MS-DOS용 선형계획법 통합환경 소프트웨어의 개발

설동렬 · 박찬규 · 서용원 · 박순달*

Development of an LP Integrated Environment Software under MS-DOS

Tongryeol Seol · Chankyoo Park · Yongwon Seo · Soondal Park*

ABSTRACT

This paper is to develop an integrated environment software on MS-DOS for linear programming.

For the purpose,

First, the linear programming integrated environment software satisfying both the educational purpose and the professional purpose was designed and constructed on MS-DOS.

Second, the text editor with big capacity was developed. The arithmetic form analyser was also developed and connected to the text editor so that users can input data in the arithmetic form.

As a result, users can learn and perform linear programming in the linear programming integrated environment software.

1. 서 론

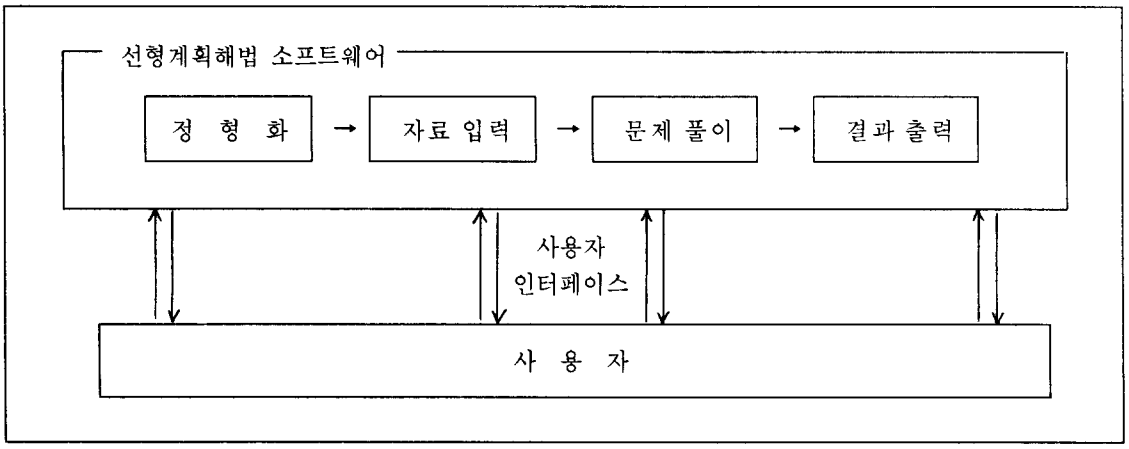
선형계획법은 경영과학 분야에서 수리모형으로 시스템의 기획 및 설계 분야, 경영관리 및 운영 분야, 공학 분야의 최적화 분야, 그밖에 환경 분야 및 공공 시스템 분야 등 다방면에 걸쳐 널리 활용되고 있다. [3] 선형계획문제를 손으로 직접 풀기에는 실제 문제들의 크기가 방대하기 때문에 컴퓨터의 사용이 필수적인데 하드웨어와 소프트웨어의 성능이 모두 향상되어서 대형 문제도 짧

* 서울대학교 산업공학과

은 시간에 풀어낸다. [6] 예를 들어 개인용 컴퓨터(486DX/33Mhz)에서 비영 요소의 개수가 367,000개인 문제를 208초 안에 풀어낼 정도로 성능이 모두 우수하다. [7]

그런데 실제 문제에 선형계획법을 응용하기 위해서는 문제의 정형화 과정, 입력 자료의 형성 과정, 적절한 해법의 선택 과정 및 해를 분석하는 과정 등 다양한 절차를 거쳐야 하며 입력 프로그램, 해법 프로그램 등 여러 개의 소프트웨어를 사용하여야 한다. 또한 각 과정은 선형계획법에 익숙하지 않은 의사결정자에게 있어서는 복잡하고 어려운 것이 사실이다. 따라서 선형계획법을 보다 쉽게 활용하려면 통합환경을 지원하는 선형계획법 소프트웨어가 요구된다. 여기서 선형계획법 통합환경 소프트웨어란 선형계획법을 수행하기 위한 여러 가지 기능들을 사용자 편의성을 강조한 통합 메뉴로 제어하는 소프트웨어를 의미한다. [2]

선형계획해법 소프트웨어의 경우에 있어서 사용자 편의성을 향상시키기 위한 요소들을 분석할 필요가 있으며, 어떠한 기능들이 통합되어 할 것인지도 재고되어야 한다. 다음의 [그림 1]은 선형계획해법 소프트웨어에서의 사용자 인터페이스를 보여 준다.



[그림 1] 선형계획해법 소프트웨어의 사용자 인터페이스

본 연구에서는 선형계획해법 소프트웨어에 있어서 자료의 입력과 출력, 사용자와의 커뮤니케이션 등 사용자 인터페이스를 향상시키기 위한 방안을 모색하고 구현하였다. 특히 사용자 인터페이스를 향상시키는 방안 가운데에 하나로 자료의 입력에 필수적인 텍스트 에디터와 편리한 입력을 돕기 위한 수식 해석기를 개발하였고, 풀다운 메뉴를 기본으로 한 통합환경 소프트웨어 LIPED (Linear Programming Educational Package) [4]을 구축하였다.

앞에서 언급했듯이 선형계획법을 수행하기 위해서는 여러 개의 프로그램이 필요하므로 사용자가 필요한 프로그램을 매번 찾아서 실행시키기란 매우 불편한 일이다. 따라서 다양한 프로그램들에 사용자가 빨리 익숙해지고 또한 쉽게 사용할 수 있는 통합된 메뉴를 지원하는 것이 필요하다. 한편 선형계획법 소프트웨어를 사용할 때에는 자료파일을 많이 사용하게 되므로 선형계획법 통합환경 소프트웨어에는 파일의 편집, 출력 등 파일을 관리하기 위한 기능이 포함되어야 한다. 또한 선

형계획해법이 사용하는 자료 파일의 형태가 다양하기 때문에 사용자가 각각의 파일 형태를 모두 기억하여 그 형태에 맞게 자료 파일을 만들어 내기 어려우므로 한 형태의 파일을 다른 형태의 파일로 변환하는 기능이 요구된다. 그리고 실제 문제에 응용하는 경우 이외에 해법 자체를 연구하기 위한 목적으로 선형계획해법 소프트웨어를 사용할 경우에는 실험을 위해 자동으로 선형계획문제를 생성해 주는 기능이 필요하다.

