CS13002 Programming and Data Structures, Spring 2005

Mid-semester examination

Total marks: 60 | February 2005 | Total time: 2 hours

Roll no: ____________________ **Section:** __________

Name: ____

- *Answer all questions.*
- *Write your answers in 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 3, 4 and 5. Evaluation will depend on the overall correctness.*

1. For each of the following parts, circle the letter corresponding to the correct answer. **(1×12)**

(a) Which of the following is not a legal name of a C variable?

(A) `_123var` (B) `123_var` (C) `var_123` (D) `var123_`

(b) Assume that the integer variables `a`, `b` and `c` respectively store the values 5, 6 and 7 respectively. At that instance the following statement is executed:

```
a = a * b % c + a;
```

What value is assigned to the variable `a`?

(A) 5 (B) 6 (C) 7 (D) 30

(c) What value is assigned to the variable `a`?

```
#define N 10+10
a = N * N;
```

(A) 400 (B) 210 (C) 200 (D) 120

(d) Assume that `a` and `b` are `int` variables and `x` is a `float` variable holding the values 3, 6 and 4.5 respectively. Which of the following conditions is true?

(A) `(a*b>x*x)` (B) `(a*b>(int)x*x)` (C) `(a*b>(int)(x*x))` (D) `(a*b>(int)x*(int)x)`

(e) Which of the following conditions is equivalent to the condition `(!((x>=y)&&(y>=z)))`?

(A) `(!(x>=z))` (B) `(x<=z)` (C) `((x<y)&&(y<z))` (D) `((x<y)||(y<z))`

(f) How many times is the body of the following loop executed?

```
int i;
for (i=0; i<100; i=i+3) printf("i = %d\n", i);
```

(A) 100 (B) 97 (C) 34 (D) 33

(g) What does the following function return in terms of the argument n?

```
unsigned int f ( unsigned int n )
{
   unsigned int s = 0, t = 0;
   while (n > 0) {
      s = s + n;
      t = t + n * n * n;
      n = n - 1;
   }
   return (t - s * s);
}
```

(A) 0 (B) 1 (C) $n^3 - n^2$ (D) $\frac{1}{12}n(n^2-1)(3n+2)$

(h) What does the following function return? Assume that $n > 0$.

```
int g ( int A[] , int n )
{
   int i, s;
   for (s=A[0], i=1; i<n; ++i) s = s + A[i] - A[i-1];
   return s;
}
```

(A) `0` (B) `A[n-1]` (C) `A[n]` (D) `A[n-1]+A[0]`

(i) What value of `s` does the following program print?

```
#include <stdio.h>
void sum ( int a , int b , int s ) { s = a + b; }
main ()
{
   int a = 3, b = 2, s = 1;
   sum(a,b,s);
   printf("s = %d\n", s);
}
```

(A) 1 (B) 2 (C) 3 (D) 5

(j) What value does the call `h(5)` return, where `h` is defined as follows?

```
int h ( int n )
{
   if (n == 0) return 1;
   return 2*h(n-1);
}
```

(A) 25 (B) 32 (C) 64 (D) 120

(k) What are the least and largest integers representable in the 10-bit signed 2's complement format?

(A) $-2^9, 2^9$ (B) $-2^9+1, 2^9-1$ (C) $-2^9+1, 2^9$ (D) $-2^9, 2^9-1$

(l) Given that `a`, `b`, `c`, `d` are very large floating point variables and we have to compute `(a*b)/(c*d)`, which will be the preferred way?

(A) `(a*b)/(c*d)` (B) `a*(b/(c*d))` (C) `(a/c)*(b/d)` (D) `((a*b)/c)/d`

2. (a) Consider the following program: **(6)**

```
#include <stdio.h>
main ()
{
    int m, n;
    printf("Supply two integers: ");
    scanf("%d%d",&m,&n);
    m = m - n;
    n = m + n;
    m = n - m;
    printf("%d %d\n", m,n);
}
```

Describe, in terms of the scanned integers `m` and `n`, what integer values are printed at the end of the above program:

Answer: ______________________________

____

____

____

(b) Consider the following function that expects a non-negative integer argument:

```
unsigned int doit ( unsigned int n )
{
    unsigned int m;
    m = 0;
    while (n > 0) {
        m = (m*10) + (n%10);
        n = n/10;
    }
    return m;
}
```

What does `doit(92429)` return? ______________ **(2)**

Describe the return value of `doit` as a function of the input argument `n`. **(4)**

____

____

____

3. Consider a game played between two players I and II. Initially there is a collection of N coins in a bag. Player I starts. Subsequently, moves alternate between the players. During a move of Player I, 1 or 2 or ... or M_1 number of coins should be take out from the bag, whereas for a move of Player II, 1 or 2 or ... or M_2 number of coins need be taken out. The player who makes the last move and empties the bag wins.

Let us now inductively define a *winning stage* of the game for a player. Suppose that at some stage of the game exactly C coins are left in the bag and it is Player I's turn to move next. (This is the situation initially with $C = N$.) If $C = 0$, Player I cannot make a move and loses. If $1 \leqslant C \leqslant M_1$, Player I can remove all the remaining coins and wins. Finally, suppose $C > M_1$. A good move for Player I is taking out m coins for $1 \leqslant m \leqslant M_1$ such that $C - m$ is *not* a winning stage for Player II. If such an m exists, the current stage of the game is winning for Player I, and m is called a *winning move* for Player I. A winning move for Player II can be analogously defined.

Let us take an example: $M_1 = 2$ and $M_2 = 3$. The stage $C = 0$ is losing for both the players, whereas the stages $C = 1, 2$ are winning for both of them. The stage $C = 3$ is losing for Player I, since either 1 or 2 coins may only be taken out, leaving 2 or 1 coins respectively in the bag. Both these stages are winning for Player II. On the other hand, Player II can draw three coins, so the stage $C = 3$ is winning for Player II.

(a) What is a winning move, if any, for Player I at stage $C = 4$? ________________ **(2)**

(b) What is a winning move, if any, for Player II at stage $C = 5$? ________________ **(2)**

(c) Complete the following recursive function that accepts the current stage C (number of coins left) and the limits M_1, M_2 (not necessarily in that order) as the sole arguments and returns a winning move for the player who is going to make a move, provided that such a move exists. If no winning move exists, the function returns 0. **(8)**

```
unsigned int winningMove ( unsigned int C , unsigned int myM, unsigned int yourM )
{
    unsigned int m;

    /* If no coins are left, the player loses. */

    if (C == 0) ______________________________________________

    /* If no more than myM coins are left, the player takes out all. */

    __________________________________________________________

    /* The player cannot take out all coins. Search for a winning move m. */

    for (m = 1; m <= __________; m = m+1) {

        /* If taking out m coins leaves the opponent in a non-winning stage, */

        if ( _____________________________________________ )

            /* then m is a winning move. */

            ______________________________________________
    }

    /* No good moves have been found. Return failure. */

    __________________________________________________________
}
```

4. Fill in the blanks to complete the following program that partitions an array `A` with respect to a number `key` into three consecutive regions `L,E,G`, such that all elements in `L,E,G` are less than, equal to or greater than `key` respectively. Note that, the elements within the individual regions L and G don't have to satisfy any ordering among themselves. The program uses a function `partition` that takes four parameters: an array `A`, two indices `start` and `end`, and a number `key` and partitions the subarray `A[start],...,A[end]` into three regions `L,E,G` described above. It does so by performing some exchanges, but without using an extra array. At any intermediate stage it maintains the following invariant: `A = LUEG`, where `L`, `E`, and `G` are described above and `U` corresponds to the elements that are yet unclassified (i.e., unprocessed). Initially all elements are unclassified, and finally `U` should be empty. Three variables `us,ue,ee` in the function keep track of the boundaries of the four regions: `us` marks the start of the block `U`, `ue` the end of `U` and `ee` the end of `E`. In every iteration, `A[ue]` is inspected and the boundaries are modified accordingly. **(12)**

