INSTRUCTOR TOOLS: TEST DATA GENERATION

Curt Hill
Department of Mathematics and Computer Science
Valley City State University
Valley City, North Dakota 58072
Curt.Hill@vcsu.edu

Abstract

This paper considers the class of programs that generate test data. These range widely from freeware to sophisticated commercial programs. The data to be generated must be specified in some way. This could be by manual input of the characteristics such as type, length, ranges or it may be automatically extracted from a database. The output can also be sent to a database or stored in flat files.

The ease of generation of two such programs is shown. The case-studies of two assignments given to computer science classes is all considered. The references cite locations of many such programs.

Introduction

The creation of computer software requires some assurance that the software is reasonably correct. Without disparaging formal verification techniques, this is usually through the use of test data. The need is felt both on campuses and corporations. Only the simplest programs in the introductory class have no input data. There are interesting programs with very simple data, but there is also a need for quantities of test data residing in files. These files could be either flat files or within a database. Therefore, students, instructors and professional developers are in need of reasonable data to test programs.

There are typically three ways to produce this kind of data: manual production of test data, extraction and modification of live data and test data generation programs.

For many programs in the CS 1 or CS 2 courses the data may be fabricated by hand using a simple text editor. Choosing data in this way allows good testing for any cases that the specification (black box testing) or the program itself (white box testing) may suggest are interesting. However, this is an approach that becomes quite onerous when there is a requirement for large quantities of data. When the amount of data is required to be large, the other two techniques are generally employed. However, that data will often be enhanced with hand generated examples to be sure that all suggested cases are covered.

Many production systems use a scheme where the live data is processed by program to give a smaller set of data for the use of testing. (The smaller set could still be much larger than one would like to generate by hand.) There are some dangers in this approach. Privacy or security may be compromised, unless the data is altered in the extraction process. This approach leaves much to be desired in the implementation phase of a large project, since there is not yet data established in the correct format. Instructors seldom have access to this kind of data, since they do not maintain production systems.

Test data generators are another attractive option and many examples exist. These range from freeware to commercial quality programs with a price to match. These generators are driven by user provided specifications that describe the form, range and quantity of the generated data.

Why does an instructor need a test data generator? Consider three scenarios from three separate Computer Science classes.

In any introductory programming class there is the typical program assignment: read a file of given specification and do something appropriate. Something appropriate could include any of the following: count certain items, sum certain values, or produce some kind of report. In most cases the course text book may supply some data for just such a program. However, that also implies that the student may be able to find an existing program that satisfies the assignment on the web or from previous students. Therefore it behooves the instructor to come up with unique format each year. The program specification may be nearly the same as last year’s provided the format of the data is sufficiently different. The data structures class provides a similar situation, except the data is often to be stored in some type of container class, such as a list or tree. The database class provides yet another opportunity. The goal is often to provide the students with usable tables to manipulate.

Specifying Test Data

A test data generator should be able to produce data in a large variety of formats. A record should be able to contain many fields and each of these may have a large variety of forms. The user may specify these fields in a variety of ways, such as through a graphical user interface, command line or saved specifications. What kinds of characteristics might one be able to specify?

First and foremost the type of the field would need to be specified. The usual types such as a variety of numerics are generally available. However, type has a different connotation in test data generators than it might in a programming language. In a programming language, a string type might be used for person names or product names. The program logic will assure that the right kind of data is inserted into each. A data generator must produce the correct values and names are not random assortments of letters. Thus in the realm of types first name is usually distinct from product name. One of the marks of better test data generator is a large dictionary of possible names, as well as a large variety of usable types to represent names and other non-random strings.

Numeric fields provide their own features of interest. The ability to supply a range is essential. This range is often less than what the length of the field might support. Less necessary is a choice of the probability distribution. The default is a uniform distribution, however this often makes for extraordinary data. Other possible choices for a distribution function might include Normal or Poisson.

Dates provide a different challenge. In the same way as names, a date is not six randomly generated digits, since there are real restrictions on valid dates. Moreover, it is generally unacceptable for all days to be in the 1 to 28 range. To complicate matters further, there are many different ways to specify a date. Should the month be first or the day first? Should the year have four digits or two? Is the month numeric, spelled out completely or abbreviated to three characters? In addition, dates may have a range just like the numbers have a range. The better data generators provide more flexibility.

