A Guide to Managing Large Financial Research Databases*

Terence Lim

First Draft: September 1997
This Draft: March 1999

Abstract

This report describes the implementation of a unix-based platform for managing large financial research databases in a university environment. It discusses several C-language programming enhancements made to improve access to databases such as CRSP, Compustat and I/B/E/S. A toolkit of supplemental subroutines that access the data via random access from individual or multiple databases simultaneously can be helpful for some empirical applications. New utility and sample programs illustrating the application of this toolkit are described. These can be combined with newly-available web-based tools to enable access to the databases through a web browser interface. A complementary SAS statistical and data management environment may be appropriate for other applications. Such enhancements can make these databases more accessible and usable by both casual users and sophisticated researchers. By jointly implementing a common set of enhancements, beginning with those described in this report, multiple institutions can pool their resources to support the installation, maintenance and continual improvement of a better system for managing large financial research databases. Further, such a standardized installation can facilitate the sharing and testing of libraries of program code for empirical research.

*Preliminary and Incomplete. Comments appreciated. All views are my own and do not reflect those of the Center for Research in Security Prices (CRSP), Standard and Poor's, I/B/E/S International, the New York Stock Exchange or the Free Software Foundation. I thank the research computing professionals at MIT Sloan School of Management (Ray Faith) and Tuck School at Dartmouth (Bob Burnham, Sarah Leonard and Peter Vishton), Boulat Minnigoulov and Kent Womack. Please address correspondence to Terence Lim, 100 Tuck Hall, Tuck School, Dartmouth College, Hanover NH 03755. Email: terence@alum.mit.edu. The latest version of this report is downloadable from the web at http://mba.tuck.dartmouth.edu/pages/faculty/terence.lim

Contents

List of Figures

1 Introduction

1.1 Motivation

The contents of this report can be used at three levels. At the first level, the report serves as a primer on financial research databases and tools for managing these data. This could be helpful for new users who are just beginning to learn about financial research databases such as CRSP, Compustat, and I/B/E/S; and about general tools for data management and access, ranging from traditional programming and statistical environments like C and SAS, to Web-based tools like HTML, CGI, Perl, JavaScript and Java. At a second level, this report illustrates and explains how to apply these tools in an academic research computing context. It tours through the steps of providing a usable data access environment, including the initial installation of the data files; enhancing access through implementing random and simultaneous access to multiple databases; adding and managing new utility subroutines; and the development of graphical interfaces utilizing the Web. Finally, this report can be used as a reference manual for installing an actual unix- and primarily C language-based database management platform (the "Platform for Empirical Research and Computing" or PERC Toolkit), incorporating all these components to help address a range of research computing needs.

At this point a warning to readers is in order. This report describes many of the algorithms used in somewhat gory detail, with the intention to elucidate. However, casual users who simply want to install the PERC programs and toolkit may wish to skip to the end of the report for installation instructions.

It is hoped that this report can provide some guidance for setting up financial research databases and useful data access tools, if not form the basis of a standard implementation to support research and coursework. Because these databases are typically provided to university subscribers "raw" with few if any interface or installation programs, many sites find that they are often "reinventing the wheel", or are not providing sufficient tools to make the databases more accessible and usable. If a common, efficient and sufficiently sophisticated standard could be adopted, beginning possibly with the model described herein, multiple sites could share their resources in implementing, supporting and enhancing such a system.

The tools and procedures described here were developed under the same constraints and conditions that many university sites face. Many problems would probably go away if we had available more hardware resources, commercial database management or data warehousing software, and abundant long-term research computing help. On the demand side, the data needs and sophistication level of users range widely, from casual users who just want some specific data quickly, to "power" users with highly demanding data and programming applications. While the toolkit cannot be guaranteed to be free of errors, I have tested and used it to support research in stock price momentum and analysts' earnings expectations and for teaching in the Tuck MBA program.

The technologies used here are standard and widely-available. I use standard unix utilities and scripts for automating installation of the databases, creating sorted index files to augment the original flat data files, performing searches and lookups, managing subroutine libraries and merging data files. The implementation required a small number of auxilliary data files (primarily to index the data by security identifiers), but this is done as sparingly as possible to avoid needlessly complicating the installation process or using up storage space. The original interface routines (if any) provided by vendors were enhanced with a common set of C-language subroutines that allow random access and simultaneous access to the databases. I illustrate how to incorporate other program libraries, such as the popular Numerical Recipes in C routines. The implementation of a web server and web browser-based interface used "free" software and standard web programming

tools only. A SAS environment has also become typical at universities.

Because of the varied characteristics of users, a support strategy is adoped of providing different methods of access, in increasing sophistication. "Casual" users, who just need quick access to certain selected time series (e.g. "I just want to download some stocks returns and interest rates data into my excel spreadsheet"), would prefer to use the web interface to specify their selections and have web server programs automatically and instantaneously extract and return their data back to their web browser (which can be saved to disk and loaded into a spreadsheet or any other program) – no programming required. Such a web interface can also help new users begin learning about the conventions and characteristics of the underlying databases. However, for "intermediate" users, web-based forms may not be flexible enough (e.g. "I want financial data on all companies that meet certain specified qualitative and financial criteria"); access via a SAS environment would be preferred. After initially installing the databases as permanent, indexed data sets, the menu-driven SAS/ASSIST graphical interface can be a powerful tool for browsing, subsetting data, and running SAS procedures, even for new SAS users. SAS programs are particularly easy and suitable for tasks such as cross-sectional data analysis. However SAS uses memory greedily, often consuming large amounts of temporary disk space, and the SAS macro programming environment can be inadequate. There remain certain applications for which SAS is not appropriate and "low-level" programming access is needed. To help "power" users, an enhanced library of common subroutines in C is provided to automate many functions, read the databases via random access, access multiples databases simultaneously, and perform analytical tasks.

The financial research databases covered in this report are described in section 2. Section 3 provides details on how to install these databases as character, binary and/or SAS data files. Section 4 describes basic access to the databases using sequential C language programs or SAS, and how to develop and manage a library of supplemental subroutines. A set of subroutines for generating weekly CRSP Stocks data is presented as an example. Section 5 discusses how to augment the original flat data files with index files so as to support random access programs. These can be used to access multiple databases simultaneously, so that all available financial, capital markets and expectational data for each company can be read and analyzed at the same time; as explained in section 6. Section 7 presents additional examples of empirical research applications. Section 8 illustrates how to incorporate other numerical or statistical program libraries for directly analyzing the data. Section 9 demonstrates how to implement a web-based interface, using basic web query forms, as well as more advanced tools such as Java applets and Microsoft Excel web queries. Section 10 concludes. The Appendix describes all the utility programs, C-language routines, web interface scripts and query forms, and installation steps and scripts that make up the PERC Toolkit.

1.2 License

The programs described in this report can be most useful to the research community if

- the source code can be freely shared, improved and re-used;
- the programs and source code can be made widely available to any set of interested users;
- the source code and any new programs that incorporate them are free from software patents that make them proprietary;
- recipients of the programs or modifications of the programs understand that there is no warranty for the software.

Most of the programs described in this report are made available under the terms of the GNU General Public License. A large suite of successful software has previously been developed and made available to a wide range of users under this type of *open source*[1] license, most notably Linux and Emacs.

The following preamble from the Free Software Foundation interprets the terms of the License. The precise terms and conditions for copying, distribution and modification can be obtained by writing to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA; or from their web site `http://www.gnu.org`. A copy is reproduced in the appendix.

> The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.
>
> When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
>
> To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.
>
> For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.
>
> We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.
>
> Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.
>
> Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

You may copy, modify and distribute the programs and source code, as long as these are accompanied by the following notice.

> PERC Toolkit for managing large financial research databases, developed by Terence Lim (terence@alum.mit.edu) and the Tuck School. Copyright (C) 1999 Trustees of Dartmouth College.
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA; or browse to http://www.gnu.org.

[1] Browse to `http://www.opensource.org` to learn more about the Open Source Initiative.

2 Overview of Research Databases

This section reviews commonly-used financial research databases, which are the subject of this report.

2.1 Center for Research in Security Prices (CRSP)

These databases were developed by Professor Lawrence Fisher of the University of Chicago in the early 1960's, in collaboration with Professor James Lorie. Their empirical work on historical stock returns was among the first to provide statistical calculations on large accumulations of financial data. CRSP (pronounced "crisp") data is now a staple of academic research.

University subscriptions are updated annually. Data up to the previous year-end become available to subscribers at the beginning of summer. In 1997, CRSP made available a new format version, unfortunately named CRSP Access97 (though it has nothing to do with Microsoft's database product), for which improved access routines (allowing direct or random access) in both C and Fortran can be used. This subsection applies to this new format; the old format will be phased out by CRSP.

Beginning in September 1997, CRSP has made available the manuals for all their data products, in Microsoft Word and Adobe Acrobat formats, at their web site `http://www.crsp.com`.

CRSP Stocks Files

CRSP provides comprehensive security price data for the New York Stock Exchange (NYSE), American Stock Exchange (AMEX) and Nasdaq Stock Market, including both the Nasdaq National Market (comprising the 2500 securities that must meet stricter operating criteria than the other Nasdaq stocks), as well as the Nasdaq Small-Cap Market. In 1997, data on over twenty thousand securities, including those that are no longer listed, were provided. A user can track a security through its entire trading history through its permanent unique PERMNO (CRSP's propietary Permanent Number), regardless of name, cusip or ticker symbol changes or capital structure changes.

- **CRSP Stocks Monthly file**

  The bulk of this data file comprises monthly price, trading volume and holding period returns (incorporating dividends and stock splits) histories for the period that a security is traded on the NYSE, AMEX and Nasdaq. In addition, names histories are also included, containing historical identifiers (such CUSIPs and ticker symbols), exchange listing, share classes, Standard Industrial Classification (SIC) codes and Nasdaq traits. Additional information include distribution histories (such as dividends, stock splits and special distributions), delisting information and shares outstanding values.

  Decile rankings are available based on market capitalization and exchange, using market capitalization statistics from the end previous calendar year (for CRSP Decile Portfolios) or prior quarter (for CRSP Cap-based Portfolios). These rankings, when combined with portfolio returns from CRSP Indices, can be used to generate monthly excess stock returns.

  Data for NYSE securities date from December 1925, while AMEX securities begin in July 1962. The Nasdaq data became available in December 1972.

  When programming against the CRSP Stocks files, special care must be taken to account for missing data codes. For example, when a security did not trade on a particular date, the price

reported by CRSP may be either the negative of the closing bid-ask quote (in which case the price variable has a *negative* value), or zero (denoting a missing price). Missing returns and trading volume are represented by large negative values (from -66.0 through -99.0, the value of which further denotes the reason for being missing).

- **CRSP Stocks Daily file**

  The CRSP Stocks Daily file provides essentially the same information as the CRSP Stocks Monthly File, but at a daily frequency beginning in July 1962[2].

  This file now incorporates data items previously distributed in a supplemental Nasdaq file. These items include Nasdaq daily closing bid, closing ask, and number of trades. These data items have been reported for issues listed on The Nasdaq National Market since November 1, 1982. Issues listed on the Nasdaq SmallCap Market have had these data reported since June 15, 1992 (this segment of the Nasdaq Stock Market became subject to real-time price and volume reporting in June 1992).

  Decile rankings are available based on risk (beta and standard deviation), market capitalization and exchange, using market capitalization or risk statistics from the previous calendar year. These rankings, when combined with portfolio returns from CRSP Indices, can be used to generate daily excess stock returns.

  The old CRSP Stocks Excess Returns file is not a part of CRSP Access97. This file used to provide excess returns data, defined as the stock's return minus the return on a portfolio of stocks with similar risk (i.e. in the same market decile when ranked either by beta or standard deviation). Instead, the new CRSP format allows CRSP Indices and Stocks data to be simultaneously accessed, and provides subroutines to generate excess returns from the underlying stock's returns and decile ranking, and the corresponding decile index returns. Furthermore, beta and standard deviation rankings and portfolio returns are now also available for Nasdaq stocks (previously these were only available for NYSE and AMEX stocks).

CRSP Indices Files

The CRSP Indices files provide summary information about the market. There are data series representing rates of return and associated index levels for various segments of the US equity market (derived from the CRSP Stocks files), constructed based on exchange, market capitalization, and risk. It also includes information on common indices, such as the Standard and Poors 500 Composite Index, Nasdaq Stock Market Composite, and by asset class.

CRSP now provides subroutines to simultaneously access Indices data along with stocks data. The indices can be identified using unique identifiers similar to the permanent numbers for stocks. Character versions of the files are also available, which can be easily imported into spreadsheets or other programs.

- **CRSP Stock File Indices**

  These series cover returns on individual and combined exchanges (NYSE, AMEX, Nasdaq) at daily up to annual frequencies. Returns series are also provided on market segment deciles constructed from stocks ranked by market capitalization, beta and standard deviation (note

[2]Researchers often wish to transform daily data to a weekly frequency for certain empirical applications. This report describes new subroutines that generate weekly calendar series, to enable weekly data such as stock returns and prices to be generated "on-the-fly" while reading the CRSP Stocks Daily file

that the latter two risk-based segments are provided only for daily data on NYSE/AMEX stocks). The monthly NYSE series begin in December 1925, while the daily NYSE series begin in July of 1962. All AMEX series begin in July 1962 and all Nasdaq series begin in December 1972.

- **CRSP Risk-Based Portfolios**

  This file provides portfolio decile returns based on two types of excess returns: beta excess returns and standard deviation excess returns, for NYSE/AMEX (or Nasdaq) stocks only. For each stock in the CRSP NYSE/AMEX (or Nasdaq) universe, its beta is computed each year the stock traded at least half of the trading days, and its standard deviation is computed if there are returns for at least 80 percent of the trading days in that year. The market is then divided into ten portfolios each year by ranking on beta and on standard deviation. The beta and standard deviation excess return for a stock is the difference between its daily return and the daily return of the decile portfolio to which it has been assigned.

  Betas are computed using methods developed by Scholes and Williams (1977) to account for the effects of non-trading days. The ranking of stocks and formation of portfolios are based on the beta and standard deviation statistic for the stock in the previous year. If that value is not available, the current year's statistic is used.

- **CRSP Cap-Based Portfolios**

  These provide monthly returns series on market-capitalization based market segments (deciles). The series were constructed in a somewhat elaborate way. Only common stocks were included in the universe; excluded were Unit Investment Trusts, Closed-End funds, Real Estate Investment Trusts, Americus Trusts, Foreign stocks and American Depository Receipts. All eligible NYSE companies were ranked by market capitalization at the end of each quarter, and ten equally-populated portfolios were formed. Using the market capitalization breakpoints of these deciles, based only on NYSE companies, all eligible stocks including AMEX and Nasdaq were placed in deciles according to their respective market capitalizations. The data series begin in December 1925.

- **CRSP Indices on the S&P500 Universe**

  These indices are similar to the standard CRSP indices but were constructed only from stocks in the S&P 500 index (or S&P 90 prior to March 1957). These indices do not exactly mirror the commonly quoted S&P 500 index. Mismatches could occur because of historical differences in handling mergers, reorganizations and other major corporate actions. Equal and value-weighted returns with and without dividends are provided. The published S&P 500 index and returns (which are value-weighted but exclude dividends) are also included for comparison. Daily data beginning July 1962 and monthly data beginning December 1925 are available.

- **CRSP CTI Indices**

  These contain historical performance information on a number of different asset classes such as stocks, small stocks, treasury securities of various maturities, and consumer prices. Most of the series, available at monthly, quarterly and annual frequencies, begin in December 1925.

  This data is designed to replace the Ibbotson and Associates Stocks, Bonds, Bills and Inflation (SBBI) series that are no longer distributed by CRSP.

CRSP US Government Bonds Monthly Files

The bulk of this dataset comprise monthly price information on virtually all negotiable direct obligations of the United States Treasury since December 1925. Price data, derived quantities such as yield and duration, as well contractual characteristics of each bond are available sorted either by issue or by quote date. In addition, a number of smaller supplemental files (the "Fama" and Fixed Term Indices Files), derive and provide historical term structure and interest rate information. The data are distributed as character files. The smaller data files are also available as excel spreadsheets.

- **CRSP Bond Master File**

  Sorted by issue, this file tracks some 4,960 US Government securities since December 1925. It provides monthly data on bid and ask prices, holding period returns, yields, duration, maturity, taxability, coupon, callability and other contractual characteristics.

- **CRSP Bond Cross-section file**

  This file provides almost identical monthly data on Treasury issues, but is sorted by quote date.

- **Teasury Bill Term Structure Files**

  This is a series of data files that provide historical monthy term structure information on selected Treasury Bills at 6-month and 12-month target maturities. Data on yields, forward rates and holding period returns are constructed from the bid, ask as well as bad/ask average prices.

- **Maturity Portfolios Returns File**

  These files contain monthly holding returns of portfolios constructed from six-month and one-year maturity bonds.

- **Fama-Bliss Discount Bonds File**

  Artifical discount bonds with one to five years to maturity, using a filtered subset of fully taxable, non-callable, and non-flower bonds. Month-by-month forward rates, discount prices and yields are reported, beginning in 1952.

- **Risk-Free Rates Files**

  These files contain one and three month risk-free rates derived from bid, ask and average quotes, beginning in 1925.

- **Fixed Term Indices Files**

  These derived bond files offer seven indices derived from bonds at 30, 20, 10, 7, 5, 2, and 1 year target maturities, providing historical bond yield curve information. Start dates vary depending on the maturity.

2.2 Standard and Poors Compustat

The full University subscription buys fundamental financial data on more than 19,000 North American companies. This includes approximately 10,000 active companies with up to 20 years and 48 quarters of history; 8,400 inactive (research) companies with up to 20 years of history; 770 Canadian companies with up to 20 years of history and quarterly history from 1991, as well as over

300 Industry Aggregates annual and quarterly data. 340 annual and 145 quarterly financial data items, industry segment (business segment line of business) information and geographic segment information are provided.

However, the standard subscription does not provide quarterly financial data, industry segment information nor geographic segment information for inactive (research) companies. This data can be purchased separately, and should be so as to mitigate the serious problem of survivorship bias. Also, back data (the start dates of which depend on the data set) can also be purchased to provide longer histories. Note that when Compustat begins coverage of a company, it may backfill several years of financial data[3].

The University subscription is updated annually in the summer, with a cut-off date of around June. Any quarterly or annual financial reports that have been released by the companies and coded by Compustat into its database as of the cut-off date would become available. No sample programs or access routines are provided. Some third parties, most notably SAS, have developed their own routines to read the data into other formats (e.g. SAS data sets). Fortunately, the data is structured in a relatively simple linear format.

When programming against the Compustat files, special care must be taken to account for missing data codes. In the binary data files, these codes can take on values of 0.0001, 0.0002, 0.0004 and 0.0008 (for subscribers to ascii character data files, these codes are respectively -0.001, -0.002, -0.004, -0.008). Data availability for a particular year can be checked from the UPDATE CODE array variable: a value of 2 or 3 in a year means that data has been updated for that year from a preliminary or final source respectively, whereas a 0 or 1 means that data is not available. Also, Compustat defines its FISCAL YEAR variable in a rather annoying manner. If the month of fiscal year-end is between January and May, its fiscal year is defined to be the previous year; for example, a year-end of March 97 is reported by Compustat to be FISCAL YEAR 96.

The data files are sorted by Industry Classification Code and CUSIP. Companies can be uniquely identified by its 9-character CUSIP, which comprises a 6-character CUSIP issuer code, followed by a three-character Issue Number and Check Digit (the check digit in the third character is actually redundant). However, only the latest CUSIP is available; if a company's CUSIP changed, due perhaps to a name change or capital structure change, the old CUSIP is discarded and the company, including all its historical data, is identified by its new CUSIP. Compustat may also assign its own CUSIP code to historical records when they are moved to the research files due to major accounting or capital structure changes: Compustat-assigned CUSIPs have a 99 in the fourth and fifth characters.

Industrial Annual

Industrial Annual data is provided in three distinct files. The Primary/Supplemental/Tertiary file contains all active companies listed on the major exchanges, including all companies comprising the S&P500 Industrial Index. The Full Coverage file contains active Nasdaq companies, companies listed on regional exchanges, publicly held companies trading common stock and wholly-owned subsidiaries trading preferred stock or debt. The Merged Industrial Research File contains companies that have been deleted due to bankruptcy, acquisition or merger, leveraged buyout, or because they became private companies.

More than 300 Income Statement, Balance Sheet, Statement of Cash Flows and supplemental data items and footnotes are available, for up to twenty years. Back data files, purchased separately,

[3]On how this may affect empirical studies, see, for example, Chan, Jegadeesh and Lakonishok (1995).

provide another twenty years of historical data, where available, for all companies in the three files. Additional "Way-back" data files provides coverage back to 1950.

The Canadian File contains major Canadian industrial companies that report in Canadian currency.

The Aggregate File contains data summed by item for S&P industry groupings.

Industrial Quarterly

The quarterly industrial data file offers restated data *as reported by the company.* A maximum of 48 quarters of data with more than 100 quarterly Income Statement, Balance Sheet, Statement of Cash Flows and supplemental data items and footnotes are available. A useful variable in the quarterly files is the report date of quarterly earnings[4], which represents the date in which quarterly earnings are first publicly reported in the various news media (such as Wall Street Journal or the Dow Jones News Service).

The University subscription comprises the Primary/Supplemental/Tertiary and Full Coverage only. See the previous section on Industrial Annual files for a description of the sets of companies these cover. Data on inactive (research) companies, are not included; they have to be purchased separately. Back and way-back data files are also separately available.

The Canadian File covers major Canadian industrial companies that report in Canadian currency.

The Aggregate File contains quarterly data summed by item for S&P industry groupings.

Industry Segments

The Industry Segment file is a companion to the industrial annual files, containing data on principle customers and products for up to ten industry segments per year for each company. Up to seven years of annual data are available.

Although a university subscription is only eligible for industry segments data for active companies, additional custom industry segments data for inactive, research companies, and back data, going back to 1982, for both active and inactive companies and respectively) can be separately purchased.

Custom and PC-based products

In addition to the full University tapes subscription, Compustat also sells PC-based products and specialised custom databases. A limited University subscription is also available which only covers a smaller subset of companies.

- **PC-Plus**

  This is a windows-based interface to the compustat database. It has the capability to generate pre-defined reports, perform screens, compute ratios, import private databases and output graphs. It also provides some additional data such as business descriptions, company officers and macroeconomic data (primarily interest rates). If the the university has installed the network version, students and faculty can also access the data (for an additional fee) from their own client PC's over the local area network.

[4]The date is represented in a Julian YYDDD format, where DDD is the day number in the year YY

- **Bank COMPUSTAT**

  This database provides financial information coverage on the largest U.S. banks.

- **Telecommunications Compustat**

  This database contains financial on all telephone companies currently filing annual reports on Form 10-K with the Securities and Exchange Commission.

- **Utility COMPUSTAT**

  This database contains financial information on the largest important utilities and utility subsidiaries.

- **ACE Analysts' Consensus Estimates**

  The ACE database contains consensus earnings estimates and buy/hold/sell recommendations from more than 200 contributing firms. Estimates are provided for annual and quarterly primary earnings per share and long term earnings growth. This data has been provided for free with both the full and limited University subscription, including monthly updates. However, data is only available for a short history, back to 1994.

- **Back Data and Inactive Companies Data**

  The University subscription does not include data for inactive (research companies) in the quarterly, business segments and geographic segments. This data can be purchased separately from Compustat. Also, additional back data providing longer histories (start dates depend on the database requested) can also be purchased for most standard and custom databases.

2.3 International Brokers Estimates System (I/B/E/S)

I/B/E/S maintains a large historical database of consensus and individual analyst earnings estimates data. For an annual license fee, an academic institution receives a set of I/B/E/S data tapes that can be installed on a network server. Any researcher within the institution would be eligible to use I/B/E/S data subject to certain conditions. Each researcher must submit a research proposal and a vita of the academic background to I/B/E/S. The user must also sign and return a condition statement.

The data are provided in numerous character files, but without any sample programs or access routines. Fortunately, the data is structured in a fairly simple linear format, although different data items (e.g. estimates, prices, actual earnings and identifier information) must be cross-reference across different files.

Each company in the database has a unique identifer, the I/B/E/S Ticker, through which a company's history can be traced regardless of changes in other identifiers such as name or cusip. The databases contain historical data on any company even if it is not longer currently viable. I/B/E/S provides historical cusips and ticker symbols which can be used to help merge with other databases. Also, all historical estimates are fully adjusted for splits and other capitalization changes; by using historical adjustment factors that are provided, it is possible to recalculate the data to a pre-adjusted basis. Analysts' estimates are of a company's operating earnings, whereas a company may report accounting earnings that include unusual, special or extraordinary items. Although I/B/E/S reports actual earnings restated on the same basis as analysts' forecasts, care must be taken particularly when analysing companies that take unusual accounting charges[5].

[5] for example, see Keane and Runkle (1997) and Philbrick and Ricks (1991).

Two versions of data are available for each of the US and International universes: summary concensus estimates and individual analyst-by-analyst estimates. Monthly summary consensus estimates of annual earnings, quarterly earnings and long-term growth, are available for US companies since 1976 and international companies since 1987. Individual analyst-by-analyst estimates (from which the original consensus estimates were built) are available for US companies from 1983 and for international companies since 1987. I/B/E/S also provides monthly stock prices and shares oustanding (in the summary version) and actual earnings reported for companies.

The detail data contains identity codes for brokers and analysts. However, the academic edition of I/B/E/S does not contain translation tables, so we do not know the actual names of the brokers or analysts. I/B/E/S attempts to follow analysts that change brokerage employers by maintaining the same analyst code. Note that an "analyst" may actually be a team of analysts: these are generally identified by a "/" or "and" in the analyst's name in the translation table. Broker and analyst translation tables can sometimes be obtained by special permission from I/B/E/S.

2.4 Other databases

Trades and Quotes (TAQ)

This database is a collection of all trades and quotes in NYSE, AMEX and Nasdaq securities, since January 1993. The TAQ database is published monthly on CD-ROM by the New York Stock exchange. Presently, one month's of data fills 3 CD-ROMs, which are distributed approximately one month after the last trading day of the month. The data are in character and binary integer values format. PC-DOS executable programs and fortran source code are provided for retrieving trades and quotes. Because the CD-ROMs are in standard ISO-9660 format, TAQ data can potentially be read on any computing platform, not just PC's TAQ documentation can be browsed at NYSE's web site `http://www.nyse.com`. Lim (1999b) describes a programming solution that attempts to tame the voluminous data spanning multiple years of stock transaction prices.

Trade observations are time-stamped and include the price, size, sale condition, correction indicator and exchange it occurred on. Neither the identity of the traders, nor their origination (e.g. SuperDot or exchange floor) are reported. Quotes are time-stamped and include the ask and bid price and size (depth), quote condition, exchange, and Nasdaq market maker (for NASD quotes). Trades and Quotes data are provided in separate files sorted by date and ticker symbol.

TAQ data are created from the exchanges' Consolidated Trades and Consolidated Quotes reporting systems, with little error checking or filtering. Care must be taken by the user to handle unusual observations, trades that were corrected or cancelled, and quotes that are not Best Bid or Offer (BBO) eligible. Additional identifying information on securities, such as CUSIP, primary exchange and industry group are also provided in a Master Table File. However, according to the NYSE's TAQ documentation, the shares outstanding and distributions (dividends and stock splits) information are unreliable.

Global Vantage

The GLOBAL Vantage database, available from Standard and Poors', provides international fundamental financial and market information. The database contains information, back to 1983 if available, for more than 11,000 companies from 70 countries representing Europe, Asia, the Pacific Rim, the Americas, Africa, and the Middle East. Additionally, a currency file provides historical cross-translation tables on more than 100 currencies. It also has pricing data on over 90 local market indexes.

The tapes subscription does not include any sample programs or access routines. When programming against the character data files, it is important to note that data units are dependent on a scaling factor for a particular period in which the data was collected. Missing data are coded with values of 0.0001, 0.0004 and 0.0008 in the binary data files (-0.001, -0.004, -0.008 in the character data files). It was also unfortunate that in the currency files, there occurred several cross-rates that had values equal to the missing data codes; it is hoped that Global Vantage will provide a solution to this problem in the future.

Companies can be uniquely identified by a six-character Global Vantage Company Key, while their associated security issues can be identified by an eight-character Global Vantage Issue Key (the last two characters identify the issue, usually beginning with "01"). These keys are also the sort order for each of the data files.

3 Loading the Data

This section describes how to load the raw data files onto disk. Vendors can provide their data in a variety of tape media and recently, more conveniently, on CD-ROMs. The data may be also available in binary machine format, if you use one of the supported hardware and operating system platforms. An advantage of obtaining the data in character format is that this provides more flexibility in installation and use[6]. Keeping files in character format on disk potentially allows almost any software package (such as SAS or SPlus) or programming language (such as Fortran or C) to directly access the data. Alternatively, disk space can be conserved by converting the data to binary format; however, the data can then only be accessed with the programming language originally used to convert the data[7].

Storing the data in a relational database system allows the user to perform sophisticated queries or generate reports with minimal programming. This sort of installation, however, usually requires a large amount of resources, often requiring more disk space than the original character files. SAS, a popular statistical package that also provides basic data management facilities like indexing and merging datasets by a common key, has gained popularity as a platform for managing many of the data files discussed in this report.

The examples used below were run on a Unix operating system[8], although the general methodology could be applied to other operating systems. Unix is a successful, popular operating system that runs on a range of computers, including microcomputers with "Intel Inside", workstations, and mainframes. It features multitasking, scripting and a large arsenal of utility programs that together provide a very powerful programming environment. Some shortcomings are that several variants of Unix exist (although end-users should see only small differences from one unix platform to another) and *administrating* unix systems, because of its flexible and customizable nature, can be very challenging. Nevertheless, once your Unix system is working, it will "forever"[9].

In the installation procedures described below,

1. CRSP Stocks and Indices data are installed in binary: this is the only option available for the new CRSP Access97 format.

2. Compustat Industrial data are stored as character files.

3. I/B/E/S data items are distributed in numerous separate character files; this report will describe how to convert and merge I/B/E/S data from their original character files into a single large binary file to reduce access time and storage space.

4. CRSP Stocks and Compustat Industrial data files can also be optionally installed as permanent, indexed SAS datasets.

[6]However, CRSP's new CRSP Access97 is only available in binary format, and only for a selective set of popular platforms.

[7]Many systems allow code in different programming languages to be linked into one executable program, thus potentially the code used to convert and access the data files can be in one language, while the rest of the code that processes or analyzes the data can be in any other programming language. However, this usually requires the programmer to code very carefully and adhere to some very strict conventions. Consult the documentation for your system's compiler

[8]Solaris 2.6 running on a Sun Ultra Enterprise 450, to be exact

[9]To quote Phillip Greenspun: Unix – you think it won't work, but if you find the right guru, you can make it work; Macintosh – you think it will work, but it won't; PC/Windows: you think it won't work, and it won't

The table below compares the approximate file sizes of character data files, binary data files and SAS data sets. SAS data sets are approximately as large as the original character files. Binary format can sometimes reduce file sizes substantially.

