OBJECT-ORIENTED DESIGN WITH C++: A LINEAR PROGRAMMING APPLICATION

By

YORE-PHANG TZAY

Bachelor of Art

National Cheng-Chi University

Taipei, Taiwan

Republic of China

1983

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1988

OBJECT-ORIENTED DESIGN WITH C++:
A LINEAR PROGRAMMING
APPLICATION

Thesis Approved:

Donald D Fisher
Thesis Adviser

J Chandler

K E Case

Norman N. Durham
Dean of the Graduate College

PREFACE

This study concerns the discussion of decomposing large and complex software system to independent and individual modulars with the concept of object-oriented methodology. The thesis presents the results of developing the linear programming object-oriented package.

I would like to express my deep appreciation and thanks to my major advisor Dr. Donald D. Fisher for his continuous guidance, motivation, and encourage throughout this study.

I would extend my appreciation to Dr. John P. Chandler,
211
and Dr. Kenneth Case for their advise and support.

I am very grateful to my parents, Mr. and Mrs. Tzay, to my husband Tong-Yuan Koo for their encouragement and understanding.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

1.1 Current Software Problem 1
1.2 Literature Review 2
1.3 Objectives 3

II. OBJECT-ORIENTED PROGRAMMING IN C++. 5

2.1 Elements of Object-Oriented Language . 5
2.2 Information Hiding 5
2.3 Data Abstraction 8
2.4 Dynamic Binding 11
2.5 Inheritance 11

III. OBJECT-ORIENTED DESIGN METHODOLOGY 14

3.1 Description of The Design Methodology . 14
3.2 An Implementation Illustration 17

IV. IMPLEMENTATION PROCESS 22

4.1 Problem Description 22
4.2 Object Classes Description 24
4.3 First Implementation: The Simplex Method 26
4.4 Second Implementation: Components Reusability Demonstration 29
4.5 An Illustration of Object Interchange . 31
4.6 Data Manipulation Objects 34
4.7 Evaluation. 40
4.8 Results 42

V. SUMMARY, CONCLUSION, AND SUGGESTIONS 47

5.1 Summary and Conclusion 47
5.2 Suggestions 50

BIBLIOGRAPHY 51

APPENDIX A. DESCRIPTION OF THE SIMPLEX METHOD AND THE REVISED SIMPLEX METHOD 55
APPENDIX B. CATOLOG OF OBJECTS 61

APPENDIX C. C++ PROGRAM LISTING 84

LIST OF FIGURES

Figure Page

1. An Example of User Defined Type 7

2. An Illustration of Overloaded Operators 9

3 Base Class and Derived Class 10

4. Class Inheritance and Virtual Function 13

5. Generic Relationship of Classes Objects 23

6. Relationship of Objects Used in the Program 25

7. Relationship of Data Manipulation Objects and Creator Objects 36

8. Double Buffering Digram 36

9. Source Bulk 43

10. Execution Bulk 44

11. Execution Time 44

12. Organization of A Simplex Tableau 57

LIST OF TABLES

Table Page

I. Table of B Size Versus Selected Value of Available Memory and Number of Constraints 37

CHAPTER I

INTRODUCTION

1.1 Current Software Problem

The essence of the software problem is simply that it is much more difficult to build large, complex, and long-lived software systems than to build unreliable, inflexible and difficult to maintain systems. Modifying a large and complex system is difficult too, since usually no one person understands its complete structure. Modifying a large and complex software system is necessary to refer to external documentation, which always seems to be weeks behind the actual design work(2,6).

At an intuitive level, it seems difficult to grasp the full impact of the software problem. As David Fisher(2,8) has said, "Although there are many widely recognized symptoms, the underlying problems are not well delineated and there are few useful quantitative measures for assessing either the importance of perceived problems or the effectiveness of proposed solutions". He enumerates some of the symptoms of the software problem as unresponsiveness, unreliabilitity, uncost, unmodifiability, untimeliness, untransportability, and unefficiency.

1.2 Literature Review

The important developments in methodology and languages in the 1960's centered on functions and procedures, which is a program segment in terms of a name and a parameter list. At that time, people only knew how to check syntactic errors. Beginning in the late 1960's, abstraction was treated as a program organization technique. Programmers could add new notations and new data types to a base language, that is they tried to build an extensible language. But this work on extensibility died out, since it was difficult to keep independent extensions compatible when all of them modified the syntax of the base language. However, this influenced the abstract data types of the 1970's. In the early 1970's, a top-down programming methodology emerged, that is, hierarchies of models in which lower level models provide more detailed explanations for the higher level models. But there were two limitations of this methodology: one was that the final program did not preserve the series of abstractions through which it was created, and the other limitation was that informal descriptions did not provide precise information. Misunderstanding could complicate the program development and modification. Later on, the development of formal program specification helped alleviate this set of problems. In the late 1970's, the methodology of abstract data types emphasized locality of related collections of information. Data was emphasized rather than control, and the strategy was to package each data structure and its associated operations into a single module. The resulting module contained the information necessary to treat the data structure and its operations as a type. The objective was to treat such modules in the same way as ordinary types such as integers and reals were treated(34). This concept led to today's object-oriented methodology.

In an object-oriented system, every module is an object, that is, a package of data and procedures that operate on the data. Object-oriented design is the process of identifying objects that constitute a useful model of the problem at hand. In the early stages of designing a program, the need to partition the problem into objects stimulates the designers to identify its principal constituents and to specify their behavior and interaction. Dividing a problem into objects and defining actions that are "natural" for those objects actually simplifies programs. When the actions of an object are divided into the right kinds of "chunks", programmers can think and write code at a higher level. Therefore, as Dave Patterson has said : "thinking of an algorithm in terms of objects makes it easier to understand. This ease of understanding often comes not from the details of how a procedure is constructed but from not having to think about the details of the program(8)".

1.3 Objectives

The objectives of this paper are to apply object-oriented techniques to a linear programming package, from which the following studies can be conducted:

1.) to state the concepts of data abstraction, object oriented constructs, and module interface specification;

2.) to introduce an object-oriented design methodology that exploits the power of C++ and helps manage the complexity of large software solution;

3.) to implement in C++ the simplex method and the revised simplex method with modular data management to show off the benefits of object-oriented approach;

4.) to demonstrate the use of reusable components and interchangeable modulars in the implementation of the modular systems.

CHAPTER II

OBJECT-ORIENTED PROGRAMMING IN C++

2.1 Elements of Object-Oriented Language

Object-oriented languages employ a data or object-centered approach to programming. Instead of passing data to procedures, objects(data) are constructed so that they perform operations on themselves. To support object-oriented programming fully, a language must exhibit four characteristics : information hiding, data abstraction, dynamic binding, and inheritance. The details are presented in the following sections (10,28,41,42).

2.2 Information Hiding

Information hiding is important to ensure reliability and modifiability of software systems by reducing interdependencies between software components. The state of a software module is contained in private variables, visible only from within the scope of the module. Only a localized set of procedures directly manipulates the data. In addition, since the internal state variables of a module are not directly accessed from without, a carefully designed module interface may permit the internal data structures and procedures to be changed without affecting the implementation of other software modules. Most modern languages, even FORTRAN, to some degree, support information hiding.

The key concept in C++ is class. A class is a user-defined type. Classes provide data hiding, guarantee initialization of data, implicit type conversion for user-defined types, dynamic typing, user-controlled memory management, and mechanisms for overloading operators. An instance of a class is called an object. Access to objects of a class can be restricted to a set of functions declared as part of the class; such functions are called member functions. Objects of a class are created and initialized by member functions specifically declared for that purpose; such functions are called constructors. A member function can be specifically declared to "clear up" each object of a class when it is destroyed; such a function is called a destructor. The fundamental idea in defining a new type is to separate the incidental details of the implementation from the properties essential to the correct use of it. Such a separation can be expressed by channeling all use of the data structure and internal housekeeping routines through a specific interface(38). For example, Following is a conventional stack type which gives an example of stack object.

```
// private part : size, top, and s.
// public  part : constructor char_stack(), destructor
                  °char_stack, push(), and pop().

class char_stack {
   int size;
   char* top;
   char* s;
public:
   int status;
   char_stack(int sz) { status = 0;
                        top=s=new char[size=sz]; }
   ~char_stack()      { delete s; }
   void push(char c)  { if(top > size) status = 1;
                        else  *top++ = c;         }
   char pop()         { if(top-1 <= 0) status = 1;
                        else  return *--top;      }
};
```

The high level object sends a message to a subobject to execute some function. After control passes back to the sender, the sender should check the status returned by the subobject. If the status indicates failure, then the sender should decide either to require the "error handler" to handle this situation or to create another subobject to continue executing the jobs.

Figure 1. An Example of User Defined Type

2.3 Data Abstraction

Data abstraction could be considered a way of using information hiding. A programmer defines an abstract data type consisting of an internal representation plus a set of procedures used to access and manipulate the data.

In C++, the definition of a user-defined type (called a class) contains a specification of the data needed to represent an object of the type and a set of operations for manipulating such objects. The definition has two parts : a private part holding information that can only be used by its implementor, and a public part presenting an interface to users of the type.

C++ solves both these problems, information hiding and data abstraction, through two language features : operator overloading and derived classes. A programmer can define a meaning for operators when applied to objects of a specific class; in addition to arithmetic, logical, and relational operators, call() and subscripting [] can be defined, and both assignment and initialization can be redefined (see Figure 2).

Derived classes provide a simple, flexible, and efficient mechanism for specifying an alternative interface for a class and for defining a class by adding facilities to an existing class without reprogramming.

```
class vector{
   int size;                    // size of vector
   double* v;                   // pointer to first element
public:                         // public interfaces
   vector operator+(vector&);
   vector operator-(vector&);
   vector operator*(vector&);
   double operator[](int);
};


void f()
{
   vector a(4), b(4), c(4);
   vector e(10);
   vector d(10), f(10);
   double value;
   int    x, y, z;
   matrix m1(2,2), m2(2,2), m3(2,2);

   z = x+y;
   a = b+c;
   m3 = m1 + m2;
   e = d*f;
   value = e[3];
}
```

Binary vector operations require vector operands to be of the same size.

Figure 2. An Illustration of Overloaded Operators

```
class get_name{          // get_name is a base class
public:
   int lowmum;           // low bound for variable name
   int upnum;            // up bound for variable name
   char tname[LIMIT];    // name type tag
   char** name;          // array of pointers

   get_name(char*);
   ~get_name(){  delete[LIMIT] name;}
   int   upbound() { return upnum; }
   int   lowbound(){ return lownum;}
   char* operator[](int i) { return *(name+i); }
   void promptfor(int);
   void list();
   void append();
   void del();
};

// class get_varname is derived from class get_name
class get_varname : public get_name{
public:
   get_name  m;
};
```

Figure 3. Base Class and Derived Class

2.4 Dynamic Binding

A problem still exists if you wish to use a list, for example, to store heterogeneous elements. Compile-time solution, operator overloading, or generic program units are not sufficient for heterogeneous elements. The solution is dynamic binding.

Dynamic binding is needed to make full use of this code for putting other types of data onto the list. In a traditional procedure-oriented approach, trying to print an integer with a procedure designed to print character strings is potentially disastrous.

In C++, objects of a list need not be homogeneous. A list specified in terms of pointers to a class can hold objects of any class derived from that class. That is, it may be heterogeneous. This is probably the single most important and useful aspect of derived classes. This style of programming is often called object based or object oriented; it relies on operations applied in a uniform manner to objects on heterogeneous lists. The meaning of such operations depends on the actual type of the objects on the list(known only at runtime), not on just the type of the list elements(known to the compiler). Figure 4. gives an example.

2.5 Inheritance

Inheritance enables programmers to create classes and objects that are specializations of other objects. Creating a specialization of an existing class is called subclassing. The new class is a subclass of the existing class, and the existing class is the superclass of the new class. The subclass inherits instance variables, class variables, and methods from its superclass.

C++ provides derived classes method, which supports inheritance. Class derivation is a way of building one class from another. A derived class can be thought of as a variation of a general base type. Through virtual functions, various types derived from the same base class can be uniformly handled as if they were the base type. A base class can declare member functions that can be redefined with a variation of meaning for each derived class. This is the basis of what is often called object-oriented programming (see Figure 4).

```
struct employee {
   employee* next;
   char*     name;
   short     department;
   //... operations
   virtual void print();
};

void employee::print()
{
   cout << name << "\t" << department << "\n";
}

struct manager : employee {
   employee*  group;
   short      level;
   //.... operations
   void print();
};

void manager::print()
{
    employee::print();
    cout << "\tlevel " << level << "\n";
    //...
}

void f(employee* ll)
   for(; ll; ll=ll->next)  ll->print();
}

main()
{
    employee e;
    e.name = "J. Brown";
    e.department = 1234;
    manager m;
    m.name = "J. Smith";
    m.department = 1234;
    m.level = 2;
    m.next = &e;
    f(&m);
}
```

will produce:

```
J. Smith 1234
         level 2
J. Brown 1234
```

Figure 4. Class Inheritance and Virtual Function

CHAPTER III

OBJECT-ORIENTED DESIGN METHODOLOGY

In this chapter, the object-oriented design methodology is introduced and followed by the implementation of the simplex method object as an illustration.

3.1 Description of The Design Methodology

The object-oriented design is viewed as a software decomposition technique. That is, this methodology allows people to think of that body of information(variables) and actions (operations) as a single unit. The world is perceived as made up of objects, and our brain arranges information into chunks. For example, a programmer looks at an object in the real world that he wishes to model, its attributes(e.g. size, weight) become the instance variables, and its behaviors(e.g. departure, arrival) become the operations. By using objects in a programming language, programmers can map an existing pattern of thought into an object. Following is the description of the object-oriented design stages(2):

Realize the problem

This stage is not different from traditional methodology. Before the implementation of a software system starts, the problem should be analyzed and understood. And then the solution strategy can be developed.

Develop the solution strategy

Using the object-oriented methodology, an informal strategy is expressed in natural English description, in terms of concept from the problem. Then, a solution that map our view of the real world object to the computer solution algorithm is developed.

Formalize the strategy

All human languages revolve around two basic structures: nouns and verbs. Other language features, such as adjectives and adverbs, serve only to amplify or constrain and are therefore dependent upon the noun/verb constructs. We would expect, then, that programming languages would reflect a similar structure of objects(nouns) and operations (verbs). Therefore, when the solution strategy of a problem to be solved is formalized, underline the nouns from the strategy description to be objects; underline the verbs and collect the house-keeping routines to be a set of operations. Then group these objects and operations to be a class.

The detailed implementation sub-stages are identifying the objects, identifying the operations and establishing the interfaces.

Identifying the objects. Underlining the nouns of interest in a software system, these nouns become the objects. But names that refer to the same objects and objects should be regarded as one object.

Identifying the operations on the objects. A set of procedures that operate on the data objects of a class is called the set of operations of that class. The data object of a class represents private data for that class. The operations (procedures) associated with the class are legitimate operators for that data. Since objects of interest have been identified as mentioned above, the operations performed on these data items can be defined now. By choosing active verbs from the informal strategy, such as "push" an element to stack, "pop" an element out of stack, and collecting the operations on the objects to be the set of operations of that class. The set of operation specifications of a class, by hiding the implementation detail, provides interfaces to the user. For example:

```
class char_stack {
  int size;          /* private data */
  char* top;
  char* s;
public:
  char_stack(int);   /* interface operations */
  void push(char);
  char pop();
};
```

Establishing the interfaces. Given the specification of the set of operations, the user interfaces are established. By referring to the interfaces, a user can perform operations on the objects, even though the implementation of the operations is hidden from the users. The user manipulates objects by declaring an object instance and specifying the certain operation(s) to tell the object what to do. This is generally called "sending the object a message". For example:

```
char_stack  A_stack;
char_stack  B_stack;
char_stack  C_stack;
char        c = 'B';
char        ch= 'F';

A_stack.push(ch);   // Push character 'F' to A_stack;
ch = A_stack.pop(); // pop 'F' from  A_stack and put
C_stack.push(ch);   // it into C_stack.
B_stack.push(c);    // push 'B' to B_stack.
```

Now, A_stack is empty, B_stack contains character 'B', and C_stack contains character 'F'.

3.2 An Implementation Illustration

By using this methodology, the implementation of the simplex method class is demonstrated belows:

Informal strategy

For the sake of clarity and maintainability, it is important that computer solutions directly map to the real world. The simplex method informal strategy is stated as follows:

> The simplex method allows us to choose a initial basic feasible solution with all real activities at the zero level. Then evaluate the current solution, if the solution is optimal, stop. Otherwise, select a variable from among the current nonbasis variables to enter the basis which can improve the value of the objective function. Determine which variable will leave the basis. Express the problem in terms of the new basis. Thus we go from one feasible solution to another until an optimum is reached. The computation is performed on a tableau(see Appendix A for detail).

Identify the objects

Underline the nouns from the informal strategy above and identify the objects of interest to be private data variables. The objects which compose the simplex method algorithm can be listed as following:

initial feasible solution
simplex tableau
nonbasis variables
basis variables

entering variable(variable which enters the basis)

leaving variable(variable which leaves the basis)

objective function value

optimal value

<u>Identify the operations on the objects and build the interface</u>

The operations to be performed on these objects are listed below:

`tableau(int numcon,int numact);`

Tableau constructor which allocates spaces for the simplex tableau.

`~tableau();`

Tableau destructor which deallocates spaces when program terminates.

`void tableauinit(transform&);`

Initialize tableau variables.

`void setup(get_coef&);`

Append an identity matrix,that is basis matrix, to tableau and put a very big value(big M) into the cost vector positions corresponding to the identity matrix.

`void solveinit();`

Initialize the first iteration variables,that is, find the variable which enters the basis and the variable which leaves the basis.

void ck_optimal(get_coef&);

Check if the solution is optimal, unbounded, or infeasible.

int entering();

Find the variable which enters the basis set. Return the index of the entering variable.

int leaving();

Find the variable which leaves the basis set. Return the index of the leaving variable.

void row_divide();

Divide the pivot row by the pivot element. Pivot element can not be zero, because when choosing leaving variable, zero valued elements are not considered.

void row_eliminate();

Eliminate the rows except pivot row. See the simplex method algorithm step 5 in Appendix A.

void computeshadowprice();

Compute values of Zj and Zj - Cj.

double obj();

Compute the objective function value.

void simplex(transform&, get_coef&);

Simplex method drive routine. Transform& is the address of the transform object and

get_coef& is the address of the get_coef
object.

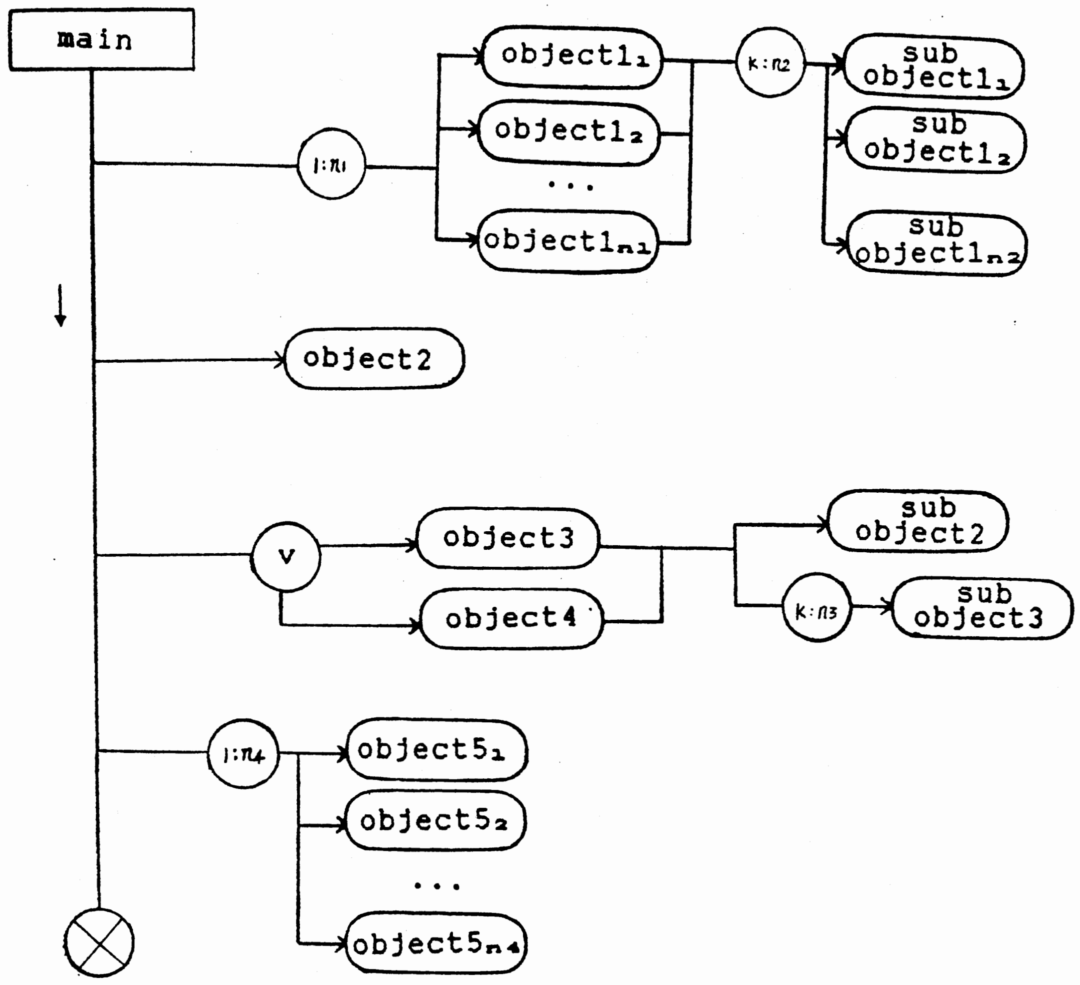
CHAPTER IV

IMPLEMENTATION PROCESS

4.1 Problem Description

The main objective is to define a set of software components that is suitable for solving linear programming. These components range from high to low level. The high level components are composed of the low level ones. None of the high level components should need to be changed when the implementation or representation of the lower components are altered. Ideally, these components should remain usable even if the LP algorithm itself is altered. Although some algorithm's performance implementation, efficiency is not the primary consideration of this problem. For example, if objects and subobjects are related according to Figure 5, the control goes from the top to the bottom along the control flow line. The user can pick up any one from the group of the object 1, which achieves a certain function with different version of implementation. And some subobjects are created by object 1 to share jobs with object 1.

When the object 2 gets a message from the main driver routine, it responds the command by executing the computation. The object 3 and the object 4 are different objects which produce some subobjects to achieve the same function, so the user can chose any one of the object 3 and 4. Then the control goes to one instance of the object 5, when the object 5 finishes its job the program is done.



Symbol	meaning
→	Control flow
⊗	Program terminates
→	Sending a message
▭	Object/subobject
(k:n)–[	Selecting k objects from n objects
(k:n)–▭	Selecting k objects from one class
(v)	Or

Figure 5. Generic Relationship of Classes Objects

4.2 Object Classes Description

This LP package is constructed using several object classes, such as input, output, transform, vector, matrix, simplex, and revised simplex classes. Each of these classes is implemented by using the methodology described in chapter III, and are put into Appendix C.