Other possible types exist, such as Booleans, enumerations and constant data. A Boolean may allow the flexibility to be one of the common forms such as T or F and Y or N, both those in either upper or lower case. An enumeration is just a series of constant strings given by the user, that are chosen at random for the data. The Boolean could be accomplished this way, as could product names or company divisions. A constant field may be used to provide a particular delimiter to variable length fields, provide a decimal point between two numbers for programs that do not generate monetary amounts, or wrap control information around the data, such as an SQL insert statement.

The field length is often required, but may be inferred. In a file where each field may vary in length the value generated determines its own length. This will not be the case for fixed length fields, in which case the length needs to given for each field that could vary in size. In fixed length fields the padding needs to be considered as well. A string will

pad with blanks, but numbers may have leading blanks or leading zeros. The ability to specify the types of padding is an asset.

Fields may need some relationship to one another. For example if there is a quantity and price field, it would be desirable to have a calculated field which is the cost. This could be rather complicated and exceeds many programs.

The specifications described above are typically given by the user. However, there is nothing given above which could not extracted from a database schema. Since the need and complexity for generating data base tables is great there are a number of programs that do exactly that. The user directs them to a database and the program finds the specifications it needs, including referential and domain integrity constraints and populates the tables accordingly. This may be from generating source SQL statements as a flat file or sending them to the database directly.

Most test data generators allow a specification based approach. There are several examples [2,3,5,6] that can examine a database and obtain the specifications.

First Example

The following program[1] is free and will suffice for many cases. It is implemented in JavaScript within web pages, so will work on any platform. It may also be downloaded. The example given here is the data needed for the first case study. Figure 1 shows data about to be generated which would contain four fields. This was to be a list of felons containing names, zip code, crime and whether they were in prison currently.