Figure 1: Size of data files

CRSP			
File	Binary Size	SAS Size	SAS Index File
Monthly Stocks and Indices	207MB	220MB	80MB
Monthly Events (SAS)		1200MB	
Stocks daily	1400MB	3400MB	1400MB
Daily Events (SAS)		960MB	

Compustat Industrial			
File	Character Size	SAS Size	SAS Index File
Annual (PST, Full, Merged Research)	1400MB	740MB	15MB
Annual Back Data (PST, Full, Merged Research)	512MB	288MB	6MB
Annual Way Back (PST, Full, Merged Research)	250MB	141MB	3MB
Quarterly (PST, Full)	1125 MB	400MB	18MB
Segment (PST/Full)	53MB	41MB	1MB

I/B/E/S				
File	Character Size	Binary Size	SAS Size	SAS Index File
Domestic (Summary and Detail)	690MB	600MB		
International (Summary and Detail)	600MB	500MB		

3.1 Directory structure

Our first step before beginning the installation process is to create a directory structure to keep our data files. A uniformly named and "flat" directory hierachy is suggested which users can easily remember. Although it is possible to reconfigure (by modifying the `db_local.h` file), it is strongly recommended that you adhere to PERC's default locations. You could use symbolic links (the `ln -s` command) to link your physical disk volumes to the structure described below. The file sizes listed in the prior table provide an indication of how much disk space should be made available in each volume; add about 25 percent more space that may be used during installation.

Figure 2: A site map (directory structure)

```
Directory                   Contents
---------                   --------
/db                         base installation directory
/db/crsp                    CRSP data root directory
/db/crsp/monthly            CRSP Monthly data
/db/crsp/daily              CRSP daily data
/db/compustat/data          Compustat data files
/db/ibes/data               I/B/E/S data files

/db/perc                    base directory for PERC programs
/db/perc/src/lib            Library of C subroutines and source code
```

```
/db/perc/src/include            Header files for C routines
/db/perc/src                    utility programs and source code
/db/perc/src/c                  sample programs and source code
/db/perc/cgi-src                (copy of) CGI Perl scripts for web interface

/db/perc/crsp                   CRSP Stocks web forms
/db/perc/indices                CRSP Indices web forms
/db/perc/bonds                  CRSP Bonds web forms
/db/perc/compustat              Compustat web forms
/db/perc/ibes                   I/B/E/S web forms
```

3.2 Loading Compustat character files from tape

Depending on the original data media, character data files can be copied onto hard disk using the unix `cp` or `dd` commands. The former command can be used, for example, if the data is on CD-ROM which can be mounted on the machine as a regular filesystem. In this case the files on CD-ROM can be accessed like normal disk files using standard commands like `cp` and `ls`. For tape media, however, you need to use the `dd` command to specify the characteristics of the file, such as its blocksize, when converting and copying the file from tape to disk.

The following command, for example, copies a Compustat Industrial Annual character file from tape (assuming its device name is `/dev/nsrt0`) to disk. The input blocksize is 8332 bytes (from the *Compustat Technical Guide 1998*. Similar commands for Compustat Quarterly (blocksize 27552) and Segment (blocksize 7740) files are listed in the appendix, and collected in a single script named `load.cst`.

Figure 3: Copying Compustat Industrial character files from tape to disk

```
dd if=/dev/nrst0 ibs=8332 > pstann.ch
```

3.3 Loading old format CRSP character files from tape

The next sequence of commands, can be used to copy CRSP Stocks Monthly calendar and data (old format) files from tape to disk. The *mt* command is used for managing tape storage devices: it can rewind or forward the tape skipping a specified number of files. In this example, we skip the first eight files on the tape which are program files (see the CRSP Stock File Guide. The appendix describes `load.crsp`, a shell script for loading old format CRSP Daily and Monthly files.

Figure 4: Copying multiple CRSP Stocks character files from tape to disk

```
mt -f /dev/nsrt0 rewind
mt -f /dev/nsrt0 fsf 8
dd if=/dev/nsrt0 ibs=31200 > cal.monthly
dd if=/dev/nsrt0 ibs=32000 > monthly.data
```

3.4 Loading new format CRSP Access97 files from CD

The new format CRSP data files are available on CD-ROM; CRSP also provides a setup script that automatices the loading process. Simply insert the data CD, copy the `setup.sh` file from the CD to the base directory where CRSP data is to be loaded, and run the script. You will be prompted to enter directory locations for the CD and where to install the data. Install the Daily and Monthly Stocks files before installing the Indices data (the Indices setup program allows you to "overlay" the Indices data over each of the Daily and Monthly Stocks files, after the latter have been installed, allowing Indices and Stocks data to be accessed at the same time). The installation section of the appendix lists the directories which the PERC toolkit assumes, by default, the data are installed in.

3.5 Loading I/B/E/S character files from CD

I/B/E/S character files are available on CD-ROM, and can be copied using the `cc` command from your CD-drive to disk. The data are distributed among several flat data files, sorted by I/B/E/S company symbol. The appendix lists `load.ibes`, a unix script to copy I/B/E/S character files to the directory `/db/ibes/data`. The files containing Summary and Detail data, respectively, are

Summary		
Domestic (US)	International	Description
hiout1.us	hiout1.int	Consensus Estimates
hiout2.us	hiout2.int	Background
hiout3.us	hiout3.int	Identification
hiout4.us	hiout4.int	Adjustment Factors (for splits)
hiout5.us	hiout5.int	Industry names
hiout6.us	hiout6.int	Currency

Detail		
Domestic (US)	International	Description
actfild.usc	actfili.int	Actual earnings
adjfild.usc	adjfili.int	Adjustment Factors (for splits)
canxfild.usc	canxfili.int	Exchange rates
curfild.usc	curfili.int	Currency
detfild.usc	detfili.int	Detailed estimates
exclfild.usc	exclfili.int	Excluded estimates
idfild.usc	idfili.int	Identification
sigfild.usc	sigfili.int	Industry names
stopfild.usc	stopfili.int	Stopped Estimates

3.6 Conversion to SAS Datasets

SAS has gained popularity amongst universities as a platform on which to access CRSP and Compustat, partly because SAS itself has done much of the work of converting CRSP and Compustat data to SAS datasets by providing a built-in SAS procedure. The data layout of the old format CRSP Stocks files (daily, monthly and excess returns) and Compustat Industrial files (annual and quarterly) have been internally pre-defined in the SAS procedure PROC DATASOURCE, from the SAS/ETS package. Note that the current release of SAS (Version 6.12) only supports the old CRSP Stocks format. All the user needs to do is specify the external input file name (which is simply the original CRSP or Compustat data file on disk or tape), and PROC DATASOURCE does all

the work of converting to a SAS dataset. PROC DATASOURCE also supports other data sources, such as Citibase. Even if your particular data file is not included in PROC DATASOURCE, SAS' DATA step and INPUT statement provide a very convenient facility to read in any character data file, particularly if it is in a simple linear format (such as the Compustat Industrial Segments or the I/B/E/S data files). The Appendix lists SAS programs for installing CRSP Stocks, and Compustat Industrial (annual, quarterly and segments) data as permanent SAS datasets.

If disk space allows, installing all data files as permanent SAS datasets can greatly speed up access to the data. Because the data is already stored in SAS internal format, any SAS program to subset, combine or analyze the data can work quickly and directly. If your SAS version supports the graphical SAS/ASSIST interface, you can even browse the data, and interactively query, extract or run statistical procedures.

If lack of disk space precludes a separate installation as permanent SAS data sets, it is still possible to use the SAS procedures by generating the data sets as needed, converting and keeping only the companies, date ranges or data variables that you require, each time you perform a SAS analysis. The drawback is that the data is actually converted (over and over) again from the original data files, greatly increasing run time.

The following example illustrates the straightforward procedure of using SAS PROC DATASOURCE, to read the CRSP Stocks Monthly file. It begins by specifying the library names where the permanent data sets are to be stored[10], and assumes that the original CRSP character data already reside in `/db/crsp/data`. PROC DATASOURCE is run with the following options:

filetype=crspmcs specifies the type of external file to be converted, in this case the CRSP monthly character files. Refer to the SAS/ETS manual for other file types that SAS supports, or contact SAS Technical Support Services.

infile=(chcal chdata) specifies the actual file names.

out=sasdata.monthly creates the permanent output data set containing the main returns, price and trading volume data contained in CRSP.

outby=saslist.monthly creates the permanent output data set containing company identifier information.

outevent=sasevent.monthly creates permanent output data set containing events information, such as delisting, name changes, distributions and shares outstanding values.

ascii specifies that the raw data are in ASCII, as opposed to other formats such as EBCDIC.

The CRSP data is stored in three datasets, named in the `out=` (containing the main time series data), `outby=` (containing a list of companies, with header information and counts of the number of observations), and `outevent=` (containing events information from the INFO data block in CRSP) options. Any of these datasets can be left out simply by dropping its optional statement.

It is also possible to modify the PROC DATASOURCE options to keep only a subset of the data, or to output the data as temporary working datasets that are not stored permanently on hard

[10]The `partsize=2G` phrase in the first `libname` statement enables SAS to handle large datasets, that are larger than the maximum file size allowed on your operating system. For example, Sun Solaris 2.5.1 requires files to be no larger than 2 Gigabytes, but the CRSP Stocks Daily dataset requires almost 3 Gigabytes. This example directs SAS to treat the `SASDATA` as a single library partitioned into 2 Gigabyte chunks. Other options are possible depending on the file size constraints of your platform, see SAS TS Note TS508 "SAS Solution for Large File Support on 32-bit UNIX Operating System" available from SAS Technical Support or web site `http://www.sas.com`.

disk after each run. This program converts and only keeps stock returns data for a specified date range (PROC DATASOURCE keeps key information such as PERMNO and date automatically).

Figure 5: PROC DATASOURCE and CRSP datasets

```
libname sasdata '/db/crsp/data' partsize=2G;
libname sasevent '/db/crsp/data/event';
libname saslist '/db/crsp/data/list';
filename chdata '/db/crsp/data/monthly.dat';
filename chcal '/db/crsp/data/cal.monthly';

proc datasource filetype=crspmcs infile=(chcal chdata)
out=work.monthly outby=saslist.monthly outevent=sasevent.monthly;
where date between '01jan92'd and '31dec96'd;
keep ret;
```

The SAS program above assumes that data are to be read from character files that have been copied to disk. SAS can also read data more generally from "pipes", that is any stream generated from perhaps another program. For example, if the character data are stored in compressed form on disk, you may specify that the data be uncompressed (using, say the standard unix `zcat` command) and piped to PROC DATASOURCE, by replacing the filename statement, as follows :

```
filename chdata pipe 'zcat /db/scratch/monthly.data.Z';
```

External files can also be read directly from other devices, such as tape drive, e.g.

```
filename chdata tape '/dev/rmt/0 lrecl=400 blksize=32000 recfm=f';
```

Some SAS precodures and data step commands, such as PROC UNIVARIATE and merge statements, that work on subcategories of the data, require that the data set first be ordered by the category. Indexing a data set enables these SAS procedures to be used without sorting the data set. It is possible to create multiple different indexes for a data set. Indexes can be simple, or composite comprising more than one key variable. Indexes are physically separate files in SAS data libraries, but are treated as extensions of the associated data set. It can be useful in the installation process to index data sets by commonly-used key variables, such as industry code, years or identifiers. Indexes are created using the PROC DATASETS command, an example of which is shown below. This SAS program associates several simple indexes for the variables NCUSIP (historical name cusip), TICKER (historical name ticker symbol), PERMNO (CRSP Permanent Number), CUSIP (header cusip) and SICCD (historical SIC code) with the CRSP Stocks monthly events data set. The event SAS data set `monthly` is assumed to have been installed in the SAS library named `SASEVENT`.

Figure 6: Creating SAS indexes for CRSP events with PROC DATASETS

```
proc datasets library=sasevent;
  modify monthly;
  index create ncusip;
  index creater ticker;
```

```
  index create permno;
  index create cusip;
  index create siccd;
run;
```

---

4 Basic Access

This section describes basic methods of accessing the data files using sequential C programs or a SAS environment. Some data vendors (CRSP) provided basic subroutines and sample access programs; for other data sets, original access programs and routines had to be developed. These access routines and methodology were designed to be as uniform as possible across heterogeneous databases. I also discuss how to manage and build up a library of shareable subroutines to aid data access and analysis.

4.1 CRSP environment variables

CRSP Access97 routines require that a number of *environment variables* be available from which to determine the names and locations of data and output files. The CRSP manual provides instructions on how to setup environment variables during installation. Alternatively, we can use the following function instead, at the start of each application program, to define these *environment variables* if they have not already been initialised. The actual path string values are defined in the PERC header file `db_local.h`; if you did not use the default PERC directories in your installation, you will need to change the path values and recompile the programs.

Figure 7: Initializing CRSP environment variables

```
void aux_put_env(char *s, char *p)
{
  char *e;
  if(!getenv(s)) {
    e=(char *)malloc(strlen(s)+strlen(p)+2);
    sprintf(e,"%s=%s",s,p);
    if(putenv(e)) db_err(stderr,e);
  }
}

void crsp_init_env()
{
  aux_put_env("CRSP_ROOT",DB_CRSP_ROOT);
  aux_put_env("CRSP_LOG",DB_CRSP_LOG);
  aux_put_env("CRSP_DSTK",DB_CRSP_DSTK);
  aux_put_env("CRSP_MSTK",DB_CRSP_MSTK);
  aux_put_env("CRSP_CST",DB_CRSP_CST);
  aux_put_env("CRSP_LIB",DB_CRSP_LIB);
  aux_put_env("CRSP_BIN",DB_CRSP_BIN);
  aux_put_env("CRSP_INCLUDE",DB_CRSP_INCLUDE);
  aux_put_env("CRSP_SAMPLE",DB_CRSP_SAMPLE);
  aux_put_env("CRSP_ENV_ROOT",DB_CRSP_ENV_ROOT);
  aux_put_env("CRSP_ENV_PATH",DB_CRSP_ENV_PATH);
  aux_put_env("CRSP_ENV_INIT",DB_CRSP_ENV_INIT);
  aux_put_env("CRSP_ENV_ELOG",DB_CRSP_ENV_ELOG);
  aux_put_env("CRSP_ENV_ULOG",DB_CRSP_ENV_ULOG);
  aux_put_env("CRSP_ENV_EMSG",DB_CRSP_ENV_EMSG);
  aux_put_env("CRSP_ENV_LMSG",DB_CRSP_ENV_LMSG);
}
```

4.2 Application example: Extracting stock distribution events

A sequential access program simply reads all data for the first security, then marches through the database one security at a time. CRSP provides sample C and Fortran programs and subroutines, that you can modify and use; however access programs for the other databases are not provided by the vendors. As an example, the following program reads in CRSP Stocks Monthly data sequentially company-by-company, and prints out the distributions records that match a range of distribution events betwen two specified dates.

Figure 8: CRSP sequential access program to display distribution events

```
#include "db.h"

#define beg 19780101      /* beginning and ending date range */
#define end 19971231
#define begcd 5000        /* distribution codes (5xxx = stocks splits/dividends) */
#define endcd 5999

CRSP_STK_STRUCT stk;
int main (int argc, char *argv[])
{
  int stkcrspnum;

  int perm;                     /* permno loaded from file   */
  int ret;                      /* return value from read    */
  int i;

  /*---------------------------------------------------------------------------
    1.  Initialise CRSP environment
  ---------------------------------------------------------------------------*/
  crsp_init_env();

  stkcrspnum = crsp_stk_open(DB_CRSP_MSTK,20,&stk,STK_ALL,"r",1);
  if (stkcrspnum == CRSP_FAIL) db_err(stderr,err_msg);

  fprintf(stdout,"%5s %4s %11s %11s %11s %8s %8s %8s %8s %5s\n",
          "PermN","Dist","DivAmt","FacPr","FacShr",
          "DclrDt","ExDt","RcrdDt","PayDt","AcPrm");

  /*---------------------------------------------------------------------------
    2.  loop over each query from stdin
  ---------------------------------------------------------------------------*/
  while((ret=crsp_stk_read(stkcrspnum,20,&perm,CRSP_NEXT,&stk,STK_ALL))!=
        CRSP_EOF) {
    for(i=0;i<stk.events.dists_arr->num;i++) {
      if (stk.events.dists[i].exdt>=beg&&
          stk.events.dists[i].exdt<=end&&
          stk.events.dists[i].distcd>=begcd&&
          stk.events.dists[i].distcd<=endcd) {
        fprintf(stdout,"%5d %4d %11.5f %11.5f %11.5f %8d %8d %8d %8d %5d\n",
                stk.header->permno,
                stk.events.dists[i].distcd,stk.events.dists[i].divamt,
                stk.events.dists[i].facpr,stk.events.dists[i].facshr,
                stk.events.dists[i].dclrdt,stk.events.dists[i].exdt,
                stk.events.dists[i].rcrddt,stk.events.dists[i].paydt,
```

```
                stk.events.dists[i].acperm);
      }
    }
  }
}
```

4.3 Compustat Industrial

Access programs are not provided by the data vendor. This subsection defines a new data structure to hold Compustat Industrial Annual data (similar structures have also been defined for Quarterly and Segments data), and a routine to read from the Compustat character data files.

`CSTANN` is a data structure to hold a company's entire history of annual industrial data. The field `y` is an array, each of which holds one year of annual data; in turn, the field `d` is an array of real numbers corresponding to a data item (numbered 1 through 350). Also the field `yeara` holds the *calendar year end* date of each year's data. The company's identification information in the `CRSP_CST_HEADER` header record (which is identical the structure defined in the CRSP Access97 header source files[11], and defined below), `h`.

Figure 9: Structure for Compustat Industrial Annual company data

```
typedef struct {
    int gvkey;              /* permanent record identifier, primary key */
    int iperm;              /* header crsp issue permno link            */
    int icomp;              /* header crsp company permco link          */
    int begyr;              /* annual date of earliest data (yyyymmdd)  */
    int endyr;              /* annual date of latest data (yyyymmdd)    */
    int begqtr;             /* quarterly date of earliest data (yyyy.q) */
    int endqtr;             /* quarterly date of latest data (yyyy.q)   */
    int availflag;          /* Code of available Compustat data types   */
    int dnum;               /* industry code                            */
    int file;               /* file identification codes                */
    int zlist;              /* exchange listing + S&P Index code        */
    int state;              /* state identification code                */
    int county;             /* county identification code               */
    int stinc;              /* state incorporation code                 */
    int finc;               /* foreign incorporation code               */
    int xrel;               /* S&P Industry Index relative code         */
    int stk;                /* Stock ownership code                     */
    int dup;                /* Duplicate file code                      */
    int ccndx;              /* Current Canadian Index Code              */
    int fundf[CRSP_CST_FILE_CNT]; /* fundamental file identification codes */
    char cpspin[CRSP_CSS_LEN];  /* Primary S&P index marker             */
    char csspin[CRSP_CSS_LEN];  /* Secondary S&P index marker           */
    char csspii[CRSP_CSS_LEN];  /* Subset S&P index marker              */
    char subdbt[CRSP_CSS_LEN];  /* Current S&P Subordinated Debt Rating */
    char cpaper[CRSP_CSS_LEN];  /* Current S&P Commercial Paper Rating  */
```

[11] CRSP maintains, internally for University of Chicago use, a merged CRSP/Compustat database, which is not available externally. PERC attempts to add to the available CRSP Access97 product much of the same capability to simultaneously access Compustat as well I/B/E/S data files.

```
    char sdbt[CRSP_CSS_LEN];    /* Current S&P Senior Debt Rating       */
    char sdbtim[CRSP_CSS_LEN];  /* Current S&P Senior Debt Rating-Footnote */
    char cnum[CRSP_CUSIP_LEN];          /* CUSIP issuer code            */
    char cic[CRSP_CSS_LEN];             /* issuer number                */
    char coname[CRSP_COMNAM_LEN];       /* Company name                 */
    char iname[CRSP_COMNAM_LEN];        /* Industry name                */
    char smbl[CRSP_TICKER_LEN];         /* Stock ticker symbol          */
    char ein[CRSP_CUSIP_LEN];   /* Employer Identification Number       */
} CRSP_CST_HEADER;

typedef struct cst_ann_struct {
  CRSP_CST_HEADER hdr;       /* identification information */
  char cusip[16];
  struct yeardat {
    float data[384];            /* data items */
    CRSP_CST_FTNT a[36],b[36];  /* data footnotes */
    int source;
    int yr,                     /* fiscal year YYYY */
        fyr,                    /* fiscal yearend month MM */
        yeara,                  /* calendar year end YYYYMMDD */
        ucode;                  /* update code */
  } y[MAXCSTANN];            /* each year's data */
} CSTANN;
```

The function `cst_open` is used to open a Compustat Industrial data file and an associated index file for random access, if any.

Figure 10: Routine to initialise Compustat Industrial data files

```
typedef struct cst_db {
  FILE *fp;
  DB_INDEX x[MAXCSTINDEX];
  int nx;
} CST_DB;

CST_DB *cst_open(char *flat, char *index)
{
  CST_DB *db;
  FILE *fp;
  db=(CST_DB *) db_malloc(sizeof(CST_DB));
  db->fp=fopen(flat,"r");
  if(!db->fp) db_err(stderr,flat);
  if(*index&&(fp=fopen(index,"r"))) {
    for(db->nx=0;fscanf(fp,"%s %d",db->x[db->nx].key,&(db->x[db->nx].loc))==2;
        db->nx++)
      if(db->nx>=MAXCSTINDEX) db_err(stderr,"MAXCSTINDEX exceeded (modify db_lo
cal.h and recompile!");
    qsort(db->x,db->nx,sizeof(DB_INDEX),db_cmp);
    fclose(fp);
  }
  return(db);
}
```

The function `CST_ANN_READ` is used to read Compustat Industrial Annual data from previously-opened Compustat data files, by company, either sequentially (if the input parameter `query` is NULL) or via random access: the latter method is described in the next section (this routine can read also the entire history of annual data for each company from multiple files spanning different periods of coverage). It also converts the company's fiscal year end in Compustat convention to a calendar year end date, which is stored in the field `yeara`.

Figure 11: Routine to read Compustat Industrial Annual data

```
static int cst_read_record(FILE *fp,char *s,int len)
{
  int flag;
  static char buf[16];
  while(flag=fread(&(s[1]),len,1,fp)) {
    if(strcmp(mkstring(buf,s,5,6),"000000")) return(1);
  }
  return(0);
}
int cst_ann_read(CST_DB *db[],int cstdesc[],int ndb,CSTANN **cst,char *query)
{
  int i,j,k,l,n,len,idb,found;
  float dec,val;
  static char buf[16384];
  if(*cst==NULL) *cst=(CSTANN *)db_malloc(sizeof(CSTANN));
  memset(*cst,0,sizeof(CSTANN));

  for(found=idb=0;idb<ndb;idb++) {

    if(!query||!strlen(query)||!(db[idb]->nx)||
       !db_seek(db[idb]->fp,db[idb]->x,db[idb]->nx,query)) {
      found=1;
      for(i=0;i<2;i++) {
        for(j=0;j<4;j++) {
          if(cst_read_record(db[idb]->fp,buf,ANNLEN)) {
            (*cst)->hdr.dnum=mkint(buf,1,4);
            mkstring((*cst)->hdr.cnum,buf,5,6);
            mkstring((*cst)->hdr.cic,buf,11,3);
            sprintf((*cst)->cusip,"%-6.6s%-2.2s",(*cst)->hdr.cnum,(*cst)->hdr.cic);
            (*cst)->hdr.file=mkint(buf,15,2);
            if(i==0) {
              (*cst)->hdr.zlist=mkint(buf,17,2);
              mkstring(j==0||j==2?(*cst)->hdr.coname:(*cst)->hdr.iname,buf,19,28);
              mkstring((*cst)->hdr.smbl,buf,47,8);
              for(k=0;k<5;k++) {
                (*cst)->y[(j*5)+k+cstdesc[idb]].fyr=mkint(buf,55+(k*2),2);
                (*cst)->y[(j*5)+k+cstdesc[idb]].yeara=mkint(buf,65+(k*2),2);
                (*cst)->y[(j*5)+k+cstdesc[idb]].ucode=mkint(buf,88+k,1);
                if((*cst)->y[(j*5)+k+cstdesc[idb]].fyr>0&&
                   (*cst)->y[(j*5)+k+cstdesc[idb]].yeara>0&&
                   (*cst)->y[(j*5)+k+cstdesc[idb]].ucode>0) {
                  (*cst)->y[(j*5)+k+cstdesc[idb]].yr=
                    19000031+((*cst)->y[(j*5)+k+cstdesc[idb]].yeara*10000)+
                    ((*cst)->y[(j*5)+k+cstdesc[idb]].fyr*100)+
```

```
          ((*cst)->y[(j*5)+k+cstdesc[idb]].fyr<=5?10000:0);
        switch((*cst)->y[(j*5)+k+cstdesc[idb]].fyr) {
        case 4:
        case 6:
        case 9:
        case 11: (*cst)->y[(j*5)+k+cstdesc[idb]].yr-=1; break;
        case 2:
          if((((*cst)->y[(j*5)+k+cstdesc[idb]].yr)/10000)%4)
            (*cst)->y[(j*5)+k+cstdesc[idb]].yr-=3;
          else
            (*cst)->y[(j*5)+k+cstdesc[idb]].yr-=2;
          break;
        }
        if((*cst)->hdr.begyr==0||
           (*cst)->y[(j*5)+k+cstdesc[idb]].yr<(*cst)->hdr.begyr)
          (*cst)->hdr.begyr=(*cst)->y[(j*5)+k+cstdesc[idb]].yr;
        if((*cst)->y[(j*5)+k+cstdesc[idb]].yr>(*cst)->hdr.endyr)
          (*cst)->hdr.endyr=(*cst)->y[(j*5)+k+cstdesc[idb]].yr;
      }
    }
    (*cst)->hdr.xrel=mkint(buf,75,4);
    (*cst)->hdr.stk=mkint(buf,79,1);
    (*cst)->hdr.dup=mkint(buf,80,2);
  }
  else {
    (*cst)->hdr.state=mkint(buf,17,2);
    (*cst)->hdr.county=mkint(buf,19,2);
    (*cst)->hdr.finc=mkint(buf,22,2);
    for(k=0;k<5;k++) {
      (*cst)->y[(j*5)+k+cstdesc[idb]].source=mkint(buf,24+(k*2),2);
    }
    mkstring((*cst)->hdr.cpspin,buf,34,1);
    mkstring((*cst)->hdr.csspin,buf,35,2);
    mkstring((*cst)->hdr.csspii,buf,37,1);
    mkstring((*cst)->hdr.sdbt,buf,38,2);
    mkstring((*cst)->hdr.sdbtim,buf,40,2);
    mkstring((*cst)->hdr.subdbt,buf,42,2);
    mkstring((*cst)->hdr.cpaper,buf,44,3);
    mkstring((*cst)->hdr.ein,buf,53,10);
  }
  for(k=0;k<5;k++) {
    for(l=0;l<35;l++) {
      mkstring(i==0?(*cst)->y[(j*5)+k+cstdesc[idb]].a[l+1].ftnt:
               (*cst)->y[(j*5)+k+cstdesc[idb]].b[l+1].ftnt,buf,93+(l*2),2);
    }
  }
  for(n=443,k=0;k<5;k++) {
    for(l=(i*175)+1;l<=(i+1)*175;l++) {
      sscanf(cst_ann_title[l].f,"%%f%d.%f",&len,&dec);
      val=mkfloat(buf,n,len,dec);
      if(val>=-0.0091 && val <=-0.0009) val+=-99.0;
      (*cst)->y[(j*5)+k+cstdesc[idb]].data[l]=val;
      n+=len;
    }
  }
 }
 else return(0);
}
```

```
      }
    }
  }
  return(found);
}
```

4.4 I/B/E/S

Access programs are not provided by the data vendor. This subsection defines a new data structure to hold I/B/E/S data for each company, combining Summary and Detail information (Domestic and International versions are identically defined), and a routine to read from the I/B/E/S character data files.

`IBES` is a combined data structure to hold a company's entire history of Detail and Summary data. Most of the data are stored in the following array fields:

did identification records (Detail)

act actual reported earnings (Detail)

dadj split adjustment factors (Detail)

det analyst-by-analyst estimates (Detail)

stp stopped estimates (Detail)

sum consensus estimates (Summary)

bak background data (Summary)

sid identification records (Summary)

sadj split adjustment factors (Summary)

The field `n` is an array containing the total number of records in each of the data arrays listed above, indexed by the following enumeration IDFIL, ACTFIL, ADJFIL, DETFIL, STOPFIL, SUMHI, BAKHI, IDHI and ADJHI.

Figure 12: Structure for I/B/E/S Detail and Summary company data

```
typedef struct {
  char ibes[9];
  long brec[9],nrec[9];
} IB_INDEX;

typedef struct ib_dbstruct {
  FILE *fileptr[9];
  IB_INDEX sec[MAXIBES];
  int n;
  int curr;
  int filemax[9];
} IB_DB;
```

```
typedef struct sumstruct {
  char ibes[7];
  int stat,end;
  char fpi;
  int estimate,up,down;
  float median,mean,std,high,low;
} SUM;

typedef struct bakstruct {
  char ibes[7];
  int stat;
  int dec;
  float price;
  int day;
  float shares;
  struct bakactstruct {
    int end;
    float actual;
    char reported;
  } f,q;
  float div,grow,stab;
} BAK;

typedef struct actstruct {
  char ibes[7],measure[4],periodicity[4];
  int end,report;
  float value;
} ACT;

typedef struct adjstruct {
  char ibes[7];
  int split,stat;
  float factor;
} ADJ;

typedef struct idstruct {
  char ibes[7],cusip[9],ticker[7],name[17];
  float dilution;
  char pdi,flag,mscip,uniform;
  int sig,start;
} IDN;

typedef struct detstruct {
  char ibes[6];
  int broker,analyst;
  char currency,pdf,fpi;
  char measure[4];
  int end,estimate,review;
  float value;
  char fx[4];
} DET;

typedef struct stoppedstruct {
  char ibes[6];
  int broker;
  char periodicity,measure[4];
  int end,stop;
```

```
} STP;

#define IDFIL 0
#define ACTFIL 1
#define ADJFIL 2
#define DETFIL 3
#define STOPFIL 4
#define SUMHI 5
#define BAKHI 6
#define IDHI 7
#define ADJHI 8

typedef struct detailstruct {
  int n[9];
  char ibes[7],cusip[9];
  char DEFCTY[4];
  ACT *act;
  ADJ *dadj,*sadj;
  DET *det;
  IDN *did,*sid;
  STP *stp;
  SUM *sum;
  BAK *bak;
} IBES;
```

The original I/B/E/S data are stored in separate character files which need to be read simultaneously, in order to access all data items for a company. To accomplish this, I first create indexes during the initial data installation process (using the following Perl script), identifying the starting location for each company's information in each file.

