ECE 264 Advanced C Programming

2009/03/25

Contents

1 Exam 3 (April 3) ENAD 240

Do **not** come to MSEE B012 for the exam. Please notice the location change.

The third exam is held on April 3 (Friday) in **ENAD 240**. This is an ITaP instruction lab. The computers run Windows XP, different from the computers in MSEE 190. You are encouraged to visit ENAD 240 in advance and check the computers in the room.

This exam can be taken on paper or on-line. The two formats have the same questions. You will receive a paper copy of the questions, regardless of the format you choose. Please choose **only one** format. You do not have to inform the instructor of the format you choose. If you decide to take the exam on paper, the format is the same as the previous two exams. You will have 55 minutes (1:30-2:25PM).

If you decide to take the exam on-line, please enter Blackboard and you will see exam 3. You will have 58 minutes for the on-line exam. The additional 3 minutes allow you to learn the new format. Please remember the time is **electronically enforced** and you must submit your answers within the duration. You are allowed to search the Internet for any help. However, you will not have time to learn new concepts during the exam. Your personal laptop is allowed; please ensure that your laptop's battery is fully charged. You **must** take the on-line exam in ENAD 240. You are **not** allowed to take the on-line exam anywhere else.

You are **not allowed** to use email, messaging, or phone to communicate with anybody during the exam.

2 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);
}
```

3 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.

Quiz and Discussion (need volunteers to share their solutions): 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;
}
```