key = 20	12	7	3	20	37	26	20	20	43	34
			us			ue		ee		
	←L→		←—— U ——→				← E →		← G →	

```
#include <stdio.h>
#define LENGTH 100

void partition ( int A[] , int key , int start , int end )
{
   int us,ue,ee;  /* Boundaries */
   int t;         /* Temporary variable that may be used for swapping */
   /* Initialize the boundaries us, ue and ee */
   us = start; ue = end; ee = end;
   /* Repeat the following loop as long as the unclassified region U is nonempty */

   while ( ____________________________ ) {
      if ( A[ue] < key ) {
         /* Append A[ue] to the region L. You can use at most three assignments. */

         ______________________________________________________________
         /* Adjust the boundary between L and U. */

         ______________________________________________________________
      } else if ( A[ue] == key ) {
         /* The region E grows, so adjust the U-E boundary. */

         ______________________________________________________________
      } else { /* Now we are in the case A[ue] > key. */
         /* The region G grows. You can use at most three assignments. */

         ______________________________________________________________
         /* Adjust the boundaries ue and ee. */

         ______________________________________________________________
      }
   }
}

main ()
{
   int A[LENGTH];
   int i, j;

   /* Scan array elements */
   printf("Supply %d integers:\n",LENGTH);
   for (i = 0; i <= (LENGTH - 1); i = i + 1) scanf("%f", &A[i]);

   /* Now partition the entire array A with respect to the key A[0] */

   partition( ______________ , ______________ , ______________ , ______________ );
   printf("The partitioned array is\n");
   for (i = 0; i <= (LENGTH - 1); i = i + 1) printf("%f\n ", A[i]);
}
```

5. A crucial step in binary search (on a sorted array) is to split the array, when required, into two nearly equal parts. It is also possible to do the search on a sorted array by splitting the array using Fibonacci numbers F_k, when required. (Recall that Fibonacci numbers are defined as $F_0 = 0$, $F_1 = 1$ and $F_k = F_{k-1} + F_{k-2}$ for $k \geqslant 2$.) The Fibonacci search scheme works as follows:

- The initial search window is the whole (sorted) array.
- If there is only one element in the search window, the search fails or succeeds.
- Otherwise, suppose that there are n elements in the search window.
  - Find F_k such that $F_k \geqslant n$ and $F_{k-1} \leqslant n$.
  - If the element at position F_{k-2} from the beginning of the search window matches the key, then the search succeeds.
  - If the key is smaller, then continue the search among the first F_{k-2} elements from the start of the current window.
  - Otherwise, continue searching among the rest of the elements in the search window. Note that the size of the window for the remaining elements need not be a Fibonacci number.

Fill in the missing parts of the program given below to perform Fibonacci search. **(12)**

```
#include <stdio.h>

void fS(int keyAr[], int key, int sIdx, int n, int Fk, int Fk_1)
{
  int t, Fk_2, loc;
  if (n == 1) {

    loc = ______________________________;
    if (key == keyAr[loc])
       printf("Search succeeded at location %d\n", loc);
    else
       printf("Search failed at location %d\n", loc);
  } else {
    Fk_2 = Fk - Fk_1;

    loc = ______________________________;
    if (key == keyAr[loc]) {
       printf("Search succeeded at location %d\n", loc);
       return;
    } else if (key < keyAr[loc]) {

      fS(keyAr, key, ___________, ___________, ___________, ________________________);
    } else {

      for (Fk = Fk_1 = 1; ____________________ > Fk; t = Fk + Fk_1, Fk_1 = Fk, Fk = t);

      fS(keyAr, key, _____________________, _____________________, __________, __________);
    }
  }
}

fSWrap(int keyAr[], int key, int n )
/* First call to fS with appropriate values for sIdx, n, Fk and Fk_1 respectively. */
{
  int t, Fk, Fk_1, Fk_2;
  for (Fk = Fk_1 = 1; n > Fk; t = Fk + Fk_1, Fk_1 = Fk, Fk = t);

  fS(keyAr, key, __________, __________, __________, __________);
}

main()
{
  int keyAr[] = { -34, -14, 5, 35, 53, 350, 376, 456, 785, 975 };
  fSWrap(keyAr, 376, 10);
}
```