Figure 13: Creating indexes for a flat data file

```
#!/usr/local/bin/Perl
$|=1;
open(IFP,$ARGV[0]) || die "cannot open input file\n";
open(OFP,sprintf(">%s.index",$ARGV[0])) || die "cannot open output file\n";
for($loc=0,$prev="";<IFP>;$prev=$datum[0],$loc=tell(IFP)) {
    @datum=split(' ',$_);
    printf(OFP "%-8.8s%11d\n",$datum[0],$loc) if ($datum[0] ne $prev && length($datum[0]) > 0);
}
```

The function `ib_init` is called at the start of an application program to read in these index files into memory, and merges them by company ticker.

Figure 14: Reading index files to initialise access to I/B/E/S character data files

```
typedef struct ib_dbstruct {
  FILE *fileptr[9];
  IB_INDEX sec[MAXIBES];
```

```
  int n;
  int curr;
  int filemax[9];
} IB_DB;

IB_DB *ib_init(char intl)
{
  FILE *fp,*efp;
  int i,n,m;
  IB_INDEX *ptr,dum,*prev;
  IB_DB *ibd;

  if(toupper(intl)=='I') {
    international=intl;
    initintl();
  }
  ibd=db_malloc(sizeof(IB_DB));

  efp=fopen("instibes.log","w");
  fp=openf(IBESDIR,filedex[IDFIL],"r");
  for(ibd->n=0;
      fscanf(fp,"%s%ld",ibd->sec[ibd->n].ibes,&(ibd->sec[ibd->n].brec[IDFIL]))==2;
      ibd->n++){
    if(ibd->n>=MAXIBES) db_err(stderr,"MAXIBES exceeded (modify db_local.h and recompile)");
  }
  fclose(fp);
  qsort(ibd->sec,ibd->n,sizeof(IB_INDEX),cmpibes);
  fprintf(stderr,"Read %d securities from %s\n",ibd->n,filename[IDFIL]);

  fp=openf(IBESDIR,filedex[IDHI],"r");
  for(m=0,n=ibd->n;
      fscanf(fp,"%s%ld",ibd->sec[n].ibes,&(ibd->sec[n].brec[IDHI]))==2;
      m++){
    ptr=(IB_INDEX *) bsearch(&(ibd->sec[n]),ibd->sec,ibd->n,sizeof(IB_INDEX),cmpibes);
    if(!ptr) n++;
    if(n>=MAXIBES) db_err(stderr,"MAXIBES exceeded (modify db_local.h and recompile)");
  }
  fclose(fp);
  fprintf(stderr,"Read %d securities from %s, added %d to %d\n",
          m,filename[IDHI],n-ibd->n,n);
  ibd->n=n;
  qsort(ibd->sec,ibd->n,sizeof(IB_INDEX),cmpibes);

  for(i=0;i<9;i++) {
    fp=openf(IBESDIR,filedex[i],"r");
    fprintf(efp,"%s\n",filedex[i]);
    for(prev=NULL,ibd->filemax[i]=m=n=0;
        fscanf(fp,"%s %ld",dum.ibes,&(dum.brec[i]))==2;
        n++) {
      ptr=(IB_INDEX *) bsearch(&dum,ibd->sec,ibd->n,sizeof(IB_INDEX),cmpibes);
      if(ptr) {
        m++;
        ptr->brec[i]=dum.brec[i];
      }
      else fprintf(efp,"%s %d\n",dum.ibes,dum.brec[i]);
      if(prev) {
        prev->nrec[i]=(dum.brec[i]-prev->brec[i])/filesize[i];
        if(prev->nrec[i]>ibd->filemax[i]) ibd->filemax[i]=prev->nrec[i];
```

```
      }
      prev=ptr;
    }
    fclose(fp);

    sprintf(buf,"%s%s",IBESDIR,filename[i]);
    if(prev) {
      prev->nrec[i]=(filebytes(buf)-prev->brec[i])/filesize[i];
      if(prev->nrec[i]>ibd->filemax[i]) ibd->filemax[i]=prev->nrec[i];
    }
    fprintf(stderr,"Found %d/%d securities from %s\n",m,n,filename[i]);
  }

  for(i=0;i<9;i++) ibd->fileptr[i]=openf(IBESDIR,filename[i],"r");
  if(ib_etag==NULL) ib_etag=(int *) calloc(ibd->filemax[DETFIL],sizeof(int));
  if(ib_btag==NULL) ib_btag=(int *) calloc(ibd->filemax[DETFIL],sizeof(int));

  for(n=ibd->n-2;n<ibd->n;n++) {
    fprintf(stderr,"%-6.6s",ibd->sec[n].ibes);
    for(i=0;i<9;i++)
      fprintf(stderr," %9ld %4d",ibd->sec[n].brec[i],ibd->sec[n].nrec[i]);
    fprintf(stderr,"\n");
  }
  fprintf(stderr,"SIG Codes %d\n",dinit_sig("sigfild.usc"));
  fclose(efp);
  return(ibd);
}
```

The function `ib_get` uses these merged indexes to read in all data for a company from the separate character data files. In the next section, I will describe how to merge the data and save as a single large binary file: thus instead of searching and reading from nine separate character files, we will only need to access a single binary file.

Figure 15: Reading I/B/E/S character data files

```
int ib_get(IB_DB *ibd,IBES **ib)
{
  int i,j;
  if(!(*ib)) *ib=ib_new(ibd);
  (*ib)->ibes[0]=(*ib)->cusip[0]='\0';

  for(i=0;i<9;i++) (*ib)->n[i]=0;
  for(i=0;i<9;i++) {
    (*ib)->n[i]=ibd->sec[ibd->curr].nrec[i];
    if((*ib)->n[i]>ibd->filemax[i]) db_err(stderr,titles[i]);
  }

  if (ibd->curr >= ibd->n) return(0);
  strcpy((*ib)->ibes,ibd->sec[ibd->curr].ibes);
  fseek(ibd->fileptr[IDFIL],ibd->sec[ibd->curr].brec[IDFIL],0);

  for(i=0;i<(*ib)->n[IDFIL];i++) {
    if(!dget_idn(ibd->fileptr[IDFIL],&((*ib)->did[i]))) db_err(stderr,"Get Detail ID");
    if(strlen((*ib)->did[i].cusip)>=8) strcpy((*ib)->cusip,(*ib)->did[i].cusip);
```

```
  }

  fseek(ibd->fileptr[ACTFIL],ibd->sec[ibd->curr].brec[ACTFIL],0);
  for(i=0;i<(*ib)->n[ACTFIL];i++) {
    if(!dget_act(ibd->fileptr[ACTFIL],&((*ib)->act[i]))) db_err(stderr,"Get Detail ACT");

    if((*ib)->act[i].report==0) {
      for(j=i-1;j>=0;j--) {
        if((*ib)->act[i].end==(*ib)->act[j].end&&(*ib)->act[i].report!=0)
          (*ib)->act[i].report=(*ib)->act[j].report;
      }
    }
  }

  fseek(ibd->fileptr[ADJFIL],ibd->sec[ibd->curr].brec[ADJFIL],0);
  for(i=0;i<(*ib)->n[ADJFIL];i++) {
    if(!dget_adj(ibd->fileptr[ADJFIL],&((*ib)->dadj[i]))) db_err(stderr,"Get Detail ADJ");
  }

  fseek(ibd->fileptr[DETFIL],ibd->sec[ibd->curr].brec[DETFIL],0);
  for(i=0;i<(*ib)->n[DETFIL];i++) {
    if(!dget_det(ibd->fileptr[DETFIL],&((*ib)->det[i]))) db_err(stderr,"Get Detail DET");
  }

  fseek(ibd->fileptr[STOPFIL],ibd->sec[ibd->curr].brec[STOPFIL],0);
  for(i=0;i<(*ib)->n[STOPFIL];i++) {
    if(!dget_stp(ibd->fileptr[STOPFIL],&((*ib)->stp[i]))) db_err(stderr,"Get Detail STOP");
  }

  fseek(ibd->fileptr[SUMHI],ibd->sec[ibd->curr].brec[SUMHI],0);
  for(i=0;i<(*ib)->n[SUMHI];i++) {
    if(!sget_sum(ibd->fileptr[SUMHI],&((*ib)->sum[i]))) db_err(stderr,"Get Summary SUM");
  }

  fseek(ibd->fileptr[BAKHI],ibd->sec[ibd->curr].brec[BAKHI],0);
  for(i=0;i<(*ib)->n[BAKHI];i++) {
    if(!sget_bak(ibd->fileptr[BAKHI],&((*ib)->bak[i]))) db_err(stderr,"Get Summary BAK");
  }

  fseek(ibd->fileptr[IDHI],ibd->sec[ibd->curr].brec[IDHI],0);
  for(i=0;i<(*ib)->n[IDHI];i++) {
    if(!sget_idn(ibd->fileptr[IDHI],&((*ib)->sid[i]))) db_err(stderr,"Get Summary ID");
    if(strlen((*ib)->sid[i].cusip)>=8) strcpy((*ib)->cusip,(*ib)->sid[i].cusip);
  }

  fseek(ibd->fileptr[ADJHI],ibd->sec[ibd->curr].brec[ADJHI],0);
  for(i=0;i<(*ib)->n[ADJHI];i++) {
    if(!sget_adj(ibd->fileptr[ADJHI],&((*ib)->sadj[i]))) db_err(stderr,"Get Summary ADJ");
  }

  ibd->curr++;
  return(1);
}
```

4.5 Managing Subroutines

CRSP distributes a number of very useful utility subroutines, in the library `$CRSP_ROOT/crsplib.a`. Understanding and using these subroutines can save much programming time. Additionally, the PERC toolkit provides a collection of new subroutines that enhance access not only to CRSP, but also to Compustat and I/B/E/S.

Unix provides a facility to make pre-defined subroutines available to any user by precompiling and storing them in "random libraries". For example, the system administrator can use the following unix commands to create an archive of the subroutines in a library file called `db.a`. The `ar` command creates a library archive, while `ranlib` converts the archive into a random library that can be linked to application programs more rapidly.

```
ar -s -r -u -c -v db.a db.o compustat.o ibes.o crsp.o
ranlib db.a
```

To aid the process of compiling a user program and linking with library routines, the following `tcc` script can be made available to users. This script essentially extends the standard `cc` C Compiler command: any command line arguments are also passed on. It automatically links the PERC, CRSP and math libraries, and specifies additional paths where PERC and CRSP header files can be found.

Figure 16: Script to automate application program compilation and linking

```
cc $* -DUNIX=1 -I/db/crsp/include -I/db/perc/src/include /db/perc/src/lib/db.a /db/crsp/acclib/crsplib.a -lm
```

Additional subroutines, which other users may have written and found useful, can be collected and made available through the library archive. The appendix lists the subroutines which comprise the PERC toolkit. They have been collected in source files named `compustat.c`, `crsp.c`, `ibes.c` and `db.c`, and made available to application programs in a library archive named `db.a`. A "bank" of such subroutines that perform common tasks can be expanded and shared, greatly reducing users' programming efforts.

4.6 Generating weekly stocks data

Weekly data can be constructed by aggregating daily CRSP Stocks data. Sampling stocks returns and prices at a weekly frequency reduces the effects of some undesirable properties associated with higher frequency daily data, such as non-trading periods and asynchronous prices[12]. To generate weekly returns from daily returns, first choose a particular weekday (say Wednesday) to mark the end of a week. Given a day-of-week, the function `crsp_cal_weeks` generates two vectors containing the beginning and ending daily date indices for weeks ending on the specified day, using the subroutine `crsp_cal_days` (which returns the number of calendar days elapsed since December 31, 1899 – a Sunday). To generate weekly returns, compound daily returns over the days in the week between the beginning and ending daily date indexes, inclusive (in our example, by compounding daily returns from Thursday to the following Wednesday where available). Similarly, weekly trading

[12]For an analysis of non-synchronous trading, see Lo and MacKinlay (1990) or Campbell, Lo and MacKinlay (1997). Lo and Wang (1999) and Adamek, Lim, Lo and Wang (1998) examine statistical properties of weekly stock returns and trading volume.

volume merely cumulates daily trading volume over the same days, and weekly closing price is given by the ending day's closing price.

Figure 17: Constructing weekly calendar of trading days

```
int crsp_cal_days(int ymd)
{
  struct tm t;
  int i,d;
  memset(&t,0,sizeof(t));
  t.tm_mday=ymd%100;
  t.tm_mon=((ymd/100)%100)-1;
  t.tm_year=(ymd/10000)-1900;
  if(t.tm_year<0) return(-1);
  while(t.tm_year<70) t.tm_year+=28;
  mktime(&t);
  t.tm_year=(ymd/10000)-1900;
  for(d=i=0;i<t.tm_year;d+=365,i++) if(i&&!(i%4)) d++;
  return(d+t.tm_yday+1);
}

int crsp_cal_weeks(CRSP_CAL *cal,int dow,int **begw,int **endw)
{
  int i,cur;
  int beg,end;
  beg=(crsp_cal_days(cal->caldt[1])-(dow+1))/7;
  end=(crsp_cal_days(cal->caldt[cal->ndays])-(dow+1))/7;
  if(*begw) free(*begw);
  if(*endw) free(*endw);
  *begw=(int *) db_malloc(sizeof(int)*(end-beg+2));
  *endw=(int *) db_malloc(sizeof(int)*(end-beg+2));
  for((*begw)[0]=(*endw)[0]=0,i=1;i<=cal->ndays;i++) {
    cur=(crsp_cal_days(cal->caldt[i])-(dow+1))/7;
    if((*begw)[cur-beg]==0) {
      (*begw)[cur-beg]=i;
      (*begw)[cur-beg+1]=0;
    }
    (*endw)[cur-beg]=i;
  }

  return(end-beg+1);
}
```

4.7 Using SAS

When resources permit, SAS provides a complementary method of accessing the databases. The SAS language and its built-in procedures are well suited for a number of applications. By installing the databases as SAS datasets, it becomes possible to directly extract data and perform analysis using SAS procedures. For cross-sectional analysis in particular (for example, extracting and processing values for a specific data item and/or daterange across all securities), SAS programs can be more efficient to write and run.

The previous section of this report described how to convert CRSP and Compustat data files (using either SAS PROC DATASOURCE, or DATA step and INPUT statements if SAS does not directly support the data format), and the disk space required for each dataset. It is possible to use SAS without initially storing as permanent data sets; the data can be kept in its original source format (perhaps in compressed or zipped form) and converted "on-the-fly" at the beginning of each SAS run. However, installing as permanent data sets allows the use of other SAS facilities such as SAS/ASSIST and indexing.

SAS/ASSIST is an interactive graphical interface to SAS. It can be used to list, browse or even edit SAS data sets. It also automatically generates SAS code (which can be saved for re-use later) by guiding users through a point-and-click interface to select SAS procedures and options. These features are particularly useful for researchers who have some familiarity with the databases and what they contain, but are not yet SAS jocks. They can still harness much of the functionality of SAS, to extract or subset data and perform data analysis.

5 Implementing Random Access

Sequential access programs require that the entire database be read even if the user only wishes to study a subset of securities. Random access subroutines enable users' programs to directly pull out data for specified securities (by identifiers such as TICKER, CUSIP, or PERMNO) quickly without having to read in other securities. This section describes how to augment flat data files with index files that drive random access routines. The appendix lists installation scripts that set up index files for CRSP (based on historical stock tickers: CRSP already provides random access routines based on PERMNO, header cusip, historical cusip and header ticker), Compustat (Industrial annual, quarterly and business segments), and I/B/E/S Summary and Detail files.

The methodology I have chosen to implement random access does not rely on data records being of fixed length; they can be of any varying length. Instead I use system routines that report the byte offset of the data file position, and that reposition the data file pointer to a specified byte offset. In the C language, these two standard routines are `ftell` and `fseek` respectively. The integer function `ftell` is used to create index files associating a company's identifiers with its position in the data file. The random access subroutines can then refer to these index files to lookup a specified security's position, and use `fseek` to reposition to the desired security's data in the data file.

Hence, implementing random access involves just the following two steps, which are described in greater detail in the following subsections.

1. Initially creating a (sorted) index file, that associates index keys (such as cusips or tickers) with the record's position in the data file.

2. Providing the user with a subroutine that takes a key (cusip or ticker), looks up the key in the index file, and then locates the desired record in the data file by using its file position.

Having the ability to read data via random access can greatly speed up many applications. It also enables the implementation of two important enhancements to the way these databases are accessed. First, it becomes very easy to provide a friendly user interface, such as using web browsers, to allow users with fairly straightforward data requests to quickly download the data they need, without any programming. Secondly, by using random access routines to access multiple database simultaneously, it is possible to read in and process data for a company spanning more than one database: the C programs to perform this sort of "simultaneous access" are essentially combinations of the random access programs that read individual databases, with minor modifications. The next three subsections detail how to implement these enhancements for the CRSP, Compustat and I/B/E/S files.

5.1 CRSP Stocks

CRSP provides a number of random access routines that can read securities information either by 5-digit PERMNO (CRSP's unique security identifier), 8-character header or historical CUSIP, or header ticker symbol. This section describes how to supplement these with a new random access routine that uses historical tickers, and also how to combine these different random access routines into one general routine.

First, I augment the CRSP Stocks data with a tickers list containing stocks' historical tickers, PERMNO and latest valid date using the following installation program `instcrsp`. As a byproduct, a listing of all securities' names records is also produced.

Figure 18: Creating an index file of historical tickers

```
CRSP_STK_STRUCT stk;        /* stock data structure */

int main (int argc, char *argv[])
{
  int stkcrspnum;
  int stkid;
  int perm,ret;
  int i,n=0;
  char prev[16],tick[16];
  char buf[128];
  FILE *lfp,*tfp;

  if(argc<=1) {
    fprintf(stderr,"usage: instcrsp [monthly | daily] ...\n");
    exit(0);
  }

  crsp_init_env();
  if(!strcmp(argv[1],"daily")) {
    stkid=10;
    stkcrspnum=crsp_stk_open(DB_CRSP_DSTK,10,&stk,STK_ALL,"r",1);
    if (stkcrspnum == CRSP_FAIL) db_err(stderr,err_msg);
    sprintf(buf,"%sdaily.list",DB_CRSP_DATA);
    lfp=fopen(buf,"w");
    tfp=NULL;
  }
  else {
    stkid=20;
    stkcrspnum=crsp_stk_open(DB_CRSP_MSTK,20,&stk,STK_ALL,"r",1);
    if (stkcrspnum == CRSP_FAIL) db_err(stderr,err_msg);
    sprintf(buf,"%smonthly.list",DB_CRSP_DATA);
    lfp=fopen(buf,"w");
    tfp=fopen(DB_CRSP_TICK_LIST,"w");
  }

  fprintf(lfp,
          "%5s,%-8.8s,%-8.8s,\"%-32.32s\", %-8.8s ,%8s,%8s,%8s,%8s,%2s,%2s, %4s ,%3s\n",
          "Permn","HdrCusip","OldCusip","Name","Ticker",
          "BegName","EndName","BegData","EndData","Sh","Ex","SIC","Rec");
  while((ret=crsp_stk_read(stkcrspnum,stkid,&perm,CRSP_NEXT,&stk,STK_ALL))!=
        CRSP_EOF) {

    for(i=0;i<stk.events.names_arr->num;i++) {
      fprintf(lfp,
              "%5d,%-8.8s,%-8.8s,\"%-32.32s\",\"%-8.8s\",%8d,%8d,%8d,%8d,%2d,%2d,\"%4d\",%3d\n",
              stk.header->permno,stk.header->hcusip,
              isalnum(stk.events.names[i].ncusip[0])?
              stk.events.names[i].ncusip:"00000000",stk.events.names[i].comnam,
              stk.events.names[i].ticker,stk.events.names[i].namedt,
              stk.events.names[i].nameenddt,
              stk.header->begdt,stk.header->enddt,
              stk.events.names[i].shrcd,
              stk.events.names[i].exchcd,stk.events.names[i].siccd,
              stk.events.names_arr->num-i);
```

```
      if(strcmp(argv[1],"daily")) {
        if(isalnum(stk.events.names[i].shrcls[0])&&
           strlen(stk.events.names[i].ticker)) {
          sprintf(tick,"%s.%s",stk.events.names[i].ticker,
                  stk.events.names[i].shrcls);
        }
        else sprintf(tick,"%s",stk.events.names[i].ticker);
        if(strlen(tick)&&strcmp(tick,prev)&&strcmp(tick,stk.header->htick)) {
          fprintf(tfp,"%-9.9s %8d %11d\n",
                  tick,stk.events.names[i].nameenddt,stk.header->permno);
          n++;
          strcpy(prev,tick);
        }
      }
    }
  }
  if(n>=MAXCRSPTICK) db_err(stderr,"MAXCRSPTICK exceeded (modify db_local.h and recompile library routines");
  exit(EXIT_SUCCESS);
}
```

Next, I provide a new random access routine that uses the ticker/PERMNO list previously created to match a specified historical ticker to the company's PERMNO. If multiple tickers exist, then it starts with the most recently valid ticker. I also generalize this routine to accept any sort of security identifer: depending on the type of identifier inferred (5-digit identifiers are assumed to be PERMNOs, 4-digit identifiers are SIC codes, 8-character identifiers are cusips, other identifiers are tickers), the routine in turn calls the specific random access routine provided by CRSP.

Figure 19: Generalized random access routine for CRSP

```
static int crsp_query_type(char *query)
{
  int i;
  if(strlen(query)==8) return(CRSP_CUSIP);
  i=atoi(query);
  if(i>=10000 && i<=99999) return(CRSP_PERMNO);
  if(i>=1000 && i<=9999) return(CRSP_SICCD);
  else return(CRSP_TICKER);
}

int crsp_stk_read_any(int stkcrspnum, int stkid, char *query, int keyflag,
                      CRSP_STK_STRUCT *stk, int stkwanted)
{
  int perm,ret,i,siccd;
  FILE *fp;
  static ntick=0;
  static DB_TICKER *ptr=NULL,dum;

  for(i=0;i<strlen(query);i++) query[i]=toupper(query[i]);
  switch(crsp_query_type(query)) {
  case CRSP_CUSIP:
    ret=crsp_stk_read_cus(stkcrspnum,stkid,query,keyflag,stk,stkwanted);
    if(ret!=CRSP_SUCCESS) ret=crsp_stk_read_hcus(stkcrspnum,stkid,query,
                                                 keyflag,stk,stkwanted);
```

```
  break;
case CRSP_PERMNO:
  perm=atoi(query);
  ret = crsp_stk_read(stkcrspnum,stkid,&perm,keyflag,stk,stkwanted);
  sprintf(query,"%05u",perm);
  break;
case CRSP_SICCD :
  siccd=atoi(query);
  ret = crsp_stk_read_siccd(stkcrspnum,stkid,&siccd,keyflag,stk,stkwanted);
  sprintf(query,"%04u",siccd);
  break;
default:
  if(!ntick) {
    if((fp=fopen(DB_CRSP_TICK_LIST,"r"))==NULL) {
      db_err(stderr,DB_CRSP_TICK_LIST);
    }
    while(fscanf(fp,"%s %d %d",tick[ntick].tick,&(tick[ntick].date),
                 &(tick[ntick].perm))==3)
      if(++ntick>MAXCRSPTICK)
        db_err(stderr,"MAXCRSPTICK exceeded (modify db_local.h and recompile)");
    fclose(fp);
    qsort(tick,ntick,sizeof(DB_TICKER),sorttick);
  }
  strcpy(dum.tick,query);
  if(keyflag==CRSP_NEXT) {
    if(ptr==NULL) {
      ptr=(DB_TICKER *)bsearch((const void *)(&dum),(const void *)tick,
                               ntick,sizeof(DB_TICKER),cmptick);
      for(;ptr&&ptr>tick;ptr--) if(strcmp((ptr-1)->tick,ptr->tick)) break;
    }
    else ptr++;
    if(ptr&&!strcmp(ptr->tick,query)) {
      ret=crsp_stk_read(stkcrspnum,stkid,&(ptr->perm),CRSP_EXACT,
                        stk,stkwanted);
    }
    else {
      ret = crsp_stk_read_ticker(stkcrspnum,stkid,query,CRSP_NEXT,
                                 stk,stkwanted);
      if(ret==CRSP_FAIL)
        ret = crsp_stk_read_ticker(stkcrspnum,stkid,query,CRSP_FORWARD,
                                   stk,stkwanted);
      ptr=NULL;
    }

  }
  else {
    ret = crsp_stk_read_ticker(stkcrspnum,stkid,query,keyflag,
                               stk,stkwanted);
    if(ret==CRSP_NOT_FOUND) {
      ptr=(DB_TICKER *)bsearch((const void *)(&dum),(const void *)tick,
                               ntick,sizeof(DB_TICKER),cmptick);
      for(;ptr&&ptr>tick;ptr--) if(strcmp((ptr-1)->tick,ptr->tick)) break;

      if(ptr) ret=crsp_stk_read(stkcrspnum,stkid,&(ptr->perm),CRSP_EXACT,
                                stk,stkwanted);
    }
    else ptr=NULL;
  }
```

```
    break;
  }
  if(ret==CRSP_SUCCESS) crsp_clean_stk(stk);
  return(ret);
}
```

5.2 Compustat Industrial

Data from Compustat are distributed in flat character files. The previous section described a new routines to read Compustat Industrial data sequentially by company. That routine can also read a company's data, given a identifier such as a ticker symbol or cusip, via random access if an index file is available. The following program generates such an index file, which associates company ticker symbols and cusip identifiers with the location of the data in the flat file.

Figure 20: Creating an index file of tickers and cusip identifiers for Compustat

```
void main(int argc,char *argv[])
{
  CST_DB *cstdb[]={NULL};
  int cstdesc[]={0};
  CSTANN *cst=NULL;
  int pos,n=0;
  char ifile[128],lfile[128];
  FILE *ifp,*lfp;

  if (argc<=1) db_err(stderr,"usage: instann [ annual | annbak | annway | can ]");

  sprintf(lfile,"%s%s",DB_CRSP_CST,argv[1]);
  cstdb[0]=cst_open(lfile,"");
  sprintf(lfile,"%s%s.list",DB_CRSP_CST,argv[1]);
  sprintf(ifile,"%s%s.index",DB_CRSP_CST,argv[1]);
  lfp=fopen(lfile,"w");
  ifp=fopen(ifile,"w");
  fprintf(lfp,"%-8s,\"%-28s\",%-10s,%-8s,%-8s,%-6s,%-4s,%s\n",
          "CUSIP","CONAME","\"SMBL\"","BEGYEAR","ENDYEAR","\"DNUM\"","ZLST","FILE");
  for(pos=0;
      cst_ann_read(cstdb,cstdesc,1,&cst,NULL);) {
    fprintf(lfp,
            "%-8s,\"%-28.28s\",\"%-8.8s\",%8d,%8d,\"%04d\",\"%02d\",%02d\n",
            cst->cusip,cst->hdr.coname,cst->hdr.smbl,
            cst->hdr.begyr,cst->hdr.endyr,cst->hdr.dnum,
            cst->hdr.zlist,cst->hdr.file);
    fprintf(ifp,"%-9.9s %11d\n",cst->cusip,pos);
    fprintf(ifp,"%-9.9s %11d\n",cst->hdr.smbl,pos);
    n+=2;
    pos=ftell(cstdb[0]->fp);
  }
  if(n>=MAXCSTINDEX) db_err(stderr,"MAXCSTINDEX exceeded (modify db_local.h and recompile library routines");
  fclose(ifp);
  fclose(lfp);
}
```

5.3 I/B/E/S

Details on reading I/B/E/S character data files via random access was described in the prior section. This enabled all data for a company to be simultaneously read in from the separate character files. The following program in turns saves the merged data into a single new large binary file, along with an associated index file. I shall use this binary file instead of the original character data files for all subsequent I/B/E/S access.

Figure 21: Creating a merged I/B/E/S binary file

```
static void writeibes(IBES *ib)
{
  int i,j;
  fprintf(idxfp,"%-8.8s %10d\n",ib->ibes,ftell(ibesfp));
  for(i=0;i<ib->n[IDFIL];i++) {
    for(j=i+1;j<ib->n[IDFIL]&&strcmp(ib->did[i].cusip,ib->did[j].cusip);j++);
    if(j>=ib->n[IDFIL])
      fprintf(idxfp,"%-8.8s %10d\n",ib->did[i].cusip,ftell(ibesfp));
  }
  fwrite(ib,sizeof(IBES),1,ibesfp);
  fwrite(ib->act,sizeof(ACT),ib->n[ACTFIL],ibesfp);
  fwrite(ib->dadj,sizeof(ADJ),ib->n[ADJFIL],ibesfp);
  fwrite(ib->sadj,sizeof(ADJ),ib->n[ADJHI],ibesfp);
  fwrite(ib->det,sizeof(DET),ib->n[DETFIL],ibesfp);
  fwrite(ib->did,sizeof(IDN),ib->n[IDFIL],ibesfp);
  fwrite(ib->sid,sizeof(IDN),ib->n[IDHI],ibesfp);
  fwrite(ib->stp,sizeof(STP),ib->n[STOPFIL],ibesfp);
  fwrite(ib->sum,sizeof(SUM),ib->n[SUMHI],ibesfp);
  fwrite(ib->bak,sizeof(BAK),ib->n[BAKHI],ibesfp);
}

main(int argc,char *argv[])
{
  int i,j,k;
  IBES *ib=NULL;
  IB_DB *ibd;
  FILE *dfp,*sfp;
  char buf[128];
  if(argc<2) db_err(stderr,"usage: instibes [ US | INTL ]");

  ibd=ib_init(toupper(*(argv[1])));
  dfp=openf(mergename,".detail","w");
  sfp=openf(mergename,".consensus","w");
  fprintf(dfp,"%-6.6s,%-8.8s,%-10.10s,%-18.18s,%-8s,%-8s,%-8s\n",
          "Symbol","Cusip","Ticker","Name","SIG","Date","BegDate");
  fprintf(sfp,"%-6.6s,%-8.8s,%-10.10s,%-18.18s,%-8s,%-6s,%-6s\n",
          "Symbol","Cusip","Ticker","Name","SIG","Date","BegDat");
  createibes(ibd);
  for(i=0;i<ibd->n;i++) {
    ib_get(ibd,&ib);
    writeibes(ib);
    for(j=0;j<ib->n[IDFIL];j++) {
```

```
      for(k=0;k<j;k++) {
        if(!strcmp(ib->did[k].cusip,ib->did[j].cusip)&&
           !strcmp(ib->did[k].ticker,ib->did[j].ticker)) break;
      }
      if(k==j){
        fprintf(dfp,"%-6.6s,%-8.8s,\"%-8.8s\",\"%-16.16s\",\"%06d\",%8d,%8d\n",
                ib->did[j].ibes,ib->did[j].cusip,
                ib->did[j].ticker,ib->did[j].name,ib->did[j].sig,
                ib->did[j].start,ib->did[0].start);
      }
    }
    for(j=0;j<ib->n[IDHI];j++) {
      for(k=0;k<j;k++) {
        if(!strcmp(ib->sid[k].cusip,ib->sid[j].cusip)&&
           !strcmp(ib->sid[k].ticker,ib->sid[j].ticker)) break;
      }
      if(k==j){
          fprintf(sfp,"%-6.6s,%-8.8s,\"%-8.8s\",\"%-16.16s\",\"%06d\",%6d,%6d\n",
                  ib->sid[j].ibes,ib->sid[j].cusip,
                  ib->sid[j].ticker,ib->sid[j].name,ib->sid[j].sig,
                  ib->sid[j].start,ib->sid[0].start);
      }
    }
  }
}
```

The function `ib_init` should be called at the start of each application program to reads in the index file associated with the merged I/B/E/S binary file.

