BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-91-M15

Automating Multi-Locus Viability Analyses for Human Linkage with "Emilie"

by

James M. Meyers

Automating Multi-Locus Viability Analyses for Human Linkage with "Emilie"

James M. Meyers

Submitted in partial fulfillment of the requirements for the
Degree of Master of Science
in the Department of Computer Science at
Brown University

July, 1991

This submission by James M. Meyers is accepted in its present form by the Department of Computer Science in partial fulfillment of the requirements for the Degree of Master of Science.

Date July 22, 1991

Daniel Lopresti
Advisor

Automating Multi-Locus Viability Analyses for Human Linkage with "Emilie"[1]

James M. Meyers

Masters Project Report

July 21, 1991

1.0 Introduction

The decision to map and sequence the entire human genome was largely motivated by the desire to locate and determine the role of genetic factors in diseases such as hemophilia, cystic fibrosis, multiple sclerosis, hypercholesteremia, and diabetes. These diseases, as well as thousands of others, affect the *viability* of the afflicted individuals. Viability refers to the probability that a given individual survives from fertilization to reproductive age.

Because genetic analyses are, for the most part, statistical analyses, the accuracy of any given analysis depends heavily upon the validity of the sample data. *Recombination rate*, a key parameter in the mathematical models used to map chromosomes, is determined through scientific observation. Viability differences, however, can bias the observed recombination rates, thus introducing errors into the gene maps. The magnitude of these errors is not presently known, but preliminary studies [Broo91] suggest that the consideration of viability factors could significantly improve the accuracy of the current methods for determining recombination rates.

The purpose of this project is to produce a software system capable of performing viability analyses on human family data compiled in the CEPH (Centre d'Etude du Polymorphisme Humain) database format. It is hoped that these analyses will provide information necessary to develop the new statistical models needed to augment current gene mapping procedures.

1.1 Introduction to Population Genetics

Population genetics is the study of Mendel's laws of inheritance and other genetic principles as they apply to entire populations of organisms. A brief discussion of the most basic of these principles follows. For a complete introduction to population genetics, refer to [Hart88].

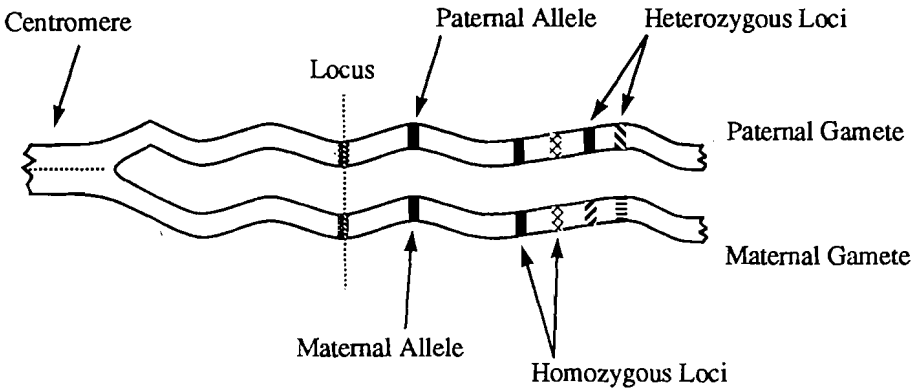
1. Emilie Bordeleau, mother of ten, is the heroine of the French novel, *Les Filles de Caleb*, by Arlette Cousture. The database that this project processes is a database of large nuclear families, managed by French researchers. Thus, the named seemed appropriate.

The science of genetics attempts to understand the properties of the genetic material, deoxyribonucleic acid (DNA). A *gene*, which corresponds to a specific sequence of nucleotides along a molecule of DNA, is the fundamental physical and functional unit of heredity. In addition to determining traits such as eye color, skin color, height, and weight, genes determine the makeup of proteins such as hemoglobin and insulin. Genes can exist in different forms called *alleles*. For example, the genes which code for the hemoglobin in red blood cells exist in multiple alleles. The normal allele produces normal hemoglobin, while the sickle-cell allele is responsible for the production of the abnormal hemoglobin associated with sickle cell anemia.

Within a cell, genes are arranged in linear order along microscopic threadlike bodies called *chromosomes*. The position of a gene along a chromosome is called a *locus*. In humans, as well as in most higher organisms, each cell contains two copies of each type of chromosome, one inherited from the mother and one inherited from the father. Male sperm and female eggs each contain only one of these copies of genetic information. These single copies are called *gametes*. At any locus, every normal individual contains exactly two alleles; one at each corresponding position in the maternal and paternal gamete. If the two alleles at a locus are chemically identical, the individual is *homozygous* at the locus in question. If the two alleles are different, the individual is *heterozygous* at that locus.

The *centromere* of a chromosome is, loosely speaking, the site where the two gametes are attached. Cells which contain only one chromosome set, such as gametes, are referred to as *haploid* cells.

FIGURE 1. Conceptual View of a Chromosome



The genetic constitution of an individual is called its *genotype*. Thus, genotype refers to the specific alleles contained in an individual at all loci that affect the trait in question. If, for example, a trait is influenced by two diallelic genes, then there are nine possible genotypes:

```
AA BB    AA Bb    AA bb
Aa BB    Aa Bb    Aa bb
aa BB    aa Bb    aa bb
```

where A and a refer to the alleles of the first gene and B and b refer to the alleles of the second gene. The physical expression of a genotype is called the *phenotype*. For example, if eye color is determined by the genes at two loci, an individual's genotype for eye color might be AA Bb. If the AA Bb genotype produces an individual with green eyes, we say that the phenotype produced by the AA Bb genotype is "green eyes". The distinction between genotype and phenotype is particularly important in cases in which viability differences occur. *A genotype reduces the viability of an individual by producing a compromised phenotype.*

When gametes are produced, the two copies of genetic information in a chromosome are separated. During this separation, genes from different loci sometimes exchange positions. When this happens, we say that *recombination* has occurred. The frequency at which recombination will occur between two genes is the *recombination rate*. Due to the physical nature of chromosomes, the recombination rate for two genes is a function of three distances; the distance between the two genes and the distances between each gene and the centromere. This is why recombination rates are useful in mapping chromosomes. *Linkage* is the tendency of genes on the same chromosome to segregate together.

The arrangement of alleles among the gametes in a given cross determines the *linkage phase* of the cross. For example, if an Aa Bb individual was produced by AA BB and aa bb parents we say that the individual's phase is in *coupling*. Conversely, if an Aa Bb individual was produced by Aa Bb and Aa Bb parents we would say that the individual's phase is in *repulsion*. Linkage phase is used to specify which parent contributed a specific allele to a given individual.

To compare different genes and different populations, it is necessary to have some convenient quantitative measure of genetic variation. Genetic variation can be quantified using the concept of *allele frequency*. The allele frequency of a prescribed allele among a group of individuals is the proportion of all alleles at the locus that are the prescribed type. A *polymorphic* gene is a gene for which the most common allele has a frequency of less than 0.95.

1.2 The Problem

The major goal of this project is to produce a software system which will aid genetic researchers in understanding the effect that viability differences have on observed recombination rates. It is hoped that a better understanding of viability-recombination interactions will facilitate the improvement of current recombination rate determination methods.

In the simplest case, recombination rate is determined in the following manner.

With a recombination rate r between the genes A and B, the double heterozygote:

$$\frac{A\ B}{a\ b}$$

produce the following chromosomes in these proportions:

Parental Chromosomes	$\frac{A\ B}{}$	$(1-r)/2$
	$\frac{a\ b}{}$	$(1-r)/2$
Recombinant Chromosomes	$\frac{A\ b}{}$	$(r)/2$
	$\frac{a\ B}{}$	$(r)/2$

The recombination rate r is calculated as the observed number of recombinants out of the total number of chromosomes produced. Suppose that, out of 100 chromosomes, 25 of each the above genotypes were produced. The recombination rate would then be:

$$r = \frac{(25 + 25)}{100} = 0.5$$

(EQ 1)

This sort of calculation is valid only when recombination is the sole factor affecting the frequency of the recombinant chromosomes. Since the value of r is based on the observed number of recombinant chromosomes, its accuracy can be

affected by survival differences among the genotypes. Suppose, in the above example, that all Ab chromosomes were lethal. The recombination rate would then be calculated as:

$$r = \frac{(0+25)}{100} = 0.25$$

(EQ 2)

Thus, the recombination rate would be severely underestimated. In practice, our hypothetical lethals would be easily identified and the recombination rate adjusted accordingly. This is a drastic example constructed to illustrate the dynamics of recombination rate and viability. In most cases, the effect of viability differences are not so drastic and not easily detectable. Standard methods for computing recombination rate account for many complexities of real data, however, they do not consider viability differences.

1.3 The CEPH database

The Centre d'Etude du Polymorphisme Humain (CEPH) was founded in 1983 as a nonprofit research institute to aid the scientific community in the study of human DNA. CEPH provides access to DNA samples from a panel of reference families for the determination of genotypes for various DNA polymorphisms. This information can be used to aid in the construction of the genetic map of the human genome.

The key premise of the CEPH collaboration is that the human genetic map will be efficiently achieved by collaborative research on DNA from the same set of families. CEPH provides collaborating investigators with cellular DNA samples from each member of a reference panel of 61 large nuclear families (each family has at least 6 offspring). The researchers independently analyze these samples for segregation of genetic markers and contribute the genotypes to CEPH for addition to a database. The database is then returned to them for linkage analysis and map construction. There are at least 63 laboratories in 6 countries that collaborate with CEPH.

2.0 The Project

The purpose of this project is to produce a software system capable of performing both haplotype and genotype viability analyses on human family data compiled in the CEPH database format. The algorithms used for these analyses are based on the genetic models being developed by Dr. Lisa D. Brooks, Assistant Professor of Biology, at Brown University. These models are the first of their kind, and utilize previously unused data to estimate viabilities independently of the recombination rate. It is hoped that these analyses will provide information necessary to develop new statistical models to augment the current gene mapping procedures.

2.1 Goals

The primary goal of Emilie is to provide utilities to extract the viability information required to determine the extent of the interaction between viability and recombination rate in humans. Of secondary consideration, is the production of a software system suitable for release to others in the research community. This system should be fairly polished and provide users with an intuitive user-interface and useful output formats. The system should also be easily modifiable by researchers as needed.

2.2 Design Environment

From its conception, Emilie was intended to be used (and modified) by biologist and geneticists. It was the opinion of Lisa Brooks and myself that the most common hardware platform for these users was the IBM PC. Furthermore, the utilities

provided with the CEPH database and most of the available linkage analysis software is written in Borland's Turbo Pascal. Thus, Emilie was developed on an IBM PC using Turbo Pascal version 6.0.

3.0 Haplotype Analyzer

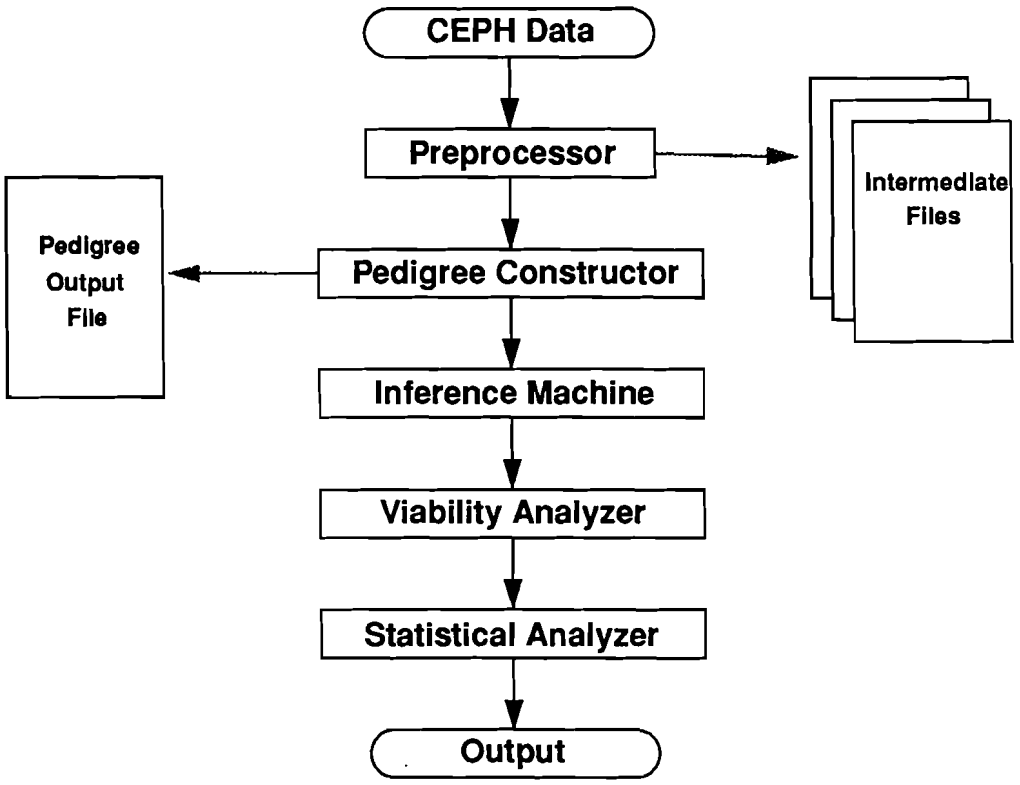
The first model implemented for viability analyses was a 2-gene, 2-allele haplotype analyzer. The term *haplotype*, derived from *haploid genotype*, refers to the alleles of a single gamete. Thus, as the name would imply, the haplotype analyzer performs analyses based on the gametes of the parents.

The haplotype analyzer was intended to be the prototype for an n-gene, n-allele haplotype analyzer, but that design has since been abandoned for a different model; the genotype analyzer. Nevertheless, the 2-gene, 2-allele haplotype analyzer has been successfully used to support the hypothesis that viability differences have introduced errors into human recombination rate calculations. Specifically, haplotype analyses of CEPH families found a 36% underestimate of recombination rate for genes 5 and 9 on chromosome 11p [Broo91]. The genotype analyzer will be discussed in detail later.

The haplotype analyzer has six main parts:

- Look-up Tables
- Preprocessor
- Pedigree Constructor
- Inference Machine
- Viability Analyzer
- Statistical Analyzer

FIGURE 2. Data Flow



Each of these parts are described below in detail.

3.1 Look-up Tables

The look-up tables in the prototype evolved as the project progressed. There are currently three tables.

3.1.1 Phase Known Table

The CEPH database does not include linkage phase information for gene pairs. There are, however, some instances where the phase of particular gene pair may be inferred. The Phase Known table contains an entry for every parent1-parent2-child combination that reveals missing phase information. Thus, the Phase Known table is used to infer the phase of a child.

3.1.2 Viability Data Table

The Viability table contains the actual viability information for an individual. The Viability table contains an entry for every possible parent1-parent2-child combination, thus, it is very large.

3.1.3 Unknown Parent Table

Occasionally, an individual will be untyped for a given gene. However, if this individual has a spouse and a child who *were* typed, some viability information may still be extracted. The Unknown Parent table contains entries for each of these combinations. The data format of this table is identical to that of the Viability table, as the Unknown Parent table is used in place of the Viability table when necessary.

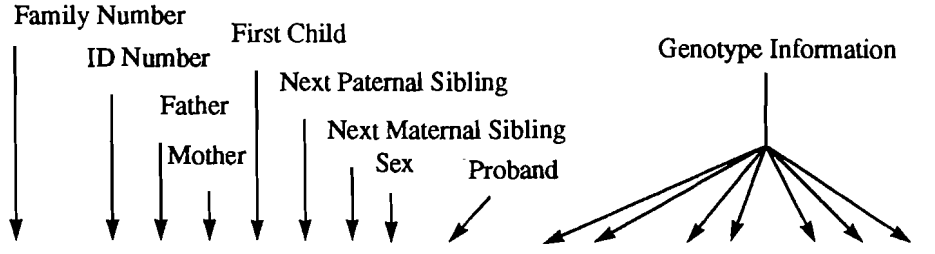
3.2 Preprocessor

The preprocessor accepts CEPH family information as input and performs the following operations.

3.2.1 Stripping

The output files produced by the CEPH utilities contain superfluous (for our purposes) information. Since these CEPH output files serve as input files for Emilie, the unneeded information is stripped.

FIGURE 3. Sample CEPH Output File (one family)



1327	1	12	9	3	0	0	1	0	0	0	1	1	0	0	1
1327	2	10	11	3	0	0	2	0	0	0	1	1	1	1	1
1327	3	1	2	0	4	4	2	0	0	0	1	1	0	0	1
1327	4	1	2	0	5	5	1	0	0	0	1	1	1	1	1
1327	5	1	2	0	6	6	2	0	0	0	1	1	0	0	1
1327	6	1	2	0	7	7	2	0	0	0	1	1	1	1	1
1327	7	1	2	0	8	8	1	0	0	0	1	1	1	0	0
1327	8	1	2	0	0	0	2	0	0	0	1	1	1	1	1
1327	9	0	0	1	0	0	2	0	0	0	1	1	0	0	1
1327	10	0	0	2	0	0	1	0	0	0	1	1	0	0	1
1327	11	0	0	2	0	0	2	0	0	0	1	1	1	1	1
1327	12	0	0	1	0	0	1	0	0	0	0	0	0	0	0

3.2.2 Parsing

CEPH output files do not contain delimiters. As a result, most of the preprocessor's code is devoted to parsing the data fields of the CEPH output.

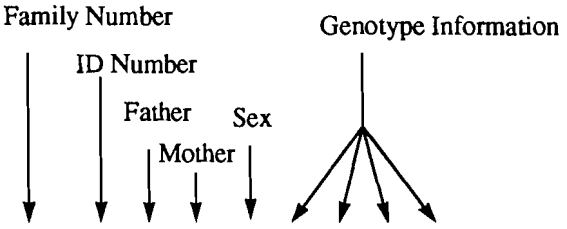
3.2.3 Data Conversion

CEPH output files express genotypes in factor-union notation. Since factor-union notation is neither convenient nor intuitive, Emilie stores genes in genotypic notation. There is not, however, a standard code for each of the three genotypes as the factor-union encoding is derived from gel electrophoresis data. Thus, Emilie's haplotype analyzer must prompt the user for each of the factor-union codes before any conversions can be made.

3.2.4 File Output

The stripped version of the CEPH output file which the preprocessor creates is in a form which is compatible with several popular linkage analysis programs. Thus, the preprocessor outputs these intermediate files for use by external linkage analysis programs.

FIGURE 4. Sample Intermediate Genotype File




```
1327   1   12  9   1   0   0   1   2
1327   2   10  11  2   0   0   1   2
1327   3   1   2   2   0   0   1   2
1327   4   1   2   1   0   0   1   2
1327   5   1   2   2   0   0   1   2
1327   6   1   2   2   0   0   1   2
1327   7   1   2   1   0   0   1   2
1327   8   1   2   2   0   0   1   2
1327   9   0   0   2   0   0   1   2
1327   10  0   0   1   0   0   1   2
1327   11  0   0   2   0   0   1   2
1327   12  0   0   1   0   0   0   0
```

3.3 Pedigree Constructor

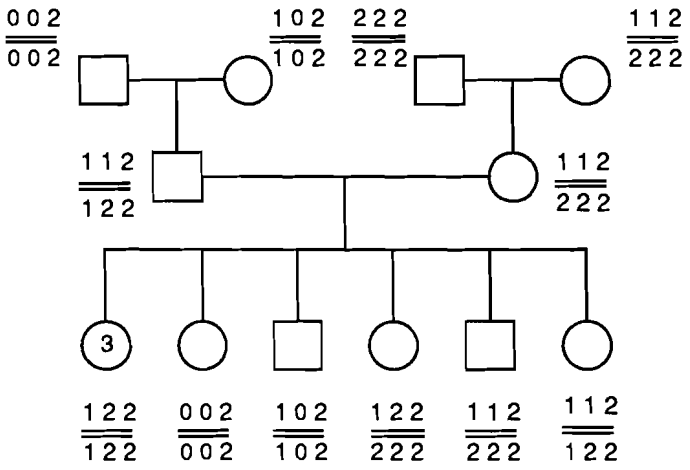
The pedigree constructor analyzes each CEPH family to establish the relationships among family members. It accomplishes this by performing the following operations on each individual in the family:

- Determines the "level" of a family member (whether the individual is a grandparent, parent, or child)
- Determines which family members have mated
- Determines which family members share common parents

The information compiled by the pedigree constructor is stored in a fashion convenient for access by other modules. Both the inference machine and the viability analyzer rely heavily upon this information.

The pedigree constructor also outputs an intermediate file of pedigree information. While its output is useful for visualizing the relationships among family members, the pedigree constructor does not output the pedigree in the traditional format - this is a possible extension for a later version.

FIGURE 5. Traditional Pedigree Format



3.4 Inference Machine

The gene typing for CEPH families is not always complete. Sometimes, however, missing information for an individual can be extrapolated from family data by considering the typing of his/her parents, grandparents, and siblings. The inference machine uses information from the pedigree constructor to make inferences about missing data in the a particular family member. The haplotype analyzer makes two types of inferences:

3.4.1 Both Parents are Homozygous

When an individual has not been typed for a given gene, the type may be accurately inferred if both of his/her parents are homozygous for that gene. The inference machine processes all families for the presence of these individuals and completes the typing when appropriate.

3.4.2 One or Both Parents are Untyped

When one parent is untyped for a given gene, the genotypes of his/her spouse and children often supply enough information to complete the typing information. The inference machine scans all families for these individuals and completes the typing when appropriate.

3.5 Viability Analyzer

The viability analyzer uses a lookup table to determine the "viability value" for a given individual. A viability value was computed, by hand, for all possible parent-offspring combinations. This value represents the probability that the parents would produce the offspring genotype. A running total is kept for each family and for the entire set of families processed.

3.5.1 Table Look-ups

The viability data table is a 4 dimensional table of polynomial expressions. Fortunately, the number of unique polynomial expression is quite low, so each expression can be stored as an index into a polynomial expression table. In the 2-gene implementation, most of the polynomials are real number constants. Thus, the table was implemented with real number entries; negative sentinel values serve as keys into the polynomial table.

The dimensions of the table are determined by [the number of possible genotypes (the genotype of parent 1)] x [the number of possible genotypes (the genotype of parent 2)] x [the number of possible genotypes (the genotype of the child in

question)] x [number of segregations (a function of the number of genes being analyzed)]. In the 2-gene, 2- allele case this is [11]x[11]x[11]x[6], or, 7986 table entries.

It is interesting to note that very few of these entries contain expressions other than "0". If we construct a two dimensional matrix indexed by the first two indices of the viability table we can visualize the distribution of the entries.

TABLE 1. Distribution of Viability Table Entries

	1	2	3	4	5	6	7	8	9	10	11
1	0	2	2	4	4	4	0	0	0	2	2
2		3	4	8	8	8	2	2	2	4	5
3			3	8	8	8	2	2	2	5	4
4				11	11	11	4	4	4	8	8
5					11	11	4	4	4	8	8
6						11	4	4	4	8	8
7							0	0	0	2	2
8								0	0	2	2
9									0	2	2
10										3	4
11											3

In fact, out of [11] x [11] x[11] (1331) possible segregations, only 280 can actually occur. This is because, for any given mating, the possible genotype of the child is limited by the genotypes of the parents. Thus, the matrix is quite sparse. Should the haplotype model ever be expanded, or re-coded, a sparse matrix implementation should be considered.

3.5.2 Polynomial Operations

As was mentioned earlier, the number of unique polynomial expressions in the table is small.

TABLE 2. Polynomial Expressions in two gene Analysis

$$v = 0 \quad \text{(EQ 3)}$$

$$v = 0.5 \quad \text{(EQ 4)}$$

$$v - 1.0 \quad \text{(EQ 5)}$$

$$v = r \quad \text{(EQ 6)}$$

$$v = 1 - r \quad \text{(EQ 7)}$$

$$v = x = \frac{r^2}{(1-r)^2 + r} \quad \text{(EQ 8)}$$

$$v = 1 - x = \frac{(1-r)^2}{(1-r)^2 + r^2} \quad \text{(EQ 9)}$$

$$v = y = \frac{0.5 + x}{2} = 0.25 + \frac{x}{2} \quad \text{(EQ 10)}$$

$$v = 1 - y = \frac{0.5 + 1 - x}{2} = 0.25 + \frac{(1 - x)}{2} \qquad \text{(EQ 11)}$$

r = recombination rate

Note that equations 6 and 7 are functions of *r* and that equations 8 and 9 are functions of the expression in equations 6 and 7. For 3+ gene analyses, these polynomials continue to fold and become very complicated.

3.6 Statistical Analyzer

Once the viability data for a group of families are compiled, they must be statistically analyzed to determine whether there are viability differences among the haplotypes. The haplotype analyzer performs a G-test to determine whether the observed segregation of haplotypes agrees with what would be expected in Mendelian segregation. Significant differences indicate viability differences among haplotypes.

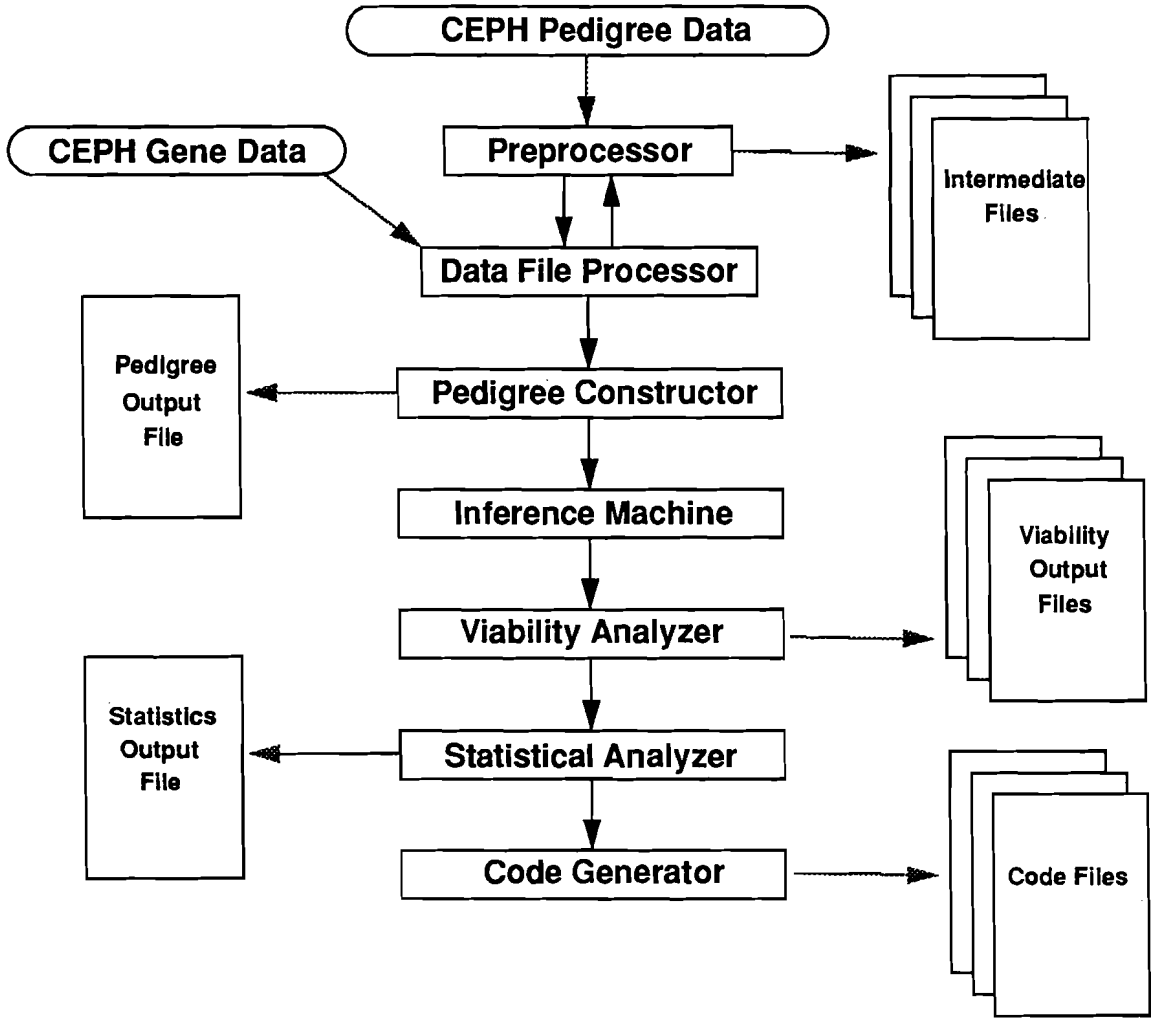
4.0 Genotype Analyzer

The genotype analyzer performs analyses based on the alleles at given locus. Recall that the haplotype analyzer based its analyses on parental gametes.

The genotype analyzer has eight main parts:

- Preprocessor
- Data File Processor
- Pedigree Constructor
- Inference Machine
- Viability Analyzer
- Statistical Analyzer
- Code Generator
- Event-Driven User Interface

FIGURE 6. Data Flow in Genotype Analyzer



The preprocessor, pedigree constructor, inference machine, and statistical analyzer are almost identical to those in the haplotype analyzer. All look-up tables have been omitted; generalized algorithms have been developed to replace them. The new additions, the data file processor, the code generator and the user interface will be described below in detail. Furthermore, the viability analyzer (which is completely different than that found in the haplotype analyzer) will be described.

