DNA HAPLOTYPES DETERMINATION

DNA HAPLOTYPES DETERMINATION
FOR MEMBERS OF FAMILIES WITH PHENYLKETONURIA (PKU)

By

AFZAL MOHAMMED QURESHI, B.Sc.

A Project

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (1990) MCMASTER UNIVERSITY
(Computation) Hamilton, Ontario

TITLE: DNA Haplotypes Determination for Members of Families with Phenylketonuria (PKU)

AUTHOR: Afzal Mohammed Qureshi, B.Sc. (McMaster University)

SUPERVISORS: Professor R. Janicki
Professor P. Chang

NUMBER OF PAGES: vii, 147

ABSTRACT

Phenylalanine hydroxylase deficiency causes phenylketonuria (PKU) in humans. PKU is a recessive genetic disease that affects 1 in 10000 births among the Caucasian population. Its gene locus is highly polymorphic in its DNA sequence among different individuals and patients with PKU. DNA polymorphisms at the PAH gene locus are used to obtain haplotypes through restriction enzyme analysis. So far forty-six distinct RFLP haplotypes have been discovered in the human population. In theory, 384 distinct RFLP haplotypes can exist.

This project is to develop a program to assist the geneticists by obtaining haplotypes for each member of the PKU family. It uses information obtained from digestion of the DNA samples from the family members with the restriction enzymes. The restriction enzymes employed for this purpose are PvuII, BglII, EcoRI, MspI, XmnI, HindIII, and EcoRV.

The program "PKU" generates all possible haplotypes for each member of the PKU family. The generated haplotypes may include haplotypes from the forty-six defined haplotypes list or from the 338 other haplotypes that may fit the description from the restriction enzyme analysis. The program then carries out an elimination phase during which the "extra" haplotypes that had been generated for the family members but whose presence was not supported by the data from the other family members are eliminated from the individuals' haplotype lists. The remaining haplotypes are then used to determine a sibling's carrier status of the PKU disease, i.e., whether or not a sibling is a carrier of the PKU disease.

ACKNOWLEDGEMENTS

I would like to thank Dr. P. Chang from the Pediatrics Department, and Dr. R. Janicki from the Computer Science and Systems Department for their guidance and invaluable feedbacks in the completion of this project.

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES:

CHAPTER 1

The Introduction

Each human being comprises of billions and billions of cells. Among other organelles, sub-cellular structures, each cell contains a controlling factory called the nucleus. The most important thing about a nucleus is that it contains chromosomes, of which there are twenty-three pairs in the human species -- different species have different number of chromosomes in their nuclei. Chromosomes are made up of DNA (deoxyribosenucleic acid) and some proteins. The DNA is a macromolecule made up of a long chain of four smaller molecules called bases: Adenine, Cytosine, Guanine, and Thymine. The precise sequence of these bases that can occur in any number and in any order, makes up the genetic material of a person, i.e., it determines a person's phenotype (physical make-up) and when combined with the environment also determines his/her personality.

Each parent contributes 23 chromosomes (one-half of each chromosome pair) to each of his/her offspring. In other words, each chromosome of a chromosome-pair comes from a different parent. Both chromosomes of a chromosome-pair perform similar functions. Because of that, if one chromosome is defective in its coding of an enzyme, it normally would not affect a person because the other normal chromosome is still coding for that same enzyme. There will be a problem, however, if both chromosomes of a chromosome-pair were defective for the same enzyme, i.e., they both had mutations in the chromosomal region that encoded that enzyme.

Classical phenylketonuria (PKU) is a recessive disease, i.e., both chromosomes of the chromosome-pair would have to be defective if a person is to suffer from this genetic disease. This disease is characterized by an inability to metabolize L-phenylalanine, an amino acid -- amino acids are the building blocks for proteins. Although L-phenylalanine is an essential amino acid, only a fraction of the amount consumed in a normal diet is used for protein synthesis; most is converted to tyrosine, another amino acid. PKU results from an absence or severe deficiency of the liver enzyme phenylalanine hydroxylase (PAH). The enzyme normally catalyzes the oxidation of phenylalanine to tyrosine. The chromosome in humans that produces this enzyme is designated by the term 12q22-24.1 (Lidsky et al. 1985). Clinical symptoms of the disorder are severe and result in permanent mental retardation in untreated children (Chakraborty et al. 1987). This disorder affects 1 in 10000 births among Caucasians and has a carrier frequency of 1 in 50 (Sullivan et al. 1989).

Phenylalanine excess is particularly dangerous during the very early development of the brain. Special diets low in phenylalanine must be given to the patients, especially in the first few weeks of life. Most patients can relax their dietary restriction of phenylalanine by the age of eight. Interestingly, however, a carefully controlled low phenylalanine diet must be reinstituted during pregnancy. Otherwise, the fetus is exposed to excessive phenylalanine levels and the newborn infant, necessarily a heterozygote, will probably develp mental retardation. As is now commonly known, heterozygotes, clinically normal carriers, i.e., one normal and one defective chromosome of the mutant gene, demonstrate a reduction, usually about 40-50 per cent, of the mutant (defective) enzyme. The same holds true for phenylketonuria. This reduction, along with the high level of serum phenylalanine in the mother, greatly increases the risk of mental retardation.

There is no biochemical test for in utero detection of PKU. To solve the problem of prenatal diagnosis, molecular approaches have been utilized. DNA polymorphisms (genetic variations) at the PAH locus, the region of the DNA that codes for the PAH enzyme, have proved extremely effective in determining disease or carrier status in families with PKU (Daiger et al. 1989). This is done by digesting the DNA samples obtained from the family members with seven restriction enzymes that yield RFLPs, Restriction Fragment Length Polymorphisms. The seven restriction enzymes are BglII, PvuII, EcoRI, MspI, XmnI, HindIII, and EcoRV. Each of these restriction enzymes makes a cut at its specific site (polymorphic sites) in the PAH gene depending on the presence of a specific DNA base sequence at that site. The restriction enzyme PvuII has two restriction sites at the PAH locus; i.e., it can make a maximum of two restrictions (cuts) in the PAH gene, while the other six restriction enzymes used for this purpose can only make one cut each at the PAH locus. Overall then, there are eight polymorphic sites in the PAH gene. Haplotypes, an RFLP pattern, or a term used to describe variations of a chromosome, were assigned by determining the presence or absence of the eight polymorphic sites in the PAH gene (Sullivan et al. 1989).

Theoretically, $2^7 \times 3 = 384$ haplotypes can be generated by digestion with the 7 restriction enzymes described above -- for HindIII there are three possibilities: presence of the restriction site (designated by a "+" sign), absence of the restriction site (designated by a "-" sign), and the presence of a 4.4 kb HindIII allele (designated by a "=" sign). To date, only 46 haplotypes have been observed in the human population. These 46 haplotypes are listed in Table 1.

Most mutations that give rise to PKU do not generate any changes in restriction sites. Therefore, the PKU mutations can not be identified by alterations in RFLP patterns. However, some haplotypes are closely, but not exclusively associated with PKU chromosomes. Over all, haplotypes 1-4 represent 80% of the normal chromosomes and 82% of the PKU chromosomes. Haplotypes 1 and 4 are common among both normal and mutant chromosomes. Whereas, haplotypes 2 and 3 represent 40% of the PKU chromosomes and only 8% of the normal chromosomes (Sullivan et al. 1989). These figures refer to the Caucasian population only and do not hold true for other populations. For example, more than 80% of Oriental PKU chromosomes are associated with haplotype 4 only. The next most common haplotype in association with PKU chromosomes in the Oriental population is haplotype 7, which has an occurrence in about 6% of Oriental PKU chromosomes (see review by Chang et al. 1990).

What all this means is that a diagnosis of PKU and carrier detection based on intragenic linkage between PKU and RFLP is more likely to succeed in the Caucasian families than in the Oriental families. The success of a diagnosis depends on a family carrying sufficient RFLP heterozygosity (variety in its haplotypes) in order to be informative in linkage analysis. 87% of the Caucasian PKU families are informative for linkage analysis, while only about 40% of Oriental families are likely to be informative because of their lower heterozygosity (Daiger et al. 1989).

Determining the carriers of the PKU disease can be a tedious work. So far, the geneticists have been using "eyeballing" process to determine the haplotype candidates for the PKU patients and their family members. The information obtained from the restriction enzymes digestion is used to construct all of the haplotypes that can possibly be generated from that information. The list of these haplotypes which may be long or short is then compared with the list of the 46 defined haplotypes chart (Table 1) in order to obtain the haplotype numbers for the generated haplotypes.

After the same procedure has been carried out for each member of the family, a collective family analysis is done in which the list of haplotypes for each family member is narrowed down, to 2 each if possible. The "unwanted" haplotypes are eliminated from each member of the family wherever possible. This is easier to do with the children since everyone of their haplotypes must also belong to their parents' haplotype lists as well. It was their parents who passed their chromosomes on to them to begin with. In other words, those haplotypes that do not belong to either of the parents' haplotype lists are eliminated from the children's haplotype lists. Haplotypes from this shortened list are then compared with those of the patient's. If any one of them gives a match then the sibling is a carrier of the PKU disease.

With the parents the elimination of haplotypes is a little more difficult to do. One would require data from a minimum of two children in order to even consider eliminations of haplotypes from the parents' haplotype lists. There is no exact number for how many children might be required in order to narrow down parents' haplotype lists. Using one's intuition, it is usually not much of a problem to narrow the parents' haplotype lists down to 2 haplotypes each using the data from their children. However, there is no systematic approach to this. Any approach would have to be a trial and error approach.

One has to keep in mind, though, that some haplotypes generated for a family member may not belong to the defined haplotypes chart since there are 384 haplotypes that can exist in theory. These "undefined" haplotypes may get eliminated after a full family analysis has been done. If an undefined RFLP haplotype was to persist in the family then it would have to be passed on to the "scientific community" so that it could be added to the defined haplotypes chart in a later edition.

At any rate, the whole procedure of first determining the possible haplotypes for each family member and then narrowing of the haplotype lists for each family member after a collective family analysis, down to 2 haplotypes each where possible, can be a tedious and time consuming work. If a computer program could do this entire work for them, it would save the geneticists concerned with the PKU family analysis plenty of time. It would be a lot quicker to begin with, and also the human error factor would be eliminated. All the user would have to do would be to enter data obtained from the chemical tests (restriction enzyme digestions) for each family member, and the computer would do the rest.

There are a few things that would have to be kept in mind when designing such a program. One of them is that the chemical analysis while very effective is not perfect. Sometimes, digestion with an enzyme gives no results. This complicates the analysis because now it could mean any one of three possibilities for that particular restriction enzyme: positive for both chromosomes (cuts were made on both chromosomes by that enzyme), negative for both chromosomes (no cuts were made on either one of the chromosomes), or positive for one chromosome and negative for the other chromosome.

Another thing to consider is the difference in the treatment of defined and undefined haplotypes. Should the undefined haplotypes carry the same weight in determining the status of carriers of PKU disease, or should that job be left to the defined haplotypes alone. Most of the inputs for restriction enzyme cuts for an individual will involve undefined haplotypes as well as the defined haplotypes. In fact, logic dictates that for most restriction enzyme cut inputs, there should be more undefined haplotypes generated than the defined haplotypes. There are 338 undefined haplotype possibilities to choose from compared with the 46 defined haplotype possibilities. However, this is true in theory only. In practice, only the defined haplotypes, 46 of them, have been observed and dealt with in humans. Therefore, while undefined haplotypes might be generated along with the defined haplotypes for a family member, they should get eliminated after a full family analysis has been done. The question remains though what should be done if some of the undefined haplotypes remained persistent even after a full family analysis. Should their presence be pointed out to the users (geneticists) and then ignored in the carrier status determination or should they be treated just like the defined haplotypes.

Last, but not least, the program would have to be very user-friendly. It is a safe assumption to make that the majority of the people in the health science profession do not possess much knowledge in computers. The programmer would have to make sure that under no circumstances should the program crash.

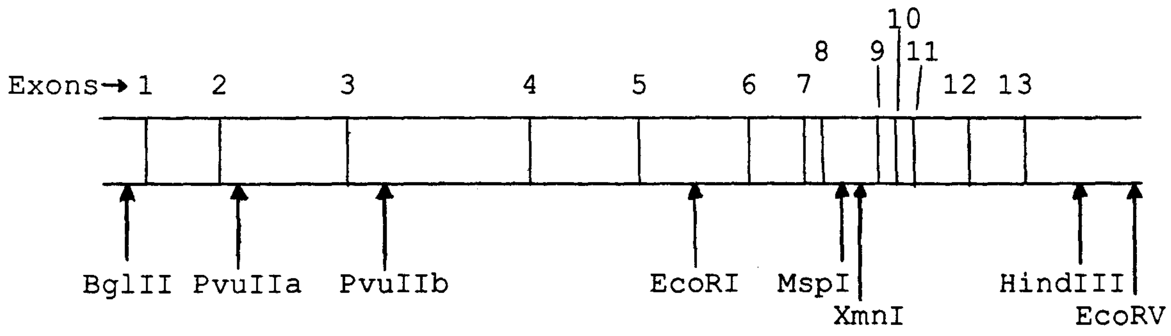


Figure 1: RFLP sites at the PAH locus. The molecular structure of the human PAH gene is shown schematically with its 13 axons (the portions of the gene that code for the PAH enzyme). The heavy arrows correspond to the polymorphic restriction sites in and immediately flanking the gene. (Woo, S.L.C. 1988)

Table 1: Defined Haplotypes Chart

	RFLP Haplotypes of PAH Locus							
	PvuIIa	BglII	PvuIIb	EcoRI	XmnI	MspI	HindIII	EcoRV
Haplotypes:								
1	+	-	-	-	-	+	-	-
2	+	-	-	-	-	+	+	+
3	+	-	-	+	+	-	-	-
4	+	-	-	+	+	-	+	+
5	-	+	+	+	-	+	-	+
6	-	+	+	+	-	+	-	-
7	-	+	-	+	+	-	-	-
8	+	-	-	+	-	+	-	+
9	+	+	-	+	-	+	-	+
10	+	-	-	+	-	+	-	-
11	-	+	-	+	-	+	-	+
12	+	-	-	-	-	+	=	+
13	+	-	+	-	-	+	=	+
14	-	-	-	+	-	+	-	-
15	-	-	-	+	-	+	-	+
16	+	-	-	+	-	-	+	+
17	+	-	-	+	-	-	-	-
18	-	+	+	+	+	-	+	+
19	+	-	-	+	+	-	-	+
20	-	-	+	+	-	+	-	-
21	-	+	+	+	-	+	=	+
22	+	-	-	+	-	+	=	+
23	-	+	+	+	+	-	-	-
24	+	-	-	-	-	+	+	-
25	+	-	-	-	-	+	=	-
26	+	+	-	-	-	+	-	-
27	+	-	-	-	-	+	-	+
28	+	-	-	-	+	-	+	+
29	-	+	-	-	-	-	-	-
30	+	+	-	-	+	-	-	-
31	+	-	-	+	+	-	+	-
32	-	+	-	-	-	+	-	-
33	-	+	-	-	-	+	-	+
34	-	+	-	+	-	+	-	-
35	-	+	-	+	+	+	-	+
36	-	+	+	-	-	+	-	-
37	-	-	-	-	-	+	-	-
38	-	+	+	-	-	+	=	-
39	+	-	+	+	-	+	-	-
40	-	+	-	-	+	-	-	-

41	+	-	-	+	-	+	+	+
42	-	+	-	+	-	-	-	-
43	+	-	-	-	+	+	-	-
44	-	+	-	+	+	-	+	+
45	-	+	-	+	+	-	+	-
46	+	+	-	-	+	-	+	+

RFLP haplotypes at the human PAH locus. The forty-six defined RFLP haplotypes are distinguishable according to their response to the seven restriction enzymes at the 8 restriction sites in the PAH locus. A plus sign (+) indicates the presence of the restriction site, a minus sign (-) indicates the absence of the site, and an equal sign (=) represents the 4.4 kb HindIII allele.

Table 2: The Relation Between The Symbols And The DNA Fragment Sizes

Symbol:	PvuIIa	BglII	PvuIIb	EcoRI	XmnI	MspI	HindIII	EcoRV
+	6.0	1.7	9.1	11.0	6.5	19.0	4.0	25.0
-	19.0	3.6	11.5	17.0	9.4	23.0	4.2	30.0
=							4.4	

The relation between the symbols and the DNA fragment sizes (in kb) obtained after digestion with the restriction enzymes at the 8 restriction sites. A cut made at the site by the enzyme is designated by a "+" sign, and a "-" sign indicates no cuts made at the site. The presence of a 4.4 kb HindIII allele is represented by an "=" sign.

CHAPTER 2

Project Developmemt

The problem specification for the project implied that the program, named "PKU", should store the defined haplotypes chart in it in some form. An alternative would have been to store the defined haplotype values in a separate data file -- a text or a binary file -- and then read from it for comparisons during the program execution. However, this option was less appealing because of the security factor involved. It would have been easy for some one to intentionally, or unintentionally, make changes to the data file, or even delete it. Therefore, the first option was chosen, and the values for the defined haplotypes were stored within the main program.

For the generation of the undefined haplotypes, there were again two options to select from: manually enter the 338 haplotypes values or make the program generate them. The advantage of the first option would have been less memory space and less compilation time; but, the vast number of haplotypes would have made it extremely difficult to pick up all of the mistakes that might have been made during their entry. Therefore, the second option was chosen because of the guarantee it offered in terms of the accuracy of the haplotypes. The draw back of this option was that it generated all 384 haplotypes; i.e., both defined and undefined haplotypes, instead of the required 338 undefined haplotypes. The overlapping 46 defined haplotypes were left to be dealt with in the further development of the program.

To make the program user friendly so people with limited knowledge in molecular diagnosis of PKU would also know what is being asked of them for inputs, a chart is displayed onto the computer terminal screen. This chart displays the names of the seven restriction enzymes used in the chemical analysis of the DNA samples obtained from each member of the PKU family. For each enzyme a listing is given of the sizes of fragments that can be obtained after digestion with that enzyme. One thing to keep in mind is that the smaller of the two numbers for each enzyme -- an exception is HindIII for which there are three numbers -- on the chart confirms that a cut was made by the enzyme on the DNA sample -- see Table 2. In other words, it confirms the presence of a polymorphic restriction site. This is designated by a "+" under the symbol column. The larger number for each enzyme refers to the fact that a cut was not made on the DNA sample by the enzyme; i.e., absence of a polymorphic restriction site. A "-" sign under the symbol column is used to represent this case. For HindIII, a third symbol, "=", designates the presence of a 4.4 kb HindIII allele.

Such a chart is displayed for each member of the family when prompting for inputs by the user. To avoid confusion and to eliminate any chances of errors in inputs, entries for each enzyme cuts are asked for separately, in separate lines: either "+", "-" (or "=" in the case of HindIII), and a null entry in the case of no or uncertain results obtained from chemical analysis. At the end of the entries for the eight restriction sites, the input is displayed back to the user in a tabular form so the user can tell whether or not mistakes were made in the input of the data. The user is then asked if any changes are required to the input data. If the answer is yes then the entries to each restriction enzyme are displayed one be one, and at the end of each display the user is asked if a change is required. If the answer is yes, then new input data is requested, otherwise, the input data for the next enzyme is displayed. Once this cycle has been completed, the complete input data -- newer version -- is displayed back to the user in a tabular form. The user is again asked if any changes to the input data are desired. The cycle will repeat itself until the user is satisfied with the input data and answers no to the change option.

The program then generates all of the haplotype candidates in a predefined sequence. After each haplotype has been generated, it is first checked with the defined haplotypes list. If a match is found then a flag is set for that particular haplotype in the list. In case of no match with the defined haplotypes list, a check is made with the list of the undefined haplotypes. A flag is then set for the undefined haplotype that gave a match. Once this process has been carried out for each generated haplotype, these haplotypes are displayed on the terminal screen. This display is carried out in two separate categories: under the "Defined Haplotypes" header, and under the "Other Haplotypes" header. In the first case, all of the defined haplotypes with the flag set are displayed preceded by their predefined numbers as described in Table 1. In the latter case, all of the undefined haplotypes with the flag set are displayed. These are preceded by their predefined numbers, but they also carry the letter "T" as a prefix. This prefix, chosen arbitrarily serves the purpose of distinguishing between the defined and the undefined haplotypes. It should be pointed out that this haplotype list for the individual is the list in full and no eliminations of haplotypes have been made yet.

The user is then asked if there is another family member that should be included in the family analysis for the RFLP haplotypes determinations. If the answer is yes then the above cycle is repeated in entirety for the next family member. This would continue until data have been entered for the entire family, in other words, until the user answers "no" to the following prompt:

```
CONTINUE WITH NEXT FAMILY MEMBER?(y/n) :
```

At this stage then, a full family analysis begins. Until this stage all of the numbers of the defined and undefined haplotypes for each family member had been stored in two lists; one for the defined haplotypes and the other for the undefined haplotypes. The eliminations of "unwanted" haplotypes begins with the children. If a child produced some haplotypes from the input values that did not belong to either of the parents' haplotype lists, they would be considered redundant and, therefore, eliminated from his/her haplotype list. This procedure would be carried out even if it resulted in a child producing a list with no candidates for possible RFLP haplotypes, i.e., an empty list. This would be a highly unlikely outcome since all of the chromosomes -- both of them -- for each child come from the parents and, therefore, must be present in the parents haplotype lists.