일반적인 문서에서 컬럼의 개수는 그리 크지 않으므로 기존의 텍스트 에디터들은 대체로 1,000개 정도로 제한하고 있다. 예를 들어 BC++ 3.1 통합환경의 에디터는 1,022개로 제한하고 있고 [5], QEdit 3.0은 1,000개로 제한하고 있다. 그러나 선형계획법 문제의 자료는 제약식의 개수는 적어도 변수의 개수가 상대적으로 매우 많은 특성을 가지고 있어서 MPS 형태와 같이 규격화된 형태가 아닌 경우에는 매우 많은 컬럼 개수를 요구한다. 따라서 컬럼 수에 제한을 두지 않으면서 최대한 많은 메모리를 사용할 수 있도록 하는 자료 구조가 필요하다.

선형계획법의 자료를 입력하는 데에는 프롬프트 방식은 프로그램의 요구에 따라서 사용자가 하나씩 답하는 프롬프트(prompt) 방식, 프로그램이 원하는 파일 형태로 자료를 만들어 놓으면 프로그램이 파일을 읽어 들이는 파일 방식, 스프레드쉬트(spread-sheet) 형태의 자료 입력기를 통해서 자료를 입력하는 방식 그리고 선형계획법 문제를 변수 이름, 사칙 연산 기호, 부등호와 등호 등을 이용해서 입력하는 수식 형태 방식 등이 있다. 스프레드쉬트 방식이나 수식 형태 방식은 파일 방식과 함께 사용될 수 있다. 각 방법은 나름대로의 장단점을 가지고 있지만 이 가운데 스프레드쉬트 방식과 수식 형태 방식이 사용자 편의성의 측면에서 프롬프트 방식에 비해 우수하다. 스프레드쉬트 방식은 입력 방법이 간단하지만 행과 열의 수는 크지만 비영 요소가 적을 경우 입력할 위치를 찾는 과정이 불편할 수 있으며 수식 형태 방식은 비영 요소만 입력하면 되기 때문에 비영 요소 개수가 적을 때에 오히려 편리하지만 변수 이름을 매번 써 주어야 하는 점이 불편할 수 있다.

2. 선형계획해법 통합환경 소프트웨어의 설계

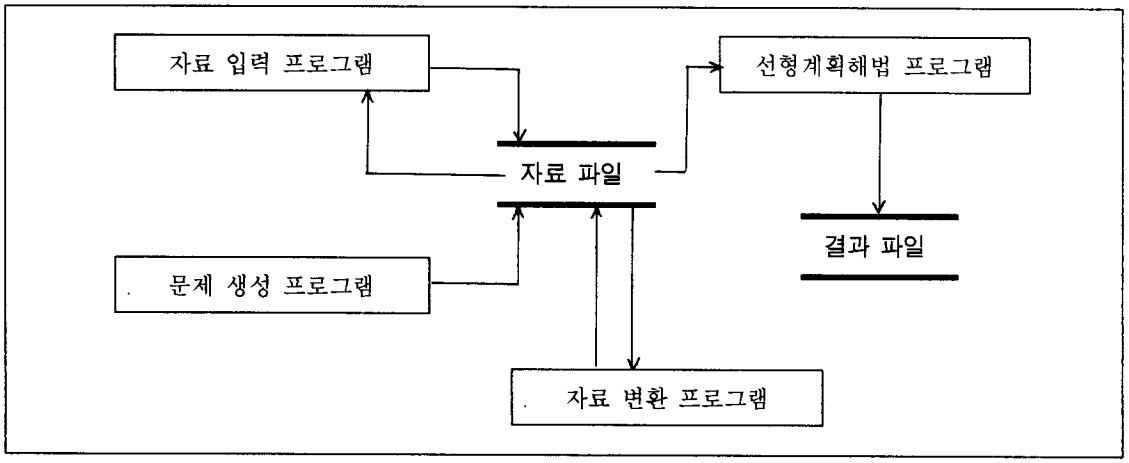
선형계획해법 통합환경 소프트웨어에서 사용하는 자료에는 선형계획법 문제를 내용으로 하는 자료 파일과 선형계획해법 프로그램이 문제를 푼 다음 그 결과를 출력하는 결과 파일이 있다.

[표 1] 선형계획법 소프트웨어의 파일 사용

	자료 파일		결과 파일	
프로그램 종류	입력	출력	입력	출력
자료 입력 프로그램	○	○		
문제 생성 프로그램		○		
자료 변환 프로그램	○	○		
선형계획해법 프로그램	○			○

자료 입력 프로그램을 사용해서 자료 파일을 새로 만들기도 하며 기존의 자료 파일을 읽어 들여서 수정하기도 한다. 또한 실험용 선형계획법 문제를 생성하는 프로그램에 의해 자료 파일이 만들어지기도 한다. 자료 파일은 행단위 형태, 열단위 형태, MPS 형태의 세 가지 파일 형태를 가지고 있다. 자료 변환 프로그램은 세 가지 가운데 어느 한 형태로 되어 있는 자료 파일을 다른 형태로 변환해 준다. 자료 파일이 선형계획해법 프로그램에 의해 읽혀지고 문제를 풀고 나면 문제를 푼 결과와 사용자가 원하는 수준에 따라 문제를 푸는 과정을 텍스트 파일로 출력한다.

이상의 내용을 자료 흐름도로 표현하면 [그림 2]와 같이 된다.



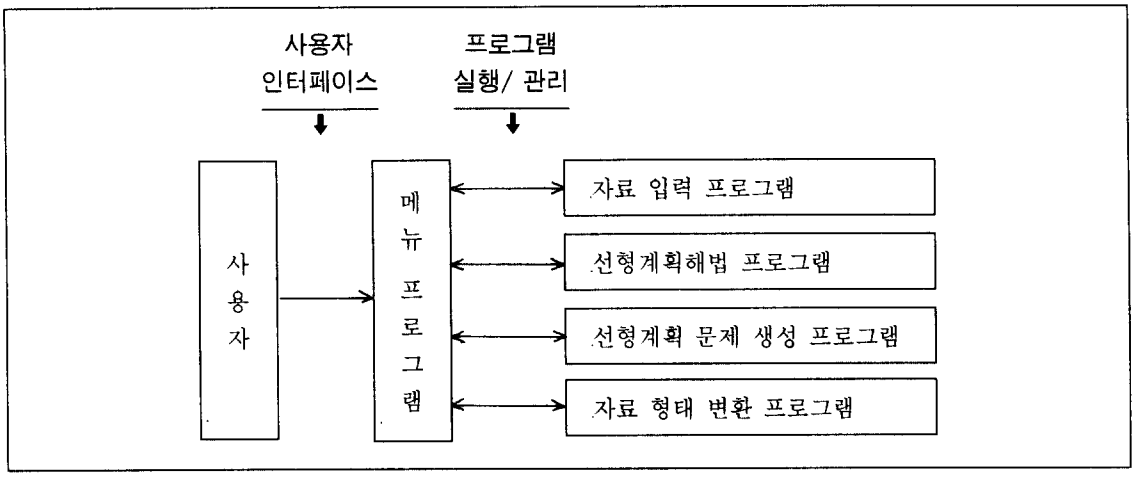
[그림 2] 선형계획해법 소프트웨어의 자료 흐름도

결과 파일은 선형계획해법 프로그램의 출력으로 생성될 뿐 다른 용도로는 사용되지 않는다. 그러나 자료 파일은 소프트웨어를 수행하는 모든 과정에서 입력이나 출력으로 사용되고 있다. 또한 한 가지 프로그램에서만 사용하는 것이 아니라 여러 개의 프로그램에 의해 입력과 출력으로 사용되므로 소프트웨어 구조를 설계할 때에 자료 파일이 원활하게 사용될 수 있도록 고려할 필요가 있다.