4.1 The Data File Processor

When the CEPH database software produces a pedigree file with the requested genes, it also produces a data file with information about that pedigree file. The data file contains, among other things, the number of genes requested, and the names and factor union codes of those genes. The data file processor parses the CEPH data file and extracts this information. Thus, the user no longer need enter any of this information by hand.

4.2 Genotype Viability Analyzer

The genotype analyzer constructs likelihood equations for CEPH families to estimate viabilities and r values via maximum likelihood methods. From these equations it is possible to estimate the viabilities of a genotypes. In contrast to the haplotype analyzer (which looked up viability data in a table), the genotype analyzer calculates all of its viability data "on the fly". The table-driven approach of the haplotype was faster than the genotype analyzer, but lacked the generality of the genotype analyzer. The algorithms used in Emilie's genotype analyzer will work for any number of alleles and, with a little work, they can be extended to work for any number of genes. Furthermore, for n alleles, the size of the haplotype viability look-up table grows ~$O(n^4)$. Building these tables by hand would be impossible for large n. Even if the tables were built automatically, their size would require that they reside in secondary storage; causing the analyses to become I/O bound.

The output of the viability analyzer is now an equation with as many as10 unknowns in the 2-gene, 2-allele case. Since the genotype analyzer supports analyses of two genes with an arbitrary number of alleles, the number of unknowns in the viability equation is $O(n^2)$ for an n-allele analysis. The unknowns in the equation are the viability values which we wish to estimate. Thus, the unknowns are evaluated via a grid search or simulated annealing for maxima. It is important to note that a CEPH family contains three crosses; one for each set of grandparents and one for their children. Thus, one CEPH family will produce three likelihood equations.

4.2.1 Parameters of Likelihood Equations

For a family, the log likelihood equation is constructed from the following information.

1. Offspring genotypes (of which there are m):

$$\frac{1\ 1}{1\ 1} \quad \frac{1\ 1}{1\ 2} \quad \cdots \quad \frac{2\ 2}{2\ 2}$$

The number of offspring genotypes is a function of the number of genes under consideration and the number of alleles for each of those genes.

2. Segregation expected frequencies, f:

$$f_1 \quad f_2 \quad \cdots \quad f_m$$

The segregation expected frequency is the probability that a given genotype will be produced from the parent gametes. Thus, segregation expected frequencies expressions are all functions of the recombination rates of the genes involved.

3. Viability, w:

$$w_1 \quad w_2 \quad \cdots \quad w_m$$

Each offspring genotype has an associated viability value, w. These values are unknowns in our likelihood equations. It is, in fact, the purpose of the likelihood equations to solve for these values.

4. Overall expected frequency, p:

$$p_1 \quad p_2 \quad \cdots \quad p_m$$

where

$$p_i = \frac{f_i w_i}{\sum_{j=1}^{m} f_j w_j}$$

(EQ 12)

The overall expected frequency of an offspring genotype refers to the probability that any offspring will exhibit the genotype in question.

5. Observed number, n:

$$n_1 \quad n_2 \quad \bullet \bullet \bullet \quad n_m$$

The observed number of a genotype offspring is the number of times a given genotype appears.

4.2.2 The Likelihood Equation

Using the parameters above, we can construct a likelihood equation for a given family. The purpose for generating these equations is to solve them for the unknown viabilities, $w_i, \ldots, w_m$, and for r. There are various methods of numerical analysis with which one may determine estimates of the viability terms. Emilie currently uses a simple grid-search to find the maxima and minima of the function L. The nature of such an analysis requires that the likelihood equation be evaluated at incremental values of the w_i terms.

The likelihood equation for a family is

$$L = \prod_{i=1}^{m} p_i^{n_i}$$

(EQ 13)

Notice that the expression for L above contains a product of an exponent. The ranges of the variables involved make evaluation of the expression intractable using standard data types. All values of p_i are probabilities and, therefore, lie in the range of 0.0 to 1.0. Since m and all of the n_i values are positive integers, the use standard floating point data types will almost always result in underflow. Thus by expanding p,

$$L = \left(\prod_{i=1}^{m} (f_i w_i)^{n_i}\right) \sum_{i=1}^{m} (f_i w_i)^{-\sum_{j=1}^{m} n_j}$$

(EQ 14)

and by taking the logarithm of both sides, we have

$$\log L = \sum_{i=1}^{m} n_i \log p_i = \sum_{i=1}^{m} n_i \log f_i + \sum_{i=1}^{m} n_i \log w_i + \left(\sum_{i=1}^{m} n_i\right) \log \sum_{i=1}^{m} f_i w_i$$

(EQ 15)

Now we can evaluate $\log L$ rather than L.

When phase is unknown, multiple frequencies may exist for each genotype. When there are l frequencies for each genotype, the expression becomes

$$\log L = \sum_{i=1}^{m} n_i \log w_i + \log \sum_{i=1}^{l} \left(\left(\prod_{j=1}^{m} (f_{i,j})^{n_j} \right) \sum_{k=1}^{m} f_{i,k} w_i \right)^{-\sum_{q=1}^{m} n_q}$$

(EQ 16)

Notice that we now have another product of an exponent. The terms being multiplied are similar to those in equation 13. Therefore, the expression cannot be able to be evaluated with standard data types. The solution to this problem is identical to the transformation we used for equation 13.

$$\log\log L = \log \sum_{i=1}^{m} n_i \log w_i + \log\log \sum_{i=1}^{l} \left(\left(\sum_{j=1}^{m} n_j \log f_{i,j} \right) - \left(\sum_{q=1}^{m} n_q \right) \log \sum_{k=1}^{m} f_{i,k} w_i \right)$$

(EQ 17)

4.3 Code Generator

The code generator produces Pascal code which, when compiled, evaluates the viability equations constructed by the viability analyzer. This code can be used by an external numerical analysis program. The following code evaluates one of the three viability equations for a family.

FIGURE 7. Sample Output Code

```
Family #1331
log L (phase unknown) =
N:= 0;
P1:= 0;
P2:= 0;
P3:= 0;
P4:= 0;

N:= N + ln(w12_13);

T1:= 1;
T1:= T1 * power(((rf + (1-rf))/4),1);

T2:= 0;
T2:= T2 + ((1-rf)/4) * w11_11;
T2:= T2 + (rf/4) * w11_13;
T2:= T2 + ((rf + (1-rf))/4) * w12_11;
T2:= T2 + ((rf + (1-rf))/4) * w12_13;
T2:= T2 + (rf/4) * w22_11;
T2:= T2 + ((1-rf)/4) * w22_13;

P1:= T1 * power(T2,-1);

T1:= 1;
T1:= T1 * power(((rf + (1-rf))/4),1);

T2:= 0;
T2:= T2 + (rf/4) * w11_11;
T2:= T2 + ((1-rf)/4) * w11_13;
T2:= T2 + ((rf + (1-rf))/4) * w12_11;
T2:= T2 + ((rf + (1-rf))/4) * w12_13;
T2:= T2 + ((1-rf)/4) * w22_11;
T2:= T2 + (rf/4) * w22_13;

P3:= T1 * power(T2,-1);

N:= N + ln(P1 + P2 + P3 + P4);
```

4.4 User Interface

Emilie's user interface is window-based event-driven interface which provides mouse support. Windows can be dynamically created, resized, moved and destroyed. If the user desires, the UI will automatically arrange existing windows in an optimal layout; either tiled or layered.

Dialog boxes allow the user to specify input and output files. After an analysis has been run, the user may display as many of the output files as desired.

5.0 Emilie's Future

Emilie is currently being used by Lisa Brooks at Brown University to study the effects of viability interactions on recombination rates. An enhanced version is being prepared for distribution to other researchers in October, 1991. Furthermore, a 3-gene, n-allele model is being developed for a future version. Expanding the capabilities of Emilie's analyses to 3 genes will provide researchers with more useful information than is provided by 2 gene analyses.

6.0 Haplotype Analyzer Code

The haplotype analyzer was intended to be the prototype for an n-gene, n-allele haplotype analyzer, but that design has since been abandoned for a different model; the genotype analyzer. Nevertheless, the 2-gene, 2-allele haplotype analyzer has been successfully used to support the hypothesis that viability differences have introduced errors into human recombination rate calculations.

6.1 Main Program

```
program go;

{$M 16384,150000,655360}

{
| File:          go.pas
| Compiles to:   go.exe
| Usage:         executable
|
| Author:        James Meyers
|                Brown University
|
| Start Date:    December 22, 1990
| Last Revision: February 20, 1991
|
| Intent:
}

uses global_2, {global constants, type definitions, and  procedures}
     lookup_2, {Table look-up module for analysis of two genes}
     in_out_2, {File input/output and pedigree construction module}
     viable_2, {Viablily information module for analysis of two genes}
     gtest;    {Does a G-test of whether numbers are in a 1:1 ratio}

var
  family_counter : integer;        {counts number of families processed}
  pause : char;

{-------------------------------------------------------------------------}

procedure initialize;

begin
  family_counter:=0;
```

```
  write('Pause between families? (Y/N): ');
  readln(pause);
end; {procedure initialize}

{-------------------------------------------------------------------------------}

procedure report_status;

begin
  writeln(vb_file,'Families processed --> ',family_counter);
end; {procedure report status}

{-------------------------------------------------------------------------------}

procedure clean_up;

begin
  close_files;
  deallocate_table;
end;

{-------------------------------------------------------------------------------}

begin
  initialize;
  while not eof(infile) do
    begin
      process_next_family;
      extract_viability_data;
      if upcase(pause)='Y' then
        readln;
      inc(family_counter)
    end;
  output_cumulative_vdata;
  writeln(vb_file);
  writeln(vb_file,'r value = ',r_value:4:3);
  writeln(vb_file);
  output_g_test_data(vdata_cum_totals);
  report_status;
  clean_up
end.
```

6.2 Global Types Unit

```
unit global_2;

{
| File:          global_2.pas
| Compiles to:   global_2.tpu
| Usage:         "uses global_2;"
|
| Author:        James Meyers
```

```
|                Brown University
|
| Start Date:     December 22, 1991
| Last Revision: February 20, 1991
|
| Intent:
}

{****} interface

uses crt;

const
  LINKED = true;
  NOT_LINKED = false;
  NUM_GENOTYPES = 11;                {number of 2 locus genotypes}
  NUM_SEGREGATIONS = 6;              {number of 2 locus segregations}
  MAX_NUM_CHILDREN = 11;             {maximum number of possible genotypes for
                                      any given parents}
  R = -1.0;
  ONE_MINUS_R = -2.0;
  X = -3.0;                          { (R^2) / ((1-R)^2 + R^2) }
  ONE_MINUS_X = -4.0;                { (1-R)^2 / ((1-R)^2 + R^2) }
  Y = -5.0;                          { (.5 + X) / 2 }
  ONE_MINUS_Y = -6.0;                { (.5 + (1-X)) / 2 }

  SEGREGATIONS:array[1..NUM_SEGREGATIONS] of string[4]= ('1112','1121','1222',
    '2122','1122','1221');

  GENOTYPES:array[1..NUM_GENOTYPES] of string[5]= ('1111','1112','1212',
    '1121','1122','1221','1122','1222','2121','2122','2222');

{-------------------------------------------------------------------------}

procedure error(message:string);
function x_value(r:real):real;
function y_value(r:real):real;

{-------------------------------------------------------------------------}

{****} implementation

procedure error(message:string);

begin
  writeln;
  writeln('*** ',message,' ***');
  halt(0)
end; {procedure error}

{-------------------------------------------------------------------------}

function x_value(r:real):real;
```

```
begin
  x_value:=(r*r) / ( ((1-r)*(1-r)) + (r*r) )
end; {function x_value}

{--------------------------------------------------------------------------}

function y_value(r:real):real;

begin
  y_value:=(0.5 + x_value(r)) / 2
end; {function y_value}

{--------------------------------------------------------------------------}

begin
  clrscr;
end.
```

6.3 Lookup Table Unit

```
unit lookup_2;

{
| File:          lookup_2.pas
| Compiles to:   lookup_2.tpu
| Usage:         "uses lookup_2;"
|
| Author:        James Meyers
|                Brown University
|
| Start Date:     February 1, 1991
| Last Revision: February 20, 1991
|
| Intent:        This unit defines the data structures necessary to construct
|                a table of all 2-locus segregations. A table is allocated and
|                initialized for use by the main program. This unit also
|                includes the necessary functions and procedures to access the
|                table.
}

{****} interface

uses global_2;

type
  seg_data_ptr = ^segregation_data_type;
  segregation_data_type = array[1..2] of real;

  lookup_table_type = array[1..NUM_GENOTYPES] of array[1..NUM_GENOTYPES] of
                      array[1..MAX_NUM_CHILDREN] of array[1..NUM_SEGREGATIONS]
                      of seg_data_ptr;
```

```
  delta_phase_table = array[1..NUM_GENOTYPES] of array[1..NUM_GENOTYPES] of
                      array[1..MAX_NUM_CHILDREN] of byte;

  useg_table_type = array[1..NUM_GENOTYPES] of array[1..NUM_GENOTYPES] of
                    array[1..NUM_SEGREGATIONS] of segregation_data_type;


{-------------------------------------------------------------------------}

procedure dump_the_table;
procedure deallocate_table;

{-------------------------------------------------------------------------}

var
  the_table:lookup_table_type;
  delta_phase:delta_phase_table;
  useg_table:useg_table_type;

{-------------------------------------------------------------------------}

{****} implementation

procedure fill_the_table;

var
  i,j,k,l:byte;
  t_file:text;
  hold:byte;

begin
  writeln('Initializing segregation table');
  for i:=1 to NUM_GENOTYPES do
    for j:=1 to NUM_GENOTYPES do
      for k:=1 to MAX_NUM_CHILDREN do
        for l:=1 to NUM_SEGREGATIONS do
          begin
            new(the_table[i][j][k][l]);
            the_table[i][j][k][l]^[1]:= 0.0;
            the_table[i][j][k][l]^[2]:= 0.0
          end;

  assign(t_file,'c:table.dat');
  reset(t_file);

  while not eof(t_file) do
    begin
      read(t_file,i,j,k);
      for l:=1 to NUM_SEGREGATIONS do
        begin
          read(t_file,the_table[i][j][k][l]^[1]);
          read(t_file,the_table[i][j][k][l]^[2]);
        end
    end;
```

```
  end; {procedure fill_the_table}

{-------------------------------------------------------------------------------}

procedure fill_delta_phase_table;

var
  i,j,k:byte;
  p_file:text;

begin
  for i:=1 to NUM_GENOTYPES do
    for j:=1 to NUM_GENOTYPES do
      for k:=1 to MAX_NUM_CHILDREN do
        delta_phase[i][j][k]:=0;

  assign(p_file,'c:phase.dat');
  reset(p_file);
  while not eof(p_file) do
    begin
      read(p_file,i,j,k);
      readln(p_file,delta_phase[i][j][k]);
    end
end; {procedure fill_the_table}

{-------------------------------------------------------------------------------}

procedure fill_useg_table;

var
  i,j,k:byte;
  u_file:text;

begin
  for i:=1 to NUM_GENOTYPES do
    for j:=1 to NUM_GENOTYPES do
      for k:=1 to NUM_SEGREGATIONS do
        begin
          useg_table[i][j][k][1]:=0.0;
          useg_table[i][j][k][2]:=0.0
        end;

  assign(u_file,'c:useg.dat');
  reset(u_file);
  while not eof(u_file) do
    begin
      read(u_file,i,j);
      for k:=1 to NUM_SEGREGATIONS do
        read(u_file,useg_table[i][j][k][1],useg_table[i][j][k][2]);
    end
end; {procedure fill_useg_table}

{-------------------------------------------------------------------------------}
```

```
procedure deallocate_table;

var
  i,j,k,l:byte;

begin
  for i:=1 to NUM_GENOTYPES do
    for j:=1 to NUM_GENOTYPES do
      for k:=1 to MAX_NUM_CHILDREN do
        for l:=1 to NUM_SEGREGATIONS do
          dispose(the_table[i][j][k][l])

end; {procedure deallocate_table}

{-------------------------------------------------------------------------------}

procedure dump_the_table;

var
  i,j,k,l:byte;
  t_file:text;
  hold:byte;

begin
  for i:=1 to NUM_GENOTYPES do
    for j:=1 to NUM_GENOTYPES do
      for k:=1 to MAX_NUM_CHILDREN do
        begin
          write(i:2,',',j:2,',',k:2,': ');
          for l:=1 to NUM_SEGREGATIONS do
            begin
              write(the_table[i][j][k][l]^[1]:2:1,' ');
              write(the_table[i][j][k][l]^[2]:2:1,' ; ')
            end;
          writeln;
        end
end; {procedure dump_the_table}

{-------------------------------------------------------------------------------}

begin
  fill_the_table;
  fill_delta_phase_table;
  fill_useg_table
end.
```

6.4 In_out_2.pas

```
unit in_out_2;

{
| File:          in_out_2.pas
```

```
| Compiles to:   in_out_2.tpu
| Usage:         "uses in_out_2;"
|
| Author:        James Meyers
|                Brown University
|
| Start Date:    December 22, 1990
| Last Revision: February 20, 1991
|
| Intent:
}

{****} interface

uses crt,
     global_2;

const
  MAX_FAMILY_MEMBERS = 22;
  MAXGENES = 3;
  MAX_BUFFER_WIDTH = 22;

  {--------  Indicies into the input buffer ---------}

  FAMILY_NUM = 1;
  ID_NUM = 2;
  FATHER = 3;
  MOTHER = 4;
  FIRST_KID = 5;
  NEXT_PAT_SIB = 6;
  NEXT_MAT_SIB = 7;
  SEX = 8;
  PRO_BAND = 9;
  FIRST_GENE = 10;

{-------------------------------------------------------------------------------}

type
  buffer_type =
    array [1..MAX_FAMILY_MEMBERS] of array [1..MAX_BUFFER_WIDTH] of integer;

  allelic_code_type = array[1..maxgenes] of array[1..4] of string;

  {
  | A person_type variable stores information about a family member which
  | is not directly represented in the input file/buffer. The information
  | must be calculated based on interrelationships among family members.
  }

  person_type = record
    level:byte;
    spouse:byte;
    siblings:array[1..MAX_FAMILY_MEMBERS] of boolean;
  end; {record}
```

```
  family_type = array[1..MAX_FAMILY_MEMBERS] of person_type;

{-------------------------------------------------------------------------}

var
  infile : text;                          {input file (CEPH file)}
  buffer : buffer_type;                   {contains current CEPH family data}
  buffer_lines : byte;                    {lines of valid data in buffer}
  family : family_type;                   {current family information}
  r_value : real;                         {value of r}
  vb_file : text;                          {viability information outfile}

procedure process_next_family;
procedure make_inference_1;
procedure close_files;
procedure dump;
procedure dump_family;

{****} implementation

var
  fu_file,                                {factor-union outfile}
  gt_file,                                {genotype outfile}
  pd_file : text;                         {graphic pedigree outfile}
  scan : integer;                         {buffers the input file buffer}
  num_genes : byte;                       {number of genes being processed}
  allelic_codes : allelic_code_type;      {table of factor union conversions}

{-------------------------------------------------------------------------}

procedure initialize;

var
  name:string;

begin
  clrscr;
  write('Please enter input filename: ');
  readln(name);
  assign(infile,name);
  reset(infile);
  write('Please enter output file for viability data: ');
  readln(name);
  assign(vb_file,name);
  rewrite(vb_file);
  assign(fu_file,'c:out_fu.dat');
  rewrite(fu_file);
  assign(gt_file,'c:out_gt.dat');
  rewrite(gt_file);
  assign(pd_file,'c:out_pd.dat');
  rewrite(pd_file);
  read(infile,scan);
  write('Please enter value of "r": ');
```

```
  readln(r_value);
end; {procedure initialize}

{-------------------------------------------------------------------------}

procedure get_allelic_codes;

var
  i:byte;
  code:string;

begin
{  write('Enter the number of genes being compared: ');}
{  readln(num_genes);}
  num_genes:=2;
  if num_genes > MAXGENES then
    error('Analyses with that many genes are not supported');
  writeln;
  for i:=1 to num_genes do
   begin
    writeln('FACTOR UNION CODES FOR GENE ',i,':');
    write('Enter code for which represents a gene with no data: ');
    readln(allelic_codes[i][1]);
    write('Enter code for homozygous   (1 1)  configuration of gene ',i,': ');
    readln(allelic_codes[i][2]);
    write('Enter code for homozygous   (2 2)  configuration of gene ',i,': ');
    readln(allelic_codes[i][3]);
    write('Enter code for heterozygous (1 2)  configuration of gene ',i,': ');
    readln(allelic_codes[i][4]);
    writeln;
   end;
end; {procedure get_allelic_codes}

{-------------------------------------------------------------------------}

procedure null_buffer;

var
  i,j:integer;

begin
  for i:=1 to MAX_FAMILY_MEMBERS do
    for j:=1 to MAX_BUFFER_WIDTH do
      buffer[i][j]:=-1
end; {procedure null_buffer}

{-------------------------------------------------------------------------}

procedure fill_buffer;

var
  i,{row}
  j  {col} :integer;
  done:boolean;
```

```
  temp:integer;

begin
  done:=false;
  i:=1;
  j:=1;
  temp:=scan;
  buffer[i][j]:=scan;
  inc(j);
  if (temp<>scan) and (scan<>-1) then
    done:=true
  else
    begin
      while (not done) and (not eof(infile)) do
        begin
          while (not done) and (not eoln(infile)) do
            begin
              read(infile,buffer[i][j]);
              if (j=1) and (buffer[i][j]<>scan) then
                  begin
                    done:=true;
                    scan:=buffer[i][j];
                    buffer[i][j]:=-9;
                    buffer_lines:=i-1;
                  end;
              inc(j)
            end; {eoln}
          inc(i);
          j:=1;
          if not done then
            readln(infile);
          if eof(infile) then
             buffer_lines:=i-1;
        end {not done}
    end {else}
end; {procedure fill_buffer}

{-------------------------------------------------------------------------------}

function fu_to_gt(gene_number:byte; fu_code:string):integer;

{
| This procedure is passed a gene number and a factor-union code. A code of
| 1,2,3, or 4 is returned.
}

var
   i,j:byte;
   s:string;
   hold:integer;

begin
     hold:=-1;
     for j:=1 to maxgenes+1 do
```

```
         begin
         s:=allelic_codes[gene_number][j];
            if s = fu_code then
               hold:=j;
         end;
     fu_to_gt:=hold
end; {function fu_to_gt}

{-------------------------------------------------------------------------}

procedure process_buffer;

var
   i,j,k,l,m,n:byte;
   s,
   fu_code:string;
   gene_number:byte;
   temp:array[1..maxgenes] of integer;

begin
  for i:=1 to MAX_FAMILY_MEMBERS do
    begin
      gene_number:=1;
      j:=10;
      while (j<=MAX_BUFFER_WIDTH) and (gene_number<=num_genes) do
        begin
          fu_code:='';
          for k:=1 to length(allelic_codes[gene_number][1]) do
            begin
              str(buffer[i][j+k-1],s);
              fu_code:=fu_code+s
            end;
          temp[gene_number]:=fu_to_gt(gene_number,fu_code);
          inc(gene_number);
          inc(j,k)
        end; {while}
      m:=10;
      for l:=1 to num_genes do
        begin
          case temp[l] of
            1:begin {00}
                buffer[i][m+l-1]:=0;
                buffer[i][m+l]:=0
              end;
            2:begin {11}
                buffer[i][m+l-1]:=1;
                buffer[i][m+l]:=1
              end;
            3:begin {22}
                buffer[i][m+l-1]:=2;
                buffer[i][m+l]:=2
              end;
            4:begin {12}
                buffer[i][m+l-1]:=1;
```

```
                buffer[i][m+1]:=2
              end
            end; {case}
            inc(m);
          end; {for}
        for n:=1+m to MAX_BUFFER_WIDTH do
          buffer[i][n]:=-1;
      end {for i}
end; {procedure process_buffer}

{-------------------------------------------------------------------------}

procedure set_gene(person,gene,allele1,allele2:byte);

var
  j:byte;

begin
  if gene=1 then
    j:=FIRST_GENE
  else
    j:=FIRST_GENE+2;
  buffer[person][j]:=allele1;
  buffer[person][j+1]:=allele2
end;

{-------------------------------------------------------------------------}

function get_gene(person,gene:byte):byte;

var
  j:byte;

begin
  if gene=1 then
    j:=FIRST_GENE
  else
    j:=FIRST_GENE+2;
  get_gene:=buffer[person][j]+buffer[person][j+1]
end; {function get_gene}

{-------------------------------------------------------------------------}

procedure make_inference_2;

const
  unknown = 0;
  one_one = 2;
  one_two = 3;
  two_two = 4;

var
  i,j,k:byte;
  oo1_count, ot1_count, tt1_count,
```

```
  oo2_count, ot2_count, tt2_count:byte;
  p1_one, p1_two, p2_one, p2_two, which:byte;

begin
  for i:=1 to buffer_lines do
    if (family[i].level<>2) and (family[i].spouse<>0) then
      begin
        p1_one:=get_gene(i,1);
        p2_one:=get_gene(family[i].spouse,1);
        p1_two:=get_gene(i,2);
        p2_two:=get_gene(family[i].spouse,2);
        if (p1_one=unknown) or (p1_two=unknown) or (p2_one=unknown) or
           (p2_two=unknown) then
          begin
            oo1_count:=0;
            oo2_count:=0;
            ot1_count:=0;
            ot2_count:=0;
            tt1_count:=0;
            tt2_count:=0;
            for j:=1 to buffer_lines do
              begin
                if (buffer[j][MOTHER]=i) or (buffer[j][FATHER]=i) then
                  begin
                    case get_gene(j,1) of
                      one_one:inc(oo1_count);
                      one_two:inc(ot1_count);
                      two_two:inc(tt1_count)
                    end; {case}

                    case get_gene(j,2) of
                      one_one:inc(oo2_count);
                      one_two:inc(ot2_count);
                      two_two:inc(tt2_count)
                    end {case}
                  end; {if buffer...}
              end; {for j...}

{ u x u }

            if (p1_one=unknown) and (p2_one=unknown) then
              begin
                if (oo1_count>0) and (tt1_count>0) then
                  begin
                    set_gene(i,1,1,2);
                    p1_one:=one_two;
                    set_gene(family[i].spouse,1,1,2)
                  end;
                if (ot1_count>0) and ((oo1_count>0) or (tt1_count>0)) and
                   (p1_two=p2_two) then
                   begin
                        set_gene(i,1,1,2);
                        p1_one:=one_two
                   end
```

```
            end; {if p1_one...}

          if (p1_two=unknown) and (p2_two=unknown) then
            begin
              if (oo2_count>0) and (tt2_count>0) then
                begin
                  set_gene(i,2,1,2);
                  p1_two:=one_two;
                  set_gene(family[i].spouse,2,1,2);
                  p2_two:=one_two
                end;
              if (ot2_count>0) and ((oo2_count>0) or (tt2_count>0)) and
                  (p1_one=p2_one) then
                  begin
                       set_gene(i,2,1,2);
                       p1_two:=one_two
                  end

            end; {if p1_two...}

{ u x 11 }

          if (p1_one=unknown) and (p2_one=one_one) then
            begin
              if (ot1_count>0) and (oo1_count>0) then
                 begin
                       set_gene(i,1,1,2);
                       p1_one:=one_two
                 end
            end;

          if (p2_one=unknown) and (p1_one=one_one) then
            begin
              if (ot1_count>0) and (oo1_count>0) then
                 begin
                       set_gene(family[i].spouse,1,1,2);
                       p2_one:=one_two
                 end
            end;

          if (p1_two=unknown) and (p2_two=one_one) then
            begin
              if (ot2_count>0) and (oo2_count>0) then
                 begin
                       set_gene(i,2,1,2);
                       p1_two:=one_two
                 end
            end;

          if (p2_two=unknown) and (p1_two=one_one) then
            begin
              if (ot2_count>0) and (oo2_count>0) then
                 begin
```

```
                    set_gene(family[i].spouse,2,1,2);
                    p2_two:=one_two
                end
            end;

{ u x 22 }

          if (p1_one=unknown) and (p2_one=two_two) then
            begin
              if (ot1_count>0) and (tt1_count>0) then
                 begin
                    set_gene(i,1,1,2);
                    p1_one:=one_two
                 end
            end;

          if (p2_one=unknown) and (p1_one=two_two) then
            begin
              if (ot1_count>0) and (tt1_count>0) then
                 begin
                    set_gene(family[i].spouse,1,1,2);
                    p2_one:=one_two
                 end
            end;

          if (p1_two=unknown) and (p2_two=two_two) then
            begin
              if (ot2_count>0) and (tt2_count>0) then
                 begin
                    set_gene(i,2,1,2);
                    p1_two:=one_two
                 end
            end;

          if (p2_two=unknown) and (p1_two=two_two) then
            begin
              if (ot2_count>0) and (tt2_count>0) then
                 begin
                    set_gene(family[i].spouse,2,1,2);
                    p2_two:=one_two
                 end
            end;

{ u x 12 }

          if (p1_one=unknown) and (p2_one=one_two) then
            begin
              if (oo1_count>0) and (tt1_count>0) then
                 begin
                    set_gene(i,1,1,2);
                    p1_one:=one_two
                 end
            end;
```

```
            if (p2_one=unknown) and (p1_one=one_two) then
              begin
                if (oo1_count>0) and (tt1_count>0) then
                   begin
                        set_gene(family[i].spouse,1,1,2);
                        p2_one:=one_two
                   end
              end;

            if (p1_two=unknown) and (p2_two=one_two) then
              begin
                if (oo2_count>0) and (tt2_count>0) then
                   begin
                        set_gene(i,2,1,2);
                        p1_two:=one_two
                   end
              end;

            if (p2_two=unknown) and (p1_two=one_two) then
              begin
                if (oo2_count>0) and (tt2_count>0) then
                   begin
                        set_gene(family[i].spouse,2,1,2);
                        p2_two:=one_two
                   end
              end;

{         }

          end {if () or () or...}
      end {for i...}
end; {procedure make_inference_2}

{-------------------------------------------------------------------------}

procedure make_inference_1;

const
  unknown = 0;
  one_one = 2;
  one_two = 3;
  two_two = 4;

var
  i,j:byte;
  p1_one, p1_two, p2_one, p2_two:byte;

begin
  for i:=1 to buffer_lines do
    if (family[i].level<>2) and (family[i].spouse<>0) then
      begin
        p1_one:=get_gene(i,1);
        p2_one:=get_gene(family[i].spouse,1);
        p1_two:=get_gene(i,2);
```

```
        p2_two:=get_gene(family[i].spouse,2);
        for j:=1 to buffer_lines do
          begin
            if (buffer[j][MOTHER]=i) or (buffer[j][FATHER]=i) then
              begin
                if ((p1_one=one_one) or (p1_one=two_two)) and
                   ((p2_one=one_one) or (p2_one=two_two)) then
                  begin
                    if (p1_one=p2_one) then
                       if p1_one=one_one then
                          set_gene(j,1,1,1)
                       else
                           set_gene(j,1,2,2);
                    if (p1_one<>p2_one) then
                       set_gene(j,1,1,2)
                  end;

                if ((p1_two=one_one) or (p1_two=two_two)) and
                   ((p2_two=one_one) or (p2_two=two_two)) then
                  begin
                    if (p1_two=p2_two) then
                       if p1_two=one_one then
                          set_gene(j,2,1,1)
                       else
                           set_gene(j,2,2,2);
                    if (p1_two<>p2_two) then
                       set_gene(j,2,1,2)
                  end;
              end {if buffer...}
          end {for j...}
      end {for i...}
end; {procedure make_inference_1}

{-------------------------------------------------------------------------}

procedure append_fu_file;

var
  i,j:byte;
  done:boolean;

begin
  done:=false;
  i:=1;
  j:=1;
  while (i<=MAX_FAMILY_MEMBERS) and (not done) do
    begin
      while (j<=MAX_BUFFER_WIDTH) and (not done) do
        begin
          if (j=1) and (buffer[i][j]<0) then
            done:=true
          else
            begin
              if (buffer[i][j]>=0) and (not(j in [5,6,7,9])) then
```

```
              write(fu_file,buffer[i][j]:3);
            inc(j)
          end;
        end;
      writeln(fu_file);
      inc(i);
      j:=1
    end;
end; {procedure append_gt_file}

{-------------------------------------------------------------------------}

procedure append_gt_file;

var
  i,j:byte;
  done:boolean;

begin
  done:=false;
  i:=1;
  j:=1;
  while (i<=MAX_FAMILY_MEMBERS) and (not done) do
    begin
      while (j<=MAX_BUFFER_WIDTH) and (not done) do
        begin
          if (j=1) and (buffer[i][j]<0) then
            done:=true
          else
            begin
              if (buffer[i][j]>=0) and (not(j in [5,6,7,9])) then
                write(gt_file,buffer[i][j]:3);
              inc(j)
            end;
        end;
      writeln(gt_file);
      inc(i);
      j:=1
    end;
end; {procedure append_gt_file}

{=========================================================================}

{                      PEDIGREE CONSTRUCTION ROUTINES                     }

{-------------------------------------------------------------------------}

procedure init_family;

var
  i,j:integer;

begin
     for i:=1 to MAX_FAMILY_MEMBERS do
```

```
         with family[i] do
          begin
             spouse:=0;
             level:=2;
             for j:=1 to MAX_FAMILY_MEMBERS do
              siblings[j]:=false
              end {with}
end; {procedure init_pedigree}

{-------------------------------------------------------------------------}

procedure assign_levels;

var
  i,j:byte;
  level_0:array[1..4] of byte;
  level_0c:byte;
  level_1:array[1..MAX_FAMILY_MEMBERS] of byte;
  level_1c:byte;
  done:boolean;

begin
  level_0c:=0;
  level_1c:=0;
  for i:=1 to 4 do
    level_0[i]:=99;
  for i:=1 to MAX_FAMILY_MEMBERS do
    level_1[i]:=99;
  for i:=1 to buffer_lines do
    if (buffer[i][MOTHER]=0) and (buffer[i][FATHER]=0) then
      begin
        family[i].level:=0;
        inc(level_0c);
        level_0[level_0c]:=buffer[i][ID_NUM]
      end;
  for i:=1 to buffer_lines do
    begin
      j:=1;
      done:=false;
      while (not done) and (j<=4) do
        begin
          if (buffer[i][MOTHER] = level_0[j]) or
            (buffer[i][FATHER] = level_0[j]) then
              begin
                family[i].level:=1;
                inc(level_1c);
                level_1[level_1c]:=buffer[i][ID_NUM];
                done:=true;
              end;
          inc(j)
        end
    end;
end; {procedure init_pedigree}
```

```
{-------------------------------------------------------------------------}

procedure link_couples;

var
  i,j:byte;
  mom, dad:byte;

begin
  for i:=1 to buffer_lines do
    begin
      if family[i].level > 0 then
        begin
          mom:=buffer[i][MOTHER];
          dad:=buffer[i][FATHER];
          family[mom].spouse:=dad;
          family[dad].spouse:=mom
        end
    end
end; {procedure link_couples}

{-------------------------------------------------------------------------}

procedure link_siblings;

var
  i,j:byte;

begin
  for i:=1 to buffer_lines do
    for j:=1 to buffer_lines do
      if buffer[i][MOTHER] = buffer[j][MOTHER] then
        family[i].siblings[j]:=true;
end; {procedure link_siblings}

{-------------------------------------------------------------------------}

procedure construct_pedigree;

begin
  init_family;
  assign_levels;
  link_couples;
  link_siblings;
end; {procedure construct_pedigree}

{-------------------------------------------------------------------------}

{=========================================================================}
{                        PEDIGREE OUTPUT ROUTINES                         }
{                     ***** UNDER CONSTRUCTION *****                      }
{=========================================================================}

procedure write_genes(id:byte);
```

```
var
  i:byte;

begin
  i:=10;
  while i<(10+(2*num_genes)) do
    begin
      write(pd_file,' (',buffer[id][i],'|');
      inc(i);
      write(pd_file,buffer[id][i],')  ');
      inc(i);
    end;
end; {procedure write_genes}

{-------------------------------------------------------------------------------}

procedure write_pedigree;

var i:byte;

begin
  writeln(pd_file,'FAMILY ',buffer[1][1],' has ',buffer_lines,' members');
  writeln(pd_file);
  write(pd_file,'level 0: ');
  for i:=1 to buffer_lines do
    if family[i].level=0 then
      begin
        write(pd_file,buffer[i][ID_NUM]:2,'  ');
        write_genes(i);
        writeln(pd_file);
        write(pd_file,'         ');
      end;
  writeln(pd_file);

  write(pd_file,'level 1: ');
  for i:=1 to buffer_lines do
    if family[i].level=1 then
      begin
        write(pd_file,buffer[i][ID_NUM]:2,'  ');
        write_genes(i);
        writeln(pd_file);
        write(pd_file,'         ');
      end;
  writeln(pd_file);

  write(pd_file,'level 2: ');
  for i:=1 to buffer_lines do
    if family[i].level=2 then
      begin
        write(pd_file,buffer[i][ID_NUM]:2,'  ');
        write_genes(i);
        writeln(pd_file);
        write(pd_file,'         ');
```

```
      end;
  writeln(pd_file);

  writeln(pd_file);
  writeln(pd_file,'------------------------------------------------');
  writeln(pd_file)
end; {procedure print_pedigree}

{-----------------------------------------------------------------------------}
{=============================================================================}

procedure dump;

var
  i,j:byte;
  done:boolean;

begin
  done:=false;
  writeln;
  i:=1;
  j:=1;
  while (i<=MAX_FAMILY_MEMBERS) and (not done) do
    begin
      while (j<=MAX_BUFFER_WIDTH) and (not done) do
        begin
          if (j=1) and (buffer[i][j]<0) then
            done:=true
          else
            begin
              if buffer[i][j]>=0 then
                write(buffer[i][j]:3);
              inc(j)
            end;
        end;
      writeln;
      inc(i);
      j:=1
    end;
end; {procedure dump}

{-----------------------------------------------------------------------------}

procedure dump_family;

var
  i:byte;

begin
  for i:=1 to buffer_lines do
    with family[i] do
      begin
        writeln(buffer[i][1],'|',buffer[i][2],' --> level: ',level,'  spouse:
',spouse);
```

```
      end; {with}
  writeln;
end; {procedure dump_family}

{-------------------------------------------------------------------------}

procedure close_files;

begin
  close(infile);
  close(fu_file);
  close(gt_file);
  close(pd_file);
  close(vb_file);
end;

{-------------------------------------------------------------------------}

procedure process_next_family;

begin
      null_buffer;
      fill_buffer;
      append_fu_file;
      process_buffer;
      construct_pedigree;
      make_inference_1;
      make_inference_2;
      append_gt_file;
      write_pedigree;
end; {process_next_family}

{-------------------------------------------------------------------------}

begin
  initialize;
  get_allelic_codes;
end.
```