Class input is the manager of getting input data. The input object is responsible for generating two window objects and getting the input data. Only the non-zero input data value and its index are stored in an array as a sparse representation of matrix. Class transform plays a role of transforming a sparse matrix and a sparse vector to a full matrix and a full vector to facilitate the simplex computation. Because after iterating, some non-zero elements become zero and some zero elements become non-zero. The simplex object is the center manager of the simplex computation in the tableau format. The simplex object generates a matrix object to store the constraints coefficients and several vector objects to store the right hand side values and the objective function values. When the simplex computation continues, the simplex object sends messages to the simplex object's member functions and vector objects to execute the computation iteratively until the optimal solution is reached. The output object is responsible for producing a window object and displaying the optimal solution to the window. The revised simplex class is implemented to handle the computation with the revised simplex algorithm. It generates instances of vector class and matrix class. By sending messages to objects of vector, matrix and revised simplex itself, the revised simplex object executes the revised simplex computation easily and successfully. Data manipulation objects have the duty of handling huge volume of data and are described in section 4.6.

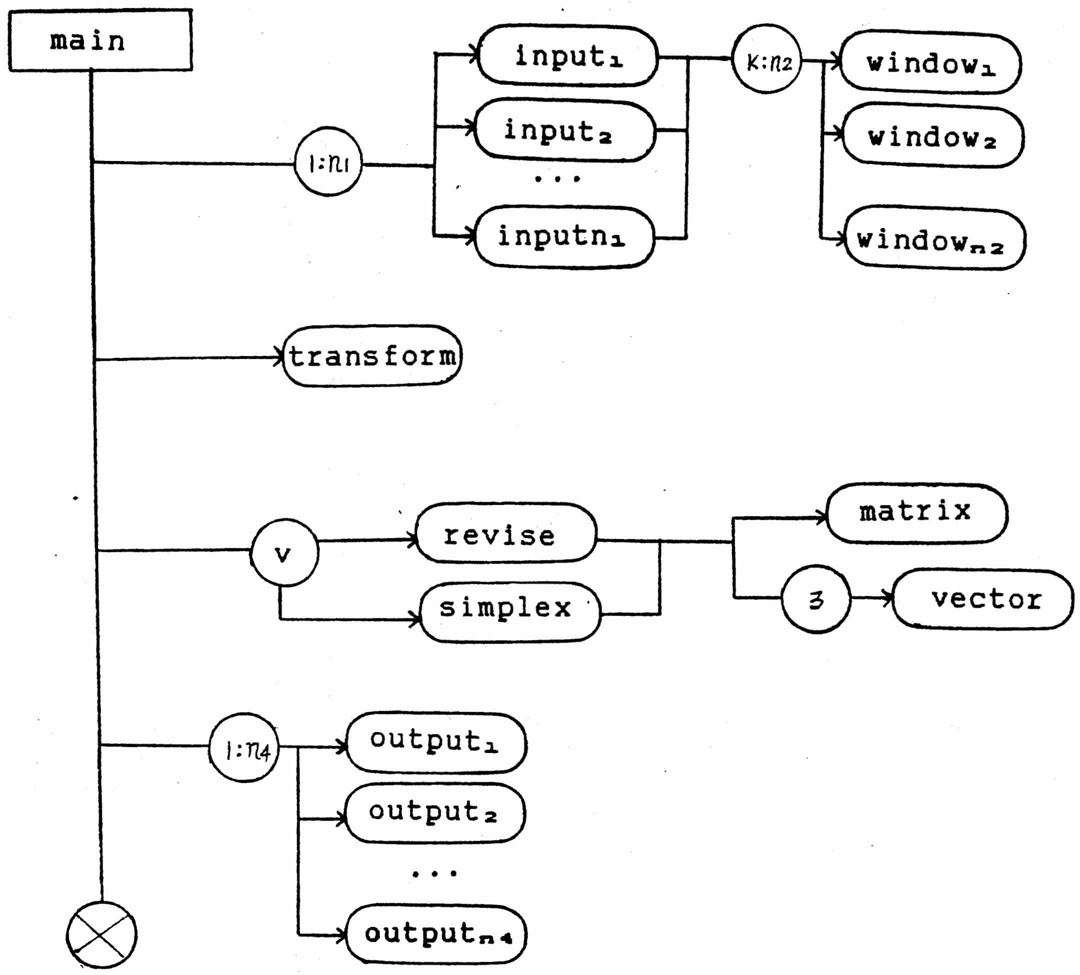


Figure 6. Relationship of Objects Used in the Program

4.3 First Implementation: The Simplex Method

The smain is the overall manager of the simplex method computation(Figure 6). One instance of the input object, which is produced by the smain, generates two window objects, one for displaying help-menu and the other for getting input data. Smain sends a message to the input object to get input data. The input object responses this message by executing the command and storing data in a condensed form, that is only non-zero values are stored. When the job is done, the control is passed back to the smain.

The smain gets the control back, it generates a transform object and sends a message to the transform object. The transform object obeys this command by transforming the sparse representation of the constraints matrix to a full matrix and the sparse vectors to full vectors.

Then, the smain produces an instance of the simplex class called the simplex. The simplex is the center controller of the computation of the simplex method. One matrix and a couple of vector objects are created by the simplex to be the tableau and the right hand side vector, objective function coefficient vector and other vectors. The simplex object itself contains a set of member functions. By sending messages to its member functions and vector objects, the computation is executed until the optimal solution is reached.

In more detail, when the control passes from the smain to the simplex, the simplex constructor creates the matrix object to be the tableau and a couple instances of vector class to be the right hand side column, shadow price row (i.e., Zj - Cj row), and cost row; then the member functions are invoked to execute the simplex method. Operations listed below are those which send messages to objects of other classes. Other operations which do not send messages out can be referred in Appendix C. (See Appendix A and C for details.)

```
simplex()
{
  1. tableauinit() : clear up spaces allocated;
  2. setup() : find initial feasible solution.
  3. entering() : find a variable to enter the basis;
  4. leaving() : find a variable to leave the basis;
  5. while(not done) do
        pivot_set() : set pivot element;
        row_divide() : step 5 of Appendix A;
        row_eliminate() : step 6 of Appendix A;
        compute_shadow_price() : update shadow price;
        compute_objective_value() : update objective
                                    value;
     end while
  6. ck_optimal() : check if the solution is unbounded,
                    infeasible, or optimal.
}
```

```
simplex constructor()
{
    1. A matrix object is generated to be the
       constraints matrix;
    2. Couple of vector objects are generated to be right
       hand side column, Zj - Cj row, cost row, pivot row,
       and others.
}

entering() -- See the simplex method step 3 of Appendix A.
{
    Send a message to a vector object to find the most
    negative element of the shadow vector object.
}

leaving() -- See Appendix A the simplex method step 4.
{
    1. Send a message to matrix object to get the column
       vector a(r).  a(r) = A(r)(j), for j = 1..m;
    2. Ask vector object to do the vector vector division.
    3. Find the smallest ratio of step 2. by invoking
       vector smallest() operation;
}
```

When the simplex computation is done, the control returns to the smain and an output object is generated to process the output. A window object is produced by the output object for displaying the optimal solution. The smain then askes for another LP problem to process, if none, the program terminates.

4.4 Second Implementation : Components Reusability Demonstration

The revised simplex class is constructed in order to demonstrate the lower level components are reusable(Figure 6). These low level components are the input class object, the transform class object, the matrix class object, the vector class object, and the output class object. The rmain driver routine produces these objects and sends messages to them to execute the input data, transformation, and displaying final solution as described in the first implementation. Even the computation algorithm is changed from the simplex method to the revised simplex method; these components remain unchanged and reusable. The only change within main routine is that the simplex object is switched to the revised simplex object whoes algorithm is different from the simplex object.

Within the revised simplex object class, a set of operations is implemented. Control flow for the revised simplex is given below,

```
revised_simplex() -- See Appendix A and Appendix C
{
  1. Revise_init() is called to clear spaces allocated;
  2. Set_up() is invoked to set up initial variables;
  3. Invoke entering() to find a variable which enters
```

```
     the basis;
  4. Call leaving() to find a variable which leaves the
     basis;
  5. while(not done) do
       Calculate the invert of new basis matrix;
       Invoke entering();
       Invoke leaving();
  6. end while
  7. Compute objective function value.
}

entering() -- this function implements the revised simplex
              method step 2 of Appendix A.
{
   1. Send a message to the matrix object to compute
      vector matrix multiplication.
   2. Require the matrix object to execute matrix vector
      multiplication.
   3. Ask the vector object to do vector vector
      subtraction.
   4. Require the vector object to return the most
      negative value.
}

leaving() -- See Appendix A the revised simplex method step
             3.
{
   1. Send a message to matrix object to compute the
```

```
      matrix vector multiplication.  The returned vector is
      called b'.
   2. Ask matrix vector multiplication to be computed,
      the returned vector object is called alpha
   3. Send vector object to do the vector division, e.g.
      b' / alpha.  If there exist a nonpositive value in
      alpha, the quotation is set a predefined constant.
   4. Send a message to vector object to find the smallest
      element of step 3.
}

compute_obj()
{
  1. send a message to the vector object to compute vector
     b' multiplies cost vector.
  2. vector operation dot() is invoked,
     and the return value is the objective function value.
}
```

And other operations which do not send messages out can be referred in Appendix C.

4.5 An Illustration of Object Interchange

In order to demonstrate that objects are interchangeable, a second form of input class is constructed. Instead of getting input from the keyboard as the first input class does, the second input class gets input data from a file. Also, a different form of output class is constructed, it writes the final result of a LP problem to a file rather than displays the solution to a window object. Therefore, we have two kinds of input classes, two kinds of output classes, and two kinds of simplex algorithm, any combination of these objects is allowed. The differences of the input objects or the output objects can be found in the program listing. Two examples are given to demonstrate this feature.

Combination 1 includes input from the keyboard, execution of the simplex algorithm and output to a window object.

PROGRAM SEGMENT	COMMENTS
`main()`	
`{`	IN is an instance of input object.
`input IN;`	Main sends a message to ask IN to
`IN.get_data();`	get input data. IN responds by getting input data from the keyboard.
`transform FORM;`	FORM is an object of class
`FORM.processmx();`	transform. Main require FORM to
`FORM.processvec();`	process sparse matrix to full matrix, sparse vector to full vector.
`simplex S;`	Main generates a simplex object
`S.simplex(FORM,IN);`	and askes it to execute the simplex method calculation.

```
  output  OUT;            An output object is produced by
   OUT.diplay(S,IN);      main, and main sends a message to
                          OUT, which displays the solution
 }                        on the screen.
```

Combination 2 includes input from an input file, execution of the revised simplex algorithm, and output to a file. The character ' is used to indicate the class version is different from the old one used above, it does not appear in the program segment actually. But the implementations of the new one is different from the old one.

```
 main()
 {
  input'  IN;              input' is another input class.
   IN.get_data();          IN gets data from a input file.
                           The character ' is deleted in the
                           program code actually.

  transform  FORM;
   FORM.processmx();
   FORM.processvec();

  revise  R;               The revised simplex algorithm is
   R.simplex(FORM,IN);     used.

  output'  OUT;           Output' is another version of the
   OUT.diplay(R,IN);      output object.  OUT writes solution
 }                        to a file.
```

Different classes with same function have identical specifications, although their implementations are different. If the user would like to replace one object with another, e.g. switch input to input', he has to use "#include" statement to include the header file of the class he wants to use. Then recompile the program segments whose including header file(s) are changed, relink and run the program. For example, files input1.h and input2.h contain the header files of of two different input classes the input' and the input which achieve the same destination. The user wants to switch the input object to the input'. He has to include input2.h file and delete input1.h file, then recompile, relink, and run the program. In order to save the execution bulk which comes from including too many unnecessary header files into the program, decomposing program into small modules is suggested. Each module only contains the code which is necessary. By selecting and interchanging objects, the user can construct the program up to his decision.

4.6 Data Manipulation Objects

The product form of the inverse is more economical for computer spaces and is used here to implement the revised simplex method(See Appendix A). The inverse of a basis after p iterations is $B^{-1} = E_pE_{p-1}...E_i...E_1$, where E_i is the elementary matrix at iteration i. Hence, a multiplication of $B^{-1}a(r)$ becomes $E_pE_{p-1}...E_1a(r)$. But if the volume of data is so large that even using the product form, the inverse of the basis still can't fit into the primary memory. For example, if a LP problem with 300 constraints runs at least 400 iterations to get the optimal solution, then the spaces for E(1..400) are

$$8 * 300 * 400 = 960K \text{ bytes.}$$

(The number '8' means each floating point value occupies 8 bytes in the IBM PC.) It is obvious that the main memory of IBM PC which only has 640K bytes is not big enough for this problem. Therefore, by using the secondary memory to store the constraints matrix can conpensate for insufficient primary memory. Then this program continually retrieves part of the constraints matrix into the primary memory to do the computation. The detailed process is described as follows (see Figure 7).

The input object does the same function as described in section 4.2. The transform object has a rule to decide whether or not to produce data manipulation objects to handle the large amount of data. The testing result shows that if the size of the constraints matrix is less than 100 by 200, then we do not bother to store the constraints matrix in secondary memory. Beyond this limit, the transform object creates a data manipulation subobject the ck_memory to check the amount of available main memory and allocate two buffers for transferring data to and from the diskette. This technique is called double buffering technique(see Fig-

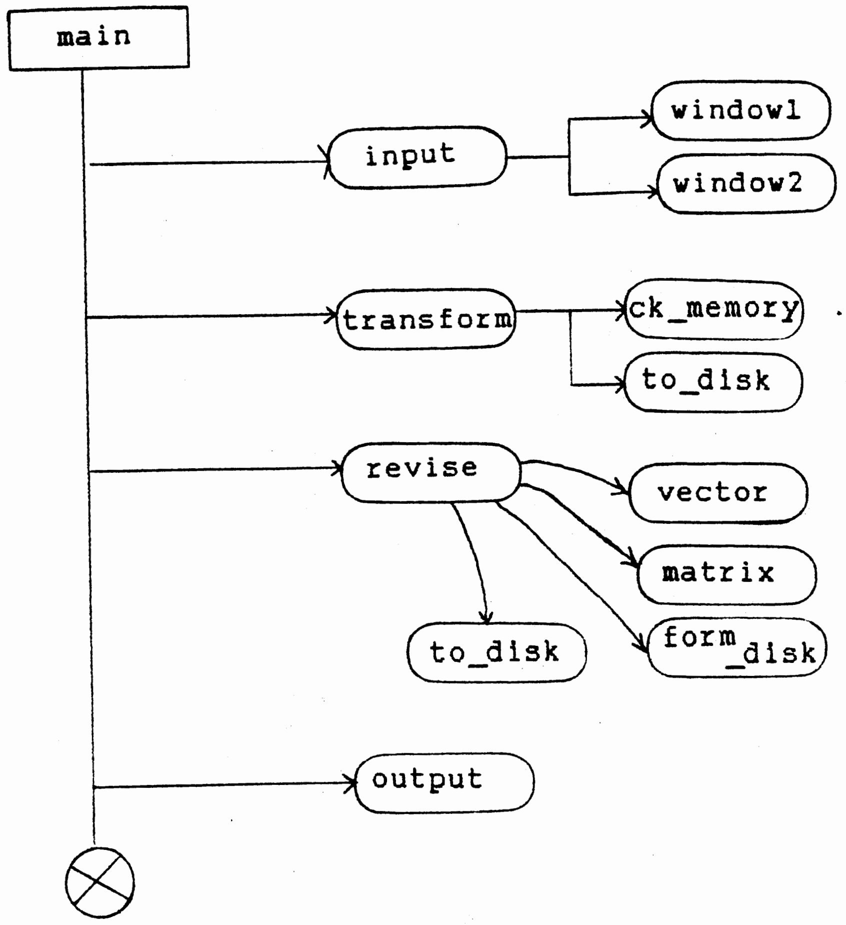


Figure 7. Relationship of Data Manipulation Objects and Creator Objects

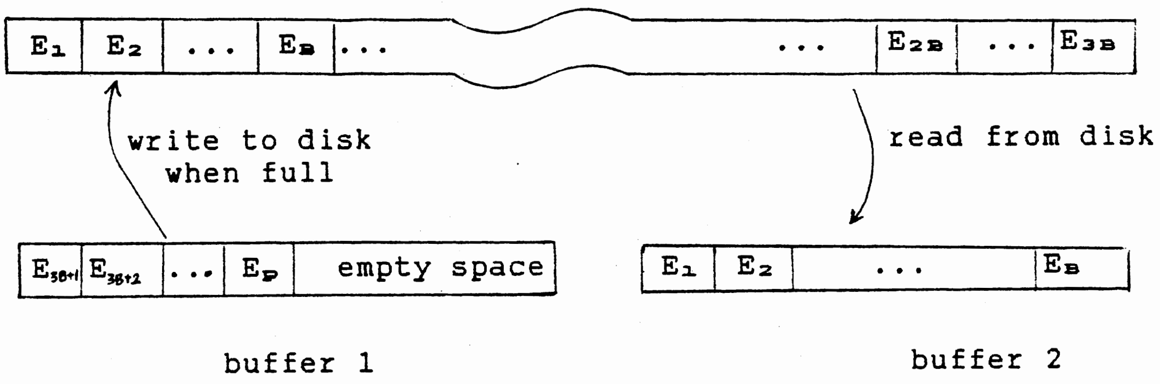


Figure 8. Double Buffering Diagram(at iteration p)

TABLE I

TABLE OF B SIZE VERSUS SELECTED VALUE OF AVAILABLE MEMORY AND NUMBER OF CONSTRAINTS

M = Memory available N = Number of constraints
B size = the number of elements of B

	100K	128K	200K	512K	(K bytes)
64	100	128	200	512	
100	64	81	128	327	
128	50	64	100	256	
150	42	54	85	218	
200	32	40	64	163	
256	25	32	50	128	
500	12	16	25	65	

Many disk drives use the sector(512 bytes) as the smallest transmittable unit, so attempts to transmit less than one sector at a time could lead to inefficiency. Some drives may operate most efficiently by transmitting several sectors at once. This suggests that program buffers be able to hold at least one sector. For a program which processes all of the data in a file in sequence, a large buffer generally results in better performance than smaller buffer. Let M be available of memory, N be the number of constraints, and B be number of columns in the buffer for memory size M and number of constraint N. Under certain available memory and number of constraints, we can determine B size by using the formula(the number '8' indicates each floating point value occupies eight bytes, and '2' indicates two buffers are allocated):

$$B = M / (8 * 2 * N)$$

It is helpful by using this formula to determine the reasonable buffer size to make the IO operations more efficiently. So, some values of size B may work better than the others(see Table I). Therefore, for a given run the buffer size is fixed, a linear mapping from buffer to elementary matrix is applied. The transform object subdivides buffer into appropriate length vectors which correspond to columns of the elementary matrices. For example, the available memory M is 128K bytes, the number of constraints N is 64, and for two buffers of floating point entries(8 bytes each), one buffer contains

$$(128 * 1024) / (64 * 8 * 2) = 128 \text{ elementary matrices.}$$

If there are several LP problems with different numbers of constraints, then the fewer of the number of constraints, and the more elementary matrices one buffer can contain, and the number of the IO operations needed is less, so the performance is better.

After two buffers are allocated, the transform object gets data from the input object one column after another. After collecting several full columns(i.e. includes zero and non-zero elements) in the buffer, the transform object creates a data manipulation subobject the to_disk, and the to_disk writes the contents of the buffer to the diskette. In like manner, the entire constraints matrix is written to the secondary memory in parts.

Then control is passed to the revised simplex object. Spaces for vectors are allocated in the primary memory and they are reused iteration after iteration. In order to save space, only one vector space for the non-basis matrix is allocated. Because the vector matrix multiplication can be executed as a vector vector multiplication whose times of computation are based on the size of the vector. For example,

```
t = y * A

for i = 1 to n, by 1
    t(i) = y * a(i)
```

where, y, t are vectors;

n is the number of columns in matrix A;

a(i) is the ith column of matrix A.

The computation of $B^{-1}a(r)$ is equal to $E_pE_{p-1}...E_1a(r)$. The double buffering is used again. A new elementary matrix is generated at each iteration, and the first buffer provides spaces for the new generated elementary matrix which is appended to the end of the buffer. When the buffer is full, the to_disk object is invoked to write the buffer to the diskette entirely. The second buffer provides spaces for the elementary matrices chunk which is retrieved from diskette. For example, at iteration p, the inverse of the basis matrix is E(1..p). The from_disk object retrieves the block which contains E(1..k) from disk to the second buffer(where p >= ck, c is a constant) to start the computation. When this part of computation is done, the second block of E(k+1..2k) is retrieved and followed by another part of computation. Therefore, control passes back and forth between the revised simplex object and the from_disk subobject executing the computation and retrieving data from diskette in turns until the revised simplex calculation is done. By the way, the elementary matrices residing in diskette need not be updated, because they are reused at each iteration. When the revised simplex method calculation is done, the output object is produced and the optimal solution is printed.

4.7 Evaluation

G. Pascoe(28) gives four fundamental characteristics of object orientation: information hiding, data abstraction, dynamic binding, and inheritance. J. D. McGreger(23) uses these four features to evaluate his object-oriented program the scissors paper-stone game. This LP package is evaluated by these four criteria.

1.) This package supports information hiding. This LP package supports information hiding by invoking operations through messages. The message sender does not know how the detailed computation is executed. The sender only cares about what is done.

2.) This package supports data abstraction. Data abstraction is achieved by operator overloading. Vector object and matrix object define several mathematical operators, i.e. +, -, *, /, and etc... So, data abstraction is definitely achieved.

3.) Dynamic binding is utilized in this package. "Dynamic binding increases flexibility by permitting the addition of new classes of objects without having to modify existing code(28)." Section 4.5 provides a good example to show that dynamic binding is utilized in this package.

4.) Inheritance is used in this package. The input class uses three lower level classes for processing the input variable names and constraints names. They are class get_name, class get_conname, and class get_varname. The get_name class is a base class. The other two classes are derived from it. So, inheritance is utilized in this package.

5.) This package shows the feature of component reusability. This package shows the resuability by implementing two different computation algorithms by using the same lower level class objects. See section 4.4 for detail.

6.) This package shows the possibility of components interchange. This feature is presented by combining objects in the main driver routine to achieve the same goal. See section 4.5 for details. And these components can be interchanged back and forth.

Stepwise abstraction(47) is one of the methods applied to program verification, which is the process of verifying the correctness of a program in a bottom-up method by verifying the correctness of a low level subprogram and replacing the subprogram with its intended function, thus, advancing the program to a higher level of abstraction. This package is implemented by using the bottom-up method and the higher level components are constructed after the lower level ones are built and tested correctly. Therefore, object-oriented design technique actually involves the method of stepwise abstraction in it.