Next, eliminations of haplotypes from the parents' lists begins provided at least two children were included in the data inputs for the family. Those haplotypes that did not belong to any of the children's haplotype lists are eliminated. However, this elimination of haplotypes is not carried out if it would result in the defined haplotype list for the parent being reduced to carry less than 2 RFLP haplotypes. No such consideration is made of the undefined haplotypes though since they carry less weight in importance than the defined haplotypes.

After the eliminations have been carried out, thus, narrowing down the lists of possible RFLP haplotypes for each member of the PKU family, the display of haplotypes takes place again. As before, the haplotype candidates for the parents are displayed on the terminal screen first. Then the display for the patient occurs, followed by the display of haplotype candidates for the siblings. The display of haplotypes for the siblings occurs in the same order as they were entered by the user. To guard against a mix up with the siblings, which may happen especially with large PKU families, the siblings are assigned numbers in the same order as they were entered. The first sibling entered, or given the input data for, is assigned the number 1, the second sibling number 2, and so on.

Although at this stage the users (geneticists) can themselves compare the haplotype lists for each sibling with that for the patient and then determine whether or not the sibling was a carrier of the PKU disease, the program would do it for them. This would eliminate any chances of human errors, especially when concerned with large lists of haplotypes for some members of the family. The program first compares the defined haplotypes for the sibling with those for the patient. If there is a match, a diagnostic statement is displayed on the computer terminal screen:

```
DIAGNOSIS:  THE SIBLING IS A POTENTIAL CARRIER.
```

If there is no match then the undefined haplotypes for the sibling and the patient are compared. In the case of a match the above diagnostic statement is displayed followed by the statement below:

> NOTE: THIS CONCLUSION IS BASED ON THE ANALYSIS OF THE UNDEFINED HAPLOTYPES. BASED ON THE ANALYSIS OF THE DEFINED HAPLOTYPES ALONE, THE SIBLING IS NOT A CARRIER.

The reason for this statement is self explanatory: To distinguish between a carrier status based on the defined haplotypes and the status based on the analysis of the undefined haplotypes. In addition to being more informative, this also gives the geneticists a chance to make up their own minds as to whether the sibling in question truly is a carrier of the PKU disease depending on how much confidence they have on the undefined haplotypes.

Other things worth mentioning that were done solely for the purpose of making things more convenient for the users include addition of the function "get_string" in the program. This function reads a string value from the terminal screen and modifies it such that any blanks/spaces are excluded from the input string. If the user was to accidently hit the "space bar" during the input of data, he/she would not be asked to input the data again; the spaces will be ignored. This function was used every time the user was required to give an input. Other facilities deemed useful include taking the "return" key to mean "yes" by default whenever the user is asked whether he/she would like to continue analysis with the next family member, or to continue display of RFLP haplotype candidates for a family member. Also, throughout the execution of the program, the users are given the option of terminating the execution of the program, or even taking "short cuts" in the displays. An example of a "short cut" would be when display of the possible RFLP haplotypes for a family member is taking place. After the display of the defined haplotypes, the user is asked if he/she would like to continue with the display of haplotypes. If the answer given by the user was "no" then the display of the undefined haplotypes for that person would be ignored, and the program would continue with the next member of the PKU family.

The program "PKU" was designed such that it would be easy to add more defined haplotypes to it as they become available. This would be done by changing the constant NUM_DEFINED from 46 to whatever the number might increase to in the future, and then storing the new haplotype values in the function "haplo_assignments" in the same format as the other haplotype values. For example, if the new haplotype value was "++++++++" for the eight restriction sites in the PAH gene locus, then it would be stored in the function "haplo_assignments" in the following format:

```
strcpy (haplotypes[46].chrom_cuts, "++++++++");
```

Also, in the main part of the program "PKU" the number 46 would be updated to equal the same number as that assigned to the constant NUM_DEFINED. The only line that would need any changing would be the one that does the declaration for the structure "haplotypes"; i.e., the line:

```
     struct haplos   haplotypes[46], other_haplos[384];
```

would be changed to:

```
     struct haplos   haplotypes[X], other_haplos[384];
```

where the "X" refers to the new number for the total defined haplotypes to date.

The result would be an updated version of the program "PKU". To make cross referencing possible, if these newly defined haplotypes were to be generated from the input values for the restriction enzymes, the output would include these, and in brackets also the previous undefined haplotype numbers that they used to possess.

For example, if a new RFLP haplotype was added to the program with the value "--------" and assigned the number 47 then this would be done through the following sequence. First change the constant NUM_DEFINED to 47:

```
        #define  NUM_DEFINED  47;
```

Then, copy the new haplotype value into the structure haplotype; add the following line to the function "haplo_assignments":

```
     strcpy (haplotypes[46].chrom_cuts, "--------");
```

and then change the declaration line in the main function to:

```
     struct haplos  haplotypes[47], other_haplos[384];
```

The display of this RFLP haplotype if generated by the input values for the restriction enzymes would look like this:

```
     47 (T-382) -    -    -    -    -    -    -    -
```

where the number 47 refers to the defined haplotype number assigned to the value "--------", and T-382 refers to what this RFLP value previously used to be refered to as in the "Other Haplotypes" category.

CHAPTER 3

Algorithms

I. The Undefined Haplotypes List Generation:

The defined haplotypes were stored manually in the structure "haplotypes". The undefined haplotypes, however, were generated using an algorithm that resembles the "Bubble Sort Algorithm". The 384 haplotypes generated using this algorithm were stored in the structure "other_haplos".

This algorithm relies on a series of "for-loops" to do the job. Since each haplotype consists of eight places of symbols, eight "for-loops" are required to generate the 384 haplotypes. The algorithm looks like this:

```
i = 0  /* a counter to assign each generated haplotype
          a number */
buffer = "+-="

for q1 = 1 to 2
  haplotype[1] = buffer[q1]
  for q2 = 1 to 2
    haplotype[2] = buffer[q2]
    for q3 = 1 to 2
      haplotype[3] = buffer[q3]
      for q4 = 1 to 2
        haplotype[4] = buffer[q4]
        for q5 = 1 to 2
          haplotype[5] = buffer[q5]
          for q6 = 1 to 2
            haplotype[6] = buffer[q6]
```

```
                  for q7 = 1 to 3
                    haplotype[7] = buffer[q7]
                    for q8 = 1 to 2
                      haplotype[8] = buffer[q8]
                      i = i+1
                      other_haplos[i] = haplotype
                               /* storing the
                                  generated haplotypes
                                  in the other_haplo list */
                    end for
                  end for
                end for
              end for
            end for
          end for
        end for
      end for
```

II. <u>Haplotype Candidates Generation</u>:

After the user had provided the input values for each of the eight restriction sites, these values were used to generate the haplotype candidates for the individual. The algorithm used was similar to the one for the undefined haplotypes list generation. A series of "for-loops" were used to generate the haplotype candidates. However, the upper limits for the loops varied depending on the input values for the restriction sites.

```
/* initial upper limits to the for-loops */
p1=p2=p3=p4=p5=p6=p7=p8=2;

for X = 1 to 8     /* to check for any blank inputs */
 if RestrictionSite.X[1]=' ' OR RestrictionSite.X[2]=' '
   then RestrictionSite.X[1]='+' AND
                                 RestrictionSite.X[2]='-'
   if X=7      /* HindIII restriction enzyme site */
     then RestrictionSite.X[3] = '='
     p7 = 3
```

```
    end if
  end if
end for

/* to check if both input values for a restriction site
   are the same */
for N = 1 to 8
  if RestrictionSite.N[1] = RestrictionSite.N[2]
    then p.N = 1
  end if
end for

   for q1 = 1 to p1
     haplotype[1] = RestrictionSite1[q1]
     for q2 = 1 to p2
       haplotype[2] = RestrictionSite2[q2]
       for q3 = 1 to p3
         haplotype[3] = RestrictionSite3[q3]
         for q4 = 1 to p4
           haplotype[4] = RestrictionSite4[q4]
           for q5 = 1 to p5
             haplotype[5] = RestrictionSite5[q5]
             for q6 = 1 to p6
               haplotype[6] = RestrictionSite6[q6]
               for q7 = 1 to p7
                 haplotype[7] = RestrictionSite7[q7]
                 for q8 = 1 to p8
                   haplotype[8] = RestrictionSite8[q8]

                   COMPARE_AND_ASSIGN (haplotype,
                             Defined_Haplo_list,
                             Undefined_Haplo_List)
                   /* compare the generated haplotype
                      first with the Defined Haplotypes
                      list and if no match is found then
                      with the Undefined Haplotypes list
                      by calling this function   */

                 end for
               end for
             end for
           end for
         end for
       end for
     end for
   end for
```

III. Scheme For The Elimination Of Haplotypes

The haplotype lists that were generated for each member of the PKU family were stored in two dynamic lists: "defined_path" and "undefined_path". The defined_path list contained the defined haplotype numbers for the entire PKU family, while the undefined_path list contained all of the undefined haplotype numbers that were generated for that family.

The entire process of eliminations of unwanted haplotypes occured within these two lists. A zero (0) was used as a divider to set a family member's haplotypes apart from those of the other members of the family. The order in which these haplotypes were stored in the list was the same as the order of the family members in which the program had asked for their inputs:

```
defined_path (or undefined_path) -> parent1 haplotypes
-> 0 -> parent2 haplotypes -> 0 -> patient haplotypes
-> 0 -> sibling1 haplotypes -> 0 -> sibling2 haplotypes
-> 0 ->  ......  -> Nil
```

When the haplotypes for a family member were generated, they were "tagged" in the Defined and the Undefined Haplotypes lists. The "presence" field of the "haplos" structure was assigned the number '1' for the haplotypes that gave a match. These tags were then used to dynamically store the corresponding haplotype numbers in the defined_path list and the undefined_path list. The reason for using pointers (dynamic storage allocation) rather than the static arrays was that theoretically there is no limit to how many haplotypes can be generated for an entire family.

CHAPTER 4

Results

Once the program had been fully developed, several different combinations for restriction enzyme cuts were used to test its accuracy. The combinations used were hypothetical cases since the real life cases are only a subset of these hypothetical ones. As expected, for most inputs there were more undefined RFLP haplotypes generated than the defined RFLP haplotypes, the reason being, there are approximately seven times as many undefined haplotypes to choose from as there are defined haplotypes. In fact, there were cases when for certain inputs only the undefined haplotypes were generated; there were no defined RFLP haplotypes generated for these input values for the restriction enzymes.

There were a few inputs for which only the defined RFLP haplotypes were generated and no undefined haplotypes were produced. However, these instances were few compared with the cases when the opposite was true.

As had been stated earlier, for each restriction site in the PAH locus there are two possibilities: a positive (+) or a negative (-) response to the restriction enzyme digestion. An exception to this is the HindIII restriction site for which a third possibility also exists; the presence of a 4.4 kb HindIII allele (=). For each person there are two chromosomes containing the PAH locus. In other words, there are 2 PAH loci per person. While one PAH gene locus might give a positive response to a restriction enzyme, the other PAH gene locus might give a negative response. Each time there is a heterozygosity at a restriction site at the PAH locus; i.e., both "+" and "-" are shown to be the case by the chemical analysis, the number of RFLP haplotypes generated doubles. This number refers to the total number of generated haplotypes; both defined and undefined haplotypes combined. If "+" and "-" were entered for each of the eight restriction sites at the PAH locus, the total number of generated haplotypes would be 256 (2^8).

Generally, when the diversity in the input values for the restriction sites was increased in the test cases, the number of the defined RFLP haplotypes for the individual under analysis also increased. It should be noted, however, that there was no real sequence to the increase in the generated defined haplotypes. For example, if "+" and "-" were entered as the input values for PvuIIa, and "-" was entered for all of the other restriction sites, no defined haplotypes were generated. Increasing the heterozygosity in the input values by entering "+" and "-" for both PvuIIa and PvuIIb still generated no defined haplotypes. However, if "+" and "-" were entered for BglII, and "-" for the rest of the restriction sites, defined haplotype number 29 was obtained; and if "+" and "-" for EcoRI were entered as well then numbers 17, 29, and 42 were generated from the defined haplotypes chart. Entering "+" and "-" for PvuIIa, PvuIIb, BglII, and EcoRI, while the rest were all "-" again only generated defined haplotypes numbered 17, 29, and 42.

As mentioned earlier, the haplotypes found most frequently in the population are haplotypes 1-4, especially haplotypes 1 and 4. Using appropriate inputs for the eight restiction sites at the PAH gene locus, combinations of these four haplotypes were generated, see Table 3. Surprisingly, when the appropriate inputs were made to generate haplotypes 1 and 4 -- "+/+" for PvuIIa, "-/-" for BglII and PvuIIb, and "+/-" for EcoRI, XmnI, MspI, HindII, and EcoRV -- a rather large number of other defined haplotypes were also generated. They were haplotypes 2, 3, 8, 10, 16, 17, 19, 24, 27, 28, 31, 41, and 43. These defined haplotypes also match the description deduced from the input values for the eight restriction sites. Also interestingly, if we assumed that parent #1 had haplotypes numbered 1 and 4, and parent #2 had haplotypes numbered 2 and 3, then the corresponding input values for them would again generate haplotypes 1, 2, 3, 4, 8, 10, 16, 17, 19, 24, 27, 28, 31, 41, and 43 for each of the two parents -- see Table 3. Depending on how many children this couple had and what the haplotype composition for these children was, it is safe to assume that the majority of these haplotypes would be retained after a full family analysis.

For example, if the couple had 2 children, one with haplotypes 1 and 2 -- the corresponding input data for the eight restriction sites would generate haplotypes 1, 2, 24, and 27 -- and the other with haplotypes 3 and 4 -- the corresponding input data for the eight restriction sites would generate haplotypes 3, 4, 19, and 31 -- the final defined haplotypes list for each parent will include 1, 2, 3, 4, 19, 24, 27, and 31. While this list is shorter than the original list it is still a rather large one.

In such cases, the geneticists would have to use their intuition and experience to guess what the two haplotypes are for each parent and their children. For instance, they might assume that since haplotypes 1-4 occur in over 80% of the population, the other haplotypes may be ignored from each of the individual's list. As a result, the haplotype lists for the children become haplotypes 1 and 2 for one child and haplotpes 3 and 4 for the other child. Consequently, the haplotypes lists for the parents becomes either 1, 3 and 2, 4 or 1, 4 and 2, 3.

The thing that would make matters more complicated by increasing the list of haplotypes for the individual under study is when the chemical analysis gives no, or poor resolution, results for a given restriction enzyme. The assumption made by the program is that one chromosome was cut by the enzyme and the other chromosome was not. If the actual values were "+" and "-" anyways then everything would be fine, but if not then the number of generated haplotypes would be doubled. Especially in the case of HindIII, blank results from the chemical analysis would be entered as blank inputs for HindIII, in which case the assumption made is that the input values were "+", "-" and "=". In the worst possible case, where the actual input should have been +/+, -/-, or =/=, the end product is three times the size of the actual haplotype list for that individual. Even in the best possible case where the actual input values for HindIII should have been +/-, +/=, or -/=, the generated list of possible haplotypes is still 1 1/2 times larger than it should have been. There is no alternative to the above choice, however, since the actual values could have been any of the three possibilities.

In such a case though, the "unwanted" haplotypes from the haplotype list should get eliminated after a full family analysis since the "extra" haplotypes would not be supported by other members of the family. These extra haplotypes may by retained by the program "PKU", however, if some other members of the family also produced blank results from the chemical analysis.

Table 3: Haplotype Lists Generated By The Program "PKU" If Both Haplotypes Of An Individual Were From Haplotypes 1-4

The Two Haplotypes Possessed By An Individual	The List Of Defined Haplotypes Generated By The Program "PKU"
1/2	1, 2, 24, 27
1/3	1, 3, 10, 17, 43
1/4	1, 2, 3, 4, 8, 10, 16, 17, 19, 24, 27, 28, 31, 41, 43
2/3	1, 2, 3, 4, 8, 10, 16, 17, 19, 24, 27, 28, 31, 41, 43
2/4	2, 4, 16, 28, 41
3/4	3, 4, 19, 31

Haplotypes 1-4 are the most common haplotypes in the human population. The left column represents the two haplotypes from haplotypes 1-4 that an individual possesses. The right column represents the list of defined haplotypes that would be displayed by the program "PKU". As can be seen, extra haplotypes are also obtained for the inputs that would generate the two haplotypes on the left column. The longest list of haplotypes are generated for haplotype combinations 1/4 and 2/3.

Table 4: A Sample Analysis For A Family With No Heterozygosity At Any Of The Parents' Eight Restriction Sites

	The Two Haplotypes Possessed By An Individual	Final Haplotype Listings By The Program "PKU"
Parent 1	1/1	1
Parent 2	2/2	2
Patient	1/2	1,2
Each sibling	1/2	1,2

Contrary to a rather complex scenario illustrated in Table 3, this example illustrates the outcome by the program "PKU" for a family with no heterozygosity in the parents restriction sites. In this extreme case, no extra haplotypes are generated for any of the family members. Since the siblings' haplotypes match those of the patient's, each of the sibling will be declared a carrier by the program.. In reality, however, there is no way of telling whether a normal or a defective chromosome had been passed on to a sibling. That is why heterozygosity should be shown for at least one of the restriction sites for each parent for this linkage analysis method to be informative.

CHAPTER 5

Discussion

Application of RFLPs to the investigation of the molecular basis of the most common inhereted disease, PKU, has been useful in providing means for prenatal diagnosis. The program "PKU" was designed to assist the geneticists concerned with the study of PKU-affected families. The program asks for the information available from the chemical analysis, and from it gives a list of possible RFLP haplotype candidates for the individuals under study. After obtaining such information for each member of the family, the program "PKU" does a collective family analysis and eliminates as many of the "unwanted" haplotypes from each family member's haplotypes list as permissible within safe limits. This narrows down the lists containing the haplotype candidates for each individual. The final haplotype lists for each member of the family are then displayed along with a diagnostic statement for the siblings stating whether or not they are carriers of the PKU disease.

The program "PKU" provides two options to the users: either determine haplotype candidates for the patient alone, or do a full family analysis whereby haplotype candidates for each member of the PKU family are determined. As stated earlier, for a full family analysis, this program determines whether or not a given sibling is a carrier of the PKU disease, which may leave one wondering why the parents' data is required. After all, it is the patient's haplotypes that are compared with those of the sibling's, and if a match is detected the sibling is declared a carrier of the PKU disease. There are two reasons for the inclusion of the parents data into the family analysis option. Number one, so that a separate analysis would not be required if the geneticists wanted to know the parents' haplotypes; and the second reason being that a combined data from the parents and children would be used to shorten the lists of possible haplotypes for each member of the family. This should make the final analysis easier for the geneticists.

As one would have expected, generally more undefined haplotypes were generated than the defined haplotypes for an individual. Even after a full family analysis this held true in the test cases. Another thing that generally held true in the test cases was that the final number for the total number of RFLP haplotypes for an individual was greater than two even after elimination of the unwanted haplotypes.

The first case can easily be explained by the fact that there are more than seven times as many undefined haplotypes as there are defined haplotypes. Therefore, since there are more undefined haplotypes to choose from than the defined haplotypes, one would expect to find more undefined haplotypes in the final outcome.

The second case is a little more complicated to explain. One has to keep in mind that more heterozygosity (diversity) in the inputs for the restriction sites -- +/- instead of +/+, -/-; and in the case of HindIII, +/-, +/=, or -/= instead of +/+, -/- or =/= -- would result in more haplotypes generated. In fact, each time heterozygosity is increased, or a blank input is made, the total number of generated haplotypes duplicates. The reason behind using a lot of heterozygosity in the test cases was to generate plenty of haplotypes so the accuracy of the program "PKU" could be determined. The bigger the haplotype lists for each individual of a family, the more overlap in the haplotypes is likely to occur; i.e., more haplotypes should occur in common within the family, or more importantly between the children and their parents. As a result, more haplotypes would be retained after a full family analysis. Only those haplotypes from the children's haplotype lists are eliminated that do not belong to either of the parents' lists. The same is true for the parents as only those haplotypes are eliminated from their lists that do not match any of the haplotypes of any of their children. However, in the case of the parents, the elimination of the defined haplotypes is witheld if it would result in their total being less than two.

In real life, however, a person can only have two haplotypes, and also, so far only the 46 defined haplotypes have been observed and none of the 338 undefined haplotypes that were observed in the test cases. These two occurrences are actually related to each other and when examined closely might explain the contradiction that seemingly appears between the test cases and the real life cases.

