CS13002 Programming and Data Structures, Spring 2005

End-semester examination

Total marks: 60	April 2005	Total time: 3 hours

Roll no: ____________________ **Section:** __________

Name: ____

- *This question paper consists of eight pages. Do not write anything on this page except your name, roll number and section.*
- *Answer all questions.*
- *Write your answers on the question paper itself. Your final answers must fit in the respective spaces provided. Strictly avoid untidiness or cancellations on the question-cum-answer paper.*
- *Do your rough work on the given answer-script or additional supplements. The rough work must be submitted, but will not be evaluated. Only answers in the question-cum-answer paper will be evaluated.*
- *Not all blanks carry equal marks for Questions 2, 4, 5 and 6. Evaluation will depend on the overall correctness.*

(To be filled in by the examiners)

Question No	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
Marks							

1. For each of the following parts, circle the letter corresponding to the correct answer. **(10)**

(a) What value does the call `strFunc1("PDS 2005")` return?

```
int strFunc1 ( char A[] )
{
   int i = 0;
   while (A[i]) i = i + 1;
   return i;
}
```

(A) 7 (B) 8 (C) 9 (D) ASCII value of `'P'`

(b) What string does the following program print?

```
#include <stdio.h>
#include <string.h>

void strFunc2 ( char A[] , int n )
{
   char t;
   if (n <= 1) return;
   t = A[0]; A[0] = A[n-1]; A[n-1] = t;
   strFunc2(&A[1],n-2);
}

int main ()
{
   char A[10] = "PDS 2005";
   strFunc2(A,strlen(A));
   printf("%s", A);
}
```

(A) `5002 SDP` (B) `5DS 200P` (C) `PDS 2005` (D) `SDP 2005`

(c) What integer value is printed by the following program?

```
int main ()
{
   int A[5] = {4,8,12,16,20}, *p;
   p = A + A[0];
   printf("%d", *p);
}
```

(A) 4 (B) 8 (C) 16 (D) 20

(d) What integer value is printed by the following program?

```
#include <stdio.h>

void what ( int *p )
{
   *p = 10; p = p + 1;
}

int main ()
{
   int A[5] = {4,8,12,16,20}, *p;
   p = A;
   what(p);
   printf("%d", *p);
}
```

(A) 4 (B) 8 (C) 10 (D) 12

(e) Assume that on a certain machine an `int` variable is of size 32 bits and each memory address is also of size 32 bits. Assume further that the `sizeof()` function call returns the size of its operand in bytes. Consider the following structure:

```
struct myStruct {
    int a;
    int b[10];
    int *c;
}
```

What does `sizeof(struct myStruct)` return on this machine?

(A) 3 (B) 12 (C) 48 (D) 384

(f) Which of the four assignments is legal after the following declaration?

```
int A[10], B[20], *C;
```

(A) `A = B;` (B) `B = A;` (C) `A = C;` (D) `C = A;`

(g) Consider the following sequence of push and pop operations on an initially empty stack S.

```
S = push(S,1);
S = push(S,2);
S = pop(S);
S = push(S,3);
S = push(S,4);
S = pop(S);
S = pop(S);
S = pop(S);
```

Which of the following is the correct order in which elements are popped?

(A) 1,2,3,4 (B) 2,1,3,4 (C) 2,3,4,1 (D) 2,4,3,1

(h) Count the exact number of multiplications performed by the following function:

```
double f ( double A[SIZE][SIZE] , int n )
{
    int i, j;
    double prod = 1;
    for (i=0; i<n; ++i) for (j=i; j<n; ++j) prod = prod * A[i][j];
    return prod;
}
```

(A) n (B) $(n^2 - n)/2$ (C) $(n^2 + n)/2$ (D) n^2

(i) Consider the three functions $f(n) = n^2 + n + 234$, $g(n) = 12n^2 + 34$ and $h(n) = 123n + 4$. Which of the following statements is *not* true?

(A) $f(n) = \mathrm{O}(g(n))$ (B) $g(n) = \mathrm{O}(f(n))$ (C) $f(n) = \mathrm{O}(h(n))$ (D) $h(n) = \mathrm{O}(f(n))$

(j) What is the running time of insertion sort on an array of n elements?