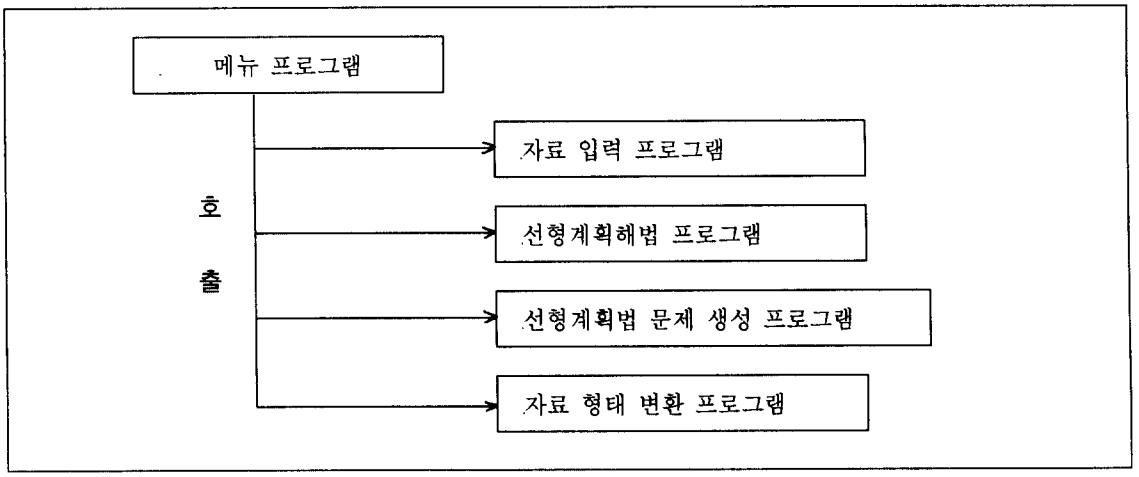
LIPED는 메뉴 프로그램, 자료 입력 프로그램, 선형계획해법 프로그램들, 선형계획법 문제 생성 프로그램들, 자료 형태 변환 프로그램으로 구성되어 있다.

LIPED에서 메뉴 프로그램은 메뉴를 통제하는 프로그램으로 하고 상위 프로그램이다. 메뉴 프로그램은 사용자와 다른 프로그램들 사이의 인터페이스 역할을 함과 동시에 다른 프로그램들의 수행을 관리하는 역할을 감당한다. [그림 3]은 메뉴 프로그램의 이러한 역할을 설명하고 있다.

사용자가 메뉴 프로그램을 제어하면 사용자의 제어에 따라 메뉴 프로그램은 하부 프로그램들을 제어한다. 즉 [그림 4]와 같이 메뉴 프로그램 아래 자료 입력 프로그램, 선형계획해법 프로그램들, 선형계획문제 생성 프로그램들, 자료 형태 변환 프로그램 등의 프로그램들이 접속되어 있어서 메뉴 프로그램에 의해서 호출되어 실행된다. 즉, 메뉴 프로그램을 상위 프로그램으로 나머지 프로그램은 그 하위 프로그램으로 조직되어 있다.



[그림 3] 메뉴 프로그램의 기능



[그림 4] 메뉴프로그램과 다른 하위 프로그램과의 관계

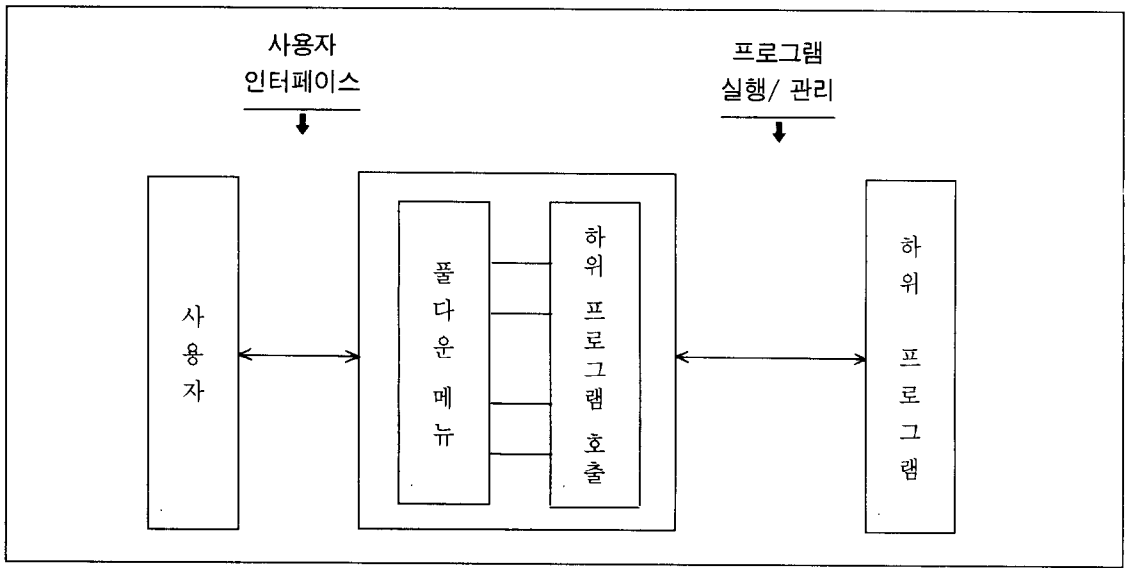
따라서 사용자가 직접 원하는 기능을 수행할 프로그램을 구동시킬 필요가 없고 메뉴 프로그램을 통해서 원하는 기능을 선택하는 일만 하면 된다. 선형계획해법 통합환경을 구성하고 있는 프로그램은 여러 개이지만 사용자가 직접 구동시키는 프로그램은 오직 메뉴 프로그램 하나이다. 결국 사용자는 프로그램 상호 간의 구조를 모르는 상태에서도 풀다운 메뉴를 통해서 각 프로그램들을 실행시켜 원하는 작업을 수행할 수 있도록 하였다.

3. 하부 프로그램의 접속

메뉴 프로그램은 사용자로 하여금 원하는 기능을 선택하도록 메뉴를 화면에 보여 주고 사용자와 인터페이스 하는 메뉴 구동 부분과 선택된 기능에 대해 해당되는 하위 프로그램을 실행시켜 주는 하위 프로그램 호출 부분으로 나뉘어진다.