6.5 Viable_2.pas

```
unit viable_2;

{
| File:          viable_2.pas
| Compiles to:   viable_2.tpu
| Usage:         "uses viable_2;"
|
| Author:        James Meyers
|                Brown University
|
| Start Date:     February 10, 1991
```

```
| Last Revision: February 20, 1991
|
| Intent:
|
}

{****} interface

uses global_2,
     in_out_2,
     lookup_2;

const
  MAX_FAMILIES = 100;              {maximum number of families in a table}

type
  polynomial_expression = record
    fltpt:real;
    r:integer;
    x:integer;
    y:integer;
  end;

  vdata_ptr = ^vdata_type;
  vdata_type = record
    family_id:integer;
    genes:array[1..2] of byte;
    data:array[1..NUM_SEGREGATIONS,1..2] of polynomial_expression;
  end;

  cumulative_vdata_type = array[1..MAX_FAMILIES] of vdata_ptr;

{-------------------------------------------------------------------------------}

var
  current_vdata:vdata_type;
  family_vdata:vdata_type;
  cumulative_vdata:cumulative_vdata_type;
  vdata_cum_totals:vdata_type;

{-------------------------------------------------------------------------------}

procedure init_vdata(var vdata:vdata_type);
procedure output_vdata(p:vdata_type; flag:byte);
procedure output_cumulative_vdata;
procedure extract_viability_data;
function sum_vdata_components(poly:polynomial_expression):real;


{****} implementation


procedure add_to_vdata(var target:vdata_type;index:byte;source1,source2:real);
```

```
begin
  with target do
    begin
      if source1=R then
        inc(data[index][1].r)
      else if source1=ONE_MINUS_R then
        begin
          dec(data[index][1].r);
          data[index][1].fltpt := data[index][1].fltpt + 1.0
        end
      else if source1=X then
        inc(data[index][1].x)
      else if source1=ONE_MINUS_X then
        begin
          dec(data[index][1].x);
          data[index][1].fltpt := data[index][1].fltpt + 1.0
        end
      else if source1=Y then
        inc(data[index][1].y)
      else if source1=ONE_MINUS_Y then
        begin
          dec(data[index][1].y);
          data[index][1].fltpt := data[index][1].fltpt + 1.0
        end
      else
          data[index][1].fltpt := data[index][1].fltpt + source1;

      if source2=R then
        inc(data[index][2].r)
      else if source2=ONE_MINUS_R then
        begin
          dec(data[index][2].r);
          data[index][2].fltpt := data[index][2].fltpt + 1.0
        end
      else if source2=X then
        inc(data[index][2].x)
      else if source2=ONE_MINUS_X then
        begin
          dec(data[index][2].x);
          data[index][2].fltpt := data[index][2].fltpt + 1.0
        end
      else if source2=Y then
        inc(data[index][2].y)
      else if source2=ONE_MINUS_Y then
        begin
          dec(data[index][2].y);
          data[index][2].fltpt := data[index][2].fltpt + 1.0
        end
      else
          data[index][2].fltpt := data[index][2].fltpt + source2;

    end {with}
end; {procedure add_to_vdata}
```

```
{-------------------------------------------------------------------------------}

procedure init_cumulative_vdata;

var
  i:integer;

begin
  for i:=1 to MAX_FAMILIES do
    cumulative_vdata[i]:=NIL;
end; {procedure init_cum_total}

{-------------------------------------------------------------------------------}

procedure init_cum_totals;

var
  i,j:byte;

begin
  with vdata_cum_totals do
    begin
      genes[1]:=0;
      genes[2]:=0;
      for i:=1 to NUM_SEGREGATIONS do
        for j:=1 to 2 do
          begin
            data[i][j].r:=0;
            data[i][j].fltpt:=0.0;
            data[i][j].x:=0;
            data[i][j].y:=0;
          end {for j}
    end {with}
end; {procedure init_cum_totals}

{-------------------------------------------------------------------------------}

procedure init_vdata(var vdata:vdata_type);

var
  i,j:byte;

begin
  with vdata do
    begin
      family_id:=0;
      genes[1]:=0;
      genes[2]:=0;
      for i:=1 to NUM_SEGREGATIONS do
        for j:=1 to 2 do
          begin
            data[i][j].r:=0;
            data[i][j].fltpt:=0.0;
            data[i][j].x:=0;
```

```
            data[i][j].y:=0;
          end {for j}
    end {with}
end; {procedure init_current_vdata}

{-------------------------------------------------------------------------------}

procedure update_cum_vdata;

var
  p:vdata_ptr;
  i,j,k:byte;

begin
  new(p);
  i:=1;
  while (cumulative_vdata[i]<>NIL) and (i<=MAX_FAMILIES) do
    inc(i);
  if i=MAX_FAMILIES then
    error('Index out of range in VIABLE_2.TPU:update_cum_vdata');
  for j:=1 to NUM_SEGREGATIONS do
    for k:=1 to 2 do
      with current_vdata do
        begin
          p^.genes[k]:=genes[k];
          p^.family_id:=family_id;
          p^.data[j][k].fltpt:=data[j][k].fltpt;
          p^.data[j][k].r:=data[j][k].r;
          p^.data[j][k].x:=data[j][k].x;
          p^.data[j][k].y:=data[j][k].y;
        end;
  cumulative_vdata[i]:=p
end; {procedure update_cum_vdata}

{-------------------------------------------------------------------------------}

procedure format_pterm(coeff:integer; term:char);

begin
  if coeff < 0 then
    write(vb_file,' - ');
  if coeff > 0 then
    write(vb_file,' + ');
  if abs(coeff) > 1 then
    write(vb_file,abs(coeff):2);
  if abs(coeff)=1 then
    write(vb_file,'  ');
  if coeff <> 0 then
    write(vb_file,term)
  else
    write(vb_file,'      ')
end; {procedure format_pterm}

{-------------------------------------------------------------------------------}
```

```
procedure output_vdata(p:vdata_type; flag:byte);

var
  i,j:byte;

begin
  writeln(vb_file);
  if flag = 1 then
    writeln(vb_file,'Totals: ')
  else
    writeln(vb_file,'Family ',p.family_id);
  for i:=1 to NUM_SEGREGATIONS do
    begin
      write(vb_file,SEGREGATIONS[i][1],SEGREGATIONS[i][2],' ');
      write(vb_file,SEGREGATIONS[i][3],SEGREGATIONS[i][4],' --> ');
        with p do
          begin
            write(vb_file,data[i][1].fltpt:7:2);
            format_pterm(data[i][1].r,'r');
            format_pterm(data[i][1].x,'x');
            format_pterm(data[i][1].y,'y');

            write(vb_file,'   <-> ');

            write(vb_file,data[i][2].fltpt:7:2);
            format_pterm(data[i][2].r,'r');
            format_pterm(data[i][2].x,'x');
            format_pterm(data[i][2].y,'y');
            writeln(vb_file)
          end
    end;
  writeln(vb_file);
end; {procedure output_vdata}

{-------------------------------------------------------------------------}

procedure output_cumulative_vdata;

begin
  output_vdata(vdata_cum_totals,1)
end; {procedure output_cumulative_vdata}

{-------------------------------------------------------------------------}

function genotype_number(i:byte):byte;

var
  s,t:string[5];
  j:byte;
  hold:byte;

begin
  str(buffer[i][FIRST_GENE],s);
```

```
  str(buffer[i][FIRST_GENE + 2],t);
  s:=concat(s,t);
  str(buffer[i][FIRST_GENE + 1],t);
  s:=concat(s,t);
  str(buffer[i][FIRST_GENE + 3],t);
  s:=concat(s,t);

  hold:=0;
  j:=1;
  while (j<=NUM_GENOTYPES) and (hold=0) do
    begin
      if s=GENOTYPES[j] then
        hold:=j;
      inc(j)
    end;

  genotype_number:=hold
end; {function genotype_number}

{-------------------------------------------------------------------------------}

procedure update_current_vdata;

begin
end; {procedure update_current_vdata}

{-------------------------------------------------------------------------------}

procedure extract_viability_data;

var
  i,j,k,temp : byte;
  _mother,_father,_child,_known : byte;
  s1,s2:real;
  phase_changes:array[1..MAX_FAMILY_MEMBERS] of byte;
  level_index:array[1..MAX_FAMILY_MEMBERS] of byte;
  informative:boolean;

begin

informative:=false;

{*****}
  for i:=1 to MAX_FAMILY_MEMBERS do
    begin
      phase_changes[i]:=0;
      level_index[i]:=0
    end;

{*****}
  i:=1;
  for j:=1 to buffer_lines do
    if family[j].level=1 then
      begin
```

```
        level_index[i]:=j;
        inc(i)
      end;
  for j:=1 to buffer_lines do
    if family[j].level=2 then
      begin
        level_index[i]:=j;
        inc(i)
      end;
  for j:=1 to buffer_lines do
    if family[j].level=0 then
      begin
        level_index[i]:=j;
        inc(i)
      end;
  if i<>buffer_lines+1 then
    error('level_index initialization error');

{*****}
  init_vdata(family_vdata);
  family_vdata.family_id:=buffer[1][FAMILY_NUM];
  for k:=1 to buffer_lines do
    begin
      i:=level_index[k];
      if family[i].level>0 then
        begin
          if (phase_changes[(buffer[i][MOTHER])] = 0) then
            begin
              _mother:=genotype_number(buffer[i][MOTHER]);
              if _mother=5 then _mother:=7
            end
          else
            _mother:=phase_changes[(buffer[i][MOTHER])];
          if (phase_changes[(buffer[i][FATHER])] = 0) then
            begin
              _father:=genotype_number(buffer[i][FATHER]);
              if _father=5 then _father:=7
            end
          else
            _father:=phase_changes[buffer[i][FATHER]];

          _child:=genotype_number(i);
          if _child=5 then _child:=7;

          if _mother > _father then
            begin
              temp:=_mother;
              _mother:=_father;
              _father:=temp
            end;
          if (_mother<>0) and (_father<>0) and (_child <> 0) then
            begin
              informative:=true;
              init_vdata(current_vdata);
```

```
              for j:=1 to NUM_SEGREGATIONS do
                begin
                  s1:=the_table[_mother][_father][_child][j]^[1];
                  s2:=the_table[_mother][_father][_child][j]^[2];
                  add_to_vdata(current_vdata,j,s1,s2);
                  add_to_vdata(vdata_cum_totals,j,s1,s2);
                  add_to_vdata(family_vdata,j,s1,s2)
                end;
              if delta_phase[_mother][_father][_child] > 0 then
                phase_changes[i]:=delta_phase[_mother][_father][_child]
            end {if}
{*****}
          else if ((_mother=0) xor (_father=0)) and (_child<>0) then
            begin
              init_vdata(current_vdata);
              informative:=true;
              if _mother=0 then
                _known:=_father
              else
                _known:=_mother;
              for j:=1 to NUM_SEGREGATIONS do
                begin
                  s1:=useg_table[_known][_child][j][1];
                  s2:=useg_table[_known][_child][j][2];
                  add_to_vdata(current_vdata,j,s1,s2);
                  add_to_vdata(vdata_cum_totals,j,s1,s2);
                  add_to_vdata(family_vdata,j,s1,s2)
                end {for}
            end {else if}
{*****}
        end
    end; {for k}
  if informative then
    output_vdata(family_vdata,0)
  else
    writeln(vb_file,'Family ',buffer[i][FAMILY_NUM]:5,' is not informative')
end; {procedure extract_viability_data}

{-------------------------------------------------------------------------}

function sum_vdata_components(poly:polynomial_expression):real;

var
  hold:real;

begin
  hold:=0;
  with poly do
    begin
      hold:=hold+fltpt;
      hold:=hold+r*r_value;
      hold:=hold+x*x_value(r_value);
      hold:=hold+y*y_value(r_value)
    end;
```

```
  sum_vdata_components:=hold
end; {function sum_vdata_components}

{-------------------------------------------------------------------------------}

begin
  init_cumulative_vdata;
  init_cum_totals;
end.
```

6.6 Gtest.pas

```
unit gtest;

{
| File:           gtest.pas
| Compiles to:    gtest.tpu
| Usage:          "uses gtest;"
|
| Author:         Lisa Brooks & James Meyers
|                 Brown University
|
| Start Date:      February 20, 1991
| Last Revision: February 20, 1991
|
| Intent:         Does a G-test of whether numbers are in a 1:1 ratio
}

{****} interface

uses global_2,
     in_out_2,
     viable_2;

procedure output_g_test_data(vdata:vdata_type);

{****} implementation

procedure g_test(obs1,obs2:real);

var
   total,
   exp,
   g, g1, g2,
   willcorr : real;

begin
  total := obs1 + obs2;
  exp   := total / 2;

  if total < 10 then writeln(vb_file,'too few')
  else
```

```
    begin {g test}
      if (obs1 > 0) then
        g1 := obs1 * ln(obs1)
      else g1 := 0;

      if (obs2 > 0) then
        g2 := obs2 * ln(obs2)
      else g2 := 0;

      g := g1 + g2 - total * ln(exp);

      {Williams correction for small samples}

      willcorr := 1 + 1 / (2 * total);
      g := 2 * g / willcorr;

      write(vb_file,'G = ', g:8:3,'  -----  ');

      if (10.828 < g)  then
        writeln(vb_file,' .001 > P       ')
      else if (7.879  < g)  then
        writeln(vb_file,' .005 > P > .001')
      else if (6.635  < g)  then
        writeln(vb_file,' .01  > P > .005')
      else if (5.024  < g)  then
        writeln(vb_file,' .025 > P > .01 ')
      else if (3.841  < g)  then
        writeln(vb_file,' .05  > P > .025')
      else if (2.706  < g)  then
        writeln(vb_file,' .1   > P > .05 ')
      else if (0.455  < g)  then
        writeln(vb_file,' .5   > P > .1  ')
      else if (0.016  < g)  then
        writeln(vb_file,' .9   > P > .5  ')
      else
        writeln(vb_file,'        P > .9  ');
    end; {g test}
end; {procedure g_test}

{-------------------------------------------------------------------------------}

procedure output_g_test_data(vdata:vdata_type);

var
  i:byte;
  obs1,obs2:real;

begin
  for i:=1 to NUM_SEGREGATIONS do
    begin
      obs1:=sum_vdata_components(vdata_cum_totals.data[i][1]);
      write(vb_file,SEGREGATIONS[i][1],SEGREGATIONS[i][2],' ');
      write(vb_file,SEGREGATIONS[i][3],SEGREGATIONS[i][4],': ',obs1:6:2);
      obs2:=sum_vdata_components(vdata_cum_totals.data[i][2]);
```

```
      write(vb_file,' | ',obs2:6:2,' ------ ');
      g_test(obs1,obs2)
    end
end;

{-------------------------------------------------------------------------}

begin
end.
```

7.0 Haplotype Analyzer Lookup Tables

7.1 Phase.dat

```
1    5     7    5
1    6     7    5
1    7     7    5
1    8     7    5
1    10    7    5
1    11    7    5

2    4     7    6
2    8     7    5
2    9     7    6
2    11    7    5

3    4     7    6
3    5     7    6
3    6     7    6
3    7     7    6
3    9     7    6
3    10    7    6

4    10    7    5
4    11    7    5

5    9     7    6
5    11    7    5

6    9     7    6
6    11    7    5

7    9     7    6
7    11    7    5

8    9     7    6
8    10    7    6
```

7.2 Useg.dat

```
 1          1      0 0      0 0      0 0      0 0      0 0      0 0
 2          1      1 0      0 0      0 0      0 0      0 0      0 0
 2          3      0 1      0 0      0 0      0 0      0 0      0 0
 2          4      1 0      0 0      0 0      0 0      0 0      0 0
 2          8      0 1      0 0      0 0      0 0      0 0      0 0
 3          3      0 0      0 0      0 0      0 0      0 0      0 0
 4          1      0 0      1 0      0 0      0 0      0 0      0 0
 4          2      0 0      1 0      0 0      0 0      0 0      0 0
 4          9      0 0      0 1      0 0      0 0      0 0      0 0
 4         10      0 0      0 1      0 0      0 0      0 0      0 0
 5          1      0 0      0 0      0 0      0 0      1 0      0 0
 5          3      0 0      0 0      0 0      0 0      0 0      1 0
 5          9      0 0      0 0      0 0      0 0      0 0      0 1
 5         11      0 0      0 0      0 0      0 0      0 1      0 0
 6          1      0 0      0 0      0 0      0 0      1 0      0 0
 6          3      0 0      0 0      0 0      0 0      0 0      1 0
 6          9      0 0      0 0      0 0      0 0      0 0      0 1
 6         11      0 0      0 0      0 0      0 0      0 1      0 0
 7          1      0 0      0 0      0 0      0 0      1 0      0 0
 7          3      0 0      0 0      0 0      0 0      0 0      1 0
 7          9      0 0      0 0      0 0      0 0      0 0      0 1
 7         11      0 0      0 0      0 0      0 0      0 1      0 0
 8          2      0 0      0 0      1 0      0 0      0 0      0 0
 8          3      0 0      0 0      1 0      0 0      0 0      0 0
 8         10      0 0      0 0      0 1      0 0      0 0      0 0
 8         11      0 0      0 0      0 1      0 0      0 0      0 0
 9          9      0 0      0 0      0 0      0 0      0 0      0 0
10          4      0 0      0 0      0 0      1 0      0 0      0 0
10          8      0 0      0 0      0 0      0 1      0 0      0 0
10          9      0 0      0 0      0 0      1 0      0 0      0 0
10         11      0 0      0 0      0 0      0 1      0 0      0 0
11         11      0 0      0 0      0 0      0 0      0 0      0 0
```

