Linked lists, stacks, queues

Data and File Structures Laboratory

Preliminaries: measuring time

- Motivation: to measure the amount of (real) time taken to execute a part of your program
- Relevant headers, types, system calls (cf. `man gettimeofday`)

```
#include <sys/time.h>

struct timeval {
    __time_t tv_sec; /* seconds */
    __suseconds_t tv_usec; /* microseconds. */
};

int gettimeofday(struct timeval *tv, struct
    timezone *tz);
```

stores the number of seconds and microseconds elapsed since the "Epoch" (00:00 of 01.01.1970)

Also see `man getrusage`.

Preliminaries: measuring time

- Auxiliary macro

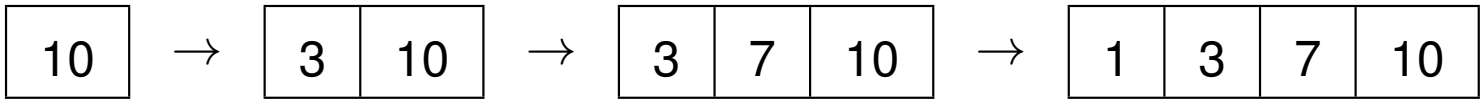
```
1  #define DURATION(start, end) ((end.tv_usec - start.tv_usec)
       + (end.tv_sec - start.tv_sec)) * 1000000
```

- Usage

```
1  struct timeval start_time, end_time;
2
3  /* store the starting time */
4  if (-1 == gettimeofday(&start_time, NULL))
5     ERR_MESG("gettimeofday failed");
6
7  /* code whose running time you want to measure */
8
9  /* store the finishing time */
10 if (-1 == gettimeofday(&end_time, NULL))
11    ERR_MESG("gettimeofday failed");
12
13 /* DURATION computes # microsecs between start and end */
14 printf("%d\n", (int) DURATION(start_time, end_time));
```

Problems I

1. Write a program that takes a single positive integer (say N) as a command line argument, generates N random integers between 0 and $10,000$ one by one, and inserts them (one by one) into an initially empty list in sorted order.

   Example:
   Generated elements: 10, 3, 7, 1, . . .
   List: 

   Use the following in turn to store the list:

   (a) an array;

   (b) a “traditional” linked list;

   (c) an array implementation of a linked list.

Problems II

Run your program 5 times each for $N = 100, 500, 1000, 2000, 3000, \ldots, 10000$. Print the sorted list to standard output, and the time taken (followed by a single tab, but no newline) to standard error. Find the average time taken for each value of N and for each implementation method given above. You may use the shell script given below.

2. Modify your program above so that it generates *two* sorted lists instead of one. Write a function to merge these two lists into a single sorted list. For this problem, use traditional linked lists only.

Problems III

3. Given a list of numbers (provided as command line arguments), write a program to compute the nearest larger value for the number at position i (nearness is measured in terms of the difference in array indices). For example, in the array $[1, 4, 3, 2, 5, 7]$, the nearest larger value for 4 is 5.
   Implement a naive, $O(n^2)$ time algorithm, as well as an $O(n)$ time algorithm for this problem. Compare the run times of your algorithms.

Problems IV

4. There are N petrol pumps $P_1, P_2, \ldots, P_N$ arranged in a clockwise direction along a circular road. Consider a truck with a fuel tank that is initially empty, but which has infinite capacity. The truck will initially fuel up at some P_i and move in a clockwise direction along the circular road. Each time it reaches a petrol pump, it will take up all the petrol available at that pump.

   Write a program to determine the first P_i such that if the truck starts from P_i, it will be able to completely traverse the circle and return to P_i.

   The amount of petrol that every petrol pump has (in litres), and the distance from that petrol pump to the next petrol pump (in km) will be given to you as command line arguments. Assume that the truck needs 1 litre of fuel to travel 1 km.

Problems V

Example: Let there be 4 petrol pumps having 4, 6, 7, and 4 litres of petrol respectively. Let the distance between these pumps be 6, 5, 3, and 5 km respectively. Then your program should output 2, since the first pump from which the truck can complete a circular tour is the 2nd petrol pump.

Also think about when such a tour will *not* be possible.

Shell script for Problem 1

```
1  # Assumes that your program prints the time followed by a tab /
       space
2  # e.g. using
3  # printf("%d\t", (int) DURATION(start_time, end_time));
4  cp /dev/null prob1-output.txt
5  for i in 100 500 {1000..10000..1000}; do
6      echo -n "$i "
7      for j in {1..5}; do
8          ./prog1 $i >> prob1-output.txt
9      done
10     echo ""
11 done
```