More Recursion Examples

Quick Review

- What is *recursion*?
  Function that uses itself.
- Why is it so useful?
  Natural formulation for many problems.
- How is it like induction?
  Base case and a general case.
- When should we use recursion?
  When it works.
  How do we know it will work?
  Works (time and memory)

1 More Examples of Recursion

As we have seen, recursion can be used to solve many different types of problems.

The following examples illustrate a few more using linked lists.

1.1 Example: Printing a list in reverse

```
1 void PrintListInReverse( NodePtr h )
2 {
3     if( h != NULL )
4     {
5         PrintListInReverse( h->next );
6         cout << " " << h->info;
7     }
8 }
```

Output:

```
Contents of list1:
{ 1  3  6  9 }
Reverse of list1:
{ 9  6  3  1 }
```

1.2 Example: Adding to a list recursively

```
1  struct node
2  {
3      int info;
4      node *next;
5  };
6
7  typedef struct node * NodePtr;
8
9
10 NodePtr head = NULL;
```

Pass by Reference

```
1  void AddNodeRecursive( NodePtr& h, int x )
2  {
3      if( h != NULL )
4      {
5          AddNodeRecursive( h->next, x );
6      }
7      else
8      {
9          NodePtr n;
10
11         n = new node;
12         n->info = x;
13         n->next = NULL;
14
15         h = n;
16     }
17 }
```

Usage:

```
AddNodeRecursive( head, 3 );
```

Pass by Pointer?

```
1  void AddNodeRecursive2( NodePtr* h, int x )
2  {
3      if( *h != NULL )
4      {
5          AddNodeRecursive2( (*h)->next, x );
6      }
7      else
8      {
9          NodePtr n;
10
11         n = new node;          // Allocate
12         n->info = x;
13         n->next = NULL;        // Initialize
14
15         *h = n;
16     }
17 }
```

Usage:

```
AddNodeRecursive2( &head, 3 );
```

Note the differences between this function and the previous function!

Is `AddNodeRecursive2( (*h)->next, x );` correct?

Should it be: `AddNodeRecursive2( &((*h)->next), x );`?

Compiling the code using:

```
AddNodeRecursive2( (*h)->next, x );
```

generates the following error message:

```
LL_RecursiveTest.cpp: In function void AddNodeRecursive2(node**, int):
LL_RecursiveTest.cpp:70: error: cannot convert node* to node** for argument 1
to void AddNodeRecursive2(node**, int)
```

Modified code:

```
1  void AddNodeRecursive2( NodePtr* h, int x )
2  {
3      if( *h != NULL )
4      {
5          AddNodeRecursive2( &((*h)->next), x );
6          //AddNodeRecursive2( (*h)->next, x );    // Fail
7      }
8      else
9      {
10         NodePtr n;
11
12         n = new node;
13         n->info = x;
14         n->next = NULL;
15
16         *h = n;
17     }
18 }
```

1.3 Example: Deleting the last node recursively

```
1  void DeleteLastNode( NodePtr& h )
2  {
3      if( h != NULL )              // Empty?
4      {
5          if( h->next != NULL )    // More nodes?
6          {
7              DeleteLastNode( h->next );
8          }
9          else
10         {
11             NodePtr  p = h;      // Auxiliary pointer
12             p->next = NULL;
13             h = NULL;
14             delete p;
15         }
16     }
17 }
```

Can the previous functions be modified to insert/delete nodes in a sorted list?

1.4 Example: Tracing recursive functions

Tracing a recursive function is a lot like trying to find where an error occurs in code—it often requires a little work. Adding a few lines of code in a systematic manner simplifies the task.

Consider the following function that sums the numbers from 1 to n:

```
1  int sumN( int n )
2  {
3      int  sum;
4
5      if( n == 1 )
6          sum = n;
7      else
8      {
9          sum = n + sumN( n-1 );
10     }
11
12     return sum;
13 }
```

```
1  /*  sum.cpp
2
3      Summation using recursion.
4   */
5
6  #include <iostream>
7
8  using namespace std;
9
10 #define TRACE_FUNCTION  1
11
12 int gDepth = 0;
13
14 int sumN( int n );
15 void Indent();
16
17 int main()
18 {
19     int  iSum;
20
21     iSum = sumN( 5 );
22     cout << "\nmain::iSum: " << iSum << endl;
23
24     return 0;
25 }
```

```
1  int sumN( int n )
2  {
3      int  sum;
4
5  #ifdef TRACE_FUNCTION
6      Indent();
7      cout << "sumN::n: " << n << endl;
8      gDepth++;
9  #endif
10
11     if( n == 1 )
12         sum = n;
13     else
14     {
15         sum = n + sumN( n-1 );
16     }
17
18 #ifdef TRACE_FUNCTION
19     gDepth--;
20     Indent();
21     cout << "sumN::sum: " << sum << endl;
22 #endif
23
24     return sum;
25 }
```

```
1 void Indent()
2 {
3     for( int i = 0 ; i < gDepth ; i++ )
4         cout << '\t';
5     cout << flush;
6 }
```

Output:

```
1  sumN : : n :  5
2        sumN : : n :  4
3              sumN : : n :  3
4                    sumN : : n :  2
5                          sumN : : n :  1
6                          sumN : : sum :  1
7                    sumN : : sum :  3
8              sumN : : sum :  6
9        sumN : : sum :  10
10 sumN : : sum :  15
11
12 main : : iSum :    15
```