It is true that the program "PKU" generates a good number of undefined haplotypes for a given set of inputs for the eight restriction sites at the PAH locus. The program "PKU" was designed to generate "all" of the defined and undefined haplotypes that fit into the description of the input data. If more than two haplotypes remained for an individual under study after a full family analysis, then it means that these haplotypes satisfied all of the conditions that were set for a haplotype to remain in the final listing. If the undefined haplotypes also satisfied these conditions then they too would be retained. Further conditions could have been set to narrow down the final listing to two haplotypes for each member of the family, but they could very well have eliminated some legitimate haplotypes. While they might have worked some times, other times they might not have. To be on the safe side and have the program "PKU" useful to the geneticists at all times, it was deemed necessary to keep the elimination conditions simple and leave the reduction of the haplotypes down to two for each member of the family up to the geneticists. Using their superior knowledge in genetics and their intuition, once the geneticists have narrowed the final number of haplotypes down to two for each member of the PKU family, it seems likely that in most cases they will find the undefined haplotypes unnecessary and redundant, and therefore, eliminate them as well as some of the defined haplotypes. This way both problems -- the presence of undefined haplotypes and the total number of haplotypes being greater than two -- would be solved.

This brings up the question of how important are the undefined haplotypes relative to the defined haplotypes? The addition of the undefined haplotypes list to this program gives it an added dimension. It provides the geneticists with a diversity in the haplotypes to choose from. While they may get eliminated most of the times by the geneticists, there might be the odd time when they actually might be retained in the final listing. After all there was a time when all 46 defined haplotypes would have been part of the undefined haplotypes list. New RFLP haplotypes are being discovered every year. By 1988, 43 distinct PAH RFLP haplotypes had been reported in the European populations (Woo 1988). By 1989, three additional PAH RFLP haplotypes had been found in Hungary and Czechoslovakia to bring the total to 46 defined haplotypes (Daiger et al. 1989). Once more populations are screened, or more people from the populations already screened are checked for their PAH RFLP composition, more RFLP haplotypes are likely to be found and added to the defined haplotypes chart. In such a case, those newly found RFLP haplotypes would be removed from the existing undefined haplotypes list. However, for now even though they have not been found in the human population yet, the geneticists are given the option of having a look at them nonetheless.

Once discovered, new haplotypes can easily be added to the defined haplotypes list in the program "PKU". The program was designed so that it could be updated easily. For the purpose of cross referencing, if these newly added RFLP haplotypes were to be generated by matching a set of inputs for the eight restriction sites, the program would also display the haplotype numbers they used to possess when they were under the "Other (undefined) Haplotypes" category. The idea was to make the program "PKU" as useful to the geneticists as possible, provided the RFLPs within the PAH structural locus are still used for tracing the inheritance of the PKU allele (chromosome) in an affected family.

In order for this method to be informative for linkage detection of affected or carrier children it is important that a carrier parent be heterozygous at one or more of the eight RFLP sites. The reason is, if there was no heterozygosity found at any of the eight restriction sites in the PAH locus then it would mean that both chromosomes of the chromosome-pair containing the PAH gene have the same haplotype number. Both the PKU (defective) chromosome and the non-PKU (normal) chromosome possess the same haplotype number. Through this method of analysis it would be impossible to determine which of the two chromosomes, i.e., the normal or the defective chromosome, had been passed on to the children. In Caucasian families the chances of this happening are low, as 87% of them show heterozygosity. In Oriental families though this method would not have the same success, as only 40% of them are heterozygous at the RFLP sites. Despite this, some successful cases of RFLP linkage analysis in determining the fetuses' status in the prenatal diagnosis of Oriental families have been reported (<u>see</u> review by Chang et al. 1990). For other populations, the frequency of variability (heterozygosity) in the haplotype composition of a parent lies between the two numbers mentioned above.

These RFLP haplotypes are used only as markers to determine which chromosomes are possesed by each member of the family, and which of them are defective. From the information obtained from the patient, it is determined which haplotypes belong to the defective chromosomes. If a sibling was to possess one of those chromosomes, i.e., the same haplotype number, the sibling must be a carrier. If both of a sibling's chromosomes were identical to those of the patient's, i.e., both RFLP haplotypes matched, the sibling would be declared "affected" by the PKU disease. In other words, these polymorphisms (haplotype variations) are a natural variation in the population and are not directly related to the PKU mutations.

However, of the 46 total haplotypes 10 of them have been found to be uniquely associated with PKU-bearing chromosomes. They are haplotypes 15, 17, 22, 25, 34, 36, 38, 39, 40, and 46. Twelve haplotypes -- haplotypes 13, 19, 21, 23, 29, 31, 33, 35, 37, 43, 44 and 45 -- have been found unique to normal chromosomes. The remainder are found in both cases (Daiger et al. 1989). This is true in the Caucasian population only. Different results have been obtained from the Asian populations. Two of the haplotypes -- haplotypes 2 and 17 -- are unique to PKU-bearing chromosomes. Three of the haplotypes -- haplotypes 6, 10, and 23 -- are unique to non-PKU chromosomes. The sample size from the Asian families was small, and because of that their results may not be very significant (Daiger et al. 1989). These assignments of certain haplotypes to either PKU-bearing chromosomes or to non-PKU chromosomes can easily change over time when more people (families) have been screened. Therefore, their significance, especially for the association of haplotypes with non-PKU chromosomes might not carry much weight. Although, for the haplotypes' association with PKU-bearing chromosomes may turn out to be of much significance if the changes in the base sequence of DNA that caused the production of these particular haplotypes are also shown to be the cause of the PKU disease. Until all mutations causing PKU have been identified, such a relation of haplotypes with the PKU disease does not mean much in determining a person's PKU status through his/her haplotypes determination alone. In the meantime, analysis of the entire family would have to be carried out in order to determine a sibling's PKU status, i.e., whether or not a sibling is a carrier or is affected by the PKU disease.

While the program "PKU" determines a sibling's carrier status, it does not inform the users whether or not a sibling is affected by the PKU disease. A sibling would be affected by the PKU disease if both of his/her chromosomes (haplotypes) were identical to those of the patient's. This could be determined if the input data for the eight restriction sites in the PAH locus produced exactly two haplotypes for each member of the PKU family after a full family analysis. However, as the results showed, obtaining two haplotypes for each member of the family from the given data was an exception, rather than the norm. Usually, the total number of haplotypes generated was more than two. Any attempts made to determine whether a sibling was affected by the PKU disease without first having narrowed the haplotypes number down to two would be futile. The idea is to have a program that is consistent and reliable in its diagnosis. Attempting to determine a sibling's PKU status in terms of being "affected" by the disease would make the program less reliable since the conditions set would not work every time.

As an example, consider two options that might have been chosen in such a case. First, the condition being if any two or more of a sibling's haplotypes matched those of the patient's, the sibling is to be declared "affected" by PKU. In setting this condition one would be disregarding the fact that once the geneticists narrow down the final haplotype listings to two for each member of the family, only one or even none of those haplotypes that the sibling had in common with the patient might remain in the final listing. In other words, one, or even two, of the haplotypes that did not match may be retained after the final analysis. The sibling then is no longer affected by the PKU disease. The sibling may only be a carrier or even a non-carrier if one or both chromosomes are normal, respectively. If a second choice was made whereby all of the haplotypes of the sibling must match those of the patient's before making the diagnosis then another problem arises: what if the total number of haplotypes for the sibling did not match that for the patient?

This can definitely happen if the sibling or the patient gave a blank result for a chemical test, in which case the number of haplotypes for that person may double the actual number. The sibling would be declared "not affected" by the PKU disease since not all of the haplotypes for the sibling and the patient gave a match. For that matter, if such a stringent condition was set it could back fire in a different way as well. Say, if the majority, but not all, of the haplotypes for the sibling and the patient matched, the sibling would not be declared "affected" by the PKU disease because not all of their haplotypes matched. However, such a diagnosis, or the lack of it, would be false if the two haplotypes retained in the final analysis by the geneticists for the patient and the sibling both matched. In such a case, the sibling was actually affected by the PKU disease but the geneticists would have been misled because they would have expected the sibling to be declared "affected" by the program if he/she really was affected.

That is why it was deemed best to let the geneticists make the final decision on whether or not a sibling is affected by the PKU disease. Once they have produced the final two haplotypes for each member of the PKU family from the haplotype listings given by the program "PKU", it would be extremely easy for the geneticists, or any one else for that matter, to make the final diagnosis on a sibling. If both haplotypes of a sibling matched the two haplotypes for the patient then the sibling is declared affected, otherwise, the sibling is not affected. Any attempts made to determine such a status for a sibling before coming up with final two haplotypes for each member of the PKU family may prove futile in a lot of the cases. They may even lead to a wrong diagnosis causing complications for the PKU family and eventually rendering the program "useless" to the geneticists.

One may argue then why a diagnosis on the "carrier" status of a sibling is made by the program "PKU". It is possible that after the geneticists narrow the haplotype lists down to two haplotypes for each member of the family, a sibling who had been declared a carrier previously may no longer be diagnosed as a carrier. In such a case, the original diagnostic statement was wrong. However, what sets diagnosis of a "carrier-status" apart from that of the "affected-status" is that only a simple condition is set for the determination of the carrier status of a sibling. The condition being if any of the haplotypes of a sibling matches any of the haplotypes of the patient, the sibling is declared a carrier. When the program displays the statement:

```
DIAGNOSIS: THE SIBLING IS A POTENTIAL CARRIER
```

the idea is to warn the geneticists that at least one of the haplotypes of the sibling matches those of the patient's in the "current" haplotype listings. That is why the term "potential carrier" is used because it is realized that after further eliminations, the sibling may no longer be diagnosed as a carrier. The diagnosis below is self-explanatory:

```
DIAGNOSIS: THE SIBLING IS A POTENTIAL CARRIER

NOTE:      THIS CONCLUSION IS BASED ON THE ANALYSIS
           OF THE UNDEFINED HAPLOTYPES.  BASED ON
           THE ANALYSIS OF THE DEFINED HAPLOTYPES
           ALONE, THE SIBLING IS NOT A CARRIER.
```

The idea is to inform the geneticists that while there is a match between the haplotypes of the sibling and the patient, the match occurs only in the undefined haplotypes. In case the geneticists put little, or no, weight in the results for the undefined haplotypes they should be informed on how the diagnosis was reached.

While the condition set for determining a sibling's carrier status is simple, agreeable, and self-explanatory, the same could not have been said for condition(s) set for the determination of a sibling's "PKU-affected" status. The condition(s) set would have been complicated, and agreed to by some but disagreed with by others. The condition(s) would have had to be stated in some form to the users so they could decide for themselves if they had any faith in the condition(s) set, and therefore, if they can trust the output of the program.

Another point worth mentioning is that while in the test cases a variety of different inputs were used for each member of the family, in real life the input values will not be vastly different from each other for the children of the PKU family. In the test cases, generally large numbers of haplotypes were generated for each member of the family. Because of the variation used in the inputs, the overlap in the haplotypes of the children was little compared with the haplotypes that the patient and the siblings did not have in common. In such relatively extreme cases, once the haplotypes had been narrowed down to two for each member of the family, the haplotypes that the patient had in common with some of the siblings may had been eliminated. As a result the diagnosis of a sibling being a "potential carrier" may had been proven false. In real life, however, if the program "PKU" declared a sibling a "carrier", then the sibling may very well turn out to be a carrier of the PKU disease. The reason is that the input values for the eight restriction sites for each child are not random values the way they were in the test cases. Since it was their parents who passed the chromosomes on to their children in the first place, the input values for the children will be dependent on the input values for the parents. Because the values for each child are coming from the same limited pool of data -- from their parents who between them have only four chromosomes to pass on to their children -- the generated haplotypes for the children should have a good deal of overlap. While it is true that "extra" haplotypes would also be generated for each member of the family, there should be overlap in these extra haplotypes as well since they were generated from the same pool of inputs. Therefore, if the diagnosis reads that a sibling is a potential carrier then he/she probably will end up being declared a carrier after the haplotypes have been narrowed down to two for each member of the family. But whether the sibling ended up being a carrier or not, the program "PKU" had accomplished the job of "alarming" the geneticists of the possibility that the sibling might be a carrier of the PKU disease.

If the program "PKU" was to display the statement:

DIAGNOSIS: THE SIBLING IS NOT A CARRIER

then regardless of what the final haplotype listing of the sibling ended up being, the sibling could not be declared a carrier of the PKU disease. The reason is the final two haplotypes for an individual will be chosen from the set of haplotypes generated for him/her by the program "PKU". Obviously, if the entire set of haplotypes generated for the sibling had no overlap with the entire set of haplotypes generated for the patient then it is safe to assume that a subset of haplotypes for the sibling also would not have any haplotypes in common with a subset of haplotypes for the patient. Therefore, unlike the diagnostic statement declaring a sibling to be a "potential carrier" of PKU which has a possiblity of being false, the diagnostic statement declaring a sibling "not a carrier" of PKU, i.e., no defective chromosomes, has no chance of being proven false. In other words, the geneticists can agree with the statement without first having to narrow down the haplotype list to two haplothypes for that sibling.

The conditions set for eliminating "unwanted" haplotypes for the children were relatively straight forward. The haplotypes for each child were checked and compared with those of the parents. Those haplotypes that belonged to either, or both, of the parents were retained, while those that belonged to neither of the parents were eliminated since all of the chromosomes for each child were obtained from their parents. Any haplotypes that suggested otherwise had to be the "extra" haplotypes generated from heterozygosities at the restriction sites at the PAH locus where homozygosity should have been the case. In other words, the data input for a restriction site for a child was +/- when it should have been either +/+ or -/-. An error in a chemical test, or in its outcome's interpretation for that restriction site, or a blank result for that test would lead to the assumption by the program "PKU" that the entries for that particular restriction site were heterozygous, +/-. A blank entry for HindIII would lead to the assumption that the input was +/-/=. The program would then remove half of the haplotypes from the child's haplotype list since each heterozygosity at a restriction site doubles the generated haplotypes.

There would be a problem, however, if the input data for a restriction site for at least one of the parents was +/-. If the actual input data for a child should have been +/+ or -/-, i.e., homozygous, but due to the possibilities mentioned above, heterozygosity (+/-) was entered upon the prompt for that particular restriction site, then some of the extra haplotypes generated because of that error might be retained by the program. The data entered for the parents might support the existence of some of the extra haplotypes generated for the child because of the error of entering heterozygosity for a restiction enzyme site where homozygosity should have been the case. Later on, the geneticists might be able to reduce those extra haplotypes with the knowledge that "uncertain" results had been obtained from the chemical tests for that particular restriction site. But the program has no way of knowing or distinguishing between suspicious chemical test results from those of feasible results. Therefore, in the above mentioned extreme case, the "extra" chromosomes for the children might be retained in the final haplotype listings.

The conditions set for the eliminations of "unwanted" haplotypes from the parents' lists were a little more complicated. After the unwanted haplotypes from the chilren's haplotypes lists had been eliminated, those lists were used as guidelines for the eliminations of the "extra" haplotypes from each of the parents' lists. Those haplotypes from the parents' haplotype lists that did not belong to any of the childrens' haplotype lists were to be eliminated, thus, narrowing down the haplotype lists for the parents. However, the danger that lay here was that there was a good possibility that even the legitimate haplotype(s) might get eliminated from the parents' haploytype lists. That danger did not lie with the children during their haplotype eliminations because each child's haplotype list is a subset of the parents' haplotype lists combined. Therefore, with children there is no danger of eliminating legitimate haplotypes.

The insecurity in carrying eliminations of "unwanted" haplotypes from the parents' lists in the same fashion as with the children's "unwanted" haplotypes stems from the probabilities involved in a parent passing both of his/her chromosomes to his/her children. For each child, both parents contribute equally in the make up of his/her genetic material. One-half of the child's chromosomes come from the male parent and the other half from the female parent. In the final analysis of the haplotypes, therefore, one of the haplotypes for each child should be from the male parent and the other haplotype from the female parent. The probability of a parent passing one particular chromosome of the chormosome-pair to a child is one-half. The probability of that parent passing the same chromosome of the chromosome-pair, later on, to the next child is still one-half. It is not necessarily the case that the second child will get the other chromosome from the chromosome-pair. Therefore, if only one chromosome of the parent's chromosome-pair had been passed on to the offspring, then the other chromosome of the chromosome-pair while present in the original list, would be missing from the final haplotype list. None of the children would support its existence. In a small family especially, the chances of this happening are quite high.

One way to guard against the elimination of a legitimate haplotype from the parents' haplotype lists is to choose a number for how many children must first be included in the analysis before eliminating any of their haplotypes. The number is arbitrary but it must be greater than 2 because a family would need at least two children for the parents to have a chance of passing both of their chromosomes of the chromosome-pair to their children. As the family size increases, the chances of this happening also increase. However, if a rather large number was chosen then unless a family had that many children, the parents haplotype lists would not be reduced by the program. It would make things a little more complicated for the geneticists, as they will have relatively large sets of haplotypes for the parents lists from which to eliminate all but two haplotypes for each parent. Things could have been simplified by the computer, but such was not done because the family did not have the required number of children.

To avoid this a second option was chosen whereby eliminations were to take place as long as the family had at least two children included in the PKU analysis -- with only one child in the family it would be nonsense to do any haplotype eliminations for the parents, as only one chromosome from each parent would have been passed on to the lone child. These eliminations are only temporary, as the program "PKU" then checks if the number of the remaining defined haplotypes equals or exceeds 2. If the number of the defined haplotypes is less than two, which can be the case for smaller families, the eliminations are withheld and the original haplotype list for the parent is displayed for the final listings. If the number of the remaining defined haplotypes is greater than or equal to 2 for the program's final analysis for the parent, then the eliminations are withstood and a shorter haplotype list from the original one is displayed on the terminal screen.

One may argue that the condition should have been for the total number of haplotypes of a parent, i.e., defined and undefined haplotypes combined rather than the defined haplotypes alone, to be greater than two before the eliminations of haplotypes for that parent should take place. The present scheme ignores any importance that the undefined haplotypes may represent. It is a fact that a person can only possess two haplotypes, either both defined haplotypes, both undefined haplotypes, or one defined and one undefined haplotype. Because of that, the condition set for eliminations should have been for the total number of haplotypes to be greater than or equal to two.

That was not done in the program "PKU", however, and the undefined haplotypes were ignored in the elimination conditions. The logic behind this is that until now only 46 haplotypes, the defined haplotypes, have been discovered in the human population. There should be more importance given to them than the undefined haplotypes, which have not been found in the human population yet. New haplotypes are being discovered in the human population every year, and because of that some of the currently undefined haplotypes may become part of the defined haplotypes chart in the future. But for now they should not carry the same weight in importance as the defined haplotypes. For now the undefined haplotypes are to be considered as haplotypes generated as "side effects" of the input values that generate the defined haplotypes. If the conditions set for eliminations included the undefined haplotypes as well then there would have been the danger of legitimate defined haplotype(s) being eliminated from a parent's haplotype list. To guard against this, the undefined haplotypes are ignored in the elimination conditions. Also, the geneticists would probably feel more comfortable with the final two haplotypes, that they come up with for each member of the PKU family, to be from the defined haplotypes chart. Only if the evidence was overwhelming would an undefined haplotype be a part of the final two haplotypes for an individual, in which case that haplotype would be added on to the defined haplotypes chart. Therefore, to be on the safe side, an effort was made to save the defined haplotypes from eliminations where possible, and let the geneticists make the final decision on which two haplotypes should be retained.

In the test cases, a large set of heterozygosity was used, thus generating a large set of haplotypes for each member of the family. Whether the same occurs in real life would remain to be seen. To test the accuracy of the program, sometimes there especially the defined haplotypes. Since the undefined haplotypes were considered dispensible, that's where most of the eliminations would be expected to occur, especially in the case of the parents with smaller families.

It is a fact that most people in the Caucasian population, about 80% of them, possess haplotypes 1-4. Interesting results were obtained when combinations of these four haplotypes were taken to be the two haplotypes for an individual, see Table 3. For each of these combinations, extra defined haplotypes were also generated. The combination of haplotypes that generated the most haplotypes on the side were 1/4 and 2/3. For each of these combinations the resulting haplotypes that matched the input data were haplotypes 1, 2, 3, 4, 8, 10, 16, 17, 19, 24, 27, 28, 31, 41, and 43. Thirteen extra haplotypes were generated for each of these combinations.

Before questioning the program "PKU's" capabilities, one needs to be reminded that the purpose of the program was to help out the geneticists in the PKU analysis of a family by generating all of the haplotypes, defined and undefined, that satisfied the input data values for the eight restriction sites at the PAH locus. Some of these haplotypes may get eliminated if the collective family data did not support their existence. But the final saying will go to the geneticists. From the final haplotype listings provided by the program "PKU" for each member was large variation and difference in the input values for each member of the family. The thing being tested was whether or not the program eliminated those haplotypes from the children's lists that did not belong in the parents' lists, and vice versa. Because of the variability used in the children's inputs, some children retained most or all of their haplotypes after the eliminations, others however lost most or, in extreme cases, all of their haplotypes from their original haplotype lists after the elimination process. In real life and in theory, however, one may expect most of the children's haplotypes to be retained. This is because since the children's chromosomes are passed on by the parents, the input values should carry less heterozygosity at the 8 restriction sites for a child than for the parents' combined heterozygosities at the eight restriction sites. Errors in the chemical tests may have an effect on the number of haplotypes being eliminated for each member of the family, as described earlier, but these errors are a rarity and cannot be considered a common event.

