GOVP1199107801

제 1 차년도
년차보고서

67/150

소프트웨어 자동생산기술 국책과제

National Project for Automated Software Development Technology

소프트웨어 테스팅 도구 개발

Development of Software Testing Tool

연 구 기 관
한국과학기술연구원
시스템공학연구소

과 학 기 술 처

배 포 선

사본번호	부 수	배 포 선
1/150	1	시스템공학연구소 영구보존용
2/150	1	시스템공학연구소 도서관 보관용
3/150 - 6/150	4	시스템공학연구소 연구개발과 보관용
7/150 - 9/150	3	과학기술처
10/150 - 88/150	79	기타 배포기관
89/150 - 150/150	62	시스템공학연구소 해당연구실

제 출 문

과학기술처장관 귀하

본 보고서를 "소프트웨어 자동생산기술 국책과제"의 세부과제 "소프트웨어 테스팅 도구 개발"의 년차 보고서로 제출합니다.

1991 년 7 월 19 일

총괄연구책임자 : 이 단 형
세부과제연구책임자 : 조 일 순
선임연구원 : 유 인 철
이 영 남
연구원 : 윤 화 묵
조 태 중
김 지 수
이 제 선
장 재 명
이 공 선
김 선 주
김 동 수
이 준 석
신 석 규
이 윤 석
김 송 희
최 청 림

여 백

요 약 문

Ⅰ. 제 목

: 소프트웨어 테스팅 도구 개발

Ⅱ. 연구 개발 목적 및 중요성

복잡한 대규모 소프트웨어의 수요 증가에 따라 소프트웨어 품질과 생산성 향상을 위한 생산 기술의 적용과 체계적 품질 관리의 필요성이 증대되고 있다. 소프트웨어 공학 기법의 적용은 생산성 및 품질 향상에 많은 기여를 하였으나, 전반적으로 표준 개발 방법론 및 절차의 체계적 적용은 아직 정착되지 못하고 있는 상황이다. 이것은 소프트웨어 공학 기법을 지원하는 도구의 보급과, 지원 기능이 미흡한 점에 주로 기인하며, 미국의 경우 이를 해결하기 위해 80년대 중반부터 추진된 공학 도구 개발은 실용화 단계에 이르렀으나 국내의 경우 연구, 도구의 개발, 적용이 초보적인 단계이다.

소프트웨어의 품질과 품질 정보에 대한 신뢰성의 요구가 증가되면서 시험에 대한 중요성이 고조되고 있으나 시험 과정 역시 위와 같은 문제로 신뢰성 및 생산성 문제가 제기되고 있다.

따라서 본 연구를 통하여 시험 절차와 문서에 대한 표준을 제시하고, 시험 활동의 각 작업을 체계적으로 지원하며 그 정보를 관리하는 시험 지원 시스템을 개발하여 시험 과정의 생산성을 향상시키고, 시험의 철저화를 기하고자 한다.

Ⅲ. 연구개발의 내용 및 범위

1. 시험 지원 도구 및 기술 현황 분석

- 국외에서 개발된 시험 절차 지원기 및 범주 분석기의 기능 분석
- 범주 분석기인 TCAT, 리그레션 시험 지원기인 DCAT, Ada 언어의 범주 분석기인 ATVS 등의 기능 분석
- 요구 명세로부터 시험 사례를 작성하는 T와 산출물로부터 시험 사례를 생성하는 SILOP 등의 기능 분석

2. 시험 절차 관리기의 설계 및 구현

시험 설계 및 수행 작업을 체계화하고 시험 정보를 관리하는 절차 관리기의 기능을 설계, 구현

- 주요 기능
  - 시험의 완전성 점검
  - 시험 항목 및 기능 관리
  - 시험 사례 및 절차 관리
  - 시험 오류 및 로그 관리
  - 시험 수행 상황 및 오류 발생 추이 분석

3. 시험 결과 처리기의 설계 및 구현

- 범주 측정 계기 삽입기 구현
  - 통계자료 및 기본속성 저장
  - 코볼 원시 프로그램내 계기 삽입
- 시험 범주 분석기 구현

• 미시험 코드 및 최초 수행 코드 지적

• 시스템 및 프로그램별 시험 범주 분석

- 사용자 접속기능 구현

• 한글 지원

• 메뉴에서 시험 설계, 정적 분석, 수행 및 결과 분석이 가능

• 시험 정보 조회 및 편집

4. 정적 품질 측정 시스템의 적용

시범 단계에서 개발된 품질 측정 및 설계복구 시스템을 국내 기업에 적용하여 활용 가능성과 유용성을 분석

Ⅳ. 연구 개발 결과 및 활용에 대한 건의

1. 연구 개발 결과

- 년차 보고서

- 시험 결과 처리기 및 절차 처리기

2. 활용에 대한 건의

- RM COBOL, LPI COBOL등 중·소형 시스템에 1단계 적용

- 1단계 적용 결과를 분석하여 도구 기능 보완

- 대형 시스템 사용자용 도구 개발 프로젝트화 및 2단계 대형 시스템 적용

여 백

SUMMARY

I. Title

Development of Software Testing Tool

II. Objectives and Scope of the Study

The rapid increase of large complex software development requires the applications of automated development technology and software quality assurance. With the application of software engineering, software development productivity and quality improved a lot, but the general situation is that the systematic application of standard development methodology and procedures is not settled in the industry yet. One of the major reasons of this situation is that automated tools available, and the functions of the tools are limited. To solve these problems CASE tool development started since 80's. As the results, many CASE tools are applicable to industry now. Our situation is that research, development and application of CASE tools are in preliminary stage. Testing process has the same problems mentioned above. Through this study we are to develop standard testing procedures and documents and, to develop automated testing system that supports testing activities systematically and manages testing information. With developed system we expect to improve productivity of testing process and achieve completeness of testing.

III. Contents and Scope of Study.

1. Analysis of existing automated tools and technology status

   - Coverage analyzers; TACT/COBOL, ATVS for Ada

   - Test driver and regression test system: DCAT

   - Test case generator; T, SILOP

2. Design and Implementation of Test Procedure Management System

   - Major functions

     . Verify completeness of testing

     . Manage test item and features to be tested

     . Manage test case and procedure

     . Manage test incident report and test log

     . Analyze test incident and trend of testing process

3. Design and implementation of Test Coverage Analyzer

   - Source Code Instrumenter

   - Test Coverage Analyzer

     . Identify untested code

     . Identify newly tested code

     . Analyze coverages for system

     . Analyze coverages for program

   - User Interface System

     . Hangul support

     . Design and execute test and analyze test results through

menu system

. Query and edit test information

4. Application of tools developed

- Quality Evaluation and Design Recovery System

IV. Results of the Study

- Test Procedure Management System

- Test Coverage Analyzer

- Annual Report

여 백

CONTENTS

여 백

목 차

여 백

제 1 장　서　　론

여 백

제1장 서 론

제1절 연구개발의 필요성

현대 정보화 사회에서 복잡한 대규모 소프트웨어 개발 요구가 급증하고 있다. 이러한 소프트웨어 개발을 관리하여 생산성을 향상하고, 품질을 실현하기 위해서는 과학적이며 자동화된 생산 기술의 적용과 함께, 고품질 실현을 위한 체계적인 품질 관리가 이루어져야 한다. 지난 20년간 소프트웨어 공학 분야에서 많은 연구가 이루어졌다. 이를 기반으로 한 개발 공정, 명세화, 시험 등의 공학 기법 적용은 소프트웨어의 품질과 생산성 향상에 기여하였으나, 전반적인 실태를 살펴보면 표준 개발 방법론 및 절차의 체계적 적용이 아직 정착되지 못하고 있는 상황이다. 이것은 소프트웨어 공학 기법을 체계적으로 지원하고 수작업의 노력을 감소할 수 있는 도구의 보급과 지원이 미흡한 점에 주로 기인한다고 할 수 있다. 소프트웨어 공학 지원 자동화 도구의 개발은 80년대 중반부터 본격적으로 추진되어, 미국의 경우 실용화 단계에 이르렀으며, 도구를 사용한 생산성 향상 결과가 많이 보고되고 있다. 반면에 국내의 소프트웨어 공학 연구와 적용은 최근 몇년전 부터 시작되어 아직 초보단계이고, 더우기 소프트웨어공학도구는 거의 전무한 상태이다.

소프트웨어의 품질과 품질 정보에 대한 신뢰성 요구가 증가되면서 시험에 대한 중요성이 고조되고 있다. 시험 과정 역시 대부분 수작업으로 수행되고 있으며, 체계적인 기법 적용과 결과 검증이 미흡하

여 시험 자체의 신뢰성 문제가 제기되고 있다. 또한 개발 및 유지 보수 과정에서 반복 사용될 수 있는 시험 정보의 재사용이 이루어지지 않아 많은 비용이 낭비되고 있다. 자동화 도구를 살펴보면 요구분석이나, 설계단계에 비하여 그 수가 적으며, 시험 절차를 체계적으로 지원하는 기능과 사용 용이성 측면에서 미흡한 점이 많다.

따라서 본 연구를 통하여 시험 절차와 문서에 대한 표준을 제시하고, 시험 활동의 각 작업을 체계적으로 지원하고 그 정보를 관리하는 시험 지원 통합 시스템을 개발하여 시험 과정의 생산성을 향상시키고, 시험의 철저화를 기하고자 한다.

제 2 절 연구개발의 목적과 범위

1. 연구개발의 최종 목표

프로그램에 대한 품질을 평가하고, 동적 시험을 지원하며 요구분석, 설계단계와 통합된 기능을 수행하는 시험 지원 시스템의 개발

- 도구는 다음과 같은 기능을 지원한다.
  - 산출물에 대한 품질 측정과 오류 진단
  - 코드로 부터 설계 복구
  - 시스템 제어구조 생성에 의한 구조적 사례 생성 지원
  - 시험 설계 및 수행 결과의 입력과 조회, 각종 통계 제공
  - 시험 범주 분석
  - 시험 진행을 통제하기 위한 정보 제공
  - 시험 구동기를 이용한 동적 시험의 자동화

• 도구가 갖추어야 할 특성은 다음과 같다.

- 각 구성 요소는 독립적인 단위 기능을 수행하며, 각 요소가 통합되어 통합 시스템으로 운영이 가능하다.
- 시험 설계 정보의 재이용 환경을 구축한다.
- 품질 정보 및 시험 수행 정보를 기록하고 관리한다.
- 도구의 사용을 통해 시험 과정의 체계화를 기할 수 있다.
- 기능과, 코드의 시험 완전성을 점검할 수 있다.
- 사용법과 제공 결과의 해석이 용이하다.

2. 연차별 연구개발 목표 및 내용

연차별	연구개발목표	연구개발 내용 및 범위	연구비
1차년도 (1990)	-테스트 절차 관리기 개발 -테스트 결과 처리기 개발	-효율적 시나리오의 작성 -테스트 시나리오의 재사용 -결과 분석 정보 제공 -수행 통계 자료 제공 • 커버리지 분석 • 프로그램 실행 순서	97,000
2차년도 (1991)	-정적, 동적 시험 지원 도구 실용화 -시험 가능한 명세화 기법 연구 -설계 단계 품질 평가 기법 연구	-품질 측정 시스템 -문서 생성 시스템 -명세서에서 시험 기능과 사례 추출 -결합도, 응집도 측정	
3차년도 (1992)	-개발 단위 지원 도구 통합	-요구분석, 설계 단계 연계 -개발 단계별 품질 측정	

3. 연구개발 내용 및 범위

가. 시험 지원 도구 및 기술 현황 분석

국외에서 개발된 시험 절차 지원기 및 범주 분석기의 기능을 분석한다. 동적 시험 지원기로서, 미국에서 개발된 범주 분석기인 TCAT, 리그레션 시험 지원기인 DCAT, Ada 언어의 범주 분석기인 ATVS 등의 기능을 분석한다. 시험 사례 생성 지원기는 요구 명세로부터 시험사례를 작성하는 T와 산출물로부터 시험 사례를 생성하는 SILOP등의 기능을 분석한다.

나. 시험 절차 관리기의 설계 및 구현

시험설계 및 시험 수행 작업을 체계화하고 시험 정보를 관리하는 절차 관리기의 기능을 설계하고 구현한다.

시험 절차 관리기는 다음과 같은 주요 기능을 가진다.

- ㅇ 시험 항목 관리
  - 항목 등록 및 조회
- ㅇ 시험 기능 관리
  - 단계별 시험 기능의 등록 및 조회
  - 시험 기능과 사례간의 전후 참조 제공
  - 시험 기능 및 미시험 기능 출력
  - 기능 시험 추이 그래프 출력
- ㅇ 시험 사례 및 절차 관리
  - 시험 사례의 작성 및 조회
  - 시험 절차의 작성 및 조회
  - 시험 사례의 작성 상황과 수행 상황 출력

- 사례 수행 추이 그래프 출력

○ 시험 오류 관리

- 시험 문제 발생 보고서 작성 및 조회

- 시험 오류 분석 및 통계

○ 시험 로그 관리

- 시험 로그의 입력 및 조회

- 시험 요약 보고서 생성

다. 시험 결과 처리기의 설계 및 구현

코볼을 지원하는 범주 분석기와 범주 분석을 위한 계기 삽입기를 설계, 구현한다. 시험 범주 분석기는 시스템내의 모듈, 모듈 호출관계, 프로그램내의 각 모듈과 분기등의 범주 출력을 제공하며, 계기 삽입기는 프로그램의 수행시 추적 결과를 기록할 수 있도록 원시 프로그램내에 호출문 등을 삽입한다.

시험 결과 처리기는 다음과 같은 주요 기능을 가진다.

○ 범주 측정 계기 삽입

- 통계자료 및 기본속성 저장

- 코볼 원시 프로그램내 계기 삽입

○ 범주 분석기

- 미시험 코드 및 최초 수행 코드 지적

- 시스템내 프로그램 범주

- 시스템내 호출관계 범주 등

- 프로그램내 레이블, 분기, 문, 조건 범주 측정

- 범주간 연계 조회 기능 등

o 사용자 접속기능

- 한글 지원
- 메뉴에서 시험 설계, 수행, 결과 분석이 가능
- 주요 기능
  - 시험 정보 조회 및 편집
    시험 항목, 시험 기능, 시험 사례 및 절차, 오류 등
  - 선택 정보의 조작
  - 계기삽입, 정적 분석, 컴파일, 실행
  - 선택
    - 디렉토리, 옵션, 초기값 지정
    - 계기 삽입 범주 지정
    - 화일 초기화
  - 도움말 및 화일 통신

라. 정적 품질 측정 시스템의 적용

시범 단계에서 개발된 품질 측정 및 설계 복구 시스템을 국내기업에 적용하여 활용 가능성과 유용성을 분석하였다.

제 2 장　기술 현황 분석

여 백

제2장 기술 현황 분석

제1절 소프트웨어 개발 단계와 시험 활동

소프트웨어 시스템의 개발단계는 다음과 같다.

- 프로젝트 개시 단계
- 요구 분석 단계
- 설계
- 구현
- 시험
- 설치 및 운영 단계

시험의 관점에서 각 단계별 행위를 정의하면 [표 2-1]와 같고 이를 도식화 하면 [그림 2-1]로 표현할 수 있다.

