INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

Department of Computer Science & Engineering

Programming and Data Structures (CS11001)

Midsem (Autumn, 1st Year)

Date: Tue, Sep 27, 2011 *Time:* 09:00-11:00am

Students: 660 *Marks:* 55

Answer ALL the questions.
Do all rough work on separate rough sheets which you should not submit.
Answer on the question paper itself in the spaces provided.

Roll no: ____________________ **Section:** _______ **Name:** ____

1. A vector $\overrightarrow{X}$ may be expressed in terms of its components as: $\overrightarrow{X} = X_x\overrightarrow{i} + X_y\overrightarrow{j} + X_z\overrightarrow{k}$, where $\langle X_x, X_y, X_z\rangle$ are the Cartesian co-ordinates of X and $\overrightarrow{i}$, $\overrightarrow{j}$, $\overrightarrow{k}$ are the unit vectors. It may be represented using an array as: **`float X[3]={`**X_x, X_y, X_z**`}`** (substituting X_x, X_y, X_z with their actual numerical values in the 'C' code). Assume that $\overrightarrow{A}$, $\overrightarrow{B}$, $\overrightarrow{C}$, $\overrightarrow{P}$, etc are similarly declared/defined (with initialisation, if necessary).

   The dot product $\overrightarrow{A} \cdot \overrightarrow{B}$ may be computed and *returned* via the following 'C' function **`vecDP()`** defined as:

```
float vecDP(float A[3], float B[3]) {

   return A[0]B[0] + A[1]B[1] + A[2]B[2] ;

}
```

5

   Also, the cross product $\overrightarrow{C} = \overrightarrow{A} \times \overrightarrow{B}$ may be computed and stored in **`C[]`** via the following 'C' function **`vecCP()`** as:

```
void vecCP(float A[3], float B[3], float C[3]) {

   ____________________________________________ // extra

   C[0] = A[1]*B[2] - B[1]*A[2];

} // C[1]=...; C[2]=...; /* computed similarly to C[0] */
```

5

   Suppose you are given two vectors $\overrightarrow{A}$ and $\overrightarrow{B}$ corresponding to the endpoints of the line segment $\overline{AB}$, the vector $\overrightarrow{AB}$ (**`float AB[3]`**) may be computed as:

```
______AB[0]= B[0] - A[0]______; // AB[1]=...; AB[2]=...; /* done similarly */
```

1

1	2	3	4	5	T

You are also given the vector $\overrightarrow{P}$ for another point P. The vector $\overrightarrow{AP}$ (**`float AP[3]`**)may be computed as:

`AP[0]= P[0] - A[0]` **`; // AP[1]=...; AP[2]=...; /* done similarly */`** 1

The cross product $\overrightarrow{Z}_{P,AB} = \overrightarrow{AB} \times \overrightarrow{AP}$ (**`float Z_P_AB[3]`**) may be computed using an above defined function as: **`vecCP(AB, AP, Z_P_AB)`** **`;`** 1

Suppose that A, B, C are vertices of a triangle which are co-planer to P. The vector $\overrightarrow{AC}$ (**`float AC[3]`**) may be computed as:

`AC[0]= C[0] - A[0]` **`; AC[1]=...; AC[2]=...; /* done similarly */`** 1

The cross product $\overrightarrow{Z}_{C,AB} = \overrightarrow{AB} \times \overrightarrow{AC}$ (**`float Z_C_AB[3]`**) may be computed using an above defined function as: **`vecCP(AB, AC, Z_C_AB)`** **`;`** 1

The dot product $d_{P,C,AB} = \overrightarrow{Z}_{P,AB} \cdot \overrightarrow{Z}_{C,AB}$ (**`float d_Z_PC_AB`**) may be computed using an above defined function as: **`d_Z_PC_AB = vecDP(P, AB)`** **`;`** 1

The condition that P and C are on the same side of $\overline{AB}$ is: **`(`** **`d_Z_PC_AB >= 0`** **`)`** 1

Similarly, let $\overrightarrow{Z}_{B,AC} = \overrightarrow{AC} \times \overrightarrow{AB}$, $d_{P,B,AC} = \overrightarrow{Z}_{P,AC} \cdot \overrightarrow{Z}_{B,AC}$ (**`d_Z_PB_AC`**), $\overrightarrow{Z}_{A,BC} = \overrightarrow{BC} \times \overrightarrow{A}$ and $d_{P,A,BC} = \overrightarrow{Z}_{P,BC} \cdot \overrightarrow{Z}_{A,BC}$ (**`d_Z_PA_BC`**) be available.