7.3 Table.dat

```
1   2   1      1   0    0   0    0   0    0   0    0   0    0   0
1   2   2      0   1    0   0    0   0    0   0    0   0    0   0

1   4   1      0   0    1   0    0   0    0   0    0   0    0   0
1   4   4      0   0    0   1    0   0    0   0    0   0    0   0

1   5   1      0   0    0   0    0   0    0   0    1   0    0   0
1   5   2      0   0    0   0    0   0    0   0    0   0    1   0
1   5   4      0   0    0   0    0   0    0   0    0   0    0   1
1   5   7      0   0    0   0    0   0    0   0    0   1    0   0

1   6   1      0   0    0   0    0   0    0   0    1   0    0   0
1   6   2      0   0    0   0    0   0    0   0    0   0    1   0
1   6   4      0   0    0   0    0   0    0   0    0   0    0   1
1   6   7      0   0    0   0    0   0    0   0    0   1    0   0
```

```
1   7   1      0   0    0   0    0   0    0   0    1   0    0   0
1   7   2      0   0    0   0    0   0    0   0    0   0    1   0
1   7   4      0   0    0   0    0   0    0   0    0   0    0   1
1   7   7      0   0    0   0    0   0    0   0    0   1    0   0

1   8   2      0   0    0   0    1   0    0   0    0   0    0   0
1   8   7      0   0    0   0    0   1    0   0    0   0    0   0

1   10  4      0   0    0   0    0   0    1   0    0   0    0   0
1   10  7      0   0    0   0    0   0    0   1    0   0    0   0

2   2   1      2   0    0   0    0   0    0   0    0   0    0   0
2   2   2      1   1    0   0    0   0    0   0    0   0    0   0
2   2   3      0   2    0   0    0   0    0   0    0   0    0   0

2   3   2      1   0    0   0    0   0    0   0    0   0    0   0
2   3   3      0   1    0   0    0   0    0   0    0   0    0   0

2   4   1      1   0    1   0    0   0    0   0    0   0    0   0
2   4   2      0   1    1   0    0   0    0   0    0   0    0   0
2   4   4      1   0    0   1    0   0    0   0    0   0    0   0
2   4   7      0   1    0   1    0   0    0   0    0   0    0   0

2   5   1      1   0    0   0    0   0    0   0    1   0    0   0
2   5   2      -1  -2   0   0    0   0    0   0    -2  0    -1  0
2   5   3      0   1    0   0    0   0    0   0    0   0    1   0
2   5   4      1   0    0   0    0   0    0   0    0   0    0   1
2   5   5      1   0    0   0    0   0    0   0    0   1    0   0
2   5   6      0   1    0   0    0   0    0   0    0   0    0   1
2   5   7      -2  -1   0   0    0   0    0   0    0   -2   0   -1
2   5   8      0   1    0   0    0   0    0   0    0   1    0   0

2   6   1      1   0    0   0    0   0    0   0    1   0    0   0
2   6   2      -2  -1   0   0    0   0    0   0    -1  0    -2  0
2   6   3      0   1    0   0    0   0    0   0    0   0    1   0
2   6   4      1   0    0   0    0   0    0   0    0   0    0   1
2   6   5      1   0    0   0    0   0    0   0    0   1    0   0
2   6   6      0   1    0   0    0   0    0   0    0   0    0   1
2   6   7      -1  -2   0   0    0   0    0   0    0   -1   0   -2
2   6   8      0   1    0   0    0   0    0   0    0   1    0   0

2   7   1      1   0    0   0    0   0    0   0    1   0    0   0
2   7   2      .5  .5   0   0    0   0    0   0    .5  0    .5  0
2   7   3      0   1    0   0    0   0    0   0    0   0    1   0
2   7   4      1   0    0   0    0   0    0   0    0   0    0   1
2   7   5      1   0    0   0    0   0    0   0    0   1    0   0
2   7   6      0   1    0   0    0   0    0   0    0   0    0   1
2   7   7      .5  .5   0   0    0   0    0   0    0   .5   0   .5
2   7   8      0   1    0   0    0   0    0   0    0   1    0   0

2   8   2      1   0    0   0    1   0    0   0    0   0    0   0
2   8   3      0   1    0   0    1   0    0   0    0   0    0   0
2   8   7      1   0              0   00   10   00   00   0
2   8   8      0   1    0   0    0   1    0   0    0   0    0   0
```

2	9	4	1	0	0	0	0	0	0	0	0	0	0
2	9	7	0	1	0	0	0	0	0	0	0	0	0
2	10	4	1	0	0	0	0	0	1	0	0	0	0	0
2	10	5	1	0	0	0	0	0	0	1	0	0	0	0
2	10	6	0	1	0	0	0	0	1	0	0	0	0	0
2	10	7	.5	.5	0	0	0	0	.5	.5	0	0	0	0
2	10	8	0	1	0	0	0	0	0	1	0	0	0	0
2	11	7	1	0	0	0	0	0	0	0	0	0	0	0
2	11	8	0	1	0	0	0	0	0	0	0	0	0	0
3	4	2	0	0	1	0	0	0	0	0	0	0	0	0
3	4	7	0	0	0	1	0	0	0	0	0	0	0	0
3	5	2	0	0	0	0	0	0	0	0	1	0	0	0
3	5	3	0	0	0	0	0	0	0	0	0	0	1	0
3	5	7	0	0	0	0	0	0	0	0	0	0	0	1
3	5	8	0	0	0	0	0	0	0	0	0	1	0	0
3	6	2	0	0	0	0	0	0	0	0	1	0	0	0
3	6	3	0	0	0	0	0	0	0	0	0	0	1	0
3	6	7	0	0	0	0	0	0	0	0	0	0	0	1
3	6	8	0	0	0	0	0	0	0	0	0	1	0	0
3	7	2	0	0	0	0	0	0	0	0	1	0	0	0
3	7	3	0	0	0	0	0	0	0	0	0	0	1	0
3	7	7	0	0	0	0	0	0	0	0	0	0	0	1
3	7	8	0	0	0	0	0	0	0	0	0	1	0	0
3	8	3	0	0	0	0	1	0	0	0	0	0	0	0
3	8	8	0	0	0	0	0	1	0	0	0	0	0	0
3	10	7	0	0	0	0	0	0	1	0	0	0	0	0
3	10	8	0	0	0	0	0	0	0	1	0	0	0	0
4	4	1	0	0	2	0	0	0	0	0	0	0	0	0
4	4	4	0	0	1	1	0	0	0	0	0	0	0	0
4	4	9	0	0	0	2	0	0	0	0	0	0	0	0
4	5	1	0	0	1	0	0	0	0	0	1	0	0	0
4	5	2	0	0	1	0	0	0	0	0	0	0	1	0
4	5	4	0	0	-1	-2	0	0	0	0	-2	0	0	-1
4	5	5	0	0	1	0	0	0	0	0	0	1	0	0
4	5	6	0	0	0	1	0	0	0	0	0	0	1	0
4	5	7	0	0	-2	-1	0	0	0	0	0	-2	-1	0
4	5	9	0	0	0	1	0	0	0	0	0	0	0	1
4	5	10	0	0	0	1	0	0	0	0	0	1	0	0
4	6	1	0	0	1	0	0	0	0	0	1	0	0	0
4	6	2	0	0	1	0	0	0	0	0	0	0	1	0
4	6	4	0	0	-2	-1	0	0	0	0	-1	0	0	-2
4	6	5	0	0	1	0	0	0	0	0	0	1	0	0

```
4   6   6     0   0   0   1   0   0   0   0   0   0   1   0
4   6   7     0   0  -1  -2   0   0   0   0   0  -1  -2   0
4   6   9     0   0   0   1   0   0   0   0   0   0   0   1
4   6   10    0   0   0   1   0   0   0   0   0   1   0   0

4   7   1     0   0   1   0   0   0   0   0   1   0   0   0
4   7   2     0   0   1   0   0   0   0   0   0   0   1   0
4   7   4     0   0  .5  .5   0   0   0   0  .5   0   0  .5
4   7   5     0   0   1   0   0   0   0   0   0   1   0   0
4   7   6     0   0   0   1   0   0   0   0   0   0   1   0
4   7   7     0   0  .5  .5   0   0   0   0   0  .5  .5   0
4   7   9     0   0   0   1   0   0   0   0   0   0   0   1
4   7   10    0   0   0   1   0   0   0   0   0   1   0   0

4   8   2     0   0   1   0   1   0   0   0   0   0   0   0
4   8   5     0   0   1   0   0   1   0   0   0   0   0   0
4   8   6     0   0   0   1   1   0   0   0   0   0   0   0
4   8   7     0   0  .5  .5  .5  .5   0   0   0   0   0   0
4   8   10    0   0   0   1   0   1   0   0   0   0   0   0

4   9   4     0   0   1   0   0   0   0   0   0   0   0   0
4   9   9     0   0   0   1   0   0   0   0   0   0   0   0

4   10  4     0   0   1   0   0   0   1   0   0   0   0   0
4   10  7     0   0   1   0   0   0   0   1   0   0   0   0
4   10  9     0   0   0   1   0   0   1   0   0   0   0   0
4   10  10    0   0   0   1   0   0   0   1   0   0   0   0

4   11  7     0   0   1   0   0   0   0   0   0   0   0   0
4   11  10    0   0   0   1   0   0   0   0   0   0   0   0

5   5   1     0   0   0   0   0   0   0   0   2   0   0   0
5   5   2     0   0   0   0   0   0   0   0   1   0   1   0
5   5   3     0   0   0   0   0   0   0   0   0   0   2   0
5   5   4     0   0   0   0   0   0   0   0   1   0   0   1
5   5   5     0   0   0   0   0   0   0   0   1   1   0   0
5   5   6     0   0   0   0   0   0   0   0   0   0   1   1
5   5   7     0   0   0   0   0   0   0   0  -4  -4  -3  -3
5   5   8     0   0   0   0   0   0   0   0   0   1   1   0
5   5   9     0   0   0   0   0   0   0   0   0   0   0   2
5   5   10    0   0   0   0   0   0   0   0   0   1   0   1
5   5   11    0   0   0   0   0   0   0   0   0   2   0   0

5   6   1     0   0   0   0   0   0   0   0   2   0   0   0
5   6   2     0   0   0   0   0   0   0   0   1   0   1   0
5   6   3     0   0   0   0   0   0   0   0   0   0   2   0
5   6   4     0   0   0   0   0   0   0   0   1   0   0   1
5   6   5     0   0   0   0   0   0   0   0   1   1   0   0
5   6   6     0   0   0   0   0   0   0   0   0   0   1   1
5   6   7     0   0   0   0   0   0   0   0  .5  .5  .5  .5
5   6   8     0   0   0   0   0   0   0   0   0   1   1   0
5   6   9     0   0   0   0   0   0   0   0   0   0   0   2
5   6   10    0   0   0   0   0   0   0   0   0   1   0   1
5   6   11    0   0   0   0   0   0   0   0   0   2   0   0
```

5	7	1	0	0	0	0	0	0	0	0	2	0	0	0
5	7	2	0	0	0	0	0	0	0	0	1	0	1	0
5	7	3	0	0	0	0	0	0	0	0	0	0	2	0
5	7	4	0	0	0	0	0	0	0	0	1	0	0	1
5	7	5	0	0	0	0	0	0	0	0	1	1	0	0
5	7	6	0	0	0	0	0	0	0	0	0	0	1	1
5	7	7	0	0	0	0	0	0	0	0	-6	-6	-5	-5
5	7	8	0	0	0	0	0	0	0	0	0	1	1	0
5	7	9	0	0	0	0	0	0	0	0	0	0	0	2
5	7	10	0	0	0	0	0	0	0	0	0	1	0	1
5	7	11	0	0	0	0	0	0	0	0	0	2	0	0
5	8	2	0	0	0	0	1	0	0	0	1	0	0	0
5	8	3	0	0	0	0	1	0	0	0	0	0	1	0
5	8	5	0	0	0	0	0	1	0	0	1	0	0	0
5	8	6	0	0	0	0	1	0	0	0	0	0	0	1
5	8	7	0	0	0	0	-1	-2	0	0	-2	0	0	-1
5	8	8	0	0	0	0	-2	-1	0	0	0	-2	-1	0
5	8	10	0	0	0	0	0	1	0	0	0	0	0	1
5	8	11	0	0	0	0	0	1	0	0	0	1	0	0
5	9	4	0	0	0	0	0	0	0	0	1	0	0	0
5	9	7	0	0	0	0	0	0	0	0	0	0	1	0
5	9	9	0	0	0	0	0	0	0	0	0	0	0	1
5	9	10	0	0	0	0	0	0	0	0	0	1	0	0
5	10	4	0	0	0	0	0	0	1	0	1	0	0	0
5	10	5	0	0	0	0	0	0	0	1	1	0	0	0
5	10	6	0	0	0	0	0	0	1	0	0	0	1	0
5	10	7	0	0	0	0	0	0	-1	-2	-2	0	-1	0
5	10	8	0	0	0	0	0	0	0	1	0	0	1	0
5	10	9	0	0	0	0	0	0	1	0	0	0	0	1
5	10	10	0	0	0	0	0	0	-2	-1	0	-2	0	-1
5	10	11	0	0	0	0	0	0	0	1	0	1	0	0
5	11	7	0	0	0	0	0	0	0	0	1	0	0	0
5	11	8	0	0	0	0	0	0	0	0	0	0	1	0
5	11	10	0	0	0	0	0	0	0	0	0	0	0	1
5	11	11	0	0	0	0	0	0	0	0	0	1	0	0
6	6	1	0	0	0	0	0	0	0	0	2	0	0	0
6	6	2	0	0	0	0	0	0	0	0	1	0	1	0
6	6	3	0	0	0	0	0	0	0	0	0	0	2	0
6	6	4	0	0	0	0	0	0	0	0	1	0	0	1
6	6	5	0	0	0	0	0	0	0	0	1	1	0	0
6	6	6	0	0	0	0	0	0	0	0	0	0	1	1
6	6	7	0	0	0	0	0	0	0	0	-3	-3	-4	-4
6	6	8	0	0	0	0	0	0	0	0	0	1	1	0
6	6	9	0	0	0	0	0	0	0	0	0	0	0	2
6	6	10	0	0	0	0	0	0	0	0	0	1	0	1
6	6	11	0	0	0	0	0	0	0	0	0	2	0	0
6	7	1	0	0	0	0	0	0	0	0	2	0	0	0

6	7	2	0	0	0	0	0	0	0	0	1	0	1	0
6	7	3	0	0	0	0	0	0	0	0	0	0	2	0
6	7	4	0	0	0	0	0	0	0	0	1	0	0	1
6	7	5	0	0	0	0	0	0	0	0	1	1	0	0
6	7	6	0	0	0	0	0	0	0	0	0	0	1	1
6	7	7	0	0	0	0	0	0	0	0	-5	-5	-6	-6
6	7	8	0	0	0	0	0	0	0	0	0	1	1	0
6	7	9	0	0	0	0	0	0	0	0	0	0	0	2
6	7	10	0	0	0	0	0	0	0	0	0	1	0	1
6	7	11	0	0	0	0	0	0	0	0	0	2	0	0
6	8	2	0	0	0	0	1	0	0	0	1	0	0	0
6	8	3	0	0	0	0	1	0	0	0	0	0	1	0
6	8	5	0	0	0	0	0	1	0	0	1	0	0	0
6	8	6	0	0	0	0	1	0	0	0	0	0	0	1
6	8	7	0	0	0	0	-2	-1	0	0	-1	0	0	-2
6	8	8	0	0	0	0	-1	-2	0	0	0	-1	-2	0
6	8	10	0	0	0	0	0	1	0	0	0	0	0	1
6	8	11	0	0	0	0	0	1	0	0	0	1	0	0
6	9	4	0	0	0	0	0	0	0	0	1	0	0	0
6	9	7	0	0	0	0	0	0	0	0	0	0	1	0
6	9	9	0	0	0	0	0	0	0	0	0	0	0	1
6	9	10	0	0	0	0	0	0	0	0	0	1	0	0
6	10	4	0	0	0	0	0	0	1	0	1	0	0	0
6	10	5	0	0	0	0	0	0	0	1	1	0	0	0
6	10	6	0	0	0	0	0	0	1	0	0	0	1	0
6	10	7	0	0	0	0	0	0	-2	-1	-1	0	-2	0
6	10	8	0	0	0	0	0	0	0	1	0	0	1	0
6	10	9	0	0	0	0	0	0	1	0	0	0	0	1
6	10	10	0	0	0	0	0	0	-1	-2	0	-1	0	-2
6	10	11	0	0	0	0	0	0	0	1	0	1	0	0
6	11	7	0	0	0	0	0	0	0	0	1	0	0	0
6	11	8	0	0	0	0	0	0	0	0	0	0	1	0
6	11	10	0	0	0	0	0	0	0	0	0	0	0	1
6	11	11	0	0	0	0	0	0	0	0	0	1	0	0
7	7	1	0	0	0	0	0	0	0	0	2	0	0	0
7	7	2	0	0	0	0	0	0	0	0	1	0	1	0
7	7	3	0	0	0	0	0	0	0	0	0	0	2	0
7	7	4	0	0	0	0	0	0	0	0	1	0	0	1
7	7	5	0	0	0	0	0	0	0	0	1	1	0	0
7	7	6	0	0	0	0	0	0	0	0	0	0	1	1
7	7	7	0	0	0	0	0	0	0	0	.5	.5	.5	.5
7	7	8	0	0	0	0	0	0	0	0	0	1	1	0
7	7	9	0	0	0	0	0	0	0	0	0	0	0	2
7	7	10	0	0	0	0	0	0	0	0	0	1	0	1
7	7	11	0	0	0	0	0	0	0	0	0	2	0	0
7	8	2	0	0	0	0	1	0	0	0	1	0	0	0
7	8	3	0	0	0	0	1	0	0	0	0	0	1	0
7	8	5	0	0	0	0	0	1	0	0	1	0	0	0

7	8	6	0	0	0	0	1	0	0	0	0	0	0	1
7	8	7	0	0	0	0	.5	.5	0	0	.5	0	0	.5
7	8	8	0	0	0	0	.5	.5	0	0	0	.5	.5	0
7	8	10	0	0	0	0	0	1	0	0	0	0	0	1
7	8	11	0	0	0	0	0	1	0	0	0	1	0	0
7	9	4	0	0	0	0	0	0	0	0	1	0	0	0
7	9	7	0	0	0	0	0	0	0	0	0	0	1	0
7	9	9	0	0	0	0	0	0	0	0	0	0	0	1
7	9	10	0	0	0	0	0	0	0	0	0	1	0	0
7	10	4	0	0	0	0	0	0	1	0	1	0	0	0
7	10	5	0	0	0	0	0	0	0	1	1	0	0	0
7	10	6	0	0	0	0	0	0	1	0	0	0	1	0
7	10	7	0	0	0	0	0	0	.5	.5	.5	0	.5	0
7	10	8	0	0	0	0	0	0	0	1	0	0	1	0
7	10	9	0	0	0	0	0	0	1	0	0	0	0	1
7	10	10	0	0	0	0	0	0	.5	.5	0	.5	0	.5
7	10	11	0	0	0	0	0	0	0	1	0	1	0	0
7	11	7	0	0	0	0	0	0	0	0	1	0	0	0
7	11	8	0	0	0	0	0	0	0	0	0	0	1	0
7	11	10	0	0	0	0	0	0	0	0	0	0	0	1
7	11	11	0	0	0	0	0	0	0	0	0	1	0	0
8	8	3	0	0	0	0	2	0	0	0	0	0	0	0
8	8	8	0	0	0	0	1	1	0	0	0	0	0	0
8	8	11	0	0	0	0	0	2	0	0	0	0	0	0
8	9	7	0	0	0	0	1	0	0	0	0	0	0	0
8	9	10	0	0	0	0	0	1	0	0	0	0	0	0
8	10	7	0	0	0	0	1	0	1	0	0	0	0	0
8	10	8	0	0	0	0	1	0	0	1	0	0	0	0
8	10	10	0	0	0	0	0	1	1	0	0	0	0	0
8	10	11	0	0	0	0	0	1	0	1	0	0	0	0
8	11	8	0	0	0	0	1	0	0	0	0	0	0	0
8	11	11	0	0	0	0	0	1	0	0	0	0	0	0
9	10	9	0	0	0	0	0	0	1	0	0	0	0	0
9	10	10	0	0	0	0	0	0	0	1	0	0	0	0
10	10	9	0	0	0	0	0	0	2	0	0	0	0	0
10	10	10	0	0	0	0	0	0	1	1	0	0	0	0
10	10	11	0	0	0	0	0	0	0	2	0	0	0	0
10	11	10	0	0	0	0	0	0	1	0	0	0	0	0
10	11	11	0	0	0	0	0	0	0	1	0	0	0	0

8.0 Genotype Analyzer Code

8.1 Globals Unit

```
unit Globals;

{
| File:           globals.pas
| Compiles to:    globals.tpu
| Usage:          "uses Globals;"
|
| Author:         James Meyers
|                 Brown University
|
| Start Date:     December 22, 1990
| Last Revision: July 14, 1991
|
| Intent:
}

interface

const
  MAX_GENES         = 2; { DO NOT CHANGE }
  MAX_ALLELES       = 3; { FROM 2 to 9 }
  MAX_CODE_LENGTH   = 10;

  MAX_T_GAMETES     = (MAX_ALLELES*(MAX_ALLELES+1)) div 2;

  NUM_GAMETES       = MAX_ALLELES * MAX_ALLELES;

  MALE              = 1;
  FEMALE            = 2;

  { file strings }

  infile_str:        string[30] = 'm23.pre';
  ceph_hdr_str:      string[30] = 'm23.dat';
  vb_file_str:       string[30] = 'vb_k.out';
  vb_sum_str:        string[30] = 'vb_sum.out';
  vb_nophase_str:    string[30] = 'vb_uk.out';
  obs_exp_file_str: string[30] = 'oe.out';
  fu_file_str:       string[30] = 'out_fu.dat';
  gt_file_str:       string[30] = 'out_gt.dat';
  pd_file_str:       string[30] = 'out_pd.dat';
  debug_str:         string[30] = 'debug.out';

  { variable names for equation output }

  LVar:  string[2] = 'L';
  NVar:  string[2] = 'N';
  T1Var: string[2] = 'T1';
```

```
  T2Var: string[2] = 'T2';
  P1Var: string[2] = 'P1';
  P2Var: string[2] = 'P2';
  P3Var: string[2] = 'P3';
  P4Var: string[2] = 'P4';

  expn: string[10] = 'power';

var
  infile,                { input file with CEPH family information }
  ceph_hdr,              { input file with CEPH header information }
  vb_file,               { output file for phase-known viability information }
  vb_sum,                { output file for total phase-known viability info }
  vb_nophase,            { output file for phase-unknown v. info }
  obs_exp_file,          { output file for observed/expected totals }
  fu_file,               { output file for factor-union output }
  gt_file,               { output file for genotype output }
  pd_file: text;         { output file for pedigree output }
  debug:   text;         { output file for debug information }

  rf_val,
  rm_val: real;          { recombination rate values }

  family_counter: integer;  { number of families processed in an analysis }


{ ------------------------------------------------------------------------ }

procedure error(message:string);
procedure info(message:string);
procedure iswap(var x,y: integer);
procedure bswap(var x,y: byte);
procedure close_files;

{ ------------------------------------------------------------------------ }

implementation

uses MsgBox;

procedure iswap(var x,y: integer);

var
  hold: integer;

begin

  hold:= x;
  x:= y;
  y:= hold

end; { procedure swap }

{ ------------------------------------------------------------------------ }
```

```
procedure bswap(var x,y: byte);

var
  hold: byte;

begin

  hold:= x;
  x:= y;
  y:= hold

end; { procedure swap }

{ ------------------------------------------------------------------------- }

procedure close_files;

begin

  writeln(vb_sum, LVar, ':= ', LVar, ' + ', NVar, ';');

  close(infile);
  close(vb_file);
  close(vb_nophase);
  close(vb_sum);
  close(obs_exp_file);
  close(fu_file);
  close(gt_file);
  close(pd_file);
  close(debug)

end;

{-------------------------------------------------------------------------}

procedure error(message:string);

var
  w: word;

begin

  w:= MessageBox(#3+message, nil, mfError + mfOKButton);
  close_files;
  halt(0)

end; {procedure error}

{-------------------------------------------------------------------------}

procedure info(message:string);

var
```

```
  w: word;
begin
  w:= MessageBox(#3+message, nil, mfInformation + mfOKButton)
end; {procedure error}
{ ------------------------------------------------------------------------ }
function power(x,y:real):real;
begin
  power:= exp(y * ln(x));
end; { function power }
{ ------------------------------------------------------------------------ }
function ipower(x,y:real):integer;
var
  hold: real;
  s:string;
  code, t: integer;
begin
  hold:= exp(y * ln(x));
  str(hold:2:0,s);
  val(s,t,code);
  ipower:= t
end; { function power }
{ ------------------------------------------------------------------------ }
procedure init_io_files;
begin
  assign(infile, infile_str);
  assign(ceph_hdr, ceph_hdr_str);
  assign(vb_file, vb_file_str);
  assign(vb_sum, vb_sum_str);
  assign(vb_nophase, vb_nophase_str);
  assign(obs_exp_file, obs_exp_file_str);
  assign(fu_file, fu_file_str);
  assign(gt_file, gt_file_str);
  assign(pd_file, pd_file_str);
  assign(debug, debug_str)
end; { procedure init_io_files }
```

```
{ ------------------------------------------------------------------------ }

procedure init_r_values;

begin

  rf_val := 0.5;
  rm_val := 0.5

end;

{ ------------------------------------------------------------------------ }

begin

  init_io_files;
  init_r_values

end.
```

8.2 Main Program

```
program Emilie;

{
| File:           Emilie.pas
| Compiles to:    Emilie.exe
| Usage:          Executable
|
| Author:         James Meyers
|                 Brown University
|
| Start Date:     June 15, 1991
| Last Revision:  July 13, 1991
|
| Intent:         To automate multi-locus viability analyses of CEPH families
}

{$M 27000, 0, 655360 }

uses Master;      { The main application object }

var
  Main : TMaster;

{ MAIN CONTROL SEQUENCE }

begin
  Main.Init;
  Main.Run;
  Main.Done
end.
```

