BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

First Semester 2011-2012

IS ZC361 Data Structures and Algorithms

Lab Sheet -3 Topic: Recursion to Iteration (using explicit stack)

General Instructions for Programming

1. All inputs to the program must be either (a) command line arguments (b) or read from a file (other than stdin). DO NOT READ anything from stdin and DO NOT USE PROMPTS like “Please enter a number …”.

2. You are required to write the output to a file (other than stdout) and errors if any to a different output channel (stderr or another file).

3. Use standard C coding conventions for multi-file programs. Separate the following: interfaces of functions (use a “.h” file), data type definitions (use another “.h” file), ADT / algorithm implementation (use a “.c” file), and driver/test code (use another ”.c” code). In general, each module has to be written in **separate** c files.

Problem Statement:

In this lab, you will convert the following few recursive problem into iterative by the explicit use of Stack ADT to simulate the role of call stack.

The problem to solve is a simplified version of 'Subset Sum Problem'. The objective of 'Subset Sum Problem' is is to find if any non empty subset from the given set of integers sums to s. For example, given the set { −7, −3, −2, 5, 8}, the answer is yes because the subset { −3, −2, 8} sums to 3.

How to solve:

Consider an arbitrary element x ∈ X. There are two possibilities to consider.

1) There is a subset of X that includes x and sums to T . Equivalently, there must be a subset of X - {x} that sums to T − x.

2) There is a subset of X that excludes x and sums to T . Equivalently, there must be a subset of

X - {x} that sums to T .

So we can solve SUBSET_ SUM(X, T) by reducing it to two simpler instances:

1) SUBSET_SUM(X-{x},T − x) and
2) SUBSET_SUM(X - {x}, T)

This implies the following recursive algorithm:

Algorithm SUBSET_ SUM(X[1 .. n], T)

```
{
        if T = 0

                return T RUE

        else if T < 0 or n = 0

                return FALSE

        else
```

return SUBSET_SUM(X[2 .. n], T) ∨ SUBSET_SUM(X[2 .. n], T − X[1])

}

You will implement int subset_sumR(x[],T), the recursive version and whose iterative version subset_sumI(x[],T) with the explicit use of stack

The function subset_sumR(x[],T) makes the following assumptions:

1) All the elements in the given array are sorted.
2) Also, we assume no duplicate entries in the given set.

You will write a recursive version of insertion sort (insertSortR ()) to solve the sorting problem and a tail recursive function to remove duplicates(deDupR()). Subsequently use the StackADT and convert the insertSortR( ) and deDupR( ) to eliminate the recursion. The converted functions are insertSortI( ) and deDupI( ). You should not do any optimization when you convert tail recursive function to an iterative one.

Working of this algorithm:
If T = 0, then the elements of the empty subset sum to T . Otherwise, if T is negative or the set X is empty, then no subset of X sums to T . Otherwise, if there is a subset that sums to T , then either it contains X[1] or it doesn't. This is solved recursively.

You should use the following version of recursive insertion sort (not tail recursive)

Algorithm insertionSort(a, first, last)
if (the array contains more than one element)
{
1. **Sort the array elements a[first] through a[last-1] recursively calling insertionSort(a, first, last-1);**
2. **Insert the last element a[last] into its correct sorted position within the rest of the array;**

}

Use the sorted array as the input to your subset sum problem. ***You can assume there is no duplicate elements in your input***.

You may refer to stack.h for the StackADT descriptions. Stack ADT pushes an array of 5 integer onto the stack each time. (it would be easier in the code if you push all the elements as a single array/struct, depending on the coding requirement). You can change this if your code requires to push more elements into the stack.

Tasks to do:

Task-1:
Write the recursive versions of the following algorithms in recur.c file
1) void insertionSort_R (int arr[], int first, int last)
precond : arr[] contains elements to be sorted in the indexs from first to last
effect : arr[] contains elements to be sorted in the indexs from first to last
2) int SUBSET_SUM_R(int arr[], int T)
precond : arr[] contains elements which are sorted, de-duplicated and T is a positive integer
effect : returns 1 if there exists a proper subset sums to T. It returns 0 otherwise.

Task 2:
Write the iterative versions of the following algorithms in iterate.c file

1) void insertionSort_I (int arr[], int first, int last)
precond : arr[] contains elements to be sorted in the indexs from first to last
effect : arr[] contains elements to be sorted in the indexs from first to last
2) int SUBSET_SUM_I (int arr[], int T)
precond : arr[] contains elements which are sorted, de-duplicated and T is a positive integer
effect : returns 1 if there exists a proper subset sums to T. It returns 0 otherwise.

Task-3:

Write the MyFuns.h header file making the appropriate definitions needed for solving the tasks 1 and 2.

Task-4:
Write a driver file main.c which accepts a infile containing n integers as a command line parameter and displays appropriate message in the outfile.

Inputfiles : stack.h, stackops.c, queens.h, queens.c

Support fuctions : Nil

Deliverables : recur.c, iter.c, main.c, Myfun.h, makefile