Computer Science 126 **Spring 1996**

Handout # 5: Solutions to First Midterm

These are model solutions. We also indicate how grading was done. If you wish to question your grade for any question, write down your question on a piece of paper, attach it to your exam, and hand it back to your class instructor.

Question 1: *Part a – 21 points*

```
-6.000000
-3.500000
4.000000
1.500000
```

Common mistakes were:

1. Not realizing the variable i in `main()` is distinct from the variable i inside `f()`.
2. Believing the printf in `f()` would not be executed.
3. Thinking the variable result and temp in `f()` were integers rather than floats.

Substantial partial credit was given if work was shown.

Part b – 12 points The following are the erroneous lines and the reasons why they are erroneous.

1. `return A[low] + A[high];` – A was not declared in sum.
2. `int A[n], B[n];` – invalid array declaration since n is a variable and not a constant.
3. `for(j = 0; j < n; j = j+1)` – j is not declared in `main`.
4. `B[j] = sum(1,j);` – B is not a validly declared array and j is not declared.
5. `charprint(B)` – B[] is an array and the argument of `charprint` has to be of type `char`. Also, B is not a validly declared array.

A common mistake was to over-estimate the number of errors. One point was deducted for each non-error. There was significant confusion on what constituted a syntax error. Significant partial credit was awarded for answers that were almost correct.

Question 2: The program below is one solution to problem 2. A perfect solution earns 33 points. Nearly perfect programs receive 29-33 points. Programs with serious flaws but which are still *near* to a solution, are assigned 24-28 points. Given multiple major flaws, but a program which is in some sense reasonable, 21-23 points are assigned. Lower scores are assigned to programs which are nowhere near to a solution. There were no points deducted for style. A single minor syntax error causes no point deduction. A few minor syntax errors cost a single point. Some common errors are:

- 2pts Failure to reset the sum accumulator inside the outer for loop, or failure to initialize it at all.

- 2pts Printed number of perfect numbers, rather than the numbers themselves.
- 2pts Use of floats where ints should have been used.
- 4pts Checked only if "m" if perfect, i.e. missing outer loop.
- 1pt Printed wrong variable.
- 1pt Missing "&" in scanf().
- 1pt Compared sum with value rather than twice value.

Some of the point deductions above are approximate and may be influenced by other factors. Also, the list is incomplete and cannot describe all problems discovered – particularly with programs which received fewer than 28 points. Another common error involved the use of arrays in the solution. The instructions said nothing about a limit on the size of "m", so programs which allocated a fixed length array did not strictly satisfy the question. In several cases, an attempt was made to allocate an array by a declaration such as "int A[i]". This does not work in C. One must use malloc() to accomplish this.

```
#include <stdio.h>

void main(void)
{
    int m, n, divisor, sum;

    scanf("%d", &m);

    for (n = 2; n <= m; ++n) {
        for (divisor = 1, sum = 0; divisor <= n; ++divisor)
            if (n % divisor == 0)
              sum = sum + divisor;
        if (sum == 2*n)
          printf("%d\n", n);
    }
}
```

Question 3: *Part a (14 points)*

```
        Memory Location     Value
        ---------------     -----
              40:            0002    value 2
              41:            0003    value 3
              42:            0001    value 1
              43:            0000    value 0
              44:            0000    value 0
```

```
This program compares the contents of memory locations 40 and 41
and if M[40] >= M[41], then it swaps the contents of the locations.
```

Grading:

- 2 points for each correct memory value (total of 10 points)
- 4 points for showing work (if you didn't show any work, but got the correct answer, you got the 4 points anyways)

Part b (20 points)

```
Memory Location     Value
---------------     -----
      40:            0002    value 2
      41:            0001    value 1
      42:            0000    value 0
      43:            0000    value 0
      44:            0003    value 3

This program implements the first pass of a ``bubble sort," in which
consecutive memory locations are compared and the contents swapped
if the value in the lower memory address is greater than or equal to
the value in the higher address.  The end result is that the largest
value in the array is ``pushed'' to location 44.
```

Grading:

- 2 points for each correct memory value (total of 10 points)
- 6 points for correct program description. In order to get full credit, you needed to mention that comparisons were being made on consecutive memory addresses, or that a partial sort was being performed, in which the largest element is pushed to the end of the array. If you said that the value of M[40] was pushed to M[44] without mentioning sorting or comparisons, you got half credit. (You didn't need to use the term *bubble sort* to get full credit.)
- 4 points for showing work (if you didn't show any work, but got the correct answer, you got the 4 points anyways)