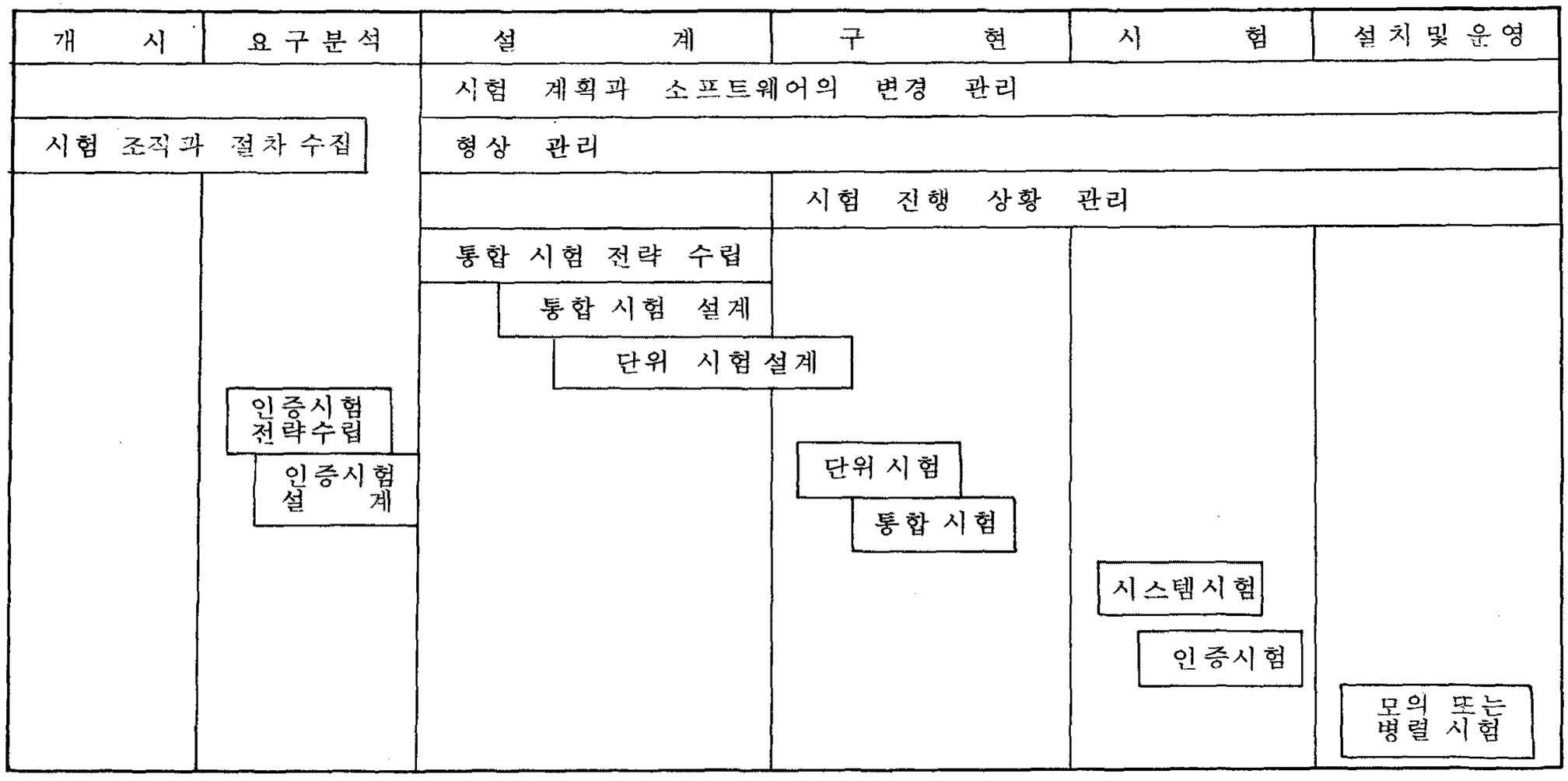
시험은 단위시험, 통합시험, 시스템시험 및 인증 시험의 4단계[시89]로 진행된다.

1. 단위 시험

단위 시험은 상세 설계 명세서를 근거로 모듈 단위의 기능, 성능 및 구조를 검사한다. 단위 시험의 설계 방법은 구현된 시스템의 구조를 고려하지 않고 기능을 시험하는 블랙 박스 방법과 시스템의 구조를 이용하여 설계하는 화이트 박스 방법이 있다. 블랙 박스 방법에는 동등 분할법, 경계값 분석법, 인과 관계 도표법 등의 기법이 사용된다. 화이트 박스 방법인 경우 시스템의 모든 경로를 시험하기는

[표 2-1] 개발 단계별 시험 활동

- 프로젝트 개시 단계 • 개괄적 시험 추진 전략의 정의 • 전반적 시험 접근 방법 정의
- 요구 분석 단계 • 시험 요구사항 정의 • 시험 도구의 개발 또는 확보 방안 수립 • 종합 시험 계획서 작성 • 시험 기능의 책임 정의 • 요구 사항의 검증 및 확인
- 설계 • 상세 시험 계획서 작성 • 설계의 확인 검증
- 구현 • 시스템 시험 계획서 완료 • 단위 시험 수행 • 통합 시험 수행
- 시험 • 시스템 시험 수행 • 시험 사후 분석 • 인증 시험
- 설치 및 운영 단계



[그림 2-1] 개발 단계별 시험 활동

불가능하므로 적절한 기준에 의해 사례를 설계하는 데 통상적으로 제어흐름에 근거한 분기 기준, 조건 기준 또는 문 기준이 많이 사용되며, 자료의 정의 참조 관계를 이용한 범주 기준[시 90]도 연구되고 있다.

2. 통합 시험

단위 시험이 완료된 후 모듈들은 결합되어 각 단위간의 인터페이스와 통합된 단위의 기능 수행 부분을 시험한다. 통합 방법은 하향식과 상향식, 혼합식이 있으며, 상향식 통합은 잘 정의된 소규모의 시스템에, 하향식은 시스템의 골격을 미리 구현해야할 필요가 있는 시스템에 적용한다. 새로운 알고리즘과 주요 기능 모듈 및 입출력 모듈에 대해 상향식으로, 그리고 시스템의 원형을 조기에 구현할 필요가 있는 부분에 대해서는 하향식으로 각 방식의 장점을 선택하여 혼합식으로 시스템을 구현하는 것이 가장 바람직하다. 시스템의 통합순서에 따라 개발. 설계 전략 및 시험 과정이 달라짐을 고려하여야 한다.

3. 시스템 시험

시스템 통합이 완료된후 기능, 유용성, 성능, 강도, 안전성, 복구성, 신뢰성등 시스템의 여러가지 품질 인자에 대한 시험을 수행한다.

시스템 시험 완료후 사용자의 운용 가능 여부를 최종적으로 확인하는 인증시험을 한다. 시험 설계 작업은 다음 단계로 이루어지며 각 과정별로 시험 문서가 생성된다. [그림 4-2]

o 시험 항목의 선정

- 시험 항목

o 시험 항목에 대한 시험 기능의 정의

- 시험 설계 명세서

o 각 기능을 시험하기 위한 사례의 설계

- 시험 사례 명세서, 시험 절차 명세서

o 시험 수행, 수행 결과의 분석 및 오류 기록

- 문제 발생 보고서, 시험 일지

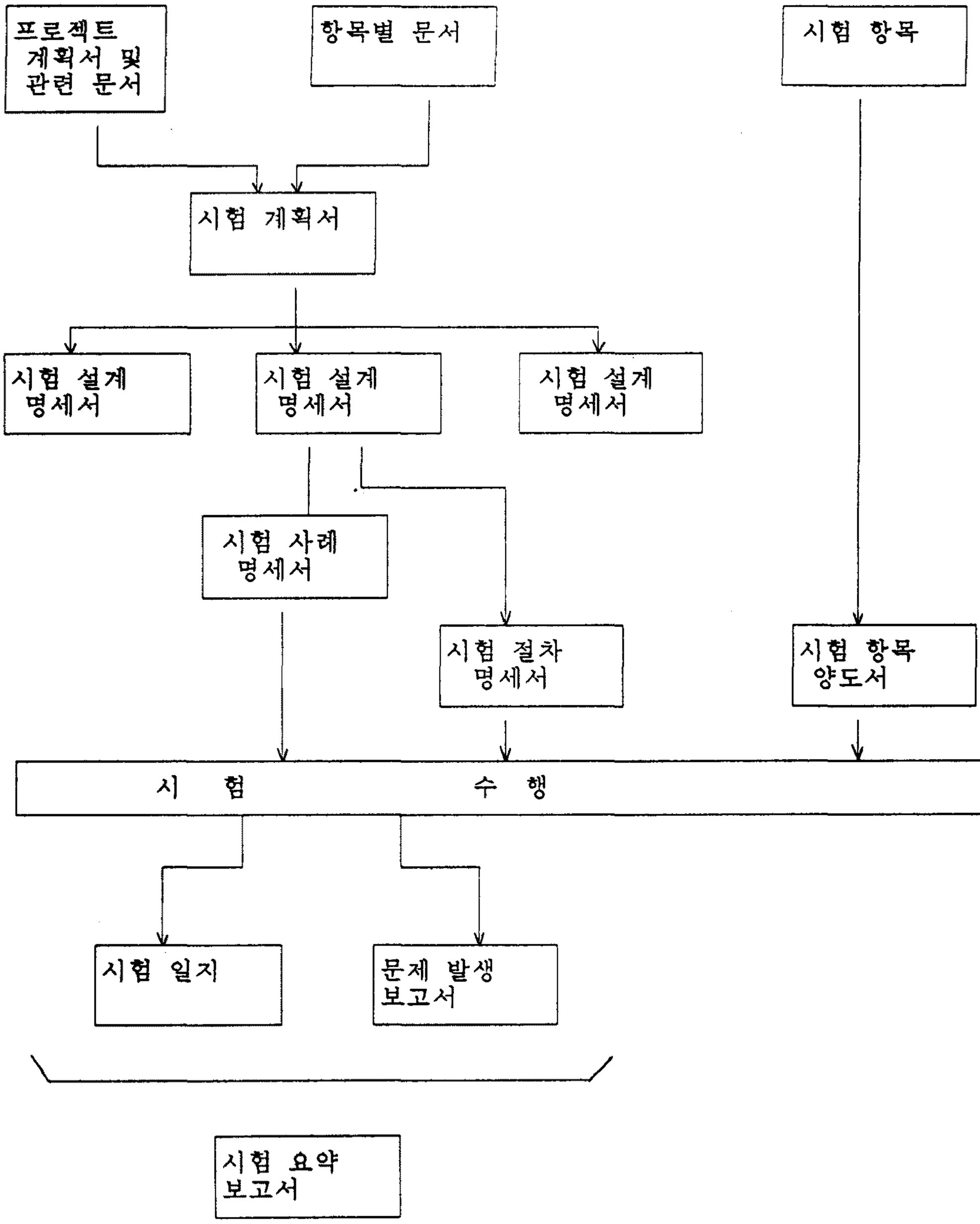
- 시험 요약 보고서

위와 같은 작업 단계는 통합시험과 단위 시험 수행에도 동일하게 적용된다. 시험 수행중에는 문제 발생 상황 및 추이에 대한 분석이 이루어진다. 문제의 분석에 사용되는 주요 문서는 시험 일지 및 시험 문제 발생보고서이다. 문제가 발생하였을 경우 다음과 같은 사항을 검토하여야 한다.

- 오류의 원인 및 만들어진 시점
- 오류를 만든 사람
- 사전에 밝혀지지 않은 이유 및 방지되지 못한 원인 등

위와 같은 분석 결과는 품질 관리 부서 및 관리층에 제공되어 개선 결과를 수립하는 데 이용될 수 있다. 시험 수행중 역시 관리하고 통제할 사항은 시험 계획서상의 업무진행 상황이다. 주요 관리 대상은 다음과 같다.

- 시험 설계 작업 현황
- 시험 사례 명세서 작성 상황
- 수행된 사례의 수 및 시험 완료된 기능의 수 등



[그림 2-2] 시험 단계별로 생성되는 문서

시험이 수행 완료된 경우에는 시험에 투입된 노력과 시험의 효과 그리고 시험 수행 및 운영의 여러 문제점과 잘된점에 대해서 사후 분석을 수행한다.

제 2 절 시험 기법

시험은 품질관리와 V & V의 가장 중요한 기법이다. 소프트웨어의 요구사항이 형식언어로 기술되어 코드로의 자동 변환이 가능하다고 해도, 그것은 이론적으로만 완전한 시스템일 뿐이다. 시험의 중요성에 반하여 대부분의 개발자는 어떻게 주어진 시간과 비용내에서 소프트웨어를 구현할 수 있을 것인가에 대해서만 노력을 집중하고, 시험을 어떻게 할 것인가 하는 문제에 대해서는 많이 생각하지 않는다. 시험에서 가장 중요한 것은 모든 가능한 상황에 대해 전부 시험을 할 수 없다는 제약 조건하에서 어떻게 효과적인 시험을 설계할 것인가라는 문제와 시험의 종결기준(Exit Criteria)의 설정 기준이다.

1. 시험 설계

시험사례의 선정은 크게 소프트웨어의 구조에 의한 선정방법, 하나의 거래(Transaction) 흐름에 따른 방법, 시스템의 논리나 상태 전이의 관계에 의한 방법, 입력 조건에 의한 방법등에 의한다. 구조적 방법에서는 조직의 시험 범주 표준에 의거하여 특정구조를 수행하기 위한 입력 조건과 예상되는 결과를 정의하는 것이다. 거래 흐름에 따른 테스트는 사용자의 관점에서 하나의 거래에 필요한 일련의 작

업을 정의하고 이에 따라 사례를 설계하는 것이다. 위와 같은 경우는 시험 결과의 확인과 함께 시험 범주를 분석하는 작업이 필요하다 시스템의 입력 조건을 형식언어로 기술하여 시험 입력을 생성하는 방법은 생성된 입력에 대한 시스템 응답의 정확성과 실패상황에 이르는 입력 조건을 확인한다. 이 방법은 시스템의 입력조건에 따라 예상결과를 예측하기가 어려우므로 주로 후자의 목적으로 사용된다.

또한 시스템의 기능을 결정 테이블이나 상태전이(State Transition) 등으로 나타내어 이를 시험할 수 있도록 설계하는 방법도 있다. 시험을 어떻게 설계할 것이냐 하는 문제는 시스템의 특성에 따라 결정될 문제로서 조직에서는 이에 따라 사례 설계의 표준을 정의하는 것이 필요하다.

2. 시험의 종결 기준 (Exit Criteria)

시험은 단위시험, 결합시험, 시스템 시험의 단계로 나누어진다. 각 시험의 종결기준은 보통 정해진 시간이 경과하거나 비용이 소요된 경우이다. 그러나 이것은 잠재적 오류가 다음 개발 단계나 유지 보수 단계로 파급되는 문제점을 가진다. 단위 시험의 가장 바람직한 종결 기준은 조직의 설계 표준에 의해 작성된 사례의 수행이 완료되는 시점까지 시험을 수행하는 것이다. 통계적인 오류의 빈도를 고려하여 예상되는 전체 오류의 일정 기준을 검출할 때까지 시험을 수행할 수도 있으나 이것은 시스템의 복잡도, 개발환경, 사례 설계의 품질등에 따라 가변적인 상황이 발생할 수 있으므로 단위 시험에서는 적절하지 않다. 단위 시험에서는 위와 같이 사례 설계 기준에 의해 시험을

수행하고 이 단계로 시험의 범주(Coverage)를 분석하여 미시험 경로에 대해 추가 사례를 만드는 것이 필요하다. 시험 범주를 분석하는 것은 적은 노력으로 아직 밝혀지지 않은 잠재적인 오류를 발견하는 효과적인 방법이다. 이미 시험된 부분의 기능이 확인되었다고 가정할 때 시스템의 품질은 미시험 부분에 의존하는 정도가 높으므로 미시험 부분을 검출하는 것은 매우 중요한 일이다. 미시험 코드가 없도록 하는 것은 단위 시험에서 지켜야 할 최소한의 요건이라고 할 수 있다. 그러나 이러한 시험이 소프트웨어의 구조적 측면을 고려한, 가능한 모든 경로를 시험할 수는 없다. 또한 기능의 누락을 밝혀낼 수 없다는 단점도 있다. 시스템 시험 단계에서는 각 시험의 유형에 따라 여러 점검기능을 정리한 후 각 기능을 시험하기 위해 단위, 결합 시험 사례를 재이용한다. 유형에 따라서 시스템의 운영 표준(Operation Profile)을 준비하여 시험중 실패 상황의 시간별 기록과, 신뢰도 모델을 비교하여 요구되는 신뢰도를 만족하는 시점에 시험을 종결할 수 있다. 시스템의 운영표준을 준비하기 어려운 경우에는, 유사한 시스템의 오류 빈도등을 이용하여 종결기준을 정의하는 방법도 있다. 다음절에서 시스템 신뢰성의 측정 방법을 소개한다.