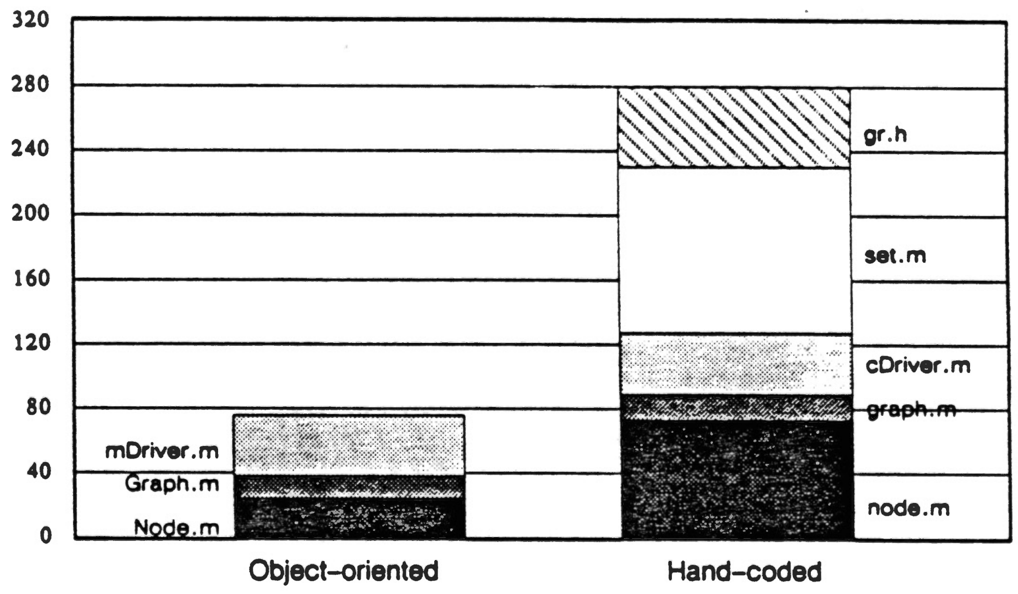
4.8 Result

Restrictions

One restriction comes from the machine, and the other comes from the object-oriented methodology.

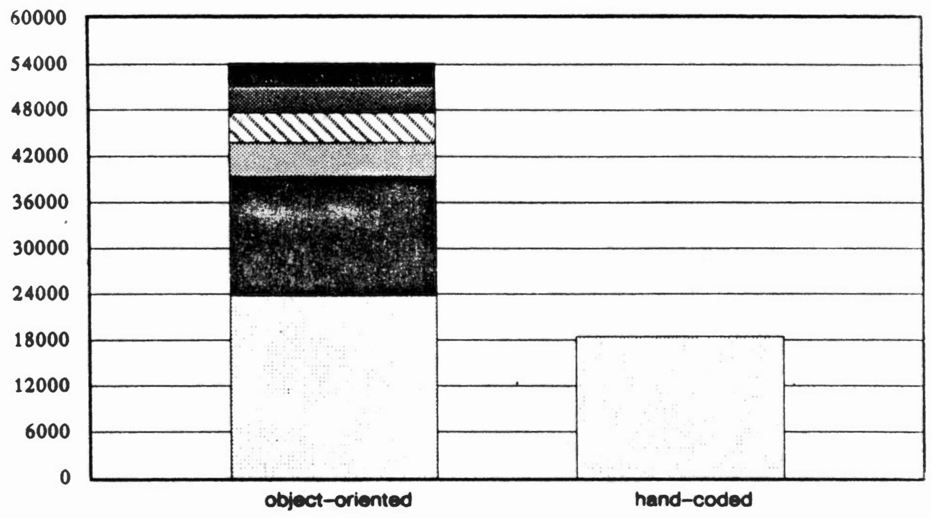
The LP package is implemented on a IBM PC XT system with the RAM size of 640K bytes. Some constants are defined under this environment. If the environment is changed, then these predefined constants should be modified to make the program more efficient and powerful. The file which these constants are defined is called "simheader.hxx".

The cost and benefits of the new technology have been measured by B. J. Cox(6). In comparison with a problem is solved by object-oriented mechanism and by conventional mechanism. The benefit was a 370% savings in development effort(based on the number of lines of source code; Figure 9), and the cost was a 43% penalty in execution speed(Figure



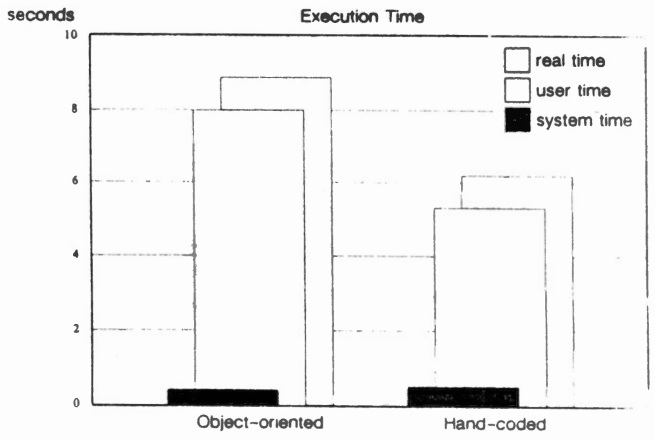
Two solutions to the same programming problem, one using object-oriented programming and the other using conventional C coding practices. The conventional version required 3.7 times as many lines of code to develop. This is primarily because conventional programming does not allow code to be captured in libraries and reused to nearly as great a degree as does object-oriented programming(6).

Figure 9. Source Bulk(lines)



When the two applications are linked to create a complete application, the hand-coded application is significantly smaller. This is primarily because inherited methods provided more generality than was needed in this application. This tends to make small applications larger than usual, but this may well be offset in large applications by the fact that inherited code is loaded only once and shared in many places, while conventional programming duplicated the functionality each time it is used(6).

Figure 10. Execution Bulk(6)



The machine cost of object-oriented programming is measured by comparing the speed of the two applications for the same 1409 line test file(6).

Figure 11. Execution Time(6)

Some maximum sizes of the predefined constants of the LP package are listed below:

For a MPSX format input file:

Input	MAX size
NONZEROS	1000
COLUMNS	200
ROWS	100

For the keyboard input :

Input	MAX size
VAR/ROW NAME CHAR	8
COLUMNS	5
ROWS	15

Evaluate the Ability to Change Modules

In a conventional procedural mechanism, several procedures can access a variable via passing by reference. The variable can be modified by any one of these procedures. Actually, this technique does not provide the feature of modularity and it increase the difficulty of modifying a large and complex software system.

While in an object-oriented system, every module is an object which can be modified without having to alter the code in other objects that access them. This is achieved "as long as the declaration of the public part of a class and the declaration of the member functions remain unchanged, the implementation of a class can be changed without affecting its users(42)". The term modularity refers to the factoring of a large program into units that can be modified independently and modularity is supported by object-oriented mechanism, especially for building complex software systems.

CHAPTER V

SUMMARY, CONCLUSION AND SUGGESTIONS

5.1 Summary and Conclusion

Object orientation means a unit of some private data and a set of operations that can access that data. An object is requested to perform one of its operations by sending it a message telling the object what to do. It responds to the message by executing the operation that implements the message, and then returns control to the caller. An object's data is private to it, and the only way to access that data is by requesting the object to do it.

Object-oriented programming is also an abstraction mechanism. A program manipulating an object uses certain defined operations to manipulate it. These operations serve as an interface, and the program does not need to know how the object implements the operations. At the same time, an object's behavior can be divided into several facets, which need not know each other's internal details. By using abstraction, large and complex systems are under control of modifying the existing codes.

Object-oriented programming language C++ is an extension of C language. It provides type checking, data abstraction, information hiding and an inheritance mechanism. By means of derived classes and virtual functions, C++ permits definition of a common calling interface for functions later defined in subclasses.

To implement a class, an English description in terms of informal strategy is constructed. The nouns of the English description are the private data variables and the verbs are the set of operations. Then private data and all useful operations become a class. The private data can only be manipulated by the set of operations and these operations also provide interfaces to the outside world. The outside world can access the class object only by sending message to the object. The implementation details are hidden from the outside world.

The term modularity refers to the factoring of a large program into units. In object-oriented system, every module is an object. When this LP package is implemented, analyzing and decomposing is the first step. Then, six major kinds of modules are constructed. They are input class, transform class, simplex class, revised simplex class and output class. In addition, some classes are constructed and used to demonstrate the power of objects reusable and interchangeable. These objects achieve the same function but with different implementations. They are input and input', output and output' and simplex and revised simplex objects.

This paper illustrates the following object-oriented implementation process:

1. The object-oriented design concepts are stated.

2. The object-oriented methodology is introduced.

3. A couple of software components are implemented by using the object-oriented methodology. The higher level components are composed of the low level ones. The low level components are the input object, the matrix object, the vector object, the transform object and, the output object. The higher level components are the simplex object and the revised simplex object.

4. The reusability of components is demonstrated. Although the algorithm of the revised simplex object is different from the simplex object, the same low level components are used to achieve the revised simplex algorithm.

5. The possibility of object interchange is demonstrated. Different versions of each of the modules used for input, output, and solution method can be invoked but the overall structure of the program remains the same.

6. A set of data manipulation objects are built to handle large volume of data which can not fit into the primary memory. By making use of the secondary memory to store the entire constraints matrix, the computation is achieved by retrieving parts of the matrix into primary memory in turn to facilitate the computation.

Finally, A set of evaluation criteria is set up and the

LP package is evaluated by these criteria, which are information hiding, data abstraction, dynamic binding, inheritance, reusability and interchangability.

5.2 Suggestions

One of the goals of object-oriented design is to establish reusable software components factory. The objects implemented in this application only allow a couple of combinations and choices. In order to make the package more versatile, more objects are suggested to be implemented.

The user interface has become fashionable. "User friendly" is an objective of today's software systems. Further efforts should be devoted to make this package more friendly and easy to use.

BIBLIOGRAPHY

(1) Agrawal, R. C., and Heady, E. O., Operations Research Methods for Argricultural Decisions. The Iowa State University Press, Iowa, 1972.

(2) Booch, G., Software Engineering with Ada. The Benjamin / Cummings Publishing, California, 1983.

(3) Brinch-Hansen, P., "Edison- A Multiprocessor Language." Software-Practice and Experience, 11, 4, (1981), 325-361.

(4) Cox, B. J., "The Object Oriented Pre-Compiler : Programming Smalltalk80 Methods in C Language." ACM Sigplan Notices, 18, 1, (1983), 15-22.

(5) Cox, B. J., "Message/Object Progromming: An Evolutionary Change in Programming Technology." IEEE Software, 1, 1, (1984), 50-61.

(6) Cox, B. J., "Object Oriented Programming: An Evolutionary Approach" Addison-Wesley, New York, 1986.

(7) Cox, B., and Hunt, B., "Objects, Icons, and Software-ICs." Byte, 11, 8, (1986), 161-176.

(8) Daehler T., and Patterson D., "A Small Taste of Smalltalk." Byte, 11, 8, (1986), 145-159.

(9) Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, New Jersey, 1963.

(10) Dewhurst, S. C., and Stark, K. T., "Out of The C World Comes C++." Computer Language, 4, 2, (1987), 73-78.

(11) Duff, C. B., "Designing An Efficient Language." Byte, 11, 8, (1986), 211-224.

(12) Fisher, D. A., "A Common Programming Language for Department of Defense-Background and Technical Requirements." Institute for Defense Analysis, Report p-1191, (June, 1976), 19-22.

(13) Fisher, A. J., "The Syntax of User-defined Dyadic Operators." Software-Practice and Experience, 12, 7, (1982), 623-625.

(14) Goldberg, A., and Robson, D., Smalltalk-80, The Language and Its Implementation. Addison-Wesley, New York, 1983.

(15) Goguen, J. A., and Mesguer, J., "Extensions and Foundations of Object-Oriented Programming." ACM Sigplan Notices, 21, 10, (1986), 153-162.

(16) Halbert, D. C., and O'Brien, P. D., "Using Types and Inheritance in Object-Oriented Programming." IEEE Software, 4, 4, (1987), 71-79.

(17) Krumme, D. W., "Comments on an Example for Procedure Parameters." ACM Sigplan Notices, 22, 6, (1987), 109-111.

(18) Jones, T. C., "Reusability in Programming: A Survey of the State of the Art." IEEE Trans. Software Engineering, 10, 5, (1984), 488-494.

(19) Liskov, B., and Guttag, J., Abstraction and Specification in Program Development. McGraw-Hill, New York, 1986.

(20) Locks, M. O., Practical Linear Progrramming with Computer Applications. Western Periodicals Company, California, 1974.

(21) MacLennan, B. J., "Values and Objects in Programming Languages." ACM Sigplan Notices, 17, 12, (1982), 70-79.

(22) Madsen, O. L., "Block Structure and Object-Oriented Languages." ACM Sigplan Notices, 21, 10, (1986), 133-142.

(23) McGregor, J. D., "Object-Oriented Programming with SCOOPS." Computer Language, 4, 7, (1987), 50-56.

(24) Meyer, B., "Reusability : The Case for Object Oriented Design." IEEE Software, 4, 2, (1987), 50-64.

(25) Nygaard, K., "Basic Concepts in Object Oriented Programming." ACM Sigplan Notices, 21, 10, (1986), 128-132.

(26) Nguyen, V., and Hailpern, B., "A Generalized Object Model" ACM Sigplan Notices, 21, 10, (1986), 78-87.

(27) Ossher, H. L., "A Mechanism for Specifying the Structure of Large, Layered, Object-Oriented Programs." ACM Sigplan Notices, 21, 10, (1986), 128-132.

(28) Pascoe, G. A., "Elements of Object-Oriented Programming." Byte, 11, 8, (1986), 139-144.

(29) Pountain, D., "Object-Oriented Forth." Byte, 11, 8, (1986), 227-233.

(30) Reference Manual for the Ada Programming Language. United States Department of Defense, 1980.

(31) Rentsch, T., "Object-Oriented Programming." ACM Sigplan Notices, 17, 9, (1982), 51-57.

(32) Schmucker, J. K., "MacApp : An Application Framework." Byte, 11, 8, (1986), 189-193.

(33) Schmucker, K. J., "Object-Oreinted languages For the Macintosh." Byte, 11, 8, (1986), 177-185.

(34) Shaw, M., "Abstraction Techniques in Modern Programming Languages," IEEE Software, 1, 4, (1984), 10-26.

(35) Shaw, M., and Wulf, W. A., "Toward Relaxing Assumptions in Languages and Their Implementations." ACM Sigplan Notices, 13, 3, (1980), 45-61.

(36) Steuer, R. E. Multiple Criteria Optimization: Theory, Computation, and Application. John Wiley & Sons, Inc., New York, 1986.

(37) Strom, R., "A Comparison of the Object-Oriented and Process Paradigms" ACM Sigplan Notices, 21, 10, (1986), 88-97.

(38) Stroustrup, B., "Adding Classes to the C Language: An Exercise in Language Evolution." Software-Practice and Experience, 13, 2, (1983), 139-161.

(39) Stroustrup, B., "Classes: An Abstract Data Type Facility for The C Language." ACM Sigplan Notices, 17, 1, (1982), 42-51.

(40) Stroustrup, B., "A set of C classes for co-routine style programming." Computer Science technical Report CSTR-90, Bell Telephone Laboratories, 1979.

(41) Stroustrup, B., "An Overview of C++." ACM Sigplan Notices, 21, 10, (1986), 7-18.

(42) Stroustrup, B., The C++ Programming Language. Addison-Wesley, Massachusetts, 1986.

(43) Tesler, L., "Programming Experiences." Byte, 11, 8, (1986), 195-206.

(44) Wegner, P., "Classification in Object-Oriented Systems." ACM Sigplan Notices, 21, 10, (1986), 173-182.

(45) Wiener, R., and Sincovec, R., Software Engineering with Modula-2 and Ada. John Wiley & Sons, Inc., New York, 1984.

(46) Wiener, R., " Object-Oriented Programming in C++ : A case study." ACM Sigplan Notices, 22, 6, (1987), 59-68

(47) Yourdan, E., Techniques of Program Structure and Design. Prentice-Hall, Inc., New Jersey, 1975.

(48) Zionts, S., Linear and Integer Programming. Prentice-Hall, Inc., New Jersey, 1974.

APPENDIX A

DESCRIPTION OF THE SIMPLEX METHOD AND THE REVISED SIMPLEX METHOD

A.1 Introduction

Many important problems are amenable to formulation as a linear programing problems and are solved by the simplex method.

The following is an objective linear program(LP):

$$\max\{c_1 x_1 + c_2 x_2 + \dots + c_n x_n = z\}$$

subject to the constraints:

$$\begin{aligned} a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n &<= b_1 \\ a_{21} x_1 + a_{22} x_2 + \dots + a_{2n} x_n &<= b_2 \\ &\vdots \\ a_{m1} x_1 + a_{m2} x_2 + \dots + a_{mn} x_n &<= b_m \\ x_1, x_2 \dots, x_n &>= 0 \end{aligned}$$

As seen, an LP has three parts: an objective function, main constraints, and nonnegativity restrictions. In addition, we call

$$c_1 x_1 + c_2 x_2 + \dots + c_n x_n$$

the objective function, where

the c_j	are objective function coefficients;
z	is the objective function value;
the x_i	are original or decision variable;
the b_j	are the right hand side(RHS) values;

the a matrix contains constraint coefficients.

A.2 The Simplex Method (36)

The calculations of the simplex method can be performed in tableaus.

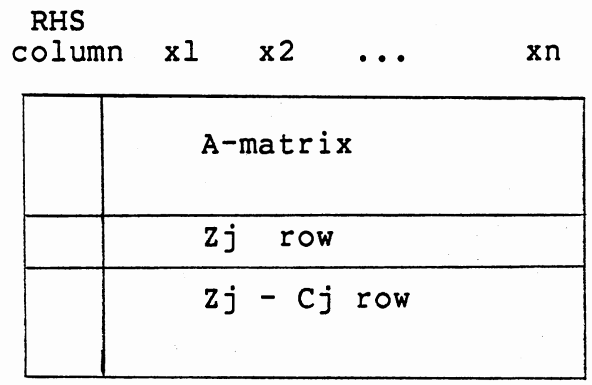


Figure 12. Organization of A Simplex Tableau

The simplex procedure is described as below: The simplex method iterates by moving from one extreme point to another in search of an optimal extreme point. At each step determine a variable to remove from the basis set and determine a variable to enter the basis set the computation is finished when the optimal solution reached, that is all element in Zj-Cj row are nonnegative.

1. Find a starting solution X.
2. Check to see if all the elements in Zj-Cj row are nonnegative. If they are, the solution is optimal. Otherwise, go to step 3.
3. Select the most negative Zj-Cj element as Xc to enter the basis which would increase the objective func-

tion.

4. Determine the removing variable Xr by considering only positive elements in the entering column of the A-matrix. Select the smallest ratio of

   RHS column / Xc column.

5. Let the a(r)(c) element denote the "pivot element" which is the intersection of the Xr row and the Xc column. For every element a(r)(j) in the pivot row, the new value is

   $$a'(r)(j) = a(r)(j) / a(r)(c).$$

6. Let a(i)(j) denote any element not in the pivot row, and a(i)(c) the element in the same row and the pivot column. The new value of a(i)(j) is

   $$a'(i)(j) = a(i)(j) - a(i)(c) * a'(r)(j)$$

   and go to step 2.

A.3 The Revised Simplex Method (20)

Rewrite the form of a linear programming problem as:

$$\begin{aligned} &\text{Max} \quad C'X \\ &\text{s.t.} \quad AX = b \\ &\qquad\quad X >= 0 \end{aligned}$$

If an extreme point is X', then the matrix A can be divided into (B,N), where B is the basis with an m x m full rank matrix comprised of the columns of coefficients of the non-zero variables, and N is the non-basis with an m x (n-m) matrix. And X can be decomposed into (Xb,Xn), AX = b can be written as BXb + NXn = b. Therefor, C'X = Cb'Xb + Cn'Xn = C'X' + (Cn' - Cb'B^{-1} N) Xn.

The procedure describes as follows:

1. Find a starting solution X with basis B.
2. If Cb'B^{-1} N - Cn' is nonnegative, the stop.
   Else pick the most negative component
   Cb'B^{-1} a(j) - c(j).
3. Let B^{-1} b = b' and pi = min { b(i)' / y(i)(j), y(i)(j) > 0 }, where y(i)(j) is the i_th component of y(j) = B^{-1} a(j).
4. Get the new extreme point X by calculating
   Xb(i) = b(i)' - pi * y(i)(j) for i=1,...,m
   X(j) = pi
   other X(i)'s are equal to zero.
5. Go to step 2.

The revised simplex method using the product form of the inverse is more economical in its demands for computer space, particularly core storage. It differs from the explicit form only in the form in which the inverse is recorded. The explicit form uses full matrices to record the inverse of the basis(B). While the product form only uses an elementary matrix(an array and a index) to record the inverse. An elementary matrix(E) is a nonsingular square matrix that is an identity matrix except for one column. Any inverse of an elementary matrix is an elementary matrix. Starting with an initial basis which is an identity matrix. We can redefine the problem after the first iteration so that the second basis is treated as an original basis for a new problem. Thus, the third basis(and its inverse) is an elementary matrix in terms of the second. In the same way, each basis is an elementary matrix in terms of the immediately preceding basis. Then successively left multiplying an identity matrix by the matrices

$$E_1, E_2, \ldots, E_m$$

we obtain B^{-1}. And a multiplication of $B^{-1}a(r)$ becomes

$$E_p E_{p-1} \ldots E_2 E_1 a(r)$$

where, p is the pth iteration.

A multiplication of $Cb'B^{-1}$ can be achieved by

$$Cb'E_1 E_2 \ldots E_p E_{p-1}$$

where, p is the pth iteration.

APPENDIX B

CATALOG OF OBJECTS

B.1 Class Vector Objects

B.1.1 Informal strategy

Vector is an one-dimentional array. A vector is an aggregate of elements of the same type. Each element is itself a data structure. All elements are of the same type and occupy the same amount of space. It allows us to relate a group of related variables as an unit. And any one element can be selected by specifying its position in the sequence with an integer subscript. The specification of a vector normally requires the name of the vector, and the range of permissible values for its subscripts. The elementary arithmetic operations of vectors are addition, subtraction, multiplication and division. Vectors are treated as a mathematical object for manipulation(20,36).

B.1.2 Objects(variables) included in a vector are:

name and size of the vector.

B.1.3 Interface operations

vector(int size=VARLIMIT);

Construct a vector object to hold the specified amount of double precision values. If size is not specified, the default size is assumed, which is defined by constant VARLIMIT.