자료 입력 프로그램, 선형계획해법 프로그램, 선형계획문제 생성 프로그램, 자료 형태 변환 프로그램은 각각 독립적인 완전한 프로그램으로 설

계되어 메뉴 프로그램에 소스 프로그램 수준에서 종속적이지 않도록 하였다. 따라서 독립적으로 프로그램 기능이 개선될 수 있도록 하였으며 통합 환경 안에서는 메뉴 프로그램의 호출에 의한 실행으로 접속되어 사용된다. 사용자가 메뉴 프로그램을 통해서 원하는 기능을 선택하면 메뉴 프로그램은 선택된 기능에 해당하는 프로그램을 호출한다. 프로그램이 실행되면 사용자는 비로소 잠시 메뉴 프로그램을 떠나서 실행된 프로그램과 인터페이스 하게 되며 하위 프로그램이 종료되면 다시 메뉴 프로그램으로 돌아온다.



[그림 5] 메뉴 프로그램의 구조

사용자는 메뉴 구동 부분만 화면을 통해 볼 수 있고, 하위 프로그램 호출 부분은 사용자가 보지 못하도록 숨겨져 있다. 사용자는 프로그램의 실행 및 관리 영역에 대해 전혀 알지 못해도 메뉴 구동 부분에만 인터페이스 함으로 원하는 기능을 수행하도록 할 수 있다. 따라서 메뉴 구동 부분은 사용자가 알기 쉽고 편리하게 사용할 수 있는 형태를 취해야 한다. 메뉴 구동 부분은 메뉴의 구조를 쉽게 이해할 수 있고, 사용하기에 편리한 풀다운 메뉴를 사용하였다.

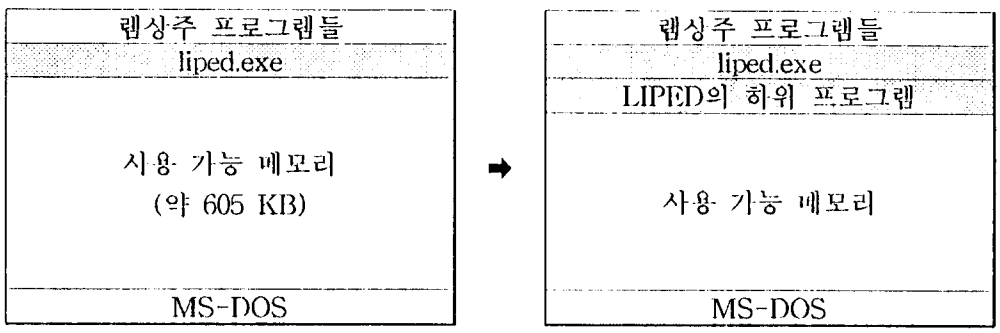
사용자가 메뉴에서 특정 항목을 선택하면, 메뉴 프로그램은 선택된 메뉴 항목에 대응하는 하위 프로그램을 메모리로 로드(load) 하여 실행시키는 역할을 해야 한다. 하위 프로그램을 실행시키는 과정은 디스크에 있는 프로그램을 읽어서 메모리로 로드 하여 실행시키는 것인데 이 때에 메모리로 로드 하여 실행시키는 프로그램을 부모 프로그램(parent program)이라고 하고 메모리에 로드 되어 실행되는 프로그램을 자식 프로그램(child program)이라고 한다. C 언어를 사용할 경우에 ANSI C의 system() 함수를 사용하면 하위 프로그램을 실행시킬 수 있다. MS Windows에서는 특별히 WinExec()이라는 함수를 따로 제공한다.

부모 프로그램이 자식 프로그램을 실행시키면 부모 프로그램이 사용하고 있는 메모리를 제외한

나머지 메모리가 자식 프로그램에게 할당되며, 실행이 종료되면 메모리가 운영체계에 반환되고 프로그램 실행의 제어권은 부모 프로그램에게로 돌아온다. 자식 프로그램은 프로그램의 실행을 종료하면서 자신의 실행 결과를 나타내는 정보를 부모 프로그램에 반환할 수 있다. exit() 함수의 파라미터로 전달된 값이 부모 프로그램에 전달될 수 있다. 자식 프로그램이 부모 프로그램에 전달하는 정보는 숫자로 표현되는 하나의 코드(code)이며, 운영체계에서 정해 놓은 것도 있으므로 중복되지 않도록 해야 한다. 부모 프로그램은 이 값을 통해 자식 프로그램이 정상적으로 종료되었는지 강제적으로 또는 에러에 의해 종료되었는지 알 수 있으며, 좀더 구체적인 프로그램 상황을 전달할 수도 있다.

메뉴 프로그램이 부모 프로그램의 역할을 하게 되며, 다른 하위 프로그램들은 모두 메뉴 프로그램의 자식 프로그램이 된다. 각 하위 프로그램들의 종료된 상황은 메뉴 프로그램에 의해 앞에서 설명한 방법으로 관리될 수 있다.

LIPED에서 접속되어 사용되는 모든 부프로그램은 각각 독립적인 실행 파일로 되어 있으며 LIPED의 메뉴 프로그램에 의해 호출되어 실행된다. 따라서 사용자가 LIPED를 사용하고 있는 도중에는 언제나 메뉴 프로그램이 컴퓨터의 메모리 상에 남아 있다. 호출된 부프로그램은 메뉴 프로그램이 차지하고 있는 메모리를 제외한 나머지 메모리를 사용한다. 메뉴 프로그램의 이름은 liped. exe인데 이 프로그램은 메모리에서 약 90 KB를 차지한다. 따라서 부프로그램은 MS-DOS와 기타 램상주 프로그램들이 사용하는 메모리를 제외한 나머지 사용 가능한 메모리 가운데 메뉴 프로그램이 차지하고 있는 약 90 KB를 뺀 나머지의 메모리를 사용할 수 있게 된다.



[그림 6] 하위 프로그램을 호출하였을 때의 메모리 상태

하위 프로그램을 실행시키는 절차를 정리하면 다음과 같다.

단계 1. 프로그램이 있는지 확인한다.
프로그램이 있으면 단계 2, 없으면 단계 3으로 간다. 프로그램이 있는지 여부를 확인하기 위해 fopen() 함수를 사용한다. 이 함수는 파일이 없을 경우에는 NULL을 되돌려 주고 파일이 있을 경우에는 파일 포인터를 되돌려 준다.

단계 2. 프로그램을 실행시킨다. 끝.
system() 함수를 사용하여 프로그램을 실행시킨다.