Figure 22: Reading index file to initialise access to merged I/B/E/S binary file

```
IB_DB *ib_init(char intl)
{
  IB_DB *ibd;
  if(toupper(intl)=='I') {
    international=intl;
    initintl();
  }
  ibd=malloc(sizeof(IB_DB));

  if(ibd->fileptr[0]=fopen(mergename,"r")) {
    fread(ibd->filemax,sizeof(ibd->filemax),1,ibd->fileptr[0]);
    if((idxfp=fopen(indexname,"r"))==NULL) db_err(stderr,indexname);
    for(ibd->n=0;fscanf(idxfp,"%s %d",ibd->sec[ibd->n].ibes,&(ibd->sec[ibd->n].brec[0]))==2;
        ibd->n++)
      if(ibd->n>=MAXIBES) db_err(stderr,"MAXIBES exceeded (modify db_local.h and recompile)");
    qsort(ibd->sec,ibd->n,sizeof(IB_INDEX),cmpibes);
    if(ib_etag==NULL) ib_etag=(int *) calloc(ibd->filemax[DETFIL],sizeof(int));
    if(ib_btag==NULL) ib_btag=(int *) calloc(ibd->filemax[DETFIL],sizeof(int));
#ifdef VERBOSE
    fprintf(stderr,"%d securities read from %s\n",ibd->n,indexname);
#endif
    return(ibd);
  }
```

```
  else db_err(stderr,mergename);
}
```

Given a company's symbol or cusip identifier, the function `ib_read_any` reads in the company's merged I/B/E/S data from the binary file, via random access.

Figure 23: Reading a merged I/B/E/S binary file

```
int ib_get(IB_DB *ibd,IBES **ib)
{
  int i;
  if (!(*ib)) *ib=ib_new(ibd);
  (*ib)->ibes[0]=(*ib)->cusip[0]='\0';
  for(i=0;i<9;i++) (*ib)->n[i]=0;
  if(fread(&fakeibes,sizeof(IBES),1,ibd->fileptr[0])==1) {
    for(i=0;i<9;i++) (*ib)->n[i]=fakeibes.n[i];
    strcpy((*ib)->ibes,fakeibes.ibes);
    strcpy((*ib)->cusip,fakeibes.cusip);
    if(fread((*ib)->act,sizeof(ACT),(*ib)->n[ACTFIL],ibd->fileptr[0])!=(*ib)->n[ACTFIL] ||
       fread((*ib)->dadj,sizeof(ADJ),(*ib)->n[ADJFIL],ibd->fileptr[0])!=(*ib)->n[ADJFIL] ||
       fread((*ib)->sadj,sizeof(ADJ),(*ib)->n[ADJHI],ibd->fileptr[0])!=(*ib)->n[ADJHI] ||
       fread((*ib)->det,sizeof(DET),(*ib)->n[DETFIL],ibd->fileptr[0])!=(*ib)->n[DETFIL] ||
       fread((*ib)->did,sizeof(IDN),(*ib)->n[IDFIL],ibd->fileptr[0])!=(*ib)->n[IDFIL] ||
       fread((*ib)->sid,sizeof(IDN),(*ib)->n[IDHI],ibd->fileptr[0])!=(*ib)->n[IDHI] ||
       fread((*ib)->stp,sizeof(STP),(*ib)->n[STOPFIL],ibd->fileptr[0])!=(*ib)->n[STOPFIL] ||
       fread((*ib)->sum,sizeof(SUM),(*ib)->n[SUMHI],ibd->fileptr[0])!=(*ib)->n[SUMHI] ||
       fread((*ib)->bak,sizeof(BAK),(*ib)->n[BAKHI],ibd->fileptr[0])!=(*ib)->n[BAKHI]) return(0);
    return(1);
  }
  else return(0);
}
int ib_read_any(char *s,IBES **ib,IB_DB *ibd)
{
  IB_INDEX *ptr,dum;
  int i;
  if (!(*ib)) *ib=ib_new(ibd);
  (*ib)->ibes[0]=(*ib)->cusip[0]='\0';
  for(i=0;i<9;i++) (*ib)->n[i]=0;
  for(i=0;i<strlen(s);i++) dum.ibes[i]=toupper(s[i]);
  dum.ibes[i]='\0';
  ptr= (IB_INDEX *) bsearch(&dum,ibd->sec,ibd->n,sizeof(IB_INDEX),cmpibes);
  if(ptr) {
    ibd->curr=((int)ptr-(int)(ibd->sec))/sizeof(IB_INDEX);
    if(fseek(ibd->fileptr[0],ibd->sec[ibd->curr].brec[0],0)) return(0);
    return(ib_get(ibd,ib));
  }
  return(0);
}
```

---

6 Simultaneous Access

"Simultaneous access" refers to the capability to simultaneously read data, for a specified security, from more than one database at the same time. The random access routines developed above are the critical components that allow us to write programs to extract data from multiple databases. Essentially, we just combine random access programs that have been developed for individual databases. An additional component to make this work is to be able to match identifiers between different databases. Different databases may not use the same identifier to identify companies, often using proprietary codes (such as CRSP's PERMNO). Usually, the databases do provide other identifiers for each company, and, with a little work, it is possible to match many of the companies by common identifiers such as CUSIP (for U.S. companies) or SEDOL (for international companies). Unfortunately, matching securities across databases is an imperfect science. Because of the different ways that data vendors track companies which undergo name or capital changes, major accounting changes, and also the timeliness which such changes are reflected in their products, mechanical matching by, say, CUSIP, can leave many unmatched companies. These remaining companies require a more manual matching process, perhaps by tracing ticker symbols or companies' name changes and history, using textual search services such as Lexis/Nexis or Dow Jones News Retrieval.

6.1 Matching Companies

In this subsection, I describe the success rates from a mechanical algorithm to match CRSP and Compustate securities. There is not an exact match between CRSP and Compustat securities. Neither database is a subset of the other. The common identifier usually used to merge these two databases is the security's cusip code. However, cusips change over time and may be re-used; furthermore the two data vendors (University of Chicago and Standard and Poors) have different ways of dealing with companies that undergo any kind of transformation, e.g. mergers, acquisitions, buyouts etc. In Compustat, each company is identified by by its "current" 8-digit cusip, whereas CRSP provides current and historical 8-digit cusips. Chan, Jegadeesh and Lakonishok (1993) examine the impact of selection bias, such as those induced by mechanical problems of matching cusip identifiers, on the performance of value-based investment strategies.

```
[ TO BE COMPLETED ]
```

6.2 Application example: Combined fundamental and price data

This subsection presents a sample program that simultaneously accesses CRSP and Compustat Industrial Annual files to extract data similar to that used for the cross-sectional analysis of Fama and French (1992). It sequentially reads through the entire CRSP Monthly file by security. For each security, it also reads the CRSP Daily file to obtain the security's beta statistics (which CRSP calculates from each year's daily stock returns). It uses the security's historical cusips to locate and read annual Compustat data as well. In June of every year, this program prints out the security's book equity and total assets value from the prior calendar year, the beta computed from daily stock returns the prior year, as well monthly stock returns for the proceeding twelve months. This program can be easily modified to extract additional fundamental data items, or to extract data items from I/B/E/S using the random access routines described in the previous section. It could also be provided with a separate list of matched identifiers, which could improve on the accuracy of mechanical matching, based on historical CRSP cusips, used here.

Figure 24: Program to extract combined fundamental and price data

```
#include "db.h"

#define BEGYR 1983     /* calendar year to begin processing */
int data[2]={60,6};    /* Compustat Annual data items to print out */

CRSP_STK_STRUCT stk,dstk;
static char buf[128];

int main ()
{
  int i,j,k,l,found,beg,end;
  float sz,ysz,beta;

  int stkcrspnum,dstkcrspnum,perm=0,ret;
  CRSP_TIMESERIES *shr=NULL,*exch=NULL,*sic=NULL;

  CST_DB *cstdb[]={NULL,NULL,NULL};
  int cstdesc[]={CST_CUR,CST_BAK,CST_WAY};
  CSTANN *cst=NULL;

  FILE *ofp[40];
  int fd;

/*---------------------------------------------------------------------------
  1. Initialise CRSP environment and open Compustat files
---------------------------------------------------------------------------*/
  crsp_init_env();

  cstdb[0]=cst_open(ANNUAL,ANNUALINDEX);
  cstdb[1]=cst_open(ANNBAK,ANNBAKINDEX);
  cstdb[2]=cst_open(ANNWAY,ANNWAYINDEX);

/*---------------------------------------------------------------------------
  2a.  Open the desired stock and indices database
---------------------------------------------------------------------------*/

  stkcrspnum = crsp_stk_open(DB_CRSP_MSTK,20,&stk,STK_ALL,"r",1);
  if (stkcrspnum == CRSP_FAIL) db_err(stderr,err_msg);
  dstkcrspnum = crsp_stk_open(DB_CRSP_DSTK,10,&dstk,STK_ALL,"r",1);
  if (dstkcrspnum == CRSP_FAIL) db_err(stderr,err_msg);

  for(k=0;k<sizeof(data)/sizeof(int);k++) {
    fprintf(stderr,"%-11s %s (#%d)\n",cst_ann_title[data[k]].s,cst_ann_title[data[k]].l,data[k]);
  }

  for(i=BEGYR;i<BEGYR+10;i++) {
    sprintf(buf,"%d",i);
    if((ofp[i-BEGYR]=fopen(buf,"w"))==NULL) db_err(stderr,buf);
  }

/*---------------------------------------------------------------------------
  3. Loop over all securities
---------------------------------------------------------------------------*/
```

```
while(crsp_stk_read(stkcrspnum,20,&perm,CRSP_NEXT,&stk,STK_ALL)!=CRSP_EOF) {
  crsp_stk_siccd(&stk,&sic);
  crsp_stk_exchcd(&stk,&exch);
  crsp_stk_shrcd(&stk,&shr);


  /*---------------------------------------------------------------------
    3a. Using historical CRSP cusips, read Compustat Industrial Annual
  ---------------------------------------------------------------------*/
  for(found=0,i=stk.events.names_arr->num-1;i>=0;i--) {
    if(isalnum(stk.events.names[i].ncusip[0])&&
       cst_ann_read(cstdb,cstdesc,sizeof(cstdb)/sizeof(cstdb[0]),&cst,
                    stk.events.names[i].ncusip)) {
      found=1;
      break;
    }
  }

  /*---------------------------------------------------------------------
    3b. If found then loop through June of each year to print out data
  ---------------------------------------------------------------------*/
  if(found) {
    for(beg=(BEGYR*10000)+630;beg<(BEGYR*10000)+100630;beg+=10000) {
      j=crsp_cal_search(stk.prc_ts->cal,CAL_TYPE_DATE,&beg,CRSP_CAL_BACK,0);
      if (j-6>0&&stk.prc_ts->cal->caldt[j-6]>=stk.header->begdt&&
          stk.prc_ts->cal->caldt[j]<=stk.header->enddt) {
        sz=stk.prc[j]*
          ((float)crsp_shr_num(&stk,stk.prc_ts->cal->caldt[j],0,NULL));

        ysz=stk.prc[j-6]*
          ((float)crsp_shr_num(&stk,stk.prc_ts->cal->caldt[j-6],0,NULL));

        fd=(beg/10000)-BEGYR;

        if(stk.header->begdt<=stk.prc_ts->cal->caldt[j-6]&&
           stk.header->enddt>=stk.prc_ts->cal->caldt[j]&&
           sz>1.0&&ysz>1.0) {

          for(k=MAXCSTANN;k>=2;k--) {
            if(cst->y[k].ucode>0&&cst->y[k-1].ucode>0&&cst->y[k-2].ucode>0&&
               (cst->y[k].yr/10000)==(stk.prc_ts->cal->caldt[j-6]/10000)) {
              fprintf(ofp[fd],"%8d\t%5d\t%8s",
                      cst->y[k].yr,stk.header->permno,cst->cusip);

              fprintf(ofp[fd],"\t%4d",((int *)sic->arr)[j]);
              fprintf(ofp[fd],"\t%2d",((int *)exch->arr)[j]);
             fprintf(ofp[fd],"\t%2d",((int *)shr->arr)[j]);

              fprintf(ofp[fd],"\t%8.1f\t%8.1f",sz,ysz);
              for(l=0;l<12;l++) {
                fprintf(ofp[fd],"\t%10.6f",
                        stk.prc_ts->cal->caldt[j+l]<=stk.header->enddt&&
                        stk.prc_ts->cal->caldt[j+l]>=stk.header->begdt?
                        stk.ret[j+l]:-99.0);
              }
              for(l=0;l<sizeof(data)/sizeof(int);l++) {
                fprintf(ofp[fd],"\t%12.3f",ismiss(cst->y[k].data[data[l]])?
                        -99.0:cst->y[k].data[data[l]]);
```

```
                }

                /*-------------------------------------------------------------------
                  3c. Also, pull out beta from prior year's daily returns.
                  -------------------------------------------------------------------*/
                if(crsp_stk_read(dstkcrspnum,10,&(stk.header->permno),
                                 CRSP_EXACT,&dstk,STK_ALL)==CRSP_SUCCESS) {
                  beta=crsp_stk_stat(&dstk,5,stk.prc_ts->cal->caldt[j-6]);
                  if(beta==-99.0) beta=crsp_stk_stat(&dstk,7,stk.prc_ts->cal->caldt[j-6]);
                }
                else beta=-99.0;
                fprintf(ofp[fd],"\t%12g",beta);
                fprintf(ofp[fd],"\n");

                break;
              }
            }
          }
        }
      }
    }
  }
}
```

---

7 Sample Applications

7.1 Application example: Profitability of momentum strategies

Prior research, such as Jegadeesh and Titman (1993), found that a strategy of buying past three-through twelve-month winners and selling loosers earn gross profits. Hong, Lim, and Stein (1999) show that profitability of this strategy depends on company size. The program described in this subsection can be used to construct momentum-based portfolio returns, for all stocks and by company capitalization, using a methodology similar to Carhart (1997). Carhart formed a PR1YR momentum factor portfolio by sorting all NYSE/AMEX/Nasdaq stocks each month based on their stock returns from twelve months through two months prior – PAST(2,12) – and holding long stocks in the highest 30 percentile and short stocks in the lowest 30 percentile. This program can generalize this approach by varying the pre-ranking period or the number of months over which to hold the portfolio.

Figure 25: Constructing Momentum-based Portfolio Returns

```
#define PASTEND 2      /* ending month of pre-ranking period, prior to current month */
#define PASTBEG 12     /* begining month of pre-ranking period, prior to current */
#define NHOLD 1        /* number of months to hold portfolio */

#define SORT "sort -T /sastemp -o mom.txt mom.txt"
                       /* system command used to sort a large (60MB) file mom.txt */

CRSP_STK_STRUCT stk;
int stkcrspnum;
int nstock[1000];

struct stkstruct {
  float past,ret[NHOLD];
  int cap;
} s[16000];

int cmp(const void *a,const void *b)
{
  float x,y;
  x=((struct stkstruct *)a)->past;
  y=((struct stkstruct *)b)->past;
  if(x>y) return(1);
  if(x<y) return(-1);
  return(0);
}

float portret[1000][NHOLD];
float nport[1000][NHOLD];

float momret[11][1000];

int main (int argc, char *argv[])
{
  int beg,end,ibeg,iend,perm,portnum;
  float past,ret,nret;
  int i,j,k,n;
```

```
FILE *fp;

crsp_init_env();
stkcrspnum=crsp_stk_open(DB_CRSP_MSTK,20,&stk,STK_ALL,"r",1);
if (stkcrspnum == CRSP_FAIL) db_err(stderr,err_msg);

fp=fopen("mom.txt","w");

beg=19260101;
beg=crsp_cal_search(stk.prc_ts->cal,CAL_TYPE_DATE,&beg,CRSP_CAL_NEXT,0);
end=19971231;
end=crsp_cal_search(stk.prc_ts->cal,CAL_TYPE_DATE,&end,CRSP_CAL_BACK,0);

while(crsp_stk_read(stkcrspnum,20,&perm,CRSP_NEXT,&stk,STK_ALL)==
      CRSP_SUCCESS) {
  for(i=stk.ret_ts->beg+PASTBEG;i<=stk.ret_ts->end;i++) {
    for(j=i-PASTBEG;j<=i-PASTEND;j++) if (stk.ret[j]<-1.0) break;
    if(stk.ret[i]>=-1.0 && j>i-PASTEND) {
      nstock[i]++;
      portnum=crsp_stk_port(&stk,5,stk.ret_ts->cal->caldt[i]);
      fprintf(fp,"%4d %2d %g",
              i,portnum,
              crsp_compnd_ret(stk.ret,i-PASTBEG,i-PASTEND));
      for(j=0;j<NHOLD;j++) {
        fprintf(fp," %g",crsp_compnd_ret(stk.ret,i,i+j));
      }
      fprintf(fp,"\n");
    }
  }
}
fclose(fp);

system(SORT);

for(portnum=0;portnum<=10;portnum++) {

  for(i=beg;i<=end;i++) for(j=0;j<k;j++) portret[i][j]=nport[i][j]=0.0;
  fp=fopen("mom.txt","r");

  for(i=beg;i<=end;i++) {

    for(j=n=0;j<nstock[i];j++) {
      fscanf(fp,"%*d %d %g",&(s[n].cap),&(s[n].past));
      for(k=0;k<NHOLD;k++) fscanf(fp,"%g",&(s[n].ret[k]));
      if(portnum==0||s[n].cap==portnum) n++;
    }

    qsort(s,n,sizeof(s[0]),cmp);
    for(j=0;j<n/3;j++) {
      for(k=0;k<NHOLD;k++) {
        if(s[j].ret[k]>=-1.0) {
          portret[i][k]-=s[j].ret[k];
          nport[i][k]+=1.0;
        }
      }
    }
    for(j=(n*2)/3;j<n;j++) {
      for(k=0;k<NHOLD;k++) {
```

```
          if(s[j].ret[k]>=-1.0) {
            portret[i][k]+=s[j].ret[k];
            nport[i][k]+=1.0;
          }
        }
      }
      for(k=NHOLD-1;k>=0;k--) {
        if(nport[i][k]>1.0) {
          portret[i][k]/=nport[i][k];
          if(k<NHOLD-1&&nport[i][k+1]>0) {
            portret[i][k+1]+=1.0;
            portret[i][k+1]/=portret[i][k];
            portret[i][k+1]-=1.0;
          }
        }
      }
    }
    fclose(fp);

    for(i=beg+PASTBEG;i<=end;i++) {
      nret=ret=0.0;
      for(k=0;k<NHOLD;k++) {
        if(i-k>=beg&&nport[i-k][k]>=1.0) {
          ret+=portret[i-k][k];
          nret+=1.0;
        }
      }
      momret[portnum][i]=(nret>=1.0?ret/nret:-99.0);
    }
  }
  for(i=beg+PASTBEG;i<=end;i++) {
    printf("%8d",stk.prc_ts->cal->caldt[i]);
    for(j=0;j<=10;j++) {
      printf(" %12.6f",momret[j][i]);
    }
    printf("\n");
  }
}
```

7.2 Application example: Constructing industry portfolios

The sample application program presented next can be used to construct industry-based value-weighted portfolios from monthly stock returns. It uses a 25-industry classification, similar to Hong, Lim and Stein (1999), derived from companies' historical two-digit SIC codes available from CRSP's historical names records.

As an aside, note that SIC or Standard Industrial Classification codes will be replaced by the NAICS, which stands for the North American Industry Classification System. The Bureau of the Census is scheduled to begin releasing advance general statistics in the NAICS format in 1999. Other statistical agencies will follow in the year 2000. The NAICS classifies industries according to the type of production activities performed, rather than a mixture of production-based and market-based categories in the SIC. The NAICS groups the economy into 20 broad sectors, twice as many as the SIC system, in addition to being a more extensive classification system (from 4 digits to 6 digits). Compustat's data will be expanded in 1999 to incorporate NAICS codes (as well

as other changes to be Y2K compliant).

Figure 26: Constructing Industry Portfolio Returns

```
#include "db.h"          /* routines to set/get command line flags */

CRSP_TIMESERIES *siccd=NULL;
CRSP_STK_STRUCT stk;
float tgt[25][MAXCRSPDAT],wt[25][MAXCRSPDAT];
float cap;

int main (int argc, char *argv[])
{
  int stkcrspnum,perm=0;
  int sic,i,j,dt;

  crsp_init_env();
  stkcrspnum=crsp_stk_open(DB_CRSP_MSTK,20,&stk,STK_ALL,"r",1);
  if (stkcrspnum == CRSP_FAIL) db_err(stderr,err_msg);
  for(i=2;i<stk.prc_ts->cal->ndays;i++) {
    for(j=0;j<25;j++) {
      tgt[j][i]=wt[j][i]=0.0;
    }
  }
  while(crsp_stk_read(stkcrspnum,20,&perm,CRSP_NEXT,&stk,STK_ALL)!=CRSP_EOF) {
    if(crsp_stk_siccd(&stk,&siccd)==CRSP_FAIL) db_err(stderr,err_msg);
    for(i=2;i<stk.prc_ts->cal->ndays;i++) {
      if(stk.prc_ts->cal->caldt[i-1]>=stk.header->begdt&&
         stk.prc_ts->cal->caldt[i]<=stk.header->enddt&&
         stk.ret[i]>=-1.0&&stk.events.shares_arr->num) {
        cap=stk.prc[i-1]*((float)crsp_shr_num(&stk,stk.prc_ts->cal->caldt[i-1],0,NULL));
        sic=sic25(*crsp_arr_int(siccd->arr,i))-1;
        if(cap>0.0) {
          tgt[sic][i]+=(cap*stk.ret[i]);
          wt[sic][i]+=cap;
        }
      }
    }
  }
  for(i=2;i<stk.prc_ts->cal->ndays;i++) {
    fprintf(stdout,"%5d",stk.prc_ts->cal->caldt[i]);
    for(j=0;j<25;j++) {
      fprintf(stdout," %12.6g",wt[j][i]>0.0?tgt[j][i]/wt[j][i]:-99.0);
    }
    fprintf(stdout,"\n");
  }
}

int sic25(int sic)
{
  if(sic>100) sic/=100;
  if(sic<=7) return(3);
  if(sic>=8 && sic<=9) return(6);
  if (sic>=10 && sic<=14) return(1);
  if (sic>=15 && sic<=19) return(2);
  if (sic>=20 && sic<=21) return(3);
```

```
  if (sic>=22 && sic<=23) return(4);
  if (sic>=24 && sic<=25) return(5);
  if (sic==26) return(6);
  if (sic==27) return(7);
  if (sic==28) return(8);
  if (sic==29) return(9);
  if (sic>=30 && sic<=32) return(10);
  if (sic>=33 && sic<=34) return(11);
  if (sic==35) return(12);
  if (sic==36) return(13);
  if (sic==37) return(14);
  if (sic>=38 && sic<=39) return(15);
  if (sic>=40 && sic<=47) return(16);
  if (sic==48) return(17);
  if (sic==49) return(18);
  if (sic>=50 && sic<=52) return(19);
  if (sic==53) return(20);
  if (sic>=54 && sic<=59) return(21);
  if (sic>=60 && sic<=61) return(22);
  if (sic==62) return(23);
  if (sic>=63 && sic<=69) return(24);
  return(25);
}
```

7.3 Application example: The Bootstrap and CRSP Indices returns

The bootstrap is a powerful technique that can be used to do Monte Carlo simultations when you do not know enough about the underlying data generating process. Suppose your data set contains N independent and indentically distributed observations. The bootstrap method uses the actual data set to generate any number of synthetic data sets also with N data points, by simply drawing N data points *with replacement*. The basic idea is that the original data set is in most cases the best estimator of the underlying probability distribution.

An application of the bootstrap method could be to simulate several random realizations of single or multivariate paths from the distribution of CRSP Indices returns. Specifically, the program below uses the bootstrap to generate a specified number of "trials" by drawing monthly returns data points between two specified dates, for one or multiple CRSP Indices; for each trial, it simply returns the ending compounded value or values of the specified Indices.

Figure 27: The Bootstrap and CRSP Indices Returns

```
#include "db.h"
#include "crsp.h"
#include "crsp_init.h"

DB_FLAG db_f[]={
  "beg",0,NULL,
  "end",19991231,NULL,
  "before",0,NULL,
  "brief",0,NULL,
  "daily",0,NULL,
};
```

```
#define db_n (sizeof(db_f)/sizeof(DB_FLAG))

CRSP_IND_STRUCT ind;
float *ts[128];
float tsprc[128];
char tsname[128][16];

int main (int argc, char *argv[])
{
  int indcrspnum;
  int indno;

  char query[32];
  int ret;
  int beg,end,ibeg,iend,before,iseed,iboot;
  int i,j,k,nperm;
  double scaling;

  if(argc==1) {
    fprintf(stderr,"usage: indboot [daily | monthly] [YYYYMMDD] [YYYYMMDD] [#trials]...\n");
    fprintf(stderr,"  <identifier>\n\n");
    for(i=0;i<db_n;i++) {
      if(db_f[i].desc) fprintf(stderr,"  %-10s - %s\n",db_f[i].name,db_f[i].desc);
    }
    db_copyright(stderr);
    exit(0);
  }

  crsp_init_env();
  db_set_flag(db_f,db_n,&(argv[1]),argc-1);

  before=db_get_flag(db_f,db_n,"before");
  ibeg=db_get_flag(db_f,db_n,"beg");
  iend=db_get_flag(db_f,db_n,"end");

  for(nperm=0;fscanf(stdin,"%s",&query) != EOF;) {
    indno=atoi(query);
    if((ret=crsp_ind_load(db_get_flag(db_f,db_n,"daily")?460:420,indno,&ind,IND_ALL))==CRSP_FAIL)
      db_err(stderr,err_msg);
    if (ret != CRSP_SUCCESS) {
      if(!db_get_flag(db_f,db_n,"brief"))
fprintf (stdout, "[%s] not found\n",query);
    }
    else {
      if(!db_get_flag(db_f,db_n,"brief")) {
fprintf(stdout,"[%s] \"%-64.64s\"\n",query,ind.indhdr->indname);
      }
      beg=crsp_cal_search(ind.tind_ts[0]->cal,CAL_TYPE_DATE,&ibeg,CRSP_CAL_NEXT,0);
      end=crsp_cal_search(ind.tind_ts[0]->cal,CAL_TYPE_DATE,&iend,CRSP_CAL_BACK,0);

      if(ind.tret_ts[0]->beg) {
ts[nperm]=(float *) malloc(sizeof(float)*(end+1));
memcpy(ts[nperm],ind.tret[0],sizeof(float)*(end+1));
      }
      else ts[nperm]=NULL;
      sprintf(tsname[nperm++],"%07d",indno);
    }
  }
```

```
  if(!db_get_flag(db_f,db_n,"brief")) {
    fprintf(stdout,"\n");
    fprintf(stdout,"Trial");
    for(i=0;i<nperm;i++) fprintf(stdout," %12s",tsname[i]);
    fprintf(stdout,"\n");
  }

  scaling=((double)(end-beg+1))/pow(2.0,15.0);
  iseed=rand();

  for(k=0;k<before;k++) {
    for(i=0;i<nperm;i++) tsprc[i]=1.0;
    for(j=beg;j<=end;j++) {
      iboot=rand_r(&iseed);
      iboot=((int)floor(((double)iboot)*scaling))+beg;
      for(i=0;i<nperm;i++) {
if(ts[i][iboot]>=-1.0) tsprc[i]*=(1.0+ts[i][iboot]);
      }
    }
    fprintf(stdout,"%5d",k);
    for(i=0;i<nperm;i++) fprintf(stdout," %12g",tsprc[i]);
    fprintf(stdout,"\n");
  }
  exit(EXIT_SUCCESS);
}
```

---

8 Numerical Analysis

Since the PERC toolkit is written entirely in the C language, it is straighforward to incorporate standard numerical libraries, such as Lapack, IMSL or Numerical Recipes in C. Judd (1998) provides an excellent review of numerical methods for economists. Researchers can directly apply such algorithms to CRSP, Compustat and I/B/E/S data, by simultaneously using C-language direct data access and analysis routines in their application programs. This may be a preferred method for performing numerical or statistical analysis on a large set of data instead of relying on a separate statistical environment, which often does not run as efficiently as code written in a "low-level" programming language, such as C.

This section describes a simple implementation to incorporate the popular Numerical Recipes in C (NR) package of routines; see Press, Teukolsky, Vetterling and Flannery (1992). Along with various optimization and numerical algorithms, this package provides matrix manipulation tools. However, its matrix data structure is difficult to use – unlike normal C language conventions, array offsets do not begin at the index 0; the data structures do not keep track of matrix dimensions; and no automatic dynamic memory management routines are provided. In this section, I provide an enhanced data structure definition that is compatible with NR data variables and routines. I then define new dynamic memory allocation and memory management routines: these can track all previously allocated memory, which can then be easily released through a single function call when no longer needed. I then show how multiple regression coefficients can be computed, as an application example.

8.1 Data structures

These new MATRIX and VECTOR data structures incorporate size dimensions, and can be linked together in a list (to facilitate memory management tasks). The same data space can be referenced using NR's convention of beginning array indices at 1, or using the C language convention of 0.

Figure 28: Data structures for matrices and vectors

```
typedef struct matrixstruct {
  int nc,nr;
  float **rc;     /* accesses data via Numerical Recipes convention */
  float **m;      /* accesses data via conventional C offsets beginning at index 0 */
  struct matrixstruct *next;
} *MATRIX;

typedef struct vectorstruct {
  int n;
  float *rc;     /* accesses data via Numerical Recipes convention */
  float *v;      /* accesses data via conventional C offsets beginning at index 0 */
  float *v;
  struct vectorstruct *next;
} *VECTOR;
```

8.2 Allocating memory

New functions are defined for allocating memory and initializing data values for matrices and vectors. The actual space to hold the data in a MATRIX or VECTOR structure is allocated using the `matrix()` and `vector()` routines provided originally by NR. The data values can be initialised with user specified values. If desired, the newly allocated memory space is appended internally to a growing linked list.