~vector();

Destructor is a storage deallocator.

vector(vector& vec);

An allocator and initializor which creats and initializes a vector object. Input is a vector object. & is the address of the object.

int mostneg(int pos);

An operation which searches for the most negative value, and return the index of the value.

int smallest();

A function which searches for the smallest positive value and returns the index of that value. Negative values are not considered.

double min();

Return the minimum value of a vector object.

double max();

Return the maximum value of a vector object.

double operator[](int index);

Subscript operator which returns the content of the specified index of the vector. For example, vector a = (1,2,3,4), the value of a(0) is 1.

void operator=(double value);

Assign a double precision value to each element of the vector object.

void operator+=(vector& vec);

The vector operand preceding the += takes on the values resulting from adding the second vector to it.

```
void operator+=(double value);
```

The vector operand preceding the += takes on the values resulting from adding the scale operand to each element of the vector.

```
void operator-=(vector& vec);
```

The vector operand preceding the -= operator takes on the values resulting from subtracting the second vector from it.

```
void operator-=(double value);
```

The vector operand preceding the -= operator takes on the values resulting from subtracting the scale operand from it.

```
void operator*=(vector& vec);
```

The vector operand preceding the *= operator takes on the values resulting from the product of it and the second vector.

```
void operator*=(double value);
```

The vector operand preceding the *= operator takes on the values resulting from the product of it and the scale operand.

```
void operator/=(vector& vec);
```

The vector operand preceding the /= operator takes on the values resulting from the division of the second operand by the first.

void operator/=(double value);

The vector operand preceding the /= operator takes on the values resulting from the division of the vector operand by the scale operand.

vector operator+(vector& vec);

Return a vector which contains the sum of the two vector operands.

vector operator+(double value);

Return a vector which contains the sum of the vector operand and the scale value.

vector operator-(vector& vec);

Return a vector which contains the differences of the two vector operands.

vector operator-(double value);

Return a vector which contains the differences of the first vector operand and the scale value.

vector operator*(vector& vec);

Return a vector which contains the product of the two vector operands.

vector operator*(double value);

Return a vector which contains the product of the vector operand and the value.

```
vector operator/(vector& vec);
```

Return the quotient of the two vector operands. If any divisor is zero, a predefined infinite value is returned. For example, vector v1 contains (4,5,6,7), vector v2 contains (1,0,2,1) and vector result = v1 / v2; Vector result contains (4,INFINITE,3,7), where INFINITE is a predefined constant.

```
vector operator/(double value);
```

Return the quotient vector of the vector operand divided by the value. If the value is zero, an predefined infinite value is returned.

```
friend istream& operator>>(istream& s, vector& vec);
```

Input operator which gets input values from screen and stores the value in the vector vec.

```
friend ostream& operator<<(ostream& s, vector& vec);
```

Output operator which prints the values of the vector elements to the screen.

B.1.4 Examples

Example 1.

```
    vector A(10);  // size of vector A is 10
    vector B(10);
    vector result = A + B;
```

Example 2.

```
    vector A(4);    // Declare one vector object A.
    cin >> A;       // Vector A gets input from keyboard,
                    // say, it contains (5,1,1,3).
    double val = 2;
    vector result(4);
    result = A * 2;     // result contains (10, 2, 2, 6)
```

Example 3.

```
    vector a(3);
    vector b = a;
```

As an example, a = (2.3, 4.0, 5.9), vector b takes the values of vector a. Therefore vector b = (2.3, 4.0, 5.9).

Example 4.

```
   vector a(4);         // say, a contains  (4,4,4,4);
   vector b(4);         // b contains  (1,2,3,4);
   b += a;
```

b takes of the values of b plus a. Therefore b contains (5,6,7,8), and values in vector a do not change.

Example 5.

```
   vector a(3);                // say, a contains (2,3,4)
   cout << " vector a contains " << a;

   Output is:
                 vector a contains 2,3,4
```

B.2 Class Matrix Object

B.2.1 Informal strategy(20,36)

A matrix is a rectangular array of elements. Matrices are rectangular in the sense that their elements are arranged in rows and columns. This enables the size of a matrix to be described with two numbers, the number of rows and the number of columns. Two matrices are equal if and only if they are of same size and their corresponding elements are equal. Matrix addition (or subtraction) is performed by adding (or subtracting) corresponding elements in matrices of the same size. In scalar multiplication, each matrix element is multiplied by the scalar. Two matrices multiplication is only defined when the number of columns of the first matrix A (where A is m x r) equals the number of rows of the second matrix B (where B is r x n). Elements Cij in the product matrix C (where C is m x n) are computed by summing the products arrived at by multiplying the elements in the ith row of A by their corresponding elements in the jth column of B.

B.2.2 Objects(variables) included in a matrix are:

row size, column size, and name of the matrix.

B.2.3 Interface operations

```
matrix(int r=CONLIMIT, int c=CONLIMIT);
```

Matrix constructor which creates a matrix object.

The default matrix size is CONLIMIT by CONLIMIT, where CONLIMIT is a predefined constant.

~matrix();

Matrix destructor which releases the space allocated for a matrix object.

int nrow();

Return the number of rows of the matrix object.

int ncol();

Return the number of columns of the matrix object.

matrix& operator=();

Assignment operator which assigns the source matrix object to the target matrix object.

matrix& operator+=();

Addition and assignment operator which adds the second operand object to the first operand object and assigns the result to the first object operand.

matrix& operator-=();

Subtraction and assignment operator which subtracts the second object operand from the first object operand and assigns the result to the first object.

operator*=();

Multiplication and assignment operator which multiplies the first operand and the second operand and assigns the result to the first object.

vector operator*(vector&, matrix&);

Multiply a vector to a matrix, return a new vector object. The input vector parameter and matrix parameter remain unchanged.

vector operator*(matrix&, vector&);

Multiply a matrix to a vector. Return a new vector object. The input vector parameter and matrix parameter remain unchanged.

matrix append_i();

Append an identity matrix to the end of the matrix object.

void i_matrix();

Construct an identity matrix.

matrix invert();

Return the inverse matrix of the member matrix object.

istream& operator>>(istream&, matrix&)

Matrix input operator which get the values of a matrix from the keyboard.

ostream& operator<<(ostream&, matrix&)

Matrix output operator which prints matrix on the screen.

B.2.4 Examples:

Example 1.

```
    matrix A(2,2);            // say, A = 1,3
                                          2,4
    cout << " Matrix A contains " << A;

    Output is :
             Matrix A contains 1,3
                                2,4
```

Example 2.

```
    matrix A(2,2);     // A gets values from the keyboard,say
    cin >> A;          // A contains  5,1
                                      2,1
    matrix B(2,2);
    cin >> B;          // B gets values 3,3 from the keyboard
                                        2,4
    matrix result(2,2);
    result = A * B;    // result contains 15,3
                                           4,4
```

Example 3.

```
    matrix A(2,3);       // say, A contains 3,4,5
                                            1,2,2
    vector b(2);         // b contains 1,2
    matrix result = b * A;

  result contains  3,4,5
                   2,4,4
```

B.3 Classes Data Manipulation Objects

The data manipulation classes manipulates data input and output. The input class contains several lower level classes, they are get_name, get_conname, get_varname, and get_coef classes. The output classes are responsible for output the final solution.

B.3.1 Informal strategy

The input classes are implemented in two different versions, one gets data from the keyboard, and the other gets data from a file. The output classes are designed in two versions too. The first one outputs the final solution to the screen. The second one outputs the solution to a file.

B.3.2 Objects included in the input classes are:

variable name table, constraint name table, objective function vector, right hand side vector, constraint coefficient matrix, and types of the constraint inequality equations, that is, greater than or equal to, less than or equal to and equal to.

B.3.3 Interface operations

Interface of GET_NAME object :

```
get_name(char* str);
```

Get_name object constructor. Allocate an instance of get_name object.

~get_name();

Get_name object destructor which decallocates get_name object just before the program terminates.

int upbound();

Return the upper bound of the name table.

int lowbound();

Return the low bound of the name table.

char* operator[](int i);

Subscript operator which returns the name string of the specified subscript.

void promptfor(int i);

Prompt for variables names, constraints names from screen.

void list();

Print the whole list of name table on the screen.

void append();

Append new name to the end of name table. And the limit number of names is defined by MAXNAME; If the entered name exceed this limit number, the system will output "Out of Limit" message.

void del();

Delete the user specified name from the name table.

Interface of GET_COEF object:

get_coef();

Get_coef object(s) constructor.

~get_coef();

Get_coef object(s) destructor.

void askfor();

Prompt for the coefficients of the constraints, right hand side vector coefficients and objective function coefficients.

void mdfy_value();

It allows users to change the values which have been entered. Users can change values before running the program. Users also allow to change data values of an existing LP problem.

void condense();

It stores non-zero values of the input data.

void print();

Operation which prints the coefficients of constraints, right hand side, and objective function in a sparse representation to a file.

void readfile();

Read input data from a file. The data file should be of the MPSX format.

Interface of Second kind of GET_COEF object:

```
get_coef();
     Get_coef object(s) constructor.

~get_coef();
     Get_coef object(s) destructor.

void get_data();
     Get input data from a MPSX format input file.

void print();
     Operation which prints the coefficients of
     constraints, right hand side, and objective function
     in a sparse representation to a file.
```

example 1.

```
  get_name  student_list;
  student_list.askfor(); //interactive input student name
  student_list.list();   //list student name
  student_list.del();    //invoke delete routine to delete
                         //   the drop-off student
```

example 2.

```
  get_coef  input_object;       // first kind of get_coef

  input_object.get_data();      // get data from screen
  input_object.mdfy_value();    // modify values entered
```

B.3.4 Classes output objects

Objects(variables) included in a output class

A buffer for storing output data tempararily.

Interface Operations

output::output();

Output object constructor.

output::~output();

Output object destructor.

void output::display(tableau&, get_coef&)

Displaying the final result to the screen. Input parameters are simplex tableau object and get_coef object.

Interface Operations of the Second Output Class

output::output();

Output object constructor.

output::~output();

Output object destructor.

void output::display(tableau&, get_coef&)

Write the final solution to a file.

B.4 Class Transform Object

B.4.1 Informal strategy

Matrices or vectors where most of the elements are zero are called sparse. The constraints coefficient matrix could be sparse, and in order to save space only the non-zero elements were stored. While a linear programming problem is solved by using the simplex method in tableau format, after iterating some zero elements become non-zero and vice visa. Therefore, it is necessary to allocate a full matrix for constraints coefficients and full vectors for right hand side row and objective function coefficients column. Therefore, the role of the class transform is to transform sparse representation to full representation to facilitate the computation.

B.4.2 Objects included in a transform object are:

the constraint coefficients matrix, the objective function coefficient vector, the right hand side vector.

B.4.3 Interface operations

`transform(int row, int col);`

Transform object constructor which constructs spaces needed for constraint matrix, right hand side row, and objective function coefficients vector.

`~transform();`

Spaces deallocator which releases the spaces when the program is terminated.

matrix transform::processmx();

Transform the sparse representation to full representation of a matrix by putting the value into a matrix cell according to the row index and the column index.

vector transform::processrhs();

Transform the sparse representation of right hand side vector to full representation. Multiply the corresponding value of matrix elements and rhs vector by -1, if the right hand side value is negative.

vector transform::processobj();

Transform the sparse representation of objective function coefficients vector to full representation.

vector transform::print();

Print the full matrix and full vector of the transformed result which were sparce before.

4.4 Example:

```
transform   FORM;
FORM.processmx(); /* transform sparse matrix to full */
FORM.print();     /* print out the full matrix       */
```

B.5 Class The Simplex Method Object

See section 3.2.

B.6 Class The Revised Simplex Object

B.6.1 Informal strategy

The revised simplex method is a systematic procedure for implementing the steps of the simplex method. The detail description of the procedures of the revised simplex method is given in Appendix A. A summary can be concluded as :

Cb	is the basis variables cost vector.
b_l	is the inverse of basis matrix
N	is the non-basis matrix.
a(i)	is the ith column of non-basis matrix N.
Z	is the objective function coefficients vector.
C	is the cost vector.
b	is the right hand side coefficient vector.

```
1. Find a starting solution X with basis B.
2. Find variable to enters the basis. The index of
   the entering variable is "j";
   compute  Cb' * b_l and put the result in vector Y;
   compute Zj - Cj = Y * N - Cn';
   if(objective is maximize)
       choose most negative value of Zj - Cj;
   else  choose most positive value of Zj - Cj;

3. Find variable which leaves the basis;
   compute b_l * b and put the result in vector b';
```

compute b_l * a(j) and put the result in vector afa;
If each value in vector afa is non-positive, then
the solution is unbounded.
Otherwise, choose smallest non-negative ratio pi
of (b' / afa) as leaving variable.

4. compute the next basis matrix for next iteration.
   A multiplication of b_l * a(j) can be constructed
   by left-multiplying elementary matrix E(i) with a(in).
   Refer reference (48) for detail.
   b_l = Ep*Ep-l*...*Ei*...*E0* a(in)

B.6.2 Objects(variables) included are:

Integer array which contains index of basis variables(Xb), integer array which contains index of non-basis variables(Xn), basis cost vector(Cb), non-basis cost vector(Cn), Zj - Cj vector, the inverse matrix of basis matrix(b_l), and non-basis matrix(N).

B.6.3 Interface operations

revise(int ncon,int nact);

Revise object constructor. Ncon is number of constraints and nact is number of activity variables.

~revise();

Revise object destructor.

void reviseinit(transform&);

Initialize variables for the revised simplex method. Input parameter is the address of an transform object. Get full constraint matrix values, right hand side vector values and objective function values from transform object.

void solveinit();

Initialize the first iteration, that is, get values for matrix b_1 and basis matrix N, vector Cb and vector Cn. Find basic feasible solution.

void parameter_set();

Append big M to the cost vector for artificial variable. Get values of basis vector Cb and non-basis vector Cn. Get values of non-basis matrix N.

void ck_optimal(get_coef&);

Check whether optimal solution is reached. If solution is infeasible or unbounded output message to indicate the solution is infeasible or the solution is unbounded.

int ck_unbound(vector&);

Check if there an unbounded solution occurred. Input parameter is the address of a vector object.

void setup(get_coef&);

Change constraint inequalities to equality equations by appending an identity matrix to the constraint matrix. Input is the address of a get_coef object.

`void simplex(transform&,get_coef&);`

Revised simplex driver routine.

`double obj();`

Compute the objective function value.

`int entering();`

Find the entering variable which enters the basis set for next iteration.

`int leaving();`

Find the leaving variable which leaves the basis variables.

`void cal_e();`

Any basis matrix or its inverse may be represented as a product of elementary matrix. Cal_e operation computes the elementary matrix of each iteration and puts E(i) on a elementary matrix array for computing inverse of basis matrix. See reference(45) for detail.

`void basis_set();`

Switch basis and non-basis variable indices positions when non-basis becomes basis and basis becomes non-basis.

`void cb_set();`

Switch costs value of basis variable and non-basis variable when they switch the status.

