Linked Lists

In this section of notes you will learn how to create and manage a dynamic list.

Arrays

Easy to use but suffer from a number of drawbacks:
1) Fixed size
2) Adding/Deleting elements can be awkward

Arrays: Fixed Size

- The size of the array cannot be dynamically changed once the memory has been allocated
- The following example won't work:

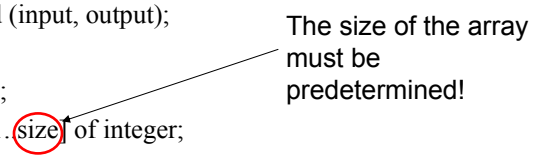
```
program notAllowed (input, output);
var
        size : integer;
        arr  : array [1..size] of integer;
begin
        write('Enter size of array: ');
        readln(size);
end.
```

- The workaround is to allocate more space than you need

Arrays: Fixed Size

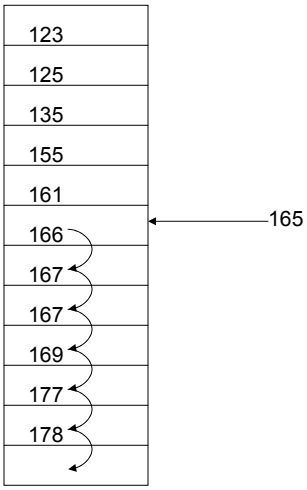
- The size of the array cannot be dynamically changed once the memory has been allocated
- The following example won't work:

```
program notAllowed (input, output);
var
        size : integer;
        arr  : array [1..size] of integer;
begin
        write('Enter size of array: ');
        readln(size);
end.
```

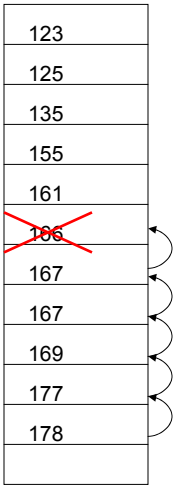


- The workaround is to allocate more space than you need

Arrays: Adding Elements In The Middle

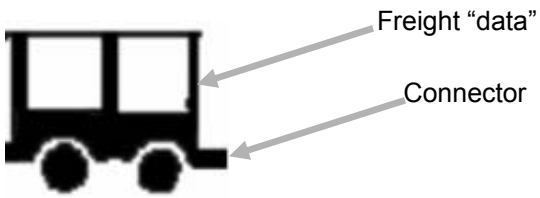


Arrays: Deleting Elements From The Middle



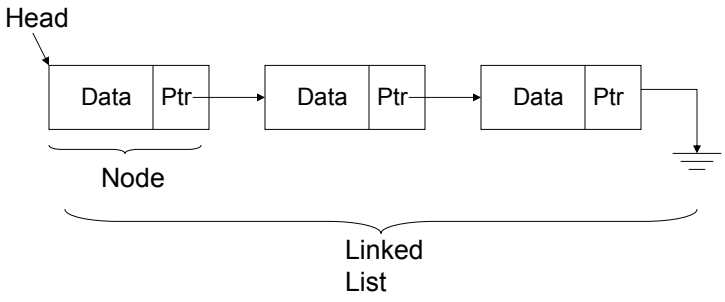
What's Needed

- A composite type that stores data dynamically and can allow for the quick addition and removal of elements



Alternative To Arrays: Linked Lists

- More complex coding may be required
- Some list management functions are more elegant (and faster)



Common List Functions

1) Declaring the list
2) Creating a new list
3) Traversing the list
4) Adding a node to the list
5) Searching the list
6) Deleting a node from the list

Note: These list functions will be illustrated by portions of an example that is a modified version of the investors program from the section on sorting, but implemented as a linked list rather than as array. The complete program can be found in Unix under:
/home/231/examples/linkedLists/investors.p

