Review:
Radix sort
void *Radix*::S*ortMgr*(*istream*& *i*, *ostream*& *o*)
1. Declare list *L*
2. Read the identifiers in stream *i* into list *L*. Use member function *TailInsert* to put the identifiers in the list.
3. Declare integer *p*. Variable *p* is the character position that is used to select the bucket where an identifier is inserted
4. Declare variable *length* and initialize it to one less than the length of the longest identifier in the list.
5. **for** *p=length* **downto** 0 **do** *BucketSort*(*L*,*p*);
6. Write list l output stream *o*.

void *Radix*::*BucketSort*(*List*& *L*, **int** *p*)
1. **while** list *L* is not empty **do**
   1.1. Use member function *HeadRemove* to remove element *e* from the head of list *L*.
   1.2. **if** the length of the identifier in element e is shorter than position p **then**
      1.2.1. use member function *TailInsert* to append element *e* on the list for the alpha bucket
   1.3. **else**
      1.3.1. use member function *TailInsert* to append element *e* on the list for the bucket whose index is given by the integer code the corresponds the p^{th} letter of the identifier in element *e*.
2. Use member function *join* to join all the buckets to list *L*.

Definitions:
1. *head*: The head of the list points to the oldest element on the list.
2. *tail*: The tail of the list points to the newest element on the list.

Requirements:
1. Lists are created by appending elements one at a time onto the tail of the list. Call the operation of appending an element *TailInsert*.
2. The list of unsorted identifiers is processed one element at a time starting with the head and moving to the tail. Elements are removed from the list in the order discussed. Call the operation *HeadRemove*.
3. The lists that form the buckets are joined. The composite list replaces the original list that was sprayed to the buckets and is now reordered by joining the lists together. Call the operation *Join*.

Element	
next	*id*
*Element****	**char****

Figure 1. Diagram of an *element*.

```
struct Element {
    Element* next;
    char* id;
    Element(char* k):next(0)
    {   id=new char[strlen(k)+1];
        strcpy(id,k);
    }
    ~Element() { if (id) delete[] id; }
}
```

Figure 2. C++ Declaration for an *Element*.