If the input values for a child were a subset of the input values for the two parents combined, then this translates to the generated haplotypes for the child also being a subset of the haplotypes generated for the two parents. In real life, one would expect such to occur. Because of this one would expect all or most of the haplotypes for the children to be retained. The parents would also probably retain most of their haplotypes, of the PKU family, the geneticists will choose two haplotypes for each family member from their respective haplotype lists. The geneticists may have to use other means, such as their experience and intuition in dealing with these cases, to come up with the final two haplotypes for each member of the PKU family. However, that is not the concern of the program "PKU", and is beyond its capabilities for reasons mentioned earlier in this chapter.

It would be a safe assumption to make that the shorter the haplotype list is for an individual, the easier it would be to narrow it down to two haplotypes. It would be a lot easier to come up with the actual combination of haplotypes 1 and 2 from the list 1, 2, 24, and 27 then it would be to come up with the haplotype combination of 1 and 4 from the generated list 1, 2, 3, 4, 8, 10, 16, 17, 19, 24, 27, 28, 31, 41, and 43. The data from the fellow family members would be used to assist in coming up with the final two haplotypes for an individual. Also, the larger a family the more insight would be provided by the collective family data. This should make the job easier in making the final decision. For a family with relatively shorter lists of haplotypes, the geneticists can show more confidence in the diagnostic statement of the program declaring a sibling to be a carrier; especially where the diagnosis is based on the results of the generated defined haplotypes rather than the undefined haplotypes. If the sibling was diagnosed to be not a carrier of the PKU disease by the program -- no defective chromosomes -- then the geneticists would not even need to narrow down the haplotype list for the sibling to two first before agreeing with the diagnostic statement. It would save the geneticists some time if all they wanted to determine was whether or not the given sibling was a carrier of the PKU disease.

Reliability is one requirement that every program needs to fulfill in order to be of much use to its users. By this, it is meant that a program should be correct in its output for all inputs, and at all times. This is especially the case for a program the magnitude of the program "PKU". Once the program is made available to the geneticists, they would definitely want to use the program where a family analysis for a PKU family is required. Although the same thing can be done manually, one would not expect the geneticists to do that in the presence of the program "PKU". It is a rather tedious, time-consuming, and error-prone method to generate the possible RFLP haplotypes by hand. This would need to be done for each member of the PKU family. Once the RFLP haplotypes for each member of the family have been generated, the rest of the diagnosis is not very difficult especially where the defined haplotypes are concerned.

The undefined haplotypes would be a little more difficult to deal with since there is no known official numbering system that would assign a number to a generated undefined haplotype. The fact so many undefined haplotypes are generated for any attempts made to generate any number of defined haplotypes, it would be extremely difficult to deal with the undefined haplotypes for a family analysis. It is a lot easier to work with a number, or a symbol, or a combination of the two, respresenting an RFLP haplotype than to work with the haplotype pattern itself. The reason being, a haplotype is composed of eight places of "+" and "-" symbols (and "=" in the case of the HindIII enzyme) which can occur in any sequence. It would be a safe assumption to make that the undefined haplotypes may be ignored for this reason, especially if the defined haplotypes solve the problem by themselves.

The program "PKU" deals with both defined and undefined haplotypes and lets the geneticists decide for themselves how important the undefined haplotypes are to them in comparison with the defined haplotypes. The program "PKU" employs the same numbering system for the defined haplotypes as the scientific community in order to maintain consistency. A different system could have been employed, however, which would have used the patterns generated by the values for the eight restriction sites at the PAH locus as the guidline for the numerical designations of the haplotypes. For the undefined haplotypes such a system for numerical designations was employed since no other system has been used by anyone else to deal with them. Even though the program did not need any numbering system for the undefined haplotypes in order to work with them, a numbering system was employed nonetheless so the users would find it easier to deal with them. Another reason was for efficiency's sake whereby the numbers, that require very little memory space, were used for comparison purposes instead of the entire string of symbols that are the true identity of an RFLP haplotype but that also require a lot more memory space.

The fact the program "PKU" deals with both defined and undefined haplotypes should give the geneticists even more incentive to let it do all the tedious and time-consuming work for them. Because of the promise this program holds in its usefulness to the geneticists and its potential widespread use among geneticists concerned with the family analysis of the PKU disease, every effort was made, especially in the early stages of the program's development, to make sure that the program gives accurate results. A systematic approach was taken in establishing the accuracy of the program. For each set of inputs for an individual, it was calculated how many haplotypes should be generated. The formula used was the total number of haplotypes equals 2^n, where "n" represents the number of heterozygosities used in the input values for the eight restriction sites. A blank value was taken as a heterozygosity as well. A blank value for HindIII though changed the formula to $3 \times 2^{n-1}$ since in this case the number of haplotypes generated would triple instead of double. The generated haplotypes, both defined and undefined haplotypes, should add up to this number.

This is done for a quick check only and does not necessarily validate the program's accuracy.

The second step was to check if the string values of the generated haplotypes matched the input values. The generated defined haplotypes were then compared with the defined haplotypes chart, Table 1. A check was then made for any errors, i.e., the numbers and the actual string values for the generated haplotypes should match those in the chart.

The last step for checking the accuracy of the program involved waiting until the program had completed the process of eliminations of the unwanted haplotypes after the inputs were made for the entire PKU family. A check was made on whether all of the "unwanted" haplotypes were eliminated from each member of the family, and if all of the legitimate haplotypes still remained in the final listings. When satisfied with that outcome then a reference was made to the chart again, or with the original listing (before the elimination process), for each individual to check if the haplotype numbers and the string values still corresponded with each other.

These checks were made repeatedly in the development of the program to make sure the program was still accurate and, thus, reliable. Other checks made throughout were whether the function "make_changes", when utilized, made the required changes after a wrong entry had been made for a member of the PKU family. After the user has made inputs for each of the eight restriction sites for an individual, these values are displayed back to the user in a chart format, and then the user is asked if he/she would like to make any changes. If the answer is a "yes" then the function "make_changes" is employed and the desired changes are made. After all the changes have been made by the user, the input values are displayed back to the user again, and again the user is prompted if any changes are required. Until the user gives a "no" to this prompt, this cycle will repeat itself. This is important because any errors made in the input will affect not only the individual's haplotypes but may very well affect the other members of the family as well during the elimination process. After repeated checking, it can be concluded that the program "PKU" gives accurate and reliable results for any set of input values.

Besides accuracy, another way of judging a program is efficiency. Although for practical purposes, it does not matter how a program works as long as it works accurately, an effort must be made to make the program efficient in terms of its use of memory and time. The idea is to use as little of both as possible and still have a functional program. Sometimes one has to optimize between the two factors.

In this program, an effort was made to not use more memory than was needed to make the program run accurately. For this purpose pointers, i.e., dynamic memory allocations, were used when it came to storing the haplotype numbers for each member of the family because theoretically there is no limit to how big a family can be. For all practical purposes, static arrays for 10 or 20 people could have been used to get the same results, since most Caucasian families consist of less than 10, and definitely less than 20 people. Using static arrays takes less time than the dynamic memory allocations. But arrays take up unnecessary space in the memory. It was for the conservation of memory space that pointers were utilized even though they take a little more time than the static arrays, and are definitely more complicated to work with.

Another means used to save memory space was to use as few of variables as possible. For this purpose, same variables were re-used for other purposes wherever possible instead of generating new variables. By saving memory space, it was made sure that the program "PKU" would not require a large amount of memory to be functional, and therefore, can be run on personal computers with rather small memories. Although a little time had to be sacrificed in order to reach this goal, it was deemed worth it.

There may be a desire among the geneticists to have this project extended in the future, provided the indirect polymorphism linkage analysis method has not been replaced by some other PKU analysis method. The design of this program would allow for an easy extension. Most likely the geneticists would want the project to be extended so that it would narrow the final haplotype listings to two haplotypes for each member of the family. It is easy to think of cases where it would be impossible to narrow down the listings to two haplotypes with certainty for each member of the PKU family, especially if the family was small and each member showed plenty of heterozygosity for the eight restriction sites at the PAH locus. For other cases though, it would be quite possible to come up with two haplotypes for each member of the PKU family under analysis. However, it would require a programmer with a superior knowledge of the linkage analysis who has had some experience in RFLP testing for genetic analysis of the PKU families.

The extension of the program would include another function, or a set of functions, that would take the haplotypes remaining from the current haplotype elimination process, and then reduce the number all the way down to two haplotypes for each member of the family. Once a way has been found to do that, it would be a simple process to determine whether or not a sibling is a carrier of, or is affected by the PKU disease in a prenatal diagnosis. The two haplotypes for the sibling would be compared with those for the patient. If both haplotypes match, i.e., both haplotypes are the same for the patient and the sibling, then the sibling is affected by the PKU disease. If only one haplotype matches then the sibling is a carrier of the PKU disease. Otherwise, the sibling is declared "not a carrier".

Documentation was added to the program code so that the reader, or someone assigned with the task of extending the program, would have no difficulty following it. An effort was made to write the code of the program in a readable format. Also, suitable names were assigned to each function so one could guess the purpose of a function, i.e., what it does, from its name alone. This not only helped during the implementation of the program, but would also help someone else trying to extend it or just trying to follow it.

Although the program "PKU" could have been implemented in any higher level programming language that allowed dynamic memory allocations, the programming language Turbo C was employed for this purpose. Currently, C is in widespread use, and its demand is still on the rise. The project provided an opportunity to get acquainted with the language. As well, the choice of this programming language would create little hindrance for someone else wanting to extend the program. There might have been a problem if a lesser known programming language were used for implementing the program, as it may have forced the programmer to learn a new programming language first before proceeding with programming.

While major extensions of the program "PKU" in the future were kept in mind during the design of the program, other smaller foreseeable additions were also taken care of. To keep the program useful and not let it become outdated, it would be necessary to add the newly defined haplotypes to the defined haplotypes chart in the program. The program was designed so that it would easily incorporate the newly defined haplotypes. The scheme for doing this has been described earlier in detail -- see Chapter 2. Those new defined haplotypes of the future would actually come from the undefined haplotypes list of the present. The program was designed so that every time those new haplotypes are generated, the older numbers from the "Other Haplotypes" chart that used to represent them would also appear right beside the new numbers assigned to them. The idea is to make it easier for the geneticists to cross reference the newly defined haplotypes with the older undefined haplotypes. This way it would be easy to check if the newly defined haplotype(s) had previously appeared in other PKU families under the "Other Haplotypes" category.

An effort was made in the design of the program to make it user-friendly. Only limited knowledge in the field of linkage analysis using RFLP testing is required of the user to run this program. Even though the users should know what the symbols "+", "-", and "=" stand for in terms of the length of fragments obtained after digestion with a given enzyme, a chart is provided nonetheless which lists such a correspondence. That chart, Table 2, is displayed at the beginning each time input values are required for a member of the PKU family. Two input values are required for each of the eight restriction sites. For the convenience of the users, and to avoid typing mistakes, each entry is taken on a separate line. In case the user makes a mistake in an input, the change option, which is provided at the end of each set of inputs for an individual, can be used to correct those errors.

Also, each time an input is required from the user for either the input values for the eight restriction sites, or to respond to an option, the function "get_string" is employed. This function specializes in getting the input string value from the user and then returning the string value to the calling procedure after modifying it such that the extra and the unnecessary spaces have been removed from it. It is not an uncommon practice for people to press the space bar by accident when typing. The employment of the function "get_string" is to avoid delays for the users in case of minor, insignificant errors. The user would not be asked to give the input value again in case of these errors.

In summary, DNA polymorphisms at the PAH locus are used to obtain haplotypes through restriction enzyme analysis. These haplotypes are used to determine disease or carrier status in families with PKU. The program "PKU" was designed to assist the geneticists concerned with this task. It does this by generating all of the haplotypes that fit into the description provided by the chemical analysis. It then does a collective family analysis to eliminate those haplotypes from each family member's haplotype lists that are not supported by other members of the PKU family. The idea is to narrow down the lists of haplotypes generated for each member of the PKU family so the geneticists will have an easier time coming up with the final 2 haplotypes for each person. The program "PKU" also determines a sibling's carrier status. If a sibling was diagnosed to be a carrier then it would not be as alarming for the sibling as it would be for his, or her, children if the mate was also a carrier of the PKU disease. There would be no cause for concern, however, if the sibling was diagnosed as "not a carrier" of the PKU disease, in which case both of his/her chromosomes are normal.

CHAPTER 6

Program Specification

#define NUM_DEFINED 46; the constant NUM_DEFINED refers to the total number of defined haplotypes to date.

#define NUM_UNDEFINED 384; the contstant NUM_UNDEFINED refers to the total number of haplotypes that can theoretically exist.

```
     struct  chrom {
               char  enz[10];
               char  status[3];
               };
```

the structure "chrom" stores the name of the enzymes in the field "enz", while the "status" is used to store the input values (+,-, or =) that were obtained from the chemical tests and entered in by the user.

```
     struct haplos {
               int   chrom_num;
               char  chrom_cuts[10];
               int   presence;
               };
```

the structure "haplos" stores the numbers of the haplotypes in the field "chrom_num", the string values of the haplotypes in the field "chrom_cuts", and the "presence" is used as a flag to determine if the given haplotype exists for the individual under analysis.

```
     struct stack {
               int   haplo_num;
      struct stack   *next;
               };
```

the list "stack" is used to store the numbers of the haplotypes that are obtained for the members of the pku family.

```
     struct chrom  chromos;
```

"chromos" is used to store the restriction enzyme information.

```
     struct haplos  haplotypes;
```

"haplotypes" is used to store the information for the defined haplotypes.

```
     struct haplos  other_haplos;
```
"other_haplos" is used to store the information for all 384 haplotypes.

```
     struct stack  *defined_path;
```
"defined_path" contains a pointer to the beginnning of the list containing the numbers of the defined haplotypes for the family.

```
     struct stack  *undefined_path;
```
"undefined_path" contains a pointer to the beginning of the list containing the numbers of the undefined haplotypes for the family.

```
     struct stack  *defined_list;
```
"defined_list" is an extension of the "defined_path" and is used to store the numbers of the defined haplotypes for the family members.

```
     struct stack  *undefined_list;
```
"undefined_list" is an extension of the "undefined_path" and is used to store the numbers of the undefined haplotypes for the family members.

print_chart ()

print_chart displays a table showing names of the restriction enzymes, the symbols used for the data inputs, and the sizes of the fragments (in kbp) that these symbols represent for the corresponding restriction enzymes digestion products.

```
name_assignments (chromos)
struct chrom   *chromos;
```

name_assignments stores the names of the restriction enzymes used for the analysis under the "chromos" structure.

```
haplo_assignments (haplotypes)
struct haplos  *haplotypes;
```

haplo_assignments stores the defined haplotypes (their configurations and their corresponding numbers) under the "haplotypes" structure. It also assigns the haplotypes.presence, which is to be used as a flag, to equal 0 for all of the haplotypes. The 0 is to be replaced by a 1 later on by another function for the corresponding haplotypes if they were determined to be present in the individual being examined.

```
other_haplos_assignments (other_haplos)
struct haplos  *other_haplos;
```

other_haplos_assignments generates all 384 possible haplotypes that can theoretically exist. They are stored under the "other_haplos" structure. This function assigns each of the haplotypes (defined and undefined) a number from 1 to 384. The "other_haplos.presence" for each of these haplos is set to equal 0 which will be changed later on to 1 by another function if the corresponding haplotypes are determined to be present in the individual being examined.

```
get_string (s)
char  s[80];
```

get_string gets the input string from the user and then removes all of the blanks (spaces) from it. It then returns the resulting string value to the calling function.

```
get_values (chromos, i)
struct chrom   *chromos;
int   i;
```

get_values prompts the user to give the input values for the enzymes. The enzymes are represented by "chromos[i]". The acceptable input values are "+", "-", or a blank entry. For the enzyme HindIII, represented by "chromos[6]", an additional symbol "=" is also accepted. In case of errors in the input, an error message is displayed on the terminal and the user is prompted to give the input values again.

```
display_data (chromos)
struct chorm  *chromos;
```

display_data displays back the data entered in by the user in a tabular form.

```
make_changes (chromos)
struct chrom  *chromos;
```

make_changes displays the inputs made by the user for the restriction enzyme cuts one by one, and after each display it asks the user if he/she would like to make any changes to the input. If the answer is a "yes" then it calls the "get_values" function to replace the previous cut with the new cut.

```
get_nums_of_chroms (haplotypes, other_haplos)
struct haplos  *haplotypes, *other_haplos;
```

get_nums_of_chroms checks for the presence of the defined chromosomes in "haplotypes" for the individual being examined. If any of the defined haplotypes are present (i.e. haplotypes.presence = 1) then the corresponding numbers of these haplotypes are stored in the structure "defined_list". If any of the undefined haplotypes are present (i.e. other_haplos.presence = 1) then the corresponding numbers of these haplotypes are stored in the structure "undefined_list".

```
eliminate_children_haplos ()
```

eliminate_children_haplos eliminates those haplotypes for the patient and the siblings from the structures "defined_list" and "undefined_list" that do not belong to the parents' haplotypes lists for their defined and undefined haplotypes. In other words, only those haplotypes are retained for the patient and the siblings that correspond with the parents' haplotypes.

```
eliminate_parents_haplos (family)
int  family;
```

eliminate_parents_haplos eliminates those haplotypes for the parents from the structures "defined_list" and "undefined_list" that do not belong to any of their childrens' haplotype lists. At least 2 children must be included in the family analysis, i.e. family >= 4, in order to eliminate any of the haplotypes from the parents' haplotypes lists. If the total number of defined haplotypes for a parent would become less than two after the eliminations then the elimination procedure is to be withheld for that parent.

```
diagnosis (i)
int  i;
```

diagnosis determines if the sibling is a carrier of the pku disease or not. It first compares the haplotypes obtained for the sibling with those obtained for the patient. If any of the haplotypes that are present in the sibling (defined or undefined haplotypes) are also present in the patient then the sibling is a carrier of the pku disease. Other wise, the sibling is not a carrier. This function then displays the diagnostic statement onto the terminal screen. If only the undefined haplotypes matched for the sibling and the patient then an appropriate message lets the user know about it.

```
compare (haplo, haplotypes, other_haplos)
char   haplo[8];
struct haplos   *haplotypes, *other_haplos;
```

compare is called by the function "determine_haplotypes" to check if the haplotype generated from the input by the user belongs to the defined haplotype list ("haplotypes") or if it belongs to one of the undefined haplotypes ("other_haplos"). The presence of that haplotype is then marked by assigning haplotypes.presence = 1, or other_haplos.presence = 1, depending on which haplotype it corresponds to.

```
determine_haplotypes (chromos, haplotypes, other_haplos)
struct chrom  *chromos;
struct haplos  *haplotypes, *other_haplos;
```

determine_haplotypes takes the values given by the user and stored in the structure "chromos", and generates the possible haplotypes from them. If a null value was given for any of the restriction enzyme cuts, this function replaces that with a '+/-' for that restriction enzyme; and if that restriction enzyme was HindIII, then a '+/-/=' is assigned for that cut. It then calls the function "compare" to assign these haplotypes to either the defined haplotypes list ("haplotypes") by assigning haplotypes.presence = 1, or the undefined haplotypes list ("other_haplos") by assigning other_haplos.presence = 1.

next_family_member ()

next_family_member asks the user if he/she would like to continue the family analysis with another member of the family. It returns '1' to the main program if the answer is yes; it returns a '0' to the main program if the answer is no. A blank entry is to be taken as a 'yes'.

print_more ()

print_more asks the user if he/she would like to continue with the display of the haplotypes for the individual. It returns '1' to the calling function if the answer is yes; it returns a '0' to the m;ain program if the answer is no. A blank entry is to be taken as a 'yes'.

```
print_haplotypes (haplotypes, other_haplos, chromos)
struct haplos  *haplotypes, *other_haplos;
struct chrom   *chromos;
```

print_haplotypes displays all the haplotypes generated from the input by the user. The haploytpes are categorized according to their presence in either the defined haplotypes list ("haplotypes") or the undefined haplotypes list ("other_haplos"). After every 20 lines, the user is asked whether he/she would like to continue with display -- function "print_more" is employed for this purpose. Also, in case of new haplotypes being added to the defined list, the display will also contain, in brakets, the number from the undefined haplotypes list that was used to categorize that haplotype previously.

```
print_family_data (haplotypes,other_haplos,chromos,family)
struct haplos  *haplotypes, *other_haplos;
struct chrom   *chromos;
int   family;
```

print_family_data displays the entire famiy's haplotypes after the family analysis had been done. It calls the function "print_haplotypes" for each family member in order to display their respective haplotypes.

```
patient_analysis_only (chromos, haplotypes, other_haplos)
struct chrom  *chromos;
struct haplos  *haplotypes, *other_haplos;
```

