# C. Getting Example Data Sets

## C.1. Star Data (HYG Database)

David Nash, amateur astronomer, has assembled a database of nearby stars that combines data from three sources:

- The Hipparcos satellite's massive survey of millions of stars
- The Yale Bright Star Catalog, containing data for about 10,000 stars
- The Gliese Catalog of Nearby Stars, containing about 4,000 stars.

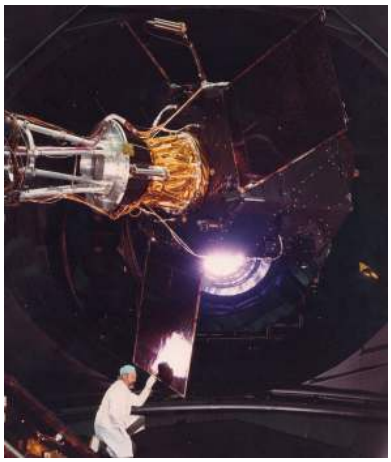

Figure C.1: The Hipparcos satellite before launch.
*Source: Wikimedia Commons*

Nash combined the nearby stars in these databases to form the HYG database (for "Hipparcos, Yale, Gliese").

For one of the exercises in Chapter 5 you'll need to download the HYG database and create a new, smaller, database from it. The resulting file will be called `stars.dat` and it's used in Exercise 31. Here's how to get the database and create `stars.dat` from it. The process will involve a couple of mysterious commands that I won't explain, but feel free to do some research on your own to find out what they do. The steps to create `stars.dat` are:

1. Fetch the HYG database. There are two tools that let you do this easily. Use whichever tool is installed on the computer you're using.

   The first tool is `wget`. The `wget` command lets you download files from a web site without needing to use a web browser. Here's how to use `wget` to download the HYG database;

   ```
   wget https://raw.github.com/astronexus/HYG-Database/master/hygdata_v3.csv
   ```

   If the computer you're using doesn't have `wget`, it probably has a similar tool named `curl`. Here's the `curl` command for downloading the database:

   ```
   curl -L -O https://raw.github.com/astronexus/HYG-Database/master/hygdata_v3.csv
   ```

2. Extract the part of the data that we'll be using in Exercise 31. Note that this is one big, long command without any line breaks. Every character in it is important, so type carefully. (If you can cut-and-paste the command, it's a good idea to do so.)

   ```
   cat hygdata_v3.csv | grep -v -E 'Sol|^id' | awk -F, '{print $18,$19,$20}' > stars.dat
   ```

   What does this command do? First, it uses the *grep* command to exclude two rows of data: a row of column headers, and the row for our Sun (which is included in the data just like other local stars). Second, it uses the *awk* command to select only three columns: just the columns that hold the $x$, $y$, and $z$ coordinates of each star.

That's it! You now have the `stars.dat` database, and you're ready for Exercise 31.

You might want to play around with other data in the HYG database. If so, you can find a description of the data it contains here:

https://github.com/astronexus/HYG-Database

## C.2. Normally-Distributed Data

Chapter 7 uses the file `energy.dat` for several exercises. This file contains simulated energy measurements from a scintillation counter. The energy values are "normally" distributed, meaning that when we make a histogram of the values it has the shape of a Normal distribution.

Figure C.2: Carl Friedrich Gauss, who studied the Normal distribution extensively.
*Source: Wikimedia Commons*

You can generate `energy.dat` by compiling Program C.1 and running it like this:

```
./mkenergy > energy.dat
```

Program C.1 uses a technique called the Box-Muller Transform to generate normally-distributed numbers. It defines a function named `normal` that takes two arguments (the mean of the distribution and its standard deviation) and returns a single pseudo-random number. (You'll understand how to create C functions after reading Chapter 9.) The program's `main` function just uses `normal` to generate 100,000 numbers. By changing the mean and standard deviation, you can change the distribution of the numbers. Try it, it's fun!