Declaring A Linked List

Format:

```
(* Part I: Defining a new type for the data (necessary if the data field is
 not a built-in type *)

(* Part II: Defining a pointer to the new type "Node" *)
 Name of the list pointer = ^ Node;

(* Part III: Defining a new type, a "Node" *)
type
  Node = record
       data        : Name of the list data;
       nextPointer : Name of the list pointer;
  end;
```

Declaring A Linked List (2)

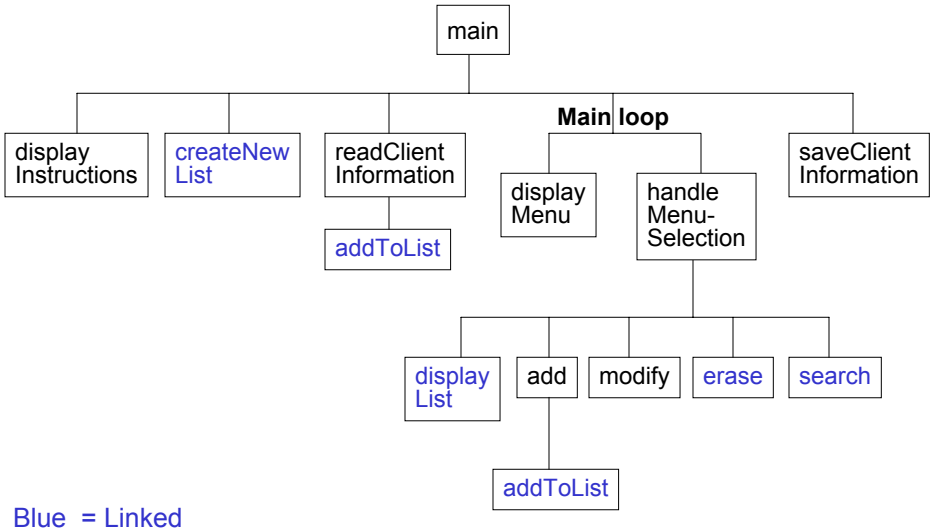
Example:

```
type
    (* Part I: Defining a new type for the data (necessary because a "Client" is
    not a built-in type *)
    Client =  record
                firstName : array [1..NAME_LENGTH] of char;
                lastName  : array [1..NAME_LENGTH] of char;
                income    : real;
                email     : array [1..EMAIL_LENGTH] of char;
              end; (* Declaration of record Client *)

    (* Part II: Defining a pointer to the new type "Node" *)
    NodePointer = ^ Node;
```


Declaring A Linked List (3)

```
(* Part III: Defining a new type, a "Node" *)
Node = record
         data        : Client;
         nextPointer : NodePointer;
       end; (* Declaration of record Node *)
```

Outline Of The Example Program



Blue = Linked list functions

Main Procedure: Example Program

```
begin
   var tamjClientList          : NodePointer;
   var menuSelection           : char;
   var desiredName             : NameArray;
   var isFound                 : boolean;
   var previousFirst           : NodePointer;
   var investorData            : text;
   var updatedInvestorData     : text;

   displayInstructions;
   createNewList(tamjClientList);
   readClientInformation(tamjClientList, investorData);
```

Main Procedure: Example Program (2)

```
  repeat
  begin
    displayMenu;
    readln(menuSelection);
    writeln;
    handleMenuSelection(menuSelection, tamjClientList);
  end; (* repeat-until *)
  until (menuSelection = 'Q') OR (menuSelection = 'q');

  (* Write updated information out to disk *)
  saveClientInformation(tamjClientList, updatedInvestorData);
end.
```


Creating A New List

Description:

The pointer to the beginning of the list is passed into the procedure as a variable parameter and initialized to NIL signifying that the new list is empty.

