C++
Raymond Klefstad, Ph.D.
Linked Lists and Recursion

Linked Lists

- more flexible than an array
- should only be used as implementation of classes
- empty list is null
- new elements are best added to front (easiest and most efficient)
- a linked list is often traversed with a for loop

Extended Example

- low-level definition for a linked-list of characters

```
struct ListNode
{
  char info;
  ListNode * next;
  ListNode( char newInfo, ListNode * newNext )
    : info( newInfo ), next( newNext )
  {
  }
};
```

- Members could be called **data** and **link**, but no standard

---

simple class implemented as a linked-list of chars

```
class CharList
{
private:
  ListNode * head;
public:
  CharList()
    : head( NULL )
  {
  }
  void enter( char c )
  {
    head = new ListNode( c, head );
  }
  bool find( char c )
  {
    for ( ListNode * p = head; p !=NULL; p = p->next )
      if ( p->info == c )
        return true;
    return false;
  }
```

```
  void print( ostream & out )
  {
    for ( ListNode * p = head; p != 0; p = p->next )
      out << p->info << ' ';
  }
  ~CharList()
  {
    ListNode * temp;
    for ( ListNode * p = head; p != 0; )
    {
      temp = p;
      p = p->next;
      delete temp;
    }
  }
};
```

---

more example

```
ostream & operator << ( ostream & out, CharList & l )
{
  l.print( out );
  return out;
}
int main()
{
  CharList l;
  for ( char c = 'A'; c <= 'Z'; ++c )
    l.enter( c );
  if ( l.find( 'K' ) )
    cout << "'K' is there\n";
  cout << l << endl;
}
```

Recursion

- recursion occurs when a function calls itself
- similar to an inductive proof
  - must handle base cases (terminal cases)
  - then handle inductive case (recursive case)
- summing up the first n integers

```
int sumN(int n)
{
  if ( n == 0 )
    return 0;
  else
    return n + sumN( n - 1 );
}
```

- ---

using the conditional expression

```
int sumN(int n)
{
  return n == 0 ? 0 : n + sumN( n - 1 );
}
```

- factorial

```
int factorial( int n )
{
  return n == 0 ? 1 : n * factorial( n - 1 );
}
```

Linked List Processing

- type IntListNode

```
class IntListNode
{
public:
  int info;
  IntListNode * next;
  IntListNode( int newInfo, IntListNode * newNext )
    : info( newInfo ), next( newNext )
  {
  }
};
```

- ---

Linked List Processing functions (not methods of any class)

```
int length( IntListNode * l )
{
}
IntListNode * find( int i, IntListNode * l )
{
}
IntListNode * copy( IntListNode * l )
{
}
```

- continued

```
IntListNode * append( IntListNode * l1, IntListNode * l2 )
{
}
IntListNode * remove( int i, IntListNode * l )
{
}
void free( IntListNode * l )
{
}
IntListNode * reverse( IntListNode * l )
```

```
{
}
```

Another Extended example

- low-level linked-list of int

```
class IntListNode
{
public:
  int info;
  IntListNode * next;
  ListNode( const int newInfo, IntListNode * newNext = 0 )
    : info( newInfo ), next( newNext )
  {
  }
  static int length( IntListNode * l )
  {
    return l == 0 ? 0 : 1 + length( l->next );
  }
  static int pop( IntListNode * & l )
  {
    int v = l->info;
    IntListNode * t = l;
    l = l->next;
    delete t;
    return v;
  }
  static void push( int e, IntListNode * & l )
  {
     l = makeNode( e, l );
  }
  static IntListNode * makeNode( int e, IntListNode * & l )
  {
     return new IntListNode( e, l );
  }
  static ListNode * copy( IntListNode * l )
  {
     return l == 0 ? 0: new IntListNode( l->info, copy( l->next ) );
  }
```

- ---

continued

```
  static void attach( int e, IntListNode * & l )
  {
    if ( l == 0 )
      l = new IntListNode( e );
    else
      attach( e, l->next );
  }
```

```
  static IntListNode * append( IntListNode * l1, IntListNode * l2 )
  {
    return l1 == 0 ? copy( l2 )
      : new IntListNode( l1->info, append( l1->next, l2 ) );
  }
  static bool equals(IntListNode * l1, IntListNode * l2)
  {
    return l1 == 0 || l2 == 0 ? l1 == l2
      : l1->info == l2->info && equals( l1->next, l2->next );
  }
  static IntListNode * find( int i, IntListNode * l )
  {
    return l == 0 ? 0 : l->info == i ? l : find( i, l->next );
  }
  static IntListNode * reverse( IntListNode * l )
  {
    IntListNode * result = 0;
    for ( IntListNode * p = l; p != 0; p = p->next )
      result = new IntListNode( p->info, result );
    return result;
  }
  static void deleteList( IntListNode * l )
  {
    if ( l != NULL )
    {
      deleteList( l->next );
      delete l;
    }
  }
};
```

- ---

Higher-level int list class

```
class IntList
{
private:
  IntListNode * head;
public:
  IntList();
  IntList( const IntList & l );
  int findIndexOf( int e );
  void insert( int e );
  int length();
  int & operator [] ( const int i );
  bool operator == ( const IntList & l );
  IntList & operator = ( const IntList & l );
  ~IntList();
};
```

```
ostream & operator << ( ostream & out, const IntListNode * & l );
```