

Linked List Exercises, #1 Phill Conrad for CISC181, Spring 2005 CIS Dept., University of Delaware

©2004, Phillip T. Conrad, Permission to copy for non-commercial purposes granted
provided this copyright notice is maintained; all other rights reserved.

Drawing Pictures of Linked Lists

One thing you should know how to do is read some C++ source code, and draw the linked list that would be created by that source code.

This handout illustrates that process.

Code to set up the struct

First, consider the following source code

```
// linkedList1.cpp for CISC181 11/16/04
// P. Conrad

#include <iostream>
#include <cstring>
using namespace std;

#define NAMELEN 10

struct Score_S
{
    char name[NAMELEN];
    int score;
    Score_S *next;
};
```

This source code sets up a struct definition called `Score_S`. A `Score_S` contains three fields: name, score, and a next pointer.

Allocate space for a node

Now consider the following code:

```
// linkedList1.cpp for CISC181 11/16/04 P. Conrad
// continued...

int main(void)
{
    Score_S *p; // a "working" pointer for allocating new structs
    Score_S *head, *tail; // points to head and tail of linked list

    p = new Score_S;

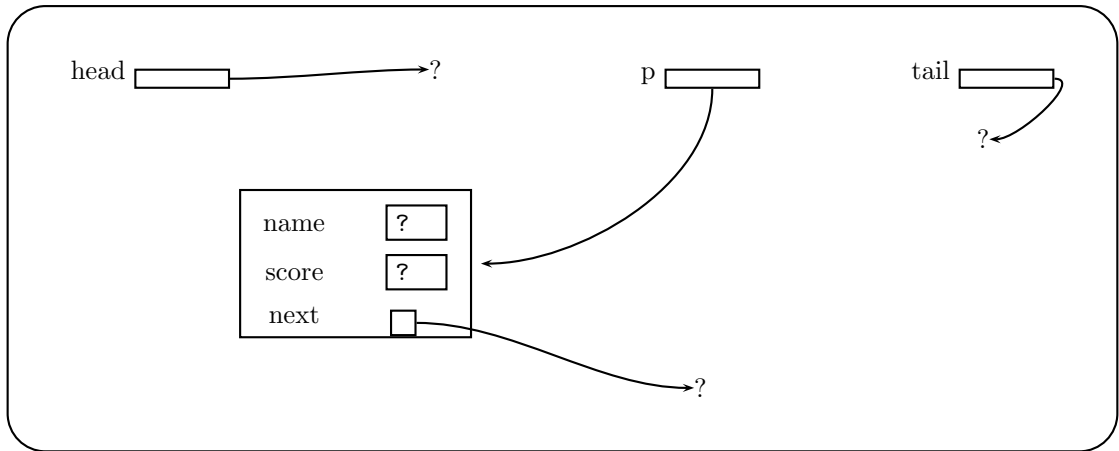
    ...
```

This code allocates three pointers called p, head and tail. Initially these pointers don't point to anything.

The code `p = new Score_S` allocates space from the heap for a new `Score_S` structure, and makes p point to it. This `Score_S` structure will be the first structure in a linked list that we will build. We sometimes use the word "Node" to refer to a structure like this that is part of a linked list.

The structure set up by this code is shown below. Note that p points to a new node, but the contents of that node are unknown (as shown by the fact that contents of the name and score fields are shown as "?").

Also note that head, tail, and the next fields of the new `Score_S` structure are all pointers that have not yet been initialized with values, therefore we shown them pointing to "?".