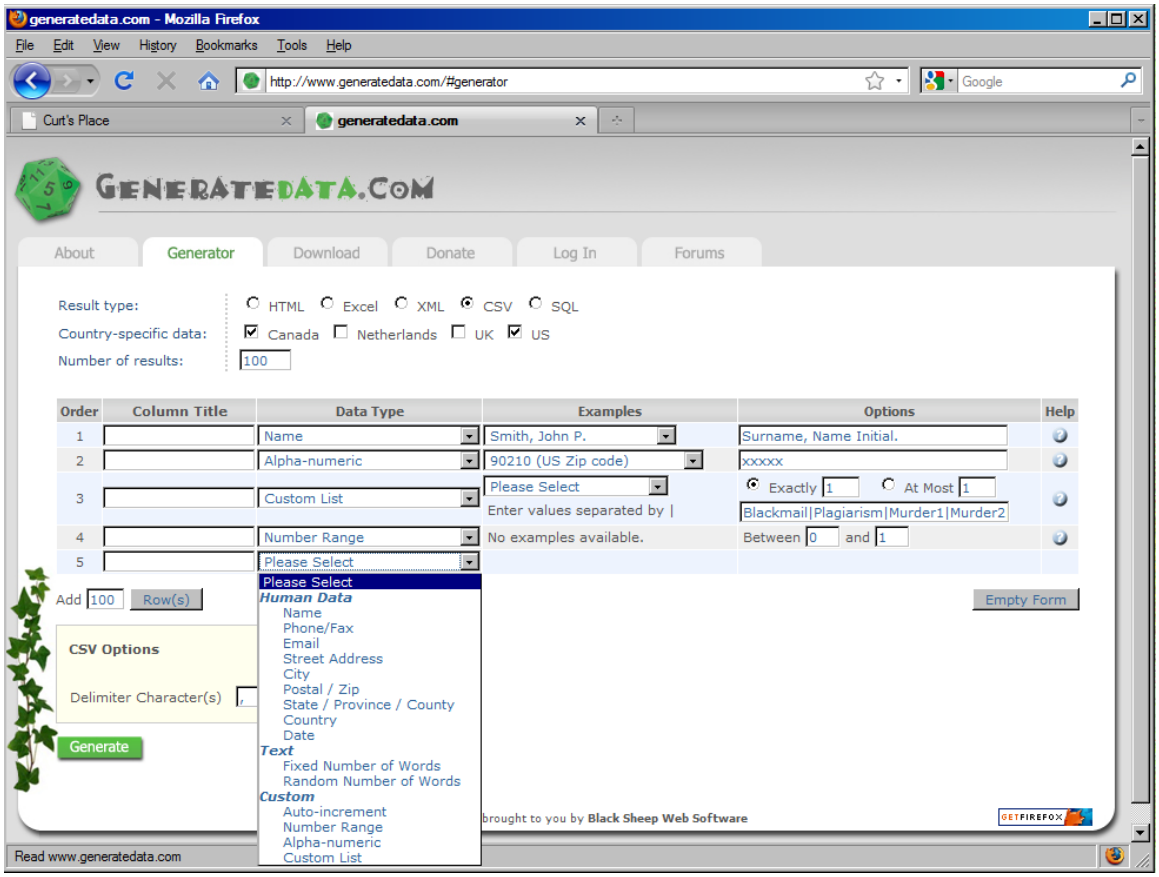

Figure 1 Types available

In Figure 1 the fifth field is currently selected showing the possible types of data that may be generated. The first field is a name of the human data category. The possible options for a name are shown in Figure 2. The second field is a zip code. The third field is an enumeration of possible crimes. These are merely entered separated by the "|" character. The fourth field functions as a Boolean, with the possibilities just zero or one.

