CSC 180 midterm #1 solutions

17 October 2001

1. [5 marks] Write a boolean expression which states that (i.e. is true exactly when) *x* is positive *and y* is between 3 and 13 inclusive.

Answer: `x > 0 && 3 <= y && y <= 13`
(no penalty for using "≤")

2. [5 marks] Explain in a sentence or two what the following program does, including explaining when it stops reading. (Your explanation should be in terms of behaviour visible to the user; for example, it should not mention x and y.)

```
#include <stdio.h>

int main()
{
    int x, y;

    y = 0;
    while (scanf("%d", &x) == 1 && x > 0)
        y = y + x;
    printf("%d\n", y);

    return 0;
}
```

Answer: It reads a sequence of integers from the standard input, until it hits end of file, non-numeric input, or an integer ≤ 0. It then outputs the total of all the values in the list (not including any final negative integer value).

3. [10 marks] Write a complete C program which takes one integer as input (prompting is not necessary for this test question) and outputs either "positive", "zero", or "negative", whichever is true of the input number. You do not need to include any comments but you must format your program in the standard style, declare all types appropriately, etc. You can assume that the scanf() succeeds.

Answer:

```
#include <stdio.h>

int main()
{
    int x;
    scanf("%d", &x);
    if (x > 0)
        printf("positive\n");
    else if (x == 0)
        printf("zero\n");
    else
        printf("negative\n");
    return 0;
}
```

(continued)

4. [15 marks] If you type ‘‘10’’ on a calculator and then press the square-root key, you get the square root of 10. If you keep pressing the square root key, the number in the display gets smaller and smaller, but more and more slowly, because the square root of any number greater than one is also greater than one. Since the calculator is not completely accurate, as you get very close to 1, eventually you get to a number which is its own square root, to the available precision of the machine. (For some calculators this is 1; for others this may be a number more like 1.0000001.)

Write a complete C program to perform this process using the sqrt math library function (in a loop) and to output the number we stabilize at.

Answer:

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

int main()
{
    double x = 10;
    double next;

    while (x != (next = sqrt(x)))
        x = next;
    printf("%g\n", x);

    return 0;
}
```

A simpler answer says instead

```
    while (x != sqrt(x))
        x = sqrt(x);
```

but this calls sqrt nearly twice as many times (one fewer than twice).

If you actually want to run this program, you’ll want to use something more like %.25f, because %g will always say 1 because it won’t show enough significant digits.

5. [15 marks] A ‘‘perfect’’ number is a number which is equal to the sum of its divisors not including itself. The positive integer *d* is a divisor of the positive integer *x* if and only if *x* % *d* == 0.

Write a function which takes one int parameter and returns a value of type int. The function is called ‘‘sum_of_divisors’’. It returns the sum of all of the divisors of the parameter, except for the parameter itself. The parameter must be a positive integer, and you do not have to check this.

For example, sum_of_divisors(3) will return 1, because the only divisors of 3 are 1 and 3. In fact, sum_of_divisors will return 1 for any parameter which is prime.

sum_of_divisors(6) will return 6, because the divisors of 6 are 1, 2, 3, and 6, so it returns 1+2+3 which is 6. Thus, 6 is a perfect number (in case you were wondering what the connection is).

(continued)

Add your function after the following main() which uses it to output a list of the perfect numbers between 1 and 100, inclusive.

Modify the main() function where necessary, if necessary.

```
#include <stdio.h>

int main()
{
    int i;


    printf("Perfect numbers between 1 and 100 (inclusive):\n");
    for (i = 1; i <= 100; i++)
        if (sum_of_divisors(i) == i)
            printf("%d\n", i);
    return 0;
}
```

Answer:

• you had to add `extern int sum_of_divisors(int x);` to main() (or before it)

```
int sum_of_divisors(int x)
{
    int d, sum = 0;
    for (d = 1; d < x; d++)
        if (x % d == 0)
            sum += d;
    return sum;
}
```