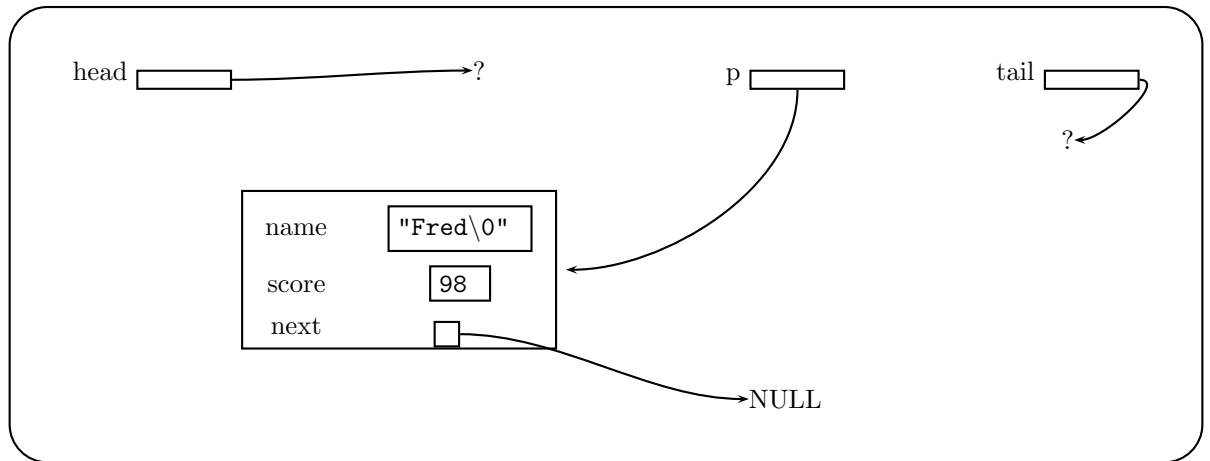


Initialize the values in the node just created

Now consider the following code that fills in the values of the fields in the `Score_S` structure:

```
...
strcpy((*p).name, "Fred", NAMELEN);
(*p).name[NAMELEN-1] = '\0';
(*p).score = 98;
(*p).next = NULL;
...
```

After this code, the picture changes to look like the following. Note that the values are filled in, and the next pointer no longer points to "?" (indicating an unknown value); instead, it points to NULL, which is a known value (but one that happens to be zero, representing the end of a list.)



Make head and tail point to p

The next two lines of code initialize the head and tail pointers to point the the node we just created. This is how a linked list starts typically—with just one node, that is both the head and the tail of the list.

...

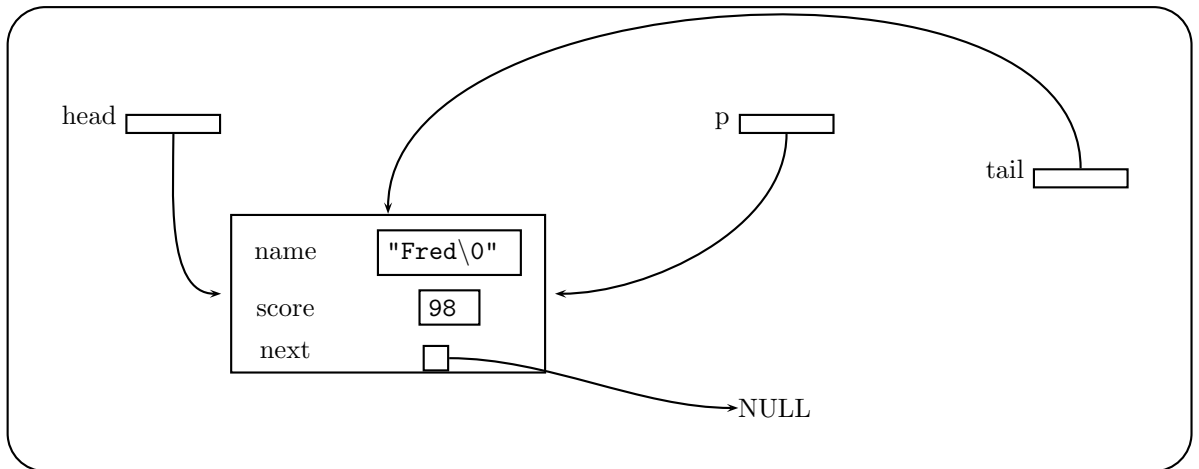
```
head = p;
tail = p;
```

...

The diagram below shows the result of these lines of code.

Note that the effect of `head=p;` is to “make head point to what p points to”. It does NOT “make head point to p.” Similarly, `tail=p;` makes tail point to what p points to. After we do the two lines below, head tail and p all point to the same thing.

Also note that although head, tail and p point to different sides of the node that contains the name "Fred", there is no difference; we just draw the pointers entering the "Fred" node on different sides of the node to make the diagram easier to read.