Figure 29: Allocating space for matrices and vectors

```
MATRIX fin_mnew(int nr,int nc,float diag,float offdiag)
{
  MATRIX m;
  int i,j;
  m=(MATRIX) calloc(sizeof(struct matrixstruct),1);
  m->nc=nc;
  m->nr=nr;
  m->rc=matrix(1,m->nr,1,m->nc);
  m->m = (float **) calloc(sizeof(float *),nr);
  for(i=0;i<nr;i++) m->m[i]=m->rc[i+1]+1;
  m->next=NULL;
  if (offdiag!=0.0) {
    for(i=0;i<nr;i++) {
      for(j=0;j<nc;j++) {
        m->m[i][j]=offdiag;
      }
    }
  }
  if (diag!=0.0) for(i=0;i<nr;i++) m->m[i][i]=diag;
  if(memauto) {
    if(lastm) lastm->next=m;
    else firstm=m;
    lastm=m;
  }

  return(m);
}
VECTOR fin_vnew(int n,float val)
{
  VECTOR v;
  int i;
  v=(VECTOR) calloc(sizeof(struct matrixstruct),1);
  v->n=n;
  v->rc=vector(1,v->n);
  v->v=v->rc+1;
  v->next=NULL;
  if (val!=0.0) for(i=0;i<n;i++) v->v[i]=val;

  if(memauto) {
    if(lastv) lastv->next=v;
    else firstv=v;
    lastv=v;
  }
  return(v);
}
```

8.3 Managing memory

The function `fin_memsweep` simply marches down the linked list of previously allocated MATRIX and VECTOR data structures, and releases the memory space back to the system's pool. Dynamic memory management hence is hidden from the user; a single function call to `fin_memsweep` at the end of a processing section of code sufficies to recover memory space.

Figure 30: Sweeping unused memory space

```
void fin_memsweep()
{
  MATRIX nextm;
  VECTOR nextv;
  for(nextm=firstm?firstm->next:NULL;firstm;firstm=nextm) fin_mfree(&firstm);
  for(nextv=firstv?firstv->next:NULL;firstv;firstv=nextv) fin_vfree(&firstv);
  lastm=NULL;
  lastv=NULL;
}
```

8.4 Application: Multiple linear regression

The following function computes multiple linear regression coefficients, given the values of a dependent variable and multiple independent variables. The critical step is a call to the `gaussj()` function provided in NR, which implements linear equation solution by Gauss-Jordan elimination. In this example, memory is managed "manually"; I override the "automatic" tracking of allocated memory for new MATRIX and VECTOR data structures, and explicitly release memory space allocated for each data variable. This function can be used, for example, to return CAPM estimates by setting a stocks' returns as the dependent variable, and a column of ones and a market index's total returns as the independent variables.

Figure 31: Computing multiple regression coefficients

```
int fin_tsreg(float *lhs,float *cols[],int nc, int obs,float *b)
{
  MATRIX xx,xy,x,y,xt;
  int i,j,oldauto;

  if(obs>nc+1) {
    oldauto=fin_setmemauto(0);
    x=fin_mnew(obs,nc+1,0.0,0.0);
    for(i=0;i<obs;i++) {
      x->m[i][0]=1.0;
      for(j=1;j<=nc;j++) x->m[i][j]=cols[j-1][i];
```

```
    }

    y=fin_mnew(obs,1,0.0,0.0);
    for(i=0;i<obs;i++) {
      y->m[i][0]=lhs[i];
    }

    xt=fin_mtrans(x);
    xx=fin_mmult(xt,x);
    xy=fin_mmult(xt,y);

    gaussj(xx->rc,xx->nc,xy->rc,xy->nc);

    for(i=0;i<xy->nr;i++) b[i]=xy->m[i][0];
    fin_mfree(&x);
    fin_mfree(&y);
    fin_mfree(&xt);
    fin_mfree(&xx);
    fin_mfree(&xy);
    fin_setmemauto(oldauto);
    return(obs);
  }
  return(0);
}
```

---

9 Web-based Access

Web browsers and internet connectivity provide a very convenient way for users to access information. Through web forms, users simply specify their data requests, using the standard graphical point-and-click interface provided by web browsers; programs on the web server process the data requests and extract and return the data directly to the user's browser. The random access routines developed above can form the core programs that drive web access. We merely need to create short scripts that simply pass parameters specified by users on web forms (such as company identifiers, dates and data item codes) to the random access programs that are already available. For data that have been installed as character files or permanent SAS datasets, these can be extracted by web server scripts directly or by submitting dynamically-generated programs to SAS respectively.

This section briefly describes how to build a such web interface to the databases. A web interface can be much more than a cute toy: it can be an important first step for new empirical researchers to gain an understanding of the characteristics of the databases. Many advanced internet-based (possibly java-coded) applications also become possible. This section provides some basic instructions for building web pages, forms and server-side programs. Some actual examples are also described; a complete listing of web scripts used to access CRSP, Compustat, and I/B/E/S files is provided in the appendix. These scripts allow the user to select specific companies or indices, date ranges and data items, and have these data automatically extracted and returned to his or her web browser (and hence potentially to anywhere on the internet). Time series data can also be graphed and displayed on a (Java-enabled) browser. Other scripts assist the user to lookup companies, by partial names, different kinds of identifiers, industry, exchange/index membership or country. The web connectivity facility within Microsoft Excel 97 enables spreadsheets to directly query for data over the web. More sophisticated web-based applications can be developed building on the base system described here, allowing more functionality, analysis and interactivity.

9.1 Why a Web Interface

There has hitherto been a very steep learning curve for researchers interested in using databases such as CRSP, Compustat and I/B/E/S. One has to learn to use a programming language, such as Fortran or more recently C or SAS, and then figure out the data structures, filenames, sample programs and utility subroutines (if these are even available), and the unique conventions and terminology of each database.

A web interface allows users to specify common data requests using a familiar and graphical tool – their favorite web browser. They can point-and-click (or copy-and-paste from a word-processing, spreadsheet or other program) their choice of company identifiers, dates and items. All programming details, data formats and file structures are hidden from the user. Researchers can quickly download, view and manipulate data subsets that they are interested in. They can immediately begin useful work, and start learning about the characteristics of the underlying database, such as the kinds of data variables available; special codes denoting missing values, industry classification, or exchange membership; or the different kinds of company identifiers available, such as cusips, PERMNOs and ticker symbols. Web pages' "hypertext" approach (where users can choose to follow web page links to related topics they have an interest in) can also be utilized to provide "context-based" explanations of common questions (for example, what are special codes for missing data values, or why are prices negative [13]?), thus providing researchers with a gentle introduction

[13]Recall that in CRSP, if a stock did not trade in a particular day, it's closing price is reported as the average of the closing bid/ask quote, with a negative sign

to the idiosyncratic conventions of the databases.

Many powerful tools for web-programming are available at very low cost, or for free. Fully-functional web servers, the actual program that sits on your server machines and processes requests from web clients (usually browsers) elsewhere on the web, are available. Web page composers can help to create useful web pages and web forms that require little knowledge of the underlying HTML page description code. Furthermore, the scripting language commonly used for web server-side scripts, Perl, is, again, freely available and comes packaged with extremely convenient CGI routines (which are used to process the parameters passed by users' web forms) that greatly simply the writing of programs on the web server. In fact, it will be seen by the end of this section that the actual web forms and Perl scripts written to my web interface require very little developmental effort; most of the functionality is already available from the random access programs developed in the previous sections.

9.2 Setting Up a Basic Web Site

To get started, you will need to download and install a web server program, the latest version of the Perl language interpreter, and, optionally, a web page editor.

Web server Web browsers and servers communicate using a standard language, HTTP, which is platform independent. End-users can use any type of machine from anywhere that is connected to the internet. The web server program runs on your server machine and waits for HTTP requests, usually from web browsers. The web server checks that there are no access restrictions. Most web server programs allow basic authentication such as verifying a username and password from a text file. They should also allow the administrator to restrict different parts of the server from certain users or internet domain addresses. If there are no access retrictions, the server returns the data or document requester, after sending a header that describes the kind of file being returned. Finally, the client (browser) looks at the header and decides how to display the file. Documents that are in HyperText Markup Language ("HTML") can be displayed directly on the browser; other types of data, such as zipped data files or adobe acrobat documents ("PDF") have to be saved to disk or submitted to an external viewer.

Web servers support the Common Gateway Interface ("CGI"). This is simply the name of the method used to parse client requests, or to run external programs. CGI enables input from web-based forms to be passed to the server, and for actions on the server machine, such as accessing data files or executing specified programs, to be performed. This capability enables dynamically-generated web data, not just static predefined web pages, to be returned.

The first web server software was written to run on Unix, and from this, two freeware programs continue to be available today: NCSA HTTPd[14] and CERN HTTPd (also known as W3C HTTPd)[15]. The widely-used freeware Apache HTTPd[16] is a descendant from these early programs, as are commercial servers such as Netscape's suite[17], and Microsoft IIS, which comes packaged with Windows NT operating system. Although freeware web servers have excellent features with respect to security and efficient, product support can be lacking.

[14] `http://hoohoo.nsa.uiuc.edu/docs/Overview.html`
[15] `http://www.w3.org/Daemon`
[16] available at http://www.apache.org
[17] `http://www.netscape.com`

Support is available via usenet newgroups[18]. A convenient feature of Apache's offering is that it can be configured directly from its download website for a variety of (Unix) platforms.

The simplest way to protect access to the data is through the use of passwords or by restricting access only to specific domains (an *intranet*). There are also more sophisticated protocols that can be used, such as Secure Sockets Layer ("SSL") or Secure HTTP ("S-HTTP"). These secure protocols encrypt and authenticate data connections; but require the use of special (read, commercial) web servers. A useful discussions of these issues can be found on the *WWW Security FAQ* [19].

During installation, you can specify the DocumentRoot directory in which to place your web pages (named `htdocs` by default in Apache) and ScriptAlias directory in which to place CGI scripts (named `cgi-bin` by default in Apache). Apache also allows you to edit and place access control files (named `.htaccess` by default) in your document directories to deny or allow access to specific client addresses.

Perl command interpreter Perl, for "Practical Extraction and Report Language", is a powerful and concise scripting language. Although Perl interpreters have been ported to other platforms, such as Windows and Macintosh operating systems, Perl is primarily a Unix tool. It uses UNIX syntax and concepts, and superbly accomplishes many programming tasks under Unix. Comprehensive information on Perl is available from the Perl web site[20]. That excellent web site contains the latest version of Perl for downloading, reference and tutorial pages, and many links to Perl libraries and shared source code.

CGI processing steps are built into Perl, hence eliminating the need to understand the complexities of CGI processing. The CGI library in Perl version 5 provides a simple interface for parsing and interpreting query strings passed to CGI scripts. Everything is done through a CGI "object". When you create one of these objects it examines the environment for a query string, parses it, and stores the results. You can then ask the CGI object to return or modify the query values. The objects also automatically take care of many CGI tasks, such as handling POST and GET methods from web forms correctly and environment variables.

Web page editor HTML is the formatting language by which web pages (containing the information you wish to display on a client's web server) and web forms (on which clients can input requests) are constructed. HTML documents are essentially text documents that have special markup tags embedded in them. These tags are keywords surrounted by angled brackets <>, that tell the browser how to display text, graphics or links to other web pages. HTML also provides a set of data input features that allow users to input and submit information and requests. Web page editors, too many to list here, are now available that can guide newcomers, without requiring any direct HTML programming, in creating very flashy web pages. Notably, Netscape Composer and FrontPage Express web page composers now accompany their respective browser products, Netscape Communicator 4.0 and Microsoft Internet Explorer 4.0. FrontPage Express also has a primitive form-creation wizard that will guide you through specifying form input items and data validation script. Furthermore, many software programs[21] now enable native files to be saved as HTML documents; there are also many

[18] `news:comp.infosystems.www.servers.unix`

[19] `http://www.w3.org/Security/Faq`

[20] `http://www.Perl.com`

[21] In particular, SAS has made available web publishing tools that can create static HTML pages from SAS data sets and reports. These can be downloaded from `http://www.sas.com`

utilities that can convert word processing documents to HTML. However, since HTML is fairly simple to learn, directly writing web pages and forms in HTML provides more efficient control of their appearance and behaviour.

9.3 Basic HTML, Frames and JavaScript

The following is a listing of a sample "table of contents" page (`menu.html`) that could form the beginning of a web site. By using a few simple HTML tags, you can create very usable documents that can be displayed on any browser. These and other HTML tags allow you to link to other documents, embed images, change font appearance, and define text placement. They are used in the example to display a title and additional entries that select other web pages. The page uses the following simple HTML tags:

<**head**> separates the header from the main body.

<**title**> the title of the document that is displayed on the browser.

<**body**> the main body of the document.

<**center**> centers text when displayed in browser

<**font**> changes font size, color or face

<**href**> used to link to other documents

<**target**> tells the browser which frame or display window an action (such as a document link or form submission action) is to be applied to.

Javascript

Javascript is a scripting language that can be embedded into an HTML document to extend the functionality of static web pages. JavaScript statements can recognize and respond to user events such as mouse clicks, form input, and page navigation. As an example, you can write a JavaScript function to verify that users enter valid information into a form. An HTML page with embedded JavaScript can interpret the entered text and alert the user with a message dialog if the input is invalid, without needing to rely on programs on the web server. Or you can use JavaScript to perform an action (such as displaying a dialog box) in response to the user actions.

The following contents page examples uses the following JavaScript tags:

onMouseOver statement to initiate an action (in this case display a helpful message in the browser's status window) when the user's mouse arrow passes over a contents item.

onMouseOut statement to clear the message when the user moves the mouse arrow away.

<**script**> tag to define JavaScript *functions* containing several JavaScript command statements. The `printS` and `clearS` functions display a message in the window status bar at the bottom of the web browser display.

Figure 32: A "table of contents" web page

```
<html>
<head>
<title>PERC web access</title>
</head>
<body>
<script language="JavaScript">
<!-- hide from old browser
function printS(string) {
        window.status = string;
        return true;
}

function clearS() {
        window.status="";
        return true;
}
// -->
</SCRIPT>

<font size="2">
<center>
<strong>PERC web-access</strong>
</center>
<p>
<a target="_bottom_" href="crsp"
onMouseOver="printS('Monthly and Daily Stock Prices and Returns');return true"
onMouseout="clearS();return true;">CRSP Stocks</a>
<p>
<a target="_bottom_" href="indices" onMouseOver="printS('Monthly and Daily Index Returns');return true"
onMouseout="clearS();return true;">CRSP Indices</a>
<p>
<a target="_bottom_" href="bonds" onMouseOver="printS('Monthly Government Bonds and Interest Rates');return true"
onMouseout="clearS();return true;">CRSP Bonds</a>
<p>
<a target="_bottom_" href="compustat" onMouseOver="printS('Annual and Quarterly Fundamental Data');return true"
onMouseout="clearS();return true;">Compustat</a>
<p>
<a target="_bottom_" href="ibes" onMouseOver="printS('Earnings Estimates');return true"
onMouseout="clearS();return true;">I/B/E/S</a>
<p>
```

Frames

Frames are an enhancement to the basic HTML specification; they allow multiple HTML documents to be displayed in a single browser window. They can be used, for example, to allow a table of contents to remain in view while navigating a large document or several documents. As illustrated by the following HTML example, there are two basic Frame tags:

<frameset> This declares a new group of frames; they can be in columns or rows.

<frame> Specifies the document to be displayed in this frame, and the appearance of the frame, such as scrollbars, margin size and frameborders.

The web page example below tells the browser to have two frames side by side, with one frame occupying 150 pixels on the left of the browser window. It also specifies the two documents to be

displayed: a "contents" page `menu.html` in the first column, and an "license page `license.html` in the second. Also, note that the second frame has been assigned the name `_bottom_`. After a frame has been named, any HTML document can use the `target` tag to tell the browser which named frame an action is to be applied to.

Figure 33: HTML frames

```
html>
<head>
<title>PERC web access</title>
</head>
  <FRAMESET COLS="150, *" border=0 frameborder=0 framespacing=0>
    <FRAME SRC="menu.html" SCROLLING="auto" MARGINWIDTH=5 MARGINHEIGHT=5
     FRAMEBORDER="no" framespacing=0 border=0>
  <FRAME SRC="license.html" SCROLLING="auto" MARGINWIDTH=5 MARGINHEIGHT=5
   FRAMEBORDER="no" NAME="_bottom_" framespacing=0 border=0>
  </FRAMESET>
<NOFRAMES>
<b>
Web Access through frame-enabled web browsers only.
</NOFRAME>
```

9.4 HTML Forms and CGI

This section introduces HTML forms and CGI scripts, which enable the user and the web server to interact and exchange information. Note that the CGI programs in the PERC toolkit create and use temporary files on the web server; these files are created in the same directory directory that hold the CGI programs (i.e. the web server's ScriptAlias or `cgi-bin` directory), and have filenames with the prefix 0 (zero). At ths end of this subsection, I will describe a unix script that can be used to periodically "clean-up" these temporary files.

HTML forms

Forms are an important part of HTML. They emulate common features found on conventional paper forms. Some tags for designing forms include:

<**form**> Begins and ends a form. Also specifies an **action**, usually a script or program on the web server, to be executed when information from the form is submitted.

<**input**> Defines an input field such as a button, checkbox or text box. Input elements have a "name", and a "value" that can be selected or changed. Pairs of variables names and values that make up the form are submitted to the web server.

<**select**> Creates a scrolling list of options that the user can select.

<**option**> Specifies values for a list

<**textarea**> Creates a scrolling box in which users can enter free text.

A handy primer on HTML forms can be found on NCSA's web site[22].
As an example, the following form allows the user to

- Select from lists to specify the fiscal year end date. This uses the **select** tag with with each item choice (year and fiscal end) specified with the **option** tag.
- Enter a company identifer into a **text** box.
- Select a **checkbox** to specify whether to download consensus estimates.
- Select from several **radio** boxes to specify the data set and display format
- And, finally, click on a **submit** button to submit the information.

Figure 34: A Web Form to query I/B/E/S data

```
<body bgcolor="#FFFFFF" link="#0000FF" alink="#0000FF" vlink="#0000FF">
<table border="0" cellpadding="4" cellspacing="10">
  <tr>
    <td bgcolor="#C0C0C0">
      <font color="#00A000" size="4" face="arial">
        <b>I/B/E/S Detail</b>
      </font>
     </td>
  </tr>
  <tr>
    <td>
      <form action="/cgi-bin/detail.txt" method="post" target="_top">
      <table border="0" cellpadding="5">
        <tr>
          <td nowrap>
            <font size="2" face="arial">
              1. Fiscal Date
            </font>
          </td>
          <td>
            <table border="1" cellpadding="5" cellspacing="0">
              <tr>
                <td>
                  <font color="#00A000" size="1" face="arial">
                    - Choose Fiscal Year and Period:<br>
                  </font>
                  <font size="2" face="arial">
                    Year:
                    <select name="yr" size="1">
                      <option>1997 </option>
                      <option>1996 </option>
                      <option>1995 </option>
                      <option>1994 </option>
                      <option>1993 </option>
                      <option>1992 </option>
                      <option>1991 </option>
```

[22] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html

```
              <option>1990 </option>
              <option>1989 </option>
              <option>1988 </option>
              <option>1987 </option>
              <option>1986 </option>
             <option>1985 </option>
              <option>1984 </option>
              <option>1983 </option>
            </select>
            Fiscal Period
            <select name="fiscal" size="1">
              <option value="A">Annual</option>
              <option value="1">1Q (Jan-Mar)</option>
              <option value="2">2Q (Apr-Jun)</option>
              <option value="3">3Q (Jul-Sep)</option>
              <option value="4">4Q (Oct-Dec)</option>
          </font>
        </td>
      </tr>
    </table>
  </td>
</tr>
<tr>
  <td nowrap>
    <font size="2" face="arial">
      3. Company Identifier:
    </font>
  </td>
  <td>
    <table border="1" cellpadding="5" cellspacing="0">
      <tr>
        <td>
          <font color="#00A000" size="1" face="arial">
            - Enter a Cusip or I/B/E/S Symbol:<br>
          </font>
          <input type="text" size="10" name="query">
        </td>
      </tr>
    </table>
  </td>
</tr>
<tr>
  <td nowrap>
    <font size="2" face="arial">
      4. Other Items:
    </font>
  </td>
  <td>
    <table border="1" cellpadding="5" cellspacing="0">
      <tr>
        <td>
          <input type="checkbox" name="consensus" value="consensus" checked>
          <font size="2" face="arial">Consensus</font>
        </td>
      </tr>
    </table>
  </td>
</tr>
```

```
        <tr>
          <td nowrap>
            <font size="2" face="arial">
              5. US/International:<br>
            </font>
          </td>
          <td>
            <table border="1" cellpadding="5" cellspacing="0">
              <tr>
                <td>
                  <input type="radio" name="select" value="US" checked>
                  <font size="2" face="arial">US</font>
                  <input type="radio" name="select" value="INTL">
                  <font size="2" face="arial">International</font>
                </td>
              </tr>
           </table>
          </td>
        </tr>
        <tr>
          <td nowrap>
            <font size="2" face="arial">
              6. Output Format:<br>
            </font>
          </td>
          <td>
            <table border="1" cellpadding="5" cellspacing="0">
              <tr>
                <td>
                  <input type="radio" name="format" value="html" checked>
                  <font size="2" face="arial">HTML web page</font>
                  <input type="radio" name="format" value="text">
                  <font size="2" face="arial">Text</font>
                </td>
              </tr>
            </table>
          </td>
        </tr>
        <tr>
          <td>
            7. <input type="submit" name="submit" value="Go!">
          </td>
          <td>
            <font color="#00A000" size="1" face="arial">
              - Click on Go! button to submit request.
            </font>
          </td>
        </tr>
      </table>
      </form>
    </td>
  </tr>
</table>
```

CGI scripts and Perl

The form above enables the user to specify a request for data from I/B/E/S. HTML forms submit queries as pairs of variable names and values. Scripts on the web server respond to such requests, parse the input parameters, and access (or call other programs to access) data files on the server machine. Ths standard format by which queries are submitted by HTML forms and received by web server scripts is called CGI. The example below is the associated CGI script, written in Perl, to return I/B/E/S data. It parses the input from a HTML form, comprising values for the following parameters:

yr the calendar year of the fiscal end date.

fiscal the quarter of the year of the fiscal end date.

consensus whether to display consensus estimates.

query list of company identifiers (symbols or cusips).

Figure 35: Perl CGI script to access I/B/E/S data

```
use CGI;
$query = new CGI;
$|=1;

do 'db.pl';
db_auth($query,"IBES");
db_head($query);

$out=db_out($query);

open(FP,sprintf("| ./detail %s %s %s",
                $query->param('select'),$query->param('consensus'),$out));
$acusip=$query->param('query');
$acusip =~ s/^[\s,]+//;
@items=split(/[\s,]+/,$acusip);
printf(FP "%s %s %-1.1s\n",$items[0],$query->param('yr'),$query->param('fiscal'));
close(FP);

db_write($query->param('format'),$out,-99.1,-98.9);
```

Perl is a powerful procedural scripting language that supports programming elements such as:

control structures such as `if/else` statements, and `while` loops;

data types and operators Perl supports scalars (such as numbers and strings), arrays and associate arrays; and many operators and functions (e.g. `substr` which returns a substring). Scalar variable names are preceded by a `$` character, arrays by an `@` character and associative arrays with a `%` character.

file operators Perl can `print` or read from files (file handles are opened with the `open` statement) and standard input and output (predefined file handles `stdin` and `stdout` respectively).

pattern matching Some of the most powerful built-in functions are those related to pattern-mathing, using the Unix concept of regular expressions; examples include the `grep` command and the *match* and *substitute* functions.

CGI built-in CGI[23] subroutines and objects handle most of the tasks of processing information submitted by web forms. For example,

- `$query = new CGI` will parse input and store in an object named `query`
- `$query->param` will return the value of the named input parameter; or an array of values of all input parameters if called without any name arguments.

The appendix lists the suite of CGI scripts that make up all the interfaces to the data files described in this report. For character files, Perl scripts can be directly used to extract data. Some Perl scripts simply call and pass on parameters (such as security identifiers and dates) to pre-compiled C programs that had been written using random access routines described earlier to extract from binary data files. Other scripts call utility programs provided by the unix operating system. For example, a script can call the unix `fgrep` (fast grep) command to search a file for records that partially match a given string.

The following library of Perl routines automates many tasks for driving web-based data access.

db_auth can be modified to implement a means of authenticating users, such as by valiging username and/or password parameter values.

db_head prints a header command to the web client (browser) describing whether the data is returned as plain text or a HTML formatted document corresponding to the value of the CGI parameter *format*.

db_out returns a unique name for an intermediate file (or blank if output goes to the standard output) depending on the value of the CGI parameter *format*.

db_write converts an intermediate file to display format specified by the value of the CGI parameter *format*, which may be plain text, a HTML table or a zipped text file.

Figure 36: Shared library of Perl routines

```
sub db_auth {
    my($query,$names) = @_;
}

sub db_head {
    my($query) = @_;
    if($query->param('format') eq "html" || $query->param('format') eq "zip") {
        print $query->header('text/html');
    }
    else {
        print $query->header('text/plain');
    }
}
```

[23] See `http://www-genome.wi.mit.edu/ftp/pub/software/WWW/cgi_docs.html` for a guide to the Perl CGI library.

```
sub db_out {
    my($query) = @_;
    if($query->param('format') eq "html" || $query->param('format') eq "zip") {
        sprintf("> 0%s",$$);
    }
    else {
        "";
    }
}

sub db_write {
    my($param,$filenam,$lo,$hi) = @_;
    $filenam =~ s/\A[^\w]//;
    my($text1,$text2);
    if($param eq "html") {
        open(FP,$filenam);
        print "<p><table border=1 cellspacing=0 cellpadding=2>\n";
        while(<FP>) {
            @text1=split(/\"/,$_);
            for(@text2=(),$itext=0;$itext<=$#text1;$itext+=2) {
                $text1[$itext] =~ s/^[\s,]+//;
                $text1[$itext] =~ s/[\s,]+$//;
                push(@text2,split(/[\s,]+/,$text1[$itext]));
                if($#text1>$itext) {push(@text2,$text1[$itext+1])};
            }
            if ($#text2<0) {
                print "</table><p><table border=1 cellspacing=0 cellpadding=2>\n";
            }
            else {
                for($i=0;$i<=$#text2;$i++) {
                    if ($text2[$i]>$lo&&$text2[$i]<$hi) { $text2[$i]=""; }
                }
                print '<tr><td nowrap STYLE="vnd.ms-excel.numberformat:@">',
                join("<td align=right nowrap>",@text2),"</tr>\n";
            }
        }
        print "</table>\n";
        close(<FP>);
    }
    if($param eq "zip") {
        system(sprintf("mv 0%d 0%d.prn; /usr/local/bin/zip -q -j -l 0%d.zip 0%d.prn",$filenam,$filenam,$filenam,$fi
        printf("<p><h3>Data has been zipped to <a href=\"/cgi-src/0%d.zip\">0%d.zip</a> (right-click on filename to
    }
    if($param ne "html" && $param ne "zip") {
        print "\n****END-OF-DATA****\n";
    }
}
```

Temporary Web Server Files

Programs that deliver data via a web server usually require temporary space to store intermediate output files. The CGI programs described in this report create such intermediate files, with filenames prefixed with the number 0 (zero), in the same web server directory that holds the programs (the ScriptAlias directory, usually `cgi-bin`). Hence the web server program and the user identify

under which the web server is run (systems administrators sometimes run the web server under a special user name) must have write access to this directory. These temporary files can be removed periodically by running the following script, from your web server's ScriptAlias (`cgi-bin`) directory, as a background daemon.

Figure 37: Deleting temporary web server files

```
#! /bin/csh -f
echo "date" >! do.csh
while ( 1 != 2 )
/bin/rm do.csh

date >! 0.date

foreach file ( 0* )
  echo "/bin/rm " $file >> do.csh
end
sleep 900
csh do.csh
end
```

9.5 Using Java Applets

Java is a new and powerful programming language, developed by Sun Microsystems. It has many advantages over traditional languages, a complete discussion of which is beyond the scope of this report; browse to Sun's Java web site[24]. Many examples of Java programs, applets, and JavaBeans components can be found at repositories such as Gamelan[25] and the Java Financial Objects Xchange[26]. It is remarkable that in the short time since the birth of Java, developing useful Java-based applications has become so much more easier, through availability of GUI-based java programming environments, JavaBeans and class library components, and the generous sharing of re-usable Java classes and source code by programmers around the world.

Java *applets* are programs written specifically to be run over the web, regardless of the client's machine platform. Since Java is a true programming language, applets can perform a large variety of tasks, such as graphing data or other analysis. Compiled Java applets are stored on the web server machine; HTML pages then load these applets using the `applet` tag, passing parameters to customize the applet.

The following Perl CGI script `timing.txt` accesses CRSP's CTI Indices files (recall that these contain long time series of returns on a number of different asset classes). Instead of just returning numbers, it can instead display a *graph* of the data, using a Java applet[27]. The script actually (dynamically) generates and returns a HTML document to the web client (browser). The document contains an `applet` tag, specifying the name of the java applet to execute, and the parameters to pass to the applet. The data extracted is simply passed as dataset parameter values to the applet, which plots line graphs in the browser display.

[24] `http://www.sun.com/java`
[25] `http://www.gamelan.com`
[26] `http://www.jfox.com`
[27] The JavaChart applet used here was distributed by Visual Engineering `http://ve.com`

The script expects the following parameters to be passed from the web client:

asset1 the first asset class

asset2 the second asset class

byr, bmo beginning date

eyr, emo ending date

Three data series are plotted: the cumulative returns on the two specified assets, and the cumulative returns to a "perfect foresight market timing" strategy of switching investments at the beginning of each month to the asset class with the higher return in the month ahead[28].