Example:

```
procedure createNewList (var clientList : NodePointer);
begin
  clientList := NIL;
end; (* createNewList *)
```

Reading The Client Information From A File

```
procedure readClientInformation (var clientList      : NodePointer;
                                 var investorData : text);
var
  newNode   : NodePointer;
  newClient : Client;
begin;
  writeln;
  reset(investorData, 'investorList');
  writeln('Opening file "investorList" for reading');
```


Reading The Client Information From A File (2)

```
 while NOT EOF (investorData) do
 begin
   new(newNode);
   with newClient do
   begin
       readln(investorData, firstName);
       readln(investorData, lastName);
       readln(investorData, income);
       readln(investorData, email);
       readln(investorData);
   end;
   newNode^.data := newClient;
   addToList (clientList, newNode);
end;
```

Handling/Executing The Main Menu Of Operations

```
procedure handleMenuSelection (     menuSelection : char;
                                    var clientList        : NodePointer);
var
  isFound         : boolean;
  previousFirst   : NodePointer;
  desiredName     : NameArray;
begin
  case (menuSelection) of
    'D', 'd' :
    begin
      displayList (clientList);
    end;

    'A', 'a' :
    begin
      add (clientList);
    end;
```


Handling/Executing The Main Menu Of Operations (2)

```
    'M', 'm' :
    begin
      modify (clientList);
    end;

    'E', 'e' :
    begin
      erase (clientList);
    end;

    'S', 's' :
    begin
      write('Enter last name of contact that you wish to search for: ');
      readln(desiredName);
      search (clientList, desiredName, isFound, previousFirst);
    end;
```

Handling/Executing The Main Menu Of Operations (3)

```
  'Q', 'q' :
    begin
      writeln;
      writeln('Thank you for using the investor 2000 (TM) program.');
      writeln('Come again!');
      writeln;
    end;

    else
    begin
      writeln;
      write('Please enter one of the following options: D, A, M, E,');
      writeln('S or Q');
      writeln;
    end;

  end; (* case *)
end; (* handleMenuSelection *)
```


Traversing The List (Display List)

Description:

Steps (traversing the list to *display* the data of each node onscreen)

1. Start by initializing a pointer to point to the beginning of the list.
2. If the pointer is NIL then display a message onscreen indicating that there are no nodes to display and stop otherwise proceed to next step.
3. Process the node (e.g., display the data onscreen)
4. Move on to the next node by following the node's nextPointer (set the pointer to point to the next node).
5. Check if the pointer is NIL.
   a) If the pointer is NIL then stop
   b) If the pointer is not NIL then go to step #3.

Traversing The List (2)

Example:

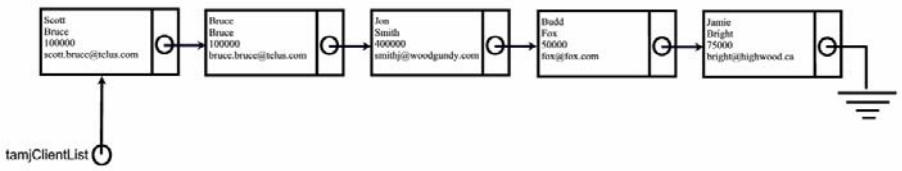
```
procedure displayList (clientList : NodePointer);
var
  currentNode : NodePointer;
begin
  currentNode := clientList;
  writeln('CLIENT LIST':20);

  if (currentNode = NIL) then
  begin
    writeln;
    writeln('List is empty, no clients to display');
    writeln;
  end;
```


Traversing The List (3)

```
  while (currentNode <> NIL) do
  begin
    writeln('First name: ':20, currentNode^.data.firstName);
    writeln('Last Name: ':20, currentNode^.data.lastName);
    writeln('Income $':20, currentNode^.data.income:0:2);
    writeln('Email: ':20, currentNode^.data.email);
    writeln;
    currentNode := currentNode^.nextPointer;
  end;
end; (* displayList *)
```

Traversing The List (4)

Traversing The List (5)



Adding A Node To The End Of The List

Description:

Variables

1. There are two pointers to the list:
   a) Current pointer – traverses the list from beginning to end
   b) Previous to first pointer – points to the node that occurs just prior to the first successful match.