8.3 Master Module

```
unit Master;

{
| File:           master.pas
| Compiles to:    master.tpu
| Usage:          'uses master'
|
| Author:         James Meyers
|                 Brown University
|
| Start Date:     June 15, 1991
| Last Revision:  July 14, 1991
|
| Intent:         Main application object definitions
}

interface

uses Globals,     { My global definitions }
     IO,          { My IO routines & preprocessor }
     HeapView,    { My heap monitor }
     MyWindow,    { My window definitions }
     MyDialog,    { My dialog box routines }
     Geno_2_n,    { The Genotype Analyzer }
     MsgBox,      { Simple dialog box routines }
     App,         { Turbo Application objects }
     Objects,     { Turbo basic object definitions }
     Menus,       { Turbo menu objects }
     Drivers,     { Turbo mouse, keyboard, system error handler }
     Views,       { Turbo windowing objects }
     TextView,    { Turbo views for advanced text display }
     Dialogs,     { Turbo dialog box definitions }
     Memory;      { Turbo memory manager }

const

  { New command definitions }

  cmFileOpen       = 200;
  cmNewWin         = 201;
  cmAboutBox       = 202;
  cmModifyInfiles  = 203;
  cmModifyOutfiles = 204;
  cmViewPKEqu      = 205;
  cmViewPUEqu      = 206;
  cmViewPedigree   = 208;
  cmViewFUFile     = 209;
  cmViewGTFile     = 210;
  cmViewHdrFile    = 211;
  cmViewPKSumFile  = 212;
  cmViewOEFile     = 214;
  cmViewInfile     = 215;
```

```
  cmRunAnalysis     = 216;
  cmScreenSize      = 217;
  cmModifyRRates    = 218;
  cmViewDebug       = 219;

  WinCount: integer = 0; { Desktop window counter }

type
  TMaster = object(TApplication)
    constructor Init;
    procedure InitMenuBar; virtual;
    procedure InitStatusLine; virtual;
    procedure NewWindow(FileName: string);
    procedure AboutDialog;
    procedure NotImplemented(S:string);
    procedure InfoBox(S:string);
    procedure RunAnalysis;
    procedure HandleEvent(var Event: TEvent); virtual;
    procedure Idle; virtual;
    procedure OutOfMemory; virtual;
  private
    HeapViewer: PHeapView;
  end;

implementation

{ ==== (START) METHODS for TMaster ================================================ }

{ ---------------------------------------------------------------------------- }

constructor TMaster.Init;

begin
  TApplication.Init;
  InitDialogData;
end; { constructor TMaster.Init }

{ ---------------------------------------------------------------------------- }

procedure TMaster.InitMenuBar;

var
  R: TRect;

begin
  GetExtent(R);          { set R to full-screen coordinates }
  R.B.Y := R.A.Y + 1;    { set bottom to one line from top }

  { Menu Definition }

  MenuBar := New(PMenuBar, Init(R, NewMenu(

    NewSubMenu('~'#240'~', hcNoContext, NewMenu(
      NewItem('~A~bout...', '',0, cmAboutBox, hcNoContext,
```

```
      nil)),

    NewSubMenu('~A~nalysis', hcNoContext, NewMenu(
      NewItem('~R~un analysis', 'F9', kbF9, cmRunAnalysis, hcNoContext,
      nil)),

    NewSubMenu('~D~efaults', hcNoContext, NewMenu(
      NewItem('~I~nput Files', '', 0, cmModifyInfiles, hcNoContext,
      NewItem('~O~utput Files', '', 0, cmModifyOutfiles, hcNoContext,
      NewItem('~R~ecombination Rates', '', 0, cmModifyRRates, hcNocontext,
      NewItem('~S~creen Size', '', 0, cmScreenSize, hcNoContext,
      nil))))),

    NewSubMenu('~O~utput', hcNoContext, NewMenu(
      NewItem('Phase ~K~nown eqs.', '', 0, cmViewPKEqu, hcNoContext,
      NewItem('Phase ~U~nknown eqs.', '', 0, cmViewPUEqu, hcNoContext,
      NewItem('~S~um Viabiliy eq.', '', 0, cmViewPKSumFile, hcNoContext,
      NewLine(
      NewItem('~O~bserved/Expected', '', 0, cmViewOEFile, hcNoContext,
      NewLine(
      NewItem('~P~edigree output', '', 0, cmViewPedigree, hcNocontext,
      NewItem('~F~actor-Union file', '', 0, cmViewFUFile, hcNoContext,
      NewItem('~G~enotype file', '', 0, cmViewGTFile, hcNoContext,
      NewLine(
      NewItem('CEPH ~D~ata file', '', 0, cmViewHdrFile, hcNoContext,
      NewItem('CEPH P~e~digree file', '', 0, cmViewInfile, hcNoContext,
      nil))))))))))))),

    NewSubMenu('~W~indow', hcNoContext, NewMenu(
      NewItem('~S~ize/Move', 'Ctrl-F5', kbCtrlF5, cmResize, hcNoContext,
      NewItem('~Z~oom', 'F5', kbF5, cmZoom, hcNoContext,
      NewItem('~N~ext', 'F6', kbF6, cmNext, hcNoContext,
      NewItem('~P~revious', 'Shift-F6', kbShiftF6, cmPrev, hcNoContext,
      NewItem('~T~ile', '', 0, cmTile, hcNoContext,
      NewItem('~C~ascade', '', 0, cmCascade, hcNoContext,
      nil))))))),
    nil)))))
  )))

end; { procedure TMaster.InitMenuBar }

{ ---------------------------------------------------------------------- }

procedure TMaster.InitStatusLine;

var
  R: TRect;

begin
  GetExtent(R);          { set R to full-screen coordinates }
  R.A.Y := R.B.Y - 1;    { set top to one line from bottom }
  StatusLine := New(PstatusLine, Init(R,
    NewStatusDef(0, $FFFF,
      NewStatusKey('~Alt-X~ Exit', kbAltX, cmQuit,
```

```
      NewStatusKey('~Alt-F3~ Close', kbAltF3, cmClose,
      NewStatusKey('~F5~ Zoom', kbF5, cmZoom,
      NewStatusKey('~F6~ Next', kbF6, cmNext,
      NewStatusKey('~F9~ Run Analysis', kbF9, cmRunAnalysis,
      NewStatusKey('~F10~ Debug Output', kbF10, cmViewDebug,
      nil))))))), { no more keys }
    nil) { no more defs }
  ));

  dec(R.A.Y,2);
  R.A.X := R.B.X - 10;
  HeapViewer:= New(PHeapView,Init(R));
  DeskTop^.Insert(HeapViewer);

end; { procedure TMaster.InitStatusLine }

{ ------------------------------------------------------------------------ }

procedure TMaster.OutOfMemory;

var
  w: word;

begin
  w:= MessageBox(#3'Insufficient Memory', nil, mfError + mfOKButton)
end;

{ ------------------------------------------------------------------------ }

procedure TMaster.NewWindow(FileName: string);

var
  R: TRect;

begin
  inc(WinCount);
  R.Assign(0,0,40,18);  { set initial size and position }
  R.Move(Random(40), Random(6));  { move to a random position }
  if LowMemory then
    OutOfMemory
  else
    OpenNewFileWindow(R, WinCount, FileName);

end; { procedure TMaster.NewWindow }

{ ------------------------------------------------------------------------ }

procedure TMaster.AboutDialog;

const
  XMIN = 20;
  XMAX = 60;
  YMIN = 6;
  YMAX = 19;
```

```
var
  Dialog: PDialog;
  R: TRect;
  Control: word;
  TextX, TextY: integer;
  B: PStaticText;
  S: string;

begin
  R.Assign(XMIN,YMIN,XMAX,YMAX);
  Dialog := New(PDialog, Init(R, 'About'));

  with Dialog^ do
    begin
      R.Assign(1,1,39,2);

      S := #3'EMILIE';
      inc(R.A.Y);
      inc(R.B.Y);
      B := New(PStaticText, Init(R,S));
      Insert(B);

      S := #3'Version  0.4';
      inc(R.A.Y);
      inc(R.B.Y);
      B := New(PStaticText, Init(R,S));
      Insert(B);

      S := #3'2-Gene N-Allele Viability Analyzer';
      inc(R.A.Y,2);
      inc(R.B.Y,2);
      B := New(PStaticText, Init(R,S));
      Insert(B);

      S := #3'Lisa Brooks & James Meyers';
      inc(R.A.Y,2);
      inc(R.B.Y,2);
      B := New(PStaticText, Init(R,S));
      Insert(B);

      S := #3'Brown University';
      inc(R.A.Y,1);
      inc(R.B.Y,1);
      B := New(PStaticText, Init(R,S));
      Insert(B);

      TextX := (XMAX-XMIN) div 2 - 5;
      TextY := (YMAX-YMIN) - 3;
      R.Assign(TextX,TextY,TextX+10,TextY+2);
      Insert(New(PButton, Init(R, '~D~ismiss', cmOK, bfDefault)));
    end;

  Control := DeskTop^.ExecView(Dialog);
```

```
  Dispose(Dialog, Done)

end; { procedure TMaster.AboutDialog }

{ ---------------------------------------------------------------------- }

procedure TMaster.NotImplemented(S:string);

var
  w: word;

begin
  w:= MessageBox(#3+S, nil, mfError + mfOKButton)
end; { procedure TMaster.NotImplemented }

{ ---------------------------------------------------------------------- }

procedure TMaster.InfoBox(S:string);

var
  w: word;

begin
  w:= MessageBox(#3+S, nil, mfInformation + mfOKButton)
end; { procedure TMaster.InfoBox }

{ ---------------------------------------------------------------------- }

procedure TMaster.RunAnalysis;

var
  p: pointer;
  s: string;

begin
  reset_files;

  mark(p);

  init_geno_analyzer;

  family_counter := 0;

  while not eof(infile) do
  begin
     process_next_family;
      genotype_analysis;
      inc(family_counter)
    end;
  output_totals;
  close_files;

  release(p);
```

```
  str(family_counter, s);
  s:= s + ' Families Processed';

  InfoBox('Analysis Complete' + #13 + #13 + #3 + s);

end; { procedure TMaster.RunAnalysis }

{ ---------------------------------------------------------------------- }

procedure TMaster.Idle;

begin
  HeapViewer^.Update;
end; { procedure TMaster.Idle }

{ ---------------------------------------------------------------------- }

procedure TMaster.HandleEvent(var Event: TEvent);

var
  R: TRect;
  w: word;
begin
  GetExtent(R);
  dec(R.B.Y,2);
  TApplication.HandleEvent(Event);
  if Event.What = evCommand then
    begin
      case Event.Command of
        cmNewWin:          NewWindow('jeanne.pas');
        cmFileOpen:        NotImplemented('Open File');
        cmAboutBox:        AboutDialog;
        cmModifyInfiles:   SetInputFileDefaults;
        cmModifyOutfiles:  SetOutputFileDefaults;
        cmModifyRRates:    SetRecombinationRates;
        cmViewPKEqu:       NewWindow(vb_file_str);
        cmViewPUEqu:       NewWindow(vb_nophase_str);
        cmViewPedigree:    NewWindow(pd_file_str);
        cmViewFUFile:      NewWindow(fu_file_str);
        cmViewGTFile:      NewWindow(gt_file_str);
        cmViewHdrFile:     NewWindow(ceph_hdr_str);
        cmViewOEFile:      NewWindow(obs_exp_file_str);
        cmViewPKSumFile:   NewWindow(vb_sum_str);
        cmViewInfile:      NewWindow(infile_str);
        cmViewDebug:       NewWindow(debug_str);
        cmRunAnalysis:     RunAnalysis;
        cmScreenSize:      begin

                             w:= SetScreenSize;
                             if (w = 0) then
                              SetScreenMode(ScreenModeArray[1])
                             else
                              SetScreenMode(ScreenModeArray[0]);
                             Dispose(HeapViewer, Done);
```

```
                                SetScreenMode(ScreenModeArray[ScreenDD]);
                                GetExtent(R);
                                R.A.Y:= R.B.Y - 3;
                                R.A.X:= R.B.X - 10;
                                HeapViewer:= New(PHeapView,Init(R));
                                DeskTop^.Insert(HeapViewer);

                              end;
        cmTile:              DeskTop^.Tile(R);
        cmCascade:           DeskTop^.Cascade(R);
      else
        exit;
    end;
    ClearEvent(Event);
  end;
end; { procedure TMaster.HandleEvent }

{ ==== (END) methods for TMaster ========================================== }

begin
end.
```

8.4 CEPH Header Processor Module

```
unit p_cephhdr;

{
| File:          p_cephhdr.pas
| Compiles to:   p_cephhdr.tpu
| Usage:         "uses p_cephhdr;"
|
| Author:        James Meyers
|                Brown University
|
| Start Date:    April 27, 1991
| Last Revision: June 27, 1991
|
| Intent:        Provides routine to process a CEPH header file.
}

{ --------------------------- INTERFACE --------------------------- }

interface

uses Globals;

type

  codes_table = array[1..MAX_ALLELES] of array[1..MAX_ALLELES]
                of string[MAX_CODE_LENGTH];

  gene_information = record
    binary_factors: byte;
```

```
    num_alleles    : byte;
    text_info      : string;
    gene_freq      : array[1..MAX_ALLELES] of real;
    num_factors    : integer;
    codes          : codes_table;
  end;

var
  num_loci       : integer;
  risk_locus     : integer;
  sex_linked     : integer;
  program_id     : integer;
  mut_locus      : integer;
  mut_male       : real;
  mut_female     : real;
  hap_freq       : integer;
  gene_order     : array[1..MAX_GENES] of integer;
  gene_info      : array[1..MAX_GENES] of gene_information;
  sex_diff       : byte;
  interference   : byte;

  {recombination values}

  r_varied    : integer;
  increment   : real;
  final_val   : real;

{ public routines }

procedure process_ceph_header_file;
function get_factor_union_string(gene, allele: byte): string;
procedure dump_header;

{ ------------------------- IMPLEMENTATION ----------------------- }

implementation

procedure init_header_data;

var
  i,j,k,l: byte;

begin
  num_loci       := 0;
  risk_locus     := 0;
  sex_linked     := 0;
  program_id     := 0;
  mut_locus      := 0;
  mut_male       := 0.0;
  mut_female     := 0.0;
  hap_freq       := 0;
  sex_diff       := 0;
  interference   := 0;
  r_varied       := 0;
```

```
  increment      := 0.0;
  final_val      := 0;

  for i:=1 to MAX_GENES do
    gene_order[i] := 0;

  for i:=1 to MAX_GENES do
    with gene_info[i] do
      begin

        binary_factors := 0;
        num_alleles    := 0;
        text_info      := '';
        num_factors    := 0;

        for j:=1 to MAX_ALLELES do
          gene_freq[j] := 0.0;

        for j:=1 to MAX_ALLELES do
          for k:=1 to MAX_ALLELES do
            codes[j][k]:='';

      end

end; { procedure init_header_data }

{ ------------------------------------------------------------------ }

procedure init_codes(var t:codes_table; len: byte);

var
  i,j,k: byte;

begin
  for i:=1 to MAX_ALLELES do
    for j:=1 to MAX_ALLELES do
      begin
        t[i][j]:= '';
        for k:=1 to len do
          t[i][j]:= t[i][j] + 'X'
      end
end; { procedure init_codes }

{ ------------------------------------------------------------------ }

procedure read_gene(i:integer);

var
  j,c:byte;
  ch: char;

begin
  with gene_info[i] do
    begin
```

```
      readln(ceph_hdr, binary_factors, num_alleles, text_info);

      for j:=1 to num_alleles do
        read(ceph_hdr, gene_freq[j]);
      readln(ceph_hdr);

      readln(ceph_hdr, num_factors);

      init_codes(codes,num_factors);

      for j:=1 to num_alleles do
        begin
          c:=1;
          while not(eoln(ceph_hdr)) do
            begin
              read(ceph_hdr, ch);
              if (ch<>' ') then
                begin
                  codes[j][j][c]:= ch;
                  inc(c)
                end
            end;
          readln(ceph_hdr)
        end;

      { skip blank line }

      readln(ceph_hdr);

    end {with}
end; {procedure read_gene}

{ ------------------------------------------------------------------ }

procedure combine(var t: codes_table; rs,cs,rd,cd: byte);

var
  i: byte;

begin
  for i:=1 to length(t[rs][cs]) do
    if (t[rs][rs][i] = '1') or (t[cs][cs][i] = '1') then
      t[rd][cd][i]:= '1'
    else if (t[rs][rs][i] = '0') and (t[cs][cs][i] = '0') then
      t[rd][cd][i]:= '0'
    else if (t[rs][rs][i] = ' ') and (t[cs][cs][i] = ' ') then
      t[rd][cd][i]:= ' '

end;

{ ------------------------------------------------------------------ }

procedure complete_codes;
```

```
var
  i,j,k: byte;

begin
  for i:=1 to num_loci do
    begin
      for j:=1 to MAX_ALLELES do
        for k:=1 to MAX_ALLELES do
          begin
            if (k<>j) then
              if (k<j) then
                combine(gene_info[i].codes,j,k,j,k)
              else
                combine(gene_info[i].codes,k,j,j,k)
          end
    end
end;

{ ------------------------------------------------------------------- }

procedure process_ceph_header_file;

var
  i     : integer;
  s1, s2: string;

begin
  init_header_data;

  reset(ceph_hdr);

  readln(ceph_hdr, num_loci, risk_locus, sex_linked, program_id);
  readln(ceph_hdr, mut_locus, mut_male, mut_female, hap_freq);

  if (num_loci>MAX_GENES) then
    begin

      str(MAX_GENES, s1);
      str(num_loci, s2);
      info('Only ' + s1 + ' of the ' + s2 + ' loci in ' + infile_str +
           ' will be processed.');
      num_loci:= MAX_GENES

    end;

  for i:=1 to num_loci do
    read(ceph_hdr, gene_order[i]);
  readln(ceph_hdr);

  { skip blank line }

  readln(ceph_hdr);

  for i:=1 to num_loci do
```

```
    read_gene(i);

  readln(ceph_hdr, sex_diff, interference);

  readln(ceph_hdr);

  readln(ceph_hdr, r_varied, increment, final_val);

  close(ceph_hdr);

  complete_codes;
end;

{ ------------------------------------------------------------------ }

function get_factor_union_string(gene,allele: byte):string;

begin
end; { function get_factor_union_string }

{ ------------------------------------------------------------------ }

procedure dump_header;

var
  i,j,k:byte;

begin

  writeln(num_loci,' ',risk_locus,' ',sex_linked,' ',program_id);
  writeln(mut_locus,' ',mut_male:4:6,' ',mut_female:4:6,' ',hap_freq);

  for i:=1 to num_loci do
    write(gene_order[i],' ');

  writeln;
  writeln;

  for i:=1 to num_loci do
    begin
      with gene_info[i] do
        begin
          writeln(binary_factors,' ',num_alleles,' ',text_info);
          for j:=1 to num_alleles do
            write(gene_freq[j]:4:6,' ');
          writeln;
          writeln(num_factors);
          for j:=1 to max_alleles do
            begin
              for k:=1 to max_alleles do
                write(codes[j][k],' ');
              writeln
            end
        end; {with}
```

```
      writeln;
    end;

  writeln(sex_diff,' ',interference);

  writeln(rm_val:4:6,' ',rf_val:4:6);

  writeln(r_varied,' ',increment:4:6,' ',final_val:4:6);


end; {procedure dump_header}

{ ------------------------------------------------------------------ }

begin
end.
```

8.5 I/O Module

```
unit IO;

{
| File:          io.pas
| Compiles to:   io.tpu
| Usage:         "uses io;"
|
| Author:        James Meyers
|                Brown University
|
| Start Date:    April 27, 1990
| Last Revision: July 21, 1991
|
| Intent:        Reads the CEPH .pre file one family at a time. The current
|                family is stored in the input buffer.
}

interface

uses Globals,
     P_CEPHhdr;

const
  MAX_FAMILY_MEMBERS = 22;
  MAX_BUFFER_WIDTH   = 30;

  {--------  Indicies into the input buffer ---------}

  FAMILY_NUM   = 1;
  ID_NUM       = 2;
  FATHER       = 3;
  MOTHER       = 4;
  FIRST_KID    = 5;
  NEXT_PAT_SIB = 6;
```

```
  NEXT_MAT_SIB = 7;
  SEX          = 8;
  PRO_BAND     = 9;
  FIRST_GENE   = 10;

{-------------------------------------------------------------------------}

type
  buffer_type =
    array [1..MAX_FAMILY_MEMBERS] of array [1..MAX_BUFFER_WIDTH] of integer;

  {
  | A person_type variable stores information about a family member which
  | is not directly represented in the input file/buffer. The information
  | must be calculated based on interrelationships among family members.
  }

  person_type = record
    level:byte;
    spouse:byte;
    siblings:array[1..MAX_FAMILY_MEMBERS] of boolean;
  end; {record}

  family_type = array[1..MAX_FAMILY_MEMBERS] of person_type;

  gene_type = array[1..2] of byte;

{-------------------------------------------------------------------------}

var
  buffer : buffer_type;  {contains current CEPH family data}
  buffer_lines : byte;   {lines of valid data in buffer}
  family : family_type;  {current family information}

procedure reset_files;
procedure process_next_family;
procedure dump;
procedure dump_family;

implementation

var
  scan : integer;                    {buffers the input file buffer}
  num_genes : byte;                  {number of genes being processed}

{-------------------------------------------------------------------------}

procedure reset_files;

var
  name:string;

begin
```

```
  process_ceph_header_file;

  reset(infile);
  rewrite(vb_file);
  rewrite(vb_nophase);
  rewrite(vb_sum);
  rewrite(obs_exp_file);
  rewrite(fu_file);
  rewrite(gt_file);
  rewrite(pd_file);
  rewrite(debug);

  read(infile,scan);

  writeln(vb_sum, 'All Families');
  writeln(vb_sum, 'log L (phase known) =');
  writeln(vb_sum, LVar,' := 0;');
  writeln(vb_sum, NVar, ' := 0;');

end; { procedure reset_files }

{-------------------------------------------------------------------------------}

procedure null_buffer;

var
  i, j:integer;

begin

  for i:=1 to MAX_FAMILY_MEMBERS do
    for j:=1 to MAX_BUFFER_WIDTH do
      buffer[i][j]:=-1

end; {procedure null_buffer}

{-------------------------------------------------------------------------------}

procedure fill_buffer;

var
  i, {row}
  j  {col} :integer;
  done     :boolean;
  temp     :integer;

begin

  done:=false;
  i:=1;
  j:=1;
  temp:=scan;

  buffer[i][j]:=scan;
```

```
  inc(j);

  if (temp<>scan) and (scan<>-1) then
    done:=true

  else
    begin
      while (not done) and (not eof(infile)) do
        begin
          while (not done) and (not eoln(infile)) do
            begin

              read(infile,buffer[i][j]);
              if (j=1) and (buffer[i][j]<>scan) then
                begin

                  done:=true;
                  scan:=buffer[i][j];
                  buffer[i][j]:=-9;
                  buffer_lines:=i-1

                end;

              inc(j)

            end; {eoln}
          inc(i);
          j:=1;

          if not done then
            readln(infile);

          if eof(infile) then
            buffer_lines:=i-1;

        end {not done}

    end {else}

end; {procedure fill_buffer}

{-------------------------------------------------------------------------}

function fu_to_gt(gene_number:byte; fu_code:string):string;

var
  i, j :byte;
  s, t :string;
  hold,
  ans  : string;

begin

  hold:='';
```

```
  ans:='';

  for i:=1 to MAX_ALLELES do
    for j:=1 to MAX_ALLELES do
      if (ans='') then
        begin
          s:= gene_info[gene_number].codes[i][j];
          if s = fu_code then
            begin
              str(i,t);
              hold:= hold + t;
              str(j,t);
              hold:= hold + t;
              ans:= hold;
            end
        end;

      if (ans='') then
        ans:= '00';

      fu_to_gt:= ans

end; {function fu_to_gt}

{-------------------------------------------------------------------------}

procedure convert_codes;

var
  i, j, k, m, n: byte;
  temp          :array[1..MAX_GENES] of string;
  fu_code, s,
  buf_string    : string;
  gene_number   : byte;
  code          : integer;

begin

  for i:=1 to buffer_lines do
    begin

      buf_string:= '';
      gene_number:= 1;
      j:= FIRST_GENE;

      while (j <= MAX_BUFFER_WIDTH) and (gene_number <= num_loci) do
        begin

          fu_code:='';
          for k:=1 to (gene_info[gene_number].num_factors) do
            begin
              str(buffer[i][j+k-1],s);
              fu_code:= fu_code + s
            end;
```

```
          temp[gene_number]:=fu_to_gt(gene_number,fu_code);
          buf_string:= buf_string + temp[gene_number];
          inc(gene_number);
          inc(j,k)

        end; { while }

      n:= 1;

      for m:=FIRST_GENE to FIRST_GENE + length(buf_string) - 1 do
        begin

          val(buf_string[n],buffer[i][m],code);
          inc(n)

        end; { for m }

      for n:=m+1 to MAX_BUFFER_WIDTH do
        buffer[i][n]:= -1

    end { for i }

end; { procedure convert_codes }

{-------------------------------------------------------------------------}

procedure set_gene(person,gene,allele1,allele2:byte);

var
  j,k :byte;

begin
  if gene=1 then
    j:=FIRST_GENE
  else
    j:=FIRST_GENE+2;

  buffer[person][j]:=allele1;
  buffer[person][j+1]:=allele2;

  if (buffer[person][j] > buffer[person][j+1]) then
    begin

      k:= buffer[person][j+1];
      buffer[person][j+1]:= buffer[person][j];
      buffer[person][j]:= k

    end
end;

{-------------------------------------------------------------------------}

procedure get_gene(person,gene:byte; var g:gene_type);
```

```
var
  j:byte;

begin

  if gene=1 then
    j:=FIRST_GENE
  else
    j:=FIRST_GENE+2;

  g[1]:=buffer[person][j];
  g[2]:=buffer[person][j+1];

  if (g[2] > g[1]) then
    begin

      j:= g[1];
      g[1]:= g[2];
      g[2]:= j

    end

end; {function get_gene}

{-------------------------------------------------------------------------}

function uk(g:gene_type):boolean;

begin

  if (g[1]=0) or (g[2]=0) then
    uk:= TRUE

  else
    uk:= FALSE

end; { function uk }

{ ----------------------------------------------------------------------- }

function homo(g: gene_type): boolean;

begin

  if (g[1] = g[2]) then
    homo:= TRUE

  else
    homo:= FALSE

end; { function homo }

{ ----------------------------------------------------------------------- }
```

```
procedure make_inference_1;

var
  i, j, k,
  m, n     : byte;
  sp       : byte;
  g        : array[1..2] of array[1..2] of gene_type;
  tgene    : gene_type;

begin

  for i:=1 to buffer_lines do
    begin

      sp:= family[i].spouse;

      if (family[i].level <> 2) and (sp <> 0) then
        begin

          get_gene(i, 1,g[1][1]);
          get_gene(sp,1,g[2][1]);
          get_gene(i, 2,g[1][2]);
          get_gene(sp,2,g[2][2]);

          for j:=1 to buffer_lines do
            if (buffer[j][MOTHER] = i) or (buffer[j][FATHER] = i) then
              begin

                if (g[1][1][1]=g[1][1][2]) and (g[2][1][1]=g[2][1][2]) then
                  if (not uk(g[1][1])) and (not uk(g[2][1])) then
                    set_gene(j,1,g[1][1][1],g[2][1][1]);

                if (g[1][2][1]=g[1][2][2]) and (g[2][2][1]=g[2][2][2]) then
                  if (not uk(g[1][2])) and (not uk(g[2][2])) then
                    set_gene(j,2,g[1][2][1],g[2][2][1])

              end { if MOTHER... }

        end { if level and spouse }

    end { for i }

end; { procedure make_inference_1 }

{ ------------------------------------------------------------------------ }

procedure make_inference_2(known: gene_type; var unknown: gene_type;
                           ik, iuk: byte; locus: byte);

{
| Known parent is a homozygote
}

var
```

```
  i   : byte;
  g   : gene_type;
  hom: boolean;

begin

  hom:= FALSE;

  for i:=1 to buffer_lines do
    begin

      if (buffer[i][MOTHER] = ik) or (buffer[i][FATHER] = ik) then
        begin

          get_gene(i, locus, g);
          if homo(g) then
            if (g[1] = known[1]) then
              hom:= TRUE

        end

    end;

  if not(hom) then
    exit;

  for i:=1 to buffer_lines do
    begin

      if (buffer[i][MOTHER] = ik) or (buffer[i][FATHER] = ik) then
        begin

          get_gene(i, locus, g);

          if (g[1] <> known[1]) or (g[2] <> known[1]) then
            begin

              unknown[1]:= g[1];
              unknown[2]:= g[2];

              if (unknown[1] > unknown[2]) then
                bswap(unknown[1], unknown[2]);

              set_gene(iuk, locus, unknown[1], unknown[2]);

              exit

            end

        end

    end

end; { procedure make_inference_2 }
```

```
{ ------------------------------------------------------------------------ }

procedure make_inference_3(known: gene_type; var unknown: gene_type;
                           ik, iuk: byte; locus: byte);

{
| Known parent is a heterozygote
}

var
  i       : byte;
  g       : gene_type;
  allele1,
  allele2 : byte;

begin

  { two homs }

  allele1:= 0;
  allele2:= 0;

  for i:=1 to buffer_lines do
    begin

      if (buffer[i][MOTHER] = ik) or (buffer[i][FATHER] = ik) then
        begin

          get_gene(i, locus, g);

          if (g[1] <> 0) and (g[2] <> 0) then
            begin

              if homo(g) then
                allele1:= g[1]

            end

        end

    end;

  for i:=1 to buffer_lines do
    begin

      if (buffer[i][MOTHER] = ik) or (buffer[i][FATHER] = ik) then
        begin

          get_gene(i, locus, g);

          if (g[1] <> 0) and (g[2] <> 0) then
            begin
```

```
            if homo(g) then
              if (g[1] <> allele1) then
                allele2:= g[1]

          end

      end

  end;

if (allele1 <> 0) and (allele2 <> 0) then
  begin

    if (allele1 > allele2) then
      bswap(allele1, allele2);

    set_gene(iuk, locus, allele1, allele2);

    exit

  end;

{ one hom of known parent allele, and an allele not from known parent }

allele1:= 0;
allele2:= 0;

{ allele 1 }

for i:=1 to buffer_lines do
  begin

    if (buffer[i][MOTHER] = ik) or (buffer[i][FATHER] = ik) then
      begin

        get_gene(i, locus, g);

        if (g[1] <> 0) and (g[2] <> 0) then
          begin

            if (g[1] <> known[1]) and (g[1] <> known[2]) then
              allele1:= g[1]
            else if (g[2] <> known[1]) and (g[2] <> known[2]) then
              allele1:= g[2];

          end

      end

  end;

{ allele 2 }

for i:=1 to buffer_lines do
```

```
  begin

    if (buffer[i][MOTHER] = ik) or (buffer[i][FATHER] = ik) then
      begin

        get_gene(i, locus, g);

        if (g[1] <> 0) and (g[2] <> 0) then
          begin

            if (g[1] <> known[1]) and (g[1] <> known[2]) then
              if (g[1] <> allele1) then
                allele2:= g[1]
            else if (g[2] <> known[1]) and (g[2] <> known[2]) then
              if (g[2] <> allele1) then
                allele2:= g[2];

          end

      end

  end;

if (allele1 = 0) and (allele2 = 0) then
  exit;

if (allele1 <> 0) and (allele2 <> 0) then
  begin

    if (allele1 > allele2) then
      bswap(allele1, allele2);

    set_gene(iuk, locus, allele1, allele2);

    exit

  end;

{ homo }

if (allele1 = 0) then
  allele1:= allele2;

for i:=1 to buffer_lines do
  begin

    if (buffer[i][MOTHER] = ik) or (buffer[i][FATHER] = ik) then
      begin

        get_gene(i, locus, g);

        if (g[1] <> 0) and (g[2] <> 0) then
          begin
```

```
            if homo(g) then
              if (g[1] = known[1]) or (g[1] = known[2]) then
                begin

                  if (g[1] > allele1) then
                    bswap(g[1], allele1);

                  set_gene(iuk, locus, g[1], allele1);

                  exit

                end

            end

        end

    end;

end; { procedure make_inference_3 }

{ ------------------------------------------------------------------------ }

procedure make_inference_4(var p1, p2: gene_type; i1, i2: byte; locus: byte);

{
| Both parents unknown
}

var
  i,
  allele1,
  allele2 : byte;
  g       : gene_type;

begin

  { two homs }

  allele1:= 0;
  allele2:= 0;

  for i:=1 to buffer_lines do
    begin

      if (buffer[i][MOTHER] = i1) or (buffer[i][FATHER] = i1) then
        begin

          get_gene(i, locus, g);

          if (g[1] <> 0) and (g[2] <> 0) then
            begin

              if homo(g) then
```

```
              allele1:= g[1]

            end

        end

    end;

  for i:=1 to buffer_lines do
    begin

      if (buffer[i][MOTHER] = i1) or (buffer[i][FATHER] = i1) then
        begin

          get_gene(i, locus, g);

          if (g[1] <> 0) and (g[2] <> 0) then
            begin

              if homo(g) then
                if (g[1] <> allele1) then
                  allele2:= g[1]

            end

        end

    end;

  if (allele1 <> 0) and (allele2 <> 0) then
    begin

      if (allele1 > allele2) then
        bswap(allele1, allele2);

      set_gene(i1, locus, allele1, allele2);
      set_gene(i2, locus, allele1, allele2);

    end;


end; { procedure make_inference_4 }

{ ------------------------------------------------------------------------ }

procedure make_parent_inferences;

var
  i, j   : byte;
  ind,
  spouse : gene_type;

begin
```

```
  for i:=1 to buffer_lines do
    if (family[i].level <> 2) and (family[i].spouse <> 0) then
      for j:=1 to NUM_GENES do
        begin

          get_gene(i, j, ind);
          get_gene(family[i].spouse, j, spouse);

          if uk(ind) then
            begin

              if uk(spouse) then
                make_inference_4(ind, spouse, i, family[i].spouse, j)

              else if homo(spouse) then
                make_inference_2(spouse, ind, family[i].spouse, i, j)

              else
                make_inference_3(spouse, ind, family[i].spouse, i, j)

            end

          else if uk(spouse) then
            begin

              if homo(ind) then
                make_inference_2(ind, spouse, i, family[i].spouse, j)

              else
                make_inference_3(ind, spouse, i, family[i].spouse, j)

            end

        end

end; { procedure make_parent_inferences }

{ ------------------------------------------------------------------------ }

procedure append_fu_file;

var
  i,j:byte;
  done:boolean;

begin
  done:=false;
  i:=1;
  j:=1;
  while (i<=MAX_FAMILY_MEMBERS) and (not done) do
    begin
      while (j<=MAX_BUFFER_WIDTH) and (not done) do
        begin
          if (j=1) and (buffer[i][j]<0) then
```

```
            done:=true
          else
            begin
              if (buffer[i][j]>=0) and (not(j in [5,6,7,9])) then
                write(fu_file,buffer[i][j]:3);
              inc(j)
            end;
        end;
      writeln(fu_file);
      inc(i);
      j:=1
    end;
end; {procedure append_gt_file}

{-------------------------------------------------------------------------}

procedure append_gt_file;

var
  i,j:byte;
  done:boolean;

begin
  done:=false;
  i:=1;
  j:=1;
  while (i<=MAX_FAMILY_MEMBERS) and (not done) do
    begin
      while (j<=MAX_BUFFER_WIDTH) and (not done) do
        begin
          if (j=1) and (buffer[i][j]<0) then
            done:=true
          else
            begin
              if (buffer[i][j]>=0) and (not(j in [5,6,7,9])) then
                write(gt_file,buffer[i][j]:3);
              inc(j)
            end;
        end;
      writeln(gt_file);
      inc(i);
      j:=1
    end;
end; {procedure append_gt_file}

{=========================================================================}

{                      PEDIGREE CONSTRUCTION ROUTINES                     }

{-------------------------------------------------------------------------}

procedure init_family;

var
```

```
  i,j:integer;

begin
     for i:=1 to MAX_FAMILY_MEMBERS do
         with family[i] do
          begin
             spouse:=0;
             level:=2;
             for j:=1 to MAX_FAMILY_MEMBERS do
              siblings[j]:=false
              end {with}
end; {procedure init_pedigree}

{-------------------------------------------------------------------------}

procedure assign_levels;

var
  i,j:byte;
  level_0:array[1..4] of byte;
  level_0c:byte;
  level_1:array[1..MAX_FAMILY_MEMBERS] of byte;
  level_1c:byte;
  done:boolean;

begin
  level_0c:=0;
  level_1c:=0;
  for i:=1 to 4 do
    level_0[i]:=99;
  for i:=1 to MAX_FAMILY_MEMBERS do
    level_1[i]:=99;
  for i:=1 to buffer_lines do
    if (buffer[i][MOTHER]=0) and (buffer[i][FATHER]=0) then
      begin
        family[i].level:=0;
        inc(level_0c);
        level_0[level_0c]:=buffer[i][ID_NUM]
      end;
  for i:=1 to buffer_lines do
    begin
      j:=1;
      done:=false;
      while (not done) and (j<=4) do
        begin
          if (buffer[i][MOTHER] = level_0[j]) or
            (buffer[i][FATHER] = level_0[j]) then
             begin
               family[i].level:=1;
               inc(level_1c);
               level_1[level_1c]:=buffer[i][ID_NUM];
               done:=true;
             end;
          inc(j)
```

```
        end
    end;
end; {procedure init_pedigree}

{-------------------------------------------------------------------------}

procedure link_couples;

var
  i,j:byte;
  mom, dad:byte;

begin
  for i:=1 to buffer_lines do
    begin
      if family[i].level > 0 then
        begin
          mom:=buffer[i][MOTHER];
          dad:=buffer[i][FATHER];
          family[mom].spouse:=dad;
          family[dad].spouse:=mom
        end
    end
end; {procedure link_couples}

{-------------------------------------------------------------------------}

procedure link_siblings;

var
  i,j:byte;

begin
  for i:=1 to buffer_lines do
    for j:=1 to buffer_lines do
      if buffer[i][MOTHER] = buffer[j][MOTHER] then
        family[i].siblings[j]:=true;
end; {procedure link_siblings}

{-------------------------------------------------------------------------}

procedure construct_pedigree;

begin
  init_family;
  assign_levels;
  link_couples;
  link_siblings;
end; {procedure construct_pedigree}

{-------------------------------------------------------------------------}

{=========================================================================}
{                        PEDIGREE OUTPUT ROUTINES                         }
```

```
{=============================================================================}

procedure write_genes(id:byte);

var
  i:byte;

begin
  i:=10;
  while i<(10+(2*num_loci)) do
    begin
      write(pd_file,'(',buffer[id][i],'|');
      inc(i);
      write(pd_file,buffer[id][i],')  ');
      inc(i);
    end;
end; {procedure write_genes}

{-----------------------------------------------------------------------------}

procedure write_pedigree;

var i:byte;

begin
  writeln(pd_file,'FAMILY ',buffer[1][1],' has ',buffer_lines,' members');
  writeln(pd_file);
  write(pd_file,'level 0: ');
  for i:=1 to buffer_lines do
    if family[i].level=0 then
      begin
        write(pd_file,buffer[i][ID_NUM]:2,'  ');
        write_genes(i);
        writeln(pd_file);
        write(pd_file,'          ');
      end;
  writeln(pd_file);

  write(pd_file,'level 1: ');
  for i:=1 to buffer_lines do
    if family[i].level=1 then
      begin
        write(pd_file,buffer[i][ID_NUM]:2,'  ');
        write_genes(i);
        writeln(pd_file);
        write(pd_file,'          ');
      end;
  writeln(pd_file);

  write(pd_file,'level 2: ');
  for i:=1 to buffer_lines do
    if family[i].level=2 then
      begin
        write(pd_file,buffer[i][ID_NUM]:2,'  ');
```

```
        write_genes(i);
        writeln(pd_file);
        write(pd_file,'          ');
      end;
  writeln(pd_file);

  writeln(pd_file);
  writeln(pd_file,'-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-');
  writeln(pd_file)
end; {procedure print_pedigree}

{-------------------------------------------------------------------------------}
{===============================================================================}

procedure dump;

var
  i,j:byte;
  done:boolean;

begin
  done:=false;
  writeln;
  i:=1;
  j:=1;
  while (i<=MAX_FAMILY_MEMBERS) and (not done) do
    begin
      while (j<=MAX_BUFFER_WIDTH) and (not done) do
        begin
          if (j=1) and (buffer[i][j]<0) then
            done:=true
          else
            begin
              if buffer[i][j]>=0 then
                write(buffer[i][j]:3);
              inc(j)
            end;
        end;
      writeln;
      inc(i);
      j:=1
    end;
end; {procedure dump}

{-------------------------------------------------------------------------------}

procedure dump_family;

var
  i:byte;

begin
  for i:=1 to buffer_lines do
    with family[i] do
```

```
      begin
        writeln(buffer[i][1],'|',buffer[i][2],' --> level: ',level,'  spouse:
',spouse);
      end; {with}
  writeln;
end; {procedure dump_family}

{--------------------------------------------------------------------------}

procedure process_next_family;

begin
  null_buffer;
  fill_buffer;
  append_fu_file;
  convert_codes;
  construct_pedigree;
  make_inference_1;
  make_parent_inferences;
  append_gt_file;
  write_pedigree;
end; {process_next_family}

{--------------------------------------------------------------------------}

begin
end.
```