3. 시스템 신뢰성 측정

소프트웨어 품질 인자중 가장 중요한 성질이 신뢰성이다. 신뢰성은 시간의 함수로서 [t, t0]의 시간내에 시스템이 실패없이 기능을 수행할 수 있는 확률을 말한다. 신뢰성이란 소프트웨어의 잠재 오류에 의존하지만 반드시 오류의 갯수에 비례하는 것은 아니며 시스템의

운영 표준에 의해 결정된다고 할 수 있다. 즉 운영중 거의 수행되지 않는 기능의 오류는 많이 실행되는 기능의 오류보다 신뢰성에 적게 영향을 미친다. 신뢰성을 측정하는 목적은 시스템의 품질을 정량적으로 나타내기 위한 것이며, 이 결과를 사용하여 사용자에 대한 시스템의 객관적 특성 표현과, 개발 과정 및 유지보수 과정에 투입될 노력의 정확한 예측이 가능하다. 시스템의 신뢰성 목표가 제시되었을 때, 시스템의 신뢰성 시험에서는 요구되는 신뢰도를 달성할 때까지 시험을 수행하는 것이 바람직하다.

시스템 신뢰성 시험 수행을 위한 기본 정의를 살펴본다.

- 수행 : 소프트웨어 운영은 일련의 이산적인 수행으로 분해할 수 있다. 각 수행은 입력 자료 집합과, 출력 자료 집합의 사상을 수행하며 일정한 수행 시간을 가진다.
- 실패와 결함 : 수행 결과의 출력이 사용자의 요구사항에 일치하지 않을때 실패 상황이 발생한다. 실패 상황은 프로그램내의 결함에 의해 발생한다.
- 운영 프로필 : 소프트웨어 운영시의 입력을 무작위로 선택한다. 그러나 어떤 입력 조건은 다른 입력 조건보다 빈번히 발생하므로 운영중의 입력과 상대적인 빈도를 고려한 명세서가 시스템의 운영 프로필이다.

제 3 절 시험 지원 자동화 도구

1. 자동화 도구의 유형과 기능

시험을 지원하는 자동화 도구의 개발 상황을 살펴보면, 범주 분석기와, 리그레션 시험 지원기의 개발과 적용이 가장 활발하게 이루어지고 있다. 코볼에 대한 구조적 정보와 품질 정보를 제공하는 정적 분석기도 상용화 되어 있다. 시험 정보 관리 시스템과 사례 설계 지원 시스템은 아직 만족할 수준의 도구가 개발되어 있지 않고 적용도 거의 이루어지지 않고 있다. 각 도구들은 독립적으로 개발되어 도구간의 연계가 이루어지지 않고 있어 시험 활동을 체계적으로 지원하는 통합도구의 필요성이 대두되고 있다. 현재 이용가능한 자동화 도구의 유형과 그 용도를 살펴보기로 한다.

ㅇ 품질 측정 및 품질 오류 진단 도구

품질 메트릭은 정적인 방법과 동적인 방법, 그리고 자동적인 방법과 수동적인 방법에 의해 측정이 가능하다. 품질 측정의 대상은 모듈과 시스템이다. 현재의 품질 측정 도구는 코드 메트릭의 자동 측정 기능과 정적인 방법에 의해 오류를 진단하는 기능을 가지고 있는데 수작업으로는 거의 불가능한 작업을 자동화하여 유용성이 높다. 형식화된 PDL이나 요구명세 기술언어를 사용하는 경우에는 모듈에 대해서 결합도와 응집도의 측정도 가능하나 상용화된 도구는 현재 나와 있지 않다. 프로그램의 특성을 측정하는 메트릭은 시스템의 크기, 구조에 의한 복잡도, 모듈화 정도, 정밀도, 독립도 등을 추출한다.

코볼을 지원하는 도구는 COSTAR(SERI), PATHVU(미국 XA/system), COBOL/METRICS(Computer Data Systems), INSPECTOR(Language Technology)등이 있으며 C를 지원하는 도구는 PC/Metrics(Set Laboratories)등이 있고 기타 AMT(General Electric Company) 및 Ada를 지원하는 ATVS등의 도구들도 있다.

o 시험 사례 생성 지원 도구

동적 시험 설계와 수행은 노력 집약적이고 어려운 작업으로 자동화 도구의 필요성이 절실하나 아직 사례 설계 자체를 지원하는 도구는 나와있지 않다. 현재 구조적 테스팅을 지원하는 도구의 기능은 프로그램 제어 구조의 생성과 구조내의 특정 경로에 대한 입력 조건을 생성하는 수준으로 COSTAR(SERI), ACT(McCabe & Associates사) 등이 있으며 입력 조건에 의한 시험 데이타 생성기는 T(PEI), TD/GEN(Software Research사)등이 있다.

o 시험 범주 분석기

범주 분석기는 TCAT(Software Research사), ATVS등 다수가 나와 있으며 모듈, 함수, 분기, 문등에 대해 미시험 부분과 시험부분의 수행 횟수등을 출력한다. 시험 범주 분석기는 프로그램 실행시 생성되는 추적(Trace)에 의해서도 일부 정보를 추출하나, 대부분 원시 프로그램에 측정계기가 삽입되어 정보를 기록하도록 구성되어 있다.

o 리그레션 시험(Regression test)지원기

AUTOTESTER(Software Recording), REPLAY(Software Research사), DCATS(Systems Design and Development Group)등 다수가 나와 있다.

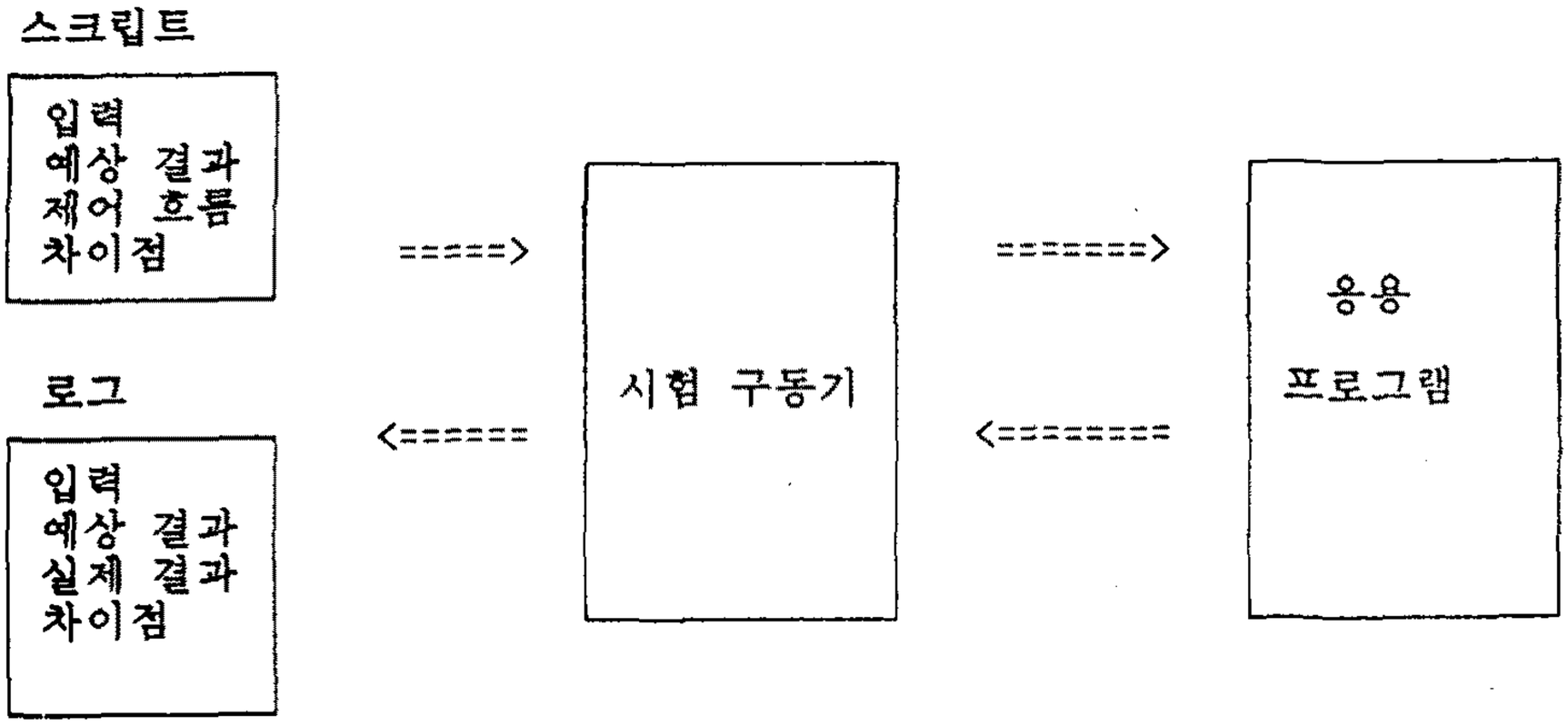
리그레션 시험 지원기는 입출력 기술언어를 이용하여 스크립트를 구성하고 시험 구동기로 응용 프로그램을 수행한 후 예상결과와 실제 결과를 비교하는 기능을 가지고 있다. [그림 2-3] 스크립트 작성은 단말기의 입력자료 저장 기능, 외부 파라메터 지정 기능, 스크립트 마크로정의 기능 등을 이용하여 사용자의 노력을 줄일 수 있다. 이러한 도구는 리그레션 시험에 많이 사용되지만 한번 스크립트를 작성하면 입력이나 예상 결과를 약간 변형하여 사람의 개입없이 응용 프로그램을 다시 시험할 수 있으므로 하나의 응용 프로그램을 많은 데이타로 시험하는 경우 그리고 그 예상 결과를 쉽게 기술할 수 있을 때 활용되어 시험 노력을 대폭 감소 시킬 수 있다.

ㅇ 설계 복구 도구

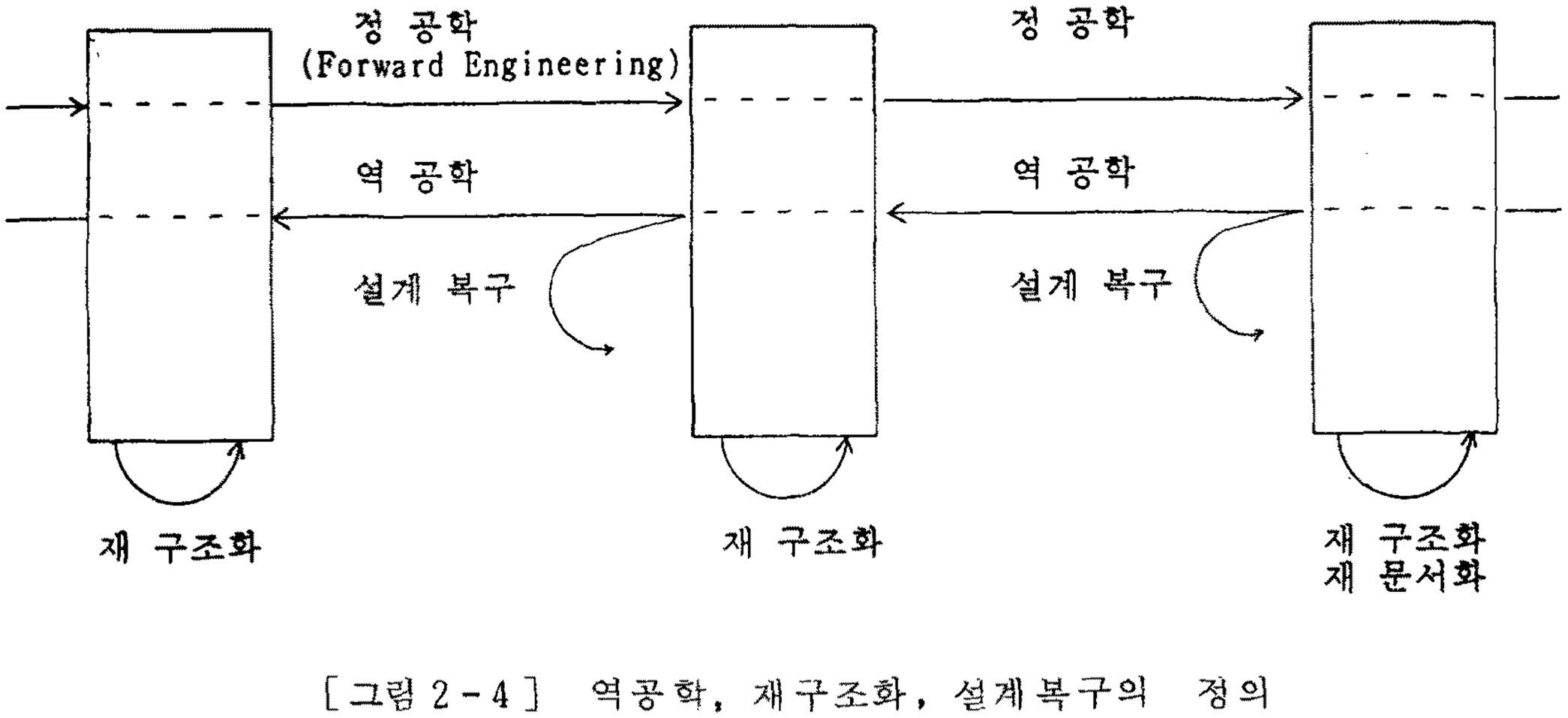
이 도구는 코드에서 프로그램 구조도, 입출력정보, 모듈간 호출관계, 데이타베이스 및 화일 접근상황등을 추출하는 도구로 COSTAR(SERI), SCAN/COBOL(Computer Data System), PINPOINT(Language Technology), Exellerator/Design Recovery(Index Technology)등이 여기에 속한다. 설계복구란 역공학의 한 형태로서 코드로부터 설계 단계의 정보를 추출하는 것이다. 역공학과 설계복구, 재공학등의 관계가 [그림 2-4]에 나타나 있다.

ㅇ 재구조화 도구

재구조화 도구는 코볼언어를 지원하는 도구가 있으며 많은 기관에서 사용되고 있다. 재구조화 도구는 구조화 되지 않은 프로그램을 동일한 기능을 수행하는 구조화 프로그램으로 변환하며, 한 프로그램내의 모듈간의 의존성을 분석하여 응집도에 의해 여러개의 프로그램으



[그림 2-3] 리그레션 시험 지원기의 구조



[그림 2-4] 역공학, 재구조화, 설계복구의 정의

로 분리할 수 있도록 한다. 여기에는 Recoder(Language Technology), SUPERSTRUCTURE(Computer Data System), RETROFIT(미국 XA/system) 등이 있다.

o 시험 및 품질 정보 관리 도구

시험 및 품질정보는 품질관리와 시험의 계획 초기단계에 어떤정보가 누구에게 어떤 목적으로 제공되어야 하는지, 그리고 정보의 제공 방법은 어떠한 것이 될 것인지 등, 정보의 수집과 배포에 대한 정의가 필요하다. 관리가 필요한 시험 정보는 시험 기능, 시험 사례, 시험 오류, 시험 로그등이다. 품질에 관한 여러 메트릭 정보와 시험 결과 정보를 관리할 수 있는 도구로서 MARS(Applied Information Development Inc.)등이 있고 시험정보 및 품질정보 관리 시스템으로는, TIGER(SERI)등이 개발되어 있다.

2. 동적 시험 도구

가. TCAT/COBOL(Test Coverage Analysis Tool for COBOL)

미국 Software Research, Inc.에서 개발하였으며, 소프트웨어 품질 분석가, 개발자 그리고 소프트웨어 메트릭 또는 독립 평가 그룹들이 사용할 수 있다. 소프트웨어 품질 분석가는 개발된 시스템을 시험하고, 개발자는 소프트웨어 프러덕트가 SQA(Software Quality Assurance)로 제출되기전에 모듈과 서브 시스템을 코드 범주 분석 기준을 만족하도록 단위 또는 분기 레벨에서 시험한다. 소프트웨어 메트릭 또는 독립 평가 그룹은 서브 시스템 또는 전체 시스템을 측정, 평가한다.