단계 3. 프로그램이 없다는 에러 메시지를 출력한다. 끝.
하위 프로그램들 가운데 선형계획법 해법 프로그램을 실행시킬 경우에는 단계

2.가 좀 더 복잡해진다. 다음은 해법 프로그램을 실행시키는 절차이다.

단계 1. 프로그램이 있는지 확인한다.
프로그램이 있으면 단계 2, 없으면 단계 5로 간다.

단계 2. 프로그램을 실행시킨다.

단계 3. 아무 키나 누를 때까지 현재 화면을 유지한다.
해법 프로그램 자체적으로 출력되는 에러 메시지 등을 사용자가 볼 수 있도록 아무 키나 누를 때까지 잠시 기다린다.

단계 4. 해법 프로그램이 오류에 의해서 비정상적으로 중단되었을 경우에 결과 파일이 정상적으로 만들어지지 않으므로 단계 5로 가지고 않고 끝낸다.
해법 프로그램이 프로그램 수행을 마쳤을 때에 시스템에 전달하는 에러 코드를 LIPED 가 받아서 정상적으로 종료되었는지 비정상적으로 종료되었는지 확인한다.

단계 5. 결과 파일 내용을 화면에 보여준다.끝.
편집기 프로그램을 사용하여 내용을 보여준다. 결과 파일 이름은 모든 해법 프로그램에 대하여 lp.out으로 출력하도록 통일되어 있다.

단계 6. 프로그램이 없다는 메시지를 출력한다. 끝.

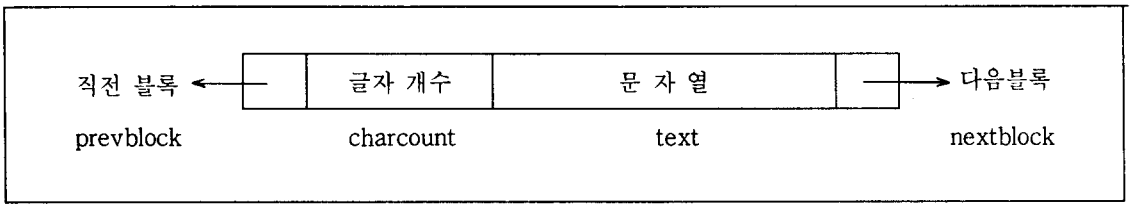
4. 선형계획법 문제 입력을 위한 텍스트 에디터의 개발

선형계획법 문제 입력에 적합한 텍스트 에디터를 개발하기 위해 사용한 자료 구조는 복잡한 이중 연결 리스트[8]이다. 이중 연결 리스트(doubly linked list)는 각 줄마다 메모리 블록(memory block)을 따로 할당하여 이것을 위 아래로 서로 연결해 놓은 자료 구조이다. 그런데 메모리의 효율적인 사용을 위해서 작은 크기의 여러 개 블록을 다시 연결해서 한 줄의 내용을 보관하도록 했기 때문에 복잡한 이중 연결 리스트라고 부른다.

복잡한 이중 연결 리스트를 구성하는 기본적인 단위는 블록(block)과 헤더(header)이다. 블록과 헤더는 다음과 같은 구조체로 되어 있다. 블록 구조체는 다음과 같이 선언되어 있다.

```
typedef struct blockstru {
    int charcount;
    char text[BLOCKSIZE];
    struct blockstru far *prevblock, far
    *nextblock;
} Block;
```

쉽게 이해하기 위해서 다음과 같이 블록 구조체의 개념도를 그릴 수 있다.



[그림 7] 블록 구조체의 구조

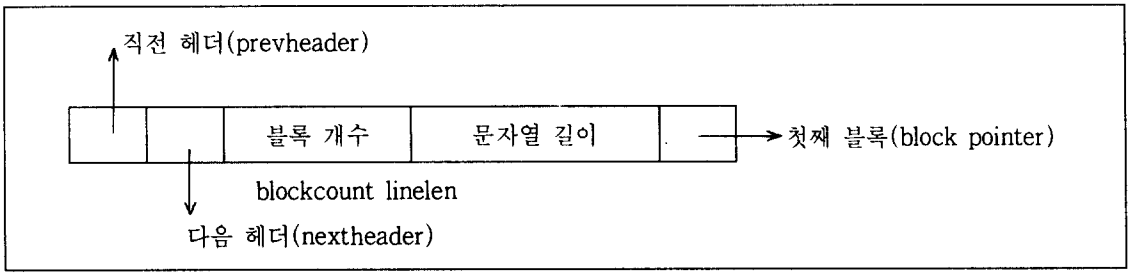
블록은 내용을 보관하는 데에 사용된다. 여러 개의 블록이 연결되어 한 줄의 내용을 보관하기 때문에, 직전 블록을 가리키는 포인터(prevblock)와 다음에 연결된 블록을 가리키는 포인터(nextblock)를 가진다. 그리고 현재 블록에 보관되어 있는 글자 개수(charcount)를 보관하고 있으며 나머지 부분에는 문자열(text)을 보관한다.

한 블록이 40개의 글자를 보관하도록 하였다. 한 줄의 글자 개수가 40개가 넘으면 블록을 하나 더 할당받아서 뒤에 연결한다. 반대로 한 줄의 글자 개수가 줄어들면 뒤에서부터 블록을 하나씩 없애 나간다. 길이가 늘어날 때마다 블록 메모리를 할당받아 연결해 나가면 되므로 길이에 거의 제한이 없다. 또한 작은 블록으로 나누어져 있어서 길이가 짧을 때에 메모리가 낭비되는 것을 최소화하고자 하였다.

헤더 구조체는 다음과 같이 선언되어 있다.

```
typedef struct headerstru {
    int blockcount;
    int linelen;
    Block far *blockpointer;
    struct headerstru far *prevheader, far
    *nextheader;
} Header;
```

헤더 구조체의 개념도는 다음과 같다.



