CS113 Assignment 3 — Memory management and File I/O, Spring 2008
Due Wednesday February 20, 2008, 11:59:59PM

Solve <u>one</u> of the following two problems. You may work with a partner if you wish.

1 People database

The U.S. Internal Revenue Service reportedly uses a 1960's-era computer system for processing income tax returns. The system is difficult to maintain because of aging equipment and software written in COBOL and assembly language. Two attempts at upgrading the system were made during the 1980's and 1990's, but both ended in failure after costing several billion dollars. The IRS is in the midst of a 10-year project to try again, this time with a budget of over $8 billion.

Suppose that you've been sub-contracted to write part of the new system: a database that stores the ID number, name, and income for each taxpayer. The program should be an interactive console application that accepts commands and then responds accordingly. In particular, the IRS requires the following commands:

- `add`: Add a taxpayer to the database.
- `list`: Display the ID, name, and income for each taxpayer in the database. The list should be sorted by ID.
- `changename`: Change a taxpayer's name.
- `changeincome`: Change a taxpayer's income.
- `stats`: Display the total number of taxpayers as well as the minimum, maximum, and mean income.
- `clear`: Remove all taxpayers from the database.
- `save`: Save the database to a file called `database.txt`.
- `load`: Clear the database, then load the data stored in file `database.txt` into memory.
- `quit`: Exit the program.

Here is what a sample execution of the program might look like (your program need not match this format exactly):

```
Welcome to the IRS!
Enter a command at the prompt.

> add
Enter a taxpayer ID: 123456
Enter the name: May O. Naise
Enter the income: 10400

> add
Enter a taxpayer ID: 35324
Enter the name: Lucy Tax
Enter the income: 30200

> stats
Number of taxpayers: 2
     Minimum income: 10400
     Maximum income: 30200
```

```
        Mean income: 20300

> changename
Enter a taxpayer ID: 1000
Error: taxpayer not found in database.

> changename
Enter a taxpayer ID: 35324
Enter new name: Lucy Goosey

> list
ID             Income          Name
----------     -----------     ------------------------
35324               30200      Lucy Goosey
123456              10400      May O. Naise

> quit
```

The file format for the `load` and `save` operations should be as follows. The first line of the file indicates the number of taxpayers in the database. Then the database contents are stored with each field on a separate line. For example:

```
2
123456
May O. Naise
10400
35324
Lucy Goosey
30200
```

Your program should check for the following situations and display helpful error messages accordingly:

1. An unknown command is entered.
2. The user tries to add a taxpayer ID that is already in the database.
3. The user tries to change a taxpayer ID that is not in the database.

Checking for other errors is optional. You may assume that both the taxpayer IDs and incomes are integers in the range $[1, 1000000000)$.[1] Further assume that no taxpayer has a name longer than 128 characters. However, **you may not assume an upper bound on the number of taxpayers in the database**.

Write a C program to implement this database application. Submit your solution as `irs.c`.

Hints: You'll probably want to design a `struct` to hold the data associated with each taxpayer. The database can be stored as an array of these structures. Since C arrays cannot be resized, you'll have to do the resizing explicitly. That is, whenever a new taxpayer is added to a database of n records, the program will have to allocate a new array of size $n + 1$, copy the n records from the old array to the new one, and then deallocate the old array. Remember `free()` every buffer you `malloc()`. Alternatively, you could store the database in a linked list to avoid the array resizing issue.

2 Computing π

The constant pi, the ratio of the circumference of a circle to its diameter, has fascinated scientists and mathematicians for thousands of years. Evidence suggests that the ancient Babylonians first studied pi

[1] The IRS's system has some hard-coded assumptions as well. Supposedly Bill Gates' tax returns must be processed specially because they otherwise would crash the IRS's software.

nearly 4,000 years ago and had an approximation accurate to two significant digits. Archimedes found the third digit of pi in the third century BC. The Chinese astronomer Zu Chongzhi had computed π to seven decimal digits by 500 AD. An English mathematician named William Shanks devoted much of his life to computing π by hand, publishing the first 707 digits in 1873. (In 1944 it was discovered that he had made an error in the 528th digit which made the rest of his calculations incorrect.) Computers have made computing pi much more practical. The millionth digit was computed in the 1940s on an early Navy computer, the billionth digit was computed in 1989 by the Chudnovsky brothers on a homemade supercomputer, and the trillionth digit was computed in 2002 at the University of Tokyo.

There are many algorithms for computing pi. Here is an iterative one discovered by Ramanujan in the early 20th century:

1. Let $x = 0.5$, $y = \sqrt{2}$, and $i = 1$.
2. Compute $\frac{1-\sqrt{1-y^2}}{1+\sqrt{1-y^2}}$, and store the result in y.
3. Compute $x(1+y)^2 - 2^i y$, and store the result in x.
4. Let $p = \frac{1}{x}$ and $i = i + 1$.
5. Goto step 2.

As the iteration progresses, p converges to π. Here is a C program that implements the algorithm:

```
int main()
{
  double x=0, y=0, p;

  y = sqrt(0.5); x = 0.5;

  for(int i=1; i<20; i++)
    {
      y = (1-sqrt(1-y*y)) / (1 + sqrt(1-y*y));
      x = (pow((1+y),2)*x)-pow(2, i)*y;
      p = 1.0/x;

      printf("As of iteration %d, pi= %lf\n", i, p);
    }

  return 0;
}
```

The problem with this code is that C's `double` type only has a precision of about 15 decimal digits on most machines. So the above code can only give an approximation of π accurate to about 13 or 14 digits, no matter how many times it iterates. Unfortunately, C does not have a floating-point type more precise than `double`.

An alternative to using the built-in floating point types is to store numbers as strings. For example, instead of storing the number 3.1415 in a variable of type `double`, one could store it as the string `"3.1415"`. The advantage is that the number can be stored as precisely as you'd like. The only limitation is the amount of memory on the machine. For a modern system with 1 gigabyte of memory, for example, you could store a string with a billion characters, which means a number precise to over a billion decimal digits!

Of course, C doesn't have built-in arithmetic operators that handle numbers stored in this way, so we have to implement our own. However implementing the operations is conceptually not difficult. We can use the techniques taught in grade school to implement addition, subtraction, multiplication, and long division. For example, the routine to add two numbers would process the strings right-to-left, computing the sum of each column and carrying over to the next column if necessary:

$$\begin{array}{ccccccc}
 & 1 & \overset{1}{2} & . & 5 & 3 & 8 \\
+ & & 1 & . & 5 & 2 & 1 \\
\hline
 & 1 & 4 & . & 0 & 5 & 9
\end{array}$$

The result of adding the string `"12.538"` to the string `"1.521"` would be `"14.059"`.

Write C functions to perform addition, multiplication, subtraction, and division on arbitrary-length numbers stored as strings. Then write a program that uses those function to compute pi to an arbitrary number of decimal digits using Ramanujan's algorithm above. Submit your solution as `pi.c`.

Hint: You'll also need to write a square root function. There are several algorithms for computing the square root of a positive number n, but perhaps the simplest is the Babylonian method:

1. Initialize x to 1.0.
2. Let y equal $\frac{1}{2}\left(x + \frac{n}{x}\right)$.
3. If $x = y$ stop; otherwise let x equal y and then goto step 2.

3 What to submit

Submit two of the following three source files:

1. `arraylist.c`
2. `unique.c`
3. `pi.c`