APPENDIX C

C++ PROGRAM LISTING

```
/*******************************************************
 *          Global Constants File
 *******************************************************/
#ifndef  NULL
#define  NULL               0
#endif
#define  BIGM               1.0E+09
#define  MINUSBIGM         -1.0E+09
#define  PINFINATE          1.0E+30
#define  NINFINATE         -1.0E+30
#define  SMALL              1.0E-15
#define  TOLERENCE          1.0E-10
#define  MINUSTOLERENCE    -1.0E-10

#define  CONLIMIT           1000    // constraint limit
#define  VARLIMIT           1000    // variable limit
#define  MARKZERO          -1
#define  TRUE               1
#define  FALSE              0

#ifndef  YES
#define  YES                1
#endif

#ifndef  NO
#define  NO                 0
#endif


extern int numcon;
extern int numrealact;
extern int numact;
extern int numlessthan;
extern int numequalto;
extern int numgreaterthan;

extern char probname[50];
extern char objective[50];
extern int  maximize;
```

```
//*******************************************************
//                 Matrix object header file
//*******************************************************
#include "vector.hxx"

struct matrix{
    int row;             // row size
    int col;             // column size
    double** m;          // pointer to 1st element of row
public:
    matrix(int r=10,int c=10);
    ~matrix();
    int nrow();
    int ncol();
    double* operator[](int);
    vector  operator()(int); // return a column of matrix
    matrix& operator=(double);
    matrix& operator=(matrix&);
    void i_matrix();
    void operator+=(matrix&);
    void operator-=(matrix&);
    matrix operator*=(matrix&);
    matrix append_i();
    int lu_fact();
    int inv_subs();
    int invert(matrix&);
    friend matrix operator+(matrix&);
    friend matrix operator-(matrix&);
    friend matrix operator*(matrix&,matrix&);
    friend vector operator*(matrix&,vector&);
    friend vector operator*(vector&,matrix&);
    friend istream& operator>>(istream&, matrix&);
    friend ostream& operator<<(ostream&, matrix&);
};

struct ele_matrix{
    int size;
    int index;
    double *vec;
public:
    ele_matrix(int,int);
    ~ele_matrix();
    friend void operator*(vector&, ele_matrix&);
    friend void operator*(ele_matrix&, vector&);
};
```

```
//*******************************************************
//              Input object header file
//*******************************************************
#include "matrix.hxx"
#include "attr.hxx"
#include "crtwnd.hxx"
#define  MAXCELL  100
/*
-------------------------------------------------------
 Interface of get_name class.  It is a base class of
 get_conname and get_varname.
-------------------------------------------------------
*/
class get_name{
public:
    int lownum;             // low bound for variable name
    int upnum;              // up bound for variable name
    char tname[30];         // name type tag
    char** name;            // pointer variable point to
                            // array of names
    get_name(char*);
    ~get_name(){  delete[VARLIMIT] name;}
    int   upbound() { return upnum; }
    int   lowbound(){ return lownum;}
    char* operator[](int i) { return *(name+i); }
    void promptfor(int);
    void list();
    void append();
    void del();
    void modify();
};
/*
-------------------------------------------------------
 class get_varname is a derived class of get_name.
-------------------------------------------------------
*/
class get_varname : public get_name{
public:
      get_name  m;
};
/*
-------------------------------------------------------
 class get_conname is a derived class of get_name.
-------------------------------------------------------
*/
class get_conname : public get_name{
public:
      get_name  m;
};
```

```
/*
---------------------------------------------------------
 Interface of get_cons_coef class.
---------------------------------------------------------
*/
struct rec{
    int     index;
    double  val;
};
struct rec2{
    int  begin;
    int  end;
};
struct ccell{   // character cell
    int  row;
    int  col;
    char ch;
};
struct numcell{  // number cell
    int  row;
    int  col;
    double val;
};
class get_cons_coef{
    rec2*  col;
    rec*   row;
    rec*   rhs;
    rec*   initcost;
    ccell* contyp;
    numcell*  pmtr;
    numcell*  prhs;
    numcell*  pcost;
    friend class transform;
public:
    char contype[CONLIMIT];
    get_name*  cnameptr;
    get_name*  vnameptr;
    int dim1;
    int dim2;
    get_cons_coef();
    ~get_cons_coef();
    void askfor();
    void mdfy_val();
    void condense();
    void readfile();
    void print();
};
```

```
//*********************************************************
//          Class transform object header file
//*********************************************************
#include "testio.hxx"

class transform{
   char*  contype;             // array of constraint type
   matrix *B;                  // target matrix
   vector *R;                  // target rhs vector
   vector *OBJ;                // target obj. coef. vector
   friend class tableau;
   friend class revise;
public:
   transform(int,int);
   ~transform();
   void processmx(get_cons_coef&);
   void processrhs(get_cons_coef&);
   void processobj(get_cons_coef&);
   void output();
};
```

```
//***************************************************************
//           Class simplex method object header file
//***************************************************************
#include "testform.hxx"

class tableau{
    char  objname[30];        // name of the obj. function
    char  rhsname[30];        // name of the r. h. s.
    int   numartvar;          // total # of arti.activity
    double  objlevel;         // val of the obj. function

    int   *basisno;           // array of basis var. ind #
    int   *basis;             // array of variables indices
    int   *bigmind;           // artifical variables indice

    vector  *cost;            // cost vector
    vector  *orhs;            // original rhs of tableau
    vector  *rhs;             // right hand side of tableau
    vector  *ratio;           // ratio of rhs/enter column
    vector  *z;               // reduced cost (Zj) row
    vector  *shadow;          // shadow price (Cj-Zj) row
    vector  *actlevel;
    vector  *pivotrow;        // row vector of pivot element

    int     bcount;           // count for basis
    char    *contype;         // array of constraint type
    int     finalrow[30];
    int     finalcol[30];
    int     itnum;            // iteration number
    int     quit;
    int     out_row;          // leaving variable index
    int     in_col;           // entering variable index
    double  pivot;            // pivot element

    matrix  *tm;              // temp matrix
    matrix  *mx;              // constraint coef. matrix
    friend class output;
public:
    tableau(int,int);
    ~tableau();
    void tableauinit(transform&);
    void setup(get_cons_coef&);
    void solveinit();
    void  ck_optimal(get_cons_coef&);
    int  entering();
    int  leaving();
    void pivot_set();
    void row_divide();
    void row_eliminate();
    int  smallest();
    void computeshadowprice();
    double obj();
    double setp(double);
    double setz(double);
    void tab_output();
    void final_output(get_cons_coef&);
    void simplex(transform&, get_cons_coef&);
};
```

```
//*********************************************************
//                Class revise object header file.
//*********************************************************
#include  "testform.hxx"

class revise{
    int   numartvar;          // total # of artifical
    int   numnonartvar;       // total # of non-artifical
    double  objlevel;         // val of the objective func
    int   *basisno;           // array of basis index
    int   *basis;             // array of  varibles
    int   *nonbasis;          // array of non-basis index
    int   *bigmind;           // array of artifical index

    vector  *cost;            // cost vector
    vector  *ocost;           // original cost vector
    vector  *rhs;             // original rhsof tableau
    vector  *afa;
    vector  *xxb;
    vector  *shadow;          // shadow price (Cj-Zj) row

    vector  *Cb;              // cost vector for basis var.
    vector  *Cn;              // cost vector for non-basis
    vector  *p;               // column vector of enter var.
    vector  *y;
    int     bcount;           // count for basis
    char    *contype;         // constraint type vector
    int     itnum;            // iteration number
    int     quit;
    int     in_col;           // index of entering variable
    int     out_row;          // index of leaving variable
    double  pivot;

    matrix  *mx;
    matrix  *A;             // non-basis matrix
    ele_matrix  **E;        // E matrix
    friend class output;
public:
    revise(int,int);
    ~revise();
    void reviseinit(transform&);
    void solveinit(get_cons_coef&);
    void parameter_set();
    void p_set();
    void ck_optimal(get_cons_coef&);
    int  ck_unbound();
    void setup(get_cons_coef&);
    void simplex(transform&,get_cons_coef&);
    double obj();
    void tab_output();
    void final_output(get_cons_coef&);
    int entering(get_cons_coef&);
    int leaving(get_cons_coef&);
    void basis_set();
    void cb_set();
    void cal_e();
    int  smallest(vector&);
};
```

```
//*********************************************************
//            Vector operation implementaton file
//*********************************************************
#include "vector.hxx"
/*
----------------------------------------------------------
 vector constructor --  allocating and initialization
----------------------------------------------------------
*/
vector::vector(int sz)
{
    if(sz < Ø){
        printf("negative vector size\n");
        exit(1);
    }
    size = sz;
    v = new double[size];
    if( v == NULL ){
        printf("insufficient memory\n");
        exit(1);
    }
    for(int i=Ø; i < size; i++)  // empty vector all Ø's
        v[i] =Ø.Ø;
}
/*
----------------------------------------------------------
 vector constructor -- second form of vector constructor.
----------------------------------------------------------
*/
vector::vector(vector& a)
{
    v = new double[size=a.size];
    if( v == NULL ){
        printf("2*****:insufficient memory\n");
        exit(1);
    }
    for(int i=Ø; i < size; i++){
        double temp = a[i];
        if(fabs(temp) <= TOLERENCE) temp = Ø.Ø;
        v[i] = temp;
    }
}
/*
----------------------------------------------------------
 vector destructor --  destroying objects.
----------------------------------------------------------
*/
vector::~vector()
{
       delete v;
}
```

```
/*
---------------------------------------------------------
 sizevec -- returning size of vector object.
---------------------------------------------------------
*/
int vector::sizevec()
{
     return(size);
}
/*
---------------------------------------------------------
 [] -- subscript operator, return value of that subscript
---------------------------------------------------------
*/
double vector::operator[](int c)
{
     return( *(v+c) );
}
/*
---------------------------------------------------------
 = assignment operator, return vector of passed in value.
---------------------------------------------------------
*/
void vector::operator=(double val)
{
    if(fabs(val) <= TOLERENCE) val = 0.0;
    for(int i=0; i < size; i++){
        v[i] = val;
    }
}
/*
---------------------------------------------------------
 += -- addition and assignment operator.
---------------------------------------------------------
*/
void vector::operator+=(vector& v1)
{
    int sz = v1.sizevec();
    if( sz != size){
        printf(" different size vector can't add\n");
        exit(1);
    }
    for(int i=0; i < size; i++){
        double temp = v1[i];
        if(fabs(temp) <= TOLERENCE) temp = 0.0;
        v[i] += temp;
 //       v[i] += v1[i];
    }
}
```

```
/*
----------------------------------------------------------
 -= -- subtraction and assignment operator.
----------------------------------------------------------
*/
void vector::operator-=(vector& v1)
{
    int sz = v1.sizevec();
    if(sz != size){
        printf("Different size vector can't subtract\n");
        exit(1);
    }
    for(int i=0; i < size; i++){
        double temp = v1[i];
        if(fabs(temp) <= TOLERENCE) temp = 0.0;
        v[i] -= temp;
    }
}
/*
----------------------------------------------------------
 *= -- multiplication and assignment operator.
----------------------------------------------------------
*/
void vector::operator*=(vector& v1)
{
    int sz = v1.sizevec();
    if(sz != size){
        printf("Different size vector can't multiple.\n");
        exit(1);
    }
    for(int i=0; i < size; i++){
        double temp = v1[i];
        if(fabs(temp) <= TOLERENCE) temp = 0.0;
        v[i] *= temp;
    }
}
/*
----------------------------------------------------------
 /= -- division and assignment operator.
----------------------------------------------------------
*/
void vector::operator/=(vector& v1)
{
    int sz = v1.sizevec();
    if(sz != size){
        printf(" Different size vector can't divide.\n");
        exit(1);
    }
    for(int i=0; i < size; i++){
       if(fabs(v1[i]) <= TOLERENCE || v1[i] == 0.0)
           v[i] = PINFINATE;
       else v[i] /= v1[i];
    }
}
/*
----------------------------------------------------------
 + -- addition operator.
----------------------------------------------------------
*/
vector vector::operator+(vector& v1)
{
    vector v3(size);
    int sz = v1.sizevec();
    if(sz != size){
        printf(" different size vector can't add\n");
        exit(1);
    }
    for(int i=0; i < size; i++){
        double temp = v1[i];
        double temp2= v[i];
        if(fabs(temp) <= TOLERENCE) temp = 0.0;
        if(fabs(temp2)<= TOLERENCE) temp2 =0.0;
        v3.v[i] = temp + temp2;
    }
    return v3;
}
/*
----------------------------------------------------------
 - -- substraction operator.
----------------------------------------------------------
*/
vector vector::operator-(vector& v1)
{
    vector v3(size);
    int sz = v1.sizevec();
    if(sz != size){
        printf("different size vector can't subtract\n");
        exit(1);
    }
    for(int i=0; i < size; i++){
        double temp = v1[i];
        double temp2= v[i];
        if(fabs(temp) <= TOLERENCE) temp = 0.0;
        if(fabs(temp2)<= TOLERENCE) temp2 =0.0;
        v3.v[i] = temp2 - temp;
    }
    return v3;
}
```

```
/*
----------------------------------------------------------
 * -- multiplication operator.
----------------------------------------------------------
*/
vector vector::operator*(vector& v1)
{
    vector v3(size);
    int sz = v1.sizevec();
    if(sz != size){
        printf(" different size vector can't multiply\n");
        exit(1);
    }
    for(int i=0; i < size; i++){
        double temp1 = v1[i];
        double temp2 = v[i];
        if(fabs(temp1) <= TOLERENCE) temp1 = 0.0;
        if(fabs(temp2) <= TOLERENCE) temp2 = 0.0;
        v3.v[i] = temp1 * temp2 ;
    }
    return v3;
}
/*
----------------------------------------------------------
 / -- division operator.
----------------------------------------------------------
*/
vector vector::operator/(vector& v1)
{
    vector v3(size);
    int sz = v1.sizevec();
    if(sz != size){
        printf(" different size vector can't divide.\n");
        exit(1);
    }
    for(int i=0; i < size; i++)
        if( fabs(v1[i]) <= TOLERENCE ){
            v3.v[i] = PINFINATE;
        }
        else{
            if(fabs(v[i]) <= TOLERENCE) v3.v[i] = 0.0;
            else    v3.v[i] = v[i] / v1[i];
        }
    return v3;
}
```

```
/*
-----------------------------------------------------------
 + -- addition operator, adding a constrant vector.
-----------------------------------------------------------
*/
vector vector::operator+(double val)
{
    vector v3(size);

    for(int i=Ø; i < size; i++){
        double temp = val;
        double temp2= v[i];
        if(fabs(temp) <= TOLERENCE) temp = Ø.Ø;
        if(fabs(temp2)<= TOLERENCE) temp2 =Ø.Ø;
        v3.v[i] = temp2 + temp;
 //       v3.v[i] = val + v[i] ;
    }
    return v3;
}
/*
-----------------------------------------------------------
 - -- substraction operator.
-----------------------------------------------------------
*/
vector vector::operator-(double val)
{
    vector v3(size);

    for(int i=Ø; i < size; i++){
        double temp = val;
        double temp2= v[i];
        if(fabs(temp) <= TOLERENCE) temp = Ø.Ø;
        if(fabs(temp2)<= TOLERENCE) temp2 =Ø.Ø;
        v3.v[i] = temp2 - temp;
    }
    return v3;
}
/*
-----------------------------------------------------------
 * -- multiplication operator.
-----------------------------------------------------------
*/
vector vector::operator*(double val)
{
    vector v3(size);

    for(int i=Ø; i < size; i++){
        double temp = val;
        double temp2= v[i];
        if(fabs(temp) <= TOLERENCE) temp = 0.0;
        if(fabs(temp2)<= TOLERENCE) temp2 =0.0;
        v3.v[i] = temp2 * temp;
    }
    return v3;
}
/*
----------------------------------------------------------
 / -- division operator.
----------------------------------------------------------
*/
vector vector::operator/(double val)
{
    if(val==0.0){
        printf("dividor element is zero\n");
        exit(1);
    }
    vector v3(size);
    for(int i=0; i < size; i++){
       double temp = v[i];
       if(fabs(temp) <= TOLERENCE) temp = 0.0;
        v3.v[i] = temp / val;
    }
    return v3;
}
/*
----------------------------------------------------------
 += -- addition and assignment operator.
----------------------------------------------------------
*/
void vector::operator+=(double val)
{
    for(int i=0; i < size; i++)
        v[i] += val;
}
/*
----------------------------------------------------------
 -= -- substraction and assignment operator.
----------------------------------------------------------
*/
void vector::operator-=(double val)
{
    for(int i=0; i < size; i++)
        v[i] -= val;
}
```

```
/*
----------------------------------------------------------
 *= -- multiplication and assignment operator.
----------------------------------------------------------
*/
void vector::operator*=(double val)
{
    for(int i=0; i < size; i++)
        v[i] *= val;
}
/*
----------------------------------------------------------
 /= division and assignment operator
----------------------------------------------------------
*/
void vector::operator/=(double val)
{
    for(int i=0; i < size; i++)
        if(val==0.0){
            printf( "divided by 0\n");
            exit(1);
        }
        else   v[i] /= val;
}
/*
----------------------------------------------------------
 sum -- return the value of a vector object.
----------------------------------------------------------
*/
double vector::sum()
{
    double result = 0.0;
    for(int i=0; i < size; i++){
        double temp = v[i];
        if(fabs(temp) < TOLERENCE) temp = 0.0;
        result += temp;
    }
    return result;
}
/*
----------------------------------------------------------
 most -- return most negative value/ most positive value.
----------------------------------------------------------
*/
int vector::most(int p)
{
    double mostval,temp;
    int    index;
    if(p){          // mostnegative
        mostval = BIGM;
        index   = MARKZERO;
        for(int i=Ø; i < size; i++){
            temp = v[i];
            if(fabs(temp) < TOLERENCE) temp = Ø.Ø;
            if( temp < Ø.Ø && temp < mostval ){
                mostval = temp;
                index = i;
            }
         }//for
    }//if
    else{    // mostpositive
        mostval = MINUSBIGM;
        index   = MARKZERO;
        for(int i=Ø; i < size; i++)
            if( v[i] > Ø.Ø && v[i] >= mostval){
                mostval = v[i];
                index = i;
            }
    } //else
    return(index);
}
/*
----------------------------------------------------------
 least -- return smallest value.
----------------------------------------------------------
*/
int vector::least(int p)
{
    double leastval;
    int    index;
    if(p){          // leastnegative
        leastval = MINUSBIGM;
        index   = MARKZERO;
        for(int i=Ø; i < size; i++)
            if( v[i] <= Ø.Ø && v[i] >= leastval ){
                leastval = v[i];
                index = i;
            }
    }//if
    else{    /* smallest  */
        leastval = BIGM;
        index   = MARKZERO;
        for(int i=Ø; i < size; i++){
           if( v[i] >= Ø.Ø && v[i] < leastval){
                leastval = v[i];
                index = i;
            }
        }
    } //else
    return(index);
}
/*
----------------------------------------------------------
 >> input operator for vector object.
----------------------------------------------------------
*/
istream& operator>>(istream& s, vector& x)
{
    int sz = x.sizevec();
    for(int i=0; i < sz; i++){
         cout << "\n number: ";
        s >> x.v[i];
         cout << "echo: " << x.v[i];
    }
    return s;
}
/*
----------------------------------------------------------
 << -- output operation for vector object.
----------------------------------------------------------
*/
ostream& operator<<(ostream& s, vector& x)
{
    int sz = x.sizevec();
    for(int i=0; i < sz; i++){
        cout <<  x.v[i] << ", ";
    }
    return s;
}
```

```
//******************************************************
//          Matrix operations implementation file
//******************************************************
#include "matrix.hxx"
/*
--------------------------------------------------------
 matrix constructor -- constructing spaces for matrix.
--------------------------------------------------------
*/
matrix::matrix(int r, int c)
{
    row = r;
    col = c;
    m = new double*[row];
    for(int i=0; i < row; i++)
        m[i] = new double[col];

    // initialize matrix contains all 0's
    for(i=0; i < row; i++)
         for(int j=0; j < col; j++)
             m[i][j] = 0.0;
}
/*
--------------------------------------------------------
 matrix destructor -- destroying matrix object.
--------------------------------------------------------
*/
matrix::~matrix()
{
    for(int i=0; i < row; i++)
        delete m[i];
    delete[row] m;
}
/*
--------------------------------------------------------
 * -- multiplication operator of a matrix and a vector.
--------------------------------------------------------
*/
vector operator*(matrix& mm, vector& vv)
{

    int sz = vv.sizevec();
    if(sz != mm.row){
        printf("\r\nConformability Error.");
        kbwait(1000);
        exit(1);
    }
    vector r(sz);
    for(int i=0; i < mm.row; i++){
        r.v[i] = Ø.Ø;
        for(int j=Ø; j < sz; j++)
            if(mm[i][j] !=Ø.Ø && vv[j] !=Ø.Ø)
                r.v[i] += mm[i][j] * vv[j];
    }
    return r;
}
/*
----------------------------------------------------------
 * -- multiplication operator of a vector and a matrix.
----------------------------------------------------------
*/
vector operator*(vector& vv, matrix& mm)
{
    double temp;
    int sz = vv.sizevec();
    if(sz != mm.row){
        printf("\r\nConformability Error.");
        kbwait(1ØØØ);
        exit(1);
    }
    vector r(mm.col);
    for(int j=Ø; j < mm.col; j++){
        r.v[j] = Ø.Ø;
        for(int i=Ø; i < sz; i++)
            if( mm[i][j] !=Ø.Ø && vv[i] !=Ø.Ø){
                temp = vv[i] * mm[i][j];
                if(fabs(temp) < TOLERENCE) temp = Ø.Ø;
                r.v[j] += temp;
            }
    }//for
    return r;
}
/*
----------------------------------------------------------
 * -- multiplication operator.
----------------------------------------------------------
*/
matrix operator*(matrix& m1, matrix& m2)
{
    double temp;
    int r1 = m1.nrow();
    int c1 = m1.ncol();
    int r2 = m2.nrow();
    int c2 = m2.ncol();

    if(r1 != c2 || c1 != r2){
        printf("\r\nError!! Conformability Error.");
        kbwait(1ØØØ);
        exit(1);
    }
    matrix m3(r1,c2);
    for(int i=Ø; i < m3.row; i++)
        for(int j=Ø; j < m3.col; j++)
            for(int k=Ø; k < m3.col; k++)
                if( m1[i][k] !=Ø.Ø && m2[k][j] != Ø.Ø){
                    temp = m1[i][k] * m2[k][j];
                    if(fabs(temp) < TOLERENCE) temp = Ø.Ø;
                    m3.m[i][j] = m3[i][j] + temp;
                }
    return m3;
}
/*
----------------------------------------------------------
 nrow -- return number of rows of a matrix.
----------------------------------------------------------
*/
int matrix::nrow()
{
    return(row);
}
/*
----------------------------------------------------------
 ncol -- return number of columns of a matrix.
----------------------------------------------------------
*/
int matrix::ncol()
{
    return(col);
}
/*
----------------------------------------------------------
 [] -- return value of a column subscript.
----------------------------------------------------------
*/
double* matrix::operator[](int i)
{
    return(*(m + i));
}
/*
----------------------------------------------------------