(A) $\mathrm{O}(n^2)$ (B) $\mathrm{O}(n \log n)$ (C) $\mathrm{O}(n)$ (D) $\mathrm{O}(1)$

2. In this exercise we deal with complex numbers of the form $(a/b) + \mathrm{i}(c/d)$ with a, b, c, d integers and with b and d positive. The following structure represents such a complex number.

```
typedef struct {
    int a;    /* Numerator of the real part */
    int b;    /* Denominator of the real part */
    int c;    /* Numerator of the imaginary part */
    int d;    /* Denominator of the imaginary part */
} ratComp;
```

Our aim is to compute the sum and product of two such complex numbers with each fraction reduced to the standard form (i.e., reduced to the form m/n with $\gcd(m, n) = 1$, $n > 0$). **(10)**

```
int gcd ( int a , int b )    /* Compute the gcd of two integers */
{
    if (a < 0) a = -a; if (b < 0) b = -b;
    if (a == 0) return b; if (b == 0) return a;
    return gcd(b, ____________________________ );
}
void ratReduce ( int *a , int *b )    /* Reduce a fraction to lowest terms */
{
    int d;
    d = gcd( _______ , _______ ); *a = _________________; *b = _________________;
}
void ratSum ( int a1 , int b1 , int a2 , int b2 , int *a , int *b ) /* (a1/b1)+(a2/b2) */
{
    *a = a1 * b2 + a2 * b1; *b = b1 * b2; ratReduce( ______________ , ______________ );
}
void ratMul ( int a1 , int b1 , int a2 , int b2 , int *a , int *b ) /* (a1/b1)*(a2/b2) */
{
    *a = a1 * a2; *b = b1 * b2; ratReduce( ______________ , ______________ );
}
ratComp compSum ( ratComp z1 , ratComp z2 )    /* Compute z1 + z2 */
{
    ratComp z;
    /* Set the real part of z */
    ratSum ( ____________________________________________________________________________ );
    /* Set the imaginary part of z */
    ratSum ( ____________________________________________________________________________ );
    return z;
}
ratComp compMul ( ratComp z1 , ratComp z2 )    /* Compute z1 * z2 */
{



}
```

3. This exercise deals with square matrices and column vectors. First define the following two data types:

```
#define n 100
typedef int matrix[n][n];  /* matrix data type */
typedef int vector[n];     /* vector data type */
```

(a) Let A be an $n \times n$ matrix and b a column vector of n elements. The product Ab is again a vector of size n and can be computed in the usual manner as taught in your math courses. Write a function to compute the product Ab. Do not use any variable other than two indices `i,j`. Assume that `A` and `b` are scanned before passing to the function and that they have respective sizes $n \times n$ and n, where n is defined as above. The product Ab is to be stored in the vector `c`. You don't have to print the elements of `c`. **(6)**

```
void matVecMul ( matrix A , vector b , vector c )
{
   int i, j;

}
```

(b) Compute, in terms of the size n, the exact number of integer additions and the exact number of integer multiplications performed by `matVecMul`. **(4)**

4. You are given a set of n points in the plane, where each point is represented as a structure having two components which are the x and y coordinates respectively. A point p *dominates* another point q iff $p.x \geqslant q.x$ and $p.y \geqslant q.y$. The problem of determining the maximal elements is to identify all points that are not dominated by any other point in the set. For example, in $\{(2,3),(3,4),(1,5)\}$ the maximal points are $(3,4)$ and $(1,5)$.

(a) What are the maximal points in the set $\{(1,5),(2,4),(3,3),(4,1),(5,2)\}$? **(2)**

____

(b) A straightforward approach to identifying maximal points will be to check for every point if it is maximal by comparing against every other point in the set — this will cost us approximately n^2 comparisons. However, if we assume that the points are sorted on the basis of their x coordinates, then for every point p, we check if there is any other point r to its right ($r.x > p.x$) that has $r.y > p.y$. If $(x_0,y_0),(x_1,y_1),\ldots,(x_{n-1},y_{n-1})$ are the input points sorted in order of their x coordinates, then for every point (x_i,y_i), it suffices to keep track of $yMax_i = \max\{y_{i+1},y_{i+2},\ldots,y_{n-1}\}$. Note that for any point p_i, if $yMax_i$ is larger than y_i, then there exists a point that dominates (x_i,y_i), otherwise there is no point that dominates (x_i,y_i) and it is maximal (all points to its left have smaller x coordinates).

