Math 211 Final
May 8, 1998
Professor L. Taylor

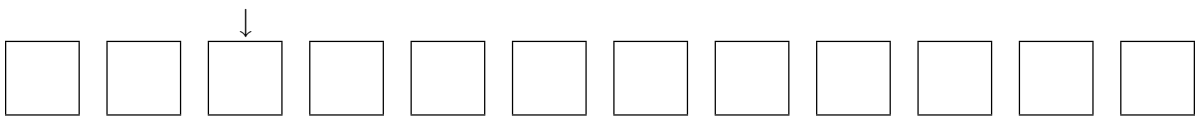
Problem 1. A program with the standard main declaration

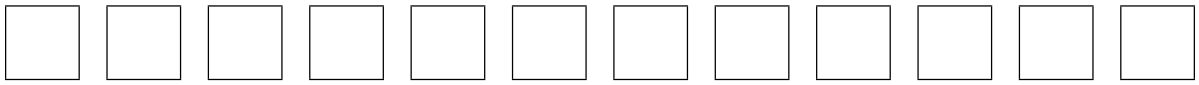
```
main(int argc, char *argv[]) {
    ⋮
    }
```

was compiled to produce `a.out`. The program was run with command line

```
a.out -kim 1 5.0 aardvark
```

What was in `argv[3]`? More explicitly, suppose the boxes below represent a collection of consecutive bytes of memory in the computer and that `argv[3]` points to the box with the arrow above it. Fill in the remaining boxes. Put a question mark (?) in any box whose contents are unknown given the data. Put your final answer in the top row of boxes, but you may use the lower row to experiment before committing yourself.





Problem 2. Suppose you are working in a programming language which has an "if" statement with the following syntax:

```
if[ test, false-expression, true-expression]
```

where `test` is an expression which evaluates `true` or `false`. If `test` is `true` we do the `true-expression`, otherwise we do the `false-expression`. Suppose we have the following problem. We have two tests, `test1` and `test2` and three expressions, `exp1`, `exp2` and `exp3`. How would we write an "else if" expression in this language? Explicitly, write an expression which will do `exp1` if `test1` is true and will do `exp2` if `test1` is false but `test2` is true and will do `exp3` if both tests are false.

Problem 3. Here is a very short program.

```
main()
{
short i;
i=12;
printf("%d\n",&i);
}
```

I compiled it and ran the `a.out` and got the answer

-268437934

What's going on? Address two points in your answer. Why didn't I get 12 and what is -268437934?

Problem 4. C has several pitfalls, among them being that unspecified details are left to the people writing the compiler and many people feel that C does not specify enough. An example of "compiler dependence" is that the order of evaluation of the arguments of a function are unspecified: the compiler may do them in any order it chooses. This leads to mistakes when trying to be clever. As an example, suppose

```
short ff(short, short);
```

is a declaration and suppose `ix` and `iy` are shorts. Suppose `ix` has the value 5 when the line.

```
iy=ff(ix++,ix);
```

is encountered. After this line `ix` is 6, but `iy` will be `ff(5,5)` if the second expression is evaluated first or else `ff(5,6)` if the first expression is evaluated first. This leads to programs which work correctly when compiled with one compiler but work wrong when compiled with another. Discuss the various possibilities for

```
iy=ff(ix++,--ix);
```

assuming that `ix` has the value 7.

(a) If the first expression is evaluated first,

(b) If the second expression is evaluated first,

Problem 5. Recall Pascal's triangle for evaluating binomial coefficients. It $\binom{n}{k}$ is defined to make the binomial theorem work,

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$

for positive integers n. Inductively, $\binom{n}{0} = \binom{n}{n} = 1$; $\binom{n}{1} = \binom{n}{n-1} = n$ and $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ for $1 < k < n-1$. Given the declaration

```
long comb(short n, short k);
```

write code for `comb` so that `comb(n,k)` recursively calculates $\binom{n}{k}$.

```
long comb(short n, short k) {/* Your code here */
```