 == -- assignment operator.
----------------------------------------------------------
*/
matrix& matrix::operator=(double val)
{
    for(int i=Ø; i < row; i++)
        delete m[i];
    delete[row] m;
    m = new double*[row];
    for(i=Ø; i < row; i++)
        m[i] = new double[col];

    for(i=Ø; i < row; i++)
        for(int j=Ø; j < col; j++)
            m[i][j] = val;
    return *this;
}
/*
----------------------------------------------------------
 = -- 2nd form of assignment operator.
----------------------------------------------------------
*/
matrix& matrix::operator=(matrix& m1)
{
    for(int i=Ø; i < row; i++)
        delete m[i];
    delete[row] m;

    m = new double*[row];
    for(i=Ø; i < row; i++)
        m[i] = new double[col];

    for(i=Ø; i < row; i++)
        for(int j=Ø; j < col; j++)
            m[i][j] = m1[i][j];
    return *this;
}
/*
----------------------------------------------------------
 i_matrix -- return an identity matrix object.
----------------------------------------------------------
*/
void  matrix::i_matrix()
{
    if( row != col ){
       printf("\r\n Error!! not square matrix.");
        kbwait(1ØØØ);
        exit(1);
    }
    for(int i=Ø; i < row; i++)
        for(int j=Ø; j < col; j++)
            if(i==j)    m[i][j] = 1.Ø;
            else    m[i][j] = Ø.Ø;
}
```

```
/*
---------------------------------------------------------
 () -- return a certain column of a matrix.
---------------------------------------------------------
*/
vector matrix::operator()(int col)
{
    vector v1(row);

    for(int i=0; i < row; i++)
        v1.v[i] = m[i][col];
    return v1;
}
/*
---------------------------------------------------------
 += -- addition and assignment operator of two matrix.
---------------------------------------------------------
*/
void matrix::operator+=(matrix& m1)
{
    int r = m1.nrow();
    int c = m1.ncol();
    if( row != r || col != c ){
        printf("\r\n Error!! matrix += operation.\n");
        exit(1);
    }
    for(int i=0; i < row; i++)
        for(int j=0; j < col; j++)
            m[i][j] += m1[i][j];
}
/*
---------------------------------------------------------
 -= -- subtraction and assignment operator
---------------------------------------------------------
*/
void matrix::operator-=(matrix& m1)
{
    int r = m1.nrow();
    int c = m1.ncol();
    if( row != r || col != c ){
        printf("\r\n Error!! matrix -= operation.\n");
        exit(1);
    }
    for(int i=0; i < row; i++)
        for(int j=0; j < col; j++)
            m[i][j] -= m1[i][j];
}
```

```
/*
----------------------------------------------------------
 *= -- multiplcation and assignment operator.
----------------------------------------------------------
*/
matrix  matrix::operator*=(matrix& m1)
{
    int r = m1.nrow();
    int c = m1.ncol();
    if(row != c || col != r){
        printf("\r\n Error!! matrix *= operation.");
        exit(1);
    }
    double sum,temp;
    matrix rm(row,col);
    for(int i=Ø; i < row; i++)
        for(int j=Ø; j < col; j++){
            sum =Ø.Ø;
            for(int k=Ø; k < col; k++){
                temp = m[i][k] * m1[k][j];
                if(fabs(temp) < TOLERENCE) temp = Ø.Ø;
                rm.m[i][j] = sum + temp;
            }
         // can't use m instead of rm.  Important!!
        }
    return rm;
}
/*
----------------------------------------------------------
 + -- addition operator.
----------------------------------------------------------
*/
matrix operator+(matrix& m1,matrix& m2)
{
    int r1 = m1.nrow();
    int c1 = m1.ncol();
    int r2 = m2.nrow();
    int c2 = m2.ncol();
    if( r1 != r2 || c1 != c2 ){
        printf("\r\n Error!! matrix + operation.");
        exit(1);
    }
    matrix m3(r1,c1);
    for(int i=Ø; i < r1; i++)
        for(int j=Ø; j < c1; j++)
            m3.m[i][j] = m1[i][j] + m2[i][j];
    return m3;
}
```

```
/*
----------------------------------------------------------
 - -- substraction operator.
----------------------------------------------------------
*/
matrix operator-(matrix& m1,matrix& m2)
{
    int r1 = m1.nrow();
    int c1 = m1.ncol();
    int r2 = m2.nrow();
    int c2 = m2.ncol();
    if( r1 != r2 || c1 != c2 ){
        printf("\r\n Error!! matrix - operation.");
        exit(1);
    }
    matrix m3(r1,c1);
    for(int i=Ø; i < r1; i++)
        for(int j=Ø; j < c1; j++)
            m3.m[i][j] = m1[i][j] - m2[i][j];
    return m3;
}
/*
----------------------------------------------------------
 append_i -- appending an identity matrix to a matrix.
----------------------------------------------------------
*/
matrix  matrix::append_i()
{
    int r = row;
    int c = row + col;
    matrix x3(r,c);
    for(int i=Ø; i < r; i++)
        for(int j=Ø; j < c; j++){
            if(j < col)
                x3.m[i][j] = m[i][j];
            else if(j == r + i)
                x3.m[i][j] = 1.Ø;
            else x3.m[i][j] = Ø.Ø;
        }
    return x3;
}
int matrix::lu_fact()
{
    // initialize
    for(int i=Ø; i < row; i++){
        pivot[i] = i;

        // Maximun value of each row
```

```
  double rmax = fabs(m[i][0]);
    for(int j=1; j < row; j++)
        if((temp =fabs(m[i][j])) > rmax )
            rmax = temp;
    if(rmax < SMALL) return 1;  // Row is all zero
    rowmax[i] = rmax;
}

for(int l=0; l < row; l++)
   cout << pivot[l] << ", ";
cout << "\n rowmax is \n";
for(l=0; l < row; l++)
   cout << rowmax[l] << ", ";
cout << "\n";

// Gauss elimination with scaled partial pivoting
if(row <= 1) return 0;

for(int k=0; k < row-1; k++){
    double temp;
    double colmax = -1.0;
    maxc = k;
    for(int i=k; i < row; i++){
        int ip = pivot[i];
        // temp = fabs((double) n);
        if((temp=fabs(m[ip][k]/rowmax[ip])) >colmax){
            colmax = temp;
            maxc = i;
        } //if
    }
    if(colmax < SMALL) return 1;
    piv_elem = pivot[maxc];
    pivot[maxc] = pivot[k];
    pivot[k] = piv_elem;

    for(l =0; l < row; l++)
    cout << pivot[l] << ", ";

   // Adjust upper diagonal only
    for(i=k+1; i < row; i++){
        int ip = pivot[i];
        double ratio = -(m[ip][k] / m[piv_elem][k]);
        m[ip][k] = ratio;
        for( int j=k+1; j < row; j++)
            m[ip][j] += ratio * m[piv_elem][j];
        for(l=0; l < row; l++){
            cout << "\n";
            for(int t=0; t < col; t++)
```

```
                cout << m[l][t] << ", ";
            }
        }
    }
    if((temp=fabs(m[piv_elem][row-1])) < SMALL)
        return 2;
    else
        return Ø;
}
----------------------------------------------------------
  inv_subs -- return inverse matrix of a matrix object.
----------------------------------------------------------
int matrix::inv_subs()
{
    matrix m3(row,col);

    for(int j=Ø; j < row; j++)
        inv_pivot[pivot[j]] = j;
    for(int ione=Ø; ione < row; ione++){
        for(int i=Ø; i < inv_pivot[ione]; i++)
            m3.m[i][ione] = Ø.Ø;
        for(i=inv_pivot[ione]; i < row; i++){
            int ip;
            sum = ((ip = pivot[i]) == ione) ? 1.Ø : Ø.Ø;
            for(j=Ø; j<i; j++)
                sum -= m[ip][j] * m3[j][ione];
            m3.m[i][ione] = sum;
         }
         cout << "\n** 2 matrix m3:\n" << m3;
         for(i = row-1; i >= Ø; i--){
             sum = m3[i][ione];
             int ip = pivot[i];
             for( j=i+1; j <row; j++)
                 sum -= m[ip][j] * m3[j][ione];
             if(fabs(m3.m[i][ione]=sum / m[ip][i])<SMALL)
                 m3.m[i][ione] = Ø.Ø;
         }
     }
     cout << "matrix m3=\n" << m3;
     for(int i=Ø; i <m3.row; i++)
         for(j=Ø; j < m3.col; j++)
             m[i][j] = m3[i][j];
}
----------------------------------------------------------
 invert -- return invert matrix of a matrix object.
----------------------------------------------------------
int matrix::invert(matrix& a)
{
    matrix w_matrix = a;
    int error = w_matrix.lu_fact();
    if(!error)
        error = w_matrix.inv_subs();
    cout << "\n w_matrix is\n" << w_matrix;
     return error;

}
----------------------------------------------------------
 >> -- matrix input operator.
----------------------------------------------------------
istream& operator>>(istream& s, matrix& x)
{
     for(int i=0; i < x.row; i++){
       cout << "\n row " << i << " :\n";
       for(int j=0; j < x.col; j++){
           cout << " number: ";
           s >> x[i][j];
       }
     }
     return s;
}
----------------------------------------------------------
 << -- matrix output operator.
----------------------------------------------------------
ostream& operator<<(ostream& s, matrix& x)
{
    printf("%d by %d matrix is :\n", x.row, x.col);
    for(int i=0; i < x.row; i++){
        s << "\n";
        for(int j=0; j < x.col; j++)
            s << x[i][j] << ", ";
    }
    return s;
}
*/
/*********************************************************
    Ele_matrix operation/ functions
*********************************************************/
ele_matrix::ele_matrix(int sz, int ind)
{
    size = sz;
    index= ind;
    vec = new double[size];
}
ele_matrix::~ele_matrix()
{
    delete  vec;
}
```

```
void operator*(vector& y, ele_matrix& F)
{
    double sum=Ø.Ø;
    int index = F.index;
    for(int j=Ø; j < y.size; j++){
        double temp = y.v[j] * F.vec[j];
        if(fabs(temp) < TOLERENCE) temp = Ø.Ø;
        sum += temp;
    }
      y.v[index]  = sum;
}

void operator*(ele_matrix& F, vector& y)
{
    int index = F.index;
    double element = y.v[index];
    if( element == Ø.Ø ) return;
    for(int i=Ø; i < y.size; i++){
        double temp =  element * F.vec[i];
        if(fabs(temp) <= TOLERENCE) temp = Ø.Ø;
        if(i != index)
                y.v[i] += temp;
        else    y.v[i] = temp;
    }//for
}
```

```
//*******************************************************
// Input objects operation implementaiton file
//*******************************************************
#include "testio.hxx"
static CRTWND wnd1(1,1,20,79);
static CRTWND wnd2(4,1,17,79);
static CRTWND wnd(1,1,23,79);
char buf0[10000];
char bufio[80];
char outfile0[15];
int fdio;
/*
-------------------------------------------------------
 get_name constructor -- construct get_name objects.
-------------------------------------------------------
*/
get_name::get_name(char* s)
{
    lownum = 0;
    strcpy(tname,s);
    name = new char*[CONLIMIT];
}
/*
-------------------------------------------------------
 promptfor -- prompt for information(variable name,
              constraint name) from screen.
-------------------------------------------------------
*/
void get_name::promptfor(int initnum)
{
    int i = initnum;
    char s[20], istr[20];

    wnd1 << "Enter " << tname << " name, (Q if no more) ";
    wnd1.newline();
    wnd1 << ":";
    gets(s);
    while(strcmp(s,"Q") != 0 && strcmp(s,"q") != 0)
    {
        name[i] = new char[30];
        strcpy(name[i],s);
        i++;
        upnum = i;
        if(i >= 30){
           wnd1.newline();
           wnd1 << " Out of Limit";
            exit(1);
        }
        wnd1 << ":";
        gets(s);
    }//while
}
/*
----------------------------------------------------------
 list -- display names (constraint & varible names)
----------------------------------------------------------
*/
void get_name::list()
{
    char  s[20];

    for(int i=lownum; i < upnum; i++){
        wndl << crtform("\r\n%d. %s",i,name[i]);
    }//for
    wndl.newline();
}

/*
----------------------------------------------------------
 append -- append new names to original table of names.
----------------------------------------------------------
*/
void get_name::append()
{
    int i=upnum;
    promptfor(i);
}
/*
----------------------------------------------------------
 del -- delete names from the original table of names.
----------------------------------------------------------
*/
void get_name::del()
{
    list();
    char s[30];
    wndl << "\r\n NUMBER to be deleted : (Q to quit)";
    wndl << "\r\n =>";
    gets(s);
    int i;
    while( strcmp(s,"q") !=0 && strcmp(s,"Q") != 0 )
    {
        i = atoi(s);
        if( i !=0 ){
            delete(name[i]);
            if( i > maxcell ){
                wndl.newline();
                wndl << "CORP NUMBER out of bound.";
                    exit(1);
                }
                while(i < upnum){
                    name[i] = name[i+1];
                    i++;
                }
                upnum = i;
            }
            wndl.clear();
            list();
            wndl << "\r\n NUMBER to be deleted : (Q to quit)";
            wndl << "\r\n =>";
            gets(s);
        }
        list();
}
/*
----------------------------------------------------------
 modify -- modify (append, delete) names table.
----------------------------------------------------------
*/
void get_name::modify()
{
/*
    wndl.clear();
    list();
    wndl << "\r\n****************************************";
    wndl << crtform("\r\n Modify %s list",tname);
    wndl << crtform("\r\n A -- Append %s to list",tname);
    wndl << crtform("\r\n D -- Del  %s from list",tname);
    wndl << crtform("\r\n L -- List the %s list", tname);
    wndl << "\r\n Q -- quit.";
    wndl << "\r\n****************************************";
    wndl << "\r\n=>";
    char s[1Ø]; char c;
    gets(s); c = s[Ø];
    while( c != 'Q' && c != 'q' ){
        switch(c){
            case 'a' :
            case 'A' : append();
                       break;
            case 'd' :
            case 'D' : del();
                       break;
            case 'l' :
            case 'L' : list();
                       break;
            default  : wndl << "\r\n ERROR! retype !";
        }//switch
    wndl.clear();
    wndl << "\r\n****************************************";
    wndl << crtform("\r\n Modify %s list",tname);
    wndl << crtform("\r\n A -- Append %s to list",tname);
    wndl << crtform("\r\n D -- Del  %s from list",tname);
    wndl << crtform("\r\n L -- List the %s list", tname);
    wndl << "\r\n Q -- quit.";
    wndl << "\r\n****************************************";
    wndl << "\r\n=>";
    gets(s);  c = s[Ø];
    }//while
    list();
*/
}// modify

/*********************************************************
 *  Object operations for  getting constraint .ficinets.
 *********************************************************/
/*
---------------------------------------------------------
get_cons_coef -- get constraints coefficient constructor.
---------------------------------------------------------
*/
get_cons_coef::get_cons_coef()
{

    cnameptr = new get_name("CONSTRAINT");
    vnameptr = new get_name("COPR");

    col = new rec2[CONLIMIT];  // #define LIMIT = 5Ø
    row = new rec[CONLIMIT];
    rhs = new rec[CONLIMIT];
    initcost = new rec[VARLIMIT];
    contyp = new ccell[MAXCELL];
    pmtr = new numcell[MAXCELL];
    prhs = new numcell[MAXCELL];
    pcost= new numcell[MAXCELL];

    for(int i=Ø; i<VARLIMIT; i++)
        initcost[i].index = -1;
    for(i=Ø; i < CONLIMIT; i++)
        rhs[i].index = -1;
    for(i=Ø; i < CONLIMIT; i++)
        col[i].begin = -1;
    for(int j=Ø; j < CONLIMIT; j++)
        row[j].index = -1;
}
```

```
/*
------------------------------------------------------------
 ~get_cons_coef -- destructor of get_cons_coef object.
------------------------------------------------------------
*/
get_cons_coef::~get_cons_coef()
{
    delete[CONLIMIT] row;
    delete[CONLIMIT] col;
    delete[CONLIMIT] rhs;
    delete[VARLIMIT] initcost;
    delete[MAXCELL] pmtr;
    delete[MAXCELL] prhs;
    delete[MAXCELL] pcost;
}
/*
------------------------------------------------------------
 askfor -- ask for constraints coefficients.
------------------------------------------------------------
*/
void get_cons_coef::askfor()
{
    int initnum =0;    int rrow, ccol;  int help=0;
    char ss[20];
   // Windows handling
   void display_menu();
   crtwndbios(NO);
   wnd1.box();          /* box and title the window    */
   wnd1.title(1, STR_CENTER, "LP problem window");
   wnd1.clear();
   wnd1.tofront();     /* move it in front of the base */
   wnd1 << "Press H for help menu or No need (H/N) :";
   wnd1.get(&rrow,&ccol);
   gets(ss);
   if(strcmp(ss,"H")==0 || strcmp(ss,"h")==0) help =1;
   while(strcmp(ss,"H") !=0 && strcmp(ss,"h") != 0 &&
         strcmp(ss,"N") != 0  && strcmp(ss,"n") != 0 ){
              sybeep();
              wnd1.loc(rrow,ccol);
              wnd1 << "  ";
              wnd1.loc(rrow,ccol);
              gets(ss);
   }
   if(help){
       wnd2.box();
       wnd2.title(1, STR_CENTER, "Help Window");
       wnd2.tofront();
       display_menu();
   }
 char s[20];
 // prompt for problem name and objective
 wndl.newline();
 wndl << "Problem Name       :";
 gets(probname);
 wndl << "\r\nObjective (max/min) :";
 gets(objective);
 if(strcmp(objective,"MAX")==0 ||
     strcmp(objective,"max")==0 )
     maximize = 1;
 else maximize = 0;
 //get constraints name
 cnameptr->promptfor(initnum);
 cnameptr->list();
 cnameptr->modify();
 dim1 = cnameptr->upbound();

 // get corp name
 vnameptr->promptfor(initnum);
 vnameptr->list();
 vnameptr->modify();
 dim2 = vnameptr->upbound();

 int  rr, cc, rr1, cc1, rr2, cc2;
 wndl.clear();
 wndl.get(&rr,&cc);
 rr1 = rr2 = rr;
 cc1 = cc2 = cc;
 cc1 += 10;
 for(int ii=0; ii < dim2; ii++){
     wndl.loc(rr,cc1);
     wndl << vnameptr->name[ii];
     cc1 += 10;
 }//for
 // display TYPE for constraint type and
    RHV for right hand side value
 wndl.loc(rr,cc1);
 wndl << "TYPE";
 cc1 += 10;
 wndl.loc(rr,cc1);
 wndl << "RHV";

//display constraint name
for(int jj=0; jj < dim1; jj++){
    wndl.loc(++rr1,cc);
    wndl << cnameptr->name[jj];
}//for
wndl.loc(++rr1,cc);
wndl << "cost";
```

```
   // get constraints matrix coefficients
   void strfmtflts();
   char   objs[20];
   double value;
   int    index=0;
   rr2 = rr + 1;
   cc2 = cc + 10;
   strfmtflts();
   for(int i=0; i < dim2; i++){
       for(int j=0; j < dim1; j++){
            wnd1.loc(rr2,cc2);
            gets(objs);
            value = atof(objs);
            pmtr[index].row = rr2;
            pmtr[index].col = cc2;
            pmtr[index++].val = value;
            rr2++;
       }//for
       rr2 = rr + 1;
       cc2 += 10;
   }//for

// prompt for constraint type and rhs value
rr2 = rr + 1;
for(int k=0; k < dim1 ; k++){
    wnd1.loc(rr2,cc2);
    gets(objs);
    contyp[k].row = rr2;
    contyp[k].col = cc2;
    contyp[k].ch= objs[0];
    rr2++;
}
cc2 += 10;
rr2 = rr + 1;
for(k=0; k < dim1; k++){
    wnd1.loc(rr2,cc2);
    gets(objs);
    value = atof(objs);
    prhs[k].row = rr2;
    prhs[k].col = cc2;
    prhs[k].val = value;
    rr2++;
}//for

// get objective function coefficients.
int cc3 = cc + 10;
for(i=0; i < dim2; i++){
```

```
          wndl.loc(rrl,cc3);
          gets(objs);
          value = atof(objs);
          pcost[i].row = rrl;
          pcost[i].col = cc3;
          pcost[i].val = value;
          cc3 += 1Ø;
      }//for
      wndl << "\r\n Change values ?(Y/N)";
      gets(objs);
      while(strcmp(objs,"Y")==Ø || strcmp(objs,"y")==Ø ){
          mdfy_val();
          wndl << "\r\n Change values ?(Y/N)";
          gets(objs);
      }//while
  }//askfor
  void get_cons_coef::mdfy_val()
  {
         char  s[15];

         // get constraints matrix coefficients
         void strfmtflts();
         strfmtflts();
         char   objs[2Ø];
         double value;
         int    index=Ø;
         wndl.tofront();
         for(int i=Ø; i < dim2; i++){
             for(int j=Ø; j < diml; j++){
                  wndl.loc(pmtr[index].row,pmtr[index].col);
                  gets(objs);
                  if(strcmp(objs,"\Ø") != Ø)
                      pmtr[index].val = atof(objs);
                  index++;
             }//for
         }//for

          // prompt for constraint type and rhs value
          for(int k=Ø; k < diml ; k++){
              wndl.loc(contyp[k].row,contyp[k].col);
              gets(objs);
              if(strcmp(objs,"\Ø") != Ø)
                  contyp[k].ch = objs[Ø];
          }
          for(k=Ø; k < diml; k++){
              wndl.loc(prhs[k].row,prhs[k].col);
              gets(objs);
              if(strcmp(objs,"\Ø") != Ø)
                  prhs[k].val = atof(objs);
          }//for
```

```
        // get objective function coefficients.
        for(i=0; i < dim2; i++){
            wnd1.loc(pcost[i].row,pcost[i].col);
            gets(objs);
            if(strcmp(objs,"\0") != 0)
                pcost[i].val = atof(objs);
        }//for
}//mdfy_val
void get_cons_coef::condense()
{
       // get constraints matrix coefficients
       void strfmtflts();
       strfmtflts();
       int rind = 0;   //row index
       int cind = 0;
       int old;
       int index = 0;
       double value;
       for(int i=0; i < dim2; i++){
           old = rind;
           for(int j=0; j < dim1; j++){
                value = pmtr[index++].val;
                if(value != 0.0){
                    row[rind].index = j;
                    row[rind].val = value;
                    rind++;
                }
           }//for
           col[cind].begin = old;
           col[cind++].end = (rind-1);
       }//for
    // prompt for constraint type and rhs value
    for(int k=0; k < dim1 ; k++){
        contype[k] = contyp[k].ch;
    }
    int ind = 0;
    for(k=0; k < dim1; k++){
        value = prhs[k].val;
        if(value != 0.0){
             rhs[ind].index = k;  // indicate rhs
             rhs[ind].val = value;
             ind++;
        }//if
    }//for

    // get objective function coefficients.
    cind = 0;
    for(i=0; i < dim2; i++){
        value = pcost[i].val;
        if(value != Ø.Ø){
            initcost[cind].index = i;
            initcost[cind].val = value;
            cind++;
        }
    }//for
}//condense
/*
----------------------------------------------------------
print--print off constraint coeffcients in original form.
----------------------------------------------------------
*/
void get_cons_coef::print()
{
    if(db_io){
        strcpy(bufØ,"\r\n-------testio-print----");
        void strfmtflts();
        strfmtflts();
        // print objective function coef
        int i=Ø;
        while( initcost[i].index !=-1 ){
             strfmt(bufØ+strlen(bufØ),"\r\n index=%d,
             value=%5.2f",initcost[i].index,
             initcost[i].val);
             i++;
        }//while
        write(fdio,bufØ,strlen(bufØ));
        bufØ[Ø] = '\Ø';
        // print constraint coefficients
        strcpy(bufØ,"\r\n Const. Coff:");
        i=Ø;
        while(col[i].begin != -1){
            strfmt(bufØ+strlen(bufØ),"\r\n col=%d", i );
            for(int j=col[i].begin; j <= col[i].end;j++){
               strfmt(bufØ+strlen(bufØ),"\r\n\t index=%d
               \t value=%5.2f",row[j].index,row[j].val);
            }//for
            i++;
        }//while
        write(fdio,bufØ,strlen(bufØ)); bufØ[Ø]='\Ø';
       //print contype
       strcpy(bufØ,"\r\n CONtype is:");
       for(int tt=Ø; tt < dim1; tt++)
            strfmt(bufØ+strlen(bufØ),"\rcontype[%d]=%d",
            tt,contype[tt]);
        write(fdio,bufØ,strlen(bufØ)); bufØ[Ø]='\Ø';
       // print rhs coefficients
       strcpy(bufØ,"\r\n RHS val");
       i=Ø;
```

```
while(rhs[i].index != -1 ){
    strfmt(buf0+strlen(buf0),"\r\n[%d] val= %5.2f",
                   rhs[i].index,rhs[i].val);
    i++;
}//while
write(fdio,buf0,strlen(buf0)); buf0[0]='\0';


wnd1.tofront();
wnd1 << "\r\n To Quit?(Y/N)";
gets(buf0);
```

```
        if(strcmp(buf0,"Y")==0 || strcmp(buf0,"y")==0)
        exit(1);
    }
}//print

void get_cons_coef::readfile()
{
    int search(char*,char**,int,int);
    int readln(int,char*);
    void substring(char*,int,int,char*);
    int fdi,n;
    char  infile[15];

    // open input file
    wnd1.clear();
    wnd1.tofront();
    wnd1 << "Enter input file name :";
    gets(infile);
    fdi = open(infile,0);
    if( fdi < 0 )
    {
      wnd1 << "input file open error -- program stopped";
      exit(1);
    }
    // read problem name and objective
    char s[30];
    n=readln(fdi,s);
    substring(s,14,23,probname);
    n=readln(fdi,objective);
    maximize = (   strcmp(objective,"MAXIMIZE")==0
           || strcmp(objective,"maximize")==0) ? 1 : 0;
    n=readln(fdi,s);           // skip the line of ROWS
    // get constraint names
    dim1=0;
    n=readln(fdi,s);
    while( n > 0 && strcmp(s,"COLUMNS")!=0){
        contype[dim1] = s[1];
        strcpy(s,s+4);
        if(strcmp(s,"OBJ")!=0){
            cnameptr->name[dim1] = new char[strlen(s)+1];
            strcpy(cnameptr->name[dim1],s);
            dim1++;
        }
        n=readln(fdi,s);
    }//while
    // get crop name and restore tableau values
    int r=0;
    int c=0;
    int numcol =0;
```

```
char line[8Ø], s1[3Ø], s2[3Ø], s3[3Ø], s4[1Ø], s5[1Ø];
int flag = Ø;
n = readln(fdi,line);
int ind = Ø;
int rhsind=Ø;
int cind = -1;
int rind =  Ø;
double value;
while( n > Ø ){
    if( strcmp(line,"RHS") == Ø ){
        flag = 1;
        n = readln(fdi,line);
    }
    if( !flag ){
        substring(line,4,13,s1);
        substring(line,14,23,s2);
        substring(line,24,35,s3);
        if(!sc)
            value = atof(s3);
        else{
            substring(line,24,31,s4);