8.6 Heap Viewer Module

```
unit HeapView;

{
| File:          heapview.pas
| Compiles to:   heapview.tpu
| Usage:         'uses heapview'
|
| Author:        James Meyers
|                Brown University
|
| Start Date:   June 21, 1991
| Last Revision: June 21, 1991
|
| Intent:        Provide an object for monitoring free heap space
}

interface

uses Objects,     { Turbo basic object definitions }
     Views;       { Turbo windowing objects }

type
  PHeapView = ^THeapView;
  THeapView = object(TView)
```

```
    OldMem : LongInt;
    constructor Init(var Bounds: TRect);
    procedure Draw; virtual;
    procedure Update;
  end;

implementation

uses Drivers;

{ ==== (START) METHODS for THeapView ================================================ }

{ ------------------------------------------------------------------------------- }

constructor THeapView.Init(var Bounds: TRect);
begin
  TView.Init(Bounds);
  OldMem := 0;
end;

{ ------------------------------------------------------------------------------- }

procedure THeapView.Draw;
var
  S: String;
  B: TDrawBuffer;
  C: Byte;
begin
  OldMem := MemAvail;
  Str(OldMem:Size.X, S);
  C := GetColor(2);
  MoveChar(B, ' ', C, Size.X);
  MoveStr(B, S, C);
  WriteLine(0, 0, Size.X, 1, B);
end;

{ ------------------------------------------------------------------------------- }

procedure THeapView.Update;
begin
  if (OldMem <> MemAvail) then DrawView;
end; { procedure THeapView.Update }

{ ==== (END) METHODS for TMaster ================================================== }

begin
end.
```

8.7 File Viewer Module

```
unit MyWindow;

{
```

```
| File:           mywindow.pas
| Compiles to:    mywindow.tpu
| Usage:          'uses MyWindow'
|
| Author:         James Meyers
|                 Brown University
|
| Start Date:     June 16, 1991
| Last Revision:  June 21, 1991
|
| Intent:         To provide window objects and routines for file viewer
}

interface

uses Objects;

procedure OpenNewFileWindow(Bounds: TRect; WindowNo: integer; F: string);

{ ------------------------------------------------------------------------ }

implementation

uses
  Globals,
  App,
  Drivers,
  MsgBox,
  Views;

const
  MaxLines = 1200;

type

  Buffer = array[0..MaxLines - 1] of PString;

  PInterior = ^TInterior;
  TInterior = object(TScroller)
    constructor Init(var Bounds:TRect;AHScrollBar,AVScrollBar:PScrollBar;
                     F:string);
    procedure Draw; virtual;
    procedure ReadFile;
    procedure DoneFile;
    destructor Done; virtual;

  private
    Buf: Buffer;
    LC: integer;
  end;

  PFileWindow = ^TFileWindow;
  TFileWindow = object(TWindow)
    constructor Init(Bounds: TRect; WindowNo:integer; F: string);
```

```
    procedure MakeInterior(Bounds: TRect);
  end;

var
  FileName: string;
  LineCount: integer;


{ === (START) METHODS for TFileWindow ========================================== }

{ ------------------------------------------------------------------------------ }

constructor TFileWindow.Init(Bounds: TRect; WindowNo:integer; F: string);

var
  S:string[3];

begin

  Options := Options + ofTileable;
  FileName := F;
  Str(WindowNo, S);
  TWindow.Init(Bounds, F, wnNoNumber);
  MakeInterior(Bounds);

end; { constructor TFileWindow.Init }

{ ------------------------------------------------------------------------------ }

procedure TFileWindow.MakeInterior(Bounds: TRect);

var
  HScrollBar, VScrollBar: PScrollBar;
  Interior: PInterior;
  R: TRect;

begin

  Options := Options + ofTileable;
  VScrollBar := StandardScrollBar(sbVertical + sbHandleKeyboard);
  HScrollBar := StandardScrollBar(sbHorizontal + sbHandleKeyboard);
  GetExtent(Bounds);
  Bounds.Grow(-1,-1);
  Interior := New(PInterior, Init(Bounds, HScrollBar,VScrollBar,FileName));
  Insert(Interior)

end; { procedure TFileWindow.MakeInterior }

{ === (END) METHODS for TFileWindow ============================================ }


{ === (START) METHODS for TInterior ============================================ }
```

```
{ ------------------------------------------------------------------------ }

procedure TInterior.ReadFile;
var
  F: Text;
  S: String;
  w: word;

begin

  LineCount:= 0;
  Assign(F, FileName);
  {$I-}
  Reset(F);
  {$I+}

  if (IOResult <> 0) then
    begin

      error(#3'TInterior.ReadFile: IO Error')

    end;

  while not Eof(F) and (LineCount < MaxLines) do
    begin
      Readln(F, S);
      Buf[LineCount]:= NewStr(S);
      Inc(LineCount)
    end;

 close(F)

end; { procedure TFileWindow.ReadFile }

{ ------------------------------------------------------------------------ }

procedure TInterior.DoneFile;

var
  i:integer;

begin

  for i:=0 to LineCount -1 do
    if Buf[i] <> nil then
      DisposeStr(Buf[i])

end; { procedure DoneFile }


{ ------------------------------------------------------------------------ }

constructor TInterior.Init(var Bounds: TRect; AHScrollBar,
                           AVScrollBar: PScrollBar; F:string);
```

```
begin

  FileName := F;
  TScroller.Init(Bounds, AHScrollBar, AVScrollBar);
  GrowMode := gfGrowHiX + gfGrowHiY;
  Options := Options + ofFramed;
  ReadFile;
  SetLimit(128,LineCount);
  LC := LineCount

end; { procedure TInterior.Init }

{ ---------------------------------------------------------------------- }

procedure TInterior.Draw;

var
  Color: byte;
  Y,I: integer;
  B: TDrawBuffer;

begin

  Color := GetColor(1);

  for y:=0 to Size.Y - 1 do
    begin

      MoveChar(B, ' ', Color, Size.X);
      I := Delta.Y + Y;

      if (I < LC) and (Buf[I] <> NIL) then
        MoveStr(B, Copy(Buf[I]^, Delta.X + 1, Size.X), Color);

      WriteLine(0, Y, Size.X, 1, B)

    end

end; { procedure TInterior.Draw }

{ ---------------------------------------------------------------------- }

destructor TInterior.Done;

begin

  TScroller.Done;
  DoneFile

end; { destructor TInterior.Done }

{ ---------------------------------------------------------------------- }
```

```
{ ═══ (END) METHODS for TMaster ═══════════════════════════════════ }

procedure OpenNewFileWindow(Bounds: TRect; WindowNo: integer; F: string);

var
  Window   : PFileWindow;
  TestFile: text;
  Control : word;

begin

  {$I-}
  assign(TestFile, F);
  reset(TestFile);
  {$I+}

  if (IOResult <> 0) then
    Control:= MessageBox(#3'Cannot Open File '+ F, nil, mfError + mfOkButton)
  else
    begin
      Window := New(PFileWindow, Init(Bounds, WindowNo, F));
      DeskTop^.Insert(Window)
    end

end; { procedure OpenNewFileWindow }

begin
end.
```

8.8 Dialog Box Module

```
unit MyDialog;

{
| File:           mydialog.pas
| Compiles to:    mydialog.tpu
| Usage:          'uses MyDialog'
|
| Author:         James Meyers
|                 Brown University
|
| Start Date:     June 16, 1991
| Last Revision:  July 15, 1991
|
| Intent:         To provide dialog box definitions
}

interface

uses Drivers,
     MsgBox;

const
```

```
  ScreenModeArray: array[0..1] of word = (smCO80, smCO80 + smFont8x8);

type
  InputDialogData = record
    InputLineData: array[0..1] of string[128];
  end;

  OutputDialogData = record
    InputLineData: array[0..6] of string[128];
  end;

  ScreenSizeData = word;

  RRatesData = record
    InputLineData: array[0..1] of string[12];
  end;

var
  InputDD:  InputDialogData;
  OutputDD: OutputDialogData;
  ScreenDD: ScreenSizeData;
  RRatesDD: RRatesData;

procedure InitDialogData;
procedure SetInputFileDefaults;
procedure SetOutputFileDefaults;
procedure SetRecombinationRates;
function  SetScreenSize: word;

implementation

uses Globals,
     App,
     Objects,
     Views,
     Dialogs;

procedure InitDialogData;

var
  i: byte;

begin

  { Screen Mode }

  ScreenDD:= 0; { 25 line mode is default}

  { Input Files }

  InputDD.InputLineData[0] := ceph_hdr_str;
  InputDD.InputLineData[1] := infile_str;

  { Output Files }
```

```
  OutputDD.InputLineData[0] := vb_file_str;
  OutputDD.InputLineData[2] := vb_sum_str;
  OutputDD.InputLineData[1] := vb_nophase_str;
  OutputDD.InputLineData[3] := obs_exp_file_str;
  OutputDD.InputLineData[4] := fu_file_str;
  OutputDD.InputLineData[5] := gt_file_str;
  OutputDD.InputLineData[6] := pd_file_str;

  { Recombination Rates }

  str(rm_val:1:5, RRatesDD.InputLineData[0]);
  str(rf_val:1:5, RRatesDD.InputLineData[1])

end; { procedure InitDialogData }

{ ------------------------------------------------------------------------ }

procedure SetOutputFileDefaults;

var
  Dialog: PDialog;
  R: TRect;
  Control: word;
  B: PView;
  i,y: byte;
  s: array[0..6] of string;

begin
  R.Assign(10, 2, 70, 20);
  Dialog:= New(PDialog, Init(R, 'Output Files'));

  s[0]:= 'Phase ~K~nown Equations';
  s[1]:= 'Phase ~U~nknown Equations';
  s[2]:= 'Family ~S~um Equation';
  s[3]:= '~O~bserved/Expected Tables';
  s[4]:= '~F~actor-Union File';
  s[5]:= '~G~enotype File';
  s[6]:= '~P~edigree File';

  for i:=0 to 3 do
    begin
      y:= i*3+1;
      R.Assign(3, y+1, 27, y+2);
      B:= New(PInputLine, Init(R, 128));
      Dialog^.Insert(B);
      R.Assign(2, y, 27, y+1);
      Dialog^.Insert(New(PLabel, Init(R, s[i], B)))
    end;

  for i:=4 to 6 do
    begin
      y:= (i-4)*3+1;
      R.Assign(31, y+1, 57, y+2);
```

```
      B:= New(PInputLine, Init(R, 128));
      Dialog^.Insert(B);
      R.Assign(30, y, 57, y+1);
      Dialog^.Insert(New(PLabel, Init(R, s[i], B)))
    end;

  R.Assign(35, 15, 45, 17);
  Dialog^.Insert(New(PButton, Init(R, '~O~K', cmOK, bfDefault)));
  R.Assign(48, 15, 58, 17);
  Dialog^.Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));

  Dialog^.SetData(OutputDD);
  Control:= DeskTop^.ExecView(Dialog);
  if Control <> cmCancel then
    begin
      Dialog^.GetData(OutputDD);
        vb_file_str := OutputDD.InputLineData[0];
        vb_sum_str := OutputDD.InputLineData[2];
        vb_nophase_str := OutputDD.InputLineData[1];
        obs_exp_file_str := OutputDD.InputLineData[3];
        fu_file_str := OutputDD.InputLineData[4];
        gt_file_str := OutputDD.InputLineData[5];
        pd_file_str :=   OutputDD.InputLineData[6]
    end;

  Dispose(Dialog, Done)

end; { procedure SetOutputFileDefaults }

{ ------------------------------------------------------------------------- }

procedure SetInputFileDefaults;

var
  Dialog : PDialog;
  R      : TRect;
  Control: word;
  w      : word;
  B      : PView;
  hold   : string;

begin
  R.Assign(20, 6, 60, 19);
  Dialog:= New(PDialog, Init(R, 'Input Files'));

  R.Assign(3, 3, 27, 4);
  B:= New(PInputLine, Init(R, 128));
  Dialog^.Insert(B);
  R.Assign(2, 2, 27, 3);
  Dialog^.Insert(New(PLabel, Init(R, 'CEPH ~D~ata File', B)));

  R.Assign(3, 6, 27, 7);
  B:= New(PInputLine, Init(R, 128));
  Dialog^.Insert(B);
```

```
  R.Assign(2, 5, 27, 6);
  Dialog^.Insert(New(PLabel, Init(R, 'CEPH ~P~edigree File', B)));

  R.Assign(15, 10, 25, 12);
  Dialog^.Insert(New(PButton, Init(R, '~O~K', cmOK, bfDefault)));
  R.Assign(28, 10, 38, 12);
  Dialog^.Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));

  Dialog^.SetData(InputDD);
  Control:= DeskTop^.ExecView(Dialog);
  if Control <> cmCancel then
    begin
      {$I-}
      Dialog^.GetData(InputDD);
        hold:= ceph_hdr_str;
        ceph_hdr_str := InputDD.InputLineData[0];
        assign(ceph_hdr,ceph_hdr_str);
        reset(ceph_hdr);
        if IOResult <> 0 then
          begin
            w:= MessageBox(#3'Cannot Open File '+ ceph_hdr_str +
                           #13 + #13 + #3 + 'Default Restored'
                       , nil, mfError + mfOkButton);
            InputDD.InputLineData[0]:= hold;
            ceph_hdr_str:= hold;
            Dialog^.SetData(InputDD)
          end;

        hold:= infile_str;
        infile_str := InputDD.InputLineData[1];
        assign(infile, infile_str);
        reset(infile);
        if IOResult <> 0 then
          begin
            w:= MessageBox(#3'Cannot Open File '+ infile_str +
                           #13 + #13 + #3 + 'Default Restored'
                       , nil, mfError + mfOkButton);
            InputDD.InputLineData[1]:= hold;
            infile_str:= hold;
            Dialog^.SetData(InputDD)
          end;
      {$I+}
    end;

  Dispose(Dialog, Done)

end; { procedure SetInputFileDefaults }

{ ------------------------------------------------------------------------ }

function SetScreenSize: word;

var
  Dialog: PDialog;
```

```
  R: TRect;
  Control: word;
  B: PView;
  Mode: word;

begin
  R.Assign(20, 6, 60, 19);
  Dialog:= New(PDialog, Init(R, 'Screen Size'));

  R.Assign(2, 2, 37, 4);
  B:= New(PRadioButtons, Init(R,
    NewSItem('~2~5 Lines',
    NewSItem('~E~GA/VGA Lines (43/50)',
    nil))
  ));
  Dialog^.Insert(B);

  R.Assign(15, 10, 25, 12);
  Dialog^.Insert(New(PButton, Init(R, '~O~K', cmOK, bfDefault)));
  R.Assign(28, 10, 38, 12);
  Dialog^.Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));

  Dialog^.SetData(ScreenDD);
  Control:= DeskTop^.ExecView(Dialog);
  if Control <> cmCancel then
    Dialog^.GetData(ScreenDD);

  Dispose(Dialog, Done);

  SetScreenSize:= ScreenDD

end; { procedure SetScreenSize }

{ ------------------------------------------------------------------------- }

procedure SetRecombinationRates;

var
  Dialog: PDialog;
  R: TRect;
  Control: word;
  B: PView;
  Mode: word;
  code: integer;

begin
  R.Assign(20, 6, 60, 19);
  Dialog:= New(PDialog, Init(R, 'Recombination Rates'));

  R.Assign(3, 3, 17, 4);
  B:= New(PInputLine, Init(R, 12));
  Dialog^.Insert(B);
  R.Assign(2, 2, 30, 3);
  Dialog^.Insert(New(PLabel, Init(R, '~M~ale Recombination Rate', B)));
```

```
  R.Assign(3, 6, 17, 7);
  B:= New(PInputLine, Init(R, 12));
  Dialog^.Insert(B);
  R.Assign(2, 5, 30, 6);
  Dialog^.Insert(New(PLabel, Init(R, '~F~emale Recombination Rate', B)));

  R.Assign(15, 10, 25, 12);
  Dialog^.Insert(New(PButton, Init(R, '~O~K', cmOK, bfDefault)));
  R.Assign(28, 10, 38, 12);
  Dialog^.Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));

  Dialog^.SetData(RRatesDD);
  Control:= DeskTop^.ExecView(Dialog);
  if Control <> cmCancel then
    begin
      Dialog^.GetData(RRatesDD);
      val(RRatesDD.InputLineData[0], rm_val, code);
      val(RRatesDD.InputLineData[1], rf_val, code)
    end;

  Dispose(Dialog, Done);

end; { procedure SetRecombinationRates }

{ ------------------------------------------------------------------------ }

begin
end.
```