Adding A Node To The End Of The List (2)

Steps:

1. Assign current pointer to the front of the list.
2. If the current pointer is NIL then the list is empty and add the node to the front of the list and stop.
3. Otherwise traverse the list with two pointers, one pointer (current pointer) goes past the end of the list (to the NIL value), the other stays one node behind it (previous pointer).
4. Attach the new node to the last node in the list (the one reached by the previous pointer).
5. The next pointer of the new node becomes NIL (indicating that this is the end of the list).

Adding A Node To The End Of The List (3)

Example:

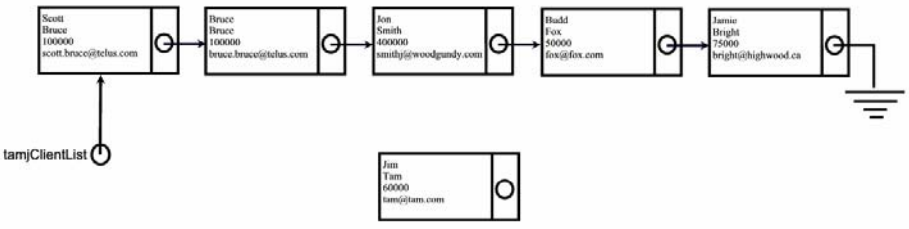
```
procedure addToList  (var clientList : NodePointer;
                                newNode    : NodePointer);
var
  currentNode  : NodePointer;
  previousNode : NodePointer;
begin
  if (clientList = NIL) then
  begin
    clientList := newNode;
    newNode^.nextPointer := NIL;
  end
```


Adding A Node To The End Of The List (4)

```
  else
  begin
    currentNode := clientList;
    while (currentNode <> NIL) do
    begin
      previousNode := currentNode;
      currentNode := currentNode^.nextPointer;
    end;
    previousNode^.nextPointer := newNode;
    newNode^.nextPointer := NIL;
  end;
end; (* addToList *)
```

Adding A Node To The End Of The List (5)

Adding A Node To The End Of The List (6)



Searching The List

Description:

The procedure is run in order to find a node or nodes that has a field which matches some desired value. Either the node or nodes will be found in the list or else the procedure will have searched every node in the list and have found no matches. A flag will be set to true or false indicating whether the search was successful or a failure.

Main variables:

1. There are two pointers to the list:
    a. Current pointer – traverses the list from beginning to end.
    b. Previous to first pointer – points to the node that occurs just prior to the first successful match.

    Note: The second pointer is not used when the user only wants to search the list. It is needed when the person wishes to erase a node from the list. Since the erase procedure calls the search procedure, it needs a pointer to the node prior to the one to be deleted.
2. A Boolean that indicates the status of the search.

James Tam

Searching The List (2)

Steps:

1. Current pointer starts at the beginning of the list. Since the search has not yet begin, previous is set to NIL and the flag is set to false.
2. A check is performed to determine if the node is a match. If this is the case and the flag is still false (indicating that we haven't found a previous node that was a successful match) set the flag to true (since a match was just found). Since the search function requires a list of all matches (and not just the first instance) don't stop searching the list.
3. The previous pointer to will be set to point to the current node if the flag was false prior to the match (i.e., this is the first instance found). The previous pointer will not change if the flag was already set to true (the previous pointer will still track the node just prior to the node which first meets the search criteria).
4. Move on to the next node (by setting the current pointer to the current node's next pointer).
5. Continue step 2 – 4 until the end of the list is reached (current node is NIL).

James Tam

Searching The List (3)

Example:

```
procedure search (     clientList       : NodePointer;
                       desiredName      : NameArray;
                  var isFound           : boolean;
                  var previousFirst     : NodePointer );
var
  currentNode : NodePointer;
begin
  if (clientList = NIL) then
      writeln('Nothing to search, list is empty')
  else
  begin
     currentNode := clientList;
     previousFirst := NIL;
     isFound := False;
```