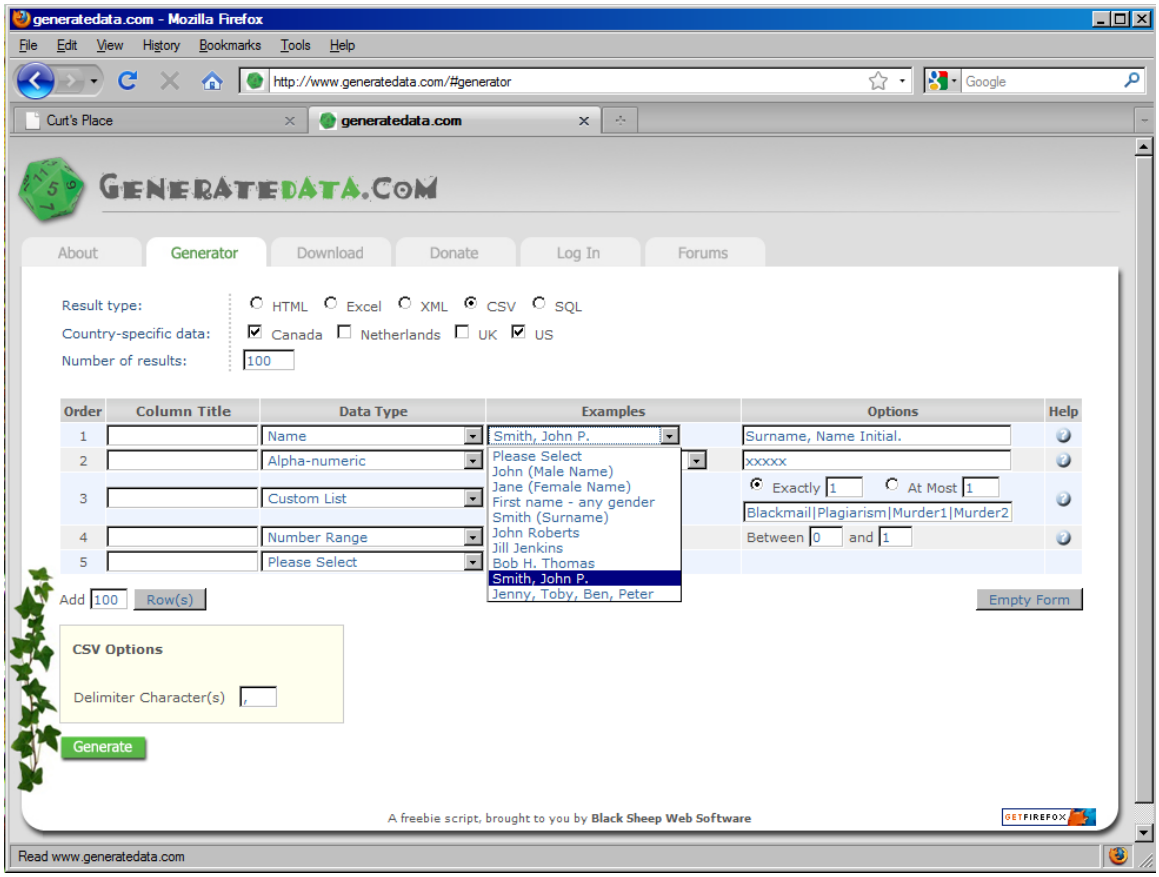

Figure 2 Possible formats for the name

Once the fields are properly set, the Generate button is clicked. If this is being used from a web page a download is generated.

Second Example

The following program[4] is somewhat more complicated but also free. It should run on any Windows platform. Figure 3 shows the start of the generation of file from case study 2. The first field is an employee number. This number is generated in ascending order, with random size gaps between successive employees.

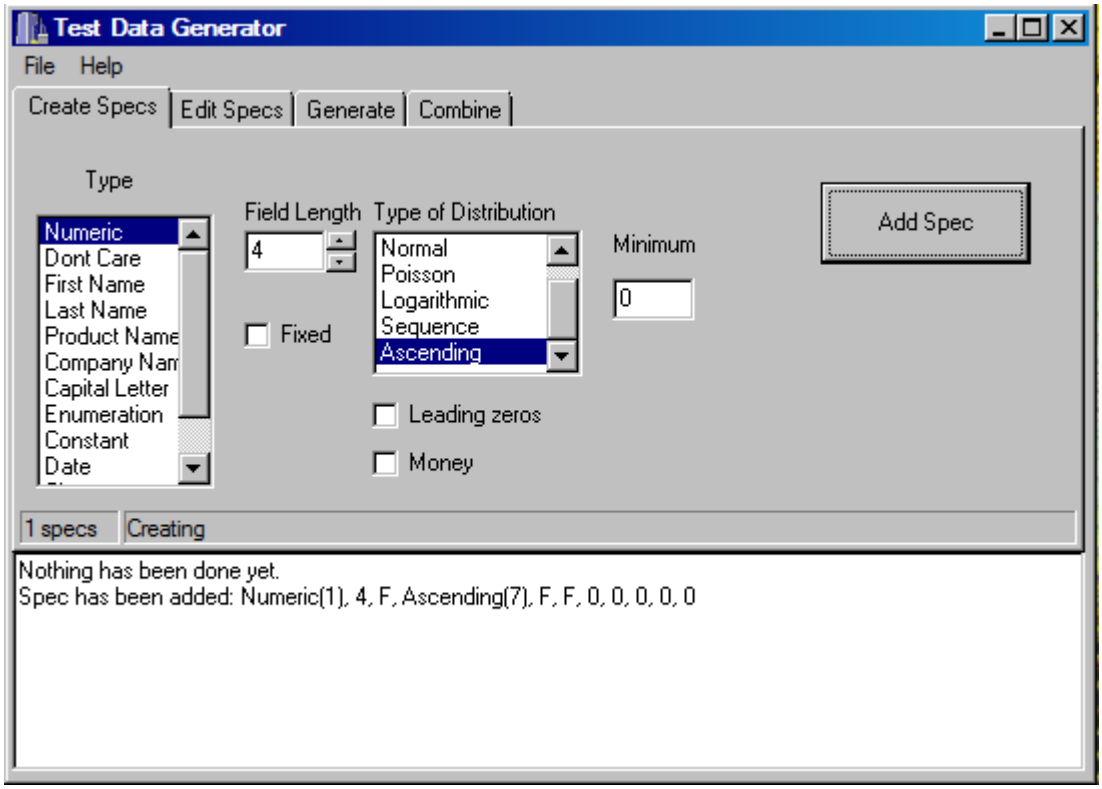

Figure 3 Generating an employee number

Once the proper number of fields has been produced. The user may edit the specifications that have been entered. Figure 4 shows this tab with four fields, the employee number, last name, first name and a middle initial.

