ECE 264 Advanced C Programming

Contents

1 Recursion and Linked List

Can we also use recursion for a linked list? We will use `List_search` as an example.

```
int List_search(Node * list, int v)
{
  if (list == 0) { return 0; }
  if ((list -> ln_value) == v) { return 1; }
  return List_search(list -> ln_next, v);
}
```

This function checks whether a list is valid. If this list is invalid (pointing to zero), it is definitely not possible to contain the value `v`. Hence, the function returns 0 to indicate that the value is not in the list. Otherwise, we check whether the **beginning** of the list contains this value. If it does, the function returns 1. If it does not, the function checks the **rest of the list** by using `ln_next`.

If the values are sorted, we can add another condition to detect whether the values in the remaining list are larger than `v`. If so, it is unnecessary to check further and we can conclude that `v` is not in the list.

```
/* values in list are sorted */
int List_search(Node * list, int v)
{
  if (list == 0) { return 0; }
  if ((list -> ln_value) == v) { return 1; }
  if ((list -> ln_value) > v)  { return 0; }
  return List_search(list -> ln_next, v);
}
```

2 Recursion and Iteration

Recursion is often a direct implementation of divide-and-conquer using a **top-down** approach. In many cases, recursions can be converted to iterations (i.e. using `for` or `while`). Iterations usually use a **bottom-up** approach by solving simpler problems first, accumulating the partial solutions, and eventually solving the original problems. Factorial and Fibonacci numbers are two simple examples.

Implement the following function **without** recursion.

$$f(n,k) = \begin{cases} 0 & \text{if } n = 0, \\ 0 & \text{if } k = 0, \\ 1 & \text{if } n = k \neq 0, \\ n \cdot f(n-1,k) & \text{if } n > k > 0, \\ \frac{f(n,k-1)}{k} & \text{otherwise}, \end{cases} \tag{1}$$

Here we use **integer division**: if a and b are two integer and $a < b$, $\frac{a}{b}$ is zero.

```
#include <stdlib.h>
#include <stdio.h>
/* do not use recursion */
int compute(int a, int b)
{
  int ind1;
  int ind2;
  printf("(a, b) = (%d, %d)\n", a, b);
  int result[a + 1][b + 1];
  /* convert
     if ((a == 0) || (b == 0)) { return 0; }
  */
  for (ind2 = 0; ind2 <= b; ind2 ++)
    {
      result[0][ind2] = 0;
      printf("condition 1: result[%d][%d] = %d\n",
             0, ind2, result[0][ind2]);
    }
  printf("\n");
  for (ind1 = 1; ind1 <= a; ind1 ++)
    {
      result[ind1][0] = 0;
      printf("condition 2: result[%d][%d] = %d\n",
             ind1, 0, result[ind1][0]);
```

```
  }
printf("\n");
/* convert
   if (a == b) { return 1; }
*/
for (ind1 = 1; (ind1 <= a) && (ind1 <= b); ind1 ++)
  {
    result[ind1][ind1] = 1;
    printf("condition 3: result[%d][%d] = %d\n",
           ind1, ind1, result[ind1][ind1]);
  }
printf("\n");
/* convert
   if (a > b)  { return a * compute(a - 1, b); }
*/
for (ind2 = 1; ind2 <= b; ind2 ++)
  {
    for (ind1 = ind2; ind1 < a; ind1 ++)
      {
        result[ind1 + 1][ind2] = (ind1 + 1) * result[ind1][ind2];
        printf("condition 4: result[%d][%d] = %d\n",
               ind1 + 1, ind2, result[ind1 + 1][ind2]);
      }
  }
printf("\n");
/* convert
   return compute(a, b - 1) / b;
   a < b must be true
*/
for (ind1 = 1; ind1 <= a; ind1 ++)
  {
    for (ind2 = ind1 + 1; ind2 <= b; ind2 ++)
      {
        result[ind1][ind2] = result[ind1][ind2 - 1] / ind2;
        printf("condition 5: result[%d][%d] = %d\n",
               ind1, ind2, result[ind1][ind2]);
      }
  }
printf("\n");
/* print all results */
for (ind1 = 0; ind1 <= a; ind1 ++)
  {
```

```
      for (ind2 = 0; ind2 <= b; ind2 ++)
        {
          printf("result[%d][%d] = %d\n",
                 ind1, ind2, result[ind1][ind2]);
        }
    }
  printf("\n");
  return result[a][b];
}

int main(int argc, char * argv[])
{
  int val1;
  int val2;
  if (argc < 3)
    {
      printf("need two integers\n");
      return -1;
    }
  val1 = strtol(argv[1], (char **)NULL, 10);
  val2 = strtol(argv[2], (char **)NULL, 10);
  printf("compute(%d,%d) = %d\n", val1, val2, compute(val1, val2));
  return 0;
}
```