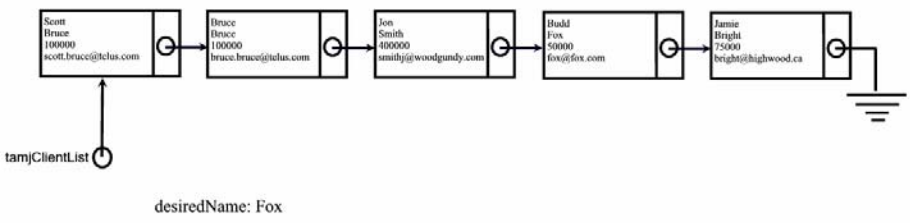

Searching The List (4)

```
  while (currentNode <> NIL) do
  begin
     if (desiredName = currentNode^.data.lastName) then
     begin
        writeln('Found contact':20);
        writeln('First name :':20, currentNode^.data.firstName);
        writeln('Last name :':20, currentNode^.data.lastName);
        writeln('Income $':20, currentNode^.data.income:0:2);
        writeln('Email :':20, currentNode^.data.email);
        writeln;
        isFound := True;
     end; (* if-then *)
     if (isFound = False) then
        previousFirst := currentNode;
     currentNode := currentNode^.nextPointer;
  end; (* while: Search list for matches *)
```

Searching The List (5)

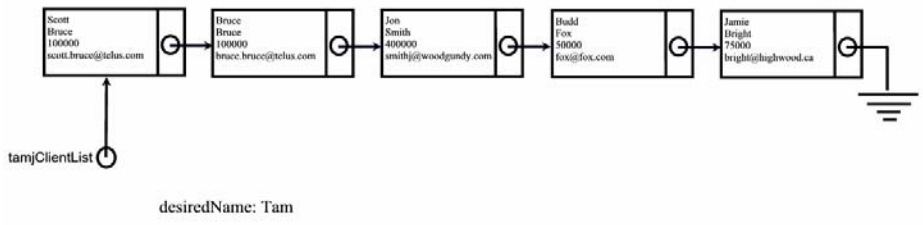
```
    (* Traversed whole list and no matches were found *)
     if (isFound = False) then
         writeln('Contact not found in list');

  end; (* else: Searching the list *)
end; (* search *)
```


Searching The List (6)



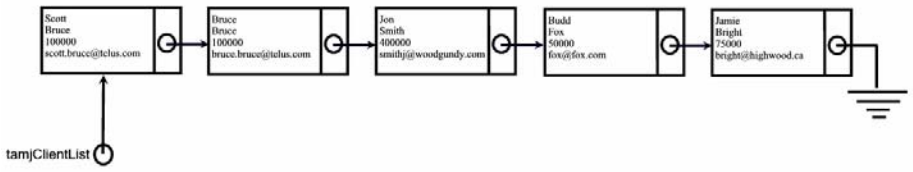
desiredName: Fox

Searching The List (7)



desiredName: Tam

Searching The List (8)



desiredName: Bruce

Deleting A Node From The List

Description:

Main variables:

1. A flag that indicates the status of the search. If the search was successful then it was true that the item was found (flag will be set to true). If the search was a failure then it was false that item was found (flag will be set to false).
2. A pointer that points to the node just prior to the one to be deleted. If the flag was set to true then the pointer contains the address of the previous node. If the pointer is NIL then the node to be deleted is the first node (nothing is previous to this node so there is no address). If the the pointer is not NIL then it contains the address of the node to be deleted.
3. A temporary pointer that points to the node to be deleted. It is needed so that the program can retain a reference to this node and free up the memory allocated for it.

Deleting A Node From The List (2)

Steps

1. Search the list (by calling the search procedure) to determine if there exists a node that matches the necessary criteria for deletion.
2. Check the flag to determine if the search was successful or not. If the flag is false then there is no matching node in the list. End procedure: There is no matching node to delete.
3. Check to see if the node to be deleted is the first node in the list or not by checking if the previous pointer is NIL.
4. If the node to be deleted is the first node then have a temporary pointer point to the first element and make the front of the list the second element.
5. If the node to be deleted is not the first node then have a temporary pointer point to the node to be deleted. Set the next pointer (of the node previous to the one to be deleted) point to the node after the node to be deleted (bypassing the node to be deleted)
6. For steps 4 & 5 free up the memory allocated by the node to be deleted by dereferencing the temporary pointer.

