class *Element* definition and implementation

```
struct Element {
    Element* next;                      //Points to the next Element (closer to the tail)
    char* id;                           //Points to the identifier referenced by this
                                        //Element
    Element(char* k);                   //Element constructor
    ~Element();                         //Element destructor
    void Print(ostream& o);             //Prints the identifier
    int Length(void);                   //Retains the length of identifier
};
```

class *List* definition and implementation.

```
class List {
    Element* head;                      //Points to the head of the list – the oldest element
    Element* tail;                      //Points to the tail of the list – the newest element
    Element* cursor;                    /Used to traverse the list (First, Next, Current)
    int longest;                        //Recor ds the length of the longest identifier
                                        //Initially 0, assigned in function Scan

    void Kill(Element* e);              //Serial killer, reclaims storage for all elements on
                                        //the list
public:
    struct ListException {
        ListException(char* m)
        {   cout << endl;
            cout << "I am the list and I am " << m << ".";
            cout << endl;
        }
    };
    List();                             //Constructor
    ~List();                            //Destructor
    void TailInsert(Element* e);        //Appends elements on the tail of the list
    Element* HeadRemove(void);          /Removes elements from the head of the list
    bool IsEmpty(void);                 //Determines if the list is empty
    void Scan(istream& i);              //Inserts elements from stream i
    void Print(ostream& o,char* title);  /Prints elements on the list
    void First(void);                   //Positions the cursor on the head element
    void Next(void);                    //Moves the cursor to the next element.  Next
                                        //means closer to the tail
    bool Eol(void);                     //Determines if the cursor is past the tail element.
                                        //Determines if the cursor is at the End Of List
    Element* Current(void);             //Returns a pointer to the current element.
                                        //Returns a pointer to the element referenced by
                                        //the cursor
    int Longest(voi d);                 //Returns the length of the longest identifier
    void Join(List& L);                 //Joins List L to this list
};
```

1. The **constructor** for class *List*.
   *List***::***List***():***head***(0),***tail***(0),***cursor***(0),***longest***(0){}**
2. The **destructor** for class *List*.
   *List***::~***List***(){***Kill***(***head***);}**

```
void List::Kill(Element* e)
{   while (e) {                          //The list is terminated with a null-pointer
                                         //Remove all elements having a valid pointer
        Element* p=e;                    //The current element e becomes the previous p
        e=e->next;                       //Go on to the next element
        delete p;                        //Remove the current/previous element
    }
}
```

3. Function *TailInsert*.

```
void List::TailInsert(Element* e)
{
    if (head) {             //Second and subsequent elements
        tail->next=e;
    } else {                //First element on the list
        head=e;
    }
    tail=e;
    e->next=0;              //Terminate the list
}
```

4. Function *HeadRemove*.

```
Element* List::HeadRemove(void)
{   if (IsEmpty()) throw ListException("empty");
    Element* e=head;
    if (head==tail) {
        head=tail=0;
    } else {
        head=head->next;
    }
    return e;
}
```

5. Function *Scan*.

```
void List::Scan(istream& i)
{   for (;;) {
        char k[255];
        i >> k;
        if (i.eof()) break;
        if (strlen(k)>longest) longest=strlen(k);
        Element* e=new Element(k);
        TailInsert(e);
    }
}
```

6. Function *Print*.
```
void List::Print(ostream& o,char* title,int pass)
{   o << endl << title;
    if (pass>=0) o << " " << pass;
    for (First();!IsEol();Next()) {
        Element* e=Current();
        o << endl;
        if (e) e->Print(o); else o << "nil";
    }
    o << endl;
}
```
7. Function *First*.
```
void List::First(void){cursor=head;}
```
8. Function *IsEol*.
```
bool List::IsEol(void){return cursor==0;}
```
9. Function *Next*.
```
void List::Next(void){if (cursor) cursor=cursor->next;}
```
10. Function *Current*.
```
Element* List::Current(void){return cursor;}
```
11. Function *Join*.
```
void List::Join(List& L)
{   if ( L.IsEmpty() &&  IsEmpty()) return;
    if ( L.IsEmpty() && !IsEmpty()) return;
    if (!L.IsEmpty() &&  IsEmpty()) {
        head=L.head; tail=L.tail; cursor=L.cursor; longest=L.longest;
        L.head=L.tail=L.cursor=0;L.longest=0;
    }
    if (!L.IsEmpty() && !IsEmpty()) {
        tail->next=L.head;
        tail=L.tail;
        L.head=L.tail=L.cursor=0;L.longest=0;
    }
}
```
12. Function *IsEmpty*.
```
bool List::IsEmpty(void){return head==0;}
```