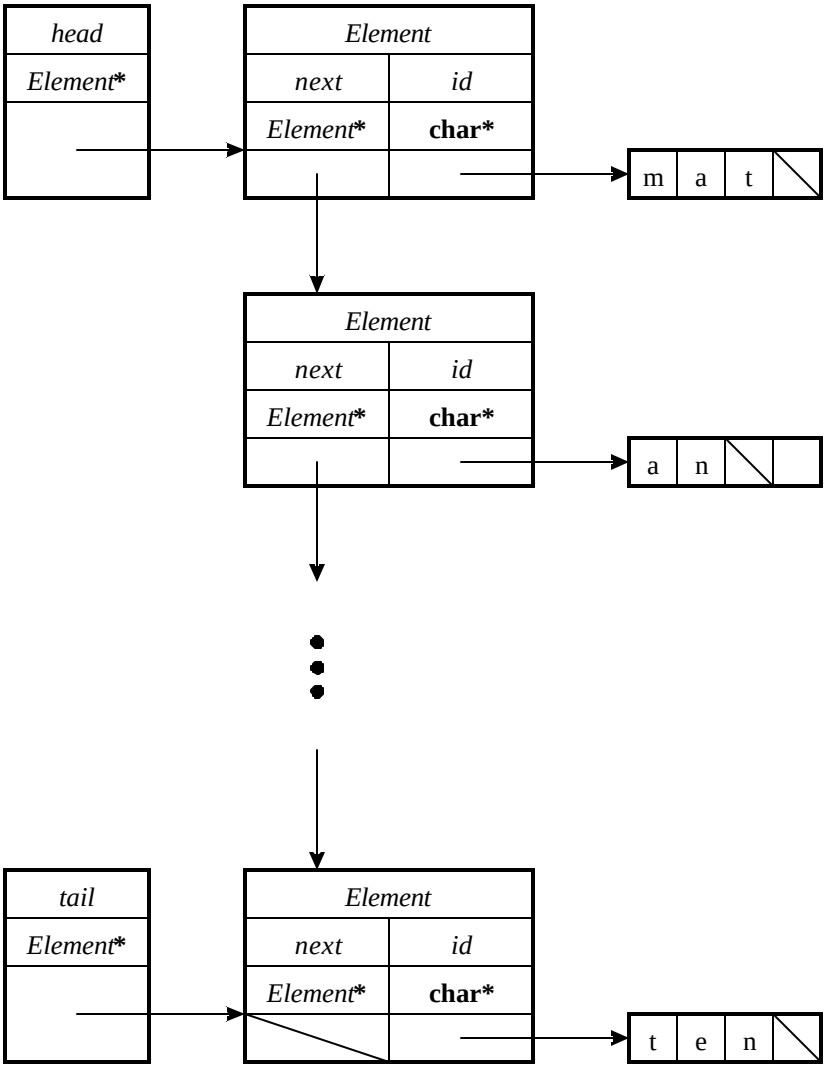

Figure 3. Diagram of a *list*.

```
class List {
    Element* head;
    Element* tail;

    void Kill(Element* e);
public:
    List();
    ~List();
    bool  IsEmpty(void);
    Element* HeadRemove(void);
    void TailInsert(Element* e);
    int Longest(void);
    void Scan(istream& i);
    void Print(ostream& o, char* title,int id);
    void Join(List& L);
};
```

Figure 4. class *list*.

Member	Description
Element head*	Member *head* points to the oldest element on the list. Member head points to the first element put on the list. Elements are linked from head to tail and terminated at the tail. The pointer to the next element in the element at the tail is NULL.
Element tail*	Member *tail* points to the newest element on the list. Member *tail* points to the last element put on the list.
void *Kill(elment* e)*	Member function is called by the destructor *~List*(). Member function *Kill* deletes all elements that remain on the list. As elements are removed, the destructor for type element is called. Storage for the identifiers in the element is reclaimed.
List()	Member function *List()* is the constructor. The list is initialized by assigning NULL values to members *head* and *tail*.
~List()	Member function *~List()* is the destructor. This list is destroyed by reclaiming storage for elements and their identifiers. The destructor *~List()* calls function *Kill* to reclaim storage for the elements on this list. Member function *kill*, in turn, implicitly calls the destructor *~Element()* to reclaim storage for identifiers.
bool *IsEmpty(void)*	Member function *IsEmpty* determines if this list is empty. If the list is empty member function *empty* returns 1 (*true*), otherwise it returns a 0 (*false*).
Element HeadRemove(***void***)*	Member function *HeadRemove* removes the oldest element from this list. A pointer to the element removed is returned to the caller.
void *TailInsert(Element* e)*	Member function *TailInsert* appends the element referenced by parameter *e* to the tail of this list.
int *Longest(***void***)*	Member function *Longest* returns the length of the longest identifier in an element on this list.
void *Scan(istream& i)*	Member function *Scan* reads input file stream *i*. Identifiers in stream *i* are separated by white space. Identifiers in stream *i* are put on the list.
void *Print* *(ostream& o* *,* **char** ** title* *,***int** *id* *)*	Member function *print* formats and writes identifiers on the list in output file stream *o*. A *title* and an integer identifying (*id*) the list are printed before identifiers on the list are printed. The list is printed from head to tail.
void *Join(List& L)*	Member function *Join* appends list *L* to this list. List *L* is emptied.