Deleting A Node From The List (3)

Example:

```
procedure erase (var clientList : NodePointer);
var
  desiredName : NameArray;
  previousFirst : NodePointer;
  temp          : NodePointer;
  isFound       : boolean;
begin
  (* Don't bother to do a search if there is nothing in the list *)
  if (clientList = NIL) then
  begin
    writeln('List is empty, nothing to erase');
  end
```

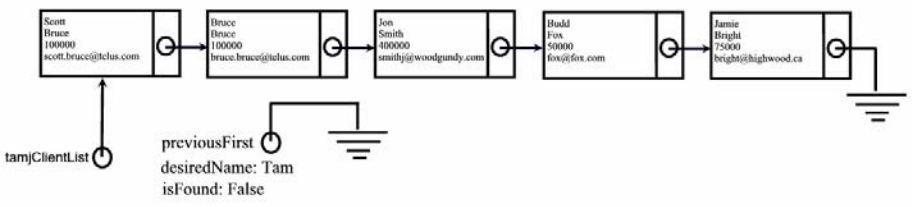

Deleting A Node From The List (4)

```
 else
  begin
    write('Enter last name of client to erase from list: ');
    readln(desiredName);
    search(clientList, desiredName, isFound, previousFirst);
```

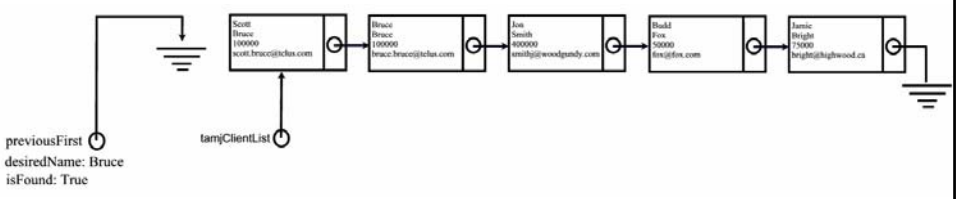
Deleting A Node From The List (5)

```
    if (isFound = True) then
    begin
       writeln('Deleting first instance of ', desiredName);
       if (previousFirst = NIL) then
       begin
          temp := clientList;
          clientList := clientList^.nextPointer;
       end (* if-then: delete first *)
       else
       begin
          temp := previousFirst^.nextPointer;
          previousFirst^.nextPointer := temp^.nextPointer;
       end; (* else: not deleting first *)
       dispose(temp);
    end; (* if-then: check if client in list *)
  end; (* else: searching list when list isn't empty *)
end; (* erase *)
```

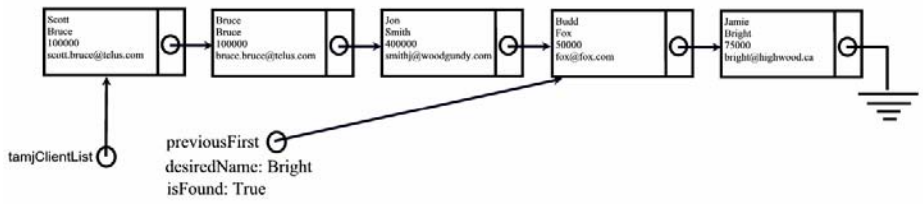

Deleting A Node From The List (5)



Deleting A Node From The List (6)



Deleting A Node From The List (7)



You Should Now Know

- What is a linked list
- What are the advantages of using a linked list over using an array
- What is the disadvantage of using a linked list over using an array
- Common list operations
  - Declaring a list
  - Creating a new list and initializing the list with data
  - Traversing the list (e.g., to display the contents of the nodes)
  - Adding new nodes to the list
  - Searching the list
  - Deleting an existing node from the list