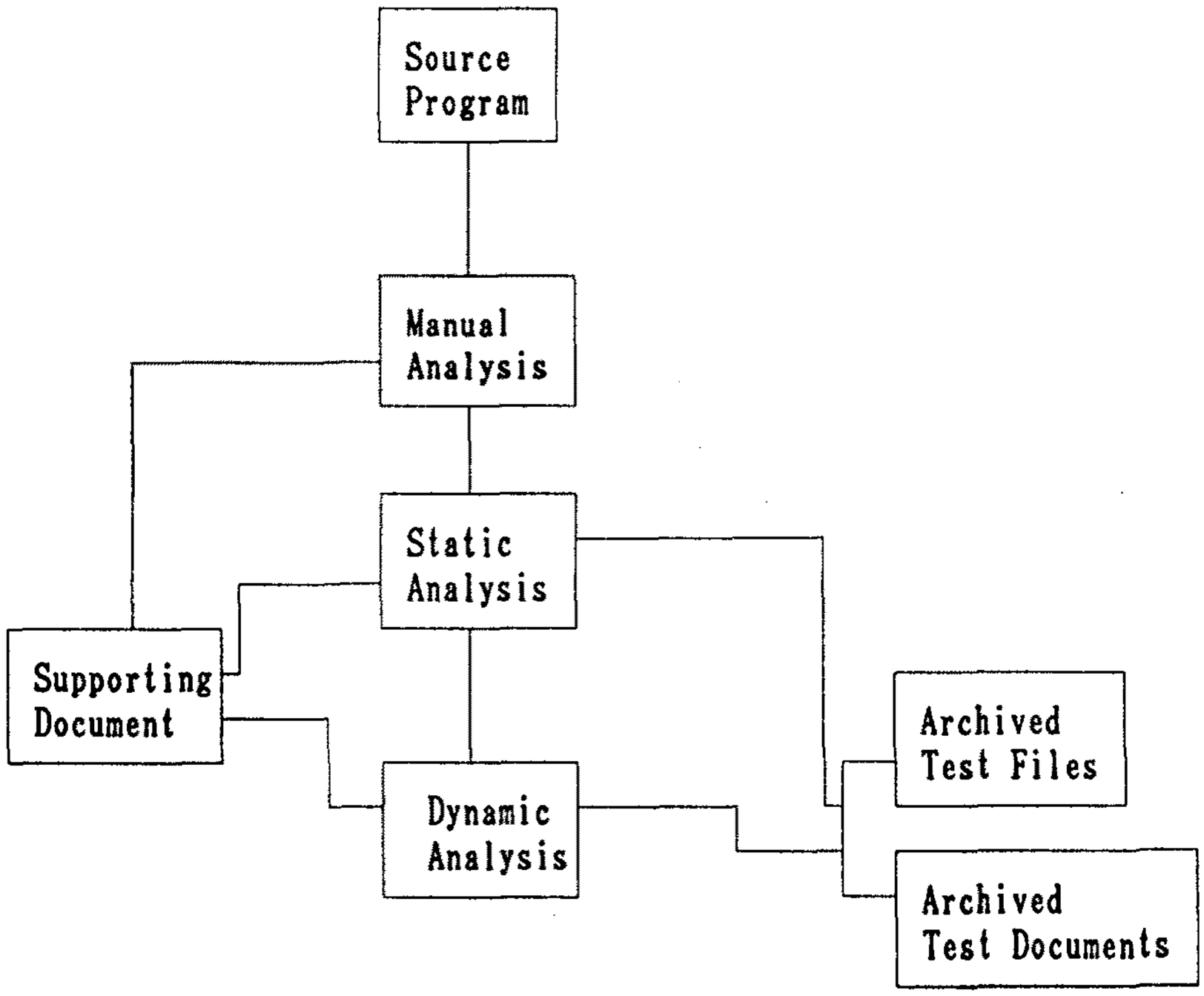
TCAT의 개발 목적은 모듈의 단위 시험과 전체 시스템의 일련의

완전한 시험 측정이다.

TCAT의 특징을 보면 다음과 같다.

- 에러 검증 기능의 향상
- 조기 에러 발견
- 효율적 시험 수행
- 최소의 시험 사례
- 진도의 향상
- 동적 분석
- 단일 모듈과 복수 모듈 시험

전체적인 시스템 형태를 보면 [그림 2-5]과 같다.



[그림 2-5] TCAT/COBOL 시스템 구조도

TCAT에서는 세그먼트 개념을 사용하는데, 세그먼트를 설명하기 전에 문을 설명하기로 한다. 프로그램의 가장 기본적인 요소는 문과 논리적 세그먼트 또는 분기이다. 문은 코드의 가장 작은 단위이다. 여기서는 단순 범주 값을 C0로 나타내는데, 시험에서 수행된 문의 수를 측정한다. C0의 범주는 다음과 같다.

C0 =실행된 문의 수/총 문의 수 * 100

세그먼트는 프로그램의 방향 그래프의 에지와 같고, 분기라고 칭한다. 세그먼트 범주의 레벨을 측정하는 값을 C1으로 표현한다. C1은 분기의 수행 횟수에 의하여 프로그램시험을 분석한다. 즉, C1은 단일 시험 결과 또는 하나 혹은 그 이상의 모듈에서 일련의 결합 시험의 누적 결과이다. C1은 1회 이상의 시험으로 수행된 분기의 백분율로 나타낸다. C1 보고서는 널(null)분기 같은 분기에 대해 결과를 산출하기 때문에 오류를 빨리 찾을 수 있고 C0의 범주보다 약 50% 정도 더 효과적이다.

TCAT에서는 다음과 같은 보고서를 산출한다.

- 금회 보고서
- 과거 요약 보고서
- 통계적 금회 범주와 누적 범주 보고서
- 수행된 세그먼트의 리스트
- 수행된 세그먼트의 횟수의 바(bar) 챠트
- 보고서 형태
  - 누적 보고서
  - 수행 보고서와 미수행 보고서
  - 부분 수행 보고서와 부분 미수행 보고서

• 성능 분석 보고서 :

*로그 히스토그램과 선형 히스토그램 보고서

TCAT-PATH

특 징

- 주어진 프로그램의 구조 그래프를 자동 생성
- 시험 사례의 완전한 셋(SET)을 자동 생성
- 제어 흐름 그래프를 디스플레이
- 프로그램의 Cyclomatic Complexity를 계산
- 계기 삽입 프로그램의 추적 화일을 자동 분석

연산 MODE

분 석

조 회

생 성

수 행

계 산

TCAT-PATH를 이용한 작업단계는 다음과 같다.

(1) 계기 삽입 (세그먼트를 표시)

원시 코드의 분기를 찾기 위해 원시 코드를 파싱하고, 계기 삽입기는 원시 코드의 계기 삽입된 버전에서 function call을 삽입한다 계기 삽입된 코드를 실행 시키면 삽입된 function은 추적 화일 (Trace File)에 프로그램 실행 위치를 저장한다.

(2) 컴파일, 링킹 (기록 표시와 카운트 표시와 함께) 및 수행

(3) 패스 생성 (완전한 패스 셋 (set) 을 생성)

(4) 범주 분석 (수행된 패스 보고서 생성)

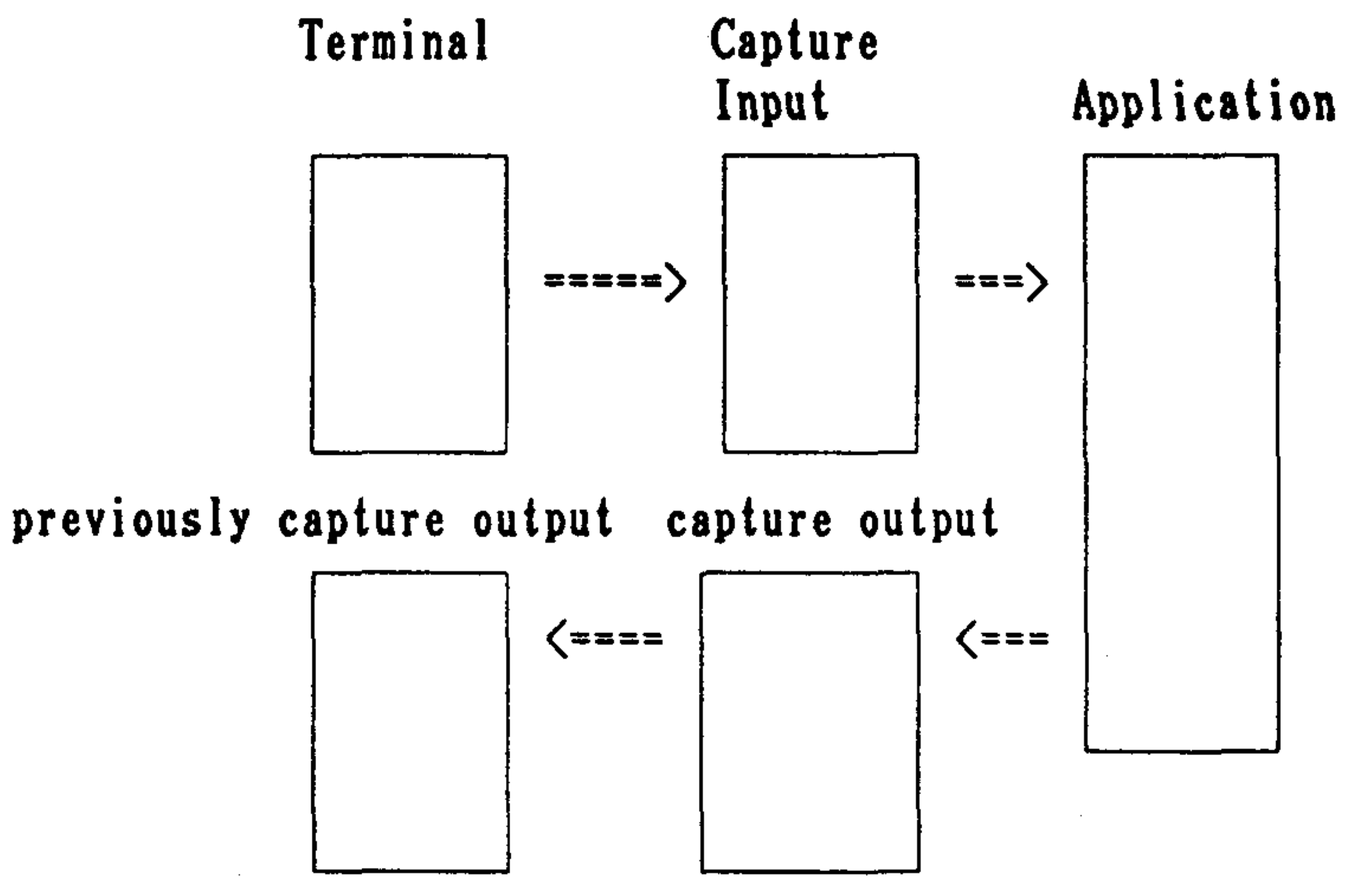
나. DCATS

DCATS 은 미국 System Design & Development Corp. 에서 개발하였으며, 온-라인 시스템을 개발, 통합, 설치 그리고 유지하는데 있어서, 노력과 시간을 절감하기 위해서 개발되었다.

DCATS 은 다음과 같은 특징을 가지고 있다.

- 응용 프로그램이 수행되는 동안 응용 프로그램의 입력 출력을 기록할 수 있다.
- 기록된 입력 메시지는 응용 프로그램에 다시 적용될 수 있고,
- 이전에 생성된 결과와 현재 생성된 결과의 비교가 가능하다.

DCATS 의 구조는 다음과 같다.



DCATS 은 capture 와 repeat 모드에서 사용되며, 다음과 같은 기능을 제공한다.

1. 프로그램이 수행되기 전에 입력과 예상 결과 메시지의 내용과

흐름(flow)을 기술하는 스크립트(script)를 생성한다.

2. 실제 하드웨어와 소프트웨어를 사용하지 않고 예외 조건 메시지를 생성한다.

3. 수행시에 입력과 예상 결과의 내용을 합성(composition)한다.

4. 메시지 내용을 기반으로 하여 각각의 응용 프로그램 결과를 지능적으로 인지한다.

5. 결과 메시지의 구조, 내용, 응답 시간을 검사한다.

6. 값의 영역으로 정의된 예상값과의 동일성, 집합내의 값들 중 하나와의 동일성 등과 같은 기준으로 출력값을 검사한다.

8. 동시에 여러개의 응용 프로그램을 수행한다.

그 밖에 여러가지 특징들이 있다.

DCATS은 스크립트 언어, workbench, Application Analyzer, Reporting System을 포함하며 사용자 관리 프로세스들과 결합하여, 온-라인 응용 프로그램을 개발, 통합, 설치, 그리고 유지하는 시간과 노력을 절감할 수 있다.

먼저 스크립트 언어에 대해서 알아 보기로 하자.

스크립트 언어는 응용 프로그램의 입력과 예상결과의 내용과 제어 흐름을 기술하는데 사용한다.

스크립트 언어를 이용하여 다음 사항을 기술한다.

- 입력과 출력 메시지의 형태
- 입력과 예상 결과 메시지의 내용
- 입력 메시지의 비율과 순서
- 출력 메시지의 인지/검증에 대한 규칙들

- 외부 매개 변수의 값
- 시험 보고서의 형태

입력과 출력 메시지를 사용하여 사용자와 다른 응용 프로그램은 온-라인 프로그램과 통신한다.

시험 결과를 문서화하는, 응용 프로그램과 독립된 보고서 시스템이 있다. 보고서는 요약 보고서와 상세 보고서로 구분할 수 있다. 그 형태는 다음과 같다.