Table 1. Member descriptions of **class** *List*

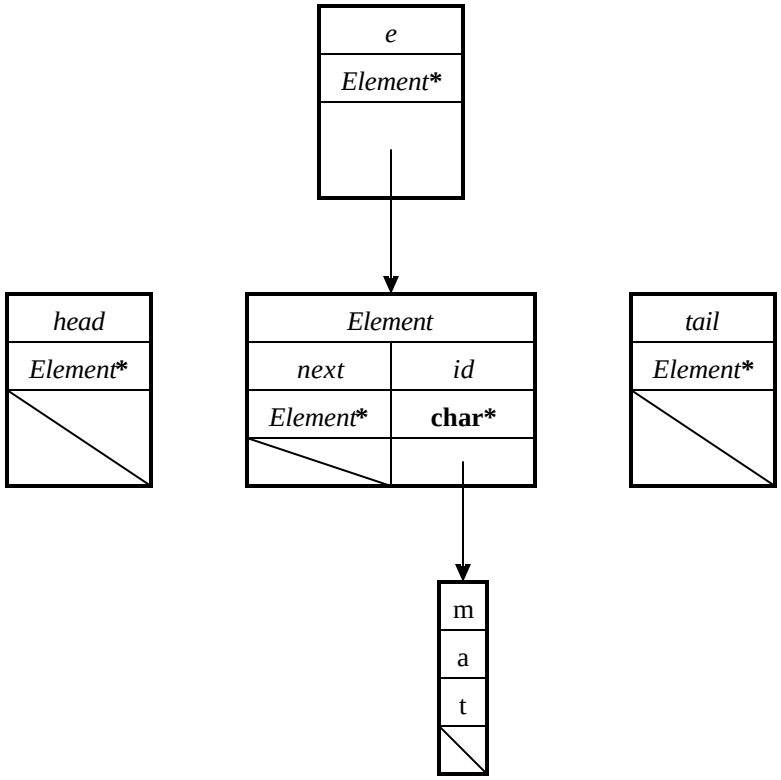

Figure 5. Before inserting an element on an empty list

Notes:
1. The value of the *head* and *tail* pointers is NULL for an empty list.
2. Parameter *e* of member function *TailInsert* references the element to be appended.

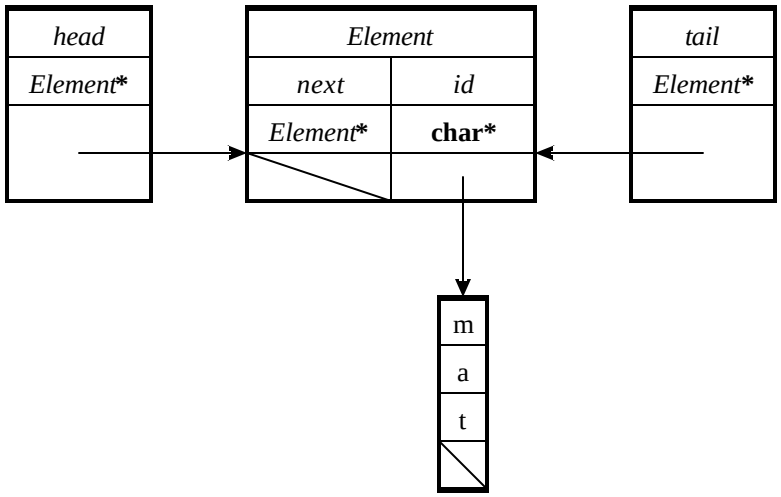

Figure 6. After inserting an element on an empty list

Notes:
1. The *head* and *tail* point to the new element after inserting the first element on a list.
2. Parameter *e* is discarded after function *TailInsert* returns.

