Data structure and algorithms lab

RECURSION

Lecturers : Cao Tuan Dung

Dept of Software Engineering
Hanoi University of Science and Technology

Today's topics

- Solve problem using recursion
- Write and call recursive functions

Introduction

- In some problems, it may be natural to define the problem in terms of the problem itself.
- A programming technique in which a function calls itself.
- One of the most effective techniques in programming.

Triangular Numbers

- Consider the numbers 1, 3, 6, 10, 15….
- What is so peculiar about them?
- The n^{th} term in the series is obtained by adding n to the previous number.
- Recursion can be used to find the nth term.

Finding nth Term

Using Loop

```
int triangle(int n)
{
   int total = 0;
   while (n > 0)
   {
       total = total + n;
       --n;
    }
    return total;
}
```

Our observation (between loop vs. recursion)

- If a loop is used, the method cycles around the loop n times, adding n to the total the first time, n-1 the second time and so on, down to 1, quitting the loop when n becomes 0.
- If recursion is used, then a base case is used that determines when the recursion ends.

Characteristics of Recursive Methods

- The recursive method calls itself to solve a smaller problem.
- The base case is the smallest problem that the routine solves and the value is returned to the calling method. **(Terminal condition)**
- Calling a method involves certain overhead in transferring the control to the beginning of the method and in storing the information of the return point.
- Memory is used to store all the intermediate arguments and return values on the internal stack.
- The most important advantage is that it simplifies the problem conceptually.

Exercise

- Write a program in C to print first 50 natural numbers using recursion.
- *Expected Output* :
- The natural numbers are : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Solution

```
#include<stdio.h>
int  numPrint(int);
int main()
{
   int n = 1;
         printf("\n\n Recursion : print first 50 natural numbers :\n");
         printf("-----------------------------------------\n");
         printf(" The natural numbers are :");
   numPrint(n);
   printf("\n\n");
   return 0;
}
int numPrint(int n)
{
   if(n<=50)
   {
       printf(" %d ",n);
       numPrint(n+1);
   }
}
```

Exercise 1

- Implement a function that multiply two unsigned integer numbers using recursion.
  - multiply(3,5) return 15

Solution

```
#include <stdio.h>
unsigned int multiply(unsigned int x, unsigned int y)
{
    if (x == 1)     {
        return y; /* Terminating case */
    }
    else if (x > 1)     {
        /* Recursive step */
        // Fill in the code here
    }

    /* Catch scenario when x is zero */
    return 0;
}

int main() {
    printf("3 times 5 is %d", multiply(3, 5));
    return 0;
}
```

Exercise 2

- Write a program in C to calculate the sum of numbers from 1 to n using recursion.

Solution

```
#include<stdio.h>
int sumOfRange(int);
int main() {
   int n1, sum;
   printf(" Recursion : calculate the sum of numbers from 1 to n
:\n");
   printf(" Input the last number of the range starting from 1 : ");
   scanf("%d", &n1);
   sum = sumOfRange(n1);
   printf("\n The sum of numbers from 1 to %d : %d\n\n", n1,
sum);
    return (0);
}
int sumOfRange(int n1) {
   int res;
   //Fill in the code here
   return (res);
}
```

Exercise 3

- Write a program in C to count the digits of a given number using recursion.
- Write a program in C to find the sum of digits of a number using recursion.

Solution

```
int noOfDigits(int n1)
{
    static int ctr=0;
     if(n1!=0)
     {
          ctr++;
         noOfDigits(n1/10);
     }
    return ctr;
}
int DigitSum(int n1)
{
    if(n1 == 0)
        return 0;

    return ((n1 % 10) + DigitSum(n1 / 10));
}
```

Exercise 4

- Write a program in C to Print Factorial and Fibonacci Series of a given number n using recursion.

Expected Output :

- Input number of terms for the Series (< 20) : 10 The Series are : 1 1 2 3 5 8 13 21 34 55

Solution

```
int main() {
   int n = 5;
   int i;
   printf("Factorial of %d: %d\n" , n , factorial(n));
   printf("Fibonacci of %d: " , n);
   for(i = 0;i<n;i++)   printf("%d ",fibonacci(i));
 }
int factorial(int n) {
   if(n == 0) {
      return 1;
   } else {
      return n * factorial(n-1);
   }
}
int fibonacci(int n) {
   if(n == 0) return 0;
   else if(n == 1)      return 1;
   else       return (fibbonacci(n-1) + fibbonacci(n-2));
}
```

Exercise 5

- Write a program in C to Check whether a given String is Palindrome or not.

Solution

```
#include<stdio.h>
void checkPalindrome(char wordPal[], int index)
{
    int len = strlen(wordPal) - (index + 1);
    /* Fill in the code here

    */
}

// in main call checkPalindrome(str, 0);
```

Exercise 6

- Write a program in C to find GCD and LCM of two numbers using recursion.

- Test Data :
  Input 1st number: 10
  Input 2nd number: 50
  Expected Output :
  The GCD of 10 and 50 is: 10
  The LCM of 10 and 50 is: 50

Solution

```
int findGCD(int a,int b)
{
    while(a!=b)
    {
        if(a>b)
            return findGCD(a-b,b);
        else
            return findGCD(a,b-a);
    }
    return a;
}
int lcm(int a, int b)
{
    static int common = 1;
    if (common % a == 0 && common % b == 0)
        return common;

    common++;
    lcm(a, b);
    return common;
}
```

Homework

- Write a program in C to reverse a string using recursion.