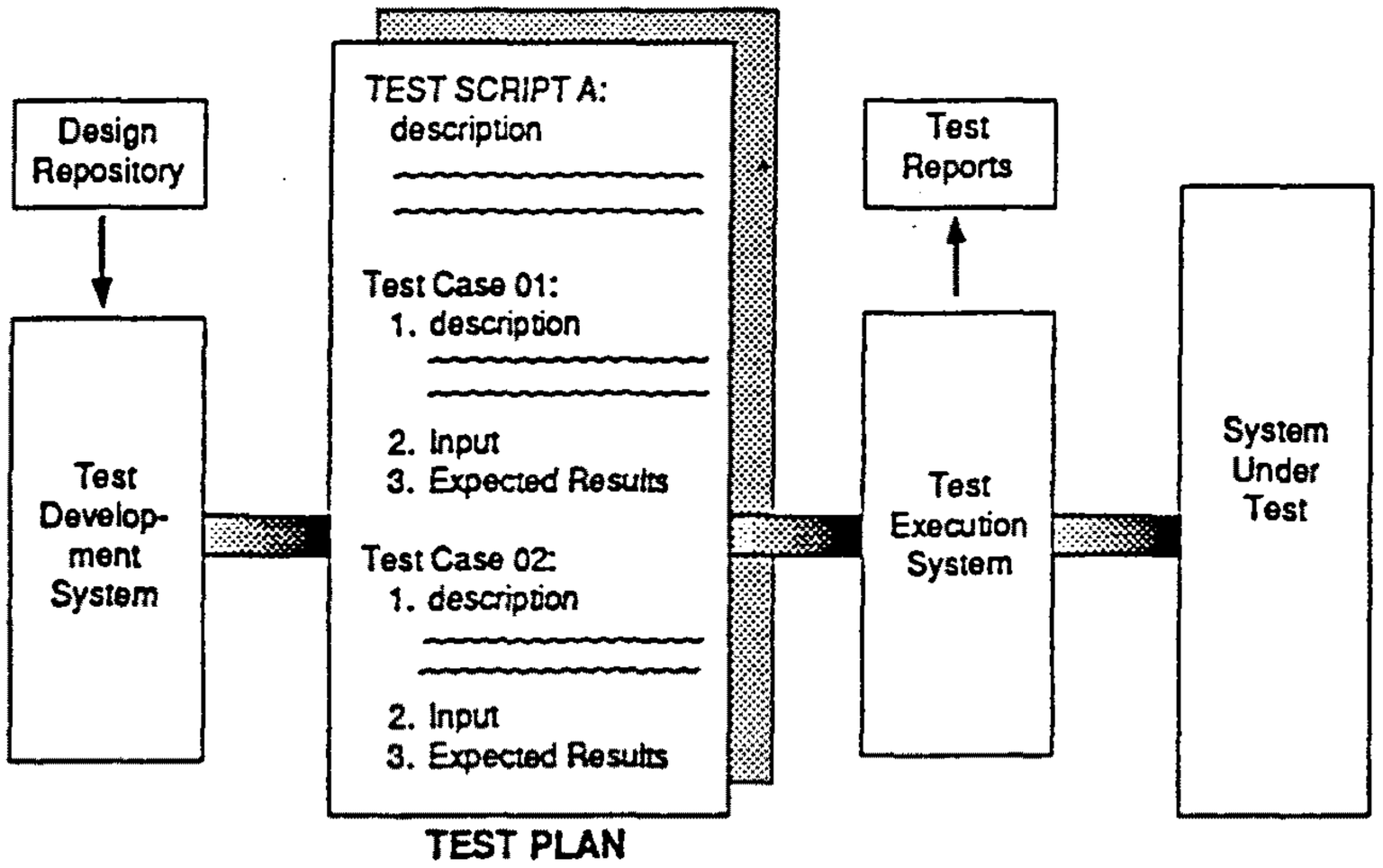
- 요약보고서

Application NODE-12
Test Script TEST-1
Parameter Table . . . . . ACCT-A
Begin Time 10:11:22
End Time 10:11:59
Begin Date 05/11/88
End Date 05/11/88
Program input messages sent 75
Program output messages recognized 70
Program output messages in error 2
Program output message didn't arrive in time 5
Unexpected messages received 2

- 상세보고서

```
10:11:22.2   COM                  CASE-A Invalid Account
10:11:23.1   GEN   A-1 A          LU-01  ACCOUNT= 4717-2345-346
10:11:25.1   VAL   A-2 A-1        LU-01  ACCOUNT= 4717-2345-346
10:11:25.2   COM                  CASE-B Invalid Amount
10:11:26.1   GEN   B-1 B          LU-02  ACCOUNT= 4717-2345-346
10:11:29.1   SUT   B-2 B-1        LU-02  RSP      4717-2575-346
             ANL                  LU-02  RSP      4717-2345-346
             err                  LU-02  .........++++++*++++++
10:11:30.1   COM                  END OF TEST-1
```

- 시험 개발 시스템과 시험 수행 시스템의 관계는 다음과 같다.



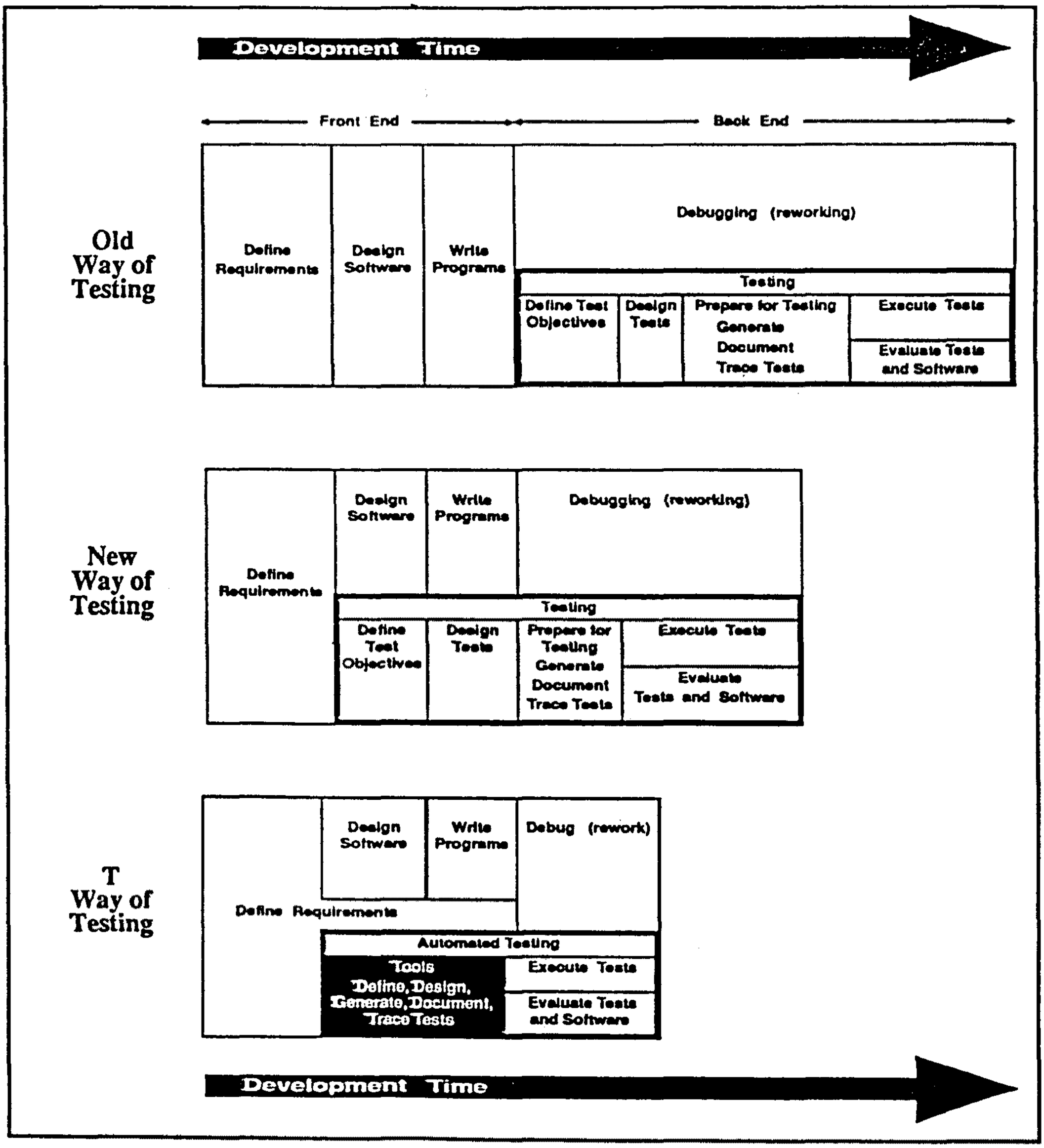
3. 사례 설계 지원 도구

가. T

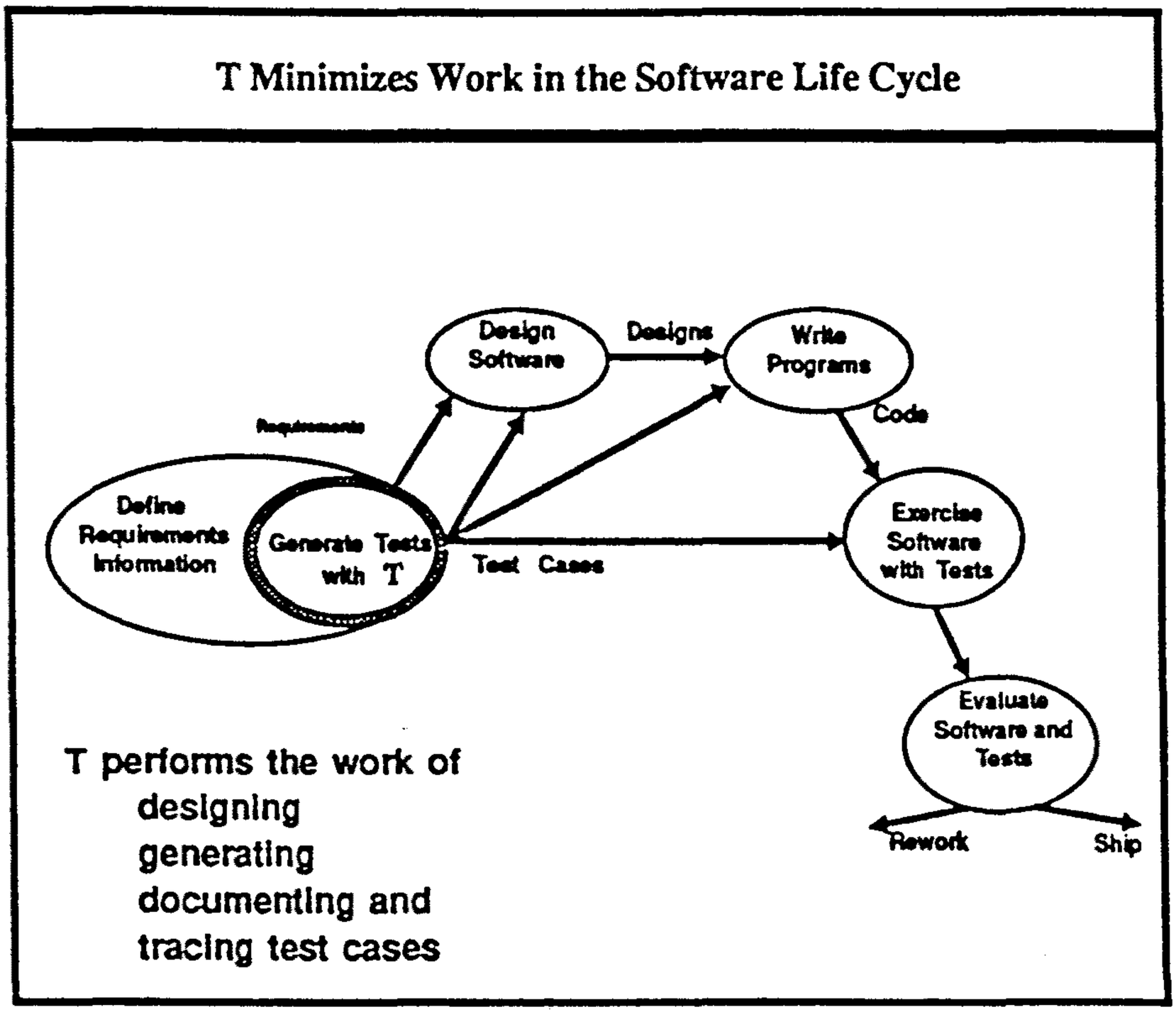
특 징

- T는 요구 분석으로부터 시험 사례를 생성
- 프로그램과 시스템을 시험하는데 사용될 시험 사례를 생성 하는 도구
- 프로그래머가 효율적으로 시험할 수 있는 최소의 시험 사례를 생성(minimal test set).
- 지식 베이스(Knowledge Base)를 도입.
  - 최소의 시험 사례 설계를 위한 규칙의 집합과 범하기 쉬운 오류의 규칙 베이스
- T는 개발자가 오류를 정정할 수 있도록 요구 정보와 문제와 불일치를 보고(reporting)한다.

* T에서의 소프트웨어 라이프 싸이클은 다음과 같다.



* T는 소프트웨어 라이프 싸이클의 작업을 최소화 한다.



T의 구성 요소별 기능은 다음과 같다.

* 시험 구동기

시험 구동기는 소프트웨어를 시험하는 동안 각 시험 사례에 대해 적어도 기본적인 기능 3가지를 수행하는 프로그램이다. 시험 구동기는 일정한 형태로 시험 사례를 읽어 들여 그 입력 형태(format)를 소프트웨어 시험에 맞게 변환한다. 시험 구동기의 시험 사례 입력은 간단하게 flat 화일 형태로 나타낼 수 있다. 시험 구동기로부터의 출력은 시험에서 소프트웨어에 명시된 형태로 정리된다.

* 시험 사례 생성기

시험 사례를 생성하는 방법은 여러가지가 있다. 사례 방법에 따라

도구를 분류하면 다음과 같다.

(1) Gode_based test case generator

- 모든 문, 분기, 패스가 적어도 한번 수행될 수 있는 시험 사례를 생성
- 시험 사례는 코드로부터 직접 생성
- missing 코드의 오류는 이 도구에서 빠뜨리기 쉽다.

(2) Specification_based test case generator

소프트웨어 요구 명세서에 기술된 사실들로부터 시험 사례를 생성한다. 생성기는 요구 명세로부터 직접 시험 사례를 유도(derivation)하기 때문에 missing과 오류를 찾기 쉽다.

T의 장점은 다음과 같다.

- 요구 사항 시험을 검증할 수 있다.
- specification_based test case를 설계하고 생성할 수 있다.
- 시험 사례를 다큐멘트할 수 있다.
- 시험 사례의 시작점과 종료점을 추적할 수 있다.
- 시험 사례의 수행 여부를 추적할 수 있다.
- 요구 범주를 측정할 수 있다.
- T의 시험 작업을 문서화하고 보고서를 작성할 수 있다.

단점은 다음과 같다.

- Product_based test case를 생성할 수 없다.
- 구조 범주를 분석할 수 없다.
- 예상 결과와 실제 결과를 비교할 수 없다.

T에서 제공하는 보고서는 다음과 같다.

* 메트릭의 보고서

  - 상황 보고서 (status report)

  - 수행 이력 보고서

  - 총 미수행 보고서

  - 시험 품질 보고서

* 상황 보고서 (status report)

  - work product와 work process의 메트릭을 나타낸다.

    • 생성된 시험 사례수

    • 수행된 시험 사례수

    • 수행되지 않은 시험 사례수

    • quantity 정보를 나타낸다.

* 수행 이력 보고서

  - 상황 보고서를 이용하여 quantity 정보를 얻고, 시험 사례가 수행된 횟수를 나타낸다. quality 정보를 제공하지 않는다.

* 총 미수행 보고서

시험 사례의 수행 시작부터 보고서가 출력되는 시점까지 시험 사례에 의해 발견되는 실패 횟수를 나타낸다. 그 표현은 그래프로 표현한다. 그래프의 경사가 급하면 코드에 많은 실패가 존재하고, 실패들을 빨리 찾을 수 있고 시험 종반에는 실패가 감소한다.

* 시험 품질 보고서

시험 사례가 기준 범주를 커버에 얼마나 만족하느냐에 달려 있다

이 보고서는 가중합 방정식을 기반으로 함.

나. SILOP(Simple Loop Patterns)

Simple Loop Pattern을 기반으로 하여 반복 루프가 포함된 프로그램을 시험하는 선택적 방법이다. Path Analysis는 프로그램 시험계획 (Planning)과 시험 설계 전략(strategy)에 유용하다.

* Simple Loop Pattern 이론 배경

Simple Loop Pattern은 "SILOP" 도구에서 사용하기 위해 정의하였다. Pattern Hyperplan은 시험 전략에서 요구된 시험 횟수를 예측하는데 고려한다. 여기서 고려된 개념은 패스 계산, 패스의 분류, Pattern Hyperplan이 있는데 이들 몇 가지만 설명하기로 한다.

- 패스 계산

프로그램의 변수는 입력 변수 또는 프로그램 변수로 분류할 수 있다. 입력 변수를 조건 제시법으로 표현하면 $X = \{x_i \mid i = 1, \cdots, k\}$로 나타내고 프로그램을 통해서 재할당 되지 않는다. 프로그램의 변수 형태는 $Z = \{z_j \mid j = 1, \cdots, m\}$이고 프로그램을 통해 할당된다. k-vector x의 집합은 프로그램 입력 X로 나타내고 m-vector z는 프로그램 변수 공간 Z_m으로 표현할 수 있다. 출력 변수는 $Y = \{y_i \mid i = 1, \cdots, t\}$, $t \leq m$로 나타낼 수 있다.

