**CSCI 204 Stack ADT Workshop**

Xiannong Meng

1. Write a function **eval\_postfix( e )** to compute the value of a postfix expression using a stack. The parameter **e** is an algebraic expression in the form of a string. We can assume each value in the expression is a single digit, and the operators are limited to +, -, \*, and /. For example 23\* should result in 6, 643+\* should result in 42, and 62/44+- should result in -5. For simplicity you may assume the original values in the expression string are non-negative.

The general algorithm is as follows. You may assume all expressions are well formed (valid). Test your program using the examples given in the problem and pick some extra ones by yourself. The test program can be found at the course website.

while expression not exhausted yet:

 read next token

 if it is a variable:

 push it onto the stack

 else: # an operator ‘op’

 right = pop()

 left = pop()

 result = left op right # you should write a function for doing the arithmetic operation

 push the result back onto the stack

the value at the top of the stack is the result of the expression

1. Airline scheduling problem (HPAir problem from Carrano’s *Data Abstraction and Problem Solving in C++* 3rd edition.) A test program and other supporting files are on the course website.

Given

* + 1. A set of cities that HPAir serves
		2. Pairs of city names, each pair represents the origin and destination of one flight
		3. Pair of cities names each of which represents a request to fly from an origin to a destination

Find whether or not a path exists between two cities requested by a passenger

**General idea**

* 1. Start from the origin city
	2. Find a city that is connected to the current city
	3. If the next city is the destination, we are done. Report a route has been found
	4. Otherwise go from this city and repeat step 2
	5. If we visited all cities and no route is found, we declare the failure

**Algorithm**

is\_path(in\_orig\_city, in\_dest\_city)

 my\_stack = create a new stack

 mark all cities un-visited;

 my\_stack.push(in\_orig\_city);

 mark in\_orig\_city visited;

 while (!my\_stack.is\_empty() and

 in\_dest\_city != my\_stack.peek())

 if (no flights exist from the city on the top

 of the stack to un-visited cities):

 my\_stack.pop() # backtrack

 else:

 select an unvisited destination city C from the city

 at the top of the stack

 my\_stack.push(C)

 mark C as visited

 if my\_stack.is\_empty():

 return False

 else:

 return True # the stack should contain the route