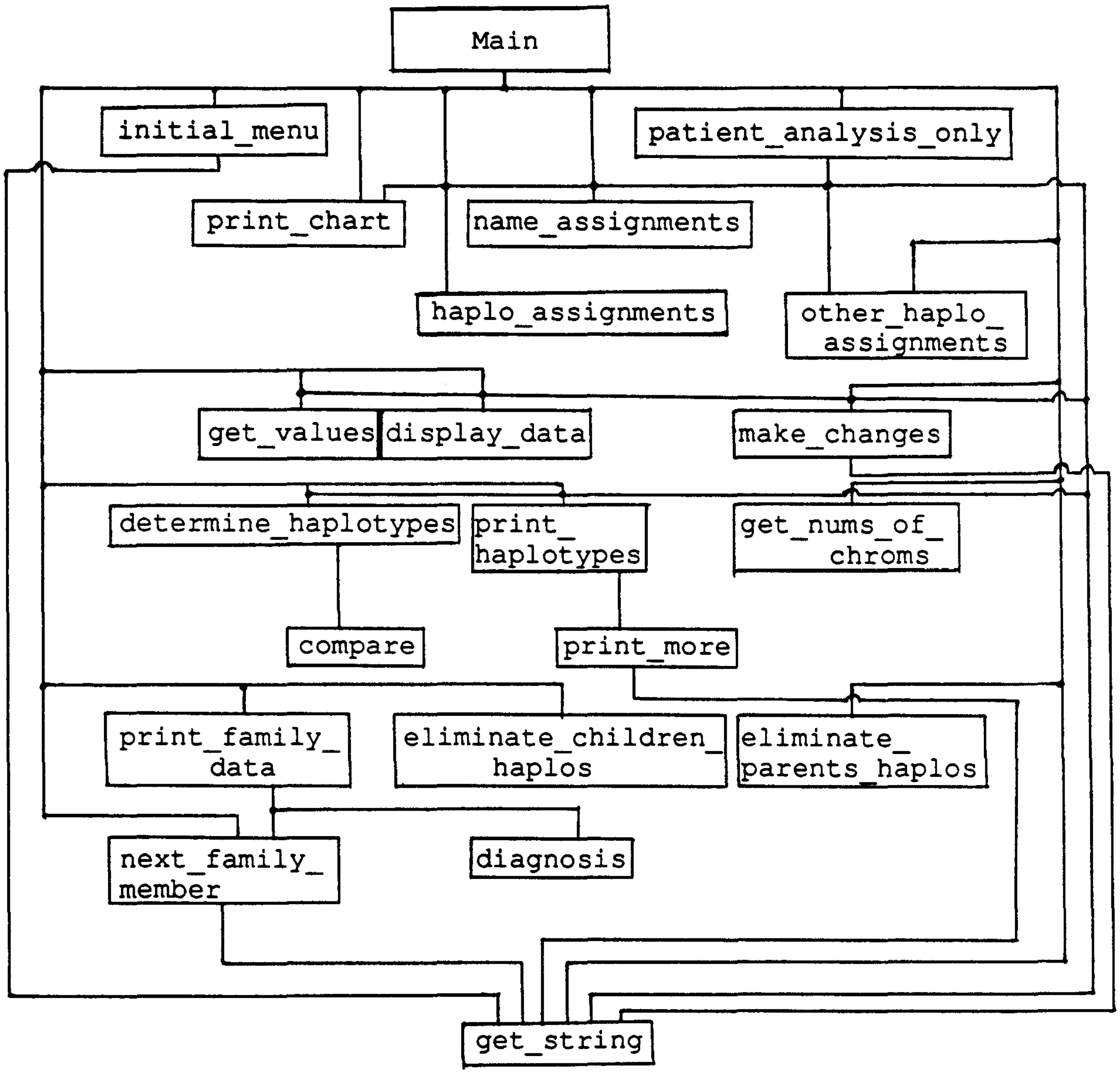
patient_analysis_only does the job of the main program if the user had selected the option of 'PATIENT ANALYSIS ONLY' from the main menu. It does the entire analysis for the patient in determining his/her haplotypes by calling up the appropriate functions.

```
initial_menu ()
```

initial_menu displays the initial menu to the user and prompts him/her to select from it. The options it offers are:

(1) a full family analysis, (2) patient analysis only, and (3) quit. It then returns the corresponding number to the main program.

Figure 2: Function Dependency Graph



BIBLIOGRAPHY

Chakraborty, Ranajit, et al. (1987). "Polymorphic DNA haplotypes at the human phenylalanine hydroxylase locus and their relationship with phenylketonuria". American Journal of Human Genetics, 76:40-46.

Chang, Patricia, et al. (1990). "Molecular Diagnosis of Phenylketonuria". J. Int. Fed. Clin. Chem. in press.

Daiger, Stephen P., et al. (1989). "Polymorphic DNA Haplotypes at the Phenylalanine Hydroxylase (PAH) Locus in European Families with Phenylketonuria (PKU)". American Journal of Human Genetics, 45:310-318.

Daiger, Stepehn P., et al. (1989). "Polymorphic DNA Haplotypes at the Phenylalanine Hydroxylase (PAH) Locus in Asian Families with Phenylketonuria (PKU)". American Journal of Human Genetics, 45:319-324.

DiLella AG, et al. (1986a). "Molecular structure and polymorphic map of the human phenylalanine hydroxylase gene". Biochemistry, 25:743-749.

DiLella AG, et al. (1984). "An amino acid substitution involved in phenylketonuria is in linkage disequilibrium with DNA haplotype 2". Nature, 327:333-336.

Fairley, Richard E. Software Engineering Concepts. McGraw-Hill, Inc., 1985, New York, USA.

Horspool, R. Nigel. C Programming in the Berkely Unix Environment, Prentice-Hall Canada Inc., 1986, Scarborough, Ontario.

Kidd KK (1987). "Phenylketonuria: population genetics of a disease". Nature, 327:282-283.

Lidsky, A.S., et al. (1985). "Extensive restriction site polymorphism at the human phenylalanine hydroxylase locus and application in prenatal diagnosis of phenylketonuria". American Journal of Human Genetics, 37:619-634.

Old, R.W., & Primrose, S.B. Principles of Gene Manipulation, Blackwell Scientific Publications, 1980, University Press, Cambridge, Great Britain.

Stryer, Lubert. Biochemistry, W.H. Freeman & Company, 1975, New York, U.S.A.

Sullivan, S.E., et al. (1989). "Haplotype Distribution of the Human Phenylalanine Hydroxylase Locus in Scotland and Switzerland". American Journal of Human Genetics, 44:652-659.

Woo, S.L.C. (1988). "Collation of RFLP haplotypes at the human phenylalanine hydroxylase (PAH) locus". American Journal of Human Genetics, 43:781-783.

Woo, S.L.C., et al. (1983). "Cloned human phenylalanine hydroxylase gene allows prenatal diagnosis and carrier detection of classical phenylketonuria". Nature, 306:151-155.

APPENDIX I

USERS' MANUAL

To get started, insert the floppy disk containing the executable file of the program "PKU" in a computer drive and type "pku". The following menu will be displayed on the computer terminal screen:

```
****************************************************
****************************************************
****                                           *****
****                                           *****
****          OPTIONS:                         *****
****                                           *****
****               1. FAMILY ANALYSIS          *****
****               2. PATIENT ANALYSIS ONLY    *****
****               3. QUIT                     *****
****                                           *****
****                                           *****
****************************************************
****************************************************

Select from above please:
```

Typing "3" will terminate the program. If one only wanted to determine haplotypes for one person then the second option would be chosen -- type "2". For a full family analysis, the first option would be chosen -- type "1".

If the first option was selected then the program "PKU" would ask for input data to PARENT #1, PARENT #2, the PATIENT, and the SIBLINGS in that order.

At the beginning of each, the following chart would be displayed.

```
          PvuIIa BglII PvuIIb EcoRI XmnI MspI HindIII EcoRV
Symbol:
 + .....    6.0   1.7    9.1   11.0  6.5 19.0   4.0    25.0
 - .....   19.0   3.6   11.5   17.0  9.4 23.0   4.2    30.0
 = .....                                        4.4
```

The program then asks for the input values for the restriction enzymes. Only "+" or "-" is accepted by the program for these input values -- for HindIII "=" is also accepted.

For example, if both positive and negative values were obtained for PvuIIa then a "+" should be entered for the prompt:

```
(i)  PvuIIa (+/-)     :
```

and a "-" should be entered for the prompt:

```
(ii) PvuIIa (+/-)     :
```

It would make no difference if "-" was entered first, and "+" entered second. In case of no result being obtained from the chemical analysis for a restriction site, a blank entry should be made by hitting the "return" key without typing anything else.

The data entered is then displayed back to the user and the user is asked if any changes are desired. Either "y" or "Y" should be entered for changes, or "n" or "N" if the user is satisfied with the input.

The program would then display the generated defined haplotypes for the individual and then ask the user whether to continue with the display of haplotypes:

```
CONTINUE DISPLAY?(y/n) :
```

A blank entry would be taken to mean "yes" by default. If the entry was a "yes" then the generated undefined haplotypes would also be displayed.

The program then asks the user if more individuals are to be involved in the family analysis:

```
CONTINUE WITH NEXT FAMILY MEMBER?(y/n) :
```

A blank entry would be taken to mean "yes" by default. The above cycle would continue until the user enters "n" or "N" for "no" to the above prompt. In that case the program would do a collective family analysis to eliminate those haplotypes from the individuals' lists that are not supported by the other family members. The program then displays the resulting lists for the members of the PKU family in the same order as they were prompted for: PARENT #1, PARENT #2, the PATIENT, SIBLING #1, SIBLING #2, and so on in that order. After display of the defined haplotypes, the program asks:

```
CONTINUE DISPLAY?(y/n) :
```

If the answer was "yes" then the undefined haplotypes would also be displayed. Otherwise, the program would ask if the display should continue with the next family member:

```
CONTINUE WITH NEXT FAMILY MEMBER?(y/n) :
```

Again, a blank entry is taken to mean "yes" by default and the haplotypes for the next family member would be displayed. To terminate the execution of the program, an "n" or "N" should be typed, and the remaining members of the family will be skipped. Otherwise, the program will terminate itself once all of the family members have been accounted for.

