Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Practice Sheet #08

Topic: Memory Allocation in C

Date: 01-03-2017

1. Given a sequence a_0; a_1; : : : ; a_{n-1} of distinct numbers, the problem of *All Nearest Smaller Value* (ANSV) is to identify for each number a_i, the *smallest* j, j > i, j <= n-1, such that $a_j < a_i$. It is undefined if no such j exists. For example, in the sequence 8; 9; 7; 5; 6, (indexed as 0; 1; 2; 3; 4), the ANSV's are 2; 2; 3; u; u, where u denotes undefined. (the ANSV for the last element is always unde_ned). Note that you cannot change the order of the input sequence, since ANSV's depend on this ordering.

   Complete the following program that computes the ANSV's by considering the elements from left to right and by keeping track of those elements for which the ANSV's are yet to be determined. Clearly, these elements form a monotonically increasing subsequence a_{i1} ; a_{i2} ; : : : ; a_{ik} , $a_{i1} < a_{i2} <$ _ _ _ $< a_{ik}$, $i_1 < i_2 <$ _ _ _ $< i_k$. Initially, the sequence consists of a_0. In the j-th step (i.e., when a_{j+1} is considered), if $a_{j+1} < a_j$ (= a_{ik}) then the ANSV of aj is j + 1, otherwise we add a_{j+1} to the subsequence. More generally, aj+1 will be the ANSV for a_{is} ; a_{is+1}; : : : ; a_{ik} , if s > 1 and $a_{is+1} < a_{j+1} < a_{is}$, or if s = 1 and $a_{j+1} < a_{i1}$.

   Notice that a_{is} can be detected by working backwards from the end of the sequence, namely a_{ik} , and deleting them one by one. The new element is added at the end, i.e., the subsequence becomes a_{i1} ; a_{i2} ; : : : ; a_{is+1} ; a_{j+1}. Evidently, the subsequence can be maintained as a stack. When we have considered the entire array, the remaining elements in the stack are the ones whose ANSV's are unde_ned (use -1 to denote it). Note that in the stack, it is desirable to store the index of the elements (i.e., j instead of a_j). We can recover a_j from j.

```
#include <stdio.h>

main ()
{
int j, length, *a, *b, *stack, bottom, top;
printf("how many integers?\n"); scanf("%d", &length);
a = (int *)malloc(length*sizeof(int)); /* for storing the input */
b = (int *)malloc(length*sizeof(int)); /* for storing the ANSV's */

/* Allocate memory to stack */
stack =_________________ ;
for (j=0; j<length; j++) scanf("%d",&a[j]); /* Read input */
top = 0; stack[top] = 0; /* Initialize the stack */
for ( ___________; j+1<length; j++) {
        /* Consider array elements one-by-one */
   while( (top _______________) && (a[j+1] <___________ ) ) {
/* j+1 is the ANSV of all elements in the stack that are larger than
a[j+1] */
b[stack[top]] = j+1;
_____________________;
}
top = top + 1;
stack[top] = _________________;
}
for (; top _____________; top- -) _____________________= -1; /*
Undefined ANSV */
```

```
for (j=0; j<length; j++) printf("%d\n", b[j]); /* Print the ANSV's */
}
```

2. How many bytes are allocated to the pointer **p** after the following call?

```
#define MAXSIZE 100
p = (long int *)malloc(MAXSIZE * sizeof(long int));
```

(a) 4
(b) 100
(c) 400
(d) 1600

3. Suppose we have the following statement inside a function foo( ), and int *p is a global variable:

```
p = (int *) malloc(50 * sizeof(int));
```

Suppose main( ) calls foo( ). From which of the following places can we access the memory allocated by this call?

(a) Only within the function foo( )
(b) Within the function foo( ) and the function main( )
(c) Within any function, that is, the memory is global

4. Code for dynamically allocating memory for an array of 10 **ratTyp** elements and storing the array address in **ratArr** is:_____

5. Dynamically allocated memory should be freed after its use. Mark as True or False.

6. Dynamically allocate a 2-D array of characters to a pointer for storing 100 strings each of maximum character length 80. Write a C program for it.

--*--