            substring(line,34,35,s5);
            value = atof(s4);
            int exp  = atoi(s5);
            if(line[33] == '-')
                for(int t=Ø; t < exp; t++)
                   value = value * Ø.1 ;
            else
                for( t=Ø;  t < exp; t++)
                   value = value * 1Ø.Ø ;
        }
        r = search(s1,vnameptr->name,numcol,1);
        if( r == -1 ){ r = numcol++; cind++;
            dim2 = cind + 1;col[cind].begin = rind;
        }
        if(strcmp(s2,"OBJ") == Ø){
            initcost[ind].index = r;
            initcost[ind].val = value;
            ind++;
        }//if
        else{
            r = search(s2,cnameptr->name,dim1,Ø);
            row[rind].index = r;
            row[rind].val   = value;
            col[cind].end = rind;
            rind++;
        }
    }// if
```

```
        else{  // get RHS value
            substring(line,14,23,s2);
            substring(line,24,34,s3);
            double value = atof(s3);
            if(sc){
                substring(line,24,31,s4);
                substring(line,34,35,s5);
                value = atof(s4);
                int exp  = atoi(s5);
                if(line[33] == '-')
                    for(int t=Ø; t < exp; t++)
                       value = value * Ø.1 ;
                else
                    for( t=Ø;  t < exp; t++)
                       value = value * 1Ø.Ø ;
            }
            c = search(s2, cnameptr->name, dim1, Ø);
            rhs[rhsind].index = c;
            rhs[rhsind].val   = value;
            rhsind++;
        }//else
        n = readln(fdi,line);
    }// while
    close(fdi);
}//readfile
int search(char *s, char **name, int num, int ins=Ø)
{
        int k=Ø;
        while(k < num){
            if(strcmp(s,name[k])==Ø)
                return(k);
            else k++;
        }
        if( ins == 1){
            name[k] = malloc(strlen(s) + 1);
            if( name[k] == NULL ){
                wndl << "memory insufficient.\n";
                exit(1);
            }
            strcpy(name[num],s);
            return(-1);
        }
}

int readln(int fdi, char* s)
{
    char c;
    int i,n;
    for(i=Ø, n=read(fdi,&c,1); n > Ø && c != '\n';
                  n=read(fdi,&c,1) )
            s[i++] = c;
    s[i] = '\Ø';
    return(n);
}
void substring(char* in, int start, int end, char* out)
{
    for(int i=Ø, j=start; j <= end; j++)
        if(in[j] != ' ' && in[j] !='\t' && in[j] != '\n')
             out[i++] = in[j];
    out[i] = '\Ø';
}
void display_menu()
{
   int  r,c;
   char s[2Ø];
   wnd2 << "\r\n this is menu";

   wnd2 << "\r\n Press Q to quit menu";
   wnd2.get(&r,&c);
   gets(s);
   while(strcmp(s,"Q") !=Ø && strcmp(s,"q") != Ø ){
              sybeep();
              wnd2.loc(r,c);
              wnd2 << "  ";
              wnd2.loc(r,c);
              gets(s);
   }
   wnd2.clear();
   wnd2.toback();
}
```

```
//*******************************************************
//     Class transform object implementation file
//*******************************************************
#include "testform.hxx"
//static CRTWND wnd3(2,1,20,79);
int db_form = 0;
int fdform;
char outfile1[15];
char buf1[10000];
/*
---------------------------------------------------------
 transform constructor -- transform object constructor.
---------------------------------------------------------
*/
transform::transform(int c,int v)
{
    contype = new char[numcon];
    B = new matrix(c,v);
    R = new vector(c);
  OBJ = new vector(v);
}
/*
---------------------------------------------------------
 ~transform -- transform object destructor.
---------------------------------------------------------
*/
transform::~transform()
{
    delete B;
    delete R;
    delete OBJ;
}
/*
---------------------------------------------------------
processmx--transform sparse representation to full matrix
---------------------------------------------------------
*/
void transform::processmx(get_cons_coef& A)
{
    (*B) = 0.0;  // clear B
    //copy contype from get_cons_coef object to
    //  transform object
    for(int i=0; i < numcon; i++){
        contype[i] = A.contype[i];
    }

    // transform mx
    i=0;
    while(A.col[i].begin != -1){
        for(int j=A.col[i].begin;j <= A.col[i].end;j++){
            int k = A.row[j].index;
            (*B).m[k][i] = A.row[j].val;
        }
        i++;
    }//while
}
/*
-------------------------------------------------------
 processrhs -- transform sparse vector respensentation
               to full vector rhs.
-------------------------------------------------------
*/
void transform::processrhs(get_cons_coef& A)
{
//    wnd3 << "\r\n this is processrhs";
    (*R) = Ø.Ø;  // clear R
    // process rhs
    int i=Ø;
    while(A.rhs[i].index != -1){
        int j = A.rhs[i].index;
        if(A.rhs[i].val >=Ø.Ø)
            (*R).v[j] = A.rhs[i].val;
        else{    // rhs < Ø.Ø
            contype[i] = (contype[i] == 'G') ? 'L':'G';
            (*R).v[j] = -A.rhs[i].val;
            for(int k=Ø; k < numrealact; k++)
                if( (*B).m[j][k] != Ø.Ø )
                    (*B).m[j][k] = -(*B).m[j][k];
        }//else
        i++;
    }//while

    // calculate actual numgreaterthan, numlessthan, and
    // numequalto constraint
    for(i=Ø; i < numcon; i++){
        switch(contype[i]){
            case 'G':
            case 'g':
                      numgreaterthan++;
                      break;
            case 'L':
            case 'l':
                      numlessthan++;
                      break;
            case 'E':
            case 'e':
                      numequalto++;
                    break;
           default :
                    exit(1);
       }//switch
    }//for
    kbwait(100);
}//processmx
/*
----------------------------------------------------------
 processobj --transform obj. coef. vector to full vector.
----------------------------------------------------------
*/
void transform::processobj(get_cons_coef& old)
{
//    wnd3 << "\r\n this is processOBJ";
    (*OBJ) = 0.0;  // clear OBJ

    // process objective function
    int i=0;
    while(old.initcost[i].index!=-1){
        int j = old.initcost[i].index;
        (*OBJ).v[j] = old.initcost[i].val;
        i++;
    } //while
}
/*
----------------------------------------------------------
 output -- print off transform object.
----------------------------------------------------------
*/
void transform::output()
{
//   wnd3 << "\r\n this is output";
   if(debug){
     void strfmtflts();
     strfmtflts();

    write(fdform,"\r\n----------TESTFORM output",27);
    write(fdform,"\r\n Contype is:\r\n",16);bufl[0]='\0';
    for(int r=0; r < numcon; r++){
       strfmt(bufl+strlen(bufl),"\r\ncontype[%d]= %d",
       r,contype[r]);
    }
    write(fdform,bufl,strlen(bufl)); bufl[0] = '\0';

    write(fdform,"\r\n matrix B is:",10);
    for(int x=0; x < B->row; x++){
         strfmt(bufl+strlen(bufl),"\r\n row %d",x);
         for(int y=0; y < B->col; y++){
                strfmt(buf1+strlen(buf1),"%10.4f, ",
                (*B).m[x][y]);
            }//for
            write(fdform,buf1,strlen(buf1)); buf1[0] = '\0';
        }//for

        write(fdform,"\r\n RHS is:\r\n",12);
            for(int kk=0; kk < R->size; kk++){
              strfmt(buf1+strlen(buf1),"%10.4f,",(*R).v[kk]);
            }//for
        write(fdform,buf1,strlen(buf1)); buf1[0] = '\0';

        write(fdform,"\r\n OBJ is:\r\n",15) ;
            for(kk=0; kk < OBJ->size; kk++){
              strfmt(buf1+strlen(buf1),"\r\n%10.4f",
              (*OBJ).v[kk]);
            }
        write(fdform,buf1,strlen(buf1));
        close(fdform);
        char s[15];
        wnd3 << "\r\n to Quit testform ? (Y/N)";
        gets(s);
        if(strcmp(s,"Y")==0 || strcmp(s,"y")==0) exit(1);
      }//if debug
    }//output
```

```
//*******************************************************
//               Vector operations header file
// ******************************************************
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
#include "simheader.hxx"

struct vector{
    int size;                   // size of vector
    double* v;                  // pointer to first element
public:
    vector(int sz =10);
    vector(vector&);
    ~vector();
    int sizevec();
    int most(int);
    int least(int);
    double operator[](int);
    void   operator=(double);
    double sum();
    void operator+=(vector&);
    void operator-=(vector&);
    void operator*=(vector&);
    void operator/=(vector&);
    vector operator+(vector&);
    vector operator-(vector&);
    vector operator*(vector&);
    vector operator/(vector&);

    vector operator+(double);
    vector operator-(double);
    vector operator*(double);
    vector operator/(double);

    void operator+=(double);
    void operator-=(double);
    void operator*=(double);
    void operator/=(double);
    friend istream& operator>>(istream&, vector&);
    friend ostream& operator<<(ostream&, vector&);
};
```

```
//********************************************************
//           Implementation file for simplex method
//********************************************************
#include "tableau.hxx"
static CRTWND wnd4(1,1,20,79);
void strfmtflts();
int fdo;
char outfile[15];
char buf[10000];
int  itnumdb;
int fd2;
char outfile2[15];
/*
---------------------------------------------------------
 simplex tableau constructor -- allocating spaces for obj
---------------------------------------------------------
*/
tableau::tableau(int numcon, int numact)
{
    contype = new char[numcon];
    basisno = new int[numact];
    basis   = new int[numact];
    bigmind = new int[numact];     // bigm indicator

    tm = new matrix(numcon,numrealact);
    mx  = new matrix(numcon,numact);
    cost = new vector(numact);
    z = new vector(numact);
    shadow = new vector(numact);
    pivotrow = new vector(numact);
    rhs  = new vector(numcon);
    ratio= new vector(numcon);
    actlevel = new vector(numcon);
}
/*
---------------------------------------------------------
 simplex tableau destructor
---------------------------------------------------------
*/
tableau::~tableau()
{
    delete mx;
    delete cost;
    delete z;
    delete shadow;
    delete pivotrow;
    delete ratio;
    delete actlevel;
}
```

```
/*
----------------------------------------------------------
 tableauinit -- intialize simplex tableau variables.
----------------------------------------------------------
*/
void tableau::tableauinit(transform& F)
{
    // get value of rhs, cost, mx, and contype
    (*rhs) = (*F.R);
    (*tm)  = (*F.B);
    (*cost) = Ø.Ø;
    for(int j=Ø; j < numrealact; j++)
        (*cost).v[j] =(*F.OBJ).v[j];
    for(int i=Ø; i< numcon; i++){
        contype[i] = F.contype[i];
    }//for

    // initialize variables
    numartvar =Ø;
    numnonartvar =Ø;
    bcount =Ø;
    *z = Ø.Ø;
    *shadow =Ø.Ø;

    for(i=Ø; i < numact; i++)
        basis[i] = MARKZERO;
    for(i=Ø; i < numact; i++)
        basisno[i] = MARKZERO;
    for(i=Ø; i < numact; i++)
        bigmind[i] = MARKZERO;
}
/*
----------------------------------------------------------
 setup--appending i-matrix to tableau and appending big M
         to cost vector.
----------------------------------------------------------
*/
void tableau::setup(get_cons_coef& AA)
{
    (*mx) = Ø.Ø;  // clear mx
    // copy tm matrix to mx
    for(int i=Ø; i< numcon; i++)
        for(int j=Ø; j < numrealact; j++)
            (*mx).m[i][j] = (*tm).m[i][j];

    // append i-matrix to end of transformed matrix
    int g;
    for(i=Ø, g=Ø; i < numcon; i++){
        int nn = numcon + numrealact;
        switch(contype[i]){
            case 'G' :
            case 'g' :
                        numartvar++;
                        (*mx).m[i][numrealact + i] = -1.Ø;
                        (*cost).v[numrealact+i] = Ø.Ø;
                        (*mx).m[i][nn + g] = 1.Ø;
                        (*cost).v[nn + g] = (maximize) ?
                          MINUSBIGM : BIGM;
                        bigmind[i] = nn + g;
                        AA.vnameptr->name[nn + g] =
                           malloc(strlen("ARTIFICAL")+1);
                        strcpy(AA.vnameptr->name[nn+g],
                                       "ARTIFICAL");
                        basisno[bcount] = nn + g;
                        basis[nn+g] = bcount;
                        g++;
                        bcount++;
                        break;
            case 'E' :
            case 'e' :
                        numartvar++;
                        (*mx).m[i][numrealact+i] = 1.Ø;
                        (*cost).v[numrealact+i] =
                          (maximize) ? MINUSBIGM : BIGM;
                        bigmind[i] = numrealact+i;
                        basisno[bcount] = numrealact + i;
                        basis[numrealact + i] = bcount;
                        bcount++;
                        break;
            case 'L' :
            case 'l' :
                        (*mx).m[i][numrealact+i] = 1.Ø;
                        (*cost).v[numrealact+i] = Ø.Ø;
                        bigmind[i] = MARKZERO;
                        basisno[bcount] = numrealact + i;
                        basis[numrealact + i] = bcount;
                        bcount++;
                        break;
            default:
                        exit(1);
        }// switch
    }
    numnonartvar = numcon + numrealact;
    computeshadowprice();
    objlevel=(maximize==1) ? MINUSBIGM*BIGM : BIGM*BIGM;
}//setup
```

```
/*
---------------------------------------------------------
 solveinit -- initialization of first iteration.
---------------------------------------------------------
*/
void tableau::solveinit()
{
    itnum = Ø;
    quit = FALSE;
    in_col = entering();
    out_row = leaving();
}
/*
---------------------------------------------------------
 computeshadowprice -- computing Zj and Zj - Cj.
---------------------------------------------------------
*/
void tableau::computeshadowprice()
{
    double sum;
    for(int j=Ø; j < numact; j++){
        if(basis[j] == MARKZERO){
            sum = Ø.Ø;
            for(int i=Ø; i < numcon; i++)
                sum += (*mx)[i][j] * (*cost)[basisno[i]];
            (*z).v[j] = sum;
            (*shadow).v[j] = sum - (*cost)[j];
        }
    else{
            (*z).v[j] = (*cost)[j];
            (*shadow).v[j] = Ø.Ø;
        }
    }
}
/*
---------------------------------------------------------
 obj -- computing objective value.
---------------------------------------------------------
*/
double tableau::obj()
{
    double sum = Ø.Ø;
    for(int i=Ø; i < numcon; i++)
        sum += rhs->v[i] * cost->v[basisno[i]];
    return sum;
}
```

```
/*
----------------------------------------------------------
 ck_optimal -- checking if optimal solution is reached.
----------------------------------------------------------
*/
void tableau::ck_optimal(get_cons_coef& AA)
{
    if(out_row != MARKZERO){
        final_output(AA);
        return;
    }
    else if(in_col != MARKZERO){
        wnd4 << "\n Unbounded solution!!!!!!!";
        kbwait(300);
        return;
    }
    else
        final_output(AA);
}//ck_optimal
/*
----------------------------------------------------------
 entering -- finding entering variable.
----------------------------------------------------------
*/
int tableau::entering()
{
    int i;
    i = shadow->most(maximize);
    return(i);
}//entering
/*
----------------------------------------------------------
 leaving -- finding leaving variable.
----------------------------------------------------------
*/
int tableau::leaving()
{
    int i;
    double temp;
    vector in_vector(numcon);
    in_vector = (*mx)(in_col);
    for(int r=0; r < numcon; r++){
        temp = in_vector[r];
        if(fabs(temp) < TOLERENCE) temp = 0.0;
        if( temp > 0.0 )
            ratio->v[r] = rhs->v[r] / temp;
        else  ratio->v[r] = PINFINATE;
    }//for
    i = ratio->least(0);
    return(1);
}//leaving
/*
----------------------------------------------------------
 pivot_set --  getting pivot element.
----------------------------------------------------------
*/
void tableau::pivot_set()
{
    pivot = (*mx).m[out_row][in_col];
    (*mx).m[out_row][in_col] = 1.Ø;
    basis[basisno[out_row]] = MARKZERO;
    basis[in_col] = out_row;
    basisno[out_row] = in_col;
}//pivot_set
/*
----------------------------------------------------------
    rowdivide -- dividing elementary row of LP tableau by
                 a real number.
----------------------------------------------------------
*/
void tableau::row_divide()
{
    (*rhs).v[out_row] /= pivot;

    // set pivot row
    for(int j=Ø; j < numact; j++)
        if(basis[j] == MARKZERO)
            (*mx).m[out_row][j] /= pivot;
}// rowdivide
/*
----------------------------------------------------------
    rowEliminate -- performing row-eliminate on tableau.
----------------------------------------------------------
*/
void tableau::row_eliminate()
{
    double vv2,v1;
    for(int i=Ø; i < numcon ; i++ )
    {
         if(i != out_row){
             double element = (*mx).m[i][in_col];
             if(element != Ø.Ø){
                 v1 = (*rhs).v[i] - element *
                      (*rhs).v[out_row];
                 (*rhs).v[i] = setp(v1);
                 for(int j=Ø; j < numact; j++)
                     if(basis[j] == MARKZERO){
                         vv2 = (*mx).m[i][j] - element *
```

```
                                         (*mx).m[out_row][j] ;
                              (*mx).m[i][j] = setz(vv2);
                          }//if
                  }//if
                  (*mx).m[i][in_col] = 0.0;
              }//if
        }//for
}//row eliminate
/*
----------------------------------------------------------
 simplex -- simplex method drive routine.
----------------------------------------------------------
*/
void tableau::simplex(transform& F, get_cons_coef& AA)
{
    char  str[15];
    void strfmtflts();
    wnd4.box();
    wnd4.title(1, STR_CENTER, "Simplex Version Window");
    wnd4.clear();
    wnd4.tofront();

    tableauinit(F);
    setup(AA);        // setup tableau
    solveinit();
    strfmtflts();
    while( in_col != MARKZERO && out_row !=
                                   MARKZERO && !quit )
    {
        wnd4 << crtform("\r\n iteration %d
                objlevel= %10.4f",itnum, objlevel);
        itnum++;
        pivot_set();
        if(pivot != 1.0)
            row_divide();
        row_eliminate();
        computeshadowprice();
        objlevel = obj();
        in_col = entering();
        out_row= leaving();
    }//while
    if(!quit) ck_optimal(AA);
}
double tableau::setz(double x)
{
    if(fabs(x) < TOLERENCE)
         return  0.0;
    else  return  x;
}
```

```
double tableau::setp(double x)
{
    if( x < TOLERENCE )
        return Ø.Ø;
    else   return x;
}
/*
----------------------------------------------------------
 tab_output
----------------------------------------------------------
*/
void tableau::tab_output()
{

    void strfmtflts();
    strfmtflts();
    write(fdo,"\r\n basis is:",12);
    for( int i=Ø; i < numact; i++){
         strfmt(buf,"%d, ",basis[i]);
         write(fdo,buf,strlen(buf));
    }

    write(fdo,"\r\n basisno is:",14);
    for( i=Ø; i < numact ; i++){
         strfmt(buf,"%d, ",basisno[i]);
         write(fdo,buf,strlen(buf));
    }

    write(fdo,"\r\n mx is:",1Ø);
    for(int x=Ø; x < mx->row; x++){
         write(fdo,"\r\n",2);
         for(int y=Ø; y < mx->col; y++){
                  strfmt(buf,"%1Ø.4f, ",(*mx).m[x][y]);
                  write(fdo,buf,strlen(buf));
         }//for
   }//for

    write(fdo,"\r\n rhs is:\r\n",12);
         for(int kk=Ø; kk < rhs->size; kk++){
           strfmt(buf,"%5.2f, ", (*rhs).v[kk]);
           write(fdo,buf,strlen(buf));
         }//for

    write(fdo,"\r\n shadow is:\r\n",15) ;
         for(kk=Ø; kk < shadow->size; kk++){
           strfmt(buf,"%5.2f, ", (*shadow).v[kk]);
           write(fdo,buf,strlen(buf));
         }
}
```

```
/*
----------------------------------------------------------
 final_output -- output final solution.
----------------------------------------------------------
*/
void tableau::final_output(get_cons_coef& AA)
{
    void strfmtflts();
    strfmtflts();
    wnd4.clear();
    wnd4 << crtform("\r\n Problem name : %s ", probname);

    // check infeasible or feasible
    int infeasible = FALSE;
    int i=0;
    while( i < numcon && !infeasible){
        int j=0;
        while( j < bcount && !infeasible){
            if(basisno[j] == bigmind[i] &&
                                (*rhs).v[j] != 0.0 ){
                infeasible = TRUE;
                wnd4 << "\r\n Infeasible solution.";
                kbwait(300);
                return;
            }//if
            j++;
        }// while
        i++;
   }//while

    // output final solution
    if( !infeasible ){
        wnd4 << "\r\nBasis activity name \t Value";
        wnd4 << "\r\n------------------- \t -----";
        int k;
        for(i=0; i < numcon; i++){
            k = basisno[i];
            if( k < numrealact)
                wnd4 << crtform("\r\n   %s     %10.4f ",
                       AA.vnameptr->name[k],(*rhs).v[i]);
            else wnd4 << crtform("\r\n    slack %10.4f ",
                                            (*rhs).v[i]);
        }//for
        wnd4 << crtform("\r\n Optimal solution is :
                                    %10.4f ", objlevel);
        kbwait(500);
    }//if
}//final_output
```

```
/*********************************************************
 *    Revised Simplex Method Implementation File         *
 *********************************************************/
#include "revise.hxx"
static CRTWND wnd4(1,1,20,79);
char buf[10000];
int fdo;
void strfmtflts();
/*
----------------------------------------------------------
 revise constructor--construct a  Revise objects.
----------------------------------------------------------
*/
revise::revise(int numcon, int numact)
{
    contype = new char[numcon];
    basisno = new int[numact];
    basis   = new int[numact];
    nonbasis= new int[numact];
    bigmind = new int[numact];

    tm = new matrix(numcon,numrealact);
    mx = new matrix(numcon,numact);
    A  = new matrix(numcon,numrealact+numgreaterthan);

    E  = new ele_matrix*[1000];

    cost  = new vector(numact);
    ocost = new vector(numact);
    shadow= new vector(numrealact+numgreaterthan);
    rhs   = new vector(numcon);
    y     = new vector(numcon);
    afa   = new vector(numcon);
    xxb   = new vector(numcon);
    Cb    = new vector(numcon);
    Cn    = new vector(numrealact+numgreaterthan);
    p     = new vector(numcon);
}
/*
----------------------------------------------------------
 revise destructor
----------------------------------------------------------
*/
revise::~revise()
{
    delete tm;
    delete mx;
    delete A;
    delete E;
    delete cost;
    delete ocost;
    delete y;
    delete shadow;
    delete rhs;
    delete afa;
    delete xxb;
    delete Cb;
    delete Cn;
    delete p;
}
/*
-----------------------------------------------------------
 reviseinit -- intialize variables for revised simplex.
-----------------------------------------------------------
*/
void revise::reviseinit(transform& F)
{

    // get value of rhs, cost, mx from transform object F
    (*rhs) = (*F.R);
    (*tm)  = (*F.B);
    (*cost) = Ø.Ø;
    for(int j=Ø; j < numrealact; j++)
        (*cost).v[j] =(*F.OBJ).v[j];

    // get constraint type from input object F
    for(int i=Ø; i< numcon; i++){
        contype[i] = F.contype[i];
    }

    // get original cost from input object
    (*ocost) = Ø.Ø;
    for(j=Ø; j < numrealact; j++)
        (*ocost).v[j] =(*F.OBJ).v[j];


    // initialize variables
    numartvar =Ø;
    numnonartvar =Ø;
    bcount =Ø;
    *afa    = Ø.Ø;
    *shadow = Ø.Ø;
    *Cn     = Ø.Ø;
    *Cb     = Ø.Ø;
    for(i=Ø; i < numact; i++)
        basis[i] = MARKZERO;
    for(i=Ø; i < numact; i++)
        basisno[i] = MARKZERO;
    for(i=Ø; i < numact; i++)
        nonbasis[i] = MARKZERO;
    for(i=Ø; i < numact; i++)
        bigmind[i] = MARKZERO;

}
/*
---------------------------------------------------------
 solveinit -- initilizing the first iteration.
              for matrix b_1 and N, vector Cb and Cn.
              Find enter variable and leaving variable.
---------------------------------------------------------
*/
void revise::solveinit(get_cons_coef& AA)
{
    itnum = Ø;
    quit = FALSE;
    parameter_set();
    tab_output();
    in_col = entering(AA);
    p_set();
    out_row = leaving(AA);
}//solveinit

/*
---------------------------------------------------------
 parameter_set -- insert big M into cost vector, get
                  Cb,Cn values.
---------------------------------------------------------
*/

void revise::parameter_set()
{
    strfmtflts();
    // cost vector handling to eleminate BIGM in it
    int k;
    for(int i=Ø; i < numcon; i++){
        if(bigmind[i] != MARKZERO){
            k = bigmind[i];
            double element = (*cost).v[k];
            for(int j=Ø; j < numact; j++)
                cost->v[j] = cost->v[j] -
                element * (*mx).m[i][j];
         }// if
    }//for

    // compute Cb
    for(i =Ø; i < bcount; i++)