Figure 38: Perl CGI script and Java applet to chart "perfect foresight" profits from CTI data

```
#!/usr/local/bin/Perl
use CGI;
$query = new CGI;
$|=1;

do 'db.pl';
db_auth($query);
print $query->header('text/html');

%list=('S&P500','1000500',
'Small Stocks','1000353',
'30-Year Bonds','1000700',
'20-Year Bonds','1000701',
'10-Year Bonds','1000702',
'7-Year Bonds','1000703',
'5-Year Bonds','1000704',
'2-Year Bonds','1000705',
'1-Year Bonds','1000706',
'90-Day Bills','1000707',
'30-Day Bills','1000708',
'CPI','1000709');

$asset1=$query->param('asset1');
$asset2=$query->param('asset2');
$beg=$query->param('byr') . $query->param('bmo');
$end=$query->param('eyr') . $query->param('emo');
open(FP,sprintf("echo %s %s | ./indices brief tret %s01 %s31 |",$list{$asset1},$list{$asset2},$beg,$end));

$n=0;
$y0[0]=1.0;
$y1[0]=1.0;
$y2[0]=1.0;

while(<FP>) {
    @datum=split(' ',$_);
    if($datum[2]>=-1.0&&$datum[3]>=-1.0) {
      $n=$n+1;
```

[28]Based on an example given in class by Professor Robert Merton

```
      if($datum[2]>$datum[3]) {
          $y0[$n]=(1.0+$datum[2])*$y0[$n-1];
      }
      else {
          $y0[$n]=(1.0+$datum[3])*$y0[$n-1];
      }
      $y1[$n]=(1.0+$datum[2])*$y1[$n-1];
      $y2[$n]=(1.0+$datum[3])*$y2[$n-1];
      $dt[$n]=$datum[0];
    }
}
close(FP);

open(FP,sprintf("> 0%s.out",$$));
for($i=1;$i<=$n;$i++) {
  $y=$dt[$i]/10000;
  $m=($dt[$i]/100)%100;
  $d=$dt[$i]%100;
  printf(FP "%d/%d/%04d, %.4f, %.4f, %.4f\n",
            $m,$d,$y,log($y0[$i]),log($y1[$i]),log($y2[$i]));
}
close(FP);

print '<HTML> <HEAD><TITLE>Date Charts</TITLE></HEAD> ' . "\n";
print '<BODY bgcolor="#ffffff" >' . "\n";
print '<center>' . "\n";
printf("<h3>Value of \$1 invested in %s or %s<br>",$asset1,$asset2);
print 'with perfect foresight each month</h3><p><hr>';

printf("<applet codebase=/javachart code=dateLineApp.class width=600 height=350>\n");
printf("<param name=customDatasetHandler value=\"../cgi-src/0%s.out\">\n",$$);
print '<param name=yAxisTitle value="Log $">',"\n";
print '<param name=legendOn value=true>',"\n";
print '<param name=legendHorizontal value=true>',"\n";
print '<param name=iconHeight value="0.02">',"\n";
print '<param name=dataset0Name value="Perfect Foresight">',"\n";
printf ("<param name=dataset1Name value=\"%s\">\n",$asset1);
printf ("<param name=dataset2Name value=\"%s\">\n",$asset2);
print '<param name=legendllX value=".0">',"\n";
print '<param name=legendllY value=".0">',"\n";
print '</applet><p>',"\n";

printf ("Final Value of \$1 (%d - %d):<table border=1 cellspacing=0 cellpadding=5>\n",$dt[1],$dt[$n]);
printf ("<tr><td>%s<td align=right>\$ %.1f\n",$asset1,$y1[$n]);
printf ("<tr><td>%s<td align=right>\$ %.1f\n",$asset2,$y2[$n]);
printf ("<tr><td>Perfect Foresight<td align=right>\$%.1f\n",$y0[$n]);
printf ("</table></center></html>\n");
```

9.6 Using Microsoft Excel Web Queries

Web queries allows you to query data from a specific web server and receive the information directly into Microsoft Excel. Hence Excel-based applications, incorporating spreadsheet formulas and analyses, can be easily developed that draw on data accessible from a web server as described earlier in this section.

Web queries are text files with the file extension `.iqy`, and contain three or four lines of text (depending on the type of Web page being queried), separated by carriage returns. Web queries are in the following syntax:

Type of Query (optional)

Version of Query (optional)

URL (required)

POST Parameters (required for queries referencing POST forms/data)

POST parameters can be static or dynamic or a combination: static parameters send query data without prompting the user; while dynamic parameters prompt the user for one or more values, which are used in the query.

Static queries include the parameter names and the values to be passed to the server. If there are multiple parameters, they are separated by an ampersand (&):

```
parameter1=value1&parameter2=value2
```

To create a dynamic parameter, replace the values with two arguments in braces. The first argument is the argument name, and the second argument is the text, enclosed in quotation marks, to be displayed in the Microsoft Excel-generated dialog box. Dynamic parameters are in the following format

```
parameter1=["value1","Prompt for first value"]&parameter2=["value2","Prompt for second value"]
```

Once you run a query in a worksheet and then save the worksheet, you no longer need the `.iqy` file for that worksheet. To run a web query in Microsoft Excel, point to `Get External Data` on the `Data` menu, click `Run Web Query`, select from the available web query files, and then click `Get Data`.

The example below is an Excel web query that, when run from Excel, asks the user to specify a list of companies, and beginning and ending years. It then submits a CGI query and directly receives monthly stock, market and T-Bill returns into the spreadsheet, which can be used for example in estimating the CAPM. Other applications could include spreadsheets for financial statement analysis, computing efficient portfolio, or estimating dividend-discount or other stock valuation models, all of which query for CRSP, Compustat and I/B/E/S company data served from a web server.

Figure 39: Excel web query to access CRSP stock and market returns

```
WEB
1
http://your.server.name/cgi-bin/mpt.txt
select=ret&query=["Companies","Companies:"]&byr=["Beg","Beginning year (YYYY):"]&eyr=["End","Ending year (YYYY):"]
```

---

10 Discussion

This report has provided a complete documentation of a unix-based platform for managing large financial research databases. It is hoped that this report can provide some guidance for setting up financial research databases and useful data access tools, if not form the basis of a standard implementation to support research and coursework. If a common and effective standard could be adopted, multiple sites could share their resources in implementing, supporting and continually enhancing such a system. Furthermore, a standard implementation of the underlying databases would facilitate sharing and testing of code for empiral research applications.

11 References

To be completed

Adamek, P., T. Lim, A. Lo and J. Wang, 1998, "Trading Volume and the MiniCRSP Databases: An Introduction and Users' Guide", manuscript.

Campbell, J., A. Lo and C. MacKinlay, 1997, *Econometrics of Financial Markets*, Princeton University Press.

Carhart, M., 1997, "Persistance in Mutual Fund Performance", *Journal of Finance.*

Chan, L., N. Jegadeesh and J. Lakonishok, 1995, "Evaluating the performance of value versus glamour stocks: The impact of selection bias", *Journal of Financial Economics* 38, 269-296.

Fama, E. and K. French, 1992, "The Cross-section of Expected Stock Returns" *Journal of Finance* 47, 427-465.

Guenther, D.A. and A. J. Rosman, 1994, "Differences between COMPUSTAT and CRSP SIC codes and related effects on research", *Journal of Accounting and Economics* 18, 1(July): 115-28.

Hong, H., T. Lim, and J. Stein, 1999, "Bad News Travels Slowly: Size, Analyst Coverage and the Profitability of Momentum of Strategies", *Journal of Finance*, forthcoming.

Judd, K., 1998, *Numerical Methods in Economics.*

Kahle, K., and R. A. Walkling, 1996, "The Impact of Industry Classifications On Financial Research", *Journal of Financial and Quantitative Analysis* 31, 3(September)

Keane, M., and D. Runkle, 1997, "Are Financial Analysts' Forecasts of Corporate Profits Rational", *Journal of Political Economy*, forthcoming

Lim, T., 1999a, "Rationality and Analysts' Forecast Bias", Tuck School working paper.

Lim, T., 1999b, "Using TAQ Transactions Data: A Research Application", manuscript.

Lo, A. and A.C. MacKinlay, 1990, "An Econometric Analysis of Nonsynchronous-Trading", *Journal of Econometrics* 45, 181-212.

Lo, A. and J. Wang, 1999, "Trading Volume: Definitions, data analysis, and implications of portfolio theory", MIT Sloan School working paper.

Philbrick, D., and W. Ricks, 1991, "Using Value Line and I/B/E/S Analyst Forecasts in Accounting Research", *Journal of Accounting Research*, 29, 397-417.

Press, W., S. Teukolsky, W, Vetterling, and B. Flannery, 1992, *Numerical Recipes in C: The Art of Scientific Computing. Second Edition*, Cambridge University Press.

Scholes, M., and J. Williams, 1977, "Estimating Betas from Asynchronous Data" *Journal of Financial Economics*, vol 5, 309-327.

A GNU GENERAL PUBLIC LICENSE Version 2, June 1991

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.) These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.) The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable. If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

B Utility Programs

This section lists the utility programs provided in the PERC toolkit, with source code. Generally, the programs are run with a number of command line options, as described below. The programs then wait for one or more lines of keyboard input, terminated by a CTRL-D (or input filtered in from a file) specifying identifiers one per line. Running the programs with no command line arguments displays the help message. These utility programs can be run as unix commands, and also to drive a web-based interface.

---

indices Returns daily, weekly or monthly time series of index levels and/or returns from CRSP Indices file.

```
usage: indices [YYYYMMDD] [YYYYMMDD] [monthly | daily] <items>...
  <identifier>
  ...

items:
  byweek     - Weekly
  bymonth    - Monthly
  byyear     - Yearly
  byqtr      - Quarterly
  tret       - Total return
  aret       - Capital Appreciation return
  iret       - Income return
  tind       - Total return Index
  aind       - Capital Appreciation Index
  iind       - Income return Index
```

---

crspfind Returns company information – by header or historical cusip, header or historical ticker, permno, or historical sic identifiers – from CRSP Monthly Stocks file

```
usage: crspfind
  <identifier> [optional YYYYMMDD]
   ...
```

---

manym Returns time series of individual stock data from CRSP Stocks Monthly File.

```
usage: manym <items>...
  <identifier> <begYYYYMMDD> <endYYYYMMDD>
  ...

items:
  sp500      - SP500 total returns
  return     - total stock return
  price      - price (or negative of bid/ask average
  lo         - lowest closing price in month
  hi         - highest closing price in month
  facpr      - factor to adjust prices
```

```
  adjpr      - adjusted price (end of period)
  bid        - Nasdaq closing bid
  ask        - Nasdaq closing ask
  divamt     - cash dividends
  shrout     - shares outstanding
  cap        - market capitalization
  volume     - share volume
  turnover   - turnover (volume/shares outstanding)
  ticker     - historical ticker symbol (.share class, if any)
  exchcd     - exchange listed
  siccd      - SIC code
  xsp500     - excess return relative to SP500
  xvwretd    - excess return relative to Value-weighted Market
  xdecnaq    - excess return relative to NYSE/AMEX/Nasdaq size decile
  xdecna     - excess return relative to NYSE/AMEX size decile
  xdecq      - excess return relative to Nasdaq size decile
  xdecn      - excess return relative to NYSE size decile
  xdeca      - excess return relative to AMEX size decile
  xcapnaq    - excess return relative to Cap-based NYSE/AMEX/Nasdaq decile
  xcapn      - excess return relative to Cap-based NYSE decile
  xcapna     - excess return relative to Cap-based NYSE/AMEX decile
  pdecnaq    - NYSE/AMEX/Nasdaq size decile portfolio assignment
  pdecna     - NYSE/AMEX size decile portfolio assignment
  pdecq      - Nasdaq size decile portfolio assignment
  pdecn      - NYSE size decile portfolio assignment
  pdeca      - AMEX size decile portfolio assignment
  pcapnaq    - NYSE/AMEX/Nasdaq Cap-based decile portfolio assignment
  pcapn      - NYSE Cap-based decile portfolio assignment
  pcapna     - NYSE/AMEX Cap-based decile portfolio assignment
```

---

manyd Returns daily or weekly time series of individual stock data from CRSP Stocks Daily File.

```
usage: manyd <items>...
  <identifier> <begYYYYMMDD> <endYYYYMMDD>
  ...

items:
  byweek     - Weekly
  bymonth    - Monthly
  byyear     - Yearly
  byqtr      - Quarterly
  sp500      - SP500 total returns
  return     - total stock return
  price      - price (or negative of bid/ask average
  facpr      - factor to adjust prices
  adjpr      - adjusted price (end of period)
  bid        - Nasdaq closing bid
  ask        - Nasdaq closing ask
  divamt     - cash dividends
  shrout     - shares outstanding
  cap        - market capitalization
  volume     - share volume
  turnover   - turnover (volume/shares outstanding)
  numtrd     - Nasdaq number of trades
  ticker     - historical ticker symbol (.share class, if any)
```

```
  exchcd     - exchange listed
  siccd      - SIC code
  xsp500     - excess return relative to SP500
  xvwretd    - excess return relative to Value-weighted Market
  xdecnaq    - excess return relative to NYSE/AMEX/Nasdaq size decile
  xdecna     - excess return relative to NYSE/AMEX size decile
  xdecq      - excess return relative to Nasdaq size decile
  xdecn      - excess return relative to NYSE size decile
  xdeca      - excess return relative to AMEX size decile
  xbetana    - excess return relative to NYSE/AMEX beta decile
  xstdna     - excess return relative to NYSE/AMEX standard deviation decile
  xbetaq     - excess return relative to Nasdaq beta decile
  xstdq      - excess return relative to Nasdaq standard deviation decile
  pdecnaq    - NYSE/AMEX/Nasdaq size decile portfolio assignment
  pdecna     - NYSE/AMEX size decile portfolio assignment
  pdecq      - Nasdaq size decile portfolio assignment
  pdecn      - NYSE size decile portfolio assignment
  pdeca      - AMEX size decile portfolio assignment
  pbetana    - NYSE/AMEX beta decile portfolio assignment
  pstdna     - NYSE/AMEX standard deviation decile portfolio assignment
  pbetaq     - Nasdaq beta decile portfolio assignment
  pstdq      - Nasdaq standard deviation decile portfolio assignment
```

---

portm Returns time series of individual stock data from CRSP Stocks Monthly File, for multiple stocks between common dates. Also (optionally) computes equal-weighted or value-weighted portfolio averages.

```
usage: portm <begYYYYMMDD> <endYYYYMMDD> <items>...
  <identifier>
  ...

items:
  nostock    - Do not show individual stock data
  noport     - Do not show portfolio data
  return     - total stock return
  price      - price (or negative of bid/ask average
  lo         - lowest closing price in month
  hi         - highest closing price in month
  adjpr      - Adjusted prices
  facpr      - Factor to adjust prices
  bid        - Nasdaq closing bid
  ask        - Nasdaq closing ask
  divamt     - cash dividends
  shrout     - shares outstanding
  cap        - market capitalization
  volume     - share volume
  turnover   - turnover (volume/shares outstanding)
  ticker     - historical ticker symbol (.share class, if any)
  exchcd     - exchange listed
  xsp500     - excess return relative to SP500
  xvwretd    - excess return relative to Value-weighted Market
  xdecnaq    - excess return relative to NYSE/AMEX/Nasdaq size decile
  xdecna     - excess return relative to NYSE/AMEX size decile
  xdecq      - excess return relative to Nasdaq size decile
  xdecn      - excess return relative to NYSE size decile
```

```
  xdeca      - excess return relative to AMEX size decile
  xcapnaq    - excess return relative to Cap-based NYSE/AMEX/Nasdaq decile
  xcapn      - excess return relative to Cap-based NYSE decile
  xcapna     - excess return relative to Cap-based NYSE/AMEX decile
  pdecnaq    - NYSE/AMEX/Nasdaq size decile portfolio assignment
  pdecna     - NYSE/AMEX size decile portfolio assignment
  pdecq      - Nasdaq size decile portfolio assignment
  pdecn      - NYSE size decile portfolio assignment
  pdeca      - AMEX size decile portfolio assignment
  pcapnaq    - NYSE/AMEX/Nasdaq Cap-based decile portfolio assignment
  pcapn      - NYSE Cap-based decile portfolio assignment
  pcapna     - NYSE/AMEX Cap-based decile portfolio assignment
  ereturn    - Equal-wtd return
  exdecnaq   - Equal-wtd excess return relative to NYSE/AMEX/Nasdaq size decile
  exdecna    - Equal-wtd excess return relative to NYSE/AMEX size decile
  exdecq     - Equal-wtd excess return relative to Nasdaq size decile
  exdecn     - Equal-wtd excess return relative to NYSE size decile
  exdeca     - Equal-wtd excess return relative to AMEX size decile
  excapnaq   - Equal-wtd excess return relative to NYSE/AMEX/Nasdaq Cap-based decile
  excapn     - Equal-wtd excess return relative to NYSE Cap-based decile
  excapna    - Equal-wtd excess return relative to NYSE/AMEX Cap-based decile
  exsp500    - Equal-wtd excess return relative to SP500
  exvwretd   - Equal-wtd excess return relative to Value-wtd Market
  vreturn    - Value-wtd return
  vxdecnaq   - Value-wtd excess return relative to NYSE/AMEX/Nasdaq size decile
  vxdecna    - Value-wtd excess return relative to NYSE/AMEX size decile
  vxdecq     - Value-wtd excess return relative to Nasdaq size decile
  vxdecn     - Value-wtd excess return relative to NYSE size decile
  vxdeca     - Value-wtd excess return relative to AMEX size decile
  vxcapnaq   - Value-wtd excess return relative to NYSE/AMEX/Nasdaq cap-based decile
  vxcapn     - Value-wtd excess return relative to NYSE Cap-based decile
  vxcapna    - Value-wtd excess return relative to NYSE/AMEX Cap-based decile
  vxsp500    - Value-wtd excess return relative to SP500
  vxvwretd   - Value-wtd excess return relative to Value-wtd Market
```

---

portd Returns daily or weekly time series of individual stock data from CRSP Stocks Daily File, for multiple stocks between common dates. Also (optionally) computes equal-weighted or value-weighted portfolio averages.

```
usage: portd <begYYYYMMDD> <endYYYYMMDD> <items>...
  <identifier>
  ...

items:
  nostock    - Do not show individual stock data
  noport     - Do not show portfolio data
  byweek     - Weekly
  bymonth    - Monthly
  byyear     - Yearly
  byqtr      - Quarterly
  return     - total stock return
  price      - price (or negative of bid/ask average
  adjpr      - Adjusted prices
  facpr      - Factor to adjust prices
  bid        - Nasdaq closing bid
```

```
ask        - Nasdaq closing ask
divamt     - cash dividends
shrout     - shares outstanding
cap        - market capitalization
volume     - share volume
turnover   - turnover (volume/shares outstanding)
numtrd     - Nasdaq number of trades
ticker     - historical ticker symbol (.share class, if any)
exchcd     - exchange listed
xsp500     - excess return relative to SP500
xvwretd    - excess return relative to Value-weighted Market
xdecnaq    - excess return relative to NYSE/AMEX/Nasdaq size decile
xdecna     - excess return relative to NYSE/AMEX size decile
xdecq      - excess return relative to Nasdaq size decile
xdecn      - excess return relative to NYSE size decile
xdeca      - excess return relative to AMEX size decile
xbetana    - excess return relative to NYSE/AMEX beta decile
xstdna     - excess return relative to NYSE/AMEX standard deviation decile
xbetaq     - excess return relative to Nasdaq beta decile
xstdq      - excess return relative to Nasdaq standard deviation decile
pdecnaq    - NYSE/AMEX/Nasdaq size decile portfolio assignment
pdecna     - NYSE/AMEX size decile portfolio assignment
pdecq      - Nasdaq size decile portfolio assignment
pdecn      - NYSE size decile portfolio assignment
pdeca      - AMEX size decile portfolio assignment
pbetana    - NYSE/AMEX beta decile portfolio assignment
pstdna     - NYSE/AMEX standard deviation decile portfolio assignment
pbetaq     - Nasdaq beta decile portfolio assignment
pstdq      - Nasdaq standard deviation decile portfolio assignment
ereturn    - Equal-wtd return
exdecnaq   - Equal-wtd excess return relative to NYSE/AMEX/Nasdaq size decile
exdecna    - Equal-wtd excess return relative to NYSE/AMEX size decile
exdecq     - Equal-wtd excess return relative to Nasdaq size decile
exdecn     - Equal-wtd excess return relative to NYSE size decile
exdeca     - Equal-wtd excess return relative to AMEX size decile
exbetana   - Equal-wtd excess return relative to NYSE/AMEX beta decile
exbetaq    - Equal-wtd excess return relative to Nasdaq beta decile
exstdna    - Equal-wtd excess return relative to NYSE/AMEX std dev decile
exstdq     - Equal-wtd excess return relative to Nasdaq std dev decile
exsp500    - Equal-wtd excess return relative to SP500
exvwretd   - Equal-wtd excess return relative to Value-wtd Market
vreturn    - Value-wtd return
vxdecnaq   - Value-wtd excess return relative to NYSE/AMEX/Nasdaq size decile
vxdecna    - Value-wtd excess return relative to NYSE/AMEX size decile
vxdecq     - Value-wtd excess return relative to Nasdaq size decile
vxdecn     - Value-wtd excess return relative to NYSE size decile
vxdeca     - Value-wtd excess return relative to AMEX size decile
vxbetana   - Value-wtd excess return relative to NYSE/AMEX beta decile
vxbetaq    - Value-wtd excess return relative to Nasdaq beta decile
vxstdna    - Value-wtd excess return relative to NYSE/AMEX std dev decile
vxstdq     - Value-wtd excess return relative to Nasdaq std dev decile
vxsp500    - Value-wtd excess return relative to SP500
vxvwretd   - Value-wtd excess return relative to Value-wtd Market
```

---

eventm Returns time series of individual stock data from CRSP Stocks Monthly File, for multiple stocks around an event date. Also (optionally) computes equal-weighted portfolio averages.

```
usage: eventm <begwindow> <endwindow> <items>...
  <identifier> <date>
  ...

items:
  nostock    - Do not show individual stock data
  noport     - Do not show portfolio data
  return     - total stock return
  price      - price (or negative of bid/ask average
  lo         - lowest closing price in month
  hi         - highest closing price in month
  adjpr      - Adjusted prices
  facpr      - Factor to adjust prices
  bid        - Nasdaq closing bid
  ask        - Nasdaq closing ask
  divamt     - cash dividends
  shrout     - shares outstanding
  cap        - market capitalization
  volume     - share volume
  turnover   - turnover (volume/shares outstanding)
  ticker     - historical ticker symbol (.share class, if any)
  exchcd     - exchange listed
  xsp500     - excess return relative to SP500
  xvwretd    - excess return relative to Value-weighted Market
  xdecnaq    - excess return relative to NYSE/AMEX/Nasdaq size decile
  xdecna     - excess return relative to NYSE/AMEX size decile
  xdecq      - excess return relative to Nasdaq size decile
  xdecn      - excess return relative to NYSE size decile
  xdeca      - excess return relative to AMEX size decile
  xcapnaq    - excess return relative to Cap-based NYSE/AMEX/Nasdaq decile
  xcapn      - excess return relative to Cap-based NYSE decile
  xcapna     - excess return relative to Cap-based NYSE/AMEX decile
  pdecnaq    - NYSE/AMEX/Nasdaq size decile portfolio assignment
  pdecna     - NYSE/AMEX size decile portfolio assignment
  pdecq      - Nasdaq size decile portfolio assignment
  pdecn      - NYSE size decile portfolio assignment
  pdeca      - AMEX size decile portfolio assignment
  pcapnaq    - NYSE/AMEX/Nasdaq Cap-based decile portfolio assignment
  pcapn      - NYSE Cap-based decile portfolio assignment
  pcapna     - NYSE/AMEX Cap-based decile portfolio assignment
  ereturn    - Equal-wtd return
  exdecnaq   - Equal-wtd excess return relative to NYSE/AMEX/Nasdaq size decile
  exdecna    - Equal-wtd excess return relative to NYSE/AMEX size decile
  exdecq     - Equal-wtd excess return relative to Nasdaq size decile
  exdecn     - Equal-wtd excess return relative to NYSE size decile
  exdeca     - Equal-wtd excess return relative to AMEX size decile
  excapnaq   - Equal-wtd excess return relative to NYSE/AMEX/Nasdaq Cap-based decile
  excapn     - Equal-wtd excess return relative to NYSE Cap-based decile
  excapna    - Equal-wtd excess return relative to NYSE/AMEX Cap-based decile
  exsp500    - Equal-wtd excess return relative to SP500
  exvwretd   - Equal-wtd excess return relative to Value-wtd Market
```

---

eventd Returns daily or weekly time series of individual stock data from CRSP Stocks Daily File, for multiple stocks around an event date. Also (optionally) computes equal-weighted portfolio averages.

```
usage: eventd <begwindow> <endwindow> <items>...
  <identifier> <date>
  ...

items:
  nostock    - Do not show individual stock data
  noport     - Do not show portfolio data
  byweek     - Weekly
  bymonth    - Monthly
  byyear     - Yearly
  byqtr      - Quarterly
  return     - total stock return
  price      - price (or negative of bid/ask average
  adjpr      - Adjusted prices
  facpr      - Factor to adjust prices
  bid        - Nasdaq closing bid
  ask        - Nasdaq closing ask
  divamt     - cash dividends
  shrout     - shares outstanding
  cap        - market capitalization
  volume     - share volume
  turnover   - turnover (volume/shares outstanding)
  numtrd     - Nasdaq number of trades
  ticker     - historical ticker symbol (.share class, if any)
  exchcd     - exchange listed
  xsp500     - excess return relative to SP500
  xvwretd    - excess return relative to Value-weighted Market
  xdecnaq    - excess return relative to NYSE/AMEX/Nasdaq size decile
  xdecna     - excess return relative to NYSE/AMEX size decile
  xdecq      - excess return relative to Nasdaq size decile
  xdecn      - excess return relative to NYSE size decile
  xdeca      - excess return relative to AMEX size decile
  xbetana    - excess return relative to NYSE/AMEX beta decile
  xstdna     - excess return relative to NYSE/AMEX standard deviation decile
  xbetaq     - excess return relative to Nasdaq beta decile
  xstdq      - excess return relative to Nasdaq standard deviation decile
  pdecnaq    - NYSE/AMEX/Nasdaq size decile portfolio assignment
  pdecna     - NYSE/AMEX size decile portfolio assignment
  pdecq      - Nasdaq size decile portfolio assignment
  pdecn      - NYSE size decile portfolio assignment
  pdeca      - AMEX size decile portfolio assignment
  pbetana    - NYSE/AMEX beta decile portfolio assignment
  pstdna     - NYSE/AMEX standard deviation decile portfolio assignment
  pbetaq     - Nasdaq beta decile portfolio assignment
  pstdq      - Nasdaq standard deviation decile portfolio assignment
  ereturn    - Equal-wtd return
  exdecnaq   - Equal-wtd excess return relative to NYSE/AMEX/Nasdaq size decile
  exdecna    - Equal-wtd excess return relative to NYSE/AMEX size decile
  exdecq     - Equal-wtd excess return relative to Nasdaq size decile
  exdecn     - Equal-wtd excess return relative to NYSE size decile
  exdeca     - Equal-wtd excess return relative to AMEX size decile
  exbetana   - Equal-wtd excess return relative to NYSE/AMEX beta decile
  exstdna    - Equal-wtd excess return relative to NYSE/AMEX std dev decile
  exbetaq    - Equal-wtd excess return relative to Nasdaq beta decile
  exstdq     - Equal-wtd excess return relative to Nasdaq std dev decile
  exsp500    - Equal-wtd excess return relative to SP500
  exvwretd   - Equal-wtd excess return relative to Value-wtd Market
```

---

riskd Returns annual risk statistics for multiple stocks, from CRSP Stocks Daily file.

```
usage: riskd [brief] <begYYYY> <endYYYY> <items>...
  <identifier>
  ...

items:
  beta       - Beta, from prior year daily returns
  std        - Standard Deviation, from prior year daily returns
  cap        - Market Cap, from end of prior year
```

---

info Returns descriptive information for multiple stocks, from CRSP Stocks Daily or Monthly files.

```
usage: info [daily | monthly] YYYMMDD YYYYMMDD [begcode] [endcode] item
  <identifier>
  ...

items:
  brief      - Do not print error messages
  header     - Display header information
  names      - Display historical names information
  dist       - Display distributions information
  shares     - Display historical shares outstanding records
  delist     - Display delisting information
```

---

mpt Computes performance and summary statistics for multiple stocks, from CRSP Stocks Monthly data.

```
usage: mpt YYYMMDD YYYYMMDD items...
  <identifier>
  ...

items:
  cov        - Compute covariance matrix
  perf       - Compute CAPM performance statistics
  summary    - Compute summary statistics
  ret        - Display mkt, tbill and stock returns
```

---

detail Returns I/B/E/S Detail data for multiple stocks; by identifier (symbol or cusip) and fiscal period end-date.

```
usage: detail [US | INTL] <items>...
   <identifier> <YYYY> <A | 1 | 2 | 3 | 4>
   ...

items:
  INTL       - International (US Domestic by default)
  consensus  - Display consensus
```

consensus Returns I/B/E/S Summary detail for multiple stocks; by stocks identifier (symbol or cusip), statistical period dates, and fiscal period indicator.

```
usage: consensus [US | INTL] <items>...
   <identifier> <begYYYYMM> <endYYYYMM> <1..5 | 6..9 | 0>
   ...

items:
  unsplit    - Adjust all dollar values to actual pre-split amounts
  end        - Fiscal period end
  p          - Fiscal period indicator
  n          - Number of analysts
  up         - Number of analysts revised up
  dn         - Number of analysts revised down
  median     - Median estimate
  mean       - Mean estimate
  std        - Standard deviation of estimates
  high       - High estimate
  low        - Low estimate
  actual     - Actual earnings
```

---

annual Returns Compustat annual data items for multiple stocks between a common date range.

```
usage: annual YYYYMMDD YYYYMMDD [brief] [stack] [name]... <data item #s>...
  <identifier>
  ...

items:
  brief      - Do not display headings
  stack      - Display stacked by year (default is display as table)
  name       - Company Name

NOTE: missing data values = -99.008 .. -99.001
```

---

qtr Returns Compustat quarterly data items for multiple stocks between a common date range.

```
usage: qtr YYYYMMDD YYYYMMDD [brief] [stack] <items>... <datum>...
  <identifier>
  ...

items:
  brief      - Do not display headings
  stack      - Display stacked by year (default is display as table)
  name       - Company Name
  report     - Earnings Report Date

NOTE: missing data values = -99.008 .. -99.001
```

---

segment Returns Compustat segment data for multiple stocks between a common date range.

```
usage: segment YYYYMMDD YYYYMMDD [brief] <items>...
  <identifier>
  ...

items:
  segment    - By industry segment
  customer   - By principal customer
  product    - By principal product
```

C C Language Routines

This section describes the usage of new access routines available in the PERC toolkit, with source code.