8.9 Polynomial Handler Module

```
unit Polymath;

{
| File:          Polymath.pas
| Compiles to:   Polymath.tpu
| Usage:         "uses Polymath;"
|
| Author:        James Meyers
|                Brown University
|
| Start Date:    July 4, 1991
| Last Revision: July 13, 1991
|
| Intent:        Provides data types and function calls for construction
|                of, and algebraic manipulation of, polynomial expressions.
}

{ --------------------------- INTERFACE --------------------------- }

interface
```

```
uses Globals;

type
  num_den = record
    constant    :integer;     { constant in the expression }
    rm          :integer;     { male recombination rate    }
    rf          :integer;     { female recombination rate  }
    onerm       :integer;     { (1-rm)                     }
    onerf       :integer;     { (1-rf)                     }
    rm_rf       :integer;     { (rm)(rf)                   }
    onerm_onerf:integer;      { (1-rm)(1-rf)               }
    rm_onerf    :integer;     { (rm)(1-rf)                 }
    rf_onerm    :integer;     { (rf)(1-rm)                 }
  end;

  term_ptr = ^term;
  term = record               { a term in an expression }
    numerator  :num_den;
    denominator:num_den;
    next       :term_ptr;
  end;

{ public routines }

procedure init_num_den(var x:num_den);
procedure init_term(var t:term);
function term_val(t:term):real;
function complex_term_val(tp:term_ptr):real;
function zero_term(x:term):boolean;
function one_term(x:term):boolean;
function div2_term(x:term):boolean;
procedure do_div2_term(var x:term);
function simple_num_den(x:num_den):boolean;
function simple_term(t:term):boolean;
procedure write_term(var out:text; x:term);
procedure write_complex_term(var out:text; tp:term_ptr);
procedure add_terms(var t1:term; t2:term);
procedure multiply_terms(t1,t2:term; var tt:term);
function equal_terms(t1,t2:term):boolean;
procedure assign_term(var st:term; t:term);
procedure compress_complex_term(var t:term_ptr);

{ ------------------------- IMPLEMENTATION ---------------------- }

implementation

procedure init_num_den(var x:num_den);

begin
  x.constant:=0;
  x.rm:=0;
  x.rf:=0;
  x.onerm:=0;
  x.onerf:=0;
```

```
  x.rm_rf:=0;
  x.onerm_onerf:=0;
  x.rm_onerf:=0;
  x.rf_onerm:=0;
end; {procedure init_num_den}

{ ---------------------------------------------------------------------- }

procedure init_term(var t:term);

begin
  init_num_den(t.numerator);
  init_num_den(t.denominator);
  t.next := NIL;
end; {procedure init_term}

{ ---------------------------------------------------------------------- }

function term_val(t:term):real;

var
  hold:real;

begin

  hold:=0.0;

  with t.numerator do
    begin
      hold:=hold + constant;
      hold:=hold + rm * rm_val;
      hold:=hold + rf * rf_val;
      hold:=hold + onerm * (1 - rm_val);
      hold:=hold + onerf * (1 - rf_val);
      hold:=hold + rm * rf * rm_val * rf_val;
      hold:=hold + onerm_onerf * (1 - rm_val) * (1 - rf_val);
      hold:=hold + rm_onerf * rm_val * (1 - rf_val);
      hold:=hold + rf_onerm * rf_val * (1 - rm_val);
    end; {with}

  if (t.denominator.constant = 0) then
    init_num_den(t.numerator)

  else
    hold:=hold / t.denominator.constant;

  term_val:=hold

end; {function term_val}

{ ---------------------------------------------------------------------- }

function complex_term_val(tp:term_ptr):real;
```

```
var
  p:term_ptr;
  hold:real;

begin

  hold:=0.0;

  if (tp<>NIL) then
    begin
      p:=tp;
      while (p<>NIL) do
        begin
          hold:=hold + term_val(p^);
          p:=p^.next
        end; {while}
    end;

  complex_term_val:=hold

end; {function complex_term_val}


{ ------------------------------------------------------------------------ }

function zero_term(x:term):boolean;

begin

  zero_term:= TRUE;

  if (x.numerator.constant<>0) then
    zero_term := false
  else if (x.numerator.rm<>0) then
    zero_term := false
  else if (x.numerator.rf <> 0) then
    zero_term := false
  else if (x.numerator.onerm<> 0) then
    zero_term := false
  else if (x.numerator.onerf<> 0) then
    zero_term := false
  else if (x.numerator.rm_rf<> 0) then
    zero_term := false
  else if (x.numerator.onerm_onerf<> 0) then
    zero_term := false
  else if (x.numerator.rm_onerf<> 0) then
    zero_term := false
  else if (x.numerator.rf_onerm<> 0) then
    zero_term := false

end; { function zero_term }

{------------------------------------------------------------------------}
```

```
function div2_term(x:term):boolean;

var
  hold:boolean;

begin
  hold:=true;
  if (x.numerator.constant mod 2<>0) then
    hold := false
  else if (x.numerator.rm mod 2<>0) then
    hold := false
  else if (x.numerator.rf mod 2<> 0) then
    hold := false
  else if (x.numerator.onerm mod 2<> 0) then
    hold := false
  else if (x.numerator.onerf mod 2<> 0) then
    hold := false
  else if (x.numerator.rm_rf mod 2<> 0) then
    hold := false
  else if (x.numerator.onerm_onerf mod 2<> 0) then
    hold := false
  else if (x.numerator.rm_onerf mod 2<> 0) then
    hold := false
  else if (x.numerator.rf_onerm mod 2<> 0) then
    hold := false
  else if (x.denominator.constant mod 2<> 0) then
    hold := false;

  div2_term := hold

end;

{-------------------------------------------------------------------------}

procedure do_div2_term(var x:term);

begin

  x.numerator.constant :=x.numerator.constant div 2;
  x.numerator.rm := x.numerator.rm div 2;
  x.numerator.rf := x.numerator.rf div 2;
  x.numerator.onerm := x.numerator.onerm div 2;
  x.numerator.onerf := x.numerator.onerf div 2;
  x.numerator.rm_rf := x.numerator.rm_rf div 2;
  x.numerator.onerm_onerf := x.numerator.onerm_onerf div 2;
  x.numerator.rm_onerf := x.numerator.rm_onerf div 2;
  x.numerator.rf_onerm := x.numerator.rf_onerm div 2;
  x.denominator.constant := x.denominator.constant div 2;

end;

{ ------------------------------------------------------------------------ }

function one_term(x:term):boolean;
```

```
begin

  one_term:= TRUE;

  if (x.numerator.constant <> x.denominator.constant) then
    one_term := false
  else if (x.numerator.constant = 0) then
    one_term := false
  else if (x.numerator.rm<>0) then
    one_term := false
  else if (x.numerator.rf <> 0) then
    one_term := false
  else if (x.numerator.onerm<> 0) then
    one_term := false
  else if (x.numerator.onerf<> 0) then
    one_term := false
  else if (x.numerator.rm_rf<> 0) then
    one_term := false
  else if (x.numerator.onerm_onerf<> 0) then
    one_term := false
  else if (x.numerator.rm_onerf<> 0) then
    one_term := false
  else if (x.numerator.rf_onerm<> 0) then
    one_term := false

end; {function one_term}

{-------------------------------------------------------------------------}

function simple_num_den(x:num_den):boolean;

var
  c:byte;

begin
  c:=0;

  if (x.constant <> 0) then
    inc(c);
  if (x.rm <> 0) then
    inc(c);
  if (x.rf <> 0) then
    inc(c);
  if (x.onerm <> 0) then
    inc(c);
  if (x.onerf <> 0) then
    inc(c);
  if (x.rm_rf <> 0) then
    inc(c);
  if (x.onerm_onerf <> 0) then
    inc(c);
  if (x.rm_onerf <> 0) then
    inc(c);
```

```
  if (x.rf_onerm <> 0) then
    inc(c);

  simple_num_den := (c<2);

end; {function simple_num_den}

{-------------------------------------------------------------------------}

function simple_term(t:term):boolean;

var
  hold:boolean;

begin
  hold := true;

  if (not simple_num_den(t.numerator)) then
    hold := false;
  if (not simple_num_den(t.denominator)) then
    hold := false;

  simple_term := hold
end; {function simple_term}

{-------------------------------------------------------------------------}

procedure write_term(var out:text; x:term);

var
  print:boolean;

begin
  print := false;

  if (zero_term(x)) then
    write(out,'0')

  else
    begin
      if not(simple_term(x)) then
        write(out,'(');

      if (x.numerator.constant<>0) then
        begin
          write(out,x.numerator.constant);
          print := true;
        end;

      if (x.numerator.rm<>0) then
        begin
          if (print) then
            write(out,' + ');
          if (x.numerator.rm > 1) then
```

```
      write(out,x.numerator.rm,'*');
    write(out,'rm');
    print := true;
  end;

if (x.numerator.rf <> 0) then
  begin
    if (print) then
      write(out,' + ');
    if (x.numerator.rf > 1) then
      write(out,x.numerator.rf,'*');
    write(out,'rf');
    print := true;
  end;

if (x.numerator.onerm<> 0) then
  begin
    if (print) then
      write(out,' + ');
    if (x.numerator.onerm > 1) then
      write(out,x.numerator.onerm,'*');
    write(out,'(1-rm)');
    print := true;
  end;

if (x.numerator.onerf<> 0) then
  begin
    if (print) then
      write(out,' + ');
    if (x.numerator.onerf > 1) then
      write(out,x.numerator.onerf,'*');
    write(out,'(1-rf)');
    print := true;
  end;

if (x.numerator.rm_rf<> 0) then
  begin
    if (print) then
      write(out,' + ');
    if (x.numerator.rm > 1) then
      write(out,x.numerator.rm,'*');
    write(out,'(rm*rf)');
    print := true;
  end;

if (x.numerator.onerm_onerf<> 0) then
  begin
    if (print) then
      write(out,' + ');
    if (x.numerator.rm > 1) then
      write(out,x.numerator.rm,'*');
    write(out,'((1-rm)*(1-rf))');
    print := true;
  end;
```

```
      if (x.numerator.rm_onerf<> 0) then
        begin
          if (print) then
            write(out,' + ');
          if (x.numerator.rm > 1) then
            write(out,x.numerator.rm,'*');
          write(out,'(rm*(1-rf))');
          print := true;
        end;

      if (x.numerator.rf_onerm<> 0) then
        begin
          if (print) then
            write(out,' + ');
          if (x.numerator.rm > 1) then
            write(out,x.numerator.rm,'*');
          write(out,'(rf*(1-rm))');
          print := true;
        end;

      if not(simple_term(x)) then
        write(out,')');
      write(out,'/',x.denominator.constant);

    end
end;

{-------------------------------------------------------------------------}

procedure write_complex_term(var out:text; tp:term_ptr);

var
  p:term_ptr;

begin
  if (tp=NIL) then
    write(out,'0')
  else
    begin
      write(out,'(');
      p:=tp;
      while (p<>NIL) do
        begin
          if (p<>tp) then
            write(out,'+');
          write_term(out,p^);
          p:=p^.next
        end; {while}
      write(out,')')
    end {else}
end; {procedure write_complex_term}

{-------------------------------------------------------------------------}
```

```
function equal_num_den(x1,x2:num_den):boolean;

var
  hold:boolean;

begin
  hold:=true;

  if (x1.constant <> x2.constant) then
    hold := false
  else if (x1.rm <> x2.rm) then
    hold := false
  else if (x1.rf <> x2.rf) then
    hold := false
  else if (x1.onerm <> x2.onerm) then
    hold := false
  else if (x1.onerf <> x2.onerf) then
    hold := false
  else if (x1.rm_rf <> x2.rm_rf) then
    hold := false
  else if (x1.onerm_onerf <> x2.onerm_onerf) then
    hold := false
  else if (x1.rm_onerf <> x2.rm_onerf) then
    hold := false
  else if (x1.rf_onerm <> x2.rf_onerm) then
    hold := false;

  equal_num_den := hold

end; {function equal_num_den}

{-------------------------------------------------------------------------}

function equal_terms(t1,t2:term):boolean;

var
  hold:boolean;

begin
  hold:=true;
  if not equal_num_den(t1.numerator,t2.numerator) then
    hold:=false;
  if not equal_num_den(t1.denominator,t2.denominator) then
    hold:=false;

  equal_terms:=hold

end; {function equal_terms}

{-------------------------------------------------------------------------}

procedure multiply_terms(t1,t2:term; var tt:term);
```

```
begin

  { trap errors }

  if ( not equal_num_den(t1.denominator,t2.denominator)) then
    error ('Unequal denominators');
  if ( not simple_term(t1)) or ( not simple_term(t2)) then
    error ('Tried to multiply complex terms');

  { multiply denominators }

  tt.denominator.constant:=t1.denominator.constant * t2.denominator.constant;

  { C * C }

  if (t1.numerator.constant <> 0) and (t2.numerator.constant <> 0) then
    tt.numerator.constant := t1.numerator.constant * t2.numerator.constant

  { C * rm }

  else if (t1.numerator.constant <> 0) and (t2.numerator.rm <> 0) then
    tt.numerator.rm := t1.numerator.constant * t2.numerator.rm

  { C * rf }

  else if (t1.numerator.constant <> 0) and (t2.numerator.rf <> 0) then
    tt.numerator.rf := t1.numerator.constant * t2.numerator.rf

  { C * onerm }

  else if (t1.numerator.constant <> 0) and (t2.numerator.onerm <> 0) then
    tt.numerator.onerm := t1.numerator.constant * t2.numerator.onerm

  { C * onerf }

  else if (t1.numerator.constant <> 0) and (t2.numerator.onerf <> 0) then
    tt.numerator.onerf := t1.numerator.constant * t2.numerator.onerf

  { C * rm_rf }

  else if (t1.numerator.constant <> 0) and (t2.numerator.rm_rf <> 0) then
    tt.numerator.rm_rf := t1.numerator.constant * t2.numerator.rm_rf

  { C * onerm_onerf }

  else if (t1.numerator.constant <> 0) and (t2.numerator.onerm_onerf <> 0) then
    tt.numerator.onerm_onerf := t1.numerator.constant * t2.numerator.onerm_onerf

  { C * rm_onerf }

  else if (t1.numerator.constant <> 0) and (t2.numerator.rm_onerf <> 0) then
    tt.numerator.rm_onerf := t1.numerator.constant * t2.numerator.rm_onerf

  { C * rf_onerm }
```

```
else if (t1.numerator.constant <> 0) and (t2.numerator.rf_onerm <> 0) then
  tt.numerator.rf_onerm := t1.numerator.constant * t2.numerator.rf_onerm

{ rm * rf }

else if (t1.numerator.rm <> 0) and (t2.numerator.rf <> 0) then
  tt.numerator.rm_rf := t1.numerator.rm * t2.numerator.rf

{ rm * C }

else if (t1.numerator.rm <> 0) and (t2.numerator.constant <> 0) then
  tt.numerator.rm := t1.numerator.rm * t2.numerator.constant

{ rm * onerf }

else if (t1.numerator.rm <> 0) and (t2.numerator.onerf <> 0) then
  tt.numerator.rm_onerf := t1.numerator.rm * t2.numerator.onerf

{ rf * C }

else if (t1.numerator.rf <> 0) and (t2.numerator.constant <> 0) then
  tt.numerator.rf := t1.numerator.rf * t2.numerator.constant

{ rf * rm }

else if (t1.numerator.rf <> 0) and (t2.numerator.rm <> 0) then
  tt.numerator.rm_rf := t1.numerator.rf * t2.numerator.rm

{ rf * onerm }

else if (t1.numerator.rf <> 0) and (t2.numerator.onerm <> 0) then
  tt.numerator.rf_onerm := t1.numerator.rf * t2.numerator.onerm

{ onerm * C }

else if (t1.numerator.onerm <> 0) and (t2.numerator.constant <> 0) then
  tt.numerator.onerm := t1.numerator.onerm * t2.numerator.constant

{ onerm * rf }

else if (t1.numerator.onerm <> 0) and (t2.numerator.rf <> 0) then
  tt.numerator.rf_onerm := t1.numerator.onerm * t2.numerator.rf

{ onerm * onerf }

else if (t1.numerator.onerm <> 0) and (t2.numerator.onerf <> 0) then
  tt.numerator.onerm_onerf := t1.numerator.onerm * t2.numerator.onerf

{ onerf * onerm }

else if (t1.numerator.onerf <> 0) and (t2.numerator.onerm <> 0) then
  tt.numerator.onerm_onerf := t1.numerator.onerf * t2.numerator.onerm
```

```
  {onerf * C }

  else if (t1.numerator.onerf <> 0) and (t2.numerator.constant <> 0) then
    tt.numerator.onerf := t1.numerator.onerf * t2.numerator.constant

  {rm_rf * C }
{
  else if (t1.numerator.rm_rf <> 0) and (t2.numerator.constant <> 0) then
    tt.numerator.onerf := t1.numerator.rm_rf * t2.numerator.constant
}
  { error }

  else
    error ('Invalid multiplication')

end; {procedure multiply_terms}

{-------------------------------------------------------------------------}

procedure add_terms(var t1:term; t2:term);

begin
  t1.numerator.constant := t1.numerator.constant + t2.numerator.constant;
  t1.numerator.rm := t1.numerator.rm + t2.numerator.rm;
  t1.numerator.rf := t1.numerator.rf + t2.numerator.rf;
  t1.numerator.onerm := t1.numerator.onerm + t2.numerator.onerm;
  t1.numerator.onerf := t1.numerator.onerf + t2.numerator.onerf;
  t1.numerator.rm_rf := t1.numerator.rm_rf + t2.numerator.rm_rf;
  t1.numerator.onerm_onerf := t1.numerator.onerm_onerf +
t2.numerator.onerm_onerf;
  t1.numerator.rm_onerf := t1.numerator.rm_onerf + t2.numerator.rm_onerf;
  t1.numerator.rf_onerm := t1.numerator.rf_onerm + t2.numerator.rf_onerm;

  if (t1.denominator.constant = 0) and (not zero_term(t2)) then
    t1.denominator.constant := t2.denominator.constant;

end; {procedure add_terms}

{-------------------------------------------------------------------------}

procedure assign_term(var st:term; t:term);

begin
  with st.numerator do
    begin
      constant:=t.numerator.constant;
      rm:=t.numerator.rm;
      rf:=t.numerator.rf;
      onerm:=t.numerator.onerm;
      onerf:=t.numerator.onerf;
      rm_rf:=t.numerator.rm_rf;
      onerm_onerf:=t.numerator.onerm_onerf;
      rm_onerf:=t.numerator.rm_onerf;
      rf_onerm:=t.numerator.rf_onerm;
```

```
    end;
  init_num_den(st.denominator);
  st.denominator.constant:=t.denominator.constant;
  st.next := t.next
end;

{-------------------------------------------------------------------------}

procedure compress_complex_term(var t:term_ptr);

var
  p:term_ptr;
  nt:term_ptr;

begin
  if (p<>NIL) then
    begin
      p:=t;
      while (p<>NIL) do
       begin
          if (div2_term(p^)) then
            do_div2_term(p^);
          p:=p^.next
        end;
    end

end; {procedure compress_complex_term}

{-------------------------------------------------------------------------}

begin
end.
```