Allocate the second node

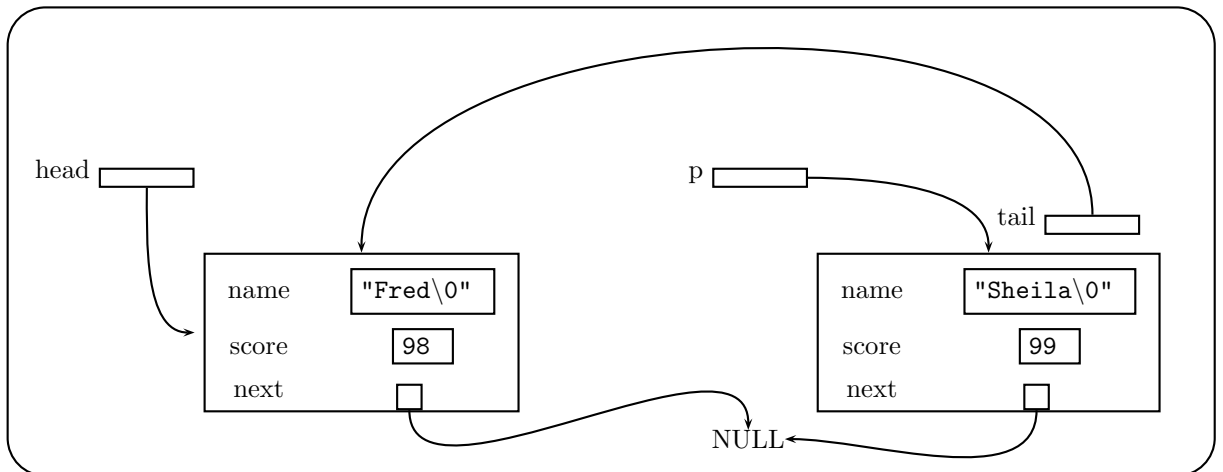
We now allocate a second node, and fill in the values:

...

```
p = new Score_S;
strcpy(p->name, "Sheila", NAMELEN);
p->name[NAMELEN-1]='\0';
p->score = 99;
p->next = NULL;
```

...

This picture shows p pointing to the new node, and the values filled in:

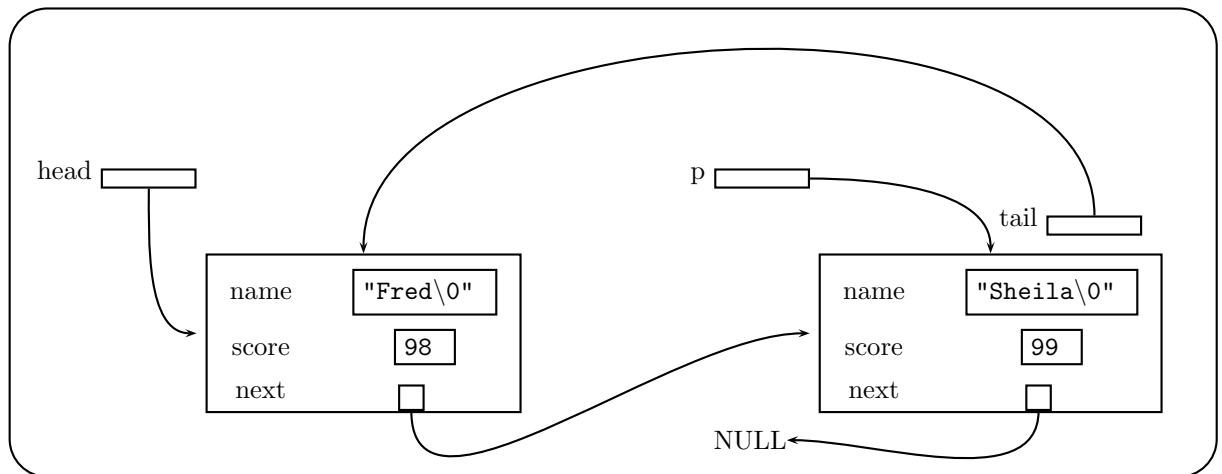


Link the first node to the second node

Now we link the node containing Fred to the node containing Sheila. We use `head -> next` to access the “next” field in the node containing Fred, and we make it point where `p` points.

```
...  
  head -> next = p;  
...
```