C.1 Compustat access routines

CST_DB *cst_open(flat,index) –

```
input parameters
   char *flat : name of flat data file
   char *index : name of associated index file (NULL if not indexed)
returns
   an opened Compustat data set
```

---

int cst_ann_read(db,cstdesc,ndb,cst,query) –

```
input parameters
   CST_DB *db[]  : Compustat data sets, previously opened by calls to cst_open
   int cstdesc[] : coverage period of each data set; indicates starting record locations for data in e
   int ndb       : number of Compustat data sets to access
   char *query   : company identifier (ticker,cusip) to seek,
                   NULL to sequentially read from first data set
output parameters
   CSTANN **cst  : data are put here
description
   reads stock data from opened Compustat data sets, spliced together
   based on the starting record numbers specified in cstdesc
   if cst is NULL, space is allocated to contain the stock data
returns
   1 if successful
   0 if unsuccessful or EOF
```

---

int cst_qtr_read(db,cstdesc,ndb,cst,query) –

```
input parameters
   CST_DB *db[] : Compustat data sets, previously opened by calls to cst_open
   int cstdesc[] : coverage period of each data set; indicates starting record locations
                   for data in each set (whether current, back or wayback)
   int ndb      : number of Compustat data sets to access
   char *query  : company identifier (ticker,cusip) to seek,
                  NULL to sequentially read from first data set
output parameters
   CSTQTR **cst : data are read here
returns
   1 if successful
   0 if unsuccessful or EOF
```

---

int cst_seg_read(db,cstdesc,ndb,cst,query) –

```
input parameters
   CST_DB *db[] : Compustat data sets, previously opened by calls to cst_open
   int cstdesc  : starting record to load data (whether current, back or wayback)
   int ndb      : number of Compustat data sets to access
   char *query  : company identifier (ticker,cusip) to seek,
                  NULL to sequentially read from first data set
output parameters
   CSTSEG **cst : data are read here
returns
   1 if successful
   0 if unsuccessful or EOF
```

---

int cst_rdqe(d) –

```
input parameters
   int d : report date of quarterly earnings, in compustat date format
returns
   as calendar date (YYYYMMDD format)
   0 if unsuccessful
```

C.2 Calendar functions

int crsp_cal_days(ymd) –

```
input parameters
   int ymd : calendar date (in YYYYMMDD format)
returns
   number of days after December 31, 1899 (Monday Jan 1, 1990 is 1)
```

---

int crsp_cal_date(d) –

```
input parameters
   int d : number of days after December 31, 1899
returns
   calendar date (in YYYYMMDD format)
```

---

int crsp_cal_trans(cal,ymd,begw,endw,type) –

```
input parameters
   CRSP_CAL *cal : pointer to a source CRSP calendar object
   int ymd       : anchor date, in YYYYMMDD format
   int type      : either BYOLWEEK (overlapping weeks),
                          BYWEEK (non-overlapping weeks, ending on day-of-week of ymd),
```

```
                          BYMONTH (monthly, ending on day-of-month of ymd) ,
                          BYQTR (three-monthtly, ending on day-of-month of ymd), or
                          BYYEAR (annually, ending on day-of-month of ymd)
output parameters
   int **begw
   int **endw    : pointers to int arrays, which should initially be
      NULL.  These will be filled with start and end dates indexes
      ofr each period
description
   reduces a calendar to a coarser periodicity, anchored on date ymd.
returns
   number of periods created
```

---

int crsp_cal_find(begw,endw,n,ymd,offset) –

```
input parameters
   int *begw
   int *endw  : pointers to int arrays with start and end dates indexes
                   for each period
   int n      : number of periods
   int d      : date index to find
   int offset : periods offset by to return
returns
   index of a date (+/- number of periods to offset by) in a range of periods.
```

---

int crsp_cal_weeks(cal,dow,begw,endw) –

```
input parameters
   CRSP_CAL *cal : pointer to a source CRSP calendar object
   int dow       : day of week (0=Sunday, ..., 6=Saturday)
output parameters
   int **begw
   int **endw    : pointers to int arrays, which should initially be
      NULL.  These will be filled with start and end date indexes
      of each period
description
   reduces a daily calendar to weekly, ending on specified day of week
returns
   number of weeks created
```

C.3 Functions to tally CRSP time series data

float crsp_adj_prc(prc,beg,end,fp) –

```
input parameters
   float prc : original stock price
   int beg   : date index of stock price
   int end   : date index to adjust to
   CRSP_TIMESERIES *fp : adjustment factors
returns
   stock price adjusted for intervening stock splits
```

---

float crsp_sum_int(x,beg,end) –

```
input parameters
   int *x    : time series of integer values
   int beg   : begin date index
   int end   : end date index
returns
   sum of values between date indices (ignores <0), inclusive; -99.0 if unsuccessful
```

---

float crsp_sum_float(x,beg,end) –

```
input parameters
   float *x  : time series of floating point values
   int beg   : begin date index
   int end   : end date index
returns
   sum of non-negative values between date indices, inclusive; -99.0 if unsuccessful
```

---

float crsp_prod_float(x,beg,end) –

```
input parameters
   float *x  : time series of floating point values
   int beg   : begin date index
   int end   : end date index
returns
   product of positive values between date indices, inclusive; -99.0 if unsuccessful
```

---

float crsp_compnd_ret(x,beg,end) –

```
input parameters
   float *x  : time series of returns
   int beg   : begin date index
   int end   : end date index
returns
   compounded return between date indices, inclusive
```

---

float crsp_mean_ret(x,beg,end) –

```
input parameters
   float *x  : time series of returns
   int beg   : begin date index
   int end   : end date index
returns
   average return between date indices, inclusive; -99.0 if unsuccessful
```

---

float crsp_cov_ret(x,y,beg,end) –

```
input parameters
   float *x,*y  : time series of returns
   int beg      : begin date index
   int end      : end date index
returns
   sample covariance between date indices, inclusive; -99.0 if unsuccessful
```

---

float crsp_var_ret(x,beg,end) –

```
input parameters
   float *x  : time series of returns
   int beg   : begin date index
   int end   : end date index
returns
   sample variance between date indices, inclusive; -99.0 if unsuccessful
```

---

float crsp_std_ret(x,beg,end) –

```
input parameters
   float *x  : time series of returns
   int beg   : begin date index
   int end   : end date index
returns
   sample standard deviation between date indices, inclusive; -99.0 if unsuccessful
```

C.4 Functions to compute aggregate portfolio returns

void crsp_event_port(tgt,wgt,src,n,beg,end) –

```
output parameters
   float **tgt : equal-wtd returns
   float **wgt : number of stocks each period
input parameters
   float *src : stock returns
   int n      : size of window (aggegate returns to be stored between 0 and n-1)
   int beg    : begin index associated with src returns
   int end    : end index associated with src returns
description
   stock returns between indices are aggregated (equal-weighting) to target series
```

---

void crsp_equal_port(tgt,wgt,src,beg,end) –

```
output parameters
   float **tgt : equal-wtd returns
   float **wgt : number of stocks each period
input parameters
   float *src : stock returns
   int beg    : begin date index, associated with src
   int end    : end date index, associated with src
description
   stock returns between date indices are aggregated (equal-weighting) to same
   indices in target series.
```

---

void crsp_buy_hold(tgt,wgt,src,beg,end) –

```
output parameters
   float **tgt : equal-wtd returns
   float **wgt : value of stocks each period
input parameters
   float *src : stock returns
   int beg    : begin date index, associated with src
   int end    : end date index, associated with src
description
   buy and hold returns for the stock between date indices are aggregated (equal
   weighting) to same date indices in target series.
```

---

void crsp_value_port(tgt,wgt,src,cap,beg,end) –

```
output parameters
   float **tgt : equal-wtd returns
   float **wgt : value of stocks each period
input parameters
   float *src : stock returns
   float *cap : weights (non-negative, but need not sum to 1.0)
   int beg    : begin date index, associated with src
   int end    : end date index, associated with src
description
   stock returns between date indices are aggregated (value weighting on
   market cap on previous date) to same date indices in target series.
```

C.5 Enhanced CRSP access routines

void crsp_clean_stk(stk) –

```
input parameters
   CRSP_STK_STRUCT *stk : stock data structure
description
   sets to missing values all out of range time series variables
```

---

int crsp_stk_read_any(stkcrspnum,stkid,query,keyflag,stk,stkwanted) –

```
input parameters
   int stkcrspnum       : previously opened CRSP data set
   int stkid            : code for periodicity (e.g. daily, monthly)
   char *query          : stock identifier (cusip,permno,sic,ticker)
   int keyflag          : how to search (see CRSP Manual)
   int stkwanted        : data to load
output parameters
   CRSP_STK_STRUCT *stk : stock data structure
description
   incorporates all the random access routines in CRSP Access97,
   as well as a kludge for historical/inactive ticker symbols (not handled
   by CRSP).  Automatically figures out from query string what kind of
   identifier (ie whether cusip,permno,sic,ticker) it is.
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_ind_load(indid,indno,ind,indwanted) –

```
NOTE : DO NOT USE THIS
   input parameters
      int indid            : 420 or 460 (monthly or daily)
      int indno            : index to load
      int indwanted        : data to load
   output parameters
      CRSP_IND_STRUCT *ind : index data structure
   description
      loads desired index data
   returns
      CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

C.6 To return derived CRSP time series

int crsp_shr_num_chk(stk,date,skipflag,share) –

```
input parameters
   CRSP_STK_STRUCT *stk  : source stock data
   int date              : date to find shares outstanding
   int skipflag          : flag for skipping certain types of dists
   CRSP_STK_SHARE *share : share info of actual observation
description
   calls crsp_shr_num, after checking input parameter errors:
   original crsp_shr_num fails miserably if dates out of range!
returns
   shares outstanding
```

---

float *crsp_arr_float(a,i) –

```
input parameters
   void *a : CRSP time-series array of floats
   int i   : index into array
description
   takes a CRSP time-series array of type float, and returns pointer to i'th item
returns
   pointer to i'th float item.
```

---

int *crsp_arr_int(a,i) –

```
input parameters
   void *a : CRSP time-series array of integers
   int i   : index into array
description
   takes a CRSP time-series array of type integer, and returns pointer to i'th item
returns
   pointer to i'th integer item.
```

---

char *crsp_char_int(a,n,i) –

```
input parameters
   void *a : CRSP time-series array of character strings
   int n   : length of character string
   int i   : index into array
description
   takes a CRSP time-series array of character strings of length n, and returns pointer to i'th item
returns
   pointer to i'th character string.
```

---

int crsp_stk_shrcd(stk,shr) –

```
input parameters
   CRSP_IND_STRUCT *stk  : contains stock data
output parameters
   CRSP_TIMESERIES **shr : to contain derived time series
description
   creates a time series of share codes
   if shr contains NULL, space is allocated; else old space is released and re-initialised
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_stk_siccd(stk,sic) –

```
input parameters
   CRSP_IND_STRUCT *stk  : contains stock data
output parameters
   CRSP_TIMESERIES **sic : to contain derived time series
description
   creates a time series of sic codes
   if sic contains NULL, space is allocated; else old space is released and re-initialised
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_stk_exchcd(stk,exch) –

```
input parameters
   CRSP_IND_STRUCT *stk  : contains stock data
output parameters
   CRSP_TIMESERIES **exch : to contain derived time series
description
   creates a time series of exchange codes
   if exch contains NULL, space is allocated; else old space is released and re-initialised
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_stk_ticker(stk,ticker) –

```
input parameters
   CRSP_IND_STRUCT *stk  : contains stock data
output parameters
   CRSP_TIMESERIES **ticker : to contain derived time series
description
   creates a time series of historical ticker symbols
   if ticker contains NULL, space is allocated; else old space is released and re-initialised
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_stk_shrcls(stk,shrcls) –

```
input parameters
   CRSP_IND_STRUCT *stk  : contains stock data
output parameters
   CRSP_TIMESERIES **shrcls : to contain derived time series
description
   creates a time series of historical share class codes
   if shrcls contains NULL, space is allocated; else old space is released and re-initialised
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_stk_divamt(stk,divamt) –

```
input parameters
   CRSP_IND_STRUCT *stk  : contains stock data
output parameters
   CRSP_TIMESERIES **divamt : to contain derived time series
description
   creates a time series of dividend amounts paid
   if divamt contains NULL, space is allocated; else old space is released and re-initialised
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_stk_facpr(stk,facpr) –

```
input parameters
   CRSP_IND_STRUCT *stk  : contains stock data
output parameters
   CRSP_TIMESERIES **facpr : to contain derived time series
description
   creates a time series of price adjustment factors due to stock splits
   if facpr contains NULL, space is allocated; else old space is released and re-initialised
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_ind_xs(xs,stk,indret) –

```
output parameters
   CRSP_TIMESERIES **xs    : to contain derived time series
input parameters
   CRSP_STK_STRUCT *stk    : contains stock data, in particular raw returns
   CRSP_TIMESERIES *indret : contains base index returns
description
   creates a time series of excess stock returns (equals raw minus base index)
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_ind_port(indret,stk,portind,portnum,portmax) –

```
output parameters
   CRSP_TIMESERIES **indret : to contain derived time series
input parameters
   CRSP_STK_STRUCT *stk     : contains stock data, in particular portfolio assignments
   CRSP_IND_STRUCT *portind : contains previously loaded segment portfolio returns
          (e.g. daily: 1000092,1000052,1000072,1000012,1000032,1000112,1000132,1000152,1000172
   int portnum              : portfolio type (0..7/8)
   int portmax              : max number of portfolios assigned for that type
description
   creates a time series of index returns, associated with a stock's portfolio
   assignment, for the portfolio type (portnum) specified
returns
   CRSP_SUCCESS if successful (see CRSP Manual for error codes)
```

---

int crsp_stk_stat(stk,p,ymd) –

```
input parameters
   CRSP_STK_STRUCT *stk : contains DAILY stock data
   int p                : portfolio type
   int ymd              : calendar date (in YYYYMMDD format)
returns
   portfolio statistic from current calendar year (-99.0 if unsuccessful)
```

---

int crsp_stk_port(stk,p,ymd) –

```
input parameters
   CRSP_STK_STRUCT *stk : contains DAILY stock data
   int p                : portfolio type (0..7/8)
   int ymd              : calendar date (in YYYYMMDD format)
returns
   returns CURRENT portfolio assignment, based on prior year statistic, 0 if unsuccessful
```

C.7 General input/ouput routines

int db_getline(fp,s) –

```
input parameters
   FILE *fp : input file pointer
output parameters
   char *s  : location to store string in
description
   reads a line from file up to CR or EOF
returns
   number of characters read, discarding CR (EOF if unsuccessful)
```

---

void db_err(ofp,s) –

```
input parameters
   FILE *ofp : log file to print error message to
   char *s   : error message
description
   prints error message and exits
```

---

void *db_malloc(siz) –

```
input parameters
   int siz : number of bytes to allocate
description
   allocates memory; prints error message and exits if unsuccessful
```

---

void *db_calloc(siz,n) –

```
input parameters
   int siz : size of an element, in bytes
   int n   : number of elements to allocate
description
   allocates memory; prints error message and exits if unsuccessful
```

C.8 Index lookup routines

int db_cmp(a,b) –

```
input parameters
   DB_INDEX *a,*b : DB_INDEX entries to compare
description
   this function passed to bsearch and qsort for index lookups
```

---

int db_seek(fp,db,n,query) –

```
input parameters
   FILE *fp     : flat data file to seek
   DB_INDEX *db : array of indices into data file
   int n        : number of index keys
   char *query  : key string to lookup on
description
   sets the data file position to the record corresponding to the query string,
returns
    0 if successful,
   -1 if unsuccessful
```

---

int db_open(s,db) –

```
input parameters
   char *s      : file name of index
output parameters
   DB_INDEX *db : array of indices read into here
description
   loads index keys and associated data file locations (in bytes)
returns
   number of index records read
```

C.9 General routines to process command line

void db_set_flag(db_f,nf,arg,n) –

```
input/output parameters
   DB_FLAG *db_f : list of potential commands and default values
   int nf        : number of potential commands
input parameters
   char *arg[]   : commands specified (usually the argv command line)
   int n         : number of command line arguments specified
description
   sets the db_f list based arguments submitted on command line
```

---

void db_get_flag(db_f,nf,f) –

```
input parameters
   DB_FLAG *db_f : list of commands and values
   int nf        : number of commands
   char *f       : command string to query
returns
   the value associated with the queried command (0 if unsuccessful)
```

C.10 Basic numeric manipulations

int db_monthbet(a,b) –

```
input parameters
   int a,b : calendar months (in YYYYMM format)
returns
   b minus a (in months), i.e. number of months a is before b
```

---

int db_monthadd(yymm,n) –

```
input parameters
   int yymm : current date (in YYYYMM format)
   int n    : number of months to add, negative to subtract
returns
   adds n months to yymm
```

---

float db_median(x,n) –

```
input parameters
   float *x : time series vector
   int n    : number of observations
returns
   median value (note vector x becomes sorted)
```

---

float db_sum(x,n) –

```
input parameters
   float *x : time series vector
   int n    : number of observations
returns
   sum of values in x
```

---

float db_mean(x,n) –

```
input parameters
   float *x : time series vector
   int n    : number of observations
returns
   mean value, 0.0 by default
```

---

float db_cov(x,y,n) –

```
input parameters
   float *x,*y : time series vectors
   int n       : number of observations
returns
   sample covariance, 0.0 if unsuccessful
```

---

float db_var(x,n) –

```
input parameters
   float *x : time series vectors
   int n    : number of observations
returns
   sample variance, 0.0 if unsuccessful
```

---

float db_std(x,n) –

```
input parameters
   float *x : time series vectors
   int n    : number of observations
returns
   sample standard deviation, 0.0 if unsuccessful
```

---

int db_regress(y,x,n,alpha,beta,reserr) –

```
input parameters
   float *y      : left-hand-side dependent variable
   float *x      : light-hand-side independent variable
   int n         : number of observations
output parameters
   float *alpha  : regression intercept
   float *beta   : regression slope
   float *reserr : sample standard deviation of regression residuals
returns
   number of degress of freedom n-1 (0 if unsuccessful)
```

---

char *sicname() –

```
input parameters
   int sic : 2-or 4-digit SIC code
returns
   2-digit SIC name
```

---

int sic25(sic) –

```
input parameters
   int sic : 4-digit SIC code
returns
   Industry portfolio (1..25), based on Hong, Lim and Stein (1999)
```

C.11 I/B/E/S routines to return specific data items

char ib_period(c) –

```
input parameters
   char c : fiscal period indicator
description
   converts the fiscal period indicator to periodicity code
returns
   'A' (annual) : if c is '1'..'5' or 'E'..'I'
   'Q' (quarterly) : if c is '6'..'9' or 'N'..'Q'
   'S' (semi-annual) : if c is 'A'..'D'
   'L' (long term growth) : if c is '0'
```

---

int ib_tofy(yymm,c) –

```
input parameters
   int yymm : fiscal end date (YYYYMM format)
   char c : periodicity code
returns
   next nearest Mar/Jun/Sep/Dec date
```

---

char *ib_dcty(ib) –

```
input parameters
   IBES *ib : stock data structure
returns
   country code of stock (based on cusip/sedol of detail records)
```

---

float ib_dactual(end,period,ib) –

```
input parameters
   int end : fiscal end date (YYYYMM format)
   char period : periodicity code ('A', 'Q', or 'S')
   IBES *ib : stock data structure
returns
   returns actual (I/B/E/S-adjusted) earnings per share (-99.0 if missing)
```

---

float ib_dstop(broker,end,period,ib) –

```
input parameters
   int broker : broker code
   int end : fiscal end date (YYYYMM format)
   char period : periodicity code ('A', 'Q', or 'S')
   IBES *ib : stock data structure
returns
   stop date (0 if not found) for broker's estimates of this fiscal date
```

---

int ib_dreport(yymm,ib) –

```
input parameters
   int yymm : fiscal end date (YYYYMM or YYYYMMDD format)
   IBES *ib : stock data structure
returns
   earnings report date
```

---

int ib_dsig(yymm,flag,ib) –

```
input parameters
   int yymm : date (YYYYMM or YYYYMMDD format)
   int flag : SIGS, SIGI or SIGG (sector, industry or group)
   IBES *ib : stock data structure
returns
   sector, industry or group code as at specified date
```

---

float ib_dfactor(yymmdd,ib) –

```
input parameters
   int yymmdd : date (YYYYMMDD format)
   IBES *ib   : stock data structure
returns
   factor to multiply all dollar amounts back to historical pre-split date (based on Detail records)
```

---

float ib_dunsplit(value,yymmdd,ib) –

```
input parameters
   float value : dollar amount, as reported in I/B/E/S
   int yymmdd  : date (YYYYMMDD format)
   IBES *ib    : stock data structure
returns
   adjusts dollar value back to a historical pre-split date (based on Detail records)
```

---

float ib_dfactor(yymmdd,ib) –

```
input parameters
   int yymmdd : date (YYYYMMDD format)
   IBES *ib   : stock data structure
returns
   factor to multiply all dollar amounts back to historical pre-split date (based on Summary records)
```

---

float ib_dunsplit(value,yymmdd,ib) –

```
input parameters
   float value : dollar amount, as reported in I/B/E/S
   int yymmdd  : date (YYYYMMDD format)
   IBES *ib    : stock data structure
returns
   adjusts dollar value back to a historical pre-split date (based on Summary records)
```

---

float ib_ssize(yymm,ib) –

```
input parameters
   int yymmdd  : date (YYYYMM or YYYYMMDD format)
   IBES *ib    : stock data structure
returns
   market size at mid-month of specified date (based on Summary records)
```

---

float ib_sprice(yymm,ib) –

```
input parameters
   int yymmdd  : date (YYYYMM or YYYYMMDD format)
   IBES *ib    : stock data structure
returns
   market price at mid-month of specified date (based on Summary records)
```

---

float ib_sactual(yymm,period,ib) –

```
input parameters
   int yymm    : date (YYYYMM or YYYYMMDD format)
   char period : periodicity code ('A' or 'Q'' or 'S')
   IBES *ib    : stock data structure
returns
   actual I/B/E/S-reported earnings per share (based on Summary records)
```

---

char ib_dpdi(end,ib) –

```
input parameters
   int end     : date (YYYYMM or YYYYMMDD format)
   IBES *ib    : stock data structure
returns
   whether company's earnings are in primary 'P' or diluted 'D' basis (based on Detail records)
```

---

char ib_spdi(end,ib) –

```
input parameters
   int end     : date (YYYYMM or YYYYMMDD format)
   IBES *ib    : stock data structure
returns
   whether company's earnings are in primary 'P' or diluted 'D' basis (based on Summary records)
```

---

char ib_scover(yymm,ib) –

```
input parameters
   int yymm    : date (YYYYMM or YYYYMMDD format)
   IBES *ib    : stock data structure
returns
   number of FY1 analysts at mid-month of specified date (based on Summary)
```

---

char ib_sest(end,fpi,ib) –

```
input parameters
   int end  : fiscal end date (YYYYMM format)
   char fpi : fiscal period indicator ('1'..'9', 'A'..'Q')
   IBES *ib : stock data structure
returns
   latest number of analysts providing estimates for specified fiscal period (based on Summary)
```

---

char ib_smedian(end,fpi,ib) –

```
input parameters
   int end  : fiscal end date (YYYYMM format)
   char fpi : fiscal period indicator ('1'..'9', 'A'..'Q')
   IBES *ib : stock data structure
returns
   latest median estimate for specified fiscal period (based on Summary)
```

---

char ib_serr(end,fpi,ib) –

```
input parameters
   int end  : fiscal end date (YYYYMM format)
   char fpi : fiscal period indicator ('1'..'9', 'A'..'Q')
   IBES *ib : stock data structure
returns
   latest standard error of estimate for specified fiscal period (based on Summary)
```

C.12 I/B/E/S data access routines

ID_DB ib_init(intl) –

```
input parameters
   char intl : 'I' to access International, 'U' (US Domestic) is default
returns
   I/B/E/S data set, which can be subsequently read via sequential or direct access
```

---

int ib_get(ibd,ib) –

```
input parameters
   IB_DB *ibd : previously initialized I/B/E/S data set
output parameters
   IBES **ib  : stock data structure to contain I/B/E/S data
description
   reads next stock's I/B/E/S data in sequence
returns
   1 if successful
   0 if unsuccessful or EOF
```

int ib_read_any(s,ib,ibd) –

```
input parameters
   char *s    : query identifier (ticker or cusip)
   IB_DB *ibd : previously initialized I/B/E/S data set
output parameters
   IBES **ib  : stock data structure to contain I/B/E/S data
description
   reads specified stock's I/B/E/S data via direct access
   if ib is NULL, space is allocated to contain the stock data
returns
   1 if successful
   0 if unsuccessful or EOF
```

C.13 Routines to filter and group Detail data

int ib_dbaddetail(z) –

```
input parameters
   DET *z : a single Detail record
returns
   1 if invalid record (invalid fiscal dates or earnings estimate)
   0 if unsuccessful (i.e. is valid record)
```

---

int ib_dfilter (fil,ib) –

```
input parameters
   int *fil(DET *z) : function to identify records to be filtered out
                      (NULL uses default of ib_dbaddetail)
   IBES *ib         : stock data structure
description
   removes unwanted detail records (placed at end of array)
returns
   number of detail records remaining
```

---

void ib_dgroup(grp,srt,n,ib) –

```
input parameters
   int *grp() : comparison function that returns 0 if two detail records
                are to be in the same group
   int *srt() : comparison function used to sort all detail records
   int n      : number of detail records to group
   IBES *ib   : stock data structure
description
   sorts all Detail records (by comparison function in second argument, w),
   then groups all records (by coarser comparison function in first argument)
```

```
by putting beginning indices, ending indices and number of indices
into global variables ib_btag,ib_etag,ib_ntag
By default (if NULL comparison functions specified), groups Detail records by
periodicity (annual, quarterly, long-term) and fiscal end date, and sorts by
date of estimate.
```

D Web Pages and CGI Programs

This section lists the HTML web query pages and associated CGI Perl scripts and utility programs which can be used to provide web-based access to CRSP Stocks, Indices and Bonds; Compustat Industrial Annual, Quarterly and Segments; and I/B/E/S US and International Detail and Summary data. The same utility programs described previously are used to drive web-based access.

CRSP Stocks: Daily –

- (crsp/portd.html, portd.txt, portd) Returns daily/weekly time series of individual stock data for a portfolio of stocks within a common date range. Optionally computes equal- or value-weighted portfolios.
- (crsp/manyd.html, manyd.txt, manyd) Returns daily/weekly time series of individual stock data for multiple stocks.
- (crsp/eventd.html, eventd.txt, eventd) Returns daily/weekly time series of individual stock data for multiple stocks around event dates. Optionally computes equal-weighted portfolios.
- (crsp/riskd.html, riskd.txt, riskd) Returns annual risk statistics for individual stocks.

CRSP Stocks: Monthly –

- (crsp/portm.html, portm.txt, portm) Returns month-end time series of individual stock data for a portfolio of stocks within a common date range. Optionally computes equal- or value-weighted portfolios.
- (crsp/manym.html, manym.txt, manym) Returns month-end time series of individual stock data for multiple stocks.
- (crsp/eventm.html, eventm.txt, eventm) Returns month-end time series of individual stock data for multiple stocks around event months. Optionally computes equal-weighted portfolios.
- (crsp/summary.html, mpt.txt, mpt) Returns summary and performance statistics for multiple stocks within a common date range.

CRSP Stocks –

- (crsp/info.html, info.txt, info) Returns descriptive header and events information for multiple stocks within a common date range.

CRSP Indices –

- (indices/cap.html, indices.txt, indices) Returns time series of index levels and returns for cap-based portfolios.
- (indices/decile.html, indices.txt, indices) Returns time series of index levels and returns for size decile portfolios.
- (indices/market.html, indices.txt, indices) Returns time series of index levels and returns for market-wide portfolios.
- (indices/risk.html, indices.txt, indices) Returns time series of index levels and returns for risk-based decile portfolios.

CRSP Bonds: Monthly –

- (bonds/mbx.html, mbx.txt) Returns cross-sectional government bonds data for a specified month.
- (bonds/riskfree.html, riskfree.txt) Returns risk-free rates, fixed term indices, forward rates and maturity portfolio returns in a specified date range.

Compustat Industrial –

- (compustat/qtr.html, qtr.txt, qtr) Returns quarterly data for multiple stocks within a common date range.
- (compustat/annual.html, annual.txt, annual) Returns annual data for multiple stocks within a common date range.
- (compustat/segment.html, segment.txt, segment) Returns segment data for multiple stocks within a common date range.

I/B/E/S –

- (ibes/consensus.html, consensus.txt, consensus) Returns consensus estimates data for multiple stocks for specified range of statistical periods and fiscal period indicator.
- (ibes/detail.html, detail.txt, detail) Returns detail estimates data for multiple stocks within specified date range for a specified fiscal end date.

E Installation Steps

1. Create Directory Structure.

   The PERC toolkit of C-language programs and routines run in a unix environment. YOu should ensure that your unix machine platform be configured to at least support the installation and anticipated usage of the CRSP Access97 data files; plus additional disk space for any Compustat and/or I/B/E/S data files.

   It is recommended that you set up a directory structure using the following default locations. If you use different physical locations, you should use the unix `ln -s` command to create symbolic links from your physical disk volumes to the recommended directory structure (an alternative, but not recommended, solution is to change all the corresponding path names in `db_local.h`, as well as all the `CRSP_ROOT` variables in the makefiles).

   /db/crsp (CRSP_ROOT) This will be the base directory for CRSP Access97 data files.

   /db/crsp/daily (CRSP_DSTK) This will be the directory for CRSP Stocks daily files, as well as overlaid Indices files.

   /db/crsp/monthly (CRSP_MSTK) This will be the directory for CRSP Stocks monthly files, as well as overlaid Indices files.

   /db/crsp/data Supplemental CRSP files create by PERC's installation programs will be placed here (such as the historical tickers index file and securities listings)

   /db/ibes/data I/B/E/S data will be copied here from CD's (supplemental data, such as cusip/symbol index files and securities listings, will also be created and placed here).

   /db/compustat/data Compustat character data files will be copied here from tapes (supplemental data, such as cusip/symbol index files and securities listings, will also be created and placed here).

   /db/perc (PERC_ROOT) This will be the base directory for all PERC programs, web pages and documents.

   /db/perc/src (PERC_SRC) This will be the directory for all PERC source code and programs

   /db/perc/cgi-src (PERC_CGISRC) This will be the directory that holds (a copy of) the PERC CGI scripts.

2. Create a new user named, say, *perc*. We shall consistently login as this user when loading the data files, and installing and modifying PERC programs. Hence this user "owns" and should have full read-write-execute privileges to the directories, programs and source code that comprise the PERC toolkit. This same user would also "own" the data files and the directories where the data is installed. Hence ALL subsequent installation steps should be taken while logged in as this user. Also, the unix commands presented below assume you are running the C-Shell (`/bin/csh` or `/bin/tcsh`) environment.

3. Load new format CRSP (Access97) from CD.

   Copy the `setup.sh` script from CRSP Access97 CD's to the base directory `/db/crsp` (in which data will be installed) and run the script. Note that CRSP's setup script will prompt for local directory names in which to install the data; you should use the directory names listed above. If you use different directory locations (not recommended), you will need to

change all the associated path names in the `db_local.h` file. Repeat this step for each of the CRSP data products Stocks Daily, Stocks Monthly and finally Indices (note that the Indices CD will ask to overlay Indices data over the Daily and Monthly Stocks data you had installed earlier). The CRSP Bonds data does not have a setup script: simply copy the files over to a directory (`/db/crsp/bonds`).