프로그램에 의해 계산된 프로그램 변수 함수 f는 $f : V \rightarrow V$, 단 $V = X_k \times Z_m$이다. 프로그램의 제어 흐름은 $G = (Q, A, s, e)$로 정의할 수 있고 Q는 노드의 유한 집합이고 A는 에지의 집합을 의미하며 s, e는 엔트리와 exit노드를 나타낸다. 임의의 $(a,b) \in A$ $(a, b \in Q)$는 기호 $g_{a,b}$의 레벨이 붙는다. 그래서 프로그램 변수 함수

는 $g_{a,b} : V \rightarrow V$로 표현할 수 있다. 노드 a에서 b까지의 패스는 일련의 노드 $q_0q_1, \cdots, q_j$, $a = q_0$, $b = q_j$이고 $i = 0, \cdots, j-1$, 그리고 a = s이고 b = e 이면 프로그램 패스이다.

패스 계산은 함수 $f : V \rightarrow V$ 이다. 여기서 $f = g_{0,1} \circ g_{1,2} \circ \cdots, \circ g_{j-1,j}$ 이고 $g_{i,i+1} : V \rightarrow V$는 함수이다. 함수 g는 패스 $q_0q_1 \cdots q_j$를 따라 에지 레벨로 표현된 함수이다.

- 패스의 분류

여기서는 simple loop를 다루는데 G의 패스 $q_0q_1 \cdots q_j$는 단순 패스이고 패스가 $q_0q_1 \ldots q_jq_0$를 단순 루프라 하고 L로 표기한다. 패스의 집합은 $\{P_n \mid n = 0,1, \ldots\}$이고 s를 출발 L을 경유하여 e로 끝나는 단순 루프 패스 형태를 나타낸다.

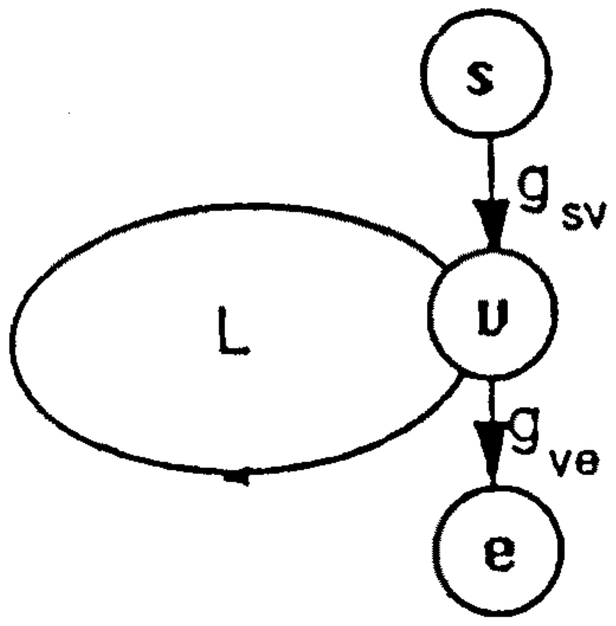
단순 루프 패스의 개념을 프로그램 변수로 변형 (transformation) 한다.

Pn 대신 fn을 정의하면

$$fn = \begin{cases} f_0, & n=0 \\ f_{n-1} \circ H, & n>0 \end{cases}$$

$\{ f_n \mid n = 0,1, \ldots \}$는 기저 함수 f_0와 pattern 함수 H로 정의한다.

g_{sv}, g_{ve}는 [그림 2-6]에서 패스 sv와 패스 ve의 계산이다.



[그림 2-6] 제어그래프의 simple loop.

h가 단순 루프 L의 계산을 나타내면 $h_n = h^\circ h \ldots^\circ$ h는 L이 n번 반복될 때 단순 루프 L을 나타낸다. n = 0일 때 h는 항등함수를 나타낸다.

$$f_{n-1} = g_{sv}{}^\circ\ h_{n-1}{}^\circ\ g_{sv}$$

$$f_n = g_{sv}{}^\circ\ h_n{}^\circ\ g_{ve}$$

그리고 $f_n = f_{n-1}{}^\circ\ H$, 단 $n > 0$

$$H = (g_{ve})_{-1}{}^\circ\ h^\circ(g_{ve})$$

$\{ f_n \mid n = 0, \ldots \}$은 일련의 단순 루프 패스를 나타낼때 Patten 함수 H를 단순 루프 Pattern이라고 한다.

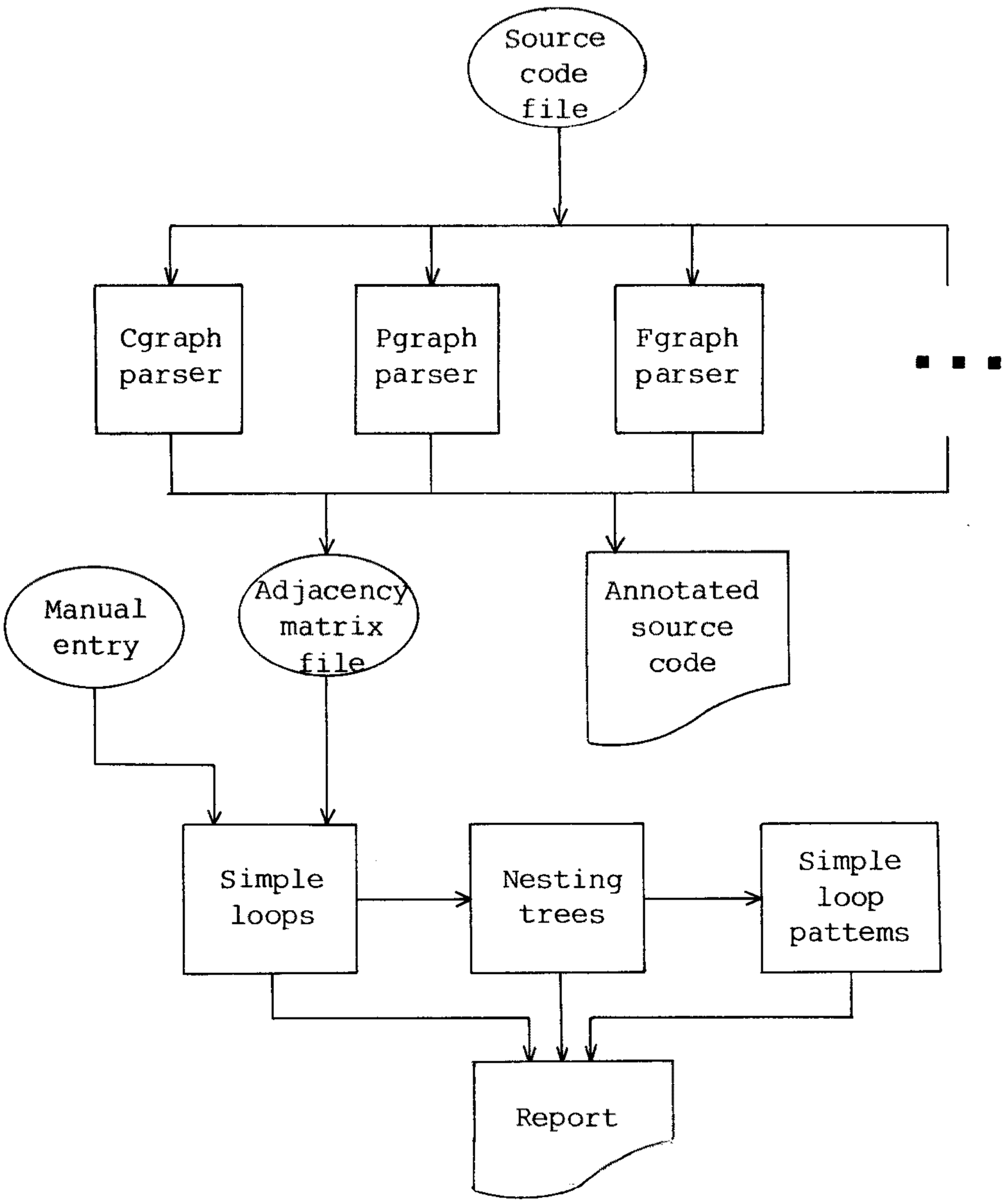
단순 루프 Pattern의 시험 전략은 다음과 같다.

1. 프로그램의 순서 제어 그래프를 얻기 위해 원시코드를 분석한다.
2. 순서 제어 그래프에서 elementary path와 단순 루프를 찾는다.
3. 단순 루프 Pattern과 단순 루프 순서를 얻는다.
4. 단순 루프 패턴을 시험할 패스 수를 결정한다.
5. 선택된 패스에 대해 시험 자료를 생성한다.

SILOP 도구는 1, 3, 4, 5만 가능하다.

단순 루프 Pattern은 시험되는 프로그램의 최상의 추상화 레벨을 표현한다. SILOP의 전체 시스템 구조는 다음과 같다.[그림 2-7]

SILOP의 기본적인 요소는 파서의 집합과 패스 분석의 집합으로 구분할 수 있다.



[그림 2-7] SILOP의 블럭구조

SILOP의 기본적인 요소는 파서의 집합과 패스 분석의 집합으로 구분할 수 있다.

* 파서들과 입력

파서의 집합은 프로시듀어 언어가 포함되는데, C(Cgraph)와 PASCAL

(Pgraph)이 포함된다. 모든 파서는 YACC를 이용해서 생성할 수 있다. YACC를 이용해서 생성된 파서가 최적화(optimization)와 효율적이 아니더라도 SILOP에 큰 영향을 주지 않는다. 왜냐하면 파서의 출력이 근접 메트릭스와 주석이 첨가된(annotate) 원시코드이기 때문이다. 문법은 YACC가 받아들일 수 있는 형태로 표현된다.

SILOP는 LEX를 사용한다. 파싱이 끝나면 입력프로그램의 제어 흐름 그래프는 근접메트릭스로 표현된다.

제어 흐름 그래프는 전체 프로그램 텍스트에 대한 주석이 첨가된 원시 코드를 생성하는데 사용한다. 텍스트의 라인은 노드번호로 테이블이 붙고 적당히 후속노드에 대해서도 레이블이 붙는다.

SILOP의 입력형태는 프로그램과 메뉴얼 엔트리(manual Entry)이다. 프로그램은 파서의 입력이 되고, 메뉴얼 엔트리는 입력의 형태가 근접 메트릭스로 되게 해준다. 입력은 의미 수행코드(pseudo code) 나 플로우챠트이다.

* 패스 분석 루틴(Path Analysis Routines)

SILOP에서 패스 분석 루틴은 Simple Loop, Nesting tree, Simple loop pattern이 있다.

Simple 루프 루틴은 패스를 찾을 근접 메트릭스의 형태에 제어 흐름 그래프를 분석한다.

Nesting tree는 후에 수축되는 트리를 동적으로 구성한다.

이 수축은 하나 이상의 패턴 함수가 동등하는 사실 기반위에서 이루어진다. 동등한 패턴 함수중의 하나는 트리에서 제거된다.

Nesting tree의 최종버전에서 루트 노드를 제외한 각 노드는 패

스 또는 Simple loop pattern과 일치한다.

Simple loop pattern은 트리운행 알고리즘을 구현한다. 트리운행 알고리즘을 일련의 simple loop path를 생성한다.

4. 정적 분석 도구

정적분석이란 프로그램을 수행하지 않고 프로그램의 구조와 특성을 분석하여, 설계 정보를 복구하고, 구조상의 또는 문맥상의 오류를 검사하거나, 프로그램의 특성을 측정한다.

정적 분석 도구들은 위의 3가지 기능의 일부 또는 전부를 지원하며 몇몇 도구는 정적 분석 기능외에 시험 범주 측정과 경로 생성등의 복합적 기능을 가지고 있다. [표 2-2]는 외국 정적 분석 도구의 주요기능 및 대상 언어이며, 정적 분석 기능만이 분석되었다.

각 도구의 품질 측정 기능을 살펴보면 Cyclomatic Complexity,LOC, Vocabulary등의 메트릭을 측정하는 것이 많다. 일부 도구는 설계 정보의 복구에 기능이 집중되어 있다. [표 2-3]에 본 과제에서 개발된 COSTAR와 외국 도구간의, 오류 진단 기능, 자동 문서화 기능(설계 복구), 품질 측정 기능 등이 비교되어 있다.

[표 2-2] 외국 정적 분석 도구의 주요기능 및 대상언어

도구(공급자)	주 요 기 능	언 어
BCT (McCabe and Associates)	Module Structure Chart Metric Measurement	C,COBOL
SCOREBOARD (TRAVTECH)	Module Size Complexity Clarity	COBOL
SCAN/COBOL (Computer Data Systems Inc.)	Hierachical Charts Variable Cross Reference Block Flow Diagram	COBOL
COBOL/METRICS (Computer Data Systems Inc.)	Metric Measurement	COBOL
COBOL SXRF (Fedaral Software Testing Center)	Super Cross Reference	COBOL
TCAT-PATH (Software Research Inc)	Complexity Measures	C, Ada, Fortran
SCOREBOARD (TravTech Inc.)	Quality Analyzer	COBOL
TESTBED (Program Analysers Ltd)	Cross Reference Complexity Measures	Fortran Pascal PL/I C, Ada

도구 (공급자)	주 요 기 능	언 어
KANGA (Institute for Information Industry, 대만)	Data Flow Analysis Cross Reference Path Analysis Interface Analysis Variable Usage Analysis	C
LOGISCOPE (American Management System)	Metrics(Complexity) Kiviat Diagram Control Graph	Focus, C,COBOL Ada,Pascal
PATHVU (KPMG Peat Marwick)	Metric Analysis(Complexity) Summary Diagnostic Report	COBOL
SOFTDOC (Software Engineering Services)	HIPO Diagrams Structure Chart Test Path Cross Reference Quality metrics	COBOL

[표 2 - 3]　COSTAR 와　외국　정적분석　도구와의　기능비교

	COST-AR	SCORE BOARD	SCAN/ COBOL	TEST-BED	Logi-scope	McCabe BCT	PATHVU
Diagnostics 제공							
1. 사용되지 않은 변수	○			○			
2. 참조되지 않은 변수	○			○			
3. 정의되지 않은 변수	○			○			
4. Dead Code	○		○				
5. 변수 Type의 불일치(Move)	○						
6. GOTO에 의한 Endless Loop가능부분	○						○
7. PERFORM에 의한 Endless Loop가능부분	○						○
8. PERFORM내에서의 Goto Runaway	○	○	○	○			○
자동문서화 기능							
1. Program Identification	○						
2. Variable참조	○			○(*)			
3. File Variable참조	○			○(*)			
4. File정보	○						
5. Accept정보	○						
6. Call정보	○						
7. 프로그램 Hierachical Chart	○		○				○

	COST-AR	SCORE BOARD	SCAN/ COBOL	TEST-BED	Logi scope	McCabe BCT	PATHVU
8. Graphic Structure Chart	○			○		○	
- Sequence	○						
- Selection	○						
9. File Reference Matrix	○						
10. Program Call Matrix	○						
11. Paragraph Entry condition in source list	×		○				
12. Control Flow Graph	○	○	×	○	○	×	
품질평가기능							
1.프로그램 단위별 Metrics							
- LOC	○						○
- ELOC	○				○		○
- Comments Line	○	○					
- Cyclomatic Complexity	○	○		○	○	○	
- Halstead S/W Science	○				○		
- File, Paragraph Count	○						○
- Verb Statistics	○		○				
- Complex Area	○						
• GOTO 사용부분	○						○
• ALTER 사용부분	○		○				○
• 혼합연산 (ADD, SUBTRACT 등)	○						

	COST-AR	SCORE BOARD	SCAN/ COBOL	TEST-BED	Logi-scope	McCabe BCT	PATAVU
• Negative Boolean 사용부분	○	○					
• Complex Boolean 시용부분	○	○					
• Multiple Exit	○						
• Large Nesting Level	○	○					○
• Module 별 Size	×	○					
2. 시스템내 프로그램별 Metrics	○						○
3. Keyword Detection	×		○				○
프로그램 수정시 관계정보 제공							
1. 변수의 수정시							
• 변수의 정의 참조 상황	○						
2. 화일의 속성 및 필드 변경시							
• 화일의 프로그램별 Usage 정보	○						
3. 프로그램 기능 변경시							
• 프로그램 호출 Matrix 정보 사용	○						
Source change comparation	×		○				

(*) 변수 참조시 정의 및 참조의 구분이 제공되지 않음.