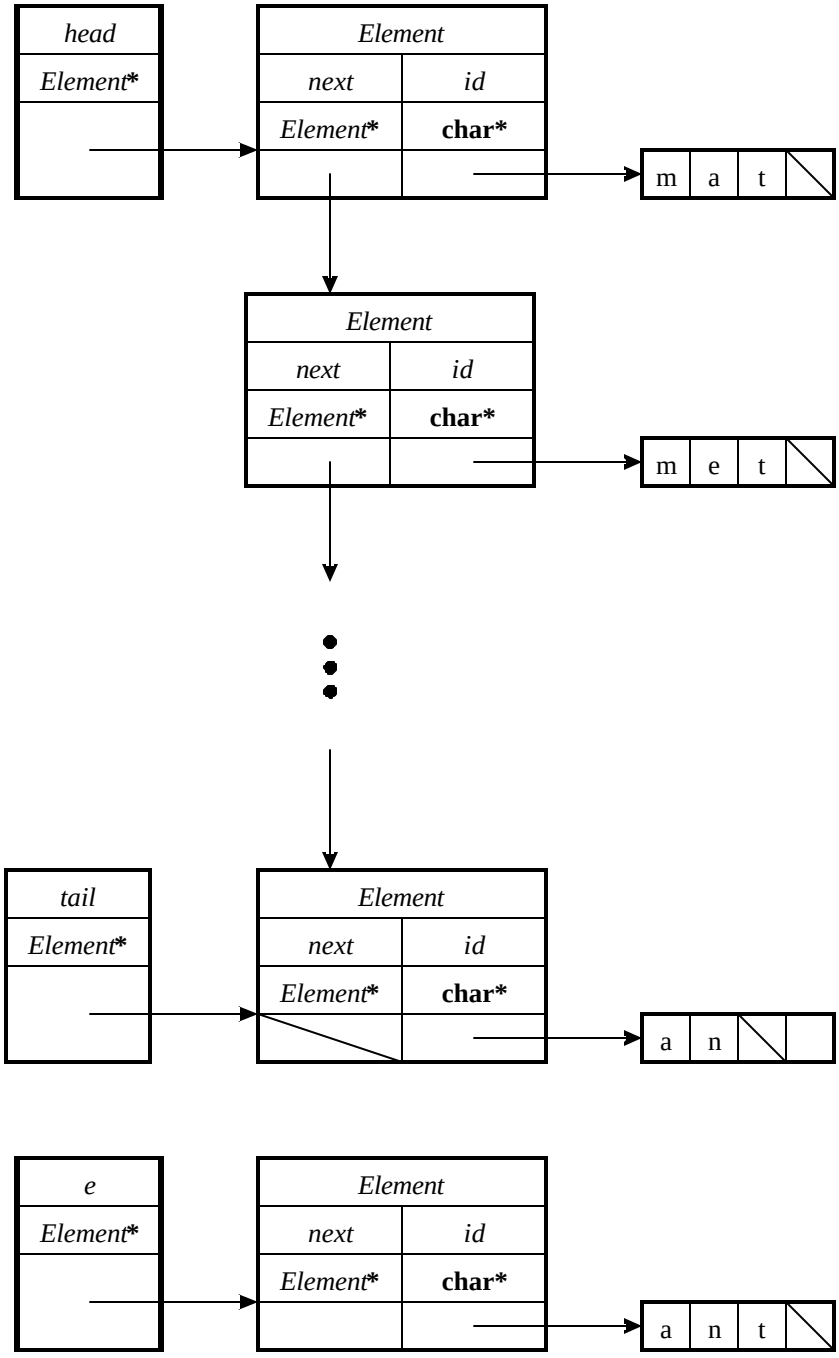

Figure 7. Before inserting an element on a non-empty list

Notes:

1. The non-empty list is illustrated by the list that begins with the element that points to identifier *mat* and ends with the identifier *an*. Member *head* points to the element that references *mat* and member *tail* points to the element that references *an*.
2. Parameter *e* points to the element to be appended to this list.