We use an array **maxima** to store information if a point is maximal. We consider the (sorted) points in decreasing order of x coordinates. Using the variable **yMax**, we keep track of the largest y coordinate till a certain point. Fill in the blanks so that the maximal points are printed out by the following program. For simplicity, assume that all coordinates are distinct. **(5)**

```
#include <stdio.h>
int main ()
{
   struct point { int x; int y; };
   struct point *A;
   int i, n, *maxima, yMax;
   printf("how many integers?\n"); scanf("%d",&n);
   A = (struct point *)malloc(n*sizeof(struct point));
   /* Allocate memory for maxima */
   maxima = ____________________________________________;
   readInput(A,n); /* This function, not to be written by you, is assumed to scan and
                      store n point structures in the array A sorted in the increasing
                      order of x coordinates (i.e., A[i].x < A[i+1].x for all i) */
   yMax = A[n-1].y;    /* Initialize the value of yMax */

   for ( _______________ ; _______________ ; i-- ) {

      if (A[i].y < yMax) ____________________________________________;
      else {
         ______________________________________________________
         ______________________________________________________
      }
   }
   printf("The maximal points are:\n");
   for (i=0; i<n; i++)
      if ( __________________ ) printf("(%d,%d) ", ______________________ );
}
```

(c) In terms of n, approximately how many comparisons are made in the above program? **(3)**

____

____

5. **(a)** We are given a linked list and we want to traverse it in a way such that the links are reversed by the time we complete the traversal. Complete the function **traverseReverse** that achieves this objective. You can assume that the input pointer is not **NULL**. Do not use any new variables or malloc/calloc instructions — your program should accomplish the goal by careful manipulation of the pointers. **(6)**

```
struct node {
   char data;
   struct node *link;
};

struct node *traverseReverse ( struct node *p1 )
{
   struct node *p2, *p3;
   /* p2 points the part of the list that has already been reversed and p1 points to
      the part that is yet to be reversed. p3 may be used as an auxiliary pointer. */
   /* Initialize the pointers */
   p2 = p1; p1 = p1->link; p2->link = NULL;
   /* As long as the unprocessed part contains more nodes to process */

   while ( ________________________________ ) {
   /* Reverse another link from the unprocessed part. Use pointer assignments only. */

      ______________________________________________________________________

      ______________________________________________________________________

      ______________________________________________________________________

      ______________________________________________________________________
   }
   p1->link = p2;                                /* Link p1 to head the reversed list */
   return _______________;  /* Return a pointer to the header of the reversed list */
}
```

(b) The following function **recReverse** recursively reverses a linked list. Which one of the two functions would you prefer and why? **(4)**

```
struct node *recReverse ( struct node *p )
{
   struct node *q, *head;
   if (p->link == NULL) return p; /* one node */
   /* Recursively reverse the list of at least 2 nodes minus first element */
   head = p; p = p->link; q = p = recReverse(p);
   /* Append the first node at the end of the reversed list */
   while (q->link != NULL) q = q->link;
   q->link = head; head->link = NULL; return p;
}
```

____

____

____

____

____

____

____

6. Given a sequence $a_0, a_1, \ldots, a_{n-1}$ of distinct numbers, the problem of *All Nearest Smaller Value* (ANSV) is to identify for each number a_i, the *smallest* j, $j > i$, $j \leqslant n-1$, 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 undefined). Note that you cannot change the order of the input sequence, since ANSV's depend on this ordering.

(a) Determine the ANSV's for the sequence $7, 2, 3, 8, 1, 5, 4, 9, 6$. **(2)**

(b) 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 $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$, $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$, $i_1 < i_2 < \cdots < i_k$. Initially, the sequence consists of a_0. In the j-th step (i.e., when a_{j+1} is considered), if $a_{j+1} < a_j$ $(= a_{i_k})$ then the ANSV of a_j is $j+1$, otherwise we add a_{j+1} to the subsequence. More generally, a_{j+1} will be the ANSV for $a_{i_s}, a_{i_{s+1}}, \ldots, a_{i_k}$, if $s > 1$ and $a_{i_{s-1}} < a_{j+1} < a_{i_s}$, or if $s = 1$ and $a_{j+1} < a_{i_1}$. Notice that a_{i_s} can be detected by working backwards from the end of the sequence, namely a_{i_k}, and deleting them one by one. The new element is added at the end, i.e., the subsequence becomes $a_{i_1}, a_{i_2}, \ldots, a_{i_{s-1}}, a_{j+1}$. 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 undefined (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. **(8)**

```
#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 */
}
```