Foundations of C Programming (Structured Programming)

- Recursion(递归)

Outline

- Recursive call of functions
- Examples
- Recursion and iteration

Dominoes

How do dominoes (多米诺骨牌) work?

Picture resource: http://www.fotosearch.com/photos-images/domino_2.html

Dominoes

- The *n*-th card is pushed by the (*n*-*1*)-th card (recursive 递归 step)
- To make the dominoes work, the *1*st card must be pushed (推倒) first (base case，终止条件)
- This is called recursion (递归)

More Examples of Recursion

- A recursive definition of person's ancestors
  - One‘s parents are one’s ancestors (祖先) (base case).
  - The ancestors of parents are also one's ancestors (recursive step).
- Natural numbers
  - 0 is a natural number (base case).
  - if n is a natural number(自然数), then n+1 is also a natural number (recursive step).

Recursive Functions

- A recursive function
  - calls itself (调用自己，递)
  - uses different parameter values
  - stops calling itself when the base case is met (归)
- Recursive functions are commonly used in the applications in which the solution to a problem can be expressed in terms of successively(连续) applying the same solution to subsets of the problem
- The famous factorial calculation problem
  - `n! = n * (n-1) * (n-2) * ... * 1`
  - $\underbrace{\texttt{n! = n * (n-1)!}}_{\text{recursive step}}$ and $\underbrace{\texttt{1! = 1}}_{\text{base case}}$

Example 1 – Factorial Number

Recursive step: `the result of fac(n) is n * fac(n - 1)`

Base case: `fac(1) is 1`

```
int fac(int n) // Assume n >= 0
{
  int product;

  if(n <= 1)
    return 1;

  product = n * fac(n-1);
  return product;
}
```

Base case

Recursive step

Example 1 – Factorial Number

- Assume in main program, we use fac(3) to call the fac function

```
int fac(int n)
{
   int product;
   if(n <= 1)
     return 1;
    product = n * fac(n-1);
    return product;
}
```

```
int main()
{
   ......
   r = fac(3);
   ......
}
```

```
fac(3):
   3 <= 1 ?
   product = 3 * fac(2)
   return product;
```

```
fac(2):
   2 <= 1 ?
   product = 2 * fac(1)
   return product;
```

```
fac(1):
 1 <= 1?
 return 1;
```

Base Case

Base case is also called stop condition （终止条件）

Example 2 – Number of Zero Digits in an Integer

- Write a recursive function **zeros** that counts the number of zero digits in a non-negative integer. E.g., `zeros(10200)` returns `3`

What is the base case?
How to express the recursive step?

Example 2 – Number of Zero Digits in an Integer

One digit:

0: return 1

others: return 0

More digits:

rightmost digit 0: return 1 + number of zeros in the rest digits

rightmost digit others: return 0 + number of zeros in the rest digits

```
int zeros(int n)
{
  if(n == 0)
    return 1;
  if(n < 10)
    return 0;

  if(n % 10 == 0)
      return 1 + zeros(n / 10);
  else
      return zeros(n / 10);
}
```

Base case (stop conditions)

Recursive step

Example 3 – Fibonacci numbers

- Fibonacci numbers:

  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

  where each number is the sum of the preceding two. Write a recursive function to solve this.

What is the base case?
How to express the recursive step?

Example 3 – Fibonacci numbers

Base case

```
int Fibonacci(int n)
{
  if(n == 0)
    return 0;
  if (n == 1)
    return 1;

  return Fibonacci(n - 2) + Fibonacci(n - 1);
}
```

Recursive step

After Class

- Write a recursive function to determine how many factors m are part of n. For example, if n = 48 and m = 4, then the result is 2 (since 48 = 4 * 4 * 3).

What is the base case?
How to express the recursive step?

Writing a Recursive Function

- Three steps
  - Find the base case
  - Find the recursive step
  - Write the recursive function

Recursive Function

- A recursive solution may be simpler to write (once you get used to the idea) than a non-recursive solution.
- But a recursive solution may not be as efficient as a non-recursive solution of the same problem.
- Each recursion call consumes some memory.

Recursion and Loops

- Recursion is based upon calling the same function successively
- Loop simply `jumps back' to the beginning of the loop
- A function call is often more expensive than a jump

```
int fac(int n)
{
  int j;

  product = 1;
  for (j = 2; j < = n; j++)
    product = product * j;
  return product;
}
```

Use loops to implement factorial number

Summary

- It is important to find the relations in the recursive functions.
- Using recursion can make programming simpler.