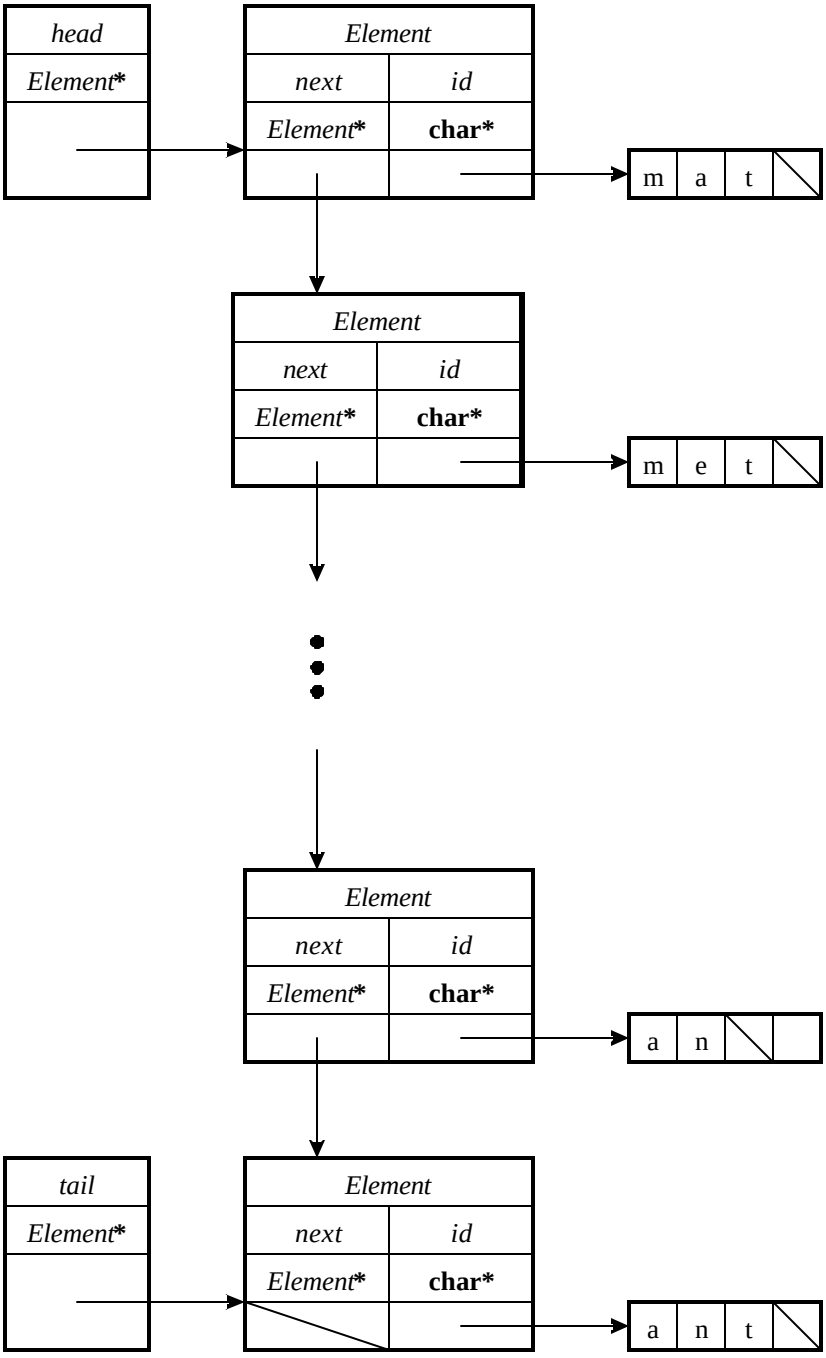

Figure 8. After inserting an element on a non-empty list

Notes:

1. A pointer to the new element at the tail of the list is assigned to member *next* in the penultimate element.
2. A pointer to the new element at the tail of the list is assigned to member *tail*.
3. The list is terminated by assigning a NULL pointer to member *next* in the element at the tail of the list.