This diagram shows the updated link:

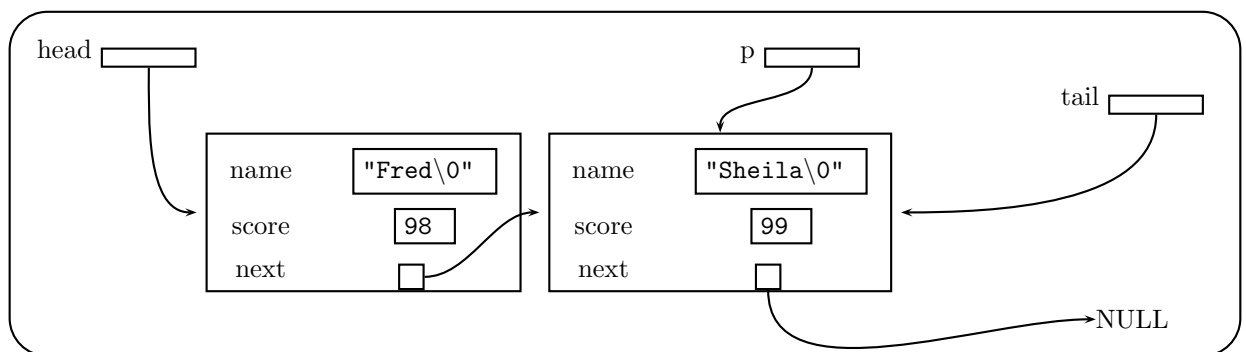


Update the tail pointer

Now, Fred is no longer the tail of the list; Sheila is the new tail. So, we update the tail pointer:

```
...  
  tail = p;  
...
```

This diagram shows that `tail` now points to the node containing Sheila.



Allocate the third node

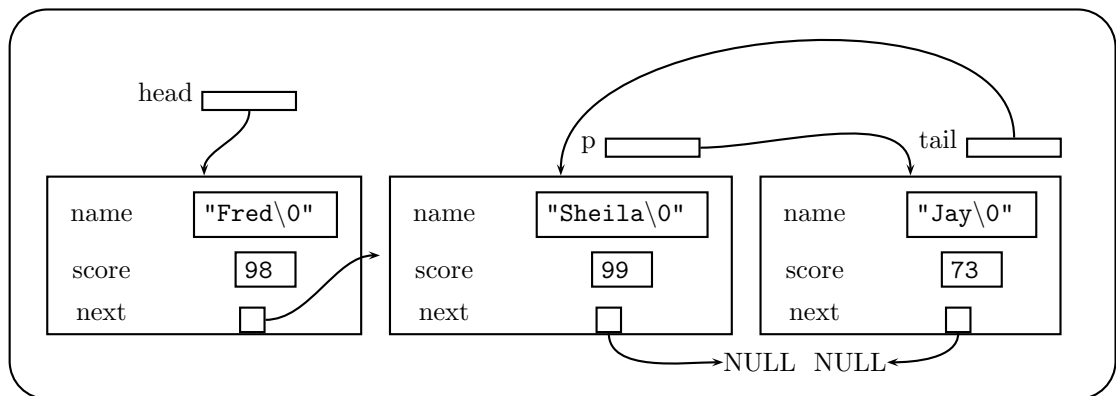
Now we allocate yet another node: this time a node for Jay, and we will in the fields.

...

```
p = new Score_S;  
strncpy(p->name, "Jay", NAMELEN);  
p->name[NAMELEN-1]='\0';  
p->next = NULL;  
p->score = 73;
```

...

This diagram shows `p` pointing to the new node, and the new node pointing to `NULL`. Note that the node containing Jay is not really a part of the list yet at all.



Add new node at end of list

The following code finally adds the third node into the list, by first updating the next pointer of the node that is currently at the tail of the list (i.e. `tail -> next`), and then updating the tail pointer.

Note that it has to be done in that order. In the next section we consider what happens if the opposite order is used.