제 4 절 소프트웨어 품질 보증

70년대 중반부터 소프트웨어 품질 개념이 출현하여 이에 대한 연구가 진행되었다. 70년대 후반에 걸쳐 품질에 대한 포괄적 정의가 완성되어 80년대부터 미국과 일본등의 기업에서 품질관리 활동을 시작하였으나, 품질관리의 관심과 중요성에 비해 그 활동은 충분히 정착되지 못하고 있는 형편이다.

이 장에서는 품질보증 활동의 개념 소개에 이어 품질 실현을 위해 적용할 수 있는 여러 기법을 점검한다.

1. 품질 보증 활동의 개요

소프트웨어 품질 보증과 확인 검증은 밀접한 관계를 가진다. 개발 조직에서 품질 보증 기능과 확인 검증 기능을 구성하는 방법은 여러 유형이 있을 수 있다. 일반적으로 품질 보증 기능의 역할을 정의 하자면 QA는 소프트웨어와 개발 과정에 대한 표준 준수와 품질 전반에 대한 검사를 시행하는 관리적 기능이다. QA의 기능의 주요 업무를 요약하면 다음과 같다.

- 개발 및 유지보수 업무의 표준 개발 공정 정의
- 개발 단계별 산출물 정의, 단계별 행위 수행자 및 관련자 정의
- 품질보증, 확인 검증, 감사, 시험 및 형상관리 기능의 역할정의
- 검토, 감사 및 개발 표준정의
- 각 검토 위원회의 구성
- 품질정보의 기록, 관리 및 배포

• 각 과정에 필요한 기법 및 도구의 정의

확인 검증은 사용자의 요구 사항 누락 여부와 각 산출물의 결함 여부를 심층적으로 분석하는 기술적 기능이라고 할 수 있다. 확인 검증은 소프트웨어가 요구된 기능을 수행하는지, 예상치 않은 행동을 하는지 품질과 신뢰성을 측정하기 위해 시스템을 분석하고 시험한다. [표 2-4]에 확인 검증(V & V)의 기본 업무가 열거 되어 있다.

[표 2-4] 확인 검증의 기본 작업

개 발 단 계	작 업
개 념 정 의 요구사항 정의	• 개념 문서 평가 • 요구사항과 개념간의 추적성 분석 • 요구사항 검증 • 하드웨어, 소프트웨어, 운영자간의 인터페이스 분석 • V & V 시스템 시험 계획 수립
설 계	• 설계와 요구사항 간의 추적성 분석 • 설계 품질 평가 • 인터페이스간 자료 항목 분석 • V & V 단위 시험 계획 시작 • V & V 결합 시험 계획 시작
구 현	• 설계와 코드간의 추적성 분석 • 코드 품질 평가 • 인터페이스간 자료 항목 분석 • 단위 시험 수행
시 험	• 결합 시험 수행 • 시스템 시험 수행 • 인증 시험 수행
설치 및 인수	• 설치 및 형상 감사 • V & V 최종 보고서 생성

이 작업외에도 시스템의 특성에 따라 추가되는 기능은 알고리즘 분석, 데이타 베이스 분석, 데이타 흐름 분석, 제어 흐름 분석이 있고, 시스템 시험이 곤란한 상황이거나 시험전에 시스템의 타당성을 분석할 필요가 있을 경우에는 시뮬레이션을 수행하여야 한다. 사용자의 입력이 대화식인 경우, 사용자 문서의 검토도 V & V의 기능에 포함된다. 기타 V & V기능이 QA기능의 일부인 경우와 사용자 그룹내에 있는 경우 등에 따라 V & V기능의 범위가 조정될 수 있다. V & V의 작업을 살펴보면 소프트웨어 구성요소간의 추적성과 구성요소의 시험용이성이 중요한 문제점임을 알 수 있다. 이 점은 개발의 표준을 정의할 때 반영되도록 고려해야 한다.

2. 품질 관리 활동의 고려사항

품질 관리 활동을 하면서 많은 조직에서 해결해야 하는 문제중의 하나가 기개발된 소프트웨어를 유지보수하는 것이다. 기개발된 소프트웨어는 대부분 관련 문서 명세서가 불충분한 경우가 많으므로 구현된 소프트웨어를 이해하기 위해서는 추상화 작업이 필요하다. 필요에 따라 소프트웨어의 일부분을 재설계, 재개발 또는 새로운 환경(하드웨어, CASE등)에 적용해야 할 경우가 있다. 이러한 작업을 위하여 역공학 및 재구조화 기법을 사용 할 수 있다. 역공학은 대상 시스템을 분석하여 시스템의 구성 요소들간의 관계를 추출하여 시스템을 다른 형태나 상위 수준의 추상화 형태로 표현하는 것으로 처리기능에 대한 추상화와 자료에 대한 추상화의 2분야로 나누어진다. 설계복구란 역공학의 한 형태로서 코드로부터 설계 단계의 정보를 추출하는

것이다. 실제 품질 관리 활동 계획할 때 성공적인 활동을 하기 위해서는 다음과 같은 사항들이 고려되어야 한다.

• 실현가능한 QA 프로그램의 확립 : QA 활동은 많은 분야에 대한 의욕적인 계획을 수립하는 것보다 중요한 관심사에 한정해야 한다. QA 활동을 시작하기 전에 조직의 품질 실현에 장애가 되는 여러 요인 중 공통적인 것을 살펴보면 비용, 시간, 개발 인원의 제한과, 개발환경과 기존의 소프트웨어가 열악하다는 것 그리고 개발 과정보다는 최종 산출물의 가시적 결과에 집중된 사용자의 관심, 잘 정의되지 않은 사용자 요구사항 및 개발 요원의 변동 등이다. QA 활동을 시작함으로써 이러한 모든 문제를 모두 해결할 수 있는 것은 아니다. 가장 파급효과가 큰것대로 우선순위를 정하여 문제를 해결해 나가야 한다.

• 경영층의 관심 : 경영층의 전적인 지원을 받을 수 없다면 QA는 실패 할 가능성이 높다. 경영층은 QA 기능의 결과를 지속적으로 검토하는등 QA 활동에 관심을 가져야 한다.

• QA 조직의 인적 구성 : QA 조직에 자질을 가진 인력과 비용이 충분치 않을 때는 QA 를 시도하는 것보다 현재의 상태를 계속 유지하는 것이 좋다. QA 그룹은 시스템 분석가 및 개발자의 10 %에 해당되는 원인이 적절하며 QA 부서의 책임자는 행정적인 관리자가 아닌 기술적인 프로젝트 관리 경험과, 최근의 개발 방법론에 익숙하고 경영층과의 의사소통 능력, 각종 표준을 잘 기술할 수 있는 능력, 여러 프로젝트 중진들과의 접촉에서 리더쉽을 발휘할 수 있는 능력이 요구된다.

기타 고려할 사항들은 다음과 같다.

• QA의 관리 대상은 먼저 코드부터 시작하여 전단계로 확장하는 것이 좋다.

• QA의 표준은 새로운 소프트웨어 개발방법론을 수용할 수 있도록 확장성을 갖추어야 한다.

• QA조직은 모든 신규 프로젝트와, 기존 소프트웨어의 주요 변경 과정에 참여하게 되므로 주어진 인력을 효과적으로 사용할 수 있도록 단계별 점검항목 등을 사전에 정의하는 것이 필요하다.

• QA그룹은 주요 업무외에도 새로운 동향 및 기술의 지속적인 파악과 EDP인원의 교육, 응용프로그램의 기술적 문제 해결, 시스템 컨설팅, 개발자와의 여러 가지 마찰의 해결 등의 역할을 요구받게 된다는 것을 미리 고려하여야 한다.

• QA계획에 참조 가능한 표준은 다음과 같다.

- IEEE-std 730 : 소프트웨어 품질 보증 계획서 표준
- IEEE-std 828 : 소프트웨어 형상관리 계획서 표준
- IEEE-std 829 : 소프트웨어 시험 문서 표준
- IEEE-std 983 : 소프트웨어 품질 보증 계획 표준
- IEEE-std 1012 : 소프트웨어 확인 및 검증 계획서 표준
- DOD-std-2167a : 소프트웨어 개발 표준
- DOD-std-2168 : 소프트웨어 품질 표준

소프트웨어 품질 실현을 위해서는 필요성 인식이 선행되어야 한다. 현재 품질관리가 정착되지 못하고 있는 가장 큰 이유는 품질관리에 소요되는 비용이 추가적 부담이라고 인식되는 것이다. 경영층은 사용자의 요구에 부응하기 위해 그리고 경쟁력을 높이기 위해 갖추어야

하는 기술로서 품질관리를 인식하고 이에 대한 투자를 할 수 있어야 한다. 개발 관리자는 품질실현을 위해 이를 경영층과 개발 요원에게 설득해야 하며, 기술을 갖춘 전문가 그룹에 의해 신뢰성있는 결과가 나올 수 있다는 사실을 인식하여야 한다. 품질관리 활동에 의한 비용증가 문제는 소프트웨어 개발주기의 비용효과 분석에서 소프트웨어 생명주기와 조직의 지속적인 비용효과를 고려하여야 한다.

품질관리와 V & V 그리고 시험 활동은 소프트웨어 생명주기의 전 과정에 걸쳐 수행되어야 하는 활동으로 각 조직은 단계별로 수행될 품질보증, V & V 활동을 정의하여 통합된 개발 공정을 정의해야 한다. 품질 구현 과정에서 자동화 도구의 이용은 각 과정과 산출물에 대한 생산성, 품질 향상에 크게 기여를 할 수 있다.

3. 품질 관리 기법

품질 관리 활동에서 사용될 수 있는 효과적인 기법은 정형적 검사와 품질 메트릭의 활용이다. 이 절에서는 정형적 검사와 품질 메트릭에 대해 점검해 본 후, 신뢰성있는 소프트웨어 구현을 하기 위한 새로운 기법으로 주목을 받고 있는 Cleanroom 공학 기법을 분석한다.

가. 정형적 검사 기법

정형적 검사 기법은 1976년 미국의 FAGAN에 의해 보고된 후, 널리 적용되고 있으며 많은 기관에서 검사 기법의 효과에 대해 보고하고 있다. V & V활동은 주로 검토와 시험 기법을 이용하여 이루어지고 있으나 오류의 검출과 교정에서 검토에 비해 검사 기법이 월

등하므로 검토 기법을 점차 검사 기법으로 바꾸어 나가는 것이 필요하다. 검사는 시험보다 오류의 발견 비용 및 오류의 제거비용이 경제적이며, 특히 오류제거에 있어서 시험보다 2-10배 효율적이다. 또한 소프트웨어 개발과정을 관리하는 측면에서도 신뢰할 수 있는 결과를 제공한다.

검사의 단계중 준비 단계가 소홀히 되고 있는 경우가 많은데 실제 회의에서 바로 검사를 시작 하기 위해 검사 대상 시스템과 체크리스트를 철저히 이해할 수 있도록, 중재자는 준비 단계를 비롯한 검사의 개시 기준(Entry Criteria)을 만족하는지 여부를 확인하여야 한다. 검사 후에는 검사의 효과를 분석하기 위해 다음과 같은 메트릭 정보를 기록하고 관리하여야 한다.

- 단위 검사물당 평균 준비 시간
- 단위 검사물당 평균 결함의 수
- 결함당 투입노력

단위 검사물은 보통 KLOC의 단위로 구분하며 여러 적용 결과를 보면 검사를 통하여 KLOC당 평균 8-12개 정도의 결함이 검출되고, 주요 결함당 3-5시간의 인력이 소요되는 것으로 나타나 있다.

나. 품질 메트릭의 적용

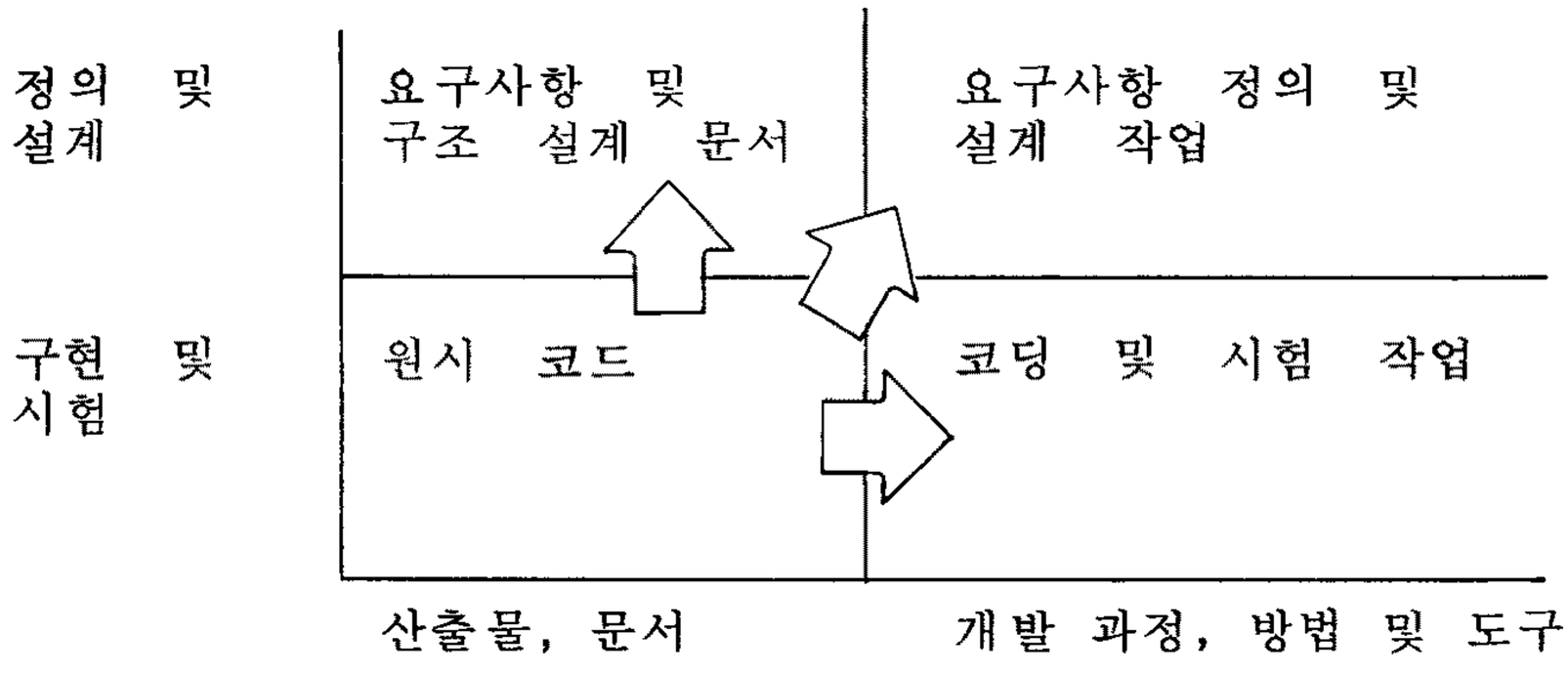
QA과정과 V & V과정에서 산출되는 결과는 정성적인 경우가 많은데, 이러한 경우 QA의 계획을 구체적으로 수립하거나, 품질 기준의 설정, 품질 평가등에 객관적인 판단이 어려운 경우가 발생한다. 이를 개선하기 위하여 개발 프로젝트의 정량적 측정 결과인 메트릭을 이용할 수 있다. 측정하지 않은 것은 관리할 수 없다는 말이 메트릭

의 중요성을 잘 나타내주고 있다. QA와 V & V기능은 메트릭을 이용하여 일정 품질 기준에 미달하는 소프트웨어를 검사하여 개선 작업을 지시하고 이러한 결과를 이용하여 품질 개선을 위한 실질적 계획 수립을 할 수 있다. 메트릭의 또 다른 잇점은 개선 과정의 품질 향상 정도를 정량적으로 측정할 수 있다는 것이다. 소프트웨어를 용역 개발하는 경우에도 메트릭을 같은 용도로 사용할 수 있다. 메트릭을 사용하기 위해서는 먼저 측정 대상과 측정 방법의 골격을 정의한 후 프로젝트 특성에 맞게 메트릭을 조정해야 한다. 측정의 목적은 대상의 이해, 평가, 예측, 관리, 개선등이 될 수 있다. 측정 방법은 직접측정(예, 개발 소요시간 측정)과 간접 측정(예. Function point를 이용한 비용산정), 실체적 측정과 추상적 측정, 객관적 측정과 주관적 측정등이 있으며 측정에는 계획, 자료 수집, 수집 자료 검증, 결과의 해석등에 많은 노력이 소요된다. 각 개발 조직에서는 측정을 통하여 계속적으로 조직의 업무를 개선하도록 측정 절차 계획과 수행, 측정치 분석 및 개선을 아래와 같이 개발에 통합된 형태로 추진하면 바람직하다.