APPENDIX II

Sample Executions

```
maccs[25] pku


     **********************************************************
     **********************************************************
     ****                                                 ****
     ****                                                 ****
     ****            OPTIONS:                             ****
     ****                                                 ****
     ****             1. FAMILY ANALYSIS                  ****
     ****             2. PATIENT ANALYSIS ONLY            ****
     ****             3. QUIT                             ****
     ****                                                 ****
     ****                                                 ****
     **********************************************************
     **********************************************************


     Select from above please:           1

ENTER DATA FOR PARENT #1 PLEASE :


                PvuIIa BglII PvuIIb EcoRI XmnI MspI HindIII EcoRV
 Symbol:
  + ..........     6.0   1.7    9.1    11.0  6.5  19.0    4.0  25.0
  - ..........    19.0   3.6   11.5    17.0  9.4  23.0    4.2  30.0
  = ..........                                            4.4

GIVE THE INPUT VALUES FOR THE RESTRICTION ENZYMES PLEASE



 (i)  PvuIIa (+/-)   :    -
```

```
(ii) PvuIIa (+/-)    :     -

(i)  BglII  (+/-)    :
(ii) BglII  (+/-)    :

(i)  PvuIIb (+/-)    :
(ii) PvuIIb (+/-)    :

(i)  EcoRI  (+/-)    :     -
(ii) EcoRI  (+/-)    :     -

(i)  XmnI  (+/-)     :
(ii) XmnI  (+/-)     :

(i)  MspI  (+/-)     :     -
(ii) MspI  (+/-)     :     -

(i)  HindIII  (+/-/=):     -
(ii) HindIII  (+/-/=):     -

(i)  EcoRV (+/-)     :     -
(ii) EcoRV (+/-)     :     -


DATA ENTERED:

 PvuIIa      BglII     PvuIIb      EcoRI      XmnI       MspI       HindIII  EcoRV
---------------------------------------------------------------------------------
|  -/-      |   /      |   /       |  -/-     |   /       |  -/-     |  -/-     |  -/-|


ANY CHANGES?(y/n) :    y

  PvuIIa    :   -/-
                     change?(y/n) :   n

  BglII     :    /
                     change?(y/n) :   y

 (i)  BglII (+/-)     :     +
 (ii) BglII (+/-)     :              -

  PvuIIb    :    /
                     change?(y/n) :   n

  EcoRI     :   -/-
                     change?(y/n) :   n

  XmnI      :    /
                     change?(y/n) :       N
  MspI      :   -/-
                     change?(y/n) :   n

  HindIII        :   -/-
                     change?(y/n) :   n

  EcoRV     :   -/-
                     change?(y/n) :   n

DATA ENTERED:

 PvuIIa      BglII     PvuIIb      EcoRI      XmnI       MspI       HindIII  EcoRV
----------------------------------------------------------------------------
|  -/-      |  +/-     |   /       |  -/-     |   /       |  -/-     |  -/-     |  -/-|

ANY CHANGES?(y/n) :   n


Defined Haplotypes:

          PvuIIa     BglII      PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV

 29  .....  -          +          -          -          -    -     -          -
 40  .....  -          +          -          -          +    -     -          -

                CONTINUE DISPLAY?(y/n) :

^[[;H^[[2J

Other Haplotypes:

          PvuIIa     BglII      PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV

T-226 .....  -          +          +          -          +    -     -          -
T-238 .....  -          +          +          -          -    -     -          -
T-322 .....  -          -          +          -          +    -     -          -
T-334 .....  -          -          +          -          -    -     -          -
T-370 .....  -          -          -          -          +    -     -          -
T-382 .....  -          -          -          -          -    -     -          -


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n) :
```

```
ENTER DATA FOR PARENT #2 PLEASE :



                PvuIIa BglII PvuIIb EcoRI XmnI MspI HindIII EcoRV
 Symbol:
  + ..........    6.0   1.7    9.1  11.0   6.5 19.0  4.0    25.0
  - ..........   19.0   3.6   11.5  17.0   9.4 23.0  4.2    30.0
  = ..........                                      4.4

GIVE THE INPUT VALUES FOR THE RESTRICTION ENZYMES PLEASE



 (i)  PvuIIa (+/-)    :    +
 (ii) PvuIIa (+/-)    :    -

 (i)  BglII (+/-)     :    +
 (ii) BglII (+/-)     :

 (i)  PvuIIb (+/-)    :
 (ii) PvuIIb (+/-)    :

 (i)  EcoRI (+/-)     :    -
 (ii) EcoRI (+/-)     :    -

 (i)  XmnI (+/-)      :    -
 (ii) XmnI (+/-)      :    -

 (i)  MspI (+/-)      :
 (ii) MspI (+/-)      :

 (i)  HindIII (+/-/=):     -
 (ii) HindIII (+/-/=):     -

 (i)  EcoRV (+/-)     :    -
 (ii) EcoRV (+/-)     :    -

DATA ENTERED:

 PvuIIa      BglII     PvuIIb      EcoRI      XmnI       MspI       HindIII  EcoRV
---------------------------------------------------------------------------
|  +/-     |  +/      |   /       |  -/-      |  -/-      |   /       |  -/-     |  -/-|
ANY CHANGES?(y/n) :    n




Defined Haplotypes:

           PvuIIa     BglII      PvuIIb     EcoRI      XmnI  MspI  HindIII    EcoRV

  1  .....  +          -          -          -          -     +     -          -
 26  .....  +          +          -          -          -     +     -          -
 29  .....  -          +          -          -          -     -     -          -
 32  .....  -          +          -          -          -     +     -          -
 36  .....  -          +          +          -          -     +     -          -
 37  .....  -          -          -          -          -     +     -          -

                CONTINUE DISPLAY?(y/n) :


Other Haplotypes:

           PvuIIa     BglII      PvuIIb     EcoRI      XmnI  MspI  HindIII    EcoRV

T-40   .....  +          +          +          -          -     +     -          -
T-46   .....  +          +          +          -          -     -     -          -
T-94   .....  +          +          -          -          -     -     -          -
T-136  .....  +          -          +          -          -     +     -          -
T-142  .....  +          -          +          -          -     -     -          -
T-190  .....  +          -          -          -          -     -     -          -
T-238  .....  -          +          +          -          -     -     -          -
T-328  .....  -          -          +          -          -     +     -          -
T-334  .....  -          -          +          -          -     -     -          -
T-382  .....  -          -          -          -          -     -     -          -


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n) :     Y


ENTER DATA FOR THE PATIENT PLEASE :


                PvuIIa BglII PvuIIb EcoRI XmnI MspI HindIII EcoRV
 Symbol:
```

```
 + ..........    6.0   1.7    9.1    11.0   6.5   19.0    4.0    25.0
 - ..........   19.0   3.6   11.5    17.0   9.4   23.0    4.2    30.0
 = ..........                                             4.4

GIVE THE INPUT VALUES FOR THE RESTRICTION ENZYMES PLEASE



 (i)   PvuIIa  (+/-)     :     -
 (ii)  PvuIIa  (+/-)     :     -

 (i)   BglII  (+/-)      :     +
 (ii)  BglII  (+/-)      :     -

 (i)   PvuIIb  (+/-)     :     +
 (ii)  PvuIIb  (+/-)     :        -

 (i)   EcoRI  (+/-)      :     -
 (ii)  EcoRI  (+/-)      :     -

 (i)   XmnI  (+/-)       :     -
 (ii)  XmnI  (+/-)       :     -

 (i)   MspI  (+/-)       :     +
 (ii)  MspI  (+/-)       :     -

 (i)   HindIII  (+/-/=)  :     -
 (ii)  HindIII  (+/-/=)  :     -

 (i)   EcoRV  (+/-)      :
 (ii)  EcoRV  (+/-)      :

DATA ENTERED:

 PvuIIa      BglII      PvuIIb      EcoRI      XmnI       MspI       HindIII  EcoRV
---------------------------------------------------------------------------
|  -/-      |  +/-      |  +/-      |  -/-      |  -/-      |  +/-      |  -/-      |    / |

ANY CHANGES?(y/n)  :    n


Defined Haplotypes:
```

```
          PvuIIa     BglII      PvuIIb     EcoRI      XmnI  MspI  HindIII   EcoRV

29  .....   -          +          -          -          -     -     -         -
32  .....   -          +          -          -          -     +     -         -
33  .....   -          +          -          -          -     +     -         +
36  .....   -          +          +          -          -     +     -         -
37  .....   -          -          -          -          -     +     -         -

               CONTINUE DISPLAY?(y/n)  :



Other Haplotypes:

          PvuIIa     BglII      PvuIIb     EcoRI      XmnI  MspI  HindIII   EcoRV

T-231 .....   -        +          +          -          -     +     -         +
T-237 .....   -        +          +          -          -     -     -         +
T-238 .....   -        +          +          -          -     -     -         -
T-285 .....   -        +          -          -          -     -     -         +
T-327 .....   -        -          +          -          -     +     -         +
T-328 .....   -        -          +          -          -     +     -         -
T-333 .....   -        -          +          -          -     -     -         +
T-334 .....   -        -          +          -          -     -     -         -
T-375 .....   -        -          -          -          -     +     -         +
T-381 .....   -        -          -          -          -     -     -         +
T-382 .....   -        -          -          -          -     -     -         -


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n)  :


ENTER DATA FOR A SIBLING PLEASE :


                PvuIIa BglII PvuIIb EcoRI XmnI MspI HindIII EcoRV
 Symbol:
  + ..........    6.0   1.7    9.1   11.0  6.5  19.0  4.0    25.0
  - ..........   19.0   3.6   11.5   17.0  9.4  23.0  4.2    30.0
  = ..........                                         4.4

GIVE THE INPUT VALUES FOR THE RESTRICTION ENZYMES PLEASE
```

```
 (i)   PvuIIa  (+/-)     :     -
 (ii)  PvuIIa  (+/-)     :     -

 (i)   BglII  (+/-)      :     +
 (ii)  BglII  (+/-)      :     -

 (i)   PvuIIb  (+/-)     :     -
 (ii)  PvuIIb  (+/-)     :     -

 (i)   EcoRI  (+/-)      :
 (ii)  EcoRI  (+/-)      :

 (i)   XmnI  (+/-)       :     -
 (ii)  XmnI  (+/-)       :     -

 (i)   MspI  (+/-)       :
 (ii)  MspI  (+/-)       :

 (i)   HindIII  (+/-/=) :     -
 (ii)  HindIII  (+/-/=) :     -

 (i)   EcoRV  (+/-)      :     -
 (ii)  EcoRV  (+/-)      :     -


DATA ENTERED:

 PvuIIa      BglII     PvuIIb      EcoRI      XmnI      MspI      HindIII  EcoRV
-----------------------------------------------------------------------------
|  -/-      |  +/-    |  -/-     |   /       |  -/-     |   /      |  -/-     |  -/-|


ANY CHANGES?(y/n)  :    n



Defined Haplotypes:

          PvuIIa     BglII     PvuIIb     EcoRI     XmnI  MspI  HindIII    EcoRV

 14  .....  -          -          -          +        -     +     -          -
 29  .....  -          +          -          -        -     -     -          -
 32  .....  -          +          -          -        -     +     -          -
 34  .....  -          +          -          +        -     +     -          -
 37  .....  -          -          -          -        -     +     -          -
 42  .....  -          +          -          +        -     -     -          -

                CONTINUE DISPLAY?(y/n)  :   y
```

```
Other Haplotypes:

          PvuIIa     BglII      PvuIIb     EcoRI      XmnI  MspI  HindIII    EcoRV

T-358 .....  -          -          -          +          -     -     -          -
T-382 .....  -          -          -          -          -     -     -          -


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n)  :



ENTER DATA FOR A SIBLING PLEASE :


                PvuIIa BglII PvuIIb EcoRI XmnI MspI HindIII EcoRV
 Symbol:
  + ..........    6.0   1.7    9.1   11.0  6.5  19.0  4.0    25.0
  - ..........   19.0   3.6   11.5   17.0  9.4  23.0  4.2    30.0
  = ..........                                         4.4

GIVE THE INPUT VALUES FOR THE RESTRICTION ENZYMES PLEASE



 (i)  PvuIIa (+/-)    :    +
 (ii) PvuIIa (+/-)    :    -

 (i)  BglII  (+/-)    :    -
 (ii) BglII  (+/-)    :    -

 (i)  PvuIIb (+/-)    :
 (ii) PvuIIb (+/-)    :

 (i)  EcoRI  (+/-)    :
 (ii) EcoRI  (+/-)    :

 (i)  XmnI  (+/-)     :
 (ii) XmnI  (+/-)     :

 (i)  MspI  (+/-)     :    +
 (ii) MspI  (+/-)       :     -

 (i)   HindIII  (+/-/=):     -
 (ii) HindIII  (+/-/=):     -

 (i)   EcoRV  (+/-)     :     -
 (ii) EcoRV  (+/-)     :     -


DATA ENTERED:

 PvuIIa      BglII     PvuIIb      EcoRI      XmnI      MspI      HindIII   EcoRV
-----------------------------------------------------------------------------
|  +/-     |  -/-     |   /      |   /      |   /      |  +/-     |  -/-     |  -/-|


ANY CHANGES?(y/n)  :    n



Defined Haplotypes:

           PvuIIa     BglII      PvuIIb     EcoRI       XmnI MspI  HindIII    EcoRV

  1  .....   +          -          -          -          -    +      -          -
  3  .....   +          -          -          +          +    -      -          -
 10  .....   +          -          -          +          -    +      -          -
 14  .....   -          -          -          +          -    +      -          -
 17  .....   +          -          -          +          -    -      -          -
 20  .....   -          -          +          +          -    +      -          -
 37  .....   -          -          -          -          -    +      -          -
 39  .....   +          -          +          +          -    +      -          -
 43  .....   +          -          -          -          +    +      -          -

                CONTINUE DISPLAY?(y/n)  :



Other Haplotypes:

           PvuIIa     BglII      PvuIIb     EcoRI       XmnI MspI  HindIII    EcoRV

T-100 .....  +          -          +          +          +    +      -          -
T-106 .....  +          -          +          +          +    -      -          -
T-118 .....  +          -          +          +          -    -      -          -
T-124 .....  +          -          +          -          +    +      -          -
T-130 .....  +          -          +          -          +    -      -          -
T-136 .....  +          -          +          -          -    +      -          -
T-142 .....  +          -          +          -          -    -      -          -
T-148 .....  +          -          -          +          +    +      -          -
```

```
T-178 .....  +         -         -         -          +    -    -         -
T-190 .....  +         -         -         -          -    -    -         -
T-292 .....  -         -         +         +          +    +    -         -
T-298 .....  -         -         +         +          +    -    -         -
T-310 .....  -         -         +         +          -    -    -         -
T-316 .....  -         -         +         -          +    +    -         -
T-322 .....  -         -         +         -          +    -    -         -
T-328 .....  -         -         +         -          -    +    -         -
T-334 .....  -         -         +         -          -    -    -         -
T-340 .....  -         -         -         +          +    +    -         -
T-346 .....  -         -         -         +          +    -    -         -
T-358 .....  -         -         -         +          -    -    -         -

                CONTINUE DISPLAY?(y/n)  :


T-364 .....  -         -         -         -          +    +    -         -
T-370 .....  -         -         -         -          +    -    -         -
T-382 .....  -         -         -         -          -    -    -         -


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n)  :



ENTER DATA FOR A SIBLING PLEASE :


                PvuIIa BglII PvuIIb EcoRI XmnI MspI HindIII EcoRV
 Symbol:
  + ..........    6.0   1.7    9.1   11.0  6.5  19.0  4.0    25.0
  - ..........   19.0   3.6   11.5   17.0  9.4  23.0  4.2    30.0
  = ..........                                         4.4

GIVE THE INPUT VALUES FOR THE RESTRICTION ENZYMES PLEASE


 (i)  PvuIIa (+/-)    :    -
 (ii) PvuIIa (+/-)    :    -
```

```
(i)  BglII (+/-)     :
(ii) BglII (+/-)     :

(i)  PvuIIb (+/-)    :    -
(ii) PvuIIb (+/-)    :    -

(i)  EcoRI (+/-)     :
(ii) EcoRI (+/-)     :

(i)  XmnI (+/-)      :    -
(ii) XmnI (+/-)      :    -

(i)  MspI (+/-)      :
(ii) MspI (+/-)      :    -

(i)  HindIII (+/-/=):    -
(ii) HindIII (+/-/=):    -

(i)  EcoRV (+/-)     :    -
(ii) EcoRV (+/-)     :    -


DATA ENTERED:

 PvuIIa     BglII    PvuIIb     EcoRI     XmnI      MspI      HindIII  EcoRV
---------------------------------------------------------------------------
|  -/-     |   /     |  -/-     |   /     |  -/-     |   /-     |  -/-     |  -/-|


ANY CHANGES?(y/n) :    y

  PvuIIa   :   -/-
                    change?(y/n) :   n

  BglII    :    /
                    change?(y/n) :   n

  PvuIIb   :   -/-
                    change?(y/n) :   n

  EcoRI    :    /
                    change?(y/n) :   y

 (i)  EcoRI (+/-)     :    -
 (ii) EcoRI (+/-)     :    +

  XmnI     :   -/-
                    change?(y/n) :   n
 MspI      :     /-
                           change?(y/n)  :    y

(i)  MspI  (+/-)       :     -
(ii) MspI  (+/-)       :     -

 HindIII        :   -/-
                           change?(y/n)  :    n

 EcoRV     :    -/-
                           change?(y/n)  :    n


DATA ENTERED:

 PvuIIa      BglII     PvuIIb      EcoRI      XmnI       MspI       HindIII  EcoRV
----------------------------------------------------------------------------
|  -/-      |    /     |  -/-     |  -/+     |  -/-     |  -/-     |  -/-     |  -/-|


ANY CHANGES?(y/n)  :    n^H ^Hn



Defined Haplotypes:

           PvuIIa     BglII      PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV
 29  .....   -          +          -          -          -    -     -          -
 42  .....   -          +          -          +          -    -     -          -

                CONTINUE DISPLAY?(y/n)  :   y




Other Haplotypes:

           PvuIIa     BglII      PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV

T-358 .....  -          -          -          +          -    -     -          -
T-382 .....  -          -          -          -          -    -     -          -



        CONTINUE WITH NEXT FAMILY MEMBER?(y/n)  :
```

```
ENTER DATA FOR A SIBLING PLEASE :



               PvuIIa BglII PvuIIb EcoRI XmnI MspI HindIII EcoRV
 Symbol:
  + ..........   6.0   1.7    9.1   11.0  6.5  19.0   4.0   25.0
  - ..........  19.0   3.6   11.5   17.0  9.4  23.0   4.2   30.0
  = ..........                                        4.4

GIVE THE INPUT VALUES FOR THE RESTRICTION ENZYMES PLEASE



 (i)  PvuIIa (+/-)    :    -
 (ii) PvuIIa (+/-)    :    -

 (i)  BglII (+/-)     :    -
 (ii) BglII (+/-)     :    -

 (i)  PvuIIb (+/-)    :
 (ii) PvuIIb (+/-)    :

 (i)  EcoRI (+/-)     :    -
 (ii) EcoRI (+/-)     :    -

 (i)  XmnI (+/-)      :
 (ii) XmnI (+/-)      :

 (i)  MspI (+/-)      :    -
 (ii) MspI (+/-)      :    -

 (i)  HindIII (+/-/=):     -
 (ii) HindIII (+/-/=):     -

 (i)  EcoRV (+/-)     :    +
 (ii) EcoRV (+/-)     :    -

DATA ENTERED:

 PvuIIa     BglII     PvuIIb     EcoRI     XmnI      MspI      HindIII  EcoRV
---------------------------------------------------------------------------
|  -/-      |  -/-     |   /      |  -/-     |   /      |  -/-     |  -/-    |  +/-|


ANY CHANGES?(y/n)  :
ANY CHANGES?(y/n)  :   n
```

```
Defined Haplotypes:

          PvuIIa     BglII  PvuIIb    EcoRI XmnI  MspI  HindIII    EcoRV

                     NONE EXIST!!!


                CONTINUE DISPLAY?(y/n)  :


Other Haplotypes:

          PvuIIa     BglII     PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV

T-321 .....  -         -         +          -          +    -      -          +
T-322 .....  -         -         +          -          +    -      -          -
T-333 .....  -         -         +          -          -    -      -          +
T-334 .....  -         -         +          -          -    -      -          -
T-369 .....  -         -         -          -          +    -      -          +
T-370 .....  -         -         -          -          +    -      -          -
T-381 .....  -         -         -          -          -    -      -          +
T-382 .....  -         -         -          -          -    -      -          -


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n)  :  n


POSSIBLE HAPLOTYPES FOR PARENT #1 AFTER FAMILY ANALYSIS :




Defined Haplotypes:

          PvuIIa     BglII     PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV

 29  .....  -          +         -          -          -    -      -          -
 40  .....  -          +         -          -          +    -      -          -

                CONTINUE DISPLAY?(y/n)  :
```

```
Other Haplotypes:

            PvuIIa     BglII      PvuIIb     EcoRI       XmnI MspI  HindIII    EcoRV

T-238 .....   -          +          +          -           -    -     -          -
T-322 .....   -          -          +          -           +    -     -          -
T-334 .....   -          -          +          -           -    -     -          -
T-370 .....   -          -          -          -           +    -     -          -
T-382 .....   -          -          -          -           -    -     -          -


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n)  :


POSSIBLE HAPLOTYPES FOR PARENT #2 AFTER FAMILY ANALYSIS :


Defined Haplotypes:

            PvuIIa     BglII      PvuIIb     EcoRI       XmnI MspI  HindIII    EcoRV

  1  .....  +          -          -          -           -    +     -          -
 29  .....  -          +          -          -           -    -     -          -
 32  .....  -          +          -          -           -    +     -          -
 36  .....  -          +          +          -           -    +     -          -
 37  .....  -          -          -          -           -    +     -          -

                CONTINUE DISPLAY?(y/n)  :


Other Haplotypes:

            PvuIIa     BglII      PvuIIb     EcoRI       XmnI MspI  HindIII    EcoRV

T-136 .....   +          -          +          -           -    +     -          -
T-142 .....   +          -          +          -           -    -     -          -
T-190 .....   +          -          -          -           -    -     -          -
T-238 .....   -          +          +          -           -    -     -          -
T-328 .....   -          -          +          -           -    +     -          -
T-334 .....   -          -          +          -           -    -     -          -
T-382 .....   -          -          -          -           -    -     -          -
```

```
        CONTINUE WITH NEXT FAMILY MEMBER?(y/n) :


POSSIBLE HAPLOTYPES FOR THE PATIENT AFTER FAMILY ANALYSIS :


Defined Haplotypes:

          PvuIIa     BglII      PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV

 29  .....  -          +          -          -          -    -     -          -
 32  .....  -          +          -          -          -    +     -          -
 36  .....  -          +          +          -          -    +     -          -
 37  .....  -          -          -          -          -    +     -          -

                CONTINUE DISPLAY?(y/n)  :


Other Haplotypes:

          PvuIIa     BglII      PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV

T-238 .....  -         +          +          -          -    -     -          -
T-328 .....  -         -          +          -          -    +     -          -
T-334 .....  -         -          +          -          -    -     -          -
T-382 .....  -         -          -          -          -    -     -          -


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n) :


POSSIBLE HAPLOTYPES FOR SIBLING #1 AFTER FAMILY ANALYSIS :


Defined Haplotypes:

          PvuIIa     BglII      PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV

 29  .....  -          +          -          -          -    -     -          -
 32  .....  -          +          -          -          -    +     -          -
 37  .....  -          -          -          -          -    +     -          -

                CONTINUE DISPLAY?(y/n)  :
```

```
Other Haplotypes:

            PvuIIa     BglII      PvuIIb     EcoRI       XmnI MspI HindIII    EcoRV

T-382 ..... -          -          -          -           -    -    -          -


DIAGNOSIS :  THE SIBLING IS A POTENTIAL CARRIER


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n)  :


POSSIBLE HAPLOTYPES FOR SIBLING #2 AFTER FAMILY ANALYSIS :


Defined Haplotypes:

            PvuIIa     BglII      PvuIIb     EcoRI       XmnI MspI HindIII    EcoRV

  1  .....  +          -          -          -           -    +    -          -
 37  .....  -          -          -          -           -    +    -          -

                CONTINUE DISPLAY?(y/n)  :



Other Haplotypes:

            PvuIIa     BglII      PvuIIb     EcoRI       XmnI MspI HindIII    EcoRV

T-136 .....  +         -          +          -           -    +    -          -
T-142 .....  +         -          +          -           -    -    -          -
T-190 .....  +         -          -          -           -    -    -          -
T-322 .....  -         -          +          -           +    -    -          -
T-328 .....  -         -          +          -           -    +    -          -
T-334 .....  -         -          +          -           -    -    -          -
T-370 .....  -         -          -          -           +    -    -          -
T-382 .....  -         -          -          -           -    -    -          -
```

```
DIAGNOSIS :  THE SIBLING IS A POTENTIAL CARRIER


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n)  :


POSSIBLE HAPLOTYPES FOR SIBLING #3 AFTER FAMILY ANALYSIS :


Defined Haplotypes:
           PvuIIa     BglII      PvuIIb     EcoRI       XmnI      MspI HindIII    Ecd
 29  .....  -          +          -          -          -    -    -          -
                CONTINUE DISPLAY?(y/n)  :


Other Haplotypes:
           PvuIIa     BglII      PvuIIb     EcoRI      XmnI MspI  HindIII    EcoRV
T-382 ..... -          -           -         -          -    -    -          -


DIAGNOSIS :  THE SIBLING IS A POTENTIAL CARRIER


        CONTINUE WITH NEXT FAMILY MEMBER?(y/n)  :


POSSIBLE HAPLOTYPES FOR SIBLING #4 AFTER FAMILY ANALYSIS :


Defined Haplotypes:
           PvuIIa     BglII      PvuIIb     EcoRI      XmnI  MspI HindIII    EcoRV

                      NONE EXIST!!!
```

```
          CONTINUE DISPLAY?(y/n) :


Other Haplotypes:

           PvuIIa     BglII      PvuIIb     EcoRI      XmnI MspI HindIII    EcoRV

T-322 ..... -          -          +          -          +    -    -          -
T-334 ..... -          -          +          -          -    -    -          -
T-370 ..... -          -          -          -          +    -    -          -
T-382 ..... -          -          -          -          -    -    -          -


DIAGNOSIS :  THE SIBLING IS A POTENTIAL CARRIER

NOTE :       THIS CONCLUSION IS BASED ON THE ANALYSIS
             OF THE UNDEFINED HAPLOTYPES.  BASED ON
             THE ANALYSIS OF THE DEFINED HAPLOTYPES
             ALONE, THE SIBLING IS NOT A CARRIER.
```

