Tutorial 04: Recursion

CSCI2520 - DATA STRUCTURES AND APPLICATIONS

TUTOR: ZHENG CHENGUANG

Outlines

1. List
2. Recursion
3. GCD of Two Integers
4. GCD of Several Integers
5. Climbing Stairs

List

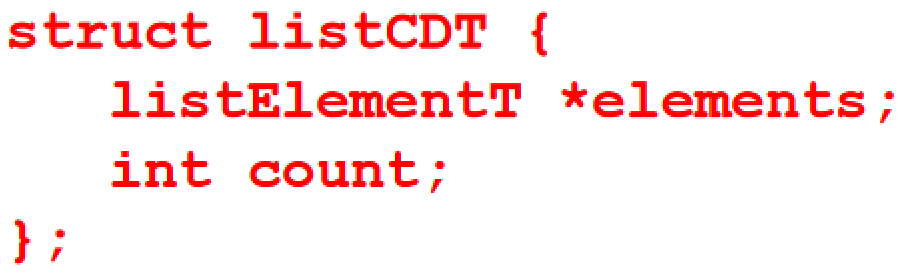
- A list is either an empty list or an element followed by a list.

  - an example of recursive definitions.

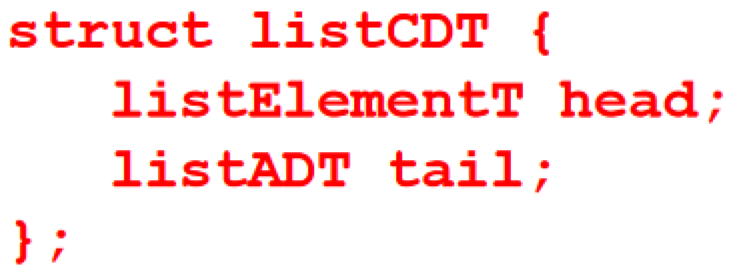
- Fundamental operations include:
  - CreateList: creating a list from a head and a tail.
  - ListHead: obtaining the head of a list.
  - ListTail: obtaining the tail of a list.
- More operations
  - EmptyList: return a new empty list to the user
  - ListIsEmpty: return true if the list is empty

List

- Ways to implement:
  – dynamic array (Ver 1.0)
  – recursive representation(Ver 2.0)



```
struct listCDT {
   listElementT *elements;
   int count;
};
```



```
struct listCDT {
   listElementT head;
   listADT tail;
};
```

- **Question**: what is the difference between the two?

Recursive Functions

- A ***recursive function*** is a function that makes a call to itself.
- Any recursive function will include the following three basic elements.
  - A *test* to stop or continue the recursion.
  - An *end case* that terminates the recursion.
  - A *recursive call* that continues the recursion.

```
int ListLength(listADT list) {
    if (ListIsEmpty(list))
        return 0;
    else
        return 1 + ListLength(ListTail(list));
}
```

A *test* to stop or continue

An *end case* to terminate the recursion

A *recursive call* to continue recursion

Recursion

- ***Recursion*** is a method where the solution to a problem depends on solutions to *smaller instances of the same problem*.
- Recursion usually leads to more *elegant* and *simpler* solutions, although it incurs larger memory and time overhead.
- An important recursive problem-solving skill is ***divide-and-conquer***.
  - *Divide* the problem into smaller pieces.
  - *Tackle* each sub-task either directly or by recursion.
  - *Combine* the solutions of the parts to form the solution of the whole

Greatest Common Divisor

- The greatest common divisor (GCD) of *two or more* integers, when at least one of them is not zero, is the largest positive integer that divides the numbers without a remainder.
- For example, the GCD of 8 and 12 is 4.

Greatest Common Divisor

- Observation: gcd(a, b) = gcd(a, a+b), because
  - A common divisor of a and b is also a common divisor of a and a+b
  - A common divisor of a and a+b is also a common divisor of a and b
- Question 1: how to reduce the problem size recursively?
- Question 2: what is the end case?

Greatest Common Divisor

- End case
  - gcd(a, 0) = a
- Recursive call
  - Assume without loss of generality that a>b
  - Version 1: gcd(a, b) = gcd(b, a-b)
  - Version 2: gcd(a, b) = gcd(b, a%b)
- For example, gcd(48, 18)=gcd(18, 12)=gcd(12, 6)=gcd(6, 0)= 6

Greatest Common Divisor

- Finish the implementation of the following function which calculates the GCD of two positive integers.
- Hint: what if a<b?

```
int GCDOfTwoNum(int a, int b);
```

Solution

```
int GCDOfTwoNum(int a, int b)
{
    if (a < b)
    {
        int temp = a;
        a = b;
        b = temp;
    } // but useless
    if (b == 0) // test
        return a; // end case
    else
        return GCDOfTwoNum(b, a % b); // recursive call
}
```

GCD of Several Integers

- Observation: gcd(a, b, c) = gcd(gcd(a, b), c)

- Question 1: how to reduce the problem size recursively?
- Question 2: what is the end case?

GCD of Several Integers

- Finish the implementation of the following function which calculates the GCD of several positive integers(> 1).

```
int GCDOfList(listADT list);
```

- The definition of listADT is as follows:

```
typedef struct listCDT *listADT;
typedef int listElementT;
listADT EmptyList();
listADT CreateList(listElementT head, listADT tail);
listElementT ListHead(listADT list);
listADT ListTail(listADT list);
int ListIsEmpty(listADT list);
int GCDOfTwoNum(int a, int b)
```

Solution

```
int GCDOfList(listADT list)
{
    if (ListTail(list) == NULL) // test
        return ListHead(list); // end case
    else
    {// recursive call
        int GCDOfTail = GCDOfList(ListTail(list));
        return GCDOfTwoNum(ListHead(list), GCDOfTail);
    }
}
```

Climbing Stairs

- You are climbing a stair case. It takes *n* steps to reach to the top.
- Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?
- **Note:** Given *n* will be a positive integer.
- For example, if n = 3, the answer is 3.

  There are three distinct ways to climb to the top.

  - 1 step + 1 step + 1 step
  - 1 step + 2 steps
  - 2 steps + 1 step

Climbing Stairs

- You are climbing a stair case. It takes *n* steps to reach to the top.
- Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

```
int ClimbingStairs(int n);
```

- Question 1: how to reduce the problem size recursively?
- Question 2: what is the end case?

Solution

```
int ClimbingStairs(int n)
{
    if (n == 1 || n == 2) // test
        return n; // end case
    else
    {// recursive call
        return ClimbingStairs(n - 1) + ClimbingStairs(n - 2);
    }
}
```

Question: What’s the time complexity of this solution?

$O(2^{\mathrm{n}})$(Mathematical Induction)