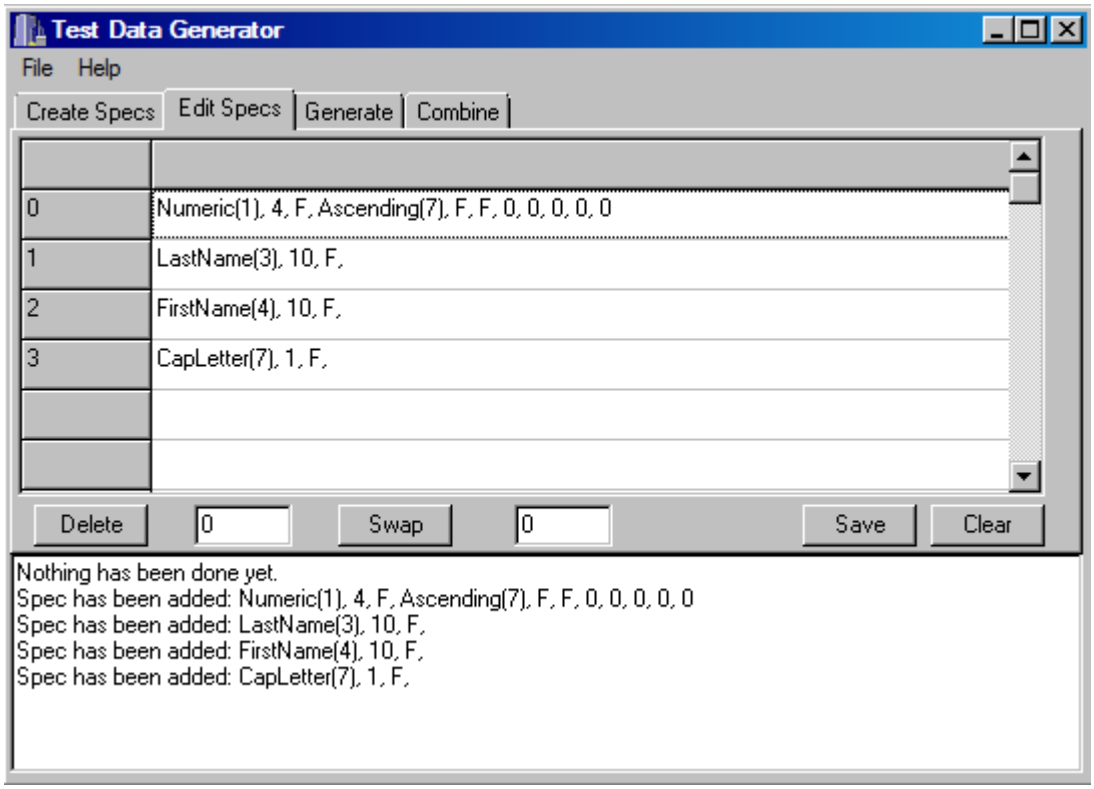

Figure 4 Examining specifications

Perhaps editing is a misnomer. The order of the field specifications may be altered or specifications may be deleted. However, the user may switch between this and the previous view any number of times. New specifications are always appended onto the list of specifications. The specifications may also be saved into a file for later use.

At some point the user will be satisfied with the set of field specifications. They then move onto the actual generation process. Figure 4 shows how this could be processed.