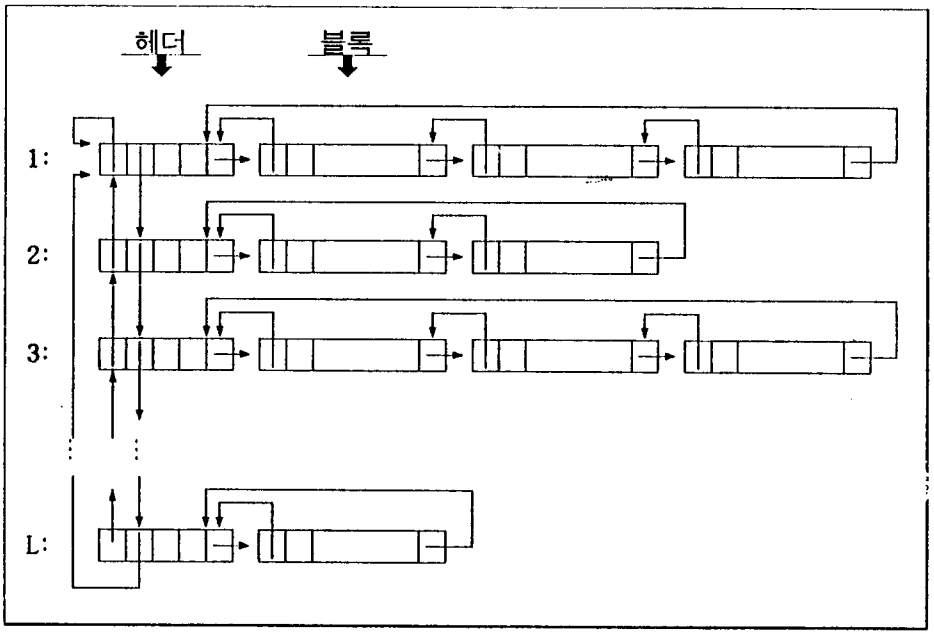
[그림 8] 헤더 구조체의 구조

헤더 구조체는 각 줄을 관리하는 데에 쓰이는 구조체이다. 한 줄의 내용을 보관하기 위해 사용된 블록의 개수와 전체 글자 개수를 보관하고 있다. 각 줄은 헤더를 하나씩 가지고 있고 전체 문서는 헤더들이 연결된 리스트로 되어 있다. 따라서 직전 줄의 헤더를 가리키는 포인터와 다음 줄의 헤더를 가리키는 포인터를 가지고 있으며, 줄의 내용을 보관하는 첫 블록을 가리키는 포인터를 가지고 있다.

텍스트 에디터에서 편집되는 문서는 여러 개의 헤더와 블록으로 구성된다. 모두 L 개의 줄로 구성되어 있다면 헤더의 개수는 L 개가 된다. 그리고 한 줄은 글자의 개수에 따라 적당한 수의 블록으로 구성된다. 한 개의 블록이 40자를 보관하므로 글자 개수/40 개의 블록으로 구성된다. 예를 들어 100자로 이루어진 줄은 1 개의 헤더와 3(= 100/40) 개의 블록이 연결되어 구성한다. 전체 자료 구조의 개념도는 [그림 9]와 같다.

[그림 9]에서 제일 왼쪽에 있는 상자들이 헤더이고, 헤더 오른쪽에 연결되어 있는 상자들이 블록이다. 각 헤더는 L 번째 줄처럼 최소한 하나 이상의 블록을 가지고 있다.

첫 줄을 가리키는 첫 번째 헤더의 직전 헤더를 가리키는 포인터는 자기 자신을 가리키며 마지막 줄에 해당하는 헤더의 다음 헤더를 가리키는 포인터는 첫 번째 헤더를 가리킨다. 아래 줄로 내려가기 위해 다음 헤더를 찾았을 때에 첫 번째 헤더를 가리키면 마지막 줄인 것을 알게 된다. 블록



[그림 9] 복잡한 이중 연결 리스트의 전체 자료 구조

에서도 비슷하게 마지막 블록의 다음 블록을 가리키는 포인터가 헤더를 가리키게 되면 마지막 블록이라는 뜻이다. 편집 도중에 한 줄이 추가되면 헤더와 블록을 새로 할당받아 위의 구조에 삽입해 넣게 되며 반대로 한 줄이 지워지면 헤더와 블록의 메모리를 해제한다. 이 때에 직전 헤더를 가리키는 포인터와 다음 헤더를 가리키는 헤더의 값들이 수정된다. 이와 같이 자료 구조가 헤더 구조체와 블록 구조체로 구성되어 작은 단위의 메모리 블록의 할당과 반환, 이동으로 편집 작업이 이루어지기 때문에 편집 속도가 매우 빠르다.

텍스트 에디터의 작업은 글자의 삽입 및 삭제로 이루어진다. 복잡한 이중 연결 리스트의 자료 구조로 만들어진 텍스트 에디터에서 글자의 삽입과 삭제는 다음과 같은 방법으로 이루어진다. 텍스트 에디터에서 편집 도중 글자가 삽입될 때에 가능한 상황은 다음과 같다.

① 새로운 줄이 추가되는 경우: 사용자가 〈Enter〉를 눌렀을 때에 새로운 줄이 추가된다. 줄은 헤더에 의해 관리되므로 헤더가 하나 필요하고, 내용을 보관하기 위해서는 블록이 필요하다. 따라서 헤더를 하나 할당받아서 추가하고 또 블록을 할당받아서 헤더에 연결시킨다. 할당받은 헤더는 추가될 위치의 앞과 뒤의 헤더에 포인터로 연결시킨다.

② 블록에 아직 여유 공간이 있는 경우: 현재 블록 위치에 그대로 글자를 삽입시킨다. 여유 공간이 있으므로 현재 위치 뒤에 있는 글자들은 블록 내에서 하나씩 뒤로 이동시킨다.

③ 블록에 더 이상 여유 공간이 없는 경우: 글자를 현재 위치에 삽입한 다음, 글자들을 블록 내에서 하나씩 뒤로 이동시킨다. 블록 내에 모두 들어가지 않을 경우에 다음 블록에 글자를 삽입하고 다시 그 블록 내에서 글자들을 하나씩 뒤로 이동시킨다. 이것을 반복하다가 더 이상 블록이 없으면 새로 블록을 하나 할당받아서 헤더에 기

록하고 글자를 삽입한다.

한편 텍스트 에디터에서 편집 도중 글자가 삭제될 때에 가능한 상황은 다음과 같다.

① 현재 블록에 아직 글자들이 남아 있는 경우 : 삭제하려는 글자를 지우고 현재 위치 뒤에 있는 글자들을 하나씩 앞으로 이동한다.

② 현재 블록에 더 이상 글자들이 없는 경우: 글자가 삭제된 다음 더 이상 그 블록에 보관할 글자가 없으므로 블록의 메모리를 시스템에 반환하고 헤더에 블록이 하나 줄었음을 기록한다.

③ 헤더에 하나 남은 블록에 더 이상 글자들이 없는 경우: 더 이상 줄에 글자가 없고 줄을 지우게 되면 그 때에 헤더와 블록을 시스템에 반환한다. 그러나 첫 줄일 경우에는 헤더와 블록을 반환하지 않는다.

5. 수식 해석기의 개발

수식 형태 방식이 문제의 구조를 한 눈에 알아볼 수 있을 뿐 아니라 비영 요소에 대해서만 입력이 필요하므로 간단하다. 또한 선형계획법을 처음 배우는 초보자의 경우 교과서의 문제 표현 형태와 거의 비슷하기 때문에 쉽게 입력 방법을 익힐 수 있어서 교육용 선형계획법 자료의 경우 특히 수식 형태 방식이 유리하다.