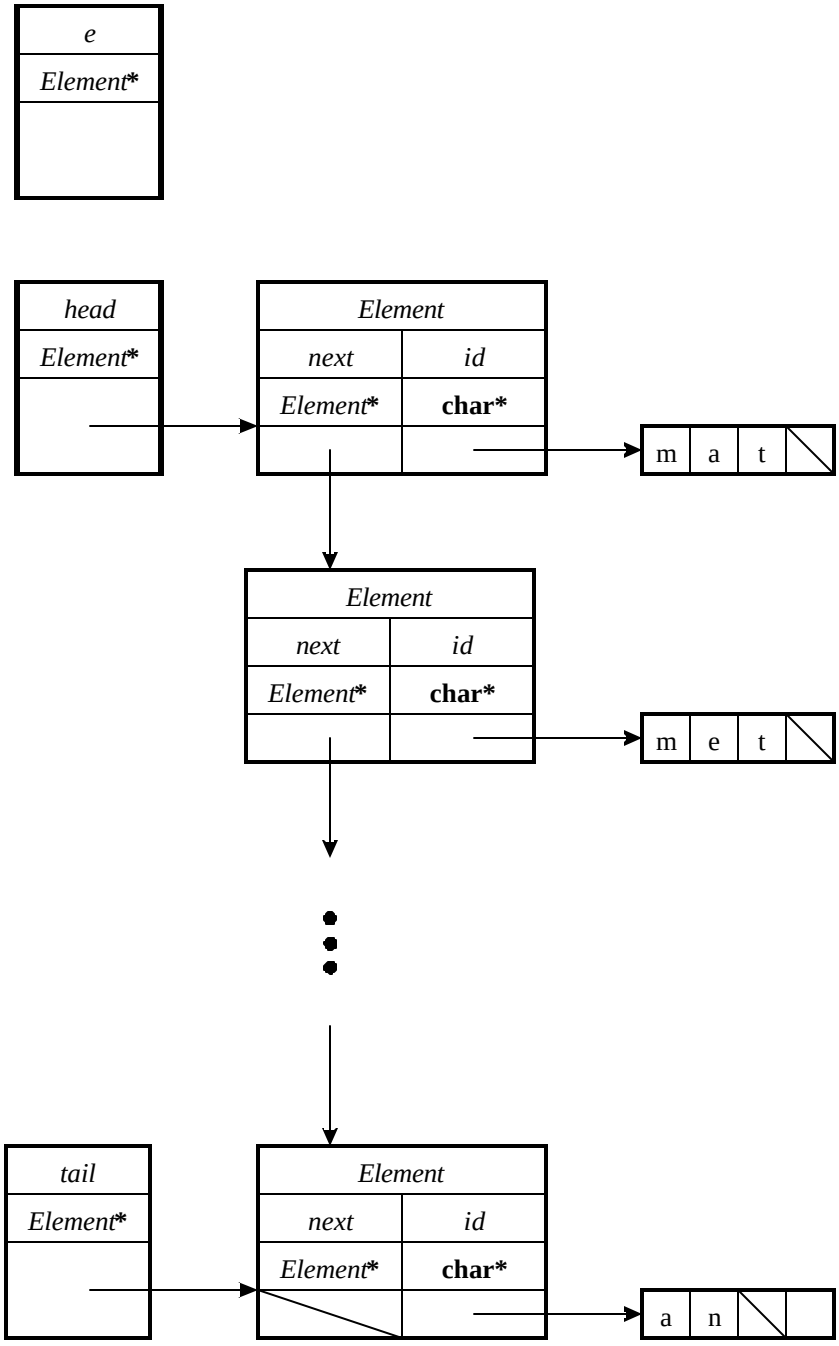

Figure 9. Before removing an element on a list having more than one element

Notes:

1. A pointer to the element at the head of the list will be assigned to local variable *e* and returned to the caller.

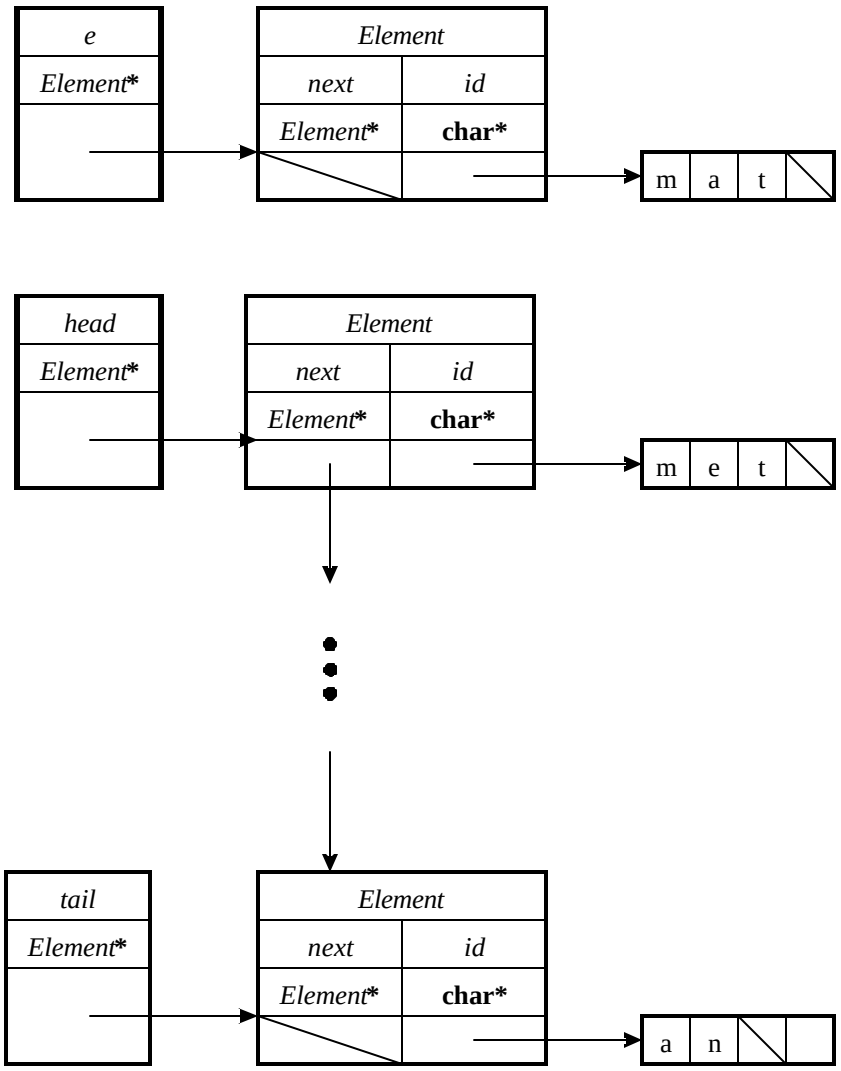

Figure 10. After removing an element on a list having more than one element

Notes:

1. A pointer to the succeeding element is assigned to member head.
2. A pointer to the element that formerly headed the list is assigned to local variable e and returned to the caller.
3. A NULL-value is assigned to member *e->next*.