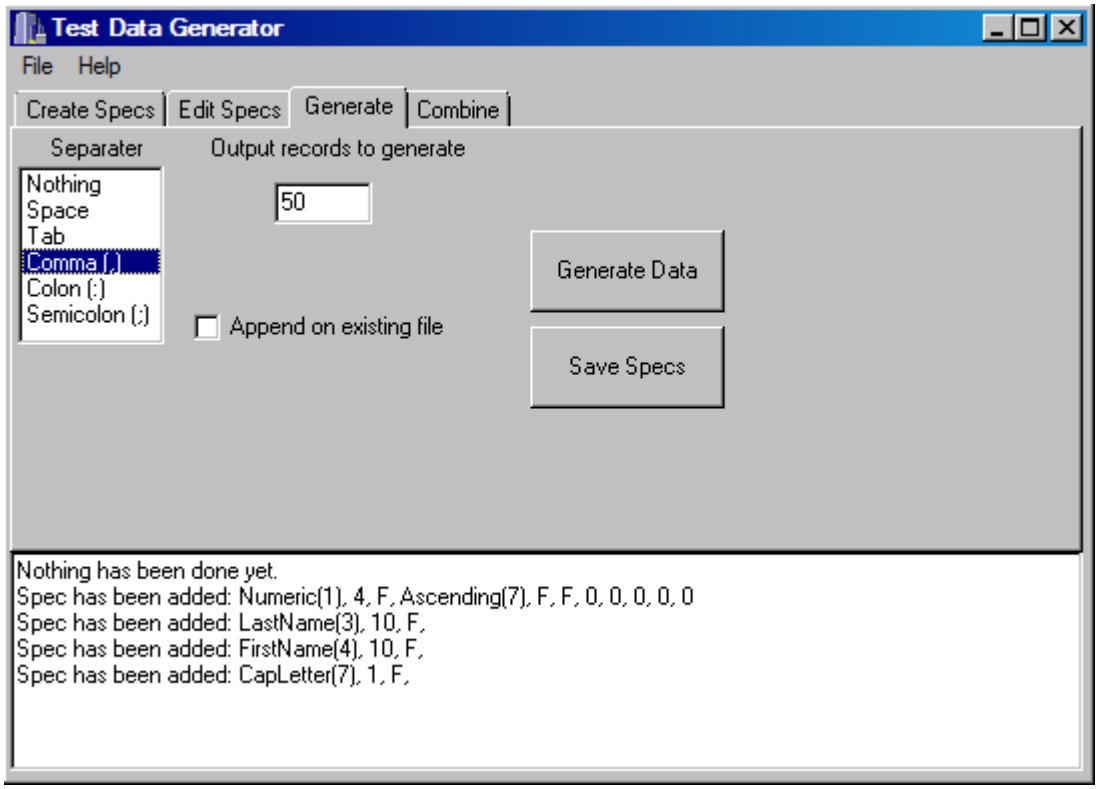

Figure 5 Generating the file

This screen gives another opportunity to save the specifications, but its main task to produce the proper output. It also allows the description of the delimiter fields. This could have been done with constant fields, but this is more convenient if the delimiter is always the same.

The final tab is optional and is one solution to the related fields issue. Figure 5 shows the options.