본 연구에서는 사용자가 수식 형태로 선형계획법 문제를 입력하면 입력된 내용을 해석해서 선형계획법 해법 프로그램이 요구하는 형태의 파일로 바꾸어 주는 프로그램을 개발하였다. 수식 해석기가 출력하는 파일의 형태는 MPS 형태와 행단위 형태, 열단위 형태이다. MPS 형태는 IBM이 발표한 이래 많은 상용 프로그램들이 지원하고 있는 표준적인 파일 형태이고 행단위 형태와 열단위 형태는 본 연구에서 개발한 통합환경 프로그램에서 하부 프로그램으로 접속하여 사용하는 프로그램들이 사용하는 고유한 파일 형태이다. 행단위 형태는 선형계획법 문제의 행렬 계수를 행 우선 형태의 행렬로 입력하는 방법이고 열단위 형태는 행렬 계수의 비영 요소를 열 우선 형태로 입력하는 방법이다.

수식 형태 입력은 책에서 사용되는 일반적인 선형계획법 문제의 정형화된 형태를 따른다. 따라서 다음과 같은 순서로 문제를 입력한다.

단계 1. 목적 함수의 형태(Max or Min)를 입력한다.

단계 2. 목적 함수를 입력한다.

단계 3. 제약식 기호(s.t.)를 입력한다.

단계 4. 제약식들을 입력한다.

목적 함수의 형태가 가장 먼저 입력된다. 일반적인 표현 방법대로 최대화 문제일 경우에는 'Max'를 입력하고, 최소화 문제일 경우에는 'Min'을 입력한다. 'Max', 'MAX', 'max' 등이 모두 동일한 문자열로 인식되도록 대소문자의 구별을 없애고, 변수 이름이나 다른 예약어에 대해서도 대소문자 구별이 없도록 하였다.

목적 함수의 형태를 정의한 다음에 목적 함수를 입력한다. 목적 함수는 여러 개의 항이 덧셈이나 뺄셈 기호로 연결된다. 하나의 항은 계수와 변수로 이루어진다. 계수와 변수 사이는 띄어 쓰기도 하고 붙여 쓰기도 한다. 수식 해석기의 수식 형태에서도 이러한 규칙이 그대로 적용되었으며 단 변수 이름은 반드시 알파벳으로 시작되어야 하고 변수 이름에 공백 문자가 섞이면 안되도록 하였다. 변수 이름이 숫자로 시작될 경우에 계수와 변수를 구별하는 것이 불가능해지기 때문에 반드시 알파벳으로 시작되도록 정하였다.

일반적으로 제약식에 대한 기호로 's.t.' 또는 'subject to'를 사용한다. 수식 해석기에서는 제약식을 입력할 때에는 'ST' 또는 'Subject to'로 제약식 입력이 시작되었음을 알려 주도록 하였다.

이 경우에도 역시 대소문자 구별이 없으므로 사용자가 입력하기 편한 모양으로 입력해도 상관없다.

선형계획법 문제의 제약식에서 부등호는 '≤' 또는 '≥'의 형태를 가지게 되는데, 텍스트 에디터에서 이들 기호를 입력하기란 사실 매우 어렵다. 따라서 '≤'는 '<'로 대신하였고 '≥'는 '>'로 대신하였다. 모든 제약식은 반드시 '<', '>', '=' 가운데 한 가지 형태를 취해야 한다. 제약식을 입력하는 요령은 목적 함수를 입력할 때와 같고 다만 제약식 형태와 우변 상수 값이 입력되는 것만 다르다. 제약식 형태가 입력되는 것으로 제약식이 끝났음을 인식하기 때문에 제약식이 길어질 경우에는 여러 줄에 걸쳐서 입력할 수도 있다. 다만 한 줄에 여러 개의 제약식을 기술할 수는 없도록 하였다.

모든 제약식의 입력이 끝나면 'END'를 자료의 맨 끝에 써 주어서 선형계획법 문제 자료의 모든 입력이 완료되었음을 표시해 주어야 한다.

[그림 10]은 교과서의 선형계획법 문제를 수식 해석기에서 사용하는 수식 형태로 표현한 예이다. 아래 예에서 보는 바와 같이 계수가 1일 경우에는 계수는 생략하고 변수 이름만을 쓸 수 있도록 하였다.

교과서 문제	수식 형태 입력 내용
$Max\ 40x_1+50x_2+80x_3+110x_4$ $s.t\ x_1+x_2+x_3+x_4 \le 15$ $3x_1+5x_2+10x_3+15x_4 \le 105$ $7x_1+5x_2+3x_3+2x_4 \le 120$ $x_j \ge \forall_j$	MAX 40X1+50X2+80X3+110X4 ST X1+X2+X3+X4<15 3X1+5X2+103X+15X4<105 7X1+5X2+3X3+2X4<120 END

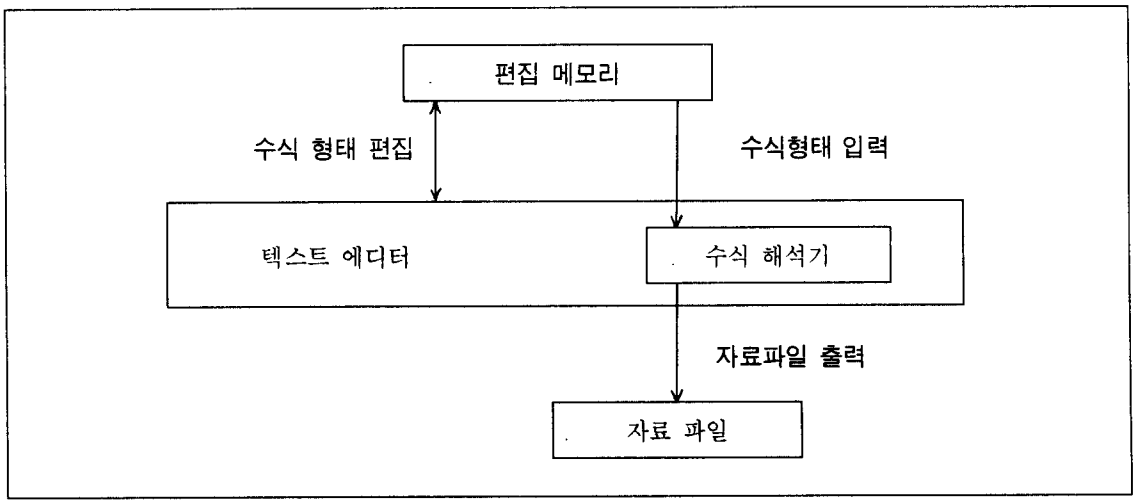
[그림 10] 수식해석기에서 지원하는 수식 형태의 예

자료를 입력할 때에 문제 자체뿐만 아니라 필요한 도움말을 함께 기록해야 할 필요가 있다. 따라서 문제의 이름과 문제에 대해 참고 사항을 기록할 수 있도록 타이틀과 주석문을 지원하였다. 타이틀은 자료의 이름을 정해 주는 문장이다. 자료를 입력할 때에 맨 처음에 'TITLE'을 친 다음 한 칸을 공백을 남기고 다음 칸에 자료의 이름을 입력하면 된다. 자료의 이름은 MPS 형태로 파일을 만들 때에 MPS의 'TITLE'로 사용된다. 입력한 내용에 설명이 필요할 경우에 주석문을 작성한다. 주석문은 '!'로 시작하며 그 줄에서 '!' 이후의 문자열은 모두 주석문으로 인식되어 수식 해석기가 입력된 내용을 해석할 때에 무시하고 넘어간다. 앞에서 예로 들었던 문제에 대해 타이틀문과 주석문을 다음 [그림 11]과 같이 사용할 수 있다.