Now the condition to determine whether P is inside ΔABC is:

```
( d_Z_PC_AB >= 0 &&  d_Z_PB_AC >= 0 &&  d_Z_PA_BC >= 0  )
```

3

2. Given a polynomial of degree n, $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x^1 + a_0$, $a_n \neq 0$, it can be rewritten as: $p(x) = (((a_n x + a_{n-1})x + ... + a_1)x + a_0$. This is the Horner's scheme for evaluating $p(x)$. The advantage of this method of evaluation is that explicit exponentiation is avoided. Let the coefficients for the various powers of x of $p(x)$ be stored in an array (say) **`P[]`**, as **`float P[]={`**a_n, a_{n-1}, ..., a_1, a_0**`}`**. An iterative function based on the above scheme is as follows:

```
hornerPoly(        float P[], int n, float x        ) {          1

   float sum=0; int i;                      // declarations     2

   for (           i=0; i<=n; i++           ) { // loop         2

        sum = sum * x + P[i];                                   2

   } // end of loop

   return sum;
}
```

A recursive function based on the above scheme is as follows:

```
hornerPoly(          float P[], int n, float x          ) {                1

      if (n==0) return P[0];                                               1

      return hornerPoly(P, n-1, x) * x + P[n];                             1

}
```

3. The digits of a given decimal integer n may be rotated right (e.g. 123 → 312) as follows: (*i*) let d be the least significant digit of n, let n' represent the number after d is removed from n (*ii*) let n' have k digits (k^{th} digit is non-zero) (*iii*) required result is $10^k d + n'$. Complete the function **`rotRight()`** given below, following the comments and computing $10^k d$ in steps as you determine k (rather than compute $10^k d$ separately).

```
int rotRight (int n)                    { // function declaration         1

   int d, nDash;        int t;         // any more declarations

   d = n % 10;                          // extract d                       1

   nDash = n / 10;                      // compute nDash                   1

   // stepwise computation of k and 10^k * d                               4

   for (t=nDash; t;) {

      d = d*10;

      t = t/10;

   }

   return d + nDash;

} // final result is computed and returned, in the last step               1
```

4. A perfect number is a positive integer n that is equal to the sum of its proper positive divisors (positive divisors excluding the number itself); e.g. 6=1+2+3 is a perfect number. It is only necessary to test whether numbers in the range $1..\lfloor\sqrt{n}\rfloor$ divide n (easily done without computing $\sqrt{n}$) to find the divisors; e.g. numbers in 1..5 are enough to find all divisors of 26. Complete the function **`isPerfect()`**, given next, following the comments.

```
int isPerfect (int n) { // return values: 1 if perfect, 0 otherwise

      int d=1, q, s=1;       // declarations with initialisations      2

   repeat { // stepwise computation of dvisors of n                    5

      d = d + 1;

      q = n / d;

      if ( q * d == n) s = s + d + q;

   ______________________________ // extra

   } until (        d >= q        );

   return n == s ;

} // final result is computed and returned, in the last step          1
```

5. (a) Representation of **`NaN`** in IEEE floating point 754 format is:

   `b' 11111111 bbbbbbbbbbbbbbbbbbbbbbb`, *at least one* **`b`** $\neq 0$. 2

   (b) Representation of ∞ in IEEE floating point 754 format is:

   `0 11111111 00000000000000000000000` 1

   (c) Representation of $(1.4)_{10}$ in IEEE floating point 754 format is:

   `0 01111111 01100110011001100110011` 3

   (d) Decimal value of the IEEE floating point 754 number **`0 10000101 10111100000000000000000`** is

   111 . 3