This tutorial covers numerical issues that relate to Assignment 1, Boolean expressions, and loops.

**Numerical issues in Assignment 1**

Assignment 1 is now posted on Blackboard. The assignment asks the students to write a program to simulate a cash register. Among others, the students need to implement these functions:

- **void purchase(int q, double p);**
  Purchase q items, each with price p (in CA dollars).
- **void cancel(int q, double p);**
  Cancel q items, each with price p. cancel() would work even if you try to cancel items that haven’t been bought yet. However, note that cancel() must not make the number of items purchased negative, nor make subtotal price negative. (See Section 4 for more information.) Any attempts to violate this requirement will have no effects.

As discussed in Section 4, there are issues with using *double* to store money.

**4 Handling round-off errors**

Floating-point numbers of type *double* are not exact. Allowing any double to be a price leads to unavoidable problems with round-off errors. For example, if a customer purchases 20 items one by one each costing $1.31, and then cancels them all at once, he gets a negative number. If a customer purchases 99 items at the same price and cancels them one by one, he again gets a negative number. In such situations, cancellation can still be made. Failure to handle these situations will result in losing 5% of the total mark for this assignment.

We would like to explain two issues

- Error accumulation
- How to do comparison of doubles

Wael’s students haven’t seen binary arithmetic yet, so we should try to elide this issue.

**Rounding doubles**

double x = 3.1416;
int three = (int)x;

- (int)x always rounds down
• To round $x$ to the nearest integer, use $(\text{int})(x+0.5)$
• 1.5 is rounded **up**
• Rounding to the nearest 1/10th: $((\text{int})(10\times x+0.5))/10.0$

**Representation of doubles**

For reference, here’s the IEEE 754 standard for double, in case they ask:

<table>
<thead>
<tr>
<th>exponent (11 bit)</th>
<th>fraction (52 bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>52</td>
</tr>
</tbody>
</table>

We’ll be explaining this with a smaller format. The format is as follows:

• 3 decimal digits for the mantissa ($x$), with the decimal point after the first digit
• 1 decimal digit for exponent ($y$)
• The number is $x \times 10^y$

**Accumulation of error and an example of using the floating point format**

Now we point out that we cannot present numbers like 1/3 exactly.

$1/3 \approx 0.33e0$

$1/3+1/3+\ldots+1/3 \approx 300\times 0.33 = 9.90e1 = 99 \text{ (but } 300\times (1/3) = 100 \text{)}$

(300 times)

There is actually not much to be done with just 3 figures in the mantissa, but suppose we have 5 figures, and we know that we are adding numbers of the form $n/3$, where $n$ is an integer. Then we can round to the nearest third each time we add.

• $1/100$ behaves similarly in binary to $1/3$ in decimal!
• In the assignment, we know that all prices are of the form $n/100$!

**Comparison of doubles**

• Truncation errors occur
• Because of this, for example, if a customer purchases 20 items one by one each costing $1.31$, and then cancels them all at once, they get a negative number.
• Use $\text{fabs}(x-y) < \epsilon$ for a suitable $\epsilon$ (ask: what’s a suitable $\epsilon$ for the assignment?) rather than $x == y$
Boolean expressions practice questions (taken from Alan J. Rosenthal’s notes)

(1 + 1) == 3 is 0
((1 + 2) == 4) || ((1 + 1) == 2) is 1
((1 + 2) == 4) || (((1 + 1) == 2) && ((2 + 2) == 4))) is 1

Loops

Write a function which returns 1 if the parameter n is a prime number, and 0 if it isn’t

int isprime(int n)
{
    int d;
    for(d = 2; d < n; d++){
        if (n % d == 0)
            return 0;
    }
    return 1;
}

• I suggest tracing the code for n = 303, n = 5
• Explain the role of return in breaking the loop: if we return 0, we never get to return 1; if we haven’t returned 0 and the loop is done, that means we must return 1

A better algorithm: if n is even and > 2, return 0 immediately. Then we can increment d by 2 at a time: for(d = 3; d < n; d+=2)

The same thing with a while loop:

int isprime(int n)
{
    int d;
    if((n>2)&&(n%2 == 0))
        return 0;
    d = 3;
    while(d < n){
        if(n %d == 0)
            return 0;
        d+=2;
    }
    return 1;
}

(aside, if there’s time: an even better solution: only check for d < sqrt(n)+1)
Nested loops

Suppose we aren’t allowed to use the division and mod operators. Here is a naïve way of checking whether n is a primer:

```c
int isprime_naive(int n)
{
    int a, b;
    for(a = 2; a < n; a++){
        for(b = 2; b < n; b++){
            if(a*b == n)
                return 0;
        }
    }
    return 1;
}
```