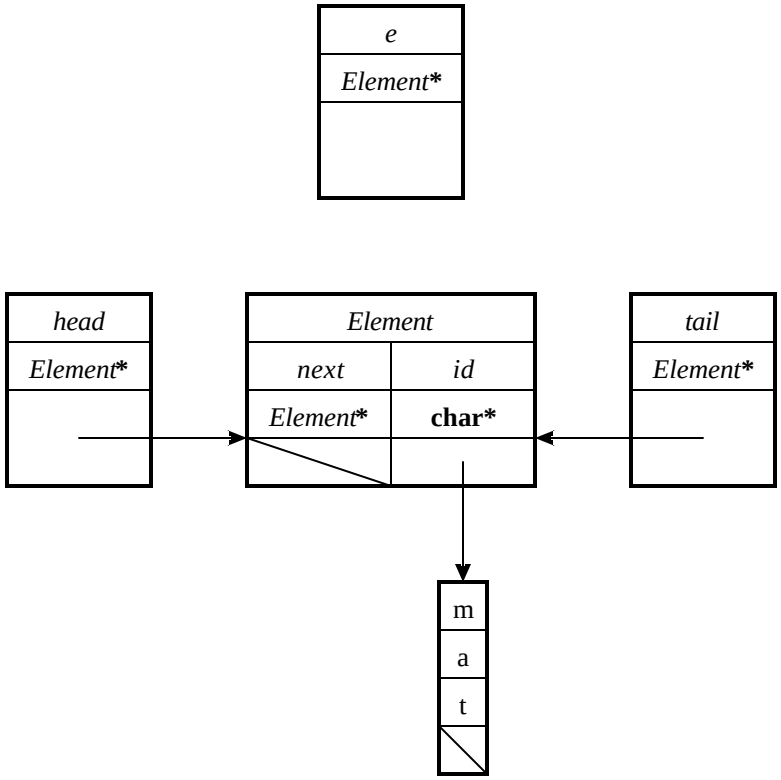

Figure 11. Before removing an element on a list having exactly one element

Notes:

1. A pointer to the last remaining element will be assigned to local variable *e* and returned to the caller.
2. Note that both the *head* and *tail* point to the last remaining element.

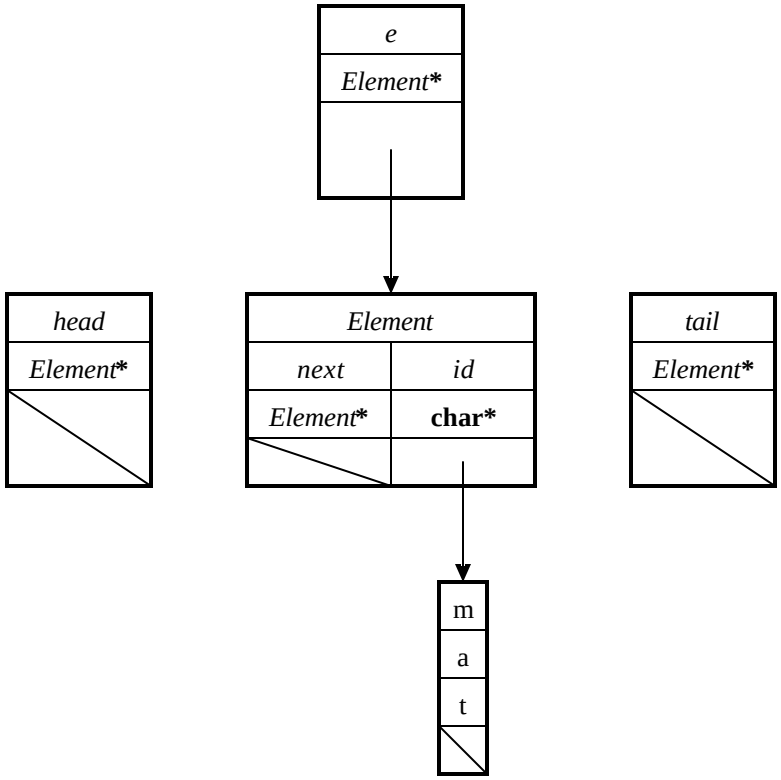

Figure 11. After removing an element on a list having exactly one element

Notes:

1. A NULL value is assigned to member head by copying the terminating NULL value from the last element on the list.
2. A NULL value must be explicitly assigned to member tail.

void *List*::*Join*(*list*& *l*)

1. **if** both lists are empty **then** return to the caller.
2. **if this** list is not empty and list *l* is empty **then** return to the caller
3. **if** this list is empty and list l is not empty **then**
   3.1. Assign the values of members *head* and *tail* in list *l* to **this** list.
   3.2. Assign null values to members *head* and *tail* in list *l*.
   3.3. return to the caller.
4. **if this** list is not empty and list *l* is not empty then append list *l* to **this** list by
   4.1. Assign a pointer to the head of list *l* to the element at the tail of **this** list.
   4.2. Assign a pointer to the tail of list *l* to the tail of **this** list.
   4.3. Assign null values to members *head* and *tail* in list *l*.
   4.4. return to the caller.

