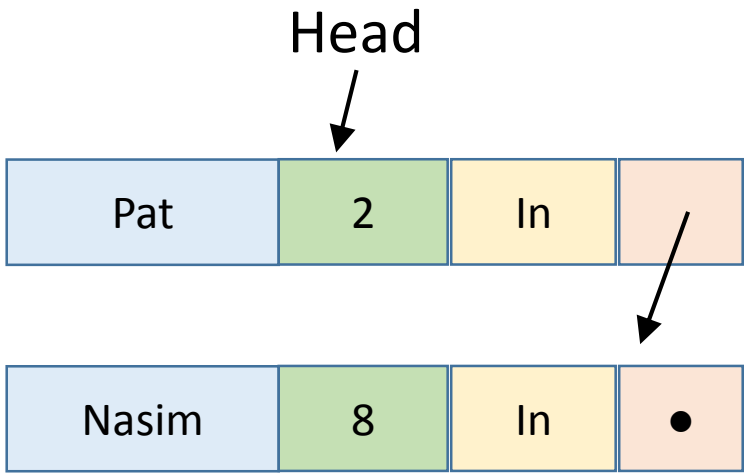
Running the Program

- Table of size 8 becomes available
- Command: r 8

- Program finds first group of size 8 or less that is “In” the restaurant

- Reports that Pat Party is to be seated
- Removes that node from the list
- Note that the program is NOT that smart

List Before the command



Running the Program

- Table of size 8 becomes available
- Command: r 8

- Program finds first group of size 8 or less that is “In” the restaurant

- Reports that Pat Party is to be seated
- Removes that node from the list
- Note that the program is NOT that smart

List After the command