APPENDIX III

Program Code

```
/**********************************************************************
**********************************************************************
**********************************************************************
**********                                                  **********
**********    M.Sc. IN COMPUTATION  PROJECT                 **********
**********    ============================                  **********
**********                                                  **********
**********     Title        :        PKU ANALYSIS           **********
**********     Name         :        Afzal M. Qureshi       **********
**********     ID           :        8405259                **********
**********     Supervisors :         Dr. R. Janicki         **********
**********                           Dr. P. Chang           **********
**********                                                  **********
**********************************************************************
**********************************************************************
**********************************************************************/


#include <stdio.h>
#include <string.h>
#include <ctype.h>


#define  NUM_DEFINED 46;
#define  NUM_UNDEFINED 384;


     struct  chrom {
          char  enz[10];
          char  status[3];
          };


     struct haplos {
          int   chrom_num;
          char  chrom_cuts[10];
          int   presence;
          };
     struct stack {
          int   haplo_num;
       struct stack   *next;
          };


     struct stack   *defined_path,   /* contains the entire family's
                             defined haplotypes' numbers */
              *undefined_path,    /* contains the entire family's
                             undefined haplotypes' numbers */
              *undefined_list,
              *defined_list;




/*************************************************************************
**  clrscr()  clears the screen on the computer terminal and starts the **
**  display on a new page.                                             **
*************************************************************************/

clrscr ()

{
  int  l;

  system ("clear", l);

  return;

}




/**************************************************************************
**  print_chart displays  a  table showing names of the restriction      **
**  enzymes, the symbols used for the data inputs, and the sizes of the  **
**  fragments (in kbp) that these symbols represent for the corresponding**
**  restriction enzymes digestion products.                             **
**************************************************************************/

print_chart ()
```

```
{

     printf ("\n\n\t\tPvuIIa\tBglII\tPvuIIb\tEcoRI\tXmnI\tMspI\tHindIII\tEcoRV\n")
     printf(" Symbol:\n   + ..........\t   6.0\t 1.7\t   9.1\t 11.0\t 6.5\t19.0\
4.0\t 25.0\n");
     printf("  - ..........\t 19.0\t 3.6\t 11.5\t 17.0\t 9.4\t23.0\t  4.2\t 30.0\n")
     printf("  = ..........\t\t\t\t\t\t\t  4.4\n");

     printf  ("\n\nGIVE THE INPUT VALUES FOR THE RESTRICTION ENZYMES PLEASE\n\n");

     return;

}





/**************************************************************************
**  name_assignments  stores the names of the restriction enzymes, used  **
**  for the analysis, under the 'chromos' structure.                     **
**************************************************************************/

name_assignments ( chromos )

  struct chrom  *chromos;

  {
    strcpy ( chromos[0].enz, "PvuIIa" );
    strcpy ( chromos[1].enz, "BglII" );
    strcpy ( chromos[2].enz, "PvuIIb" );
    strcpy ( chromos[3].enz, "EcoRI" );
    strcpy ( chromos[4].enz, "XmnI" );
    strcpy ( chromos[5].enz, "MspI" );
    strcpy ( chromos[6].enz, "HindIII" );
    strcpy ( chromos[7].enz, "EcoRV" );

    return;

  }




/*************************************************************************
**  haplo_assignments  stores the defined haplotypes (their configura- **
**  tions and their corresponding numbers ) under the 'haplotypes'     **
**  structure.  It also assigns the haplotypes.presence to equal 0     **
**  for all of the haplotypes.  The 0 will be replaced by a  1  later   **
**  on  by another function for the corresponding haplotypes if they   **
**  were determined to be present in the individual being examined.    **
***********************************************************************/


haplo_assignments (haplotypes)

  struct haplos  *haplotypes;

  {
    int  i,num;


    strcpy ( haplotypes[0].chrom_cuts, "+----+--" );
    strcpy ( haplotypes[1].chrom_cuts, "+----+++" );
    strcpy ( haplotypes[2].chrom_cuts, "+--++---" );
    strcpy ( haplotypes[3].chrom_cuts, "+--++-++" );
    strcpy ( haplotypes[4].chrom_cuts, "-+++-+-+" );
    strcpy ( haplotypes[5].chrom_cuts, "-+++-+--" );
    strcpy ( haplotypes[6].chrom_cuts, "-+-++---" );
    strcpy ( haplotypes[7].chrom_cuts, "+--+-+-+" );
    strcpy ( haplotypes[8].chrom_cuts, "++-+-+-+" );
    strcpy ( haplotypes[9].chrom_cuts, "+--+-+--" );
    strcpy ( haplotypes[10].chrom_cuts, "-+-+-+-+" );
    strcpy ( haplotypes[11].chrom_cuts, "+----+=+" );
    strcpy ( haplotypes[12].chrom_cuts, "+-+--+=+" );
    strcpy ( haplotypes[13].chrom_cuts, "---+-+--" );
    strcpy ( haplotypes[14].chrom_cuts, "---+-+-+" );
    strcpy ( haplotypes[15].chrom_cuts, "+--+--++" );
    strcpy ( haplotypes[16].chrom_cuts, "+--+----" );
    strcpy ( haplotypes[17].chrom_cuts, "-++++-++" );
    strcpy ( haplotypes[18].chrom_cuts, "+--++--+" );
    strcpy ( haplotypes[19].chrom_cuts, "--++-+--" );
    strcpy ( haplotypes[20].chrom_cuts, "-+++-+=+" );
    strcpy ( haplotypes[21].chrom_cuts, "+--+-+=+" );
    strcpy ( haplotypes[22].chrom_cuts, "-++++---" );
    strcpy ( haplotypes[23].chrom_cuts, "+----++-" );
    strcpy ( haplotypes[24].chrom_cuts, "+----+=-" );
    strcpy ( haplotypes[25].chrom_cuts, "++---+--" );
    strcpy ( haplotypes[26].chrom_cuts, "+----+-+" );
    strcpy ( haplotypes[27].chrom_cuts, "+---+-++" );
    strcpy ( haplotypes[28].chrom_cuts, "-+------" );
    strcpy ( haplotypes[29].chrom_cuts, "++--+---" );
    strcpy ( haplotypes[30].chrom_cuts, "+--++-+-" );
    strcpy ( haplotypes[31].chrom_cuts, "-+---+--" );
    strcpy ( haplotypes[32].chrom_cuts, "-+---+-+" );
    strcpy ( haplotypes[33].chrom_cuts, "-+-+-+--" );
```

```
   strcpy ( haplotypes[34].chrom_cuts, "-+-+++-+" );
   strcpy ( haplotypes[35].chrom_cuts, "-++--+--" );
   strcpy ( haplotypes[36].chrom_cuts, "-----+--" );
   strcpy ( haplotypes[37].chrom_cuts, "-++--+=-" );
   strcpy ( haplotypes[38].chrom_cuts, "+-++-+--" );
   strcpy ( haplotypes[39].chrom_cuts, "-+--+---" );
   strcpy ( haplotypes[40].chrom_cuts, "+--+-+++" );
   strcpy ( haplotypes[41].chrom_cuts, "-+-+----" );
   strcpy ( haplotypes[42].chrom_cuts, "+---++--" );
   strcpy ( haplotypes[43].chrom_cuts, "-+-++-++" );
   strcpy ( haplotypes[44].chrom_cuts, "-+-++-+-" );
   strcpy ( haplotypes[45].chrom_cuts, "++--+-++" );

   num = NUM_DEFINED;

   for ( i=0; i<num; ++i )
    {
    haplotypes[i].chrom_num = i+1;   /* assign the number of haplotype */
    haplotypes[i].presence = 0;      /* assign the presence to equal 0
                                    for each haplotype */
    }


    return;

}






/*********************************************************************
**  other_haplos_assignments  generates all 384 possibilities for     **
**  haplotypes that can be generated during the analysis.  They are   **
**  stored under the 'other_haplos' structure.  These possibilities   **
**  also include the defined haplotypes as well.   This function      **
**  assigns each of the haplotypes (defined and undefined) a number   **
**  from 1 to 384.  The  'other_haplos.presence' for each of these    **
**  haplos is set to equal  0  which may by changed later on to  1    **
**  by  another function if the corresponding haplotypes are          **
**  determined to be present in the individual being examined.        **
*********************************************************************/

other_haplos_assignments (other_haplos)

  struct haplos  *other_haplos;

 {
   int  i;
   int  p1,p2,p3,p4,p5,p6,p7,p8;
   int  q1,q2,q3,q4,q5,q6,q7,q8;
   char  haplo[8],buffer[3];

   strcpy ( buffer, "+-=" );

   p1=2;
   p2=2;
   p3=2;
   p4=2;
   p5=2;
   p6=2;
   p7=3;
   p8=2;

   i = 0;

/* this series of 'for-loops' generates the total 384 possibilities
   for haplotypes */

       for ( q1=0; q1<p1; ++q1 )
        {
           haplo[0] = buffer[q1];

           for ( q2=0; q2<p2; ++q2 )
            {
               haplo[1] = buffer[q2];

            for ( q3=0; q3<p3; ++q3 )
             {
               haplo[2] = buffer[q3];

               for ( q4=0; q4<p4; ++q4 )
                {
                   haplo[3] = buffer[q4];

                for ( q5=0; q5<p5; ++q5 )
                 {
                   haplo[4] = buffer[q5];

                   for ( q6=0; q6<p6; ++q6 )
                    {
                       haplo[5] = buffer[q6];

                       for ( q7=0; q7<p7; ++q7 )
                     {
                         haplo[6]=buffer[q7];

                         for ( q8=0; q8<p8; ++q8 )
```

```
                                  {
                                  haplo[7] = buffer[q8];

                                  strcpy (other_haplos[i].chrom_cuts,
                                                haplo);
                                  ++i;

                                  }

                             }

                               }

                      }

                      }

                 }

                 }

            }


    for ( i=0; i<384; ++i )
     {
     other_haplos[i].chrom_num = i+1;  /* assign numbers to these
                                           haplotypes */
     other_haplos[i].presence = 0;     /* set presence to eqaul  0  for
                                  each of these haplotypes   */
     }


     printf ("\n\n");

     return;

}





/***************************************************************************
**  get_string  gets the input value from the user and then eliminates   **
**  the unnecessary blanks (spaces) from the input.  It then returns the **
**  resulting string value to the calling procedure.                    **
***************************************************************************/
```

```
get_string (s)

  char  s[80];

{
  char  temp[80], c;
  int   i, j;

  gets (temp);
  i=0;
  j=0;

  while (1)
   {
     c = temp[i];
     ++i;

     if (c=='\0')
       {
      s[j] = '\0';
      break;
       }

     if (isspace(c))  continue;
     s[j] = c;
     ++j;
   }

  return;

}




/**************************************************************************
**  get_values  prompts the user to give the input values (+/-/=) for     **
**  the enzyme represented by  'chromos[i]'.  In case of error in the     **
**  input, an error message is displayed on the terminal and the user is  **
**  prompted to give the input values again.                                 **
**************************************************************************/

get_values (chromos, i)

  int  i;
  struct chrom  *chromos;

  {
    char  cut[80];
```

```
int   flag;

flag = 0;  /*  to check presence of error in the input string  */

printf("\n");

while ( 1 )
 {
 if ( flag==1 )
      {
      printf("ERROR IN INPUT!!!\n");
      flag = 0;
   }

 if (i==6)
    printf(" (i)  %s (+/-/=)\t:    ",chromos[i].enz);
 else
    printf (" (i)  %s (+/-)\t:    ",chromos[i].enz);

 get_string(cut);

 if ( cut[0]=='\0' )
   {
      chromos[i].status[0]=' ';
      break;
   }

  /*  store the input values under 'chromos.status'  */

 if ( i==6 )
   {
     if (strcmp(cut,"+")==0 || strcmp(cut,"-")==0 || strcmp(cut,"=")==0)
       {
      chromos[i].status[0] = cut[0];
       }
     else
      flag = 1;
   }

 else
   {
     if ( strcmp(cut,"+")==0 || strcmp(cut,"-")==0 )
       {
      chromos[i].status[0] = cut[0];
       }
     else
      flag = 1;
   }

 if ( flag==0 ) break;
}


while ( 1 )
 {
 if ( flag==1 )
      {
      printf("ERROR IN INPUT!!!\n");
      flag = 0;
   }

 if (i==6)
    printf (" (ii) %s (+/-/=)\t:    ",chromos[i].enz);
 else
    printf (" (ii) %s (+/-)\t:    ",chromos[i].enz);

 get_string(cut);

 if ( cut[0]=='\0' )
   {
       chromos[i].status[1]=' ';
       break;
   }

  /*  store the input values under 'chromos.status'   */

 if ( i==6 )
   {
     if (strcmp(cut,"+")==0 || strcmp(cut,"-")==0 || strcmp(cut,"=")==0)
       {
      chromos[i].status[1] = cut[0];
       }
     else
      flag = 1;
   }

 else
   {
     if ( strcmp(cut,"+")==0 || strcmp(cut,"-")==0 )
       {
      chromos[i].status[1] = cut[0];
       }
     else
      flag = 1;
   }

 if ( flag==0 ) break;

 }
```

```
   return;

}







/***************************************************************************
**  display_data  displays back the data entered in by the user.        **
***************************************************************************/

display_data ( chromos )

  struct chrom  *chromos;

{
     int  j;

     printf ("\n\nDATA ENTERED:\n");

     printf ("\n\t PvuIIa\t  BglII\t PvuIIb\t  EcoRI\t  XmnI\t  MspI\t HindII
EcoRV\n");
     printf("\t---------------------------------------------------------------\n")

        for ( j=0; j<8; ++j)
        {
           printf("\t|  %c/%c", chromos[j].status[0], chromos[j].status[1]);
        }

     printf ("\t|\n\n\n");

     return;

}
```

```
/**************************************************************************
**  make_changes  displays the inputs by the user for the restriction    **
**  enzyme cuts one by one, and after each display it asks the user if   **
**  he/she would like to make any changes to the input. If the answer is **
**  a 'yes' then it calls the 'get_values' function to replace the       **
**  previous cut with the new cut.                                          **
**************************************************************************/

make_changes ( chromos )

  struct chrom  *chromos;

 {
   char  s[80];
   int  i;

   clrscr();

   for (i=0; i<8; ++i)
    {
      printf ("\n  %s  \t:   %c/%c\n", chromos[i].enz, chromos[i].status[0],
                                     chromos[i].status[1]);
      while (1)
       {
       printf("\t\t\t\tchange?(y/n) :   ");
       get_string(s);

       if ( strcmp(s,"y")==0 || strcmp(s,"Y")==0 )
        {
          get_values (chromos,i);
          break;
        }

       if ( strcmp(s,"n")==0 || strcmp(s,"N")==0 ) break;

       }
    }

     display_data (chromos);

     return;

 }
```

```
/************************************************************************
**  get_nums_of_chroms  checks for the presence of the defined chromo-   **
**  somes (in 'haplotypes') for the individual being examined.  If       **
**  any of the defined haplotypes are present ( i.e. haplotypes.presence **
**  = 1 )  then the corresponding numbers of these haplotypes are stored **
**  in 'f_def'.  The function then checks for the undefined chromosomes  **
**  (in 'other_haplos') for the individual being examined.  If any of    **
**  undefined haplotypes are present (i.e. other_haplos.presence = 1)    **
**  then the corresponding numbers of these haplotypes are stored in     **
**  'f_undef'.                                                              **
*************************************************************************/


get_nums_of_chroms (haplotypes,other_haplos)

  struct haplos  *haplotypes, *other_haplos;

  {
    int  i, num;
    struct stack  *temp;

    num = NUM_DEFINED;

  /*  checking for the presence of defined haplotypes   */

    for ( i=0; i<num; ++i )
     {
     if (haplotypes[i].presence==1)
       {
          temp=(struct stack *) malloc(sizeof(struct stack));
          temp->haplo_num = haplotypes[i].chrom_num;
          temp->next = NULL;
          defined_list->next=temp;
          defined_list=temp;
       }
     }

     temp=(struct stack *) malloc(sizeof(struct stack));
     temp->haplo_num = 0;
     temp->next=NULL;
     defined_list->next=temp;
     defined_list=temp;


  /*  checking for the presence of undefined haplotypes   */

     for (i=0; i<384; ++i)
     {
    if (other_haplos[i].presence==1)
      {
         temp=(struct stack *) malloc(sizeof(struct stack));
         temp->haplo_num = other_haplos[i].chrom_num;
         temp->next = NULL;
         undefined_list->next = temp;
         undefined_list = temp;
      }
     }

    temp=(struct stack *) malloc(sizeof(struct stack));
    temp->haplo_num = 0;
    temp->next=NULL;
    undefined_list->next=temp;
    undefined_list=temp;

     return;


}
```

```
/************************************************************************
**  eliminate_children_haplos   eliminates those haplotypes from the   **
**  defined and undefined lists of patient/siblings that do not belong **
**  to the parents haplotypes' lists for their defined and undefined   **
**  haplotypes.  In other words, only those haplotypes are retained    **
**  for the patient and the siblings that correspond with the parents' **
**  haplotypes.                                                        **
************************************************************************/

eliminate_children_haplos ()

  {
   struct stack   *temp, *parent1, *parent2, *prev, *tempP1, *tempP2;
   int  flag;

/*  eliminating those haplotypes from the defined haplotypes list of the
   patient/siblings that do not belong to the parents' defined haplotypes
    list */

   temp = defined_path;
   parent1 = temp;
   tempP1 = parent1;
```

```
while ( temp->haplo_num != 0 )
 {
 temp = temp->next;
 }

temp = temp->next;
parent2 = temp;
tempP2 = parent2;

while ( temp->haplo_num != 0 )
 {
 temp = temp->next;
 }


while ( temp->next != NULL )
 {
    flag = 0;

 tempP1 = parent1;
 tempP2 = parent2;
 prev = temp;

 temp = temp->next;

 if ( temp->haplo_num == 0 )
   {
      continue;
   }

  while ( tempP1->haplo_num != 0 )
   {
      if ( temp->haplo_num == tempP1->haplo_num )
        {
         flag = 1;
         break;
        }
      tempP1 = tempP1->next;
   }

  while ( tempP2->haplo_num != 0 )
   {
      if ( temp->haplo_num == tempP2->haplo_num )
        {
         flag = 1;
         break;
        }
      tempP2 = tempP2->next;
   }
```

```
     if ( flag==0 )
       {
          prev->next = temp->next;
          temp = prev;
       }

     }


/*  eliminating those haplotypes from the undefined haplotypes list of the
    patient/siblings that do not belong to the parents' undefined haplotypes
     list */

    temp = undefined_path;
    parent1 = temp;
    tempP1 = parent1;

    while ( temp->haplo_num != 0 )
     {
     temp = temp->next;
     }

    temp = temp->next;
    parent2 = temp;
    tempP2 = parent2;

    while ( temp->haplo_num != 0 )
     {
     temp = temp->next;
     }


    while ( temp->next != NULL )
     {
        flag = 0;

     tempP1 = parent1;
     tempP2 = parent2;
     prev = temp;

     temp = temp->next;

     if ( temp->haplo_num == 0 )
       {
          continue;
       }

      while ( tempP1->haplo_num != 0 )
       {
          if ( temp->haplo_num == tempP1->haplo_num )
            {
              flag = 1;
              break;
            }
          tempP1 = tempP1->next;
       }

      while ( tempP2->haplo_num != 0 )
       {
          if ( temp->haplo_num == tempP2->haplo_num )
            {
              flag = 1;
              break;
            }
          tempP2 = tempP2->next;
       }

     if ( flag==0 )
       {
          prev->next = temp->next;
          temp = prev;
       }

     }

   return;

}
```

```
/***************************************************************************
**  eliminate_parents_haplos  eliminates those haplotypes (defined and   **
**  undefined) from the parents' lists that do not belong to any of      **
**  their children's haplotypes list.  At least 2 children must be       **
**  included in the family analysis in order to eliminate any of the     **
**  haplotypes from the parents' haplotypes lists.                     **
***************************************************************************/

eliminate_parents_haplos ( family )

  int  family;

{
   struct stack  *temp, *prev, *parent1, *tempP1, *parent2, *tempP2, *children;
  struct stack  *buffer;
  int   flag, flagP1, flagP2;

  if ( family == 3) return;

/*  removing those haplotypes from the defined haplotypes lists of the
    parents that do not belong in any of their children's defined
    haplotypes list  */

  temp = defined_path;

  parent1 = (struct stack *) malloc(sizeof(struct stack));
  parent1->haplo_num = 0;
  parent1->next = NULL;
  tempP1 = parent1;

  while ( temp->haplo_num != 0 )
   {
    buffer = (struct stack *) malloc(sizeof(struct stack));
    buffer->haplo_num = temp->haplo_num;
    buffer->next = NULL;
    tempP1->next = buffer;
    tempP1 = buffer;
    temp = temp->next;
   }

  buffer = (struct stack *) malloc(sizeof(struct stack));
  buffer->haplo_num = 0;
  buffer->next = NULL;
  tempP1->next = buffer;

  temp = temp->next;

  parent2 = (struct stack *) malloc(sizeof(struct stack));
  parent2->haplo_num = 0;
  parent2->next = NULL;
  tempP2 = parent2;

  while ( temp->haplo_num != 0 )
   {
    buffer = (struct stack *) malloc(sizeof(struct stack));
    buffer->haplo_num = temp->haplo_num;
    buffer->next = NULL;
    tempP2->next = buffer;
    tempP2 = buffer;
    temp = temp->next;
   }

  buffer = (struct stack *) malloc(sizeof(struct stack));
  buffer->haplo_num = 0;
```

```
buffer->next = NULL;
tempP2->next = buffer;

temp = temp->next;
tempP1 = parent1;
tempP2 = parent2;

while ( tempP1->next != NULL )
 {
  flag = 0;
  prev = tempP1;
  tempP1 = tempP1->next;
  children = temp;

  while ( children )
    {
       if ( tempP1->haplo_num == children->haplo_num )
        {
        flag = 1;
        break;
        }
       children = children->next;
    }

        if ( flag==0 )
    {
      prev->next = tempP1->next;
      tempP1 = prev;
    }

 }

parent1 = parent1->next;
tempP1 = parent1;
flagP1 = 0;

while ( tempP1->haplo_num != 0 )
 {
  ++flagP1;
  tempP1 = tempP1->next;
 }

while ( tempP2->next != NULL )
 {
  flag = 0;
  prev = tempP2;
  tempP2 = tempP2->next;
  children = temp;

  while ( children )
```

```
     {
        if ( tempP2->haplo_num == children->haplo_num )
         {
         flag = 1;
         break;
         }
        children = children->next;
     }

         if ( flag==0 )
    {
      prev->next = tempP2->next;
      tempP2 = prev;
    }

 }

tempP2->next = temp;
parent2 = parent2->next;
tempP2 = parent2;
flagP2 = 0;

while ( tempP2->haplo_num != 0 )
 {
  ++flagP2;
  tempP2 = tempP2->next;
 }

if ( flagP1 >= 2 )
  {
  temp = defined_path;
  while ( temp->haplo_num != 0 )
   {
       temp = temp->next;
   }

  temp = temp->next;
  defined_path = parent1;
  tempP1->next = temp;
  }

if ( flagP2 >=2 )
  {
  temp = defined_path;
  while ( temp->haplo_num != 0 )
    {
       temp = temp->next;
    }

  temp->next = parent2;
```

```
    }


/*  removing those haplotypes from the undefined haplotypes lists of the
    parents that do not belong in any of their children's undefined
    haplotypes list  */

  temp = undefined_path;

  parent1 = (struct stack *) malloc(sizeof(struct stack));
  parent1->haplo_num = 0;
  parent1->next = NULL;
  tempP1 = parent1;

  while ( temp->haplo_num != 0 )
   {
    buffer = (struct stack *) malloc(sizeof(struct stack));
    buffer->haplo_num = temp->haplo_num;
    buffer->next = NULL;
    tempP1->next = buffer;
    tempP1 = buffer;
    temp = temp->next;
   }

  buffer = (struct stack *) malloc(sizeof(struct stack));
  buffer->haplo_num = 0;
  buffer->next = NULL;
  tempP1->next = buffer;

  temp = temp->next;

  parent2 = (struct stack *) malloc(sizeof(struct stack));
  parent2->haplo_num = 0;
  parent2->next = NULL;
  tempP2 = parent2;

  while ( temp->haplo_num != 0 )
   {
    buffer = (struct stack *) malloc(sizeof(struct stack));
    buffer->haplo_num = temp->haplo_num;
    buffer->next = NULL;
    tempP2->next = buffer;
    tempP2 = buffer;
    temp = temp->next;
   }

  buffer = (struct stack *) malloc(sizeof(struct stack));
  buffer->haplo_num = 0;
  buffer->next = NULL;