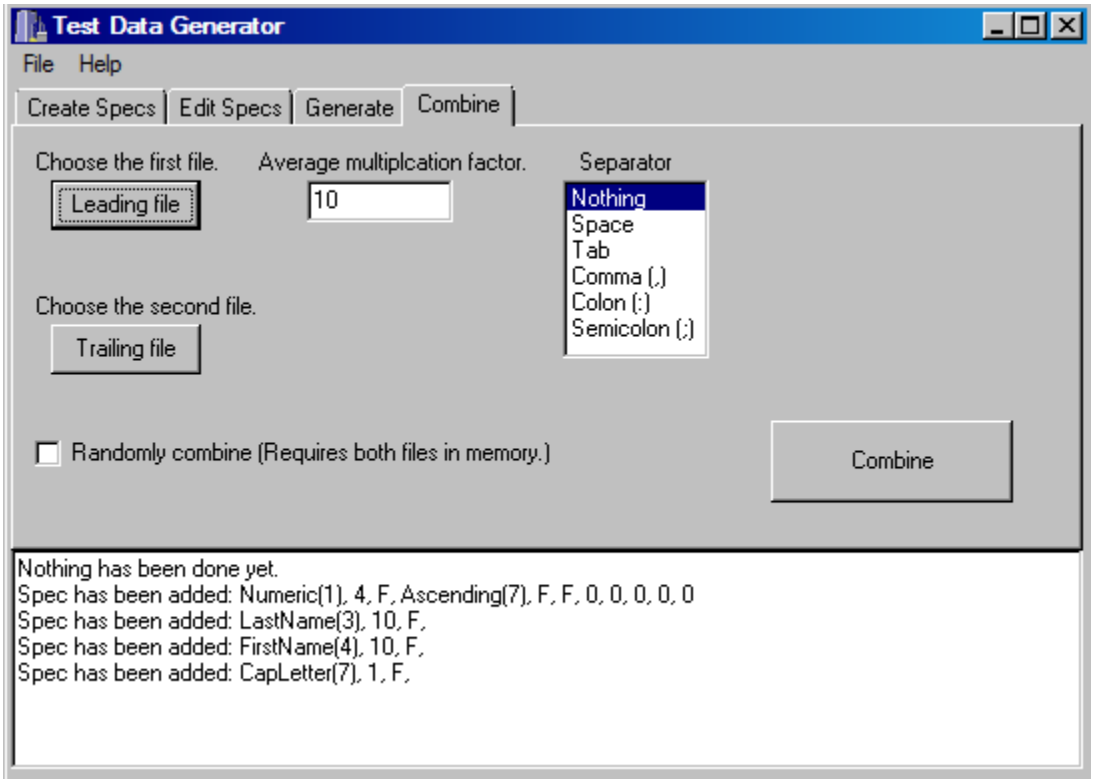

Figure 6 Combining files tab

Suppose that the employee records have been generated into a file and these represent a sales staff. The intent is to generate transaction data with this set of sales representatives and the products sold. In the simpler test data generators, either the connection between the employee and transaction would be lost or the sales representatives would never sell more than one item. The solution is to generate a limited number of records describing sales representatives, a second file representing products and a third that represents a count of the items sold. The three files are then combined in something akin to a limited Cartesian Product of relational databases. The number of records balloons but both the number of unique sales representatives and the number of products remains limited. A program that tallies these will get multiple hits for most of the employees and products.

Case-Studies

Two case studies will be considered. The first is the generation of a data for a data structures class. The second is for the use of a database class.

The topic of the data structures class was container classes suitable for quick retrieval. An in-class demonstration was to produced that used a binary tree. All the code was examined and posted on a web page. The program used a Graphical User Interface that allowed insertions, lookups, deletions and replacements. The data consisted of felons and had four pieces: a name, a zip code, a crime and whether the felon was currently

incarcerated. This data was generated from the web using the GenerateData.com[1] web site. (Screen shots are in Figures 1 and 2.)

It is a tedious task to enter enough data through the GUI to suitably test a program of this sort. Thus the program had an option for getting data from a file. The first character of each line was a single letter to indicate the function. An 'A' indicates an add. An 'L' allowed the lookup of a name, etc. About two thousand records of data were generated. There were further processed by prepending an 'A' to signify that this line was to be inserted. Subsets of this data were also generated for lookups and deletions. Errors had to be inserted manually as well, such as duplicate adds or lookups that failed. The manual processing to this was quite simple and the author is not aware of a data generator that could have done all of this without some intervention.

This demonstration and the corresponding code was then recycled. All the data and code was made available to the students. Their assignment was to remove the binary tree classes and replace them with a set of either BTree or Trie classes. They were to use the same main program, same data and the same method names in newly written classes.

The data for the database class was much more complicated. The data was presented as SQL inserts for a single table. The students were to use the SQL Select Into statements to create four separate tables. The first table described company sales representatives. The generation of this file was shown in Figures 3 – 5. The second table would describe products. The third table showed customer companies. The fourth table contained only a transaction ID and the number of items sold.

The first three of these were generated with [4] and then combined into a single file. At that time the transaction ID and count were appended onto this file. Approximately 17,000 records were generated. Only the transaction ID was unique in these records. There were about 50 to 150 unique items in the first three files.

References

[1] Black Sheep Web Software. GenerateData.com. http://www.generatedata.com/ Date accessed 17 March 2010.

[2] Databene. Databene Benerator. http://databene.org/databene-benerator Date accessed 16 March 2010.

[3] Grid-Tools. Grid-Tools Datamaker. http://www.grid-tools.com/datamaker.php Date accessed 17 March 2010.

[4] Hill, Curt. Curt's Program Store. http://community.vcsu.edu/facultypages/curt.hill/My_Webpage/download.htm Date accessed 19 March 2010.

[5] Redgate. SQL Data Generator http://www.red-gate.com/products/SQL_Data_Generator/index.htm Date accessed 17 March 2010.

[6] Turbo Computer Systems. TurboData. http://www.turbodata.ca/ Date accessed 17 March 2010.