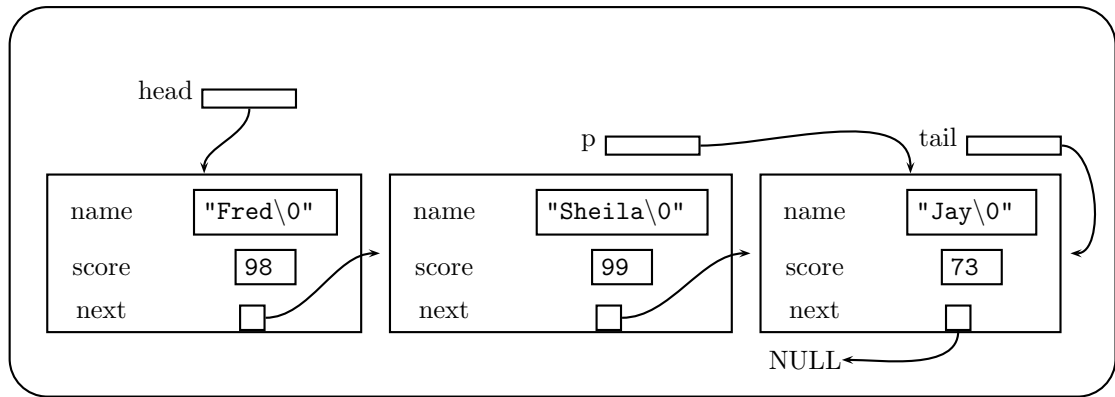
...

```
tail -> next = p;  
tail = p;
```

...

Here's the completed list. Note that `head` points to the first node in the list, and `tail` points to the last node in the list.

The p pointer is just a left over pointer that was used temporarily in the construction of the list, and its value is no longer really relevant, but we show its final value anyway.



An example of what can go wrong

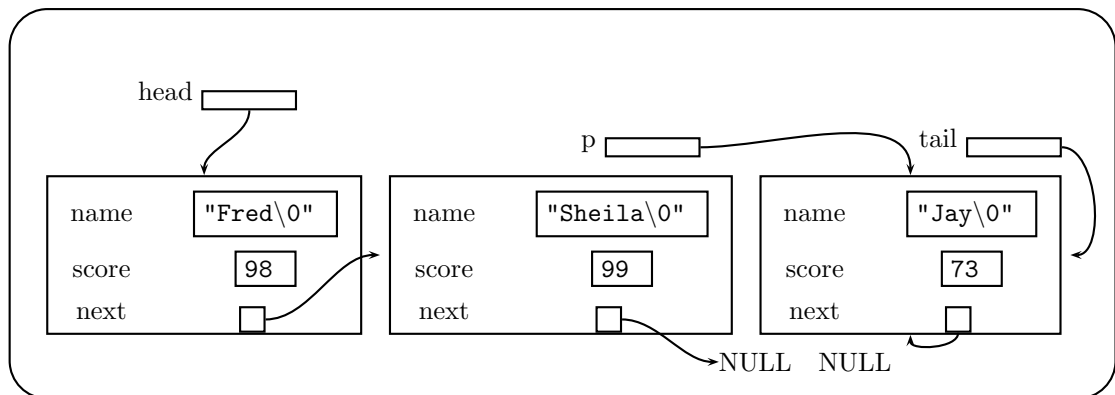
Consider what would happen if we reversed the last two lines of code, as follows:

...

```
tail = p;  
tail -> next = p; /* wrong order !!!!! */
```

...

In this case, the pointers would end up looking like the following diagram. Note that the tail ends up pointing to the right place (Jay), but the node containing Sheila is still the end of the list (if we follow the pointers from the head), Jay is not really part of the list (again, following the pointers from head), and the node containing Jay ends up pointing to itself.



Clearly this is not what we want! However, you might be given code like this as a homework exercise or an exam question, and asked to draw the diagram corresponding to the code. The reason for this is so that if and when you are debugging your own code (or that of another programmer) containing such errors, you'll be able to draw a picture and diagnose the problem.

Therefore it is important to be able to draw a picture of what the code *actually does* rather than what you think it *should do*.

