CS11001/11002/13002 Programming and Data Structures, Spring 2008

Mid-semester examination

Maximum marks: 50	February 25, 2008 (FN)	Total time: 2 hours

Roll no: ____________________ **Section:** __________

Name: ____

- *This question paper consists of five pages.*
- *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.*
- *Use of calculators is allowed.*

(To be filled in by the examiners)

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

1. (a) Find the 32-bit floating point representation of 35.6 in the IEEE 754 format. Show your calculations. **(4)**

Solution: $35 = 32 + 2 + 1 = 2^5 + 2^1 + 2^0 = (100011)_2$. Also, we have

$$\begin{aligned} 0.6 \times 2 &= 1.2, \\ 0.2 \times 2 &= 0.4, \\ 0.4 \times 2 &= 0.8, \\ 0.8 \times 2 &= 1.6, \end{aligned}$$

that is, $0.6 = (0.1001\,1001\,1001\,1001\,1001\,1001\,\ldots)_2$. Therefore,

$$35.6 = (1.00011\,1001\,1001\,1001\,1001\,1001\,1001\,\ldots)_2 \times 2^{132-127}.$$

Finally, $132 = 128 + 4 = (10000100)_2$.

Write your final answer below.

0		1	0	0	0	0	1	0	0		0	0	0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0

(b) How many floating point numbers x can be represented in the 32-bit IEEE 754 format with $1 \leqslant x \leqslant 2$?

$2^{23} + 1$ **(2)**

In Parts (c)–(d), a 32-bit pattern is interpreted as an unsigned fractional number with the *implicit* binary point to the left of bit 15. As an example, the following pattern is interpreted as 22.6875 in decimal.

31 16	15 0
0000 0000 0001 0110 .	1011 0000 0000 0000

(c) The smallest positive number that can be represented in this scheme is 2^{-16} . **(2)**

(d) The largest number that can be represented in this scheme is $2^{16} - 2^{-16}$. **(2)**

(e) Write a function that, given a floating point number x as a parameter, prints m and e, where $x = m \times 2^e$ with $0.5 \leqslant |m| < 1$ and with e an integer. For $x = 0$, take $m = e = 0$. **(7)**

```
void fracexp ( double x )
{
```

```
int sign = 0, e = 0;
double m;

m = x;
if (m) {
    if (m < 0) { sign = 1; m = -m; }
    if (m >= 1) {
        while (m >= 1) { m /= 2.0; ++e; }
    } else if (m < 0.5) {
        while (m < 0.5) { m *= 2.0; --e; }
    }
}

if (sign) m = -m;
printf("%lf = (%lf) x 2^(%d)\n", x, m, e);
```

```
}
```

2. **(a)** Consider the following recursive C function.

```
unsigned int f ( unsigned int n )
{
    if (n < 10) printf("%d",n);
    else { printf("%d", n % 10); f(n/10); printf("%d", n % 10); }
}
```

What does the call **`f(351274)`** print? **`47215351274`** **(2)**

Parts (b)–(e) are based on the following recursive function. Assume that both n, k are positive.

```
int S ( int n, int k )
{
   if (k > n) return 0;
   if ( (k == 1) || (k == n) ) return 1;
   return S(n-1,k-1) + k * S(n-1,k);
}
```

(b) What is the value returned by **`S(5,3)`**? Show your calculations. **(3)**

$$
\begin{aligned}
S(5,3) &= S(4,2) + 3 \times S(4,3) \\
&= \Big[S(3,1) + 2 \times S(3,2)\Big] + 3 \times \Big[S(3,2) + 3 \times S(3,3)\Big] \\
&= \Big[1 + 2 \times \Big(S(2,1) + 2 \times S(2,2)\Big)\Big] + 3 \times \Big[\Big(S(2,1) + 2 \times S(2,2)\Big) + 3 \times 1\Big] \\
&= \Big[1 + 2 \times (1 + 2 \times 1)\Big] + 3 \times \Big[(1 + 2 \times 1) + 3 \times 1\Big] \\
&= 25.
\end{aligned}
$$

Therefore, **`S(5,3)`** returns ______25______.