Program C.1: mkenergy.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
double normal(double mean, double sigma) {
  // Use Box-Mueller Tranform to generate
  // normally-distributed numbers.
  const double epsilon = 1e-9;
  const double two_pi = 2.0*M_PI;
  static double z0, z1;
  static int generate=1;
  static int initialized=0;
  double u1, u2;

  if (!initialized) {
    srand(time(NULL));
    initialized = 1;
  }

  if (!generate) {
    generate = 1;
    return z1 * sigma + mean;
  } else {
    do
      {
        u1 = rand() * (1.0 / RAND_MAX);
        u2 = rand() * (1.0 / RAND_MAX);
      }
    while ( u1 <= epsilon );

    z0 = sqrt(-2.0 * log(u1)) * cos(two_pi * u2);
    z1 = sqrt(-2.0 * log(u1)) * sin(two_pi * u2);
    generate = 0;
    return z0 * sigma + mean;
  }
}
int main () {
  int i;
  for ( i=0; i<100000; i++ ) {
    printf ("%lf\n",normal(35.0,2.5) );
  }
}
```

Figure C.3: Statistician George E.P. Box, one of the inventors of the Box-Muller transform.

*Source: Wikimedia Commons*

## C.3. Census Data (American Community Survey)

Figure C.4: A U.S. census worker transcribing data onto punched cards during the 1950s.
*Source: Wikimedia Commons*

In addition to the decennial census mandated by the U.S. constitution, the Census Bureau conducts many other surveys. One of these is the ongoing *American Community Survey* (ACS), which gathers data about how Americans live in their communities. Data from the ACS help local governments decide how to spend their money.

ACS data can be downloaded from the Census Bureau's web site. In order to protect the identities of the citizens who respond to the survey, only an anonymized sample of the data (called a "Public Use Microdata Sample" or "PUMS") is provided. These data sets are available here:

https://www.census.gov/programs-surveys/acs/data/pums.html

Exercise 40 on page 252 uses a data file named `census.dat` derived from the ACS data collected during the years 2011 through 2013. Here's how to make it:

1. First, as in Section C.1 above, you'll need to fetch some data from a web site. If your computer has the `wget` command, you can do it this way:

   ```
   wget http://www2.census.gov/acs2013_3yr/pums/csv_hus.zip
   ```

   otherwise, you can use the `curl` command like this:

   ```
   curl -L -O http://www2.census.gov/acs2013_3yr/pums/csv_hus.zip
   ```

2. The file you downloaded is named `csv_hus.zip`. This file has several data sets packed inside it, so the next step is to unpack them. You can do this by using the following command:

   ```
   unzip csv_hus.zip
   ```

   This will extract five files:

   ```
   ss13husa.csv
   ss13husb.csv
   ss13husc.csv
   ss13husd.csv
   ACS2011-2013_PUMS_README.pdf
   ```

   The last file contains documentation describing the data, and the other files contain the actual data, broken into four parts.

   If you looked inside one of the `csv` files you unpacked, you'd see that each line of the files was just a list of values separated by commas.

The "csv" in the file name stands for "comma-separated values". The data is organized in rows and columns. Each row represents a single household, and each column is a particular kind of data about that household (number of children or household income, for example). The top row is a comma-separated list of abbreviations telling us what each column represents.

3. To produce our census.dat file we're going to extract just a few of these columns. We could do this using awk and grep, as we did in Section C.1, but this is a book about C programming, so let's use a C program to do it this time.

   The program datafilter.cpp (Program C.2) contains a lot of stuff that you haven't seen before unless you've already finished reading this book. Much of it will become clear after you reach Chapter 8, and most of the rest after you read Chapter 9. The only parts that we won't cover in this book are the malloc and free functions. You'll have to learn about those in a different book, or do some research on your own.

   For now, just save this program as datafilter.cpp and compile it by typing "g++ -Wall -o datafilter datafilter.cpp".

4. The datafilter program needs a configuration file to tell it what to do. Using *nano*, create a file called census.conf containing the following lines:

```
,
-1
NRC
ACR
BDSP
FINCP
FULP
GASP
GRNTP
```

   Notice that the first line is just a comma on a line by itself. This tells datafilter that the columns in our data file will be separated by commas. The second line of the file tells datafilter that it should replace any missing data values with "-1". The rest of the lines are a list of columns that datafilter should select. These are the names that appear in the top row of each csv file.

Program C.2: datafilter.cpp

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
int main( int argc, char *argv[] )
{
  const int maxcolumns = 100;
  const int maxlength = 4096;
  char *output[maxcolumns];
  char *wanted[maxcolumns];
  int wantedfield[maxcolumns];
  int nwanted;
  char line[maxlength];
  char *position;
  char delimiter[maxlength];
  char blankvalue[maxlength];
  char *word;
  int i, field;
  FILE *input;
  FILE *setup;

  // Check syntax:
  if ( argc > 3 ) {
    fprintf ( stderr, "Syntax: %s datafile configfile\n", argv[0] );
    exit (1);
  }

  // Open data file:
  if ( !strcmp( argv[1], "-" ) ) {
    fprintf ( stderr, "Reading data from stdin.\n" );
    input = stdin;
  } else {
    if ( ( input = fopen ( argv[1], "r" ) ) ) {
      fprintf ( stderr, "Reading data from %s.\n", argv[1] );
    } else {
      fprintf ( stderr, "Error opening \"%s\": %s\n", argv[1], strerror(errno) );
      exit(1);
    }
  }

  // Open configuration file:
  if ( ( setup = fopen ( argv[2], "r" ) ) ) {
    fprintf ( stderr, "Reading setup from %s.\n", argv[2] );
  } else {
    fprintf ( stderr, "Error opening \"%s\": %s\n", argv[2], strerror(errno) );
    exit(1);
  }


  // Read delimiter:
  fgets( delimiter, 10, setup );
  delimiter[strcspn(delimiter, "\r\n")] = 0;

  // Read blank value:
  fgets( blankvalue, maxlength, setup );
  blankvalue[strcspn(blankvalue, "\r\n")] = 0;


  // Read fields:
  nwanted = 0;
```

```
  for ( i=0; i<maxcolumns; i++ ) {
    if ( !fgets ( line, maxlength, setup ) ) { // Break at EOF.
      break;
    }
    line[strcspn(line, "\r\n")] = 0;
    wanted[i] = (char *)malloc( strlen( line ) + 1 );
    snprintf( wanted[i], strlen( line ) + 1, "%s", line );
    nwanted++;
  }

  // Close configuration file:
  fclose ( setup );

  // Read header:
  fgets( line, maxlength, input);
  line[strcspn(line, "\r\n")] = 0;
  position = line;
  field = 0;
  while (position != NULL) {
    word = strsep(&position, delimiter);
    for ( i=0; i<nwanted; i++ ) {
      if ( !strcmp( word, wanted[i] ) ) {
        wantedfield[i] = field;
      }
    }
    field++;
  }

  // Read data:
  while ( fgets( line, maxlength, input) ) {
    line[strcspn(line, "\r\n")] = 0;
    position = line;
    field = 0;
    while (position != NULL) {
      word = strsep(&position, delimiter);
      for ( i=0; i<nwanted; i++ ) {
        if ( field == wantedfield[i] ) {
          output[i] = (char *)malloc( strlen(word) + 1 );
          if ( strlen(word) ) {
            snprintf( output[i], strlen(word) + 1, word );
          } else {
            snprintf( output[i], strlen(blankvalue) + 1, blankvalue );
          }
        }
      }
      field++;
    }
    for ( i=0; i<nwanted; i++ ) {
      printf ( "%s ", output[i] );
      free ( output[i] );
    }
    printf ("\n");
  }

  if ( input != stdin ) {
    fclose ( input );
  }
}
```

5. Now we're ready to create `census.dat`. Type the following commands to do it:

```
./datafilter ss13husa.csv census.conf > census.dat
./datafilter ss13husb.csv census.conf >> census.dat
./datafilter ss13husc.csv census.conf >> census.dat
./datafilter ss13husd.csv census.conf >> census.dat
```

Each of these lines processes one of the `csv` data files and appends the columns extracted from it onto the end of the file `census.dat`.

Seven columns are extracted from the original data. These columns are[1]:

| | | |
|---|---|---|
| 0 | NRC | Number of related children in household |
| 1 | ACR | Lot size, in acres |
| 2 | BDSP | Number of bedrooms |
| 3 | FINCP | Family income |
| 4 | FULP | Annual fuel cost |
| 5 | GASP | Monthly gas cost |
| 6 | GRNTP | Monthly rent |

[1] Notice that we've numbered them like the elements of a C array, starting with zero instead of one.

If you'd like to do further research with this data you can find a complete description of each of the columns in the `csv` files here:

http://www2.census.gov/programs-surveys/acs/tech_docs/pums/data_dict/PUMS_Data_Dictionary_2011-2013.txt