8.10 Genotype Analyzer Module

```
unit Geno_2_n;

{
| File:          geno_2_n.pas
| Compiles to:   geno_2_n.tpu
| Usage:         "uses Geno_2_n;"
|
| Author:        James Meyers
|                Brown University
|
| Start Date:    July 7, 1991
| Last Revision: July 13, 1991
|
| Intent:
}

interface
```

```
uses Globals,
     IO,
     Polymath;

type
  side_type = (LEFT, RIGHT);

  gamete = array[1..MAX_GENES] of byte;

  phase_type = (Unknown, Coupling, Repulsion, Dont_care);

  individual = record
    gametes: array[1..2] of gamete;
    sex    : byte;
    phase  : phase_type;
    index  : byte;
  end;

  gamete_probs_type = array[1..NUM_GAMETES] of term;

  np_gamete_probs_ptr = ^np_gamete_probs_type;

  np_gamete_probs_type = array[1..2] of gamete_probs_type;

  Etable_ptr  = ^Etable_type;

  Etable_type = array[1..MAX_T_GAMETES] of array[1..MAX_T_GAMETES] of term_ptr;

  Otable_type = array[1..MAX_T_GAMETES] of array[1..MAX_T_GAMETES] of integer;

  real_Etable_ptr = ^real_Etable_type;

  real_Etable_type = array[1..MAX_T_GAMETES] of array[1..MAX_T_GAMETES] of real;

var
  Otable, L0a_Otable, L0b_Otable, global_Otable, np_Otable: Otable_type;

  cur_Etable, sum_Etable, sum_0a_Etable, sum_0b_Etable: Etable_ptr;

  sum_np1_Etable, sum_np2_Etable, sum_np3_Etable, sum_np4_Etable: Etable_ptr;

  F1, F2, F3, F4: Etable_ptr;

  father_gamete_probs, mother_gamete_probs: gamete_probs_type;

  fnpg_probs, mnpg_probs: np_gamete_probs_ptr;

  real_Etable, np_real_Etable: real_Etable_ptr;

  np_count: integer;

procedure init_geno_analyzer;
procedure genotype_analysis;
procedure output_totals;
```

```
{*************************************************************************}

implementation

{ ------------------------------------------------------------------------ }

procedure init_Etable(var E:Etable_ptr);

var
  i, j: byte;

begin


  new(E);

  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      E^[i][j]:= NIL

end;

{ ------------------------------------------------------------------------ }

procedure dealloc_Etable(var E:Etable_ptr);

{
| This procedure deallocates the heap space of the indicated Etable. Note
| that the table is traversed and all entries disposed. Those table entries
| which are linked lists will leave dangling pointers.
}

var
  i,j: byte;

begin

  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      if E^[i][j] <> NIL then
        dispose(E^[i][j]);    { WARNING: possible dangling pointers }

  dispose(E)

end; { procedure dealloc_Etable }

{ ------------------------------------------------------------------------ }

procedure init_real_Etable(var E: real_Etable_ptr);

var
  i, j: byte;
```

```
begin

  new(E);

  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      E^[i][j]:= 0.0

end; { procedure init_real_Etable }

{ ------------------------------------------------------------------------ }

procedure init_Otable(var O:Otable_type);

var
  i, j: byte;

begin

  for i:= 1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      O[i][j]:= 0

end; { procedure init_Otable }

{ ------------------------------------------------------------------------ }

procedure init_gamete_probs(var g: gamete_probs_type);

var
  i: byte;

begin

  for i:=1 to NUM_GAMETES do
    begin

      init_num_den(g[i].numerator);
      init_num_den(g[i].denominator);
      g[i].next:= NIL

    end

end; { procedure init_gamete_probs }

{ ------------------------------------------------------------------------ }

procedure dump_Etable(var E: Etable_ptr);

var
  i, j: byte;

begin
```

```
  for i:=1 to MAX_T_GAMETES do
    begin

      for j:=1 to MAX_T_GAMETES do
        begin

          write(i,',',j,': ');
          if (E^[i][j] <> NIL) then
            begin
              write_complex_term(debug,E^[i][j]);
              writeln
            end
          else
            writeln(' --- NIL --- ')

        end { for j }

    end { for i }

end; { procedure dump_Etable }

{ ------------------------------------------------------------------------ }

function hetero(ind: individual; locus: byte): boolean;

begin

  if (ind.gametes[1][locus] <> ind.gametes[2][locus]) then
    hetero:= TRUE
  else
    hetero:= FALSE

end; { function hetero }

{ ------------------------------------------------------------------------ }

function homo(ind: individual; locus: byte): boolean;

begin

  if (ind.gametes[1][locus] = ind.gametes[2][locus]) then
    homo:= TRUE
  else
    homo:= FALSE

end; { function homo }

{ ------------------------------------------------------------------------ }

function same_gamete(g1,g2: gamete): boolean;

var
  hold: boolean;
```

```
begin

  hold:= TRUE;

  if (g1[1] <> g2[1]) then
    hold:= FALSE
  else if (g1[2] <> g2[2]) then
    hold:= FALSE;

  same_gamete:= hold

end; { function same_gamete }

{ ------------------------------------------------------------------------ }

function gamete_table_index(g: gamete): integer;

{
| The function calculates the rank of the given gamete. i.e. the appropriate
| index for an Etable or Otable.
}

var
  i, j, hold: integer;

begin

  i:= g[1];
  j:= g[2];

  if (i>j) then
    iswap(i,j);

  hold:= MAX_T_GAMETES;
  hold:= hold - (((MAX_ALLELES-i)*(MAX_ALLELES-i)+MAX_ALLELES-i ) div 2);
  hold:= hold - MAX_ALLELES + j;

  gamete_table_index:= hold

end;

{ ------------------------------------------------------------------------ }

function gamete_prob_index(i, j: integer): integer;

begin

  gamete_prob_index:= ((i-1)*MAX_ALLELES) + j;

end; { function gamete_prob_index }

{ ------------------------------------------------------------------------ }

function index_to_gamete_str(i: integer): string;
```

```
var
  j, k: integer;
  g   : gamete;
  s,
  hold: string;

begin

  hold:= '';

  for j:=1 to MAX_ALLELES do
    for k:=1 to MAX_ALLELES do
      begin

        g[1]:= j;
        g[2]:= k;
        if ((gamete_table_index(g)) = i) then
          begin

            if (j>k) then iswap(j,k);
            str(j,s);
            hold:= hold + s;
            str(k,s);
            hold:= hold + s;
            index_to_gamete_str:= hold;
            exit

          end

      end

end; { function index_to_gamete }

{ ------------------------------------------------------------------------ }

procedure dump_gamete_probs(var gp: gamete_probs_type);

var
  i: integer;

begin

  for i:=1 to MAX_T_GAMETES do
    if not(zero_term(gp[i])) then
      begin
        write(debug, i,'-> ');
        write(debug, index_to_gamete_str(i),': ');
        write_term(debug, gp[i]);
        writeln(debug)
      end

end; { procedure dump_gamete_probs }
```

```
{ ------------------------------------------------------------------------ }

function phase_alert(ind: individual): boolean;

begin

  if (hetero(ind,1)) and (hetero(ind,2)) then
    phase_alert:= TRUE

  else
    phase_alert:= FALSE

end; { function phase_alert }

{ ------------------------------------------------------------------------ }

function ind_g_string(ind: individual): string;

var
  hold, s: string;

begin

  hold:= '';

  str(ind.gametes[1][1],s);
  hold:= hold + s;

  str(ind.gametes[2][1],s);
  hold:= hold + s;

  str(ind.gametes[1][2],s);
  hold:= hold + s;

  str(ind.gametes[2][2],s);
  hold:= hold + s;

  ind_g_string:= hold

end; { function g_string }

{ ------------------------------------------------------------------------ }

function buf_g_string(index: byte): string;

var
  hold, s: string;

begin

  hold:= '';

  if (index <> 0) then
    begin
```

```
      str(buffer[index][FIRST_GENE],s);
      hold:= hold + s;
      str(buffer[index][FIRST_GENE+2],s);
      hold:= hold + s;
      str(buffer[index][FIRST_GENE+1],s);
      hold:= hold + s;
      str(buffer[index][FIRST_GENE+3],s);
      hold:= hold + s;

    end;

  buf_g_string:= hold

end; { function buf_g_string }

{ ------------------------------------------------------------------------ }

procedure assign_gametes(var ind: individual; index: byte);

var
  i: byte;

begin

  if (index = 0) then
    begin

      ind.gametes[1][1]:= 0;
      ind.gametes[1][2]:= 0;
      ind.gametes[2][1]:= 0;
      ind.gametes[2][2]:= 0

    end { if }

  else
    begin

      ind.sex:= buffer[index][SEX];

      i:= FIRST_GENE;
      ind.gametes[1][1]:= buffer[index][i];
      inc(i);
      ind.gametes[2][1]:= buffer[index][i];
      inc(i);
      ind.gametes[1][2]:= buffer[index][i];
      inc(i);
      ind.gametes[2][2]:= buffer[index][i];

      ind.index:= index

    end { else }

end;
```

```
{ ------------------------------------------------------------------------ }

procedure calc_gamete_probs(var ind:individual; var gp:gamete_probs_type);

{
|
| Assigns probabilities to gametes. All unassigned probabilities are 0.
|
|                                                          A B
| Patterns generated by the subprocedures are based on a = = model.
|                                                          C D
}

var
  A,
  B,
  C,
  D  : byte;

  procedure pattern1;

  {
  | p(AB):= 1
  }

  begin

    gp[gamete_prob_index(A,B)].numerator.constant:= 2

  end; { procedure pattern1 }

  {------------------------------------------------------------------------ }

  procedure pattern2;

  {
  | p(AB):= 1/2;
  | p(CD):= 1/2;
  }

  begin

    gp[gamete_prob_index(A,B)].numerator.constant:= 1;
    gp[gamete_prob_index(C,D)].numerator.constant:= 1

  end; {procedure pattern2 }

  {------------------------------------------------------------------------ }

  procedure pattern3;

  {
  | p(AB):= (1-r)/2;
```

```
  | p(CD):= (1-r)/2;
  | p(AD):= r/2;
  | p(CB):= r/2;
  }

  begin

    if (ind.sex = MALE) then
      begin

        gp[gamete_prob_index(A,B)].numerator.onerm:= 1;
        gp[gamete_prob_index(C,D)].numerator.onerm:= 1;
        gp[gamete_prob_index(A,D)].numerator.rm   := 1;
        gp[gamete_prob_index(C,B)].numerator.rm   := 1

      end { MALE }

    else { female }
      begin

        gp[gamete_prob_index(A,B)].numerator.onerf:= 1;
        gp[gamete_prob_index(C,D)].numerator.onerf:= 1;
        gp[gamete_prob_index(A,D)].numerator.rf   := 1;
        gp[gamete_prob_index(C,B)].numerator.rf   := 1

      end { FEMALE }

  end; { procedure pattern3 }

  {------------------------------------------------------------------------ }

var
  i    : integer;

begin

  { denominator is 2 for all probabilities }

  for i:= 1 to NUM_GAMETES do
    gp[i].denominator.constant:= 2;

  { read alleles from individual }

  A:= ind.gametes[1][1];
  B:= ind.gametes[1][2];
  C:= ind.gametes[2][1];
  D:= ind.gametes[2][2];

  { compute probabilities }

  if ( (homo(ind,1)) and (homo(ind,2)) ) then
    begin

      if (A=B) and (C=D) then
```

```
        pattern1

      else if (A=C) and (B=D) then
        pattern1

      else
        error('calc_gamete_probs: double homozygote exception');

    end

  else if ( (hetero(ind,1)) and (hetero(ind,2)) ) then
    begin

      if (A=B) and (C=D) then
        pattern3

      else if (A=B) xor (C=D) then
        pattern3

      else if (A<>B) and (C<>D) then
        pattern3

      else
        error('calc_gamete_probs: double heterozygote exception');

    end

  else { one homo, one hetero }
    begin

      if (A=B) xor (C=D) then
        pattern2

      else if (A<>B) and (C<>D) then
        pattern2

      else
        error('calc_gamete_probs: homozygote/heterozygote exception');

    end

end; {procedure calc_gamete_probs}


{ ------------------------------------------------------------------------ }

function valid_individual(ind: individual): boolean;

var
  hold: boolean;
  i, j: byte;

begin
```

```
  hold:= TRUE;

  for i:=1 to MAX_GENES do
    for j:=1 to 2 do
      if (ind.gametes[i][j] = 0) then
        hold:= FALSE;

  valid_individual:= hold

end; { function valid_individual }

{ ------------------------------------------------------------------------ }


function get_phase(ind: individual):phase_type;

var
  hold      : phase_type;
  mom,
  dad       : individual;
  i, j,
  m, n,
  x, y,
  xx, yy,
  count,
  ii, jj    : integer;
  g1, g2    : gamete;

begin

  hold:= Dont_care;

  if (phase_alert(ind)) then
    begin

      hold:= Unknown;

      count:= 0;

      bswap(ind.gametes[1][2], ind.gametes[2][1]);

      xx:= gamete_table_index(ind.gametes[1]);
      yy:= gamete_table_index(ind.gametes[2]);

      assign_gametes(mom, buffer[ind.index][MOTHER]);
      assign_gametes(dad, buffer[ind.index][FATHER]);

      if (valid_individual(mom)) and (valid_individual(dad)) then
        begin

          init_gamete_probs(father_gamete_probs);
          init_gamete_probs(mother_gamete_probs);

          calc_gamete_probs(dad ,father_gamete_probs);
```

```
        calc_gamete_probs(mom ,mother_gamete_probs);

        for i:=1 to MAX_ALLELES do
          for j:=1 to MAX_ALLELES do

            for m:=1 to MAX_ALLELES do
              for n:=1 to MAX_ALLELES do

                begin

                  g1[1]:= i;
                  g1[2]:= m;
                  g2[1]:= j;
                  g2[2]:= n;

                  x:= gamete_table_index(g1);
                  y:= gamete_table_index(g2);

                  ii:= gamete_prob_index(i,j);
                  jj:= gamete_prob_index(m,n);

                  if (x = xx) and (y = yy) then

                     if (not zero_term(father_gamete_probs[ii])) and
                        (not zero_term(mother_gamete_probs[jj])) then

                        inc(count)

                end;

        if (count < 2) then
          begin

            g1[1]:= ind.gametes[1][1];
            g1[2]:= ind.gametes[1][2];
            g2[1]:= ind.gametes[2][1];
            g2[2]:= ind.gametes[2][2];

            if ((g1[1] < g2[1]) = (g1[2] < g2[2])) then
              hold:= Coupling

            else
              hold:= Repulsion

          end

      end

    end;

  get_phase:= hold

end; { function get_phase }
```

```
{ ------------------------------------------------------------------------ }

procedure set_individual(var ind: individual; index: byte);

begin

  assign_gametes(ind, index);

  if (valid_individual(ind)) then
    ind.phase:= get_phase(ind)
  else
    ind.phase:= Dont_care

end; { procedure assign_phase }

{ ------------------------------------------------------------------------ }

function valid_Otable_entry(ind: individual; E:Etable_ptr): boolean;

var
  i, j: byte;

begin

  if (not valid_individual(ind)) then
    begin

      valid_Otable_entry:= FALSE;
      exit

    end;

  bswap(ind.gametes[1][2], ind.gametes[2][1]);

  i:= gamete_table_index(ind.gametes[1]);
  j:= gamete_table_index(ind.gametes[2]);

  if (E^[i][j] = NIL) or (zero_term(E^[i][j]^)) then
    valid_Otable_entry:= FALSE

  else
    valid_Otable_entry:= TRUE

end; { function valid_Otable_entry }

{ ------------------------------------------------------------------------ }

procedure build_Etable(var E: Etable_ptr; var fgp,mgp: gamete_probs_type);

  function valid_probs(var t1, t2: term): boolean;

    function zero_term(x:term):boolean;

    begin
```

```
  zero_term:= TRUE;

  if (x.numerator.constant<>0) then
    zero_term := false
  else if (x.numerator.rm<>0) then
    zero_term := false
  else if (x.numerator.rf <> 0) then
    zero_term := false
  else if (x.numerator.onerm<> 0) then
    zero_term := false
  else if (x.numerator.onerf<> 0) then
    zero_term := false
  else if (x.numerator.rm_rf<> 0) then
    zero_term := false
  else if (x.numerator.onerm_onerf<> 0) then
    zero_term := false
  else if (x.numerator.rm_onerf<> 0) then
    zero_term := false
  else if (x.numerator.rf_onerm<> 0) then
    zero_term := false

end; { function zero_term }

{-------------------------------------------------------------------------}

function one_term(x:term):boolean;

  begin

    one_term:= TRUE;

    if (x.numerator.constant <> x.denominator.constant) then
      one_term := false
    else if (x.numerator.constant = 0) then
      one_term := false
    else if (x.numerator.rm<>0) then
      one_term := false
    else if (x.numerator.rf <> 0) then
      one_term := false
    else if (x.numerator.onerm<> 0) then
      one_term := false
    else if (x.numerator.onerf<> 0) then
      one_term := false
    else if (x.numerator.rm_rf<> 0) then
      one_term := false
    else if (x.numerator.onerm_onerf<> 0) then
      one_term := false
    else if (x.numerator.rm_onerf<> 0) then
      one_term := false
    else if (x.numerator.rf_onerm<> 0) then
      one_term := false

   end; {function one_term}
```

```
      {-------------------------------------------------------------------}

  begin

    valid_probs:= FALSE;

    if (not zero_term(t1)) and (not zero_term(t2)) then
      if (not one_term(t1)) or (not one_term(t2)) then
        valid_probs:= TRUE

  end; { function valid_probs }

  { ------------------------------------------------------------------- }

var
  i, j,
  m, n,
  x, y,
  ii, jj : integer;

  t      : term;

  g1, g2 : gamete;

  pt: pointer;

  alloc  : boolean;

begin

  init_Etable(E);

  for i:=1 to MAX_ALLELES do
    for j:=1 to MAX_ALLELES do

      for m:=1 to MAX_ALLELES do
        for n:=1 to MAX_ALLELES do

          begin

            alloc:= FALSE;

            ii:= gamete_prob_index(i,j);
            jj:= gamete_prob_index(m,n);

            if valid_probs(fgp[ii],mgp[jj]) then
              begin

                g1[1]:= i;
                g1[2]:= m;
                g2[1]:= j;
                g2[2]:= n;
```

```
                x:= gamete_table_index(g1);
                y:= gamete_table_index(g2);

                init_term(t);

                if (E^[x][y] = NIL) then
                  begin
                    mark(pt);
                    new(E^[x][y]);
                    init_term(E^[x][y]^);
                    alloc:= TRUE
                  end;

                multiply_terms(fgp[ii], mgp[jj], t);

                if (zero_term(E^[x][y]^)) then
                  assign_term(E^[x][y]^, t)

                else
                  add_terms(E^[x][y]^, t)

              end;

              if (alloc) and (E^[x][y] <> NIL) then
                if   (complex_term_val(E^[x][y]) = 0) then
                   begin

                     if (E^[x][y] <> NIL) then
                       release(pt);

                     E^[x][y]:= NIL

                   end

          end

end; { procedure build_Etable }

{ ------------------------------------------------------------------------ }

procedure add_Etables(var source, target: Etable_ptr);

var
  i, j: integer;
  t, p: term_ptr;

begin
  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      if (source^[i][j] <> NIL) then
        begin

          if (not zero_term(source^[i][j]^)) then
            begin
```

```
            new(t);
            assign_term(t^,source^[i][j]^);

            { create first node }

            if (target^[i][j] = NIL) then
              target^[i][j]:= t

            else { add next node }
              begin

                p:= target^[i][j];
                while (p^.next <> NIL) do
                  p:= p^.next;
                p^.next:= t

              end

          end

        end

end; { procedure add_Etable }

{ ------------------------------------------------------------------------ }

procedure add_to_Otable(var O: Otable_type; ind: individual);

var
  i, j: integer;

begin

  bswap(ind.gametes[1][2],ind.gametes[2][1]); { convert indicies }

  i:= gamete_table_index(ind.gametes[1]);
  j:= gamete_table_index(ind.gametes[2]);

  inc(O[i][j])

end; { procedure add_to_Otable }

{ ------------------------------------------------------------------------ }

function sum_Otable(var O: Otable_type): integer;

var
  i, j, hold: integer;

begin

  hold:= 0;
```

```
  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      hold:= hold + O[i][j];

  sum_Otable:= hold

end;

{ ------------------------------------------------------------------------ }

function sum_real_Etable(var E:real_Etable_type): real;

var
  i, j: integer;
  hold: real;

begin

  hold:= 0.0;

  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      hold:= hold + E[i][j];

  sum_real_Etable:= hold

end; { function sum_real_Etable }

{ ------------------------------------------------------------------------ }

procedure update_real_Etable(var target: real_Etable_ptr;
                             source: Etable_ptr;
                             O: Otable_type);

var
  i, j,
  sum  : integer;
  r    : real;

begin

  sum:= sum_Otable(O);

  for i:=1 to MAX_T_GAMETES do
    for J:=1 to MAX_T_GAMETES do
      begin

        r:= complex_term_val(source^[i][j]);
        target^[i][j]:= target^[i][j] + (sum * r)

      end

end; { procedure update_real_Etable }
```

```
{ ------------------------------------------------------------------------ }

function empty_Etable(var E: Etable_ptr): boolean;

var
  i, j: integer;

begin

  empty_Etable:= TRUE;

  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      if (E^[i][j] <> NIL) then
        empty_Etable:= FALSE

end; { function empty_Etable }

{ ------------------------------------------------------------------------ }

procedure build_np_real_table(var E: Etable_ptr);

var
  i, j, k, l: byte;
  sum, count: integer;

begin

  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      begin

        sum:=0;
        count:=0;

        if (E^[i][j] <> NIL) then
          for k:=1 to MAX_T_GAMETES do
            for l:=1 to MAX_T_GAMETES do
              if (E^[k][l] <> NIL) and equal_terms(E^[i][j]^,E^[k][l]^) then
                begin
                  sum:= sum + np_Otable[k][l];
                  inc(count)
                end;

        if (count <> 0) then
          np_real_Etable^[i][j]:= sum / count;

        if (count = 1) then
          begin
            global_Otable[i][j]:= global_Otable[i][j] - np_Otable[i][j];
            np_real_Etable^[i][j]:= 0;
            np_Otable[i][j]:=0
          end
```

```
      end; {j}

end; { procedure build_np_real_Etable }

{ ------------------------------------------------------------------------ }

procedure no_phase_analysis(ind, spouse: individual);

  procedure assign_individual(var target: individual; source: individual;
                              swap: boolean);

  begin

    target.gametes[1][1]:= source.gametes[1][1];
    target.gametes[1][2]:= source.gametes[1][2];
    target.gametes[2][1]:= source.gametes[2][1];
    target.gametes[2][2]:= source.gametes[2][2];

    if (swap) then
      bswap(target.gametes[1][2], target.gametes[2][2]);

    target.index:= 0;

    target.sex:= source.sex

  end; { procedure assign_individual }

  { ------------------------------------------------------------------------ }

var
  mcind, mrind, fcind, frind, npkid: individual;
  i, j, ii, jj: byte;

begin

  new(fnpg_probs);
  new(mnpg_probs);

  init_gamete_probs(fnpg_probs^[1]);
  init_gamete_probs(fnpg_probs^[2]);
  init_gamete_probs(mnpg_probs^[1]);
  init_gamete_probs(mnpg_probs^[2]);

  init_Etable(F1);
  init_Etable(F2);
  init_Etable(F3);
  init_Etable(F4);

  init_Etable(sum_np1_Etable);
  init_Etable(sum_np2_Etable);
  init_Etable(sum_np3_Etable);
  init_Etable(sum_np4_Etable);

  init_Otable(np_Otable);
```

```
if (ind.sex = MALE) then
  begin

    assign_individual(fcind, ind, FALSE);
    assign_individual(frind, ind, TRUE);
    assign_individual(mcind, spouse, FALSE);
    assign_individual(mrind, spouse, TRUE)

  end

else { ind is FEMALE }
  begin

    assign_individual(fcind, spouse, FALSE);
    assign_individual(frind, spouse, TRUE);
    assign_individual(mcind, ind, FALSE);
    assign_individual(mrind, ind, TRUE)

  end;

if (ind.phase = Unknown) and (spouse.phase = Unknown) then
  np_count:= 2

else
  np_count:= 1;

if (ind.phase=unknown) then
  begin
    if (ind.sex=MALE) then
      begin
        calc_gamete_probs(fcind,fnpg_probs^[1]);
        calc_gamete_probs(frind,fnpg_probs^[2])
      end
    else
      begin
        calc_gamete_probs(mcind,mnpg_probs^[1]);
        calc_gamete_probs(mrind,mnpg_probs^[2])
      end
  end
else
  begin
    if (ind.sex=MALE) then
      calc_gamete_probs(ind,fnpg_probs^[1])
    else
      calc_gamete_probs(ind,mnpg_probs^[2])
  end;

if (spouse.phase=unknown) then
  begin
    if (spouse.sex=MALE) then
      begin
        calc_gamete_probs(fcind,fnpg_probs^[1]);
        calc_gamete_probs(frind,fnpg_probs^[2])
```

```
      end
    else
      begin
        calc_gamete_probs(mcind,mnpg_probs^[1]);
        calc_gamete_probs(mrind,mnpg_probs^[2])
      end
  end
else
  begin
    if (spouse.sex=MALE) then
      calc_gamete_probs(spouse,fnpg_probs^[1])
    else
      calc_gamete_probs(spouse,mnpg_probs^[2])
  end;

build_Etable(F1,fnpg_probs^[1],mnpg_probs^[1]);
add_Etables(F1,sum_np1_Etable);

build_Etable(F2,fnpg_probs^[2],mnpg_probs^[1]);
add_Etables(F2,sum_np2_Etable);

build_Etable(F3,fnpg_probs^[1],mnpg_probs^[2]);
add_Etables(F3,sum_np3_Etable);

build_Etable(F4,fnpg_probs^[2],mnpg_probs^[2]);
add_Etables(F4,sum_np4_Etable);

dispose(fnpg_probs);
dispose(mnpg_probs);

for j:=1 to buffer_lines do
  begin
    if (buffer[j][MOTHER]=ind.index) or (buffer[j][MOTHER]=spouse.index) then
      begin

        set_individual(npkid,j);

        if (valid_Otable_entry(npkid, F1)) or
           (valid_Otable_entry(npkid, F2)) or
           (valid_Otable_entry(npkid, F3)) or
           (valid_Otable_entry(npkid, F4)) then
          begin

            add_to_Otable(np_Otable,npkid);
            add_to_Otable(global_Otable,npkid)

          end

      end

  end;

{ build_np_real_Etable}
```

```
  init_real_Etable(np_real_Etable);

  if not(empty_Etable(sum_np1_Etable)) then
    build_np_real_table(sum_np1_Etable)

  else if not(empty_Etable(sum_np2_Etable)) then
    build_np_real_table(sum_np2_Etable)

  else if not(empty_Etable(sum_np3_Etable)) then
    build_np_real_table(sum_np3_Etable)

  else if not(empty_Etable(sum_np4_Etable)) then
    build_np_real_table(sum_np4_Etable)

  else
    error('All sum_np_Etables are empty');

  { add to global expected tables }

  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      real_Etable^[i][j]:=real_Etable^[i][j] + np_real_Etable^[i][j]

end; { procedure no_phase_analysis }

{ ------------------------------------------------------------------------ }

function get_side(ind: individual): side_type;

var
  i: byte;

begin

  get_side:= RIGHT;

  for i:=1 to buffer_lines do
    if family[i].level = 0 then
      if (i <> ind.index) and (i <> family[ind.index].spouse) then
        if (i > ind.index) then
          get_side:= LEFT

end; { function get_side }

{ ------------------------------------------------------------------------ }

function w_string(i, j: integer): string;

var
  hold: string;

begin

  hold:='w';
```

```
  hold:= hold + index_to_gamete_str(i);
  hold:= hold + '_';
  hold:= hold + index_to_gamete_str(j);

  w_string:= hold

end; { function w_string }

{ ------------------------------------------------------------------------ }

procedure write_prod_term(var out:text; f:Etable_ptr; s:integer; V:string);

  procedure Add(variable: string);
    begin
      write(out, variable, ':= ', variable, ' + ')
    end;

  procedure Sub(variable: string);
    begin
      write(out, variable, ':= ', variable, ' - ')
    end;

  procedure Mul(variable: string);
    begin
      write(out, variable, ':= ', variable, ' * ')
    end;

  procedure Zip(variable: string);
    begin
      writeln(out, variable, ':= 0;')
    end;

  procedure One(variable: string);
    begin
      writeln(out, variable, ':= 1;')
    end;

  procedure comment(c: string);
    begin
      writeln(out);
      writeln(out, '{ ', c, ' }');
      writeln(out)
    end;

var
  i,j:byte;
  lead_term:boolean;

begin
  lead_term:=true;

  { PROD(fi^ni) }

  One(T1Var);
```

```
  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      begin
        if (np_otable[i][j]<>0) then
          begin
            Mul(T1Var);
            write(out,expn,'(');
            write_complex_term(out,f^[i][j]);
            if (np_Otable[i][j]<>1) then
              begin
                writeln(out,',',np_Otable[i][j],');')
              end
            else
              writeln(out,',1);');
            lead_term:=false
          end
      end;

  writeln(out,'');

  { write T }

  Zip(T2Var);

  lead_term:=true;
  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      if (f^[i][j]<>NIL) then
        begin
          Add(T2Var);
          write_complex_term(out,f^[i][j]);
          write(out,' * ');
          writeln(out,w_string(i,j),';');
          lead_term:=false
        end;

  writeln(out);

  writeln(out,V,':= ',T1Var,' * ', expn,'(', T2Var, ',',-s,');');

  writeln(out)

end; {procedure write_prod_term}

{-------------------------------------------------------------------------}

procedure calc_nophase_L(var out:text; header:boolean);

  procedure Add(variable: string);
    begin
      write(out, variable, ':= ', variable, ' + ')
    end;
```

```
  procedure Sub(variable: string);
    begin
      write(out, variable, ':= ', variable, ' - ')
    end;

  procedure Mul(variable: string);
    begin
      write(out, variable, ':= ', variable, ' * ')
    end;

  procedure Zip(variable: string);
    begin
      writeln(out, variable, ':= 0;')
    end;

  procedure One(variable: string);
    begin
      writeln(out, variable, ':= 1;')
    end;

  procedure comment(c: string);
    begin
      writeln(out);
      writeln(out, '{ ', c, ' }');
      writeln(out)
    end;

var
  i,j:byte;
  lead_term:boolean;
  lead_prod_term:boolean;
  s:integer;

begin
  if header then
    begin
      writeln(out,'Family #',buffer[1][FAMILY_NUM]);
      writeln(out,'log L (phase unknown) =');
      Zip(NVar);
    end;

  Zip(P1Var);
  Zip(P2Var);
  Zip(P3Var);
  Zip(P4Var);

  writeln(out);

  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      begin
        compress_complex_term(sum_np1_Etable^[i][j]);
        compress_complex_term(sum_np2_Etable^[i][j]);
        compress_complex_term(sum_np3_Etable^[i][j]);
```

```
      compress_complex_term(sum_np4_Etable^[i][j])
    end;

{ calc SUM(ni) }

s:=0;
for i:=1 to MAX_T_GAMETES do
  for j:=1 to MAX_T_GAMETES do
    s := s + np_Otable[i][j];

{ SUM ni log(wi) }

Add(NVar);
lead_term:=true;

for i:=1 to MAX_T_GAMETES do
  begin
    for j:=1 to MAX_T_GAMETES do
      if np_Otable[i][j] <> 0 then
        begin
          if not(lead_term) then
            write(out,' + ');
          if np_Otable[i][j] <> 1 then
            write(out,np_Otable[i][j],' * ');
          write(out,'ln(',w_string(i,j),')');
          lead_term:=false;
      end
  end;

if lead_term then
  write(out,'0');

writeln(out,';');

writeln(out);

{ log[prod(f1i^ni) * T1^(-SUM(ni)) + ... }

lead_prod_term:= true;

if not(empty_Etable(sum_np1_Etable)) then
  begin
    write_prod_term(out,sum_np1_Etable,s,P1Var);
    lead_prod_term:=false
  end;
if not(empty_Etable(sum_np2_Etable)) then
  begin
    write_prod_term(out,sum_np2_Etable,s,P2Var);
  end;
if not(empty_Etable(sum_np3_Etable)) then
  begin
    write_prod_term(out,sum_np3_Etable,s,P3Var);
  end;
if not(empty_Etable(sum_np4_Etable)) then
```

```
    begin
      write_prod_term(out,sum_np4_Etable,s,P4Var);
    end;

  Add(NVar);
  writeln(out,'ln(',P1Var,' + ',P2Var,' + ',P3Var,' + ',P4Var,');');

  writeln(out);
end; {procedure calc_nophase_L}

{-------------------------------------------------------------------------}

procedure perform_analysis;

var
  i,j,k: byte;
  ind,spouse,kid: individual;
  side: side_type;

begin

  for i:=1 to buffer_lines do

    { don't process level 2 kids }

    if family[i].level<>2 then
      begin

        np_count:=0;
        if (family[i].spouse <> 0) and (family[i].spouse > i) then
          begin

            set_individual(ind,i);
            set_individual(spouse,family[i].spouse);

            { check for complete typing and a spouse }

            if (valid_individual(ind)) and (valid_individual(spouse)) then
               if ((ind.phase=unknown) or (spouse.phase=unknown)) then
                 begin
                   no_phase_analysis(ind,spouse);
                 end

               else
                 begin

                   init_gamete_probs(father_gamete_probs);
                   init_gamete_probs(mother_gamete_probs);

                   if ind.sex = MALE then
                     begin

                       calc_gamete_probs(ind,father_gamete_probs);
                       calc_gamete_probs(spouse,mother_gamete_probs)
```

```
                    end

                  else { sex = FEMALE }
                    begin

                      calc_gamete_probs(ind,mother_gamete_probs);
                      calc_gamete_probs(spouse,father_gamete_probs)

                    end;

                  if ind.sex = MALE then

build_Etable(cur_Etable,father_gamete_probs,mother_gamete_probs)

                  else

build_Etable(cur_Etable,mother_gamete_probs,father_gamete_probs);

                  if (family[ind.index].level=0) then
                    begin

                      side:= get_side(ind);
                      if (side=LEFT) then
                        add_Etables(cur_Etable,sum_0a_Etable)
                      else
                        add_Etables(cur_Etable,sum_0b_Etable)

                    end
                  else
                    add_Etables(cur_Etable,sum_Etable);

                  for j:=1 to buffer_lines do
                    begin

                      if (buffer[j][MOTHER] = i) or (buffer[j][FATHER] = i) then
                        begin

                          set_individual(kid,j);
                          if (valid_Otable_entry(kid, cur_Etable)) then
                            begin

                              if (family[ind.index].level=0) then
                                begin

                                  side:= get_side(ind);
                                  if side=LEFT then
                                    add_to_Otable(L0a_Otable,kid)
                                  else
                                    add_to_Otable(L0b_Otable,kid)

                                end
                              else
                                add_to_Otable(Otable,kid);
```

```
                                add_to_Otable(global_Otable,kid)

                              end
                          end

                      end; {for j}

                  end { else phase<>unknown }

          end; { if (family...}

        if (np_count<>0) then
          begin

            calc_nophase_L(vb_nophase, TRUE);
            calc_nophase_L(vb_sum, FALSE)

          end

      end {for i}

end; {procedure perform_analysis}

{-------------------------------------------------------------------------}

procedure calc_L(var out:text; header:boolean; Otable:Otable_type;
                      sum_Etable:Etable_ptr);

  procedure Add(variable: string);
    begin
      write(out, variable, ':= ', variable, ' + ')
    end;

  procedure Sub(variable: string);
    begin
      write(out, variable, ':= ', variable, ' - ')
    end;

  procedure Mul(variable: string);
    begin
      write(out, variable, ':= ', variable, ' * ')
    end;

  procedure Zip(variable: string);
    begin
      writeln(out, variable, ':= 0;')
    end;

  procedure One(variable: string);
    begin
      writeln(out, variable, ':= 1;')
    end;
```

```
  procedure comment(c: string);
    begin
      writeln(out);
      writeln(out, '{ ', c, ' }');
      writeln(out)
    end;

var
  i,j:byte;
  t:term;
  s:integer;
  lead: boolean;

begin
  if header then
    begin

      writeln(out,'Family #',buffer[1][FAMILY_NUM]);
      writeln(out,'log L (phase known) =');
      Zip(LVar);
      writeln(out)

    end;

  if empty_Etable(sum_Etable) then
    exit;

  init_term(t);

  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      compress_complex_term(sum_Etable^[i][j]);

  { SUM(ni log(fi)) }

  Zip(T1Var);

  writeln(out);

  for i:=1 to MAX_T_GAMETES do
    begin
      for j:=1 to MAX_T_GAMETES do
        begin
          if (Otable[i][j]<>0) then
            begin
              Add(T1Var);
              if (Otable[i][j] > 1) then
                begin
                  write(out,Otable[i][j]);
                  write(out,' * ')
                end;
              write(out,'ln');
              write_complex_term(out,sum_Etable^[i][j]);
              writeln(out,';')
```

```
          end
        end {for j}
    end; {for i}

writeln(out);

Add(LVar);
writeln(out,T1Var,';');

writeln(out);

{ SUM(ni log(wi)) }

Add(LVar);

lead:=true;

for i:=1 to MAX_T_GAMETES do
  begin
    for j:=1 to MAX_T_GAMETES do
      begin
        if (Otable[i][j]<>0) then
          begin
            if not lead then
              write(out, ' + ');
            if (Otable[i][j] > 1) then
              begin
                write(out,Otable[i][j]);
                write(out,' * ');
                lead:= false
              end;
            write(out,'ln');
            write(out,'(');
            write(out,w_string(i,j));
            write(out,')');
          end
      end {for j}
  end; {for i}
writeln(out,';');

{ -SUM(ni)*log(SUM(fi wi)) }

{ SUM(ni) part }

Zip(T1Var);

s:=0;

for i:=1 to MAX_T_GAMETES do
  for j:=1 to MAX_T_GAMETES do
    s:=s + Otable[i][j];

{ log(SUM(fi wi)) part }
```

```
  Zip(T2Var);

  writeln(out);

  lead:= true;
  if (s>0) then
    begin
      for i:=1 to MAX_T_GAMETES do
        for j:=1 to MAX_T_GAMETES do
          begin
            if (sum_Etable^[i][j]<>NIL) then
              begin
                Add(T2Var);
                write_complex_term(out,sum_Etable^[i][j]);
                write(out,'*');
                writeln(out,w_string(i,j),';');
                lead:= false
              end
            end; {for j}
    end;

  if not (lead) then
    begin
      writeln(out);
      Sub(T1Var);
      writeln(out,s,' * ln(',T2Var,');')
    end;

  writeln(out);

  Add(LVar);
  writeln(out, T1Var,';');

  writeln(out);

end; {procedure calc_L}

{-------------------------------------------------------------------------}

procedure genotype_analysis;

var
  p: pointer;

begin

  mark(p);

  init_Etable(sum_Etable);
  init_Etable(sum_0a_Etable);
  init_Etable(sum_0b_Etable);
  init_Otable(Otable);
  init_Otable(L0a_Otable);
  init_Otable(L0b_Otable);
```

```
  init_Otable(np_Otable);

  writeln(debug, MemAvail);
  flush(debug);

  perform_analysis;

  writeln(debug, MemAvail);
  flush(debug);

  update_real_Etable(real_Etable,sum_Etable,Otable);
  update_real_Etable(real_Etable,sum_0a_Etable,L0a_Otable);
  update_real_Etable(real_Etable,sum_0b_Etable,L0b_Otable);

  calc_L(vb_file, TRUE, L0a_Otable, sum_0a_Etable);
  calc_L(vb_file, FALSE, L0b_Otable, sum_0b_Etable);
  calc_L(vb_file, FALSE, Otable, sum_Etable);

  calc_L(vb_sum, FALSE, L0a_Otable, sum_0a_Etable);
  calc_L(vb_sum, FALSE, L0b_Otable, sum_0b_Etable);
  calc_L(vb_sum, FALSE, Otable, sum_Etable);

  release(p)

end; { procedure_genotype analysis }

{-------------------------------------------------------------------------}

procedure gt(var out:text);

const
  RANGE = MAX_T_GAMETES * MAX_T_GAMETES;

var
  i, d:integer;
  g: single;
  o,e:array[1..RANGE] of single;
  x:array[1..RANGE-1] of single;
  j,k,c:integer;

  {=-=-=-=-=-=-=-=-=-=-}
  procedure prob;
  begin
        if (26.124 < g) then writeln(out,' .001 > P       ')
    else if (21.955 < g) then writeln(out,' .005 > P > .001')
    else if (20.090 < g) then writeln(out,' .01  > P > .005')
    else if (17.535 < g) then writeln(out,' .025 > P > .01 ')
    else if (15.507 < g) then writeln(out,' .05  > P > .025')
    else if (13.362 < g) then writeln(out,' .1   > P > .05 ')
    else if ( 7.344 < g) then writeln(out,' .5   > P > .1  ')
    else if ( 3.490 < g) then writeln(out,' .9   > P > .5  ')
    else if (g <= 3.490) then writeln(out,'        P > .9  ')
  end;
  {-=-=-=-=-=-=-=-=-=-=}
```

```
begin
  c:=1;
  for j:=1 to MAX_T_GAMETES do
    for k:=1 to MAX_T_GAMETES do
      begin
        o[c]:=global_Otable[j][k];
        e[c]:=real_Etable^[j][k];
        inc(c)
      end;

  g:= 0;
  d:= RANGE -1;
  for i:=1 to RANGE do
    if (e[i] = 0) then
      dec(d)

    else if (o[i] > 0) then
      g:=g + o[i] * ln(o[i]/e[i]);

  g:=2 * g;

  write(out,'G = ', g:8:3, '  df =',d:2);
  if d=8 then
    prob
  else
    writeln(out);

  writeln(out)

end; {procedure gt}

{-------------------------------------------------------------------------}

procedure output_totals;

var
  i, j: byte;
  r   : real;

begin
  writeln(obs_exp_file,'              OBS      EXP');
  writeln(obs_exp_file);
  for i:=1 to MAX_T_GAMETES do
    for j:=1 to MAX_T_GAMETES do
      if (global_Otable[i][j]<>0) or (real_Etable^[i][j]<>0) then
        begin

          write(obs_exp_file,'    ',w_string(i,j),' --> ',global_Otable[i][j]:3);
          writeln(obs_exp_file,'    ',real_Etable^[i][j]:9:5);

        end;

  r:= sum_Otable(global_Otable);
```

```
  writeln(obs_exp_file);
  writeln(obs_exp_file,'Total observed: ',r:2:1);
  writeln(obs_exp_file,'Total expected: ',sum_real_Etable(real_Etable^):2:1);
  writeln(obs_exp_file);

  gt(obs_exp_file)

end; {procedure output totals}

{ ------------------------------------------------------------------------ }

procedure init_geno_analyzer;

begin

  init_Otable(global_Otable);
  init_real_Etable(real_Etable)

end; { procedure init_geno_analyzer }

{ ------------------------------------------------------------------------ }

begin
end.
```

References

[Broo91] L. D. Brooks," Gene mapping methods with viability and gene interaction", Grant proposal submitted to NIH, June 1991.

[Cava71] L. L. Cavalli-Sforza and W. F. Bodmer, "The Genetics of Human Populations", San Francisco, CA: W. H. Freeman and Company, 1971.

[Daus90] J. Dausset et al., "Centre d'etude du polymorphisme humain (CEPH): collaborative genetic mapping of the human genome", Genomics, vol 6, pp. 575-77, 1990.

[Hart88] D. L. Hartl, "A Primer of Population Genetics", Sunderland, MA: Sinauer Associates, 1988.

[Suzu89] D. T. Suzuki, "An Introduction to Genetic Analysis", New York, NY: W. H. Freeman and Company, 1989.