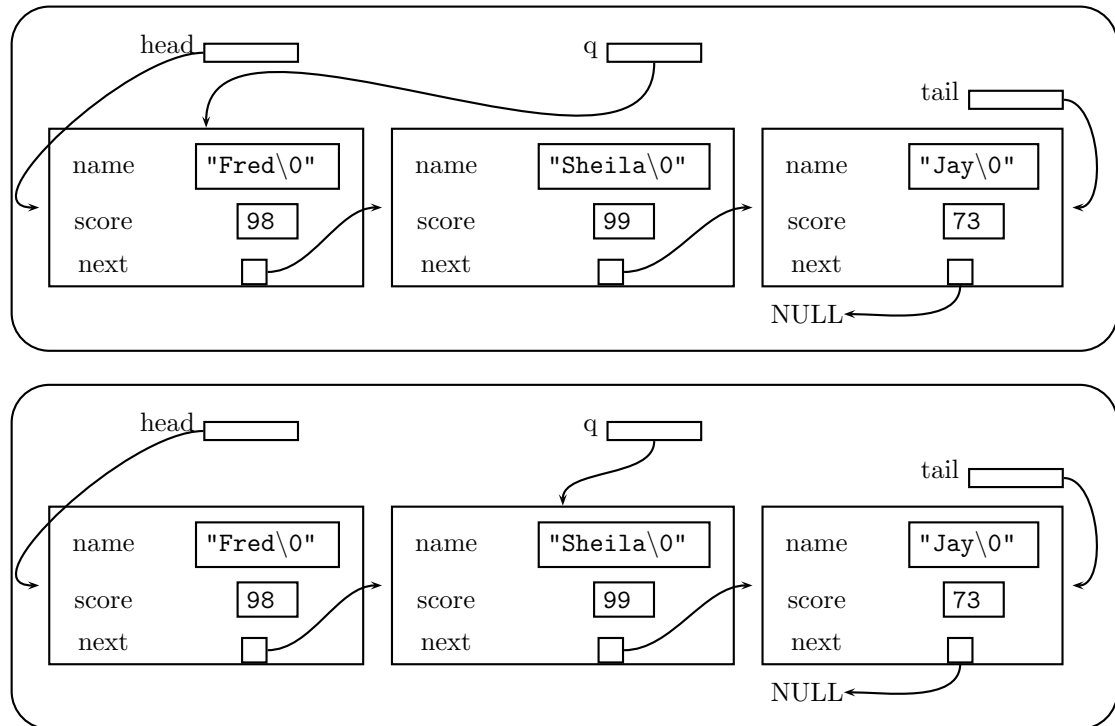
Traversing the list

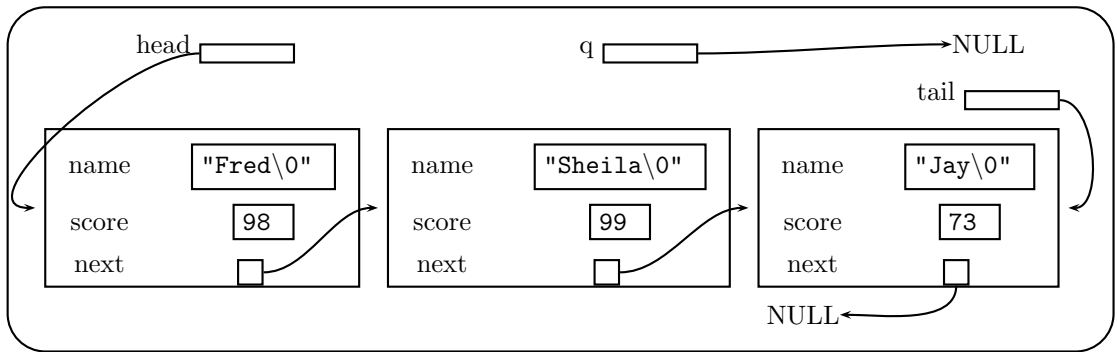
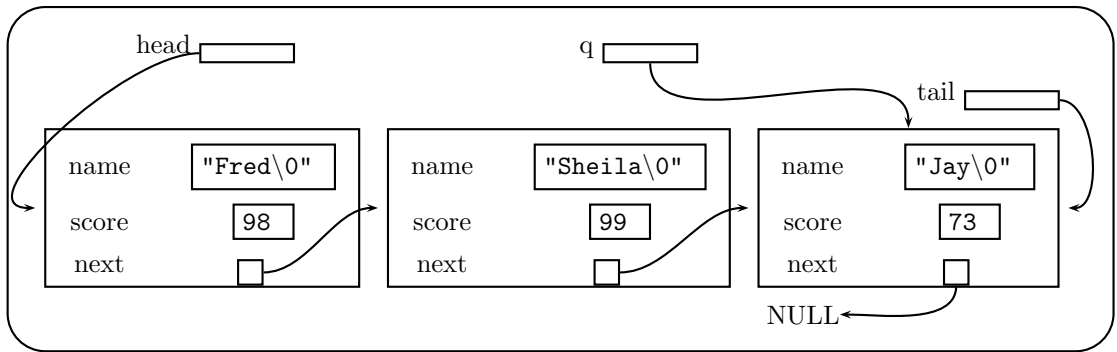
The following code traverses a list, printing out all the values.

```
...
for (Score_S *q=head; q!=NULL; q=q->next)
    cout << q->name<< " " << q->score << endl;

return 0;
}
```