- Enter a string to reverse: cprogramming
- The string after reversing is: gnimmargorpc

Exercise 8: Hanoi tower

- Tower of Hanoi is a mathematical puzzle where we have three rods (needles) and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
  - 1) Only one disk can be moved at a time.
  - 2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
  - 3) No disk may be placed on top of a smaller disk.
- Write a program that solves this puzzle.

Example 7: Towers of Hanoi

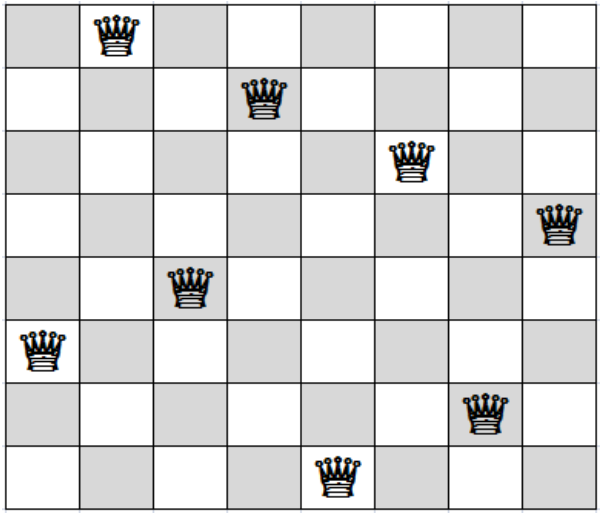
see applet

Solution

```
#include <stdio.h>
#include <string.h>
void towers(int, char, char, char);
int main() {
    int num;
    printf("Enter the number of disks : ");   scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi are :\n");
    towers(num, 'A', 'C', 'B');
    return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
    if (num == 1)    {
        printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
        return;
    }
    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);
}
```

N-Queen problem

- The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.
- Write a program that using recursive backtracking to solve this problem



Backtracking algorithm

1) Start in the leftmost column
2) If all queens are placed
   return true
3) Try all rows in the current column. Do following
   for every tried row.
   a) If the queen can be placed safely in this row
      then mark this [row, column] as part of the
      solution and recursively check if placing
      queen here leads to a solution.
   b) If placing queen in [row, column] leads to a
      solution then return true.
   c) If placing queen doesn't lead to a solution
      then unmark this [row, column] (Backtrack)
      and go to step (a) to try other rows.
3) If all rows have been tried and nothing worked,
   return false to trigger backtracking.

Solution

```
#include <stdio.h>
#define N 4

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    static int k = 1;
    printf("%d-\n",k++);
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
    printf("\n");
}
```

Solution

```
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;
    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;
    /* Check upper diagonal on left side */
    for (i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j])
            return false;

    /* Check lower diagonal on left side */
    for (i=row, j=col; j>=0 && i<N; i++, j--)
        if (board[i][j])
            return false;
      return true;
}
```

Solution

```
bool solveNQUtil(int board[N][N], int col)
{
    /* base case: If all queens are placed then return true */
    if (col == N) {
        printSolution(board); return true;
    }
    /* Consider this column and try placing this queen in all rows one by one */
    bool res = false;
    for (int i = 0; i < N; i++)     {
        /* Check if queen can be placed on board[i][col] */
        if ( isSafe(board, i, col) )  {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;
            // Make result true if any placement is possible
            res = solveNQUtil(board, col + 1) || res;
              /* If placing queen in board[i][col] doesn't lead to a solution, then
            remove queen from board[i][col] */
            board[i][col] = 0; // BACKTRACK
        }
    }
    /* If queen can not be place in any row in this column col then return false */
    return res;
}
```

Solution

```
/* this function prints one of the feasible solutions.*/
void solveNQ()
{
    int board[N][N];
    memset(board, 0, sizeof(board));

    if (solveNQUtil(board, 0) == false)
    {
        printf("Solution does not exist");
        return ;
    }
    return ;
}

// driver program to test above function
int main()
{
    solveNQ();
    return 0;
}
```

Homework

- Write a program that convert a number from decimal system to binary system using recursion.

Homework

- Write a program that find out the first capital character in a string using recursion

Homework

- Print all possible combinations of k elements in a given array of size n. Given an array of size n, generate and print all possible combinations of k elements in array.
  - For example, if input array is {1, 2, 3, 4} and k is 2, then output should be {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4} and {3, 4}.

Solution

```
void printCombination(int arr[], int n, int r) {
    int data[r];
    // Print all combination using temprary array 'data[]'
    combinationUtil(arr, data, 0, n-1, 0, r);
}
void combinationUtil(int arr[], int data[], int start, int end,
                     int index, int r) {
    // Current combination is ready to be printed, print it
    if (index == r)    {
        for (int j=0; j<r; j++) printf("%d ", data[j]);
        printf("\n");
        return;
    }
 /* replace index with all possible elements. The condition "end-i+1 >= r-index"
makes sure that including one element at index will make a combination with
remaining elements at remaining positions */
    for (int i=start; i<=end && end-i+1 >= r-index; i++)
    {
        data[index] = arr[i];
        combinationUtil(arr, data, i+1, end, index+1, r);
        while (arr[i] == arr[i+1])      i++;
    }
}
```

Solution

```
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = sizeof(arr)/sizeof(arr[0]);
    printCombination(arr, n, r);
}
```