4. Load PERC program files

   - Copy the PERC distribution file `perc.tar` to the directory `/db/perc`. Unpack this distribution file, with the following `tar` command:

     ```
     tar -xvf perc.tar
     ```

   - compile the programs by running `make` (you may need to modify the `makefile`, listed later in this report, to reflect where the `ar`, `ranlib` and `cc` programs reside on your system). Note: PERC toolkit requires CRSP Access97 C programs to be installed first.

5. Install Perl Version 5

   - Download the latest version of Perl version 5 from `http://www.perl.com`. If available, install the pre-compiled binaries version for your particular unix machine.
   - Ensure that Perl is accessible at `/usr/local/bin/perl` (i.e., the executable program `perl` that you have installed should be symbolically linked as `/usr/local/bin/perl`). All the Perl programs in PERC assume this full path name for the Perl executable program file.

6. Load Compustat data files from tapes:

   The data will be loaded to `/db/compustat/data`. You may skip those commands corresponding to the Compustat files you did not subscribe to.

   - Load Annual data files from tape.

     Change directory to `/db/perc/src/lib`, and run unix script `load.cst` to copy (and compress) character data from tapes to disk. This script takes two arguments: the coverage period (annual, annbak or annway denoting the current 20-year, previous 20-year or earliest 20-year period), and the company coverage (pst, full or mrg for primary/supplemental/tertiary companies, full coverage OTC companies or merged research inactive companies).

     ```
     load.cst annual pst  (Annual PST current data.)
     load.cst annual full (Annual Full Coverage current data.)
     load.cst annual mrg  (Annual Merged Research current data.)
     load.cst annbak pst  (Annual PST back data.)
     load.cst annbak full (Annual Full Coverage back data.)
     load.cst annbak mrg  (Annual Merged Research back data.)
     load.cst annway pst  (Annual PST wayback data.)
     load.cst annway full (Annual Full Coverage wayback data.)
     load.cst annway mrg  (Annual Merged Research wayback data.)
     ```

   - Combine annual data files by coverage period.

     Change directory to `/db/perc/src/lib`, and run the unix script `load.cst` to merge annual data files by coverage period. This script takes one argument: the coverage

period (annual, annbak or annway denoting the current 20-year, previous 20-year or earliest 20-year period).

```
load.cst annual  (Annual data for current 20-year period;
                  all companies covered in PST, FULL and MRG will be merged into one file)
load.cst annbak  (Annual Backdata for previous 20-year period;
                  all companies covered in PST, FULL and MRG will be merged into one file)
load.cst annway  (Annual wayback data for earliest 20-year period;
                  all companies covered in PST, FULL and MRG will be merged into one file)
```

- Load Quarterly data files from tape.

  Change directory to `/db/perc/src/lib`, and run the unix script `load.cst` to copy (and compress) character data from tapes to disk. This script takes two arguments: the coverage period (qtr, qtrbak or qtrway denoting the current 48-quarter, previous 48-quarter or earliest 48-quarter period), and the company coverage (pst, full or mrg for primary/supplemental/tertiary companies, full coverage OTC companies or merged research inactive companies).

```
load.cst qtr pst     (Quarterly PST current data.)
load.cst qtr full    (Quarterly Full Coverage current data.)
load.cst qtr mrg     (Quarterly Merged Research current data.)
load.cst qtrbak pst  (Quarterly PST back data.)
load.cst qtrbak full (Quarterly Full Coverage back data.)
load.cst qtrbak mrg  (Quarterly Merged Research back data.)
load.cst qtrway pst  (Quarterly PST wayback data.)
load.cst qtrway full (Quarterly Full Coverage wayback data.)
load.cst qtrway mrg  (Quarterly Merged Research wayback data.)
```

- Combine quarterly data files by coverage period.

  Change directory to `/db/perc/src/lib`, and run the unix script `load.cst` to merge quarterly data files by coverage period. This script takes one argument: the coverage period (qtr, qtrbak or qtrway denoting the current 48-quarter, previous 48-quarter or earliest 48-quarter period).

```
load.cst qtr     (Quarterly data for current 48-quarter period;
                  all companies covered in PST, FULL and MRG will be merged into one file.
load.cst qtrbak  (Quarterly Backdata for previous 48-quarter period;
                  all companies covered in PST, FULL and MRG will be merged into one file)
load.cst qtrway  (Quarterly wayback data for earliest 48-quarter period;
                  all companies covered in PST, FULL and MRG will be merged into one file)
```

- Load Segment data files from tape.

  Change directory to `/db/perc/src/lib`, and run unix script `load.cst` to copy (and compress) character data from tapes to disk. This script takes two arguments: the coverage period (segment or segbak denoting the current 7-year, previous 7-year or earliest 7-year period), and the company coverage (full or mrg for full/PST coverage or merged research inactive companies).

```
load.cst segment full (Segment PST/Full Coverage current data.)
load.cst segment mrg  (Segment Merged Research current data.)
load.cst segbak  full (Segment PST/Full Coverage back data.)
load.cst segbak  mrg  (Segment Merged Research back data.)
```

- Combine segment data files by coverage period.

  Change directory to `/db/perc/src/lib`, and run the unix script `load.cst` to merge annual data files by coverage period. This script takes one argument: the coverage period (segment or segbak denoting the current 7-year, previous 7-year or earliest 7-year period).

```
load.cst segment  (Segment data for current 7-year period;
                   all companies covered in FULL/PST and MRG will be merged into one file)
load.cst segbak   (Segment Backdata for previous 7-year period;
                   all companies covered in FULL/PST and MRG will be merged into one file)
```

- If desired, load the data files as permanent, indexed SAS data sets. Change directory to `/db/perc/src/lib`, and run the SAS programs `annual.sas`, `quarterly.sas`, and `segment.sas`.

7. Load old format CRSP character data from tapes (if desired: this step is used ONLY if you are loading the CRSP data as permanent SAS data sets).

   - Change directory to `/db/perc/src/lib`, and run the unix script `load.crsp` to copy (old format) CRSP character data from tapes to disk.
   - Run SAS programs `monthly.sas` and `daily.sas` to load Monthly and Daily CRSP Stocks data respectively as indexed, permanent SAS data sets.

8. Load I/B/E/S character data from CD's.

   The data will be copied into /db/ibes/data. Copy all the I/B/E/S character data from each CD, using the unix script `load.ibes`: this is essentially repeats the copy command.

9. Recompile PERC library and installation programs.

   - You may need to first update the parameters in your local configuration file, depending on the data sets (particularly Compustat) that you subscribed to. The configuration file is `db_local.h`, in the `/db/perc/src/include` directory: see the instructions therein.
   - Change directory to `/db/perc` and run `make`. You may need to modify the `ar`, `ranlib`, `cc` and `CRSP_ROOT` variables defined at the top of the `makefile` to correspond to the locations of these programs or paths on your workstation. You can safely ignore errors caused by the make command trying to symbolically link (using the `ln -s` command) files that already exist or remove files that may not yet exist.

10. Create PERC supplemental data files. Change directory to `/db/perc/src/lib`, and run the installation script `install.csh`. These run a number of C programs to generate supplemental files. These include index files containing symbol/cusip indices and securities listings, and new merged large binary files for I/B/E/S data. For your information, these C programs are described below (you need not run these programs individually, they are already contained in the `install.csh` script).

    - `instibes` to generate index and securities listing, and create merged binary file for I/B/E/S US and Domestic Detail and Summary data.
    - `instcrsp` to generate supplemental index (of historical tickers) and securities listings for CRSP.

- `instann` to generate random access index and a listing of securities for Compustat annual data.
- `instqtr` to generate random access index and a listing of securities for Compustat quarterly data.
- `instseg` to generate random access index and a listing of securities for Compustat segment data.

11. Install Web Server and PERC CGI programs (if web-access is desired)

    - If you wish to serve out the data through a web server, obtain and install a web server program on the same machine that the data are loaded. The PERC system was developed with the Apache server (freely available for download from `http://www.apache.org`: if possible, download the precompiled binary version for your computing platform), but any server should suffice. The web server program should run from the same machine that houses the data files, and be capable of running CGI programs and Perl (Version 5) scripts. You should also ensure that the *zip* compression program (which is used by the web-interface programs to return large data files in a compressed format suitable for download) is accessible from your server's ScriptAlias (usually *cgi-bin*) directory.

      It is critical that the web server owner AND the user name under which the server is run must have write access to the server's ScriptAlias (*cgi-bin*) directory: PERC's CGI programs create and use temporary files in this directory. The user name is typically set in the web server's configuration file (for Apache, this is `~/apache/conf/httpd.conf`). Also the web server owner should have read and executable access to the PERC_ROOT (`/db/perc`) directory and all the files and subdirectories it contains, as well as to the data files. You could change the access mode of `/db/perc` to enable all users to have read and execute premissions by using the following `chmod` command

      ```
      chmod -R a+rx /db/perc /db/perc/*
      chmod a+rx / /db
      ```

    - To install PERC HTML web query pages, login as your web server's owner, after ensuring that your web server owner has read permission to the PERC_ROOT (`/db/perc`) directory and all the files and subdirectories it contains. Change directory to the DocumentRoot (usually *htdocs*) directory of your web server, and symbolically link the PERC_ROOT (`/db/perc`) directory as a subdirectory there, using the command

      ```
      ln -s /db/perc
      ```

    - To install PERC CGI programs, login as your web server's owner, after ensuring that your web server owner has read permission to both the PERC_SRC (`/db/perc/src`) and and PERC_CGISRC (`/db/perc/cgi-src`) directories, which contain the CGI programs and Perl scripts respectively. Change directory to `/db/perc/cgi-src` and run `install.web`. This install script will prompt you to enter the path name of your ScriptAlias (*cgi-bin*) and DocumentRoot (*htdocs*) directories. It then copies the required CGI programs and creates a symbolic link to the ScriptAlias in the DocumentRoot directory to allow web access to output data files created by the CGI programs.
    - As the root user, run the script `cleanup.csh` from your web server's ScriptAlias (*cgi-bin*) directory as a background daemon (this script was installed there in the previous steps).

This script monitors and removes temporary files periodically (every 900 seconds) that are created in the ScriptAlias (*cgi-bin*) directory by PERC's web access programs. Along with the web server program, this script should be called in your unix machine's startup file, to automatically run each time your machine boots up.

F Unix scripts to load data files from tape/CD to disk

These unix scripts can be used to load character data files from tape or CD (distributed by the data vendors) onto disk.

CRSP Stocks Monthly and Daily (old format)

Figure 40: Script to load CRSP Stocks old format files

```
#! /bin/csh
#--------------------------------------
# To install monthly tape
#--------------------------------------
set $CAL = /db/crsp/data/cal.monthly
set $STK = /db/crsp/data/monthly
echo Please make sure the monthly tape is write-protected and mount it.
echo Press ENTER when ready.
set ans = $<

mt -f /dev/rmt/0n rewind
mt -f /dev/rmt/0n fsf 8
dd if=/dev/rmt/0n ibs=31200 cbs=130 conv=unblock of=$CAL
dd if=/dev/rmt/0n ibs=32000 cbs=400 conv=unblock of=$STK
compress $STK1


#--------------------------------------
# To install daily tapes
#--------------------------------------
set $CAL = /db/crsp/data/cal.daily
set $STK1 = /db/crsp/data/daily.1
set $STK2 = /db/crsp/data/daily.2

echo Please make sure the daily tape, vol 1 is write-protected and mount it.
echo Press ENTER when ready.
set ans = $<

mt -f /dev/rmt/0n rewind
mt -f /dev/rmt/0n fsf 8
dd if=/dev/rmt/0n ibs=31200 cbs=130 conv=unblock of=$CAL
dd if=/dev/rmt/0n ibs=32000 cbs=400 conv=unblock of=$STK1

echo Please make sure the daily tape, vol 2 is write-protected and mount it.
echo Press ENTER when ready.
set ans = $<

mt -f /dev/rmt/0n rewind
dd if=/dev/rmt/0n ibs=32000 cbs=400 conv=unblock of=$STK2
compress $STK1 $STK2
```

Compustat Industrial (Annual, Quarterly, Segment)

Figure 41: Script to load Compustat Industrial files

```
#! /bin/csh
if ($1 =="") then
  echo "usage: load.cst [ annual | annbak | annway | qtr | qtrbak | qtrway | seg | segbak ] [ pst | full | mrg ]"
  exit
endif

#--------------------------------------
# To load annual tapes
#--------------------------------------
if ($1 == "annway" || $1 == "annual" || $1 == "annbak") then
  if ($2 == "") then
    set STK = /db/compustat/data/"$1".*.gz
    set OUT = /db/compustat/data/"$1"
    gunzip -c $STK >! $OUT
  else
    set STK = /db/compustat/data/"$1"."$2"
    echo Please make sure the "$1"."$2" tape is write-protected and mount it.
    echo Data will be saved as "$STK" and gzipped.
    echo Press ENTER when ready.
    set ans = $<
    echo Thank you, please be patient...

    mt -f /dev/rmt/1n rewind
    dd if=/dev/rmt/1 ibs=8332 cbs=8332 of=$STK
    gzip $STK
    mt -f /dev/rmt/1n offline
  endif
  exit
endif

#--------------------------------------
# To load quarterly tapes
#--------------------------------------
if ($1 == "qtr" || $1 == "qtrbak" || $1 == "qtrway") then
  if ($2 == "") then
    set STK = /db/compustat/data/"$1".*.gz
    set OUT = /db/compustat/data/"$1"
    gunzip -c $STK >! $OUT
  else
    set STK = /db/compustat/data/"$1"."$2"
    echo Please make sure the "$1"."$2" tape is write-protected and mount it.
    echo Press ENTER when ready.
    set ans = $<
    echo Thank you, please be patient...
    mt -f /dev/rmt/1n rewind
    dd if=/dev/rmt/1 ibs=27552 cbs=9184 of=$STK
    gzip $STK
  endif
  exit
endif
```

```
#-------------------------------------
# To load business segment tapes
#-------------------------------------
if ($1 == "segment" || $1 == "segbak") then
  if ($2 == "") then
    set STK = /db/compustat/data/"$1".*.gz
    set OUT = /db/compustat/data/"$1"
    gunzip -c $STK >! $OUT
  else
    set STK = /db/compustat/data/"$1"."$2"
    echo Please make sure the "$1"."$2" tape is write-protected and mount it.
    echo Press ENTER when ready.
    set ans = $<
    echo Thank you, please be patient...
    mt -f /dev/rmt/1n rewind
    dd if=/dev/rmt/1 bs=7740 cbs=774 of=$STK
    gzip $STK
  endif
  exit
endif

#-------------------------------------
# To load annual canadian tapes
#-------------------------------------
set STK = /db/compustat/data/anncan
echo Please make sure the Annual PST tape is write-protected and mount it.
echo Press ENTER when ready.
set ans = $<
echo Thank you, please be patient...

mt -f /dev/rmt/1n rewind
dd if=/dev/rmt/1 ibs=8332 cbs=8332 of=$STK
```

I/B/E/S US/Domestic Detail/Summary

Figure 42: Script to load I/B/E/S files

```
#!/bin/csh
while (1)
  echo "Load next I/B/E/S CD, and press ENTER (or hit CTRL-C to end)"
  set ans = $<
  /usr/bin/cp /cdrom/cdrom0/* /db/ibes/data
  echo "Copying files, please wait...."
end;
```

G Regenerating and recompiling toolkit programs

The following `makefile` supports the regeneration of PERC toolkit and routines; simply run `make` from the the base PERC directory (usually `/db/perc`). In turn, it uses secondary description files, also listed below, to regenerate the routines library and installation programs; utility programs; and sample programs.

Figure 43: Main makefile for compiling toolkit library and programs

```
#   You may need to modify the CRSP_ROOT, CC, AR, RANLIB variables...
CRSP_ROOT=/db/crsp
CC=/opt/SUNWspro/bin/cc
AR=/usr/ccs/bin/ar
RANLIB=/usr/ccs/bin/ranlib

all : library utility sample

library :
cd src/lib; make -i -e all
utility :
cd src; make -i -e all
sample :
cd src/c; make -i -e all
```

Figure 44: Secondary makefile for compiling library routines and installation programs

```
CC=/opt/SUNWspro/bin/cc
AR=/usr/ccs/bin/ar
RANLIB=/usr/ccs/bin/ranlib
CRSP_ROOT=/db/crsp

I=../include/
IFLAG=-DUNIX=1 -I../include/ -I$(CRSP_ROOT)/include
LFLAG=../lib/db.a $(CRSP_ROOT)/acclib/crsplib.a -lm

default : db.a instibes instann instqtr instseg instcrsp

all: sweep db.a instibes instann instqtr instseg instcrsp

sweep:
/bin/rm instibes instann instqtr instseg instcrsp db.o compustat.o ibes.o crsp.o db.a
clean:
/bin/rm *~ #*
instibes : ibes.c db.a
$(CC) ibes.c -o instibes $(IFLAG) -DINSTIBES $(LFLAG)
instann : compustat.c db.a
$(CC) compustat.c -o instann $(IFLAG) -DVERBOSE -DINSTANN $(LFLAG)
instqtr : compustat.c db.a
$(CC) compustat.c -o instqtr $(IFLAG) -DVERBOSE -DINSTQTR $(LFLAG)
instseg : compustat.c db.a
```

```
$(CC) compustat.c -o instseg $(IFLAG) -DVERBOSE -DINSTSEG $(LFLAG)
instcrsp : crsp.c db.a
$(CC) -DINSTCRSP crsp.c -o instcrsp $(IFLAG) $(LFLAG)
db.o : db.c $(I)db.h $(I)db_local.h
$(CC) -c db.c $(IFLAG)
compustat.o : compustat.c $(I)db.h $(I)db_local.h
$(CC) -c compustat.c $(IFLAG)
ibes.o : ibes.c $(I)db.h $(I)db_local.h
$(CC) -c ibes.c -DMERGED $(IFLAG)
crsp.o : crsp.c $(I)db.h $(I)db_local.h
$(CC) -c crsp.c $(IFLAG)
db.a : db.o crsp.o compustat.o ibes.o
$(AR) -s -r -u -c -v db.a db.o compustat.o ibes.o crsp.o; $(RANLIB) db.a
```

Figure 45: Secondary makefile for compiling utility programs

```
CC=/opt/SUNWspro/bin/cc
CRSP_ROOT=/db/crsp/

I=include/
IFLAG=-DUNIX=1 -Iinclude/ -I$(CRSP_ROOT)/include
LFLAG=lib/db.a $(CRSP_ROOT)/acclib/crsplib.a -lm

default: crsp ibes compustat

all : sweep crsp ibes compustat

sweep:
/bin/rm indices manym manyd portm portd eventm eventd riskd info crspfind mpt detail consensus annual qtr segment
clean:
/bin/rm *~ #*
crsp : indices manym manyd portm portd eventm eventd riskd info crspfind mpt

ibes : detail consensus

compustat : annual qtr segment

detail : detail.c
$(CC) detail.c -o detail $(IFLAG) $(LFLAG)
consensus : consensus.c
$(CC) consensus.c -o consensus $(IFLAG) $(LFLAG)
annual : annual.c
$(CC) annual.c -o annual $(IFLAG) $(LFLAG)
qtr : qtr.c
$(CC) qtr.c -o qtr $(IFLAG) $(LFLAG)
segment : segment.c
$(CC) segment.c -o segment $(IFLAG) $(LFLAG)
eventd: eventd.c
$(CC) eventd.c -o eventd $(IFLAG) $(LFLAG)
manyd: manyd.c
$(CC) manyd.c -o manyd $(IFLAG) $(LFLAG)
portd: portd.c
$(CC) portd.c -o portd $(IFLAG) $(LFLAG)
riskd: riskd.c
```

```
$(CC) riskd.c -o riskd $(IFLAG) $(LFLAG)
eventm: eventm.c
$(CC) eventm.c -o eventm $(IFLAG) $(LFLAG)
manym: manym.c
$(CC) manym.c -o manym $(IFLAG) $(LFLAG)
portm: portm.c
$(CC) portm.c -o portm $(IFLAG) $(LFLAG)
info : info.c
$(CC) info.c -o info $(IFLAG) $(LFLAG)
crspfind : crspfind.c
$(CC) crspfind.c -o crspfind $(IFLAG) $(LFLAG)
indices : indices.c
$(CC) indices.c -o indices $(IFLAG) $(LFLAG)
mpt : mpt.c
$(CC) mpt.c -o mpt $(IFLAG) $(LFLAG)
```

Figure 46: Secondary makefile for compiling sample programs

```
CC=/opt/SUNWspro/bin/cc
CRSP_ROOT=/db/crsp

I=../include/
IFLAG=-DUNIX=1 -I../include/ -I$(CRSP_ROOT)/include
LFLAG=../lib/db.a $(CRSP_ROOT)/acclib/crsplib.a -lm

all: mom splits merged industry

industry: industry.c
$(CC) industry.c -o industry $(IFLAG) $(LFLAG)
merged: merged.c
$(CC) merged.c -o merged $(IFLAG) $(LFLAG)
splits: splits.c
$(CC) splits.c -o splits $(IFLAG) $(LFLAG)
mom: mom.c
$(CC) mom.c -o mom $(IFLAG) $(LFLAG)
```

H Installation programs for SAS

These SAS programs can be used to convert character data files into permanent, indexed SAS data sets.

CRSP Stocks Daily

Figure 47: SAS program to install CRSP daily files

```
libname crspdata '/db/crsp/data' partsize=2G;
libname crsplist '/db/crsp/data/list';
libname cal '/db/crsp/data/cal';
libname event '/db/crsp/data/event';

filename chdata1 pipe 'zcat /db/crsp/data/daily.1';
filename chdata2 pipe 'zcat /db/crsp/data/daily.2';
filename chcal '/db/crsp/data/cal.daily';

proc datasource filetype=crspdcs infile=(chcal chdata1 chdata2)
out=crspdata.daily outby=crsplist.daily outevent=event.daily;
keep prc ret vol askhi bidlo;

proc datasets library=crspdata;
  modify daily;
  index create permno;
  index create cusip;
  index create date;

proc datasets library=event;
  modify daily;
  index create permno;
  index create cusip;
  index create ticker;
  index create siccd;
  index create ncusip;
  index create event;

proc datasets library=crsplist;
  modify daily;
  index create permno;
  index create cusip;
  index create st_date;
  index create end_date;

proc contents data=crspdata.daily;
proc contents data=event.daily;

proc datasource filetype=crspdci infile=chcal
out=cal.daily;
```

Figure 48: SAS program to install CRSP monthly files

```
libname crspdata '/db/crsp/data' partsize=2G;
libname crsplist '/db/crsp/data/list';
libname cal '/db/crsp/data/cal';
libname event '/db/crsp/data/event';

filename chdata pipe 'zcat /db/crsp/data/monthly.Z';
filename chcal '/db/crsp/data/cal.monthly';

proc datasource filetype=crspmcs infile=(chcal chdata)
out=crspdata.monthly outby=crsplist.monthly outevent=event.monthly;

proc datasets library=crspdata;
  modify monthly;
    index create permno;
    index create cusip;
    index create date;
proc datasets library=event;
  modify monthly;
  index create permno;
  index create cusip;
  index create ticker;
  index create siccd;
  index create ncusip;
  index create event;

proc datasets library=crsplist;
  modify monthly;
  index create permno;
  index create cusip;
  index create st_date;
  index create end_date;

proc datasource filetype=crspmca infile=chdata
out=crspdata.yearly outby=crsplist.yearly;

data crspdata.yearly;
  set crspdata.yearly;
  format date 4.;
  date=year(date)+24;
  if capn>0;

proc datasets library=crspdata;
  modify yearly;
    index create permno;
    index create cusip;
    index create date;
    index create capn;

proc datasets library=crsplist;
  modify yearly;
  index create permno;
```

```
  index create cusip;
  index create date;

proc contents data=crspdata.monthly;
proc contents data=crspdata.annual;
proc contents data=event.monthly;

proc datasource filetype=crspmci infile=chcal
out=cal.monthly;
```

Compustat Industrial Annual

Figure 49: SAS program to install Compustat annual files

```
libname cstdata '/db/compustat/data' partsize=2G;
libname cstlist '/db/compustat/data/list';

%macro install(_dname,_chfile);
proc datasource filetype=csauc infile=&_chfile
out=cstdata.&_dname outby=cstlist.&_dname ascii;

proc contents data=cstdata.&_dname;
proc contents data=cstlist.&_dname;

proc datasets library=cstdata;
  modify &_dname;
  index create cusip=(cnum cic);
  index create dnum;
  index create date;
  index create ucode;

proc datasets library=cstlist;
  modify &_dname;
  index create cusip=(cnum cic);
%mend;

filename chfile '/db/compustat/data/annual' lrecl=8332 recfm=f;
%install(annual,chfile);
filename chfile '/db/compustat/data/annbak' lrecl=8332 recfm=f;
%install(annbak,chfile);
filename chfile '/db/compustat/data/annway' lrecl=8332 recfm=f;
%install(annway,chfile);

run;
```

Compustat Industrial Quarterly

Figure 50: SAS program to install Compustat quarterly files

```
libname sasdata '/db/compustat/data' partsize=2G;
libname saslist '/db/compustat/data/list';

%macro install(_dname,_chfile);
proc datasource filetype=cs48quc infile=&_chfile
out=sasdata.&_dname outby=saslist.&_dname ascii;

data sasdata.&_dname;
  set sasdata.&_dname;
  format reportdt date9.;
  reportdt=datejul(reportdt);

proc contents data=sasdata.&_dname;
proc contents data=saslist.&_dname;

proc datasets library=sasdata;
  modify &_dname;
  index create cusip=(cnum cic);
  index create dnum;
  index create date;
  index create ucode;

proc datasets library=saslist;
  modify &_dname;
  index create cusip=(cnum cic);
%mend;

filename chfile '/db/compustat/data/quarterly';
%install(qtr,chfile);
```

Compustat Industrial Segments

Figure 51: SAS program to install Compustat segment files

```
libname sasdata '/db/compustat/data' partsize=2G;

%macro install(_dname,_chfile);
data sasdata.&_dname;
infile &_chfile lrecl=774 pad;
  input dnum 1-4 cnum $ 5-10  cic $ 11-13 stk 14 smbl $ 15-22  file 23-24
        zlist 25-26 @55 coname $ 55-82  ssrce 83-84 fyr 85-86 year 87-88
        fundf 89-94
        sucode 95 sid 96-97 segn 98-99 sname $ 100-127 pname1 $ 128-147
        pname2 $ 148-167 pname3 $ 168-187 pname4 $ 188-207 cname1 $ 208-223
        cname2 $ 224-239
        cname3 $ 240-255 cname4 $ 256-271 cname5 $ 272-287  cname6 $ 288-303
        ssic1 304-307 ssic2 308-311 psic1 312-315 psic2 316-319 psic3 320-323
        psic4 324-327
        @428  sales 10.3  opprofit 10.3  deprec 10.3
```

```
  capex 10.3  identass 10.3  eqincome 10.3
  eqinvest 10.3  nworkers 10.3  backlog 10.3
  rDcust 10.3  rdcomp 10.3  dgovrev 10.3
  fgovrev 10.3  csale1 10.3  csale2 10.3 csale3 10.3
  csale4 10.3  psale1 10.3  psale2 10.3
  psale3 10.3  psale4 10.3
  @735 foot1 $ 735-736 foot2 $ 737-738 foot3 $ 739-740 foot4 $ 741-742
       foot5 $ 743-744 foot6 $ 745-746 foot7 $ 747-748 foot8 $ 749-750
       foot9 $ 751-752 foot10 $ 753-754
    foot11 $ 755-756 foot12 $ 757-758 foot13 $ 759-760 foot4 $ 761-762
    foot15 $ 763-764 foot16 $ 765-766 foot17 $ 767-768 foot8 $ 769-770
    foot19 $ 771-772 foot20 $ 773-774;

if sales = -.001 then sales=.;
if opprofit = -.001 then opprofit=.;
if deprec = -.001 then deprec=.;
if capex = -.001 then capex=.;
if identass = -.001 then identass=.;
if eqincome = -.001 then eqincome=.;
if eqinvest = -.001 then eqinvest=.;
if nworkers = -.001 then nworkers=.;
if backlog = -.001 then backlog=.;
if rdcust = -.001 then rdcust=.;
if rdcomp = -.001 then rdcomp=.;
if dgovrev = -.001 then dgovrev=.;
if fgovrev = -.001 then fgovrev=.;
if csale1 = -.001 then csale1=.;
if csale2 = -.001 then csale2=.;
if csale3 = -.001 then csale3=.;
if csale4 = -.001 then csale4=.;
if psale1 = -.001 then psale1=.;
if psale2 = -.001 then psale2=.;
if psale3 = -.001 then psale3=.;
if psale4 = -.001 then psale4=.;

if sales = -.004 then sales=.;
if opprofit = -.004 then opprofit=.;
if deprec = -.004 then deprec=.;
if capex = -.004 then capex=.;
if identass = -.004 then identass=.;
if eqincome = -.004 then eqincome=.;
if eqinvest = -.004 then eqinvest=.;
if nworkers = -.004 then nworkers=.;
if backlog = -.004 then backlog=.;
if rdcust = -.004 then rdcust=.;
if rdcomp = -.004 then rdcomp=.;
if dgovrev = -.004 then dgovrev=.;
if fgovrev = -.004 then fgovrev=.;
if csale1 = -.004 then csale1=.;
if csale2 = -.004 then csale2=.;
if csale3 = -.004 then csale3=.;
if csale4 = -.004 then csale4=.;
if psale1 = -.004 then psale1=.;
if psale2 = -.004 then psale2=.;
if psale3 = -.004 then psale3=.;
if psale4 = -.004 then psale4=.;

if sales = -.008 then sales=.;
```

```
      if opprofit = -.008 then opprofit=.;
      if deprec = -.008 then deprec=.;
      if capex = -.008 then capex=.;
      if identass = -.008 then identass=.;
      if eqincome = -.008 then eqincome=.;
      if eqinvest = -.008 then eqinvest=.;
      if nworkers = -.008 then nworkers=.;
      if backlog = -.008 then backlog=.;
      if rdcust = -.008 then rdcust=.;
      if rdcomp = -.008 then rdcomp=.;
      if dgovrev = -.008 then dgovrev=.;
      if fgovrev = -.008 then fgovrev=.;
      if csale1 = -.008 then csale1=.;
      if csale2 = -.008 then csale2=.;
      if csale3 = -.008 then csale3=.;
      if csale4 = -.008 then csale4=.;
      if psale1 = -.008 then psale1=.;
      if psale2 = -.008 then psale2=.;
      if psale3 = -.008 then psale3=.;
      if psale4 = -.008 then psale4=.;
proc datasets library=sasdata;
  modify &_dname;
  index create cusip=(cnum cic);
%mend;

filename chfile '/db/compustat/data/segment' lrecl=774 recfm=f;
%install(segments,chfile);
```
