**Recursion and Linked Lists With Recursion**

|  |  |
| --- | --- |
| **Recursion**  A recursive algorithm uses recursion (a routine calling itself) to solve the problem. Some problems can be solved easily using recursion or iteration.  It is extremely important to determine the termination conditions for a recursive algorithm. For fib(n), the termination conditions are when n = 0 or n = 1. | **Example#1: recursive Fibonacci Number**  definition  F(0) = 0  F(1) = 1  F(n) = F(n-1)+F(n-2)  The sequence is  0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597, ...  printf("fib(4) is %d\n", fib(4));  // recursive function in C  int fib(int n)  {  switch (n)  {  case 0:  case 1:  return n;  default:  return fib(n-1) + fib(n-2);  }  } |
| **Show how to trace recursion.** |  |
| **Example Linked List** | **Example #2: example linked list** |
| **Linked Lists and Recursion**  Suppose we had a singly linked list that we want to traverse to print the info for each node. What would be the termination condition for printLL(Node \*p)? | **Example#3: printLL prints the iInfo in each node.**  void printLL(Node \*p)  {  if (p == NULL)  return;  printf("%d\n", p->iInfo);  printLL(p->pNext);  } |
| **Show a trace of printLL** |  |
| **Exercise**  Show code for the function countLL which is initially passed a pointer to the first node of a linked list. countLL should return a count of the nodes in the list.  Termination condition(s) / special cases?  1. Empty list of end of list  // initial call  iCount = countLL(pHead); | **Example#4: countLL recursively counts the number of nodes in a linked list**  int countLL(Node \*p)  {  if (p == NULL)  return 0;  return 1 + countLL(p->pNext);  } |
| **Show a trace of countLL** |  |
| **Exercise**  Show code for sumLL(Node \*p) using recursion. It should return the sum of the iInfo values for the entire list.  Termination condition(s) / special cases?  1. **Empty** list or **End of list**  // initial call  iSum = sumLL(pHead); | **Example#5: sumLL recursively totals the iInfo values in each node.**  int sumLL(Node \*p)  {  if (p == NULL)  return 0;  return p->iInfo + sumLL (p->pNext);  } |
| **Show a trace of sumLL** |  |
| **How would you print the contents of a linked list in reverse order?**  For the linked list of example #3, we want to print  30  20  10  The example will show how we recursively print the contents in reverse order. We could easily do printLL, countLL, and sumLL using iterative algorithms. Think about doing reversePrintLL iteratively. | **Example#6: reversePrintLL recursively prints the iInfo from each node in the reverse order.**  void reversePrintLL(Node \*p)  {  if (p == NULL)  return;  reversePrintLL(p->pNext);  printf("%d\n", p->iInfo);  } |
| **Show a trace of reversePrintLL** |  |
|  |  |