Name: _____

Directions: **Work only on this sheet** (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing. In order to get full credit, **SHOW YOUR WORK**.

1. This problem concerns the Curses PLN.

(a) (10) What member variable of the **curses** class stores the number of rows in the window?

The remaining parts of this problem concern the **psax** program.

(b) (10) The window will initially be blank. Give the number of the line of code at which the window ceases to be all blank.

(c) (10) Give a function call that could be inserted into **psax** in order to move the highlighting to the top row of the window.

2. (5) Fill in the blank with an official term from our course: Consider the port scanner example. We could have each thread check b consecutive ports, rather than just 1, for better efficiency due to reduced thread startup overhead. But if we make b too large, we will likely have a _____ problem.

3. (20) I wrote a program that I use to produce roll sheets for each class I teach. It inputs the student list I receive from the Registrar, and outputs the same list but with only some information retained. Specifically, the input variables are number on the roll sheet (1, 2, 3, ...), student ID, surname, first name, middle name if any, class (FR, SO, JR, SR, etc.), major (ECSE, LCSI, etc.), enrollment status (RE for enrolled, WL for wait list, etc.) and UCD e-mail address (e.g. jjones). I retain just the last name, first name, middle name if any, class level and major. The input file name is the first command line argument. The output file name will have the form GradesXXX.txt, e.g GradesS07.txt; the XXX part is the second command line argument. Fill in the blanks in my code:

```
import _____________
infilename = open(sys.argv[1])
outfilename = _____________________
g = open(_____________________)
for l in infilename:
   w = l.split()
   del w[len(w)_________________]
   del w[:2]
   g.________________
```

4. (15) Suppose we wish to be able to sort strings, with the sort criterion being dictionary order within length. In other words, the primary sort criterion is string length, and the secondary criterion is ordinary string ordering. For example, 'eas' would be considered > 'et' while the latter would be < 'eu'. Fill in the blanks for the function **llcmp()**, to be used as an argument to **sort()** in the **list** class:

```
def llcmp(x,y):
   if x == y: return 0
   elif len(x) != len(y): return ____________________________
   elif ____________________________: ____________________________
   return 1
```

Note: The member functions **__lt__()**, **__le__()**, etc. in the **string** class do the ordinary compare, in which for instance 'eas' < 'eb'.

5. (10) Our program will generate a random matrix. The command-line arguments consist of the number of rows and columns. Fill in the blank:

```
(rows,cols) = ________________________________________________
```

6. (20) This problem deals with our homework program which implemented the A Priori algorithm for rule finding in data mining. Recall that the rules were stored in a dictionary, with the keys being tuples of the form (antecedents, consequent) and the values being tuples of the form (support, confidence). For instance, if the value for the key (0,2,7) is (0.24, 0.66), then the rule 0,2 => 7 has support 0.24 and confidence 0.66. Say the variable for our dictionary is **ruled**.

Say we also have a dictionary **freqd** containing all itemsets with frequency at least equal to **minsupp** $\times$ **minconf**, the product of the minimum support and confidence levels. The keys are tuples for the itemsets and the values are the support levels. So for example if we have a key (3,4) with value 0.51, then 0.51 of the records in the database contain both items 3 and 4.

But support and confidence may not be the only criteria which are of practical value. Suppose for instance the singleton (7) has support (0.60). Then knowing that 0 and 2 occurred makes 7 only slightly more likely to occur than if we don't know whether 0 and 2 occurred. The likelihood ratio here is $0.66/0.60 = 1.10$. This is called the *lift*.

The code below will compute a list **z** of all rules in **ruled** that have lift at least equal to **minlift**.

```
z = []
for (rule,suppconf) in ruled:
   if ______________________________:
      z.append(rule)
```

1.a curses.LINES

1.b 65 (when **refresh()** is called)

1.c updown(-gb.winrow

2. load balancing

3.

```
sys
Grades + sys.argv[2] + '.txt'
outfilename, 'w'
-2:
write(outfilename,' '.join(w) + '\n'
```

4.

```
len(x) - len(y)
x.__lt__(y): return -1
```

5.

```
(int(sysargv[1]), int(sysargv[2]))
```

or

```
map(int,sysargv[1:])
```

6.

```
suppconf[1] >= minlift * freqd[rule[-1]:]
```