(c) How many times is **`S()`** called (including the outermost call) to compute **`S(5,3)`**? ___11 times___ **(1)**

(d) How many multiplications are performed to compute the value of **`S(5,3)`**? ___5___ **(1)**

(e) Write a recursive function **`SMul()`** to count the number of multiplications in the call **`S(n,k)`**. **(5)**

```
int SMul ( int n , int k )
{
   if (k > n) return 0;
   if ( (k == 1) || (k == n) ) return 0;
   return SMul(n-1,k-1) + SMul(n-1,k) + 1;
}
```

3. Let $a_0, a_1, \ldots, a_{n-1}$ be $n \geqslant 1$ positive integers. The *continued fraction* $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ stands for the rational number: $\langle a_0, a_1, \ldots, a_{n-1} \rangle = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ddots + \cfrac{1}{a_{n-1}}}}$.

(a) Write an iterative function that reads positive integers $a_0, a_1, \ldots, a_{n-1}$ (not necessarily in that order). The function computes and prints the value of $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ as a rational number in the form p/q and also its floating-point value. The number of terms, that is, n is supplied to the function as an argument. **(6)**

```
void cfracitr ( int n )
{
   int num, den; /* Numerator and denominator */
   /* Declare other int variables, if necessary */
```

```
int i, tmp, a;

num = 1; den = 0;
for (i = n-1; i >= 0; --i) {
   printf("a_%d = ", i); scanf("%d", &a);
   tmp = num;
   num = a * num + den;
   den = tmp;
}
```

```
   printf("Value = %d/%d = %lf\n", num, den, (double)num / (double)den);
}
```

(b) Complete the following recursive function that returns the floating point value of a continued fraction $\langle a_0, a_1, \ldots, a_{n-1} \rangle$. Note that $\langle a_i, a_{i+1}, \ldots, a_{n-1} \rangle = a_i + \dfrac{1}{\langle a_{i+1}, a_{i+2}, \ldots, a_{n-1} \rangle}$ for $i = 0, 1, \ldots, n-2$. **(6)**

```
double cfracrec ( int i , int n )
{
   int a;
   printf("Enter a_%d: ", i); scanf("%d", &a);    /* Read a_i in a */
   /* The terminating case, no recursive call */

   if ( i == ______n-1______ ) return ________(double)a________ ;
   /* Make a recursive call and return */

   return ________(double)a + 1.0 / cfracrec(i+1,n)________ ;
}
```

The outermost call for computing $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ should be: **`cfracrec( ___0___ , ___n___ )`** **(1)**

4. Let $a_1, a_2, \ldots, a_n$ be a sequence of positive integers. An increasing subsequence of length l is a contiguous block $a_i, a_{i+1}, \ldots, a_{i+l-1}$ satisfying $a_i \leqslant a_{i+1} \leqslant \cdots \leqslant a_{i+l-1}$.

Write a C program to read a sequence of positive integers and to print the length of the longest increasing subsequence in it. In order to terminate the sequence, the user should enter zero or a negative value. Your program must contain *only one loop*. Both scanning the next integer and processing the scanned integer should be done in that loop. Write no functions other than **`main()`**. Do not use any array.

Here is a sample run of your program. **(8)**

```
Enter an integer:  9
Enter an integer:  2
Enter an integer:  6
Enter an integer:  8
Enter an integer:  5
Enter an integer:  7
Enter an integer:  -1
Length of the longest increasing subsequence = 3
```

```c
#include <stdio.h>

int main ()
{
   int a, prev = -1, runningmaxlen = 0, maxlen = 0;

   while (1) {
      printf("Enter an integer: "); scanf("%d", &a);
      if (a <= 0) {
         if (runningmaxlen > maxlen) maxlen = runningmaxlen;
         break;
      }
      if (a >= prev) {
         ++runningmaxlen;
      } else {
         if (runningmaxlen > maxlen) maxlen = runningmaxlen;
         runningmaxlen = 1;
      }
      prev = a;
   }
   printf("Length of the longest increasing sequence = %d\n", maxlen);
}
```