텍스트 에디터에서 수식 형태로 자료를 입력하면 수식 해석기가 그 내용을 해석해서 선형계획 해법에서 사용하는 자료 형태로 바꾼다. 따라서 수식 해석기가 텍스트 에디터의 메모리 내용을 읽어 들일 수 있어야 하기 때문에 수식 해석기는 텍스트 에디터에 소스 파일 수준에서 연결된다. [그림 12]는 수식 해석기가 텍스트 에디터에 접속되어 작동되는 원리를 보여 준다.

```
! This is an example of arithmetic-formed input
TITLE EXAMPLE1
MAX 40X1+50X2+80X3+110X4
ST
        X1+X2+X3+X4<15
        3X1+5X2+10X3+15X4<105
        7X1+5X2+3X3+2X4<120
END
```

[그림 11] 주석문과 타이틀문을 사용한 예



[그림 12] 수식해석기의 접속 및 작동 원리

텍스터의 편집 메모리에서 한 줄씩 수식 해석기의 임시 버퍼에 읽어 들인 다음 수식 해석기가 해석을 수행한다. 전체 내용을 모두 해석한 다음에는 그 결과를 사용자가 지정한 형태의 파일로 출력한다. 따라서 텍스트 에디터의 자료 구조에 따라서 수식 해석기의 접속을 위해 수정이 필요한 부분은 텍스트 에디터의 메모리에서 한 줄씩 임시 버퍼에 읽어 들이는 부분(수식 해석기 프로그램에서 하나의 함수로 되어 있다.)이다. 나머지 부분은 임시 버퍼에 읽어 놓은 내용을 가지고 처리되기 때문에 수정할 필요가 없다. 텍스트 에디터마다 줄, 칸에 대한 정보를 보관하고 있고, 편집이 이루어지는 중요한 단위 가운데 하나가 줄이기 때문에 자료 구조의 복잡한 정도에 상관없이 한 줄씩 읽어 들이는 것은 간단한 작업이므로 수식 해석기를 텍스트 에디터에 접속하는 것은 쉽게 해결될 수 있다.

수식 해석기가 수행되는 절차는 단계별로 살펴보면 다음과 같다.

단계 1. 텍스트 에디터로 문제를 입력받는다.

수식 해석기가 연결된 텍스트 에디터에서 문제를 입력받는다. 이 때에 문제는 수식 해석기에서 정의된 문법을 따라서 입력해야 한다.

단계 2. 텍스트 에디터의 메모리에 있는 내용을 해석한다.

수식 해석기는 텍스트 에디터의 메모리에서 한 줄씩 내용을 읽어 들여서 문제를 만들어 나간다. 목적 함수가 첫 줄에 들어가고 제약식이 그 다음에 들어간다. 마지막 줄까지 해석이 끝나면 각 열에서 비영 요소의 순서를 행 번호가 작은 순서대로 정렬한다.

단계 3. 파일로 문제를 출력한다.

MPS 형태 또는 행 우선 형태나 열 우선 형태의 파일로 출력한다. 사용자는 현재 수식 해석기가 연결된 텍스트 에디터를 통해서 만들어진 결과 파일을 읽어 들여 그 내용을 볼 수 있다.

6. 결론

본 연구에서는 선형계획해법 통합환경 소프트웨어를 설계하고 구축하였다. 선형계획해법 통합환경 소프트웨어 LIPED는 자료 입력 기능, 문제 풀이 기능, 실험용 문제 생성 기능 및 자료 변환 기능을 가지고 있다. LIPED는 사용자 인터페이스와 프로그램의 실행 및 관리를 담당하는 메뉴 프로그램과 텍스트 에디터, 해법 프로그램, 문제 생성 프로그램, 자료 변환 프로그램으로 구성되어 있다. 해법 프로그램은 교육용과 실무용으로 나누어서 선형계획법의 교육뿐만 아니라 일반 실무에도 사용할 수 있도록 하였다.

LIPED의 메뉴는 전체 메뉴 구조를 볼 수 있어서 사용하기에 편리한 풀다운 메뉴와 계단식 메뉴를 혼합한 형태를 사용하였다. 또한 이와 같은 메뉴 형태는 LIPED가 지원하는 다양한 해법들을 체계적으로 정리, 분류할 수 있어서 사용자가 원하는 해법을 쉽게 선택할 수 있다. 선형계획해법 통합환경에서 사용자 인터페이스가 가장 활발하게 일어나는 자료 입력 부분의 성능을 향상시키기 위해서 텍스트 에디터와 수식 해석기를 개발하였다. 텍스트 에디터는 선형계획법 문제 자료를 편집하기에 편리하도록 줄의 길이에 제한이 없도록 설계하였다. 또한 선형계획법을 처음 배우는 사람도 쉽게 자료를 입력할 수 있도록 수식 해석기를 개발하였다. 수식 해석기는 텍스트 에디터에 접속하여 사용자가 수식 형태로 자료를 입력하면 해법 프로그램이 사용할 수 있는 자료로 변환해 주도록 하였다.

참고문헌

[1] 권재락, "한글 문서 편집기를 만들어 보자 1", *Program World,* 1994. 6

[2] 김우제, 지식기반을 활용한 선형계획법 지원 시스템의 개발에 관한 연구, 서울대학교 공학박사 학위논문, 1994

[3] 박순달, 「선형계획법(3정판)」, 민영사, 1992

[4] 서울대 산업공학과 체계분석실, 「LIPED version 1. 3 사용자 안내서」, 1994

[5] Borland International, *Borland C++ Version 3. 1 Programmer's Guide,* 1991

[6] Orden, Alex, "LP from the '40s to the '90s", *INTERFACES,* vol. 23, no. 5, Sep-Oct 1993, pp. 2-12

[7] Sharda, Ramesh, "Linear Programming Software for Personal Computers: 1992 Survey", OR/MS Today, JUN 1992, pp. 44-60

[8] Smith, P. D., An Introduction to *TEXT PROCESSING,* the MIT Press, 1990