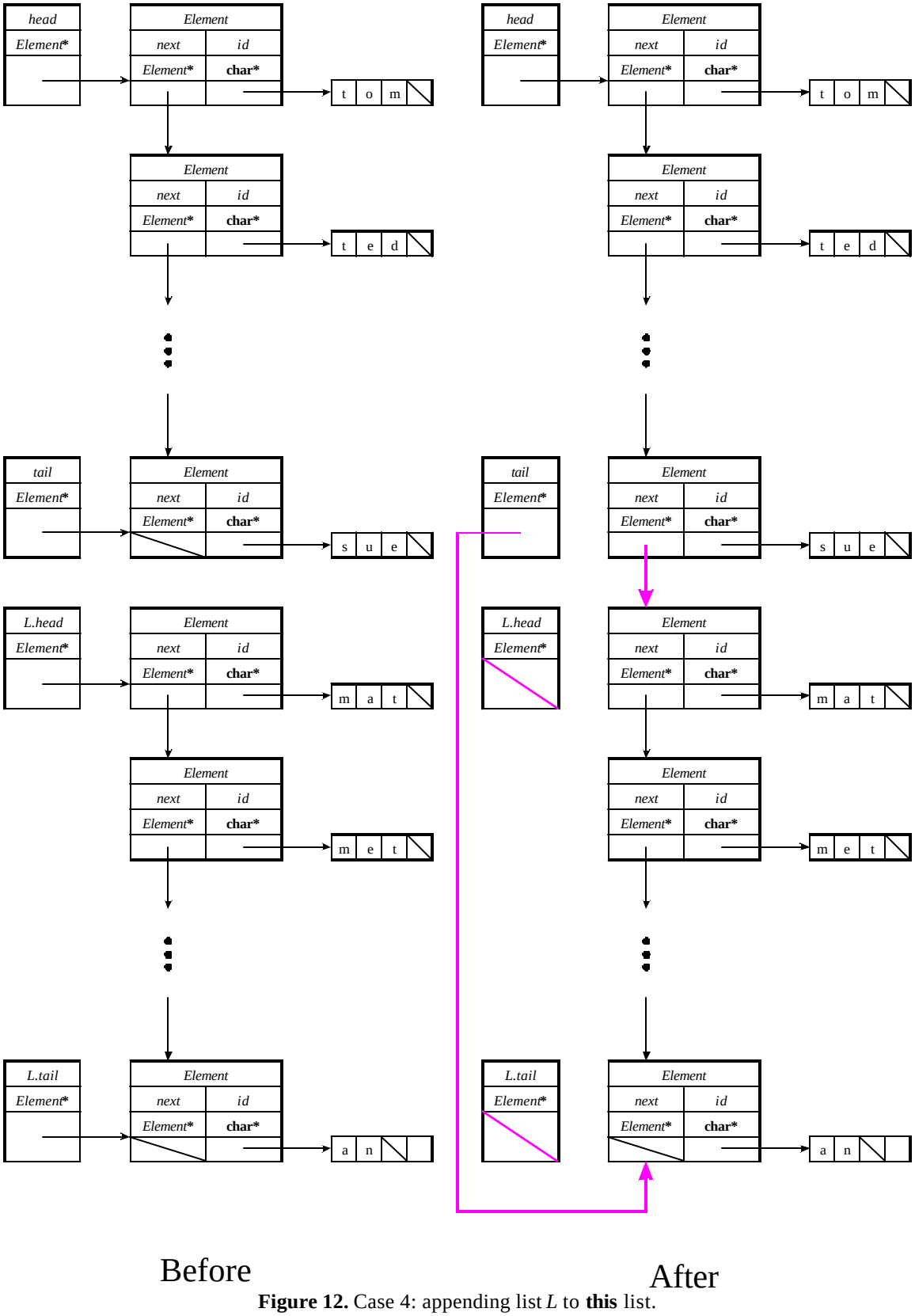

Figure 12. Case 4: appending list *L* to **this** list.