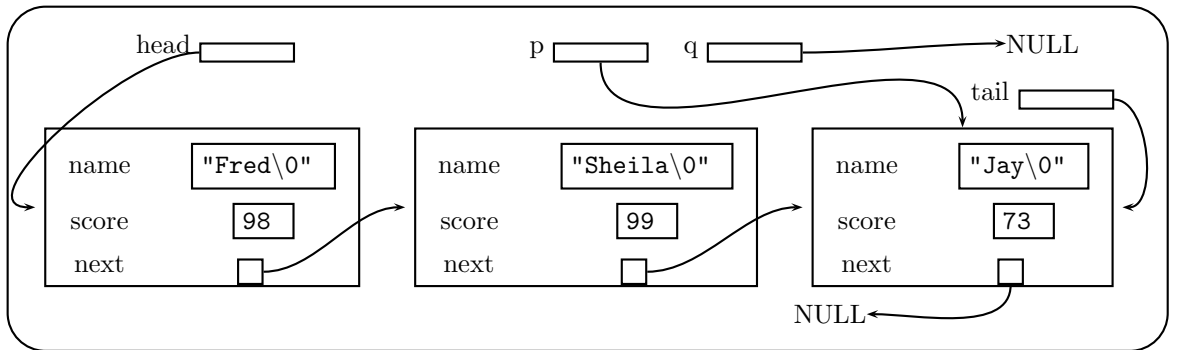
The idea is that the q pointer starts out pointing to where head points, and then points to each of the successive values, finally ending up pointing to NULL, which ends the while loop. The following diagrams illustrate this process:





The entire program and the finished list

Now, for reference, here's the final diagram again, showing the final values of both p and q, and the entire source code listing.



```

// linkedList1.cpp for CISC181 11/16/04
// P. Conrad

#include <iostream>
#include <cstring>
using namespace std;

#define NAMELEN 10

struct Score_S
{
    char name[NAMELEN];
    int score;
    Score_S *next;
};

int main(void)
{
    Score_S *p; // a "working" pointer for allocating new structs
    Score_S *head, *tail; // points to head and tail of linked list

    p = new Score_S;

    strncpy((*p).name, "Fred", NAMELEN);
    (*p).name[NAMELEN-1] = '\0';
    (*p).score = 98;
    (*p).next = NULL;

    head = p;
    tail = p;

    p = new Score_S;
    strncpy(p->name, "Sheila", NAMELEN);
    p->name[NAMELEN-1] = '\0';
    p->score = 99;
    p->next = NULL;

    head -> next = p;
    tail = p;

    p = new Score_S;
    strncpy(p->name, "Jay", NAMELEN);
    p->name[NAMELEN-1] = '\0';
    p->next = NULL;
    p->score = 73;

    tail -> next = p;
    tail = p;

    for (Score_S *q=head; q!=NULL; q=q->next)
        cout << q->name<< " " << q->score << endl;

    return 0;
}

```