  tempP2->next = buffer;
```

```
temp = temp->next;
tempP1 = parent1;
tempP2 = parent2;

while ( tempP1->next != NULL )
 {
  flag = 0;
  prev = tempP1;
  tempP1 = tempP1->next;
  children = temp;

  while ( children )
    {
       if ( tempP1->haplo_num == children->haplo_num )
        {
        flag = 1;
        break;
        }
       children = children->next;
    }

        if ( flag==0 )
   {
     prev->next = tempP1->next;
     tempP1 = prev;
   }

 }

parent1 = parent1->next;
tempP1 = parent1;
flagP1 = 0;

while ( tempP1->haplo_num != 0 )
 {
  ++flagP1;
  tempP1 = tempP1->next;
 }

while ( tempP2->next != NULL )
 {
  flag = 0;
  prev = tempP2;
  tempP2 = tempP2->next;
  children = temp;

  while ( children )
    {
       if ( tempP2->haplo_num == children->haplo_num )
```

```
          {
         flag = 1;
         break;
          }
         children = children->next;
     }

         if ( flag==0 )
    {
      prev->next = tempP2->next;
      tempP2 = prev;
    }

  }

 tempP2->next = temp;
 parent2 = parent2->next;
 tempP2 = parent2;
 flagP2 = 0;

 while ( tempP2->haplo_num != 0 )
  {
   ++flagP2;
   tempP2 = tempP2->next;
  }

  temp = undefined_path;
  while ( temp->haplo_num != 0 )
    {
   temp = temp->next;
    }

  temp = temp->next;
  undefined_path = parent1;
  tempP1->next = temp;

  temp = undefined_path;

  while ( temp->haplo_num != 0 )
   {
   temp = temp->next;
   }
  temp->next = parent2;


 return;

}
```

```
/*************************************************************************
**  diagnosis  determines if the sibling is a carrier of the pku disease **
**  or not.  It first compares the haplotypes obtained for the sibling   **
**  with those obtained for the patient.  if any of the haplotypes that  **
**  are present in the sibling ( defined or undefined haplotypes )       **
**  are also present in the patient then the sibling is a crrier of the  **
**  pku disease.  Other wise, the sibling is not a carrier.  This        **
**  function then displays the diagnostic statement onto the terminal    **
**  screen.                                                                 **
*************************************************************************/


diagnosis ( i )

   int  i;

 {
   int   flag, count;
   struct stack  *temp, *patient, *tempP;

   temp = defined_path;

   while ( temp->haplo_num != 0 )
    {
     temp = temp->next;
    }
   temp = temp->next;

   while ( temp->haplo_num != 0 )
    {
     temp = temp->next;
    }
   temp = temp->next;

   patient = temp;

   while ( temp->haplo_num != 0 )
    {
     temp = temp->next;
    }

   count = i-2; /* to get the number of the sibling */
```

```
while ( count != 1 )
 {
  temp = temp->next;
  if ( temp->haplo_num == 0 )     --count;
 }
 temp = temp->next;

flag = 0;
while ( temp->haplo_num != 0 )
 {
  tempP = patient;

  while ( tempP->haplo_num != 0 )
   {
     if ( tempP->haplo_num  == temp->haplo_num )
       {
       flag = 1;
       break;
       }
     tempP = tempP->next;
   }

  if ( flag == 1 )
   {
     printf ("\n\nDIAGNOSIS :  THE SIBLING IS A POTENTIAL CARRIER \n\n");
     return;
     break;
   }

  temp = temp->next;

 }


temp = undefined_path;

while ( temp->haplo_num != 0 )
 {
  temp = temp->next;
 }
temp = temp->next;

while ( temp->haplo_num != 0 )
 {
  temp = temp->next;
 }
temp = temp->next;

patient = temp;
```

```
while ( temp->haplo_num != 0 )
 {
  temp = temp->next;
 }

count = i-2; /* to get the number of the sibling */

while ( count != 1 )
 {
  temp = temp->next;
  if ( temp->haplo_num == 0 )    --count;
 }
 temp = temp->next;

flag = 0;
while ( temp->haplo_num != 0 )
 {
  tempP = patient;

  while ( tempP->haplo_num != 0 )
   {
     if ( tempP->haplo_num  == temp->haplo_num )
       {
       flag = 1;
       break;
       }
     tempP = tempP->next;
   }

  if ( flag == 1 )
   {
     printf ("\n\nDIAGNOSIS :  THE SIBLING IS A POTENTIAL CARRIER \n\n");
     printf ("NOTE :       THIS CONCLUSION IS BASED ON THE ANALYSIS\n");
     printf ("             OF THE UNDEFINED HAPLOTYPES.  BASED ON \n");
     printf ("             THE ANALYSIS OF THE DEFINED HAPLOTYPES \n");
     printf ("             ALONE, THE SIBLING IS NOT A CARRIER.\n\n");
     return;
     break;
   }

  temp = temp->next;

 }


printf ("\n\nDIAGNOSIS :   THE SIBLING IS NOT A CARRIER \n\n");


return;
```

```
}




/*************************************************************************
**  compare  is called by the function 'determine_haplotypes' to check if **
**  the haplotype generated from the input by the user belongs to the     **
**  'defined haplotype list' (hapotypes) or if it belongs to one of       **
**  the undefined haplotypes (other_haplos).  The presence of that        **
**  haplotype is then marked by assigning haplotypes.presence = 1, or     **
**  other_haplos.presence = 1, depending on which haplotype it            **
**  corresponds to.                                                      **
*************************************************************************/

compare ( haplo, haplotypes, other_haplos )

  char   haplo[8];
  struct haplos  *haplotypes, *other_haplos;

  {
    int  i,j,k,num,tag;

    tag = 0;
    num = NUM_DEFINED;

    for ( i=0; i<num; ++i )
     {
       k=0;

       for (j=0; j<8; ++j)
      {
        if ( haplo[j]==haplotypes[i].chrom_cuts[j] ) ++k;
      }

       if (k==8)
        {
          haplotypes[i].presence = 1;
          tag = 1;
          break;
        }

     }

    if ( tag == 0 )
      {
         for ( i=0; i<384; ++i )
```

```
          {
            k=0;

            for (j=0; j<8; ++j)
          {
            if ( haplo[j]==other_haplos[i].chrom_cuts[j] ) ++k;
          }

            if (k==8)

            {
               other_haplos[i].presence = 1;
               break;
            }
           }
      }

    return;

  }




/***********************************************************************
**  determine_haplotypes  takes the values given by the user and stored **
**  in 'chromos', and generates the possible haplotypes from them.   If **
**  a null value was given for any of restriction enzyme cuts, this     **
**  function replaced that with a '+/-' for that restriction enzyme;    **
**  and if that restriction enzyme was 'HindIII', then a '+/-/=' is     **
**  assigned for that cut.                                              **
**  It then calls the function 'compare' to assign these haplotypes to  **
**  either the defined haplotypes list (haplotypes) by assigning        **
**  haplotypes.presence = 1, or the undefined haplotypes list (other_   **
**  haplos) by assigning other_haplos.presence = 1.                   **
***********************************************************************/

determine_haplotypes ( chromos, haplotypes, other_haplos )

   struct chrom  *chromos;
   struct haplos  *haplotypes, *other_haplos;

  {
     int  q1,q2,q3,q4,q5,q6,q7,q8,i,n;
     int  p1,p2,p3,p4,p5,p6,p7,p8;
     char  haplo[8];

     n = 0;
```

```
p1 = 2;
p2 = 2;
p3 = 2;
p4 = 2;
p5 = 2;
p6 = 2;
p7 = 2;
p8 = 2;

for ( i=0; i<8; ++i )
 {
   if ( chromos[i].status[0]==' ' || chromos[i].status[1]==' ' )
      {
      chromos[i].status[0] = '+';
      chromos[i].status[1] = '-';
         if ( i==6 )
           {
           chromos[i].status[2] = '=';
           p7=3;
           }
      }
  }

if ( chromos[0].status[0] == chromos[0].status[1] ) p1=1;
if ( chromos[1].status[0] == chromos[1].status[1] ) p2=1;
if ( chromos[2].status[0] == chromos[2].status[1] ) p3=1;
if ( chromos[3].status[0] == chromos[3].status[1] ) p4=1;
if ( chromos[4].status[0] == chromos[4].status[1] ) p5=1;
if ( chromos[5].status[0] == chromos[5].status[1] ) p6=1;
if ( chromos[6].status[0] == chromos[6].status[1] ) p7=1;
if ( chromos[7].status[0] == chromos[7].status[1] ) p8=1;


   for ( q1=0; q1<p1; ++q1 )
    {
         haplo[0] = chromos[0].status[q1];


        for ( q2=0; q2<p2; ++q2 )
         {
        haplo[1] = chromos[1].status[q2];


           for ( q3=0; q3<p3; ++q3 )
            {
            haplo[2] = chromos[2].status[q3];


             for ( q4=0; q4<p4; ++q4 )
```

```
              {
                haplo[3] = chromos[3].status[q4];


                  for ( q5=0; q5<p5; ++q5 )
                   {
                     haplo[4] = chromos[4].status[q5];


                    for ( q6=0; q6<p6; ++q6 )
                     {
                       haplo[5] = chromos[5].status[q6];


                      for ( q7=0; q7<p7; ++q7 )
                       {
                         haplo[6] = chromos[6].status[q7];


                          for ( q8=0; q8<p8; ++q8 )
                           {
                           haplo[7] = chromos[7].status[q8];
                           ++n;

                           compare (haplo, haplotypes,
                                    other_haplos);

                           }

                         }

                     }

                   }

              }

            }

         }

     }

    printf("\n");

    return;

}
```

```
/**********************************************************************
**  next_family_member  asks the user if he/she would like to continue **
**  the family analysis with another member of the family.             **
**  It returns '1' to the main program if the answer is yes; it returns**
**  '0' to the main program if the answer is no.                          **
**********************************************************************/

int next_family_member ()

{
  char  reply[80];

     printf("\n\n");

     while (1)
      {
      printf("\t   CONTINUE WITH NEXT FAMILY MEMBER?(y/n) :  ");
      get_string(reply);
      if ( strcmp(reply,"y")==0 || strcmp(reply,"Y")==0 ||
           reply[0]=='\0' || strcmp(reply,"n")==0 ||
                          strcmp(reply,"N")==0 ) break;
      }

    printf("\n");
    if ( (strcmp(reply,"y")==0) || (strcmp(reply,"Y")==0) ||
          reply[0]=='\0' ) return(1);
    else
     return (0);

}
```

```
/**********************************************************************
**  print_more  asks the user whether he/she would like to continue and  **
**  returns  1  if the answer is 'yes', and returns  0  if the answer is  **
**  'no'.                                                                     **
*****************************************************************************/
int print_more()

{
   char  reply[80];

   printf("\n\n");

   while (1)
      {
     printf("\t\t\tCONTINUE DISPLAY?(y/n)  :  ");
     get_string(reply);
     if ( strcmp(reply,"y")==0 || strcmp(reply,"Y")==0 ||
          reply[0]=='\0' || strcmp(reply,"n")==0 ||
                          strcmp(reply,"N")==0 ) break;
      }

    printf("\n");
    if ( (strcmp(reply,"y")==0) || (strcmp(reply,"Y")==0) ||
          reply[0]=='\0' ) return(1);
    else
     return (0);

}
```

```
/**************************************************************************
**  print_haplotypes  displays all the haplotypes generated from the    **
**  input by the user.  The haplotypes are categorized according to     **
**  their presence in either the defined haplotypes list (haplotypes)   **
**  or the undefined haplotypes list (other_haplos).  After every 20    **
**  lines, the user is asked whether he/she would like to continue with **
**  display.  Also, in case of new haplotypes being added to the        **
**  defined list, the printout will also contain, in brakets, the       **
**  number from the undefined haplotypes list that was used to          **
**  categorize that haplotype previously.                              **
**************************************************************************/

print_haplotypes (haplotypes, other_haplos, chromos)

  struct haplos  *haplotypes,*other_haplos;
  struct chrom   *chromos;

  {
   int  i,j,k,m,n,p,tag,num,reply;

/*  displaying the defined haplotypes in the individual    */

   printf ("\n\nDefined Haplotypes:\n\n\t");

   for ( i=0; i<8; ++i)
     {
       printf("\t%s", chromos[i].enz);
     }

   printf ("\n");

   tag = 0;
   k = 0;
   num = NUM_DEFINED;

   for ( i=0; i<num; ++i )
    {
    if ( haplotypes[i].presence == 1 )
      {
       tag = 1;

       if ( i>45 )
        {
          for ( m=0; m<384; ++m )
         {
            p = 0;
            for ( n=0; n<8; ++n )
              {
                if ( haplotypes[i].chrom_cuts[n] ==
                         other_haplos[m].chrom_cuts[n] ) ++p;
              }
            if ( p==8 )    break;
         }
          printf("\n%3d (T-%d) ", haplotypes[i].chrom_num,
                         other_haplos[m].chrom_num);
         }
      else
       printf ("\n%3d  ..... ", haplotypes[i].chrom_num );

       for (j=0; j<8; ++j)
        {
          printf ("\t  %c", haplotypes[i].chrom_cuts[j]);
        }

       ++k;
       if ( !(k%20) )         /* after every 20 lines the user is asked
                         if he/she would like to continue  */
         {
```

```
        reply = print_more();
        clrscr();
        if ( reply==0 ) return;
       }

     }

    }

    if ( tag == 0 ) printf ("\n\n\t\t\t\tNONE EXIST!!!\n\n");

    reply = print_more();
    clrscr();
    if ( reply==0 ) return;

/*  displaying the undefined haplotypes in the individual   */

   printf ("\n\nOther Haplotypes:\n\n\t");

   k=0;
   tag = 0;

   for ( i=0; i<8; ++i)
     {
       printf("\t%s", chromos[i].enz);
     }

   printf ("\n\t");

   for ( i=0; i<384; ++i )
    {
    if ( other_haplos[i].presence == 1 )
      {
       tag = 1;
       printf("\nT-%d \t.....", other_haplos[i].chrom_num);

       for (j=0; j<8; ++j)
        {
          printf ("\t  %c", other_haplos[i].chrom_cuts[j]);
        }

       ++k;
       if ( !(k%20) )          /*  after every 20 lines the user is asked
                         if he/she would like to continue  */
         {
           reply = print_more();
           clrscr();
           if ( reply==0 ) return;
         }
        }

      }

     if ( tag == 0 ) printf ("\n\n\t\t\t\tNONE EXIST!!!\n\n");

    printf ("\n\n");

    return;

}







/***********************************************************************
**  print_family_data  prints out the entire family's haplotypes after **
**  the family analysis had been done.  It calls the function        **
**  'print_haplotypes' for each family member in order to display      **
**  their respective haplotypes.                                          **
***********************************************************************/

print_family_data ( haplotypes, other_haplos, chromos, family )

  struct haplos  *haplotypes,*other_haplos;
  struct chrom   *chromos;
  int  family;

{
   struct stack  *temp_def, *temp_undef;
   int   i, j, reply, num;

   num = NUM_DEFINED;
   temp_def = defined_path;
   temp_undef = undefined_path;

   for ( i=0; i<family; ++i )
    {
       clrscr();

       for (j=0; j<num; ++j)    haplotypes[j].presence = 0;
       for (j=0; j<384; ++j)    other_haplos[j].presence = 0;

       if ( i==0 )
          printf("\n\nPOSSIBLE  HAPLOTYPES  FOR  PARENT  #1  AFTER  FAMILY  ANALYSI
:\n\n");
```

```
       if ( i==1 )
          printf("\n\nPOSSIBLE  HAPLOTYPES  FOR  PARENT  #2  AFTER  FAMILY  ANALYSI
:\n\n");

       if ( i==2 )
          printf("\n\nPOSSIBLE  HAPLOTYPES  FOR  THE  PATIENT  AFTER  FAMILY  ANALYSI
:\n\n");

       if ( i>2 )
        {
          printf("\n\nPOSSIBLE  HAPLOTYPES  FOR  SIBLING  #%d  AFTER  FAMILY  ANALYSI
:\n\n", i-2);
        }

      while ( temp_def->haplo_num != 0 )
       {
       haplotypes[temp_def->haplo_num - 1].presence = 1;
       temp_def= temp_def->next;
       }

      temp_def = temp_def->next;

      while ( temp_undef->haplo_num != 0 )
       {
       other_haplos[temp_undef->haplo_num - 1].presence = 1;
       temp_undef = temp_undef->next;
       }

      temp_undef = temp_undef->next;

      print_haplotypes ( haplotypes, other_haplos, chromos );

      if ( i>2 )   diagnosis (i);

      if ( i == (family-1) ) break;

      reply = next_family_member();
      if ( reply==0 ) break;

    }

  return;

}
```

```
/*************************************************************************
**  patient_analysis_only  does the job of the main program if the user **
**  had selected the option of 'PATIENT ANALYSIS ONLY' from the main    **
**  menu.  It does the entire analysis for the patient in determining   **
**  his/her haplotypes by calling up the apropriate funtions.           **
*************************************************************************/

patient_analysis_only ( chromos, haplotypes, other_haplos )

  struct chrom   *chromos;
  struct haplos  *haplotypes, *other_haplos;

 {
     int  i;
     char  s[80];
     clrscr();

     printf("\n\nENTER DATA FOR THE PATIENT PLEASE :\n\n");

     print_chart();    /* display the table showing the names of the
                         restrction enzymes and the symbols to be used
                         for the input  */

        name_assignments (chromos);     /* assign the names of the restrction
                                  enzymes to 'chromos'  */

           haplo_assignments (haplotypes);    /* assign the defined haplotypes
                                    to 'haplotypes'  */

     other_haplos_assignments (other_haplos);    /* generate all the
                                             undefined as well as the
                                            defined haplotypes and
                                            assign them to
                                            other_haplos   */

        for ( i=0; i<8; ++i )     /* get input values (+/-/=) for each R.E. */
       {
           get_values (chromos,i);
       }

     display_data (chromos);  /* display the data entered back to the user */

     while (1)
      {
        printf("ANY CHANGES?(y/n) :   ");
        get_string(s);
        if ( strcmp(s,"y")==0 || strcmp(s,"Y")==0 ) make_changes(chromos);
        if ( strcmp(s,"n")==0 || strcmp(s,"N")==0 )  break;
      }

     clrscr();

    /*  generate the haplotypes from the input by the user and assign them
     to 'haplotypes' if they belong to the defined haplotypes list, and
     to 'other_haplos' if they belong to the undefined haplotypes list */

     determine_haplotypes (chromos, haplotypes, other_haplos);

    /*  display all the generated haplotypes for the individual onto the
     terminal screen    */

     print_haplotypes (haplotypes,other_haplos,chromos);

     return;

}





/***************************************************************************
**  initial_menu  displays the initial menu to the user and prompts       **
**  him/her to select from it.  It then returns the corresponding number **
**  to the main program.                                                  **
***************************************************************************/

int initial_menu ()

{
   char  s[80];

   clrscr();

   printf("\n\n\n\t****************************************************");
   printf("********");
   printf("\n\t***************************************************");
   printf("********\n\t****\t\t\t\t\t\t\t****\n");
   printf("\t****\t\t\t\t\t\t\t****\n");
   printf("\t****\t      OPTIONS:\t\t\t\t\t****\n\t****\t\t\t\t\t\t\t****\n");
   printf("\t****\t\t  1. FAMILY ANALYSIS\t\t\t****\n");
   printf("\t****\t\t  2. PATIENT ANALYSIS ONLY\t\t****\n");
   printf("\t****\t\t  3. QUIT\t\t\t\t****\n\t****\t\t\t\t\t\t\t****\n");
   printf("\t****\t\t\t\t\t\t\t****\n");
```

```
  printf("\t****************************************************");
  printf("********\n");
  printf("\t****************************************************");
  printf("********\n\n\n");

  while (1)
   {
    printf("\tSelect from above please:    ");
    get_string(s);
    if ( strcmp(s,"1")==0 )
      {
         return (1);
         break;
      }
    if ( strcmp(s,"2")==0 )
      {
         return (2);
         break;
      }
    if ( strcmp(s,"3")==0 )
      {
         clrscr();
         return (3);
         break;
      }
    }

    return (0);

}




/***************************************************************************
**********************************   MAIN   ********************************
***************************************************************************/


main ()
 {
    char  s[80];
    int  i, reply, family;
    struct chrom  chromos[8];
    struct haplos  haplotypes[46], other_haplos[384];

   defined_list = (struct stack *) malloc(sizeof(struct stack));
   defined_list->next = NULL;
   defined_path = defined_list;
 undefined_list = (struct stack *) malloc(sizeof(struct stack));
 undefined_list->next = NULL;
 undefined_path = undefined_list;

family = 0;

while (1)
 {

   if (family==0)    /* in the very first case display the menu to
                 the user and ask if he/she would like to
                 analyse the whole family or the patient alone */
    {
      reply = initial_menu();   /* display the initial menu and prompt
                           the user to select from it */
      if (reply==3) break;        /* if the user chose the quit option */

      if (reply==2)          /* if the user chose the 'PATIENT ANALYSIS ONLY'
                           option  */
        {
        patient_analysis_only (chromos,haplotypes,other_haplos);
        break;
        }
    }

   ++family;

   clrscr();

   if (family==1) printf("\n\nENTER DATA FOR PARENT #1 PLEASE : \n\n");
   if (family==2) printf("\n\nENTER DATA FOR PARENT #2 PLEASE : \n\n");
   if (family==3) printf("\n\nENTER DATA FOR THE PATIENT PLEASE : \n\n");
   if (family>3)  printf("\n\nENTER DATA FOR A SIBLING PLEASE : \n\n");


   print_chart();    /* display the table showing the names of the
                  restrctoin enzymes and the symbols to be used
                  for the input  */

      name_assignments (chromos);     /* assign the names of the restrction
                              enzymes to 'chromos'  */

        haplo_assignments (haplotypes);    /* assign the defined haplotypes
                                to 'haplotypes'  */

   other_haplos_assignments (other_haplos);    /* generate all the
                                     undefined as well as the
                                    defined haplotypes and
                                    assign them to
                                    other_haplos   */
   for ( i=0; i<8; ++i )     /* get input values (+/-/=) for each R.E. */
   {
      get_values (chromos,i);
   }


 display_data (chromos);  /* display the data entered back to the user */

 while (1)
  {
    printf("ANY CHANGES?(y/n) :   ");
    get_string(s);
    if ( strcmp(s,"y")==0 || strcmp(s,"Y")==0 ) make_changes(chromos);
    if ( strcmp(s,"n")==0 || strcmp(s,"N")==0 )  break;
  }

 clrscr();

/*  generate the haplotypes from the input by the user and assign them
 to 'haplotypes' if they belong to the defined haplotypes list, and
 to 'other_haplos' if they belong to the undefined haplotypes list */

 determine_haplotypes (chromos, haplotypes, other_haplos);

/*  display all the generated haplotypes for the individual onto the
 terminal screen   */

 print_haplotypes (haplotypes,other_haplos,chromos);

 get_nums_of_chroms (haplotypes,other_haplos);

 reply = next_family_member();  /* ask the user if he/she would like to
                     continue with the next family member */
 if ( reply==0 ) break;


}

defined_path = defined_path->next;
undefined_path = undefined_path->next;

 if ( family==1 || family==2 )
  {
 print_family_data (haplotypes, other_haplos, chromos, family);
  }

 if ( family > 2 )
  {
   eliminate_children_haplos ();    /* to remove unwanted haplotypes
```

```
                              from the patient/siblings
                              defined and undefined
                              haplotypes lists.   */
       eliminate_parents_haplos (family); /* to remove unwanted haplotypes
                              from the parents' defined and
                              undefined haplotypes lists */

       print_family_data (haplotypes, other_haplos, chromos, family);

      }


    free(defined_path);
    free(defined_list);
    free(undefined_path);
    free(undefined_list);



}




/****************************** END OF PKU.C ********************************/
```