        (*Cb).v[i] = (*cost).v[basisno[i]];
    // compute Cn
    for(i=0; i < numact-numcon; i++){
        (*Cn).v[i] = cost->v[nonbasis[i]];
    }

    // compute A
    for(int j=0; j < numact; j++){
        if(nonbasis[j] != MARKZERO)
            for(i=0; i < numcon; i++)
                (*A).m[i][j] =(*mx).m[i][nonbasis[j]];
    }//for
}//param_set
/*
---------------------------------------------------------
 p_set -- get enter variable column vector from N matrix.
---------------------------------------------------------
*/
void revise::p_set()
{
   (*p) =(*mx)(nonbasis[in_col]);
}

/*
---------------------------------------------------------
 entering -- Finding entering variable procedure is :
                1. compute y = Cb * b_1
                2.   Zj - Cj = y * N - Cn
                3.   pick up most negative value.
---------------------------------------------------------
*/
int revise::entering(get_cons_coef& AA)
{
    int i;
    (*y) = (*Cb);        //for 1st iteration
    if(itnum > 0){       //for 2nd and rest iteration
        for(int i=itnum-1; i>=0; i--)
                (*y) * (*E[i]);
    }//if

    vector t = (*y) * (*A);
    (*shadow) = t - (*Cn);
    i = shadow->most(maximize);
    return i;

}//entering
```

```
/*
-----------------------------------------------------------
 leaving -- Finding leaving variable procedure is :
               1. compute xxb = Cb * rhs
               2.         afa = b_1 * p
               3. pick up smallest value.
-----------------------------------------------------------
*/
int revise::leaving(get_cons_coef& AA)
{
    double temp;
    (*afa) = (*p);
    if(itnum > Ø){
         for(int n=Ø; n < itnum; n++)
                (*E[n]) * (*afa);
    }//if
    int j = ck_unbound();
    if (j == 1)  return(-1);   // return out index = -1

    // xxb = b_1 * (*rhs)
    for(int ii=Ø; ii < numcon; ii++)  //don't change
       xxb->v[ii] = rhs->v[ii];
    if(itnum > Ø){
         for(int ii=Ø; ii < itnum; ii++){
              (*E[ii]) * (*xxb);
          }//for
    }//if
       for(int k=Ø; k < numcon; k++ )
        if( afa->v[k] > Ø.Ø ){
           temp = xxb->v[k] / afa->v[k];
           if(temp < TOLERENCE) temp = Ø.Ø;
           xxb->v[k] = temp;
        }
        else  xxb->v[k] = PINFINATE;

    int i = xxb->least(Ø);
    return i;
}
/*
-----------------------------------------------------------
 ck_unbound -- check unbounded solution.
-----------------------------------------------------------
*/
int revise::ck_unbound()
{
    double temp;
    int j=Ø;
    while(j < afa->size){
       temp = afa->v[j];
         if(fabs(temp) < TOLERENCE) temp = Ø.Ø;
         if(temp > Ø.Ø)
             return(Ø);   // no unbound occurs
         else  j++;
    }
    return(1);  // unbound solution  (*afa) <= Ø.Ø
}
/*
---------------------------------------------------------
 cal_e -- get vector e as one column of elemary matrix E.
---------------------------------------------------------
*/
void revise::cal_e()
{
    double value;
    vector temp(numcon);
    for(int i=Ø; i < numcon; i++)
        if(i==out_row)  temp.v[i] = 1.Ø;
        else            temp.v[i] = -(afa->v[i]);
    double dividor = afa->v[out_row];

    E[itnum] = new ele_matrix(numcon,out_row);
    for(i=Ø; i < numcon; i++){
        value = temp.v[i] / dividor;
//        if(fabs(value) < TOLERENCE) value = Ø.Ø;
        E[itnum]->vec[i]  = value;
    }
    E[itnum]->index = out_row;
}
/*
---------------------------------------------------------
 obj -- objective value computation.
---------------------------------------------------------
*/
double revise::obj()
{
        for(int j=Ø; j < itnum; j++)
               (*E[j]) * (*rhs);

        vector CCB(numcon);
        for(int i=Ø; i < numcon; i++)
            CCB.v[i] = ocost->v[basisno[i]];

        vector result(numcon);
            result = CCB * (*rhs);
        double value = result.sum();
        return value;
}
```

```
/*
---------------------------------------------------------
 simplex -- revise simplex method driver routine.
---------------------------------------------------------
*/
void revise::simplex(transform& F, get_cons_coef& AA )
{
    char str[15];
    void strfmtflts();
    wnd4.box();
    wnd4.title(1,STR_CENTER,"Revised Version Window");
    wnd4.clear();
    wnd4.tofront();
    reviseinit(F);
    setup(AA);
    solveinit(AA);
    while( in_col != MARKZERO && out_row != MARKZERO
           && !quit)
    {
        wnd4 << crtform("\r\n iteration %d", itnum);
        cal_e();
        basis_set();
        cb_set();
        itnum++;
        if( (in_col = entering(AA)) == MARKZERO ) break;
        p_set();
        out_row = leaving(AA);
    }
    objlevel = obj();
    if(!quit) ck_optimal(AA);
}
/*
---------------------------------------------------------
 cb_set -- changing cost value for basis cost vector and
           non-basis cost vector.
---------------------------------------------------------
*/
void revise::cb_set()
{
    // compute Cb
    for(int i=0; i < numcon; i++)
        (*Cb).v[i] = (*cost)[basisno[i]];

    //compute Cn
    for(i=0; i < numact-numcon; i++)
        (*Cn).v[i] = (*cost)[nonbasis[i]];
}
```

```
/*
---------------------------------------------------------
 basis_set -- changing index for basis array and nonbasis
---------------------------------------------------------
*/
void revise::basis_set()
{
    basis[basisno[out_row]] = MARKZERO;
    int t = nonbasis[in_col];
    nonbasis[in_col] = basisno[out_row];
    basis[in_col] = out_row;
    basisno[out_row] = t;

    // compute A
    for(int j=0; j < numact-bcount; j++){
        for(int i=0; i <numcon; i++)
            (*A).m[i][j] = (*mx).m[i][nonbasis[j]];
    }//for
}
/*
---------------------------------------------------------
 ck_optimal -- see if solution is optimal
---------------------------------------------------------
*/
void revise::ck_optimal(get_cons_coef& AA)
{
    if(out_row != MARKZERO){
        final_output(AA);
        return;
    }
    else if(in_col != MARKZERO){
        wnd4 << "\r\n Unbounded solution!!!!!!!";
    }
    else
        final_output(AA);
}
/*
---------------------------------------------------------
 setup -- appending identity matrix to the input matrix
          to get constraint matrix; adding big M to cost
          vector for artifical variable.
---------------------------------------------------------
*/
void revise::setup(get_cons_coef& AA)
{
    (*mx) = 0.0;  // clear mx

    // copy tm matrix to mx
```

```
for(int i=Ø; i< numcon; i++)
      for(int j=Ø; j < numrealact; j++)
          (*mx).m[i][j] = (*tm).m[i][j];

  // append i-matrix to end of transformed matrix
  int g;
  for(i=Ø, g=Ø; i < numcon; i++){
      int nn = numcon + numrealact;
      switch(contype[i]){
          case 'G' :
          case 'g' :
                     numartvar++;
                     (*mx).m[i][numrealact + i] = -1.Ø;
                     (*cost).v[numrealact+i] = Ø.Ø;
                     (*mx).m[i][nn + g] = 1.Ø;
                     (*cost).v[nn + g] = (maximize) ?
                                  MINUSBIGM : BIGM;
                     bigmind[i] = nn + g;
                     AA.vnameptr->name[nn + g] =
                          malloc(strlen("ARTIFICAL")+1);
                     strcpy(AA.vnameptr->name[nn+g],
                                    "ARTIFICAL");
                     basisno[bcount] = nn + g;
                     basis[nn+g] = bcount;
                     g++;
                     bcount++;
                     break;
          case 'E' :
          case 'e' :
                     numartvar++;
                     (*mx).m[i][numrealact+i] = 1.Ø;
                     (*cost).v[numrealact+i] =
                      (maximize) ? MINUSBIGM : BIGM;
                     bigmind[i] = numrealact+i;
                     basisno[bcount] = numrealact + i;
                     basis[numrealact + i] = bcount;
                     bcount++;
                     break;
          case 'L' :
          case 'l' :
                     (*mx).m[i][numrealact+i] = 1.Ø;
                     (*cost).v[numrealact+i] = Ø.Ø;
                     bigmind[i] = MARKZERO;
                     basisno[bcount] = numrealact + i;
                     basis[numrealact + i] = bcount;
                     bcount++;
                     break;
          default:
                printf("\r\n no such contype:%d ",
                                 contype[i]);
                     exit(1);
          }// switch
      }
      numnonartvar = numcon + numrealact;

      // set nonbasis index for compute A later
      j=Ø;
      for(i=Ø; i < numact; i++)
          if(basis[i] == MARKZERO)
              nonbasis[j++] = i;

      objlevel =(maximize==1) ? MINUSBIGM*BIGM : BIGM*BIGM;

  }//setup


  /*
  ---------------------------------------------------------
      tab_output
  ---------------------------------------------------------
  */
  void revise::tab_output()
  {
     int yy;
    if(itnum == Ø){
      write(fdo,"\r\n mx is:",1Ø); buf[Ø] = '\Ø';
      for(int x=Ø; x < mx->row; x++){
           strfmt(buf,"\nrow=%d\n",x);
           for(int y=Ø; y < mx->col; y++){
                 if( (yy = y % 6) == Ø )  strcat(buf,"\n");
                    strfmt(buf+strlen(buf),"%1Ø.4f, ",
                    (*mx).m[x][y]);
           write(fdo,buf,strlen(buf)); buf[Ø] = '\Ø';
           }//for
      }//for
     }//if itnum ==Ø
      write(fdo,"\r\n basisNO is:",14);
      for( int i=Ø; i < numact; i++){
           strfmt(buf+strlen(buf),"%d, ",basisno[i]);
           write(fdo,buf,strlen(buf)); buf[Ø] = '\Ø';
      }
      write(fdo,"\r\n NON-basis is:",16); buf[Ø] = '\Ø';
      for( i=Ø; i < numact ; i++){
           strfmt(buf+strlen(buf),"%d, ",nonbasis[i]);
      }
      write(fdo,buf,strlen(buf)); buf[Ø] = '\Ø';
    write(fdo,"\r\n Cb is:",9);
    for(i=Ø; i < Cb->size; i++)
        strfmt(buf+strlen(buf),"%1Ø.4f, ",(*Cb).v[i]);
    write(fdo,buf,strlen(buf)); buf[Ø] = '\Ø';

    write(fdo,"\r\n Cn is:",9);
    for(i=Ø; i < Cn->size; i++)
        strfmt(buf+strlen(buf),"%1Ø.4f, ",(*Cn).v[i]);
    write(fdo,buf,strlen(buf)); buf[Ø] = '\Ø';

    write(fdo,"\r\n A is: ",9);
    for(int k=Ø; k< A->row; k++){
        strcpy(buf,"\r\n");
        for(int t=Ø; t < A->col; t++){
           strfmt(buf+strlen(buf),"%1Ø.4f,",(*A).m[k][t]);
           write(fdo,buf,strlen(buf)); buf[Ø] = '\Ø';
        }
    }
    write(fdo,"\r\n RHS is:",1Ø);
    for(i=Ø; i < rhs->size; i++)
        strfmt(buf+strlen(buf),"%1Ø.4f, ",(*rhs).v[i]);
    write(fdo,buf,strlen(buf)); buf[Ø] = '\Ø';

   if(itnum>Ø){
    strfmt(buf,"\r\nE[%d] index=%d",itnum-1,
           E[itnum-1]->index);
    write(fdo,buf,strlen(buf));  buf[Ø] = '\Ø';
    for(int t=Ø; t < numcon; t++){
        strfmt(buf+strlen(buf),"%1Ø.4f,",
        E[itnum-1]->vec[t]);
    }
    write(fdo,buf,strlen(buf)); buf[Ø] = '\Ø';
   }//if itnum>Ø
    cout << "\n Finished Iteration " << itnum;
    cout << "\n Q to stop processing(Q)=>";
    gets(s);
    if(strcmp(s,"Q")==Ø || strcmp(s,"q")==Ø ){
         close(fdo);
         exit(1);
    }
}
/*
----------------------------------------------------------
 final_output -- output the final solution
----------------------------------------------------------
*/
void revise::final_output(get_cons_coef& AA)
{
    wnd4.clear();
    wnd4 << "\r\n Problem name : " <<  probname;
    // check infeasible or feasible
    int infeasible = FALSE;
    int i=Ø;
    while( i < numcon && !infeasible){
        int j=Ø;
        while( j < bcount && !infeasible){
            if(basisno[j] == bigmind[i] &&
                                    (*rhs).v[j] != Ø.Ø ){
                infeasible = TRUE;
                wnd4 << "\r\n Infeasible solution.";
                return;
            }//if
            j++;
        }// while
        i++;
   }//while

    // output final solution
    if( !infeasible ){
        wnd4 << "\r\nBasis activity name \t Value";
        wnd4 << "\r\n------------------ \t -----";
        int k;
        for(i=Ø; i < numcon; i++){
            k = basisno[i];
            if( k < numrealact){
                wnd4 << crtform("\r\n %s \t %1Ø.4f",
                    AA.vnameptr->name[k],(*rhs).v[i]);
            }
        }//for
        wnd4 << crtform("\r\n Optimal solution is:%1Ø.4f",
                         objlevel);
    }//if
}//final_output


int revise::smallest(vector& xxb)
{
    double leastval;
    int    index, same;
    leastval = BIGM;
    index    = MARKZERO;
    for(int i=Ø; i < rhs->size; i++){
        if( xxb.v[i] >= Ø.Ø && xxb.v[i] < leastval){
            leastval = xxb.v[i];
            index = i;
        }
        if( xxb.v[i] >= 0.0 && xxb.v[i] == leastval)
             same = i;
    }//for
    if( xxb.v[same] == leastval ){
        double val1 = (*mx).m[index][0]/ xxb.v[index];
        double val2 = (*mx).m[same ][0]/ xxb.v[same];
        if( (val1 <= 0.0  && val2 >= 0.0)
        || ( val1 >= 0.0 && val2 >=0.0 &&  val2 < val1))
             index = same;
    }//if
    return(index);
}
```

```
/*********************************************************
 *    The Simplex Method Driver Routine
 *********************************************************/
#include "tableau.hxx"
#include "crt.hxx"

static CRTWND wnd6(22,1,1,79);
int pivot[30];
double rowmax[30];
double temp;
int piv_elem;
int maxc;
double sum;
int inv_pivot[30];
int   numcon;
int   numrealact;
int   numact;              // # of total activities
int   numlessthan=0;       // # of less than constraints
int   numequalto=0;        // # of equality constraints
int   numgreaterthan=0;    // # of greater than constraints

char probname[50];
char objective[50];
int  maximize;


main(int argc, char *argv[])
{
    void mody(get_cons_coef&, transform&, tableau&);
    int  interactive=0;
    char s[20], ss[10];


    wnd6.box();
    wnd6.title(1, STR_CENTER, "OSU Computer Science");
    wnd6.tofront();

    int quit = 0;
while(!quit){
    wnd6 << "\r\n Please wait !!";
    kbwait(200);

    get_cons_coef  A;
    wnd6 << "\r\n Interactive, File, or Quit? (I/F/Q)";
    gets(s);
    if(strcmp(s,"Q")==0 || strcmp(s,"q")==0) exit(1);
    if(strcmp(s,"I")==0 || strcmp(s,"i")==0){
        interactive = 1;
        A.askfor();
        A.condense();
    }
    else{
        interactive = 0;
        A.readfile();
    }
    numcon = A.dim1;
    numrealact = A.dim2;

    // transform from sparce to full representation
    transform F(numcon,numrealact);
    F.processmx(A);
    F.processrhs(A);
    F.processobj(A);
    F.output();

    // simplex version
    numact = numcon + numrealact + numgreaterthan;
    tableau  S(numcon,numact);
    S.simplex(F,A);

    if(interactive){
        wnd6.tofront();
        wnd6 << "\r\nModify values ?(Y/N)";
        gets(ss);
        while(strcmp(ss,"Y") == 0 || strcmp(ss,"y") ==0){
            mody(A,F,S);
            wnd6 << "\r\n Modify values ?(Y/N)";
            gets(ss);
        }//while
     }//if

     wnd6 << "\r\nAnother LP problem ? (Y/N)";
     gets(ss);
     if(strcmp(ss,"N")==0 || strcmp(ss,"n")==0 ){
         quit = 1;
         wnd6 << "\r\n    Good Bye! ";
     }
     kbwait(200);
}//while
}//main
void mody(get_cons_coef& A, transform& F, tableau& S)
{
   A.mdfy_val();
   A.condense();
```

```
    A.print();
    numcon = A.dim1;
    numrealact = A.dim2;

    // transform from sparce to full representation
    F.processmx(A);
    F.processrhs(A);
    F.processobj(A);
    F.output();

    // simplex version
    numact = numcon + numrealact + numgreaterthan;

    S.simplex(F,A);
}
```

```
/**********************************************************
 *      The Revised Simplex Method Driver Routine
 *********************************************************/
#include "revise.hxx"
#include "crt.hxx"
static CRTWND wnd6(22,1,1,79);
int pivot[30];
double rowmax[30];
double temp;
int piv_elem;
int maxc;
double sum;
int inv_pivot[30];
int   numcon;
int   numrealact;
int   numact;               // # of total activities
int   numlessthan=0;        // # of less than constraints
int   numequalto=0;         // # of equality constraints
int   numgreaterthan=0;     // # of greater than constraints

char probname[50];
char objective[50];
int  maximize;

CRT    *crt;

main(int argc, char *argv[])
{
    void mody(get_cons_coef&, transform&, revise&);
    int  interactive=0;
    char s[20], ss[10];


    wnd6.box();
    wnd6.title(1, STR_CENTER, "OSU Computer Science");
    wnd6.tofront();

    int quit = 0;
while(!quit){
    wnd6 << "\r\n Please wait !!";
    kbwait(200);

    get_cons_coef  A;
    wnd6 << "\r\n Interactive, File, or Quit? (I/F/Q)";
    gets(s);
    if(strcmp(s,"Q")==0 || strcmp(s,"q")==0) exit(1);
    if(strcmp(s,"I")==0 || strcmp(s,"i")==0){
        interactive = 1;
        A.askfor();
        A.condense();
    }
    else{
        interactive = 0;
        A.readfile();
    }
    numcon = A.dim1;
    numrealact = A.dim2;

    // transform from sparce to full representation
    transform F(numcon,numrealact);
    F.processmx(A);
    F.processrhs(A);
    F.processobj(A);
    F.output();

    // simplex version
    numact = numcon + numrealact + numgreaterthan;
    revise  R(numcon,numact);
    R.simplex(F,A);

    if(interactive){
        wnd6.tofront();
        wnd6 << "\r\nModify values ?(Y/N)";
        gets(ss);
        while(strcmp(ss,"Y") == 0 || strcmp(ss,"y")==0){
            mody(A,F,R);
            wnd6 << "\r\n Modify values ?(Y/N)";
            gets(ss);
        }//while
     }//if

     wnd6 << "\r\nAnother LP problem ? (Y/N)";
     gets(ss);
     if(strcmp(ss,"N")==0 || strcmp(ss,"n")==0 ){
         quit = 1;
         wnd6 << "\r\n    Good Bye! ";
     }
     kbwait(200);
}//while
}//main
void mody(get_cons_coef& A, transform& F, revise& R)
{
   A.mdfy_val();
   A.condense();
   A.print();
   numcon = A.dim1;
   numrealact = A.dim2;
   // transform from sparce to full representation
   F.processmx(A);
   F.processrhs(A);
   F.processobj(A);
   F.output();

   // simplex version
   numact = numcon + numrealact + numgreaterthan;

   R.simplex(F,A);
}
```

```
//*******************************************************
//               Output Class Header File
//*******************************************************
#include "revise1.hxx"

class output{
      char  *buf;
      int   fdo;
public:
      output();
      ~output();
      void display(revise&,get_cons_coef&);
};
```

```
//*******************************************************
//                  Output Implementation File
//*******************************************************
#include "output1.hxx"
extern infeasible;
static CRTWND  wnd5(1,1,20,79);
/*
--------------------------------------------------------
              output constructor
--------------------------------------------------------
*/
output::output()
{
    buf = new char[10000];
}
/*
--------------------------------------------------------
              output class destructor
--------------------------------------------------------
*/
output::~output()
{
    delete buf;
}
/*
--------------------------------------------------------
              display operation
--------------------------------------------------------
*/
void output::display(revise& R, get_cons_coef& AA)
{
    wnd5.box();
    wnd5.title(1, STR_CENTER, "Output Window");
    wnd5.tofront();

    void strfmtflts();
    strfmtflts();
    wnd5.clear();
    wnd5 << crtform("\r\n Problem name : %s ", probname);

    // check infeasible or feasible
     if(infeasible){
           wnd5 << "\r\n Infeasible Solution.";
           return;
     }
    // output final solution
    if( !infeasible ){
        wnd5 << "\r\nBasis activity name \t Value";
        wnd5 << "\r\n------------------- \t -----";
        int k;
        for(int i=0; i < numcon; i++){
            k = R.basisno[i];
            if( k < numrealact)
                wnd5 << crtform("\r\n   %s      %10.4f ",
                       AA.vnameptr->name[k],R.rhs->v[i]);
            else wnd5 << crtform("\r\n    slack %10.4f ",
                                              R.rhs->v[i]);
        }//for
        wnd5 << crtform("\r\nOptimal solution is:%10.4f "
                     , R.objlevel);
        kbwait(200);
    }//if
}//output
```

```
//*******************************************************
//               Second Output Implementation File
//*******************************************************
#include "output1.hxx"
extern infeasible;
static CRTWND  wnd5(1,1,20,79);
/*
-------------------------------------------------------
                 Output Class Constructor
-------------------------------------------------------
*/
output::output()
{
    buf = new char[10000];
}
/*
-------------------------------------------------------
                 Output Class Destructor
-------------------------------------------------------
*/
output::~output()
{
    delete buf;
}
/*
-------------------------------------------------------
                 Display Operations
-------------------------------------------------------
*/
void output::display(revise& R, get_cons_coef& AA)
{
    wnd5.box();
    wnd5.title(1, STR_CENTER, "Output Window");
    wnd5.tofront();

    void strfmtflts();
    strfmtflts();
    wnd5.clear();
    wnd5 << crtform("\r\n Problem name : %s ", probname);

    // check infeasible or feasible
     if(infeasible){
           wnd5 << "\r\n Infeasible Solution.";
           return;
     }

    char s[10];
    wnd5(1,25) << "\r\n Final Output file name :";
    gets(s);
    fdo = creat(s,Ø755);
    if(fdo < Ø){
        wnd5 << "\r\n output file open error --program
                                         stopped.";
        exit(1);
    }//if
    buf[Ø] = '\Ø';
    strcpy(buf+strlen(buf),"\r\nBasis activity name
                                             \t Value");
    strcpy(buf+strlen(buf),"\r\n------------------
                                             \t -----");
    for(int i=Ø; i < numcon; i++){
        int kk = R.basisno[i];
        if( kk < numrealact)
            strfmt(buf+strlen(buf),"\r\n%s \t\t %1Ø.4f",
                    AA.vnameptr->name[kk],R.rhs->v[i]);
    }//for
    strfmt(buf+strlen(buf),"\r\nOptimal solution is :
                                        %1Ø.4f ",R.objlevel);
    write(fdo,buf,strlen(buf));
    close(fdo);
}//output
```

```
//*******************************************************
//                Output Class Header File
//*******************************************************
#include "revise1.hxx"

class output{
      char  *buf;
      int   fdo;
public:
      output();
      ~output();
      void display(revise&,get_cons_coef&);
};
```

```
//*********************************************************
//                Output Implementation File
//*********************************************************
#include "output1.hxx"
extern infeasible;
static CRTWND  wnd5(1,1,20,79);
/*
---------------------------------------------------------
              output constructor
---------------------------------------------------------
*/
output::output()
{
    buf = new char[10000];
}
/*
---------------------------------------------------------
              output class destructor
---------------------------------------------------------
*/
output::~output()
{
    delete buf;
}
/*
---------------------------------------------------------
              display operation
---------------------------------------------------------
*/
void output::display(revise& R, get_cons_coef& AA)
{
    wnd5.box();
    wnd5.title(1, STR_CENTER, "Output Window");
    wnd5.tofront();

    void strfmtflts();
    strfmtflts();
    wnd5.clear();
    wnd5 << crtform("\r\n Problem name : %s ", probname);

    // check infeasible or feasible
     if(infeasible){
          wnd5 << "\r\n Infeasible Solution.";
          return;
     }
    // output final solution
    if( !infeasible ){
        wnd5 << "\r\nBasis activity name \t Value";
        wnd5 << "\r\n------------------- \t -----";
        int k;
        for(int i=0; i < numcon; i++){
            k = R.basisno[i];
            if( k < numrealact)
                wnd5 << crtform("\r\n   %s      %10.4f ",
                              AA.vnameptr->name[k],R.rhs->v[i]);
            else wnd5 << crtform("\r\n    slack %10.4f ",
                                                  R.rhs->v[i]);
        }//for
        wnd5 << crtform("\r\nOptimal solution is:%10.4f "
                            , R.objlevel);
        kbwait(200);
    }//if
}//output
```

VITA

Yore-Phang Tzay

Candidate for the Degree of

Master of Science

Thesis: OBJECT-ORIENTED DESIGN WITH C++ : A LINEAR PROGRAMMING APPLICATION

Major Field: Computing and Information Science

Biographical:

Personal Data: Born in Taipei, Taiwan, Republic of China, March 2, 1960, the daughter of Mr. and Mrs. Lie-Guang Tzay.

Education: Graduate from Kaohsiung Senior High Shool for Girls, Kaohsiung, Taiwan, Republic of China, in June, 1977; received Bachelor of Art degree in Public Administration from National Cheng-Chi University, Taipei, Taiwan, in June, 1983; completed requirements for the Master of Science degree in Computer Science at Oklahoma State University, Stillwater, Oklahoma, in July, 1988.

Professional Experience: Assistant Engineer, International AOC Co., Taipei, Taiwan, 1983-1985.