1. 프로젝트 환경 특성 파악
2. 개선 목표 기술
3. 프로젝트 계획(적절한 방법론 및 도구 선택), 측정 절차 계획
4. 프로젝트 수행, 자료 수집, 검증
5. 측정치 분석
6. 1단계부터 계속

메트릭을 원하는 시간에 적절히 이용하려면 자동화가 필요하며 이를 데이타베이스에 저장하여 조직에서 계속적으로 유지 관리할 필요

가 있다. 메트릭의 측정 대상은 코드나 문서와 같은 산출물, 개발과정 그리고 개발 환경등이 될 수 있다. 현재의 메트릭은 코드와 시험에 집중되어 있으나 점차 개발 과정과 산출물의 여러 분야로 메트릭을 확장하는 연구가 이루어지고 있다. [그림 2-7] 메트릭에 관한 상세한 연구 내용은 [이 89]에 기술되어 있다.



[그림 2-7] 메트릭 분야의 확장

다. Cleanroom 공학

Cleanroom공학은 신뢰성 있는 소프트웨어 개발을 위한 조직의 구성과 오류의 사전 방지에 강조를 두고 있는 기법이다. 이것은 소프트웨어의 개발 단계에서 오류가 개입될 수 없도록 하는 기법과 팀웍을 강조하며 "Cleanroom"이라는 단어가 이 정신을 잘 나타내고 있다. Cleanroom 공학 기법의 특징은 통계적 방법에 근거한 독립적 시험과 프로그램을 수행하지 않는 개발 기법 및 시스템을 수행 가능한 산출물 단위로 구분하여 점진적으로 개발하는 것이다. Cleanroom기법의 특성은 아래와 같다.

- 개발과 시험을 서로 다른 팀이 책임진다.
- 개발 과정중 Rigorous한, 수학적인 공학 기법을 적용하여 통제

한다.

- 품질에 대해 팀이 책임을 지는, 그리고 오류는 허용되서는 안된다는 자세를 가진다.
- 통계적 품질 관리 기법의 적용을 통해 소프트웨어의 신뢰성을 보증한다.

Cleanroom기법의 조직은 명세화 팀, 개발 팀, 보증 팀으로 기능이 구분된다. 명세화 팀은 명세를 준비하고 유지보수하며, 개발팀은 소프트웨어를 설계하고 구현한다. 보증팀은 소프트웨어를 컴파일하고 시험하여 품질 평가를 한다. 개발은 다음과 같은 과정으로 이루어진다.

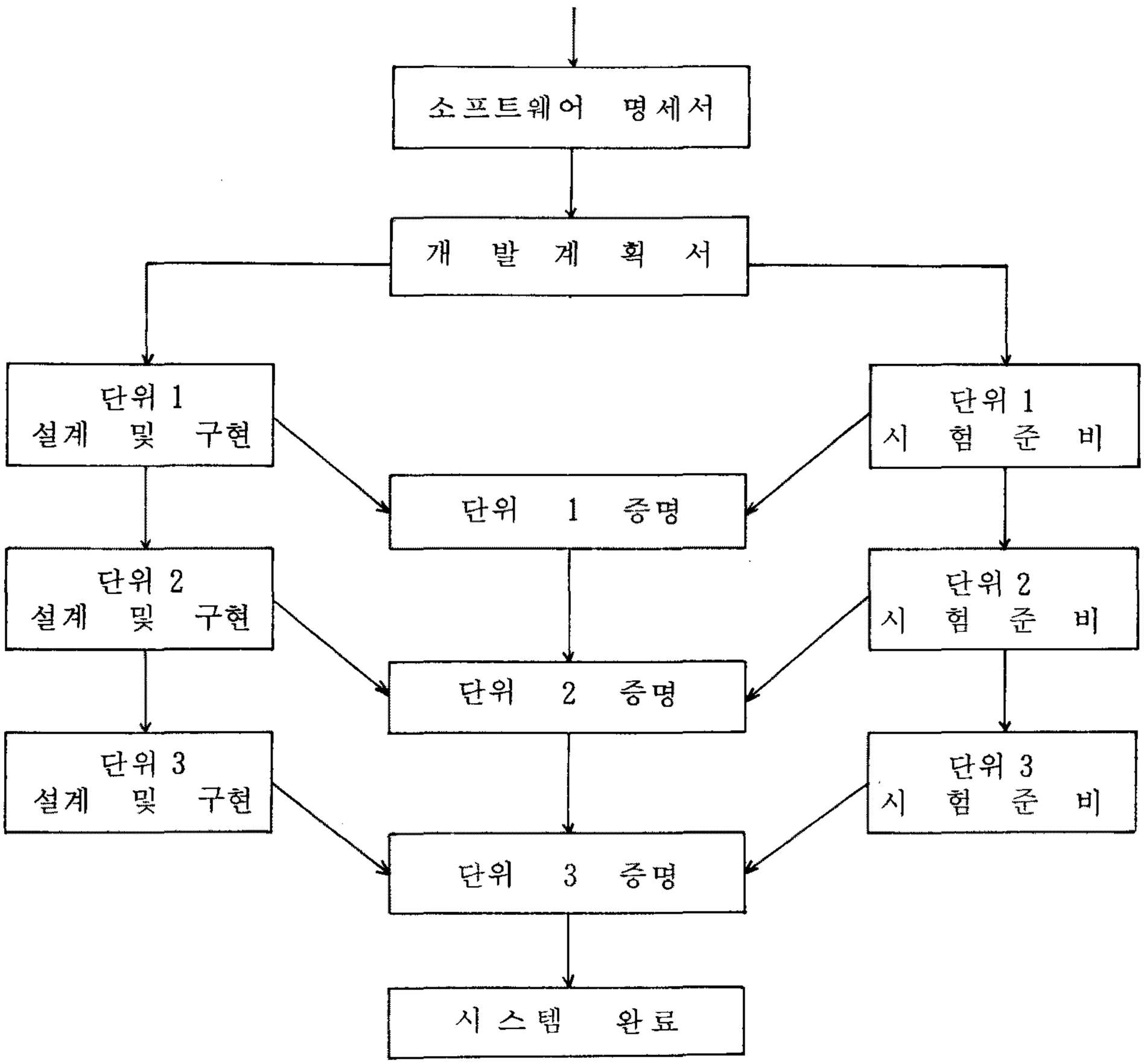
- 정형적이고 rigorous한 명세서를 완성, 정보가 불충분하여 명세를 완성할 수 없는 경우에도 공식적 문서를 작성한다.
- 명세서를 분석하며, 수행 가능한 단위로 분해하여 개발 계획서를 작성한다.
- 각 단위를 설계하고 구현한 후 확인과정 (Verification)을 거친다.
- 품질을 평가한다.

[그림 2-8]은 여러 단계로 나누어진 통상적인 개발 과정을 나타내고 있다.

각 개발 단계별 진행 과정은 다음과 같다.

(1) 명세화 작업

분석된 요구사항을 명세서로 만든후 상세한 사항을 구체화 하는 과정을 거쳐 정형적인 명세서를 완성한다. 초기 버전은 정보의 부족으로 개략적일 수 있으나 이러한 경우에도 공식적인 명세서로 완성하여 개발 시작 전에 승인 절차를 거친다. 명세서는 외부 명세서, 내부 명세서, 예상되는 운영 프로필의 세가지 부분으로 이루어진다.



[그림 2-8] Cleanroom기법의 점진적 개발 단계

외부 명세서는 사용자의 관점에서 소프트웨어의 외부적 형태와 특징, 모든 사용자와 소프트웨어간의 인터페이스 등 다음과 같은 사항에 대해 상세히 기술한다.

- 시스템 환경 (하드웨어, 주변장치, 운영체제, 관련 소프트웨어, 인원 등)
- 적용 환경 (자료 및 자료 이용 구조)

- 초기 작업과 작업 완료시의 절차 등
- 시스템 이용 (지령, 메뉴, 이벤트, 모드등)
- 사용자, 컴퓨터, 외부 장치의 모든 자극 (stimulus) 에 대한 시스템의 응답
- 응답시간, 정밀도등의 성능 기준
- 예외 사항에 대한 시스템 상태

외부 명세서의 작성 목적은 소프트웨어의 동작 사항을 정확하기 기술하기 위한 것이다. 사용자가 이해할 수 있는 언어로 기술되어야 하지만 사용법 설명서와 같은 형식으로 작성하는 것은 아니다. 내부 명세서는 프로그램을 구현하고 이것의 정확성을 증명할 수 있도록 수학적으로 기술되며 구현 방법에는 독립적으로 기술된다. 외부 명세서에서 기술된 시스템의 자극과 응답을 내부 명세서에서는 하나의 응답이 일어나기 위한 시스템의 자극 이력으로 표현한다. 이러한 방법의 명세화 기법은 초기에는 습득하기 어려운 점이 있다. 그러나 하나의 응답을 그 응답이 발생하기 위한 상태의 자료로 표현하는 경우에는 구현 지식이 개입 되는 것이므로 허용되지 않는다. 예상 운용 프로필은 소프트웨어의 기대되는 사용을 정의하는 것으로 이 문서는 시험 준비의 기준이 된다. 소프트웨어의 예상되는 신뢰도를 측정하기 위해서는 모든 가능한 자극 집단중 일부를 선택하여 시험을 설계하고 수행한다. 그러나 시스템의 사용 비율은 각 자극 집단에 따라 달라질 수 있으므로 이 비율에 따라 비례하게 사례를 개발하여야 한다.

(2) 시스템 구축

이 단계에서는 단계와 증명 절차를 결정한다. 먼저 명세서를 수행

가능한 단위로 분해한다. 각 단위는 사용자 지령이나 외부 자극에의 해 시험 가능해야 한다. 구축 순서를 결정하기 위한 기준은 다음 상황에 따라 달라질 수 있다.

- 재이용 가능한 소프트웨어 활용여부와 신뢰도
- 각 단위의 크기
- 개발 팀

점진적 개발은 새로운 개념은 아니다. 여기서 중요한 아이디어는 구축 계획서의 모든 요소는 사용자 지령에 의해 수행 가능해야 한다는 점이다. 이것을 만족시키기 위해서는 시스템이 하향식으로 개발되어야 한다. 이 경우 시험을 위한 구동기나 스터브의 개발이 필요없다는 잇점이 있다. 각 단위가 구축될 때마다 점진적 통합시험이 필요하다. 명세서가 단위로 분해된 후에 설계와 구현, 시험이 시작된다. 이 단계는 병렬로 진행될 수 있으며, 한 단위의 설계와 구현은 개인이 아닌 개발 팀의 책임이라는 인식을 가진다. 개발팀은 각 요소를 구축하기 위해 박스 구조와 단계별 정제 (Stepwise Refinement), 기능적 확인 (Functional Verification) 기법을 이용한다.

개발 과정에서 각 요소의 운용 계층을 3가지 관점으로 기술한 후 정확성을 확인한다.

- 블랙박스 관점
  - stimulus 정의
  - stimulus history로 응답 정의
- 상태박스 관점
  - stimulus history를 표현하기 위한 상태 자료 정의
  - 현재 수준에서 유지해야할 상태 자료를 선택한다.

- 블랙박스 기술내용을 수정하여 응답을 stimulus와 이 수준에서 유지할 상태 자료로 표현
- 클리어 박스로 가기전에 상태 박스 기능의 상태 자료에 대한 참조를 제거하여 블랙 박스와 동일한 가를 확인한다.

- 클리어박스 관점
- stimulus에 의한 상태 자료의 접근 유형 기록
- 각 상태 자료를 ADT를 선택한다.
- 상태 박스를 변경하여 응답을 stimulus와 이 수준의 상태 자료와 하위 수준의 블랙 박스 호출로 표현
- 클리어 박스 완료시 하위 수준 블랙 박스 참조를 제거하여 상태 박스와 확인한다.

위와 같은 절차로 모든 블랙 박스를 확장한 후 설계가 완료된다. 개발팀은 코드를 컴파일하거나 시험하지 않고 각 단위의 정확성을 나타내기 위해 수학적 증명과 기능적 확인을 수행한다. 시험과 측정은 시험팀의 임무이다.

(3) 시스템 시험

시험팀은 외부 명세서와 예상되는 사용 프로필을 이용하여 사례를 개발한다. 시험팀은 각 단위에 대해 운영 프로필에서 사례를 무작위로 수행하여 현재 시스템의 신뢰도를 측정한 후 실패 상황에 대해 개발팀의 보완을 요구한 후 보완된 새로운 버젼의 시험으로 신뢰도를 측정한다.

Cleanroom기법을 적용하여 개발된 여러 시스템에서 품질과 생산성이 향상된 것으로 보고 되었다[RIC 90].

제 5 절 관련 기술

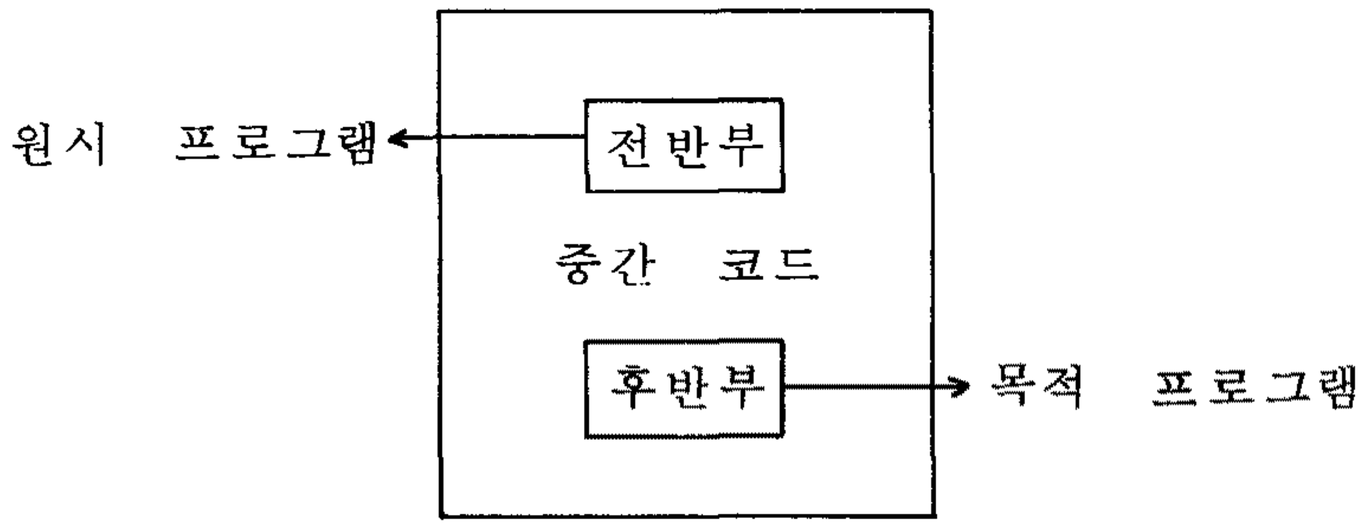
1. 컴파일러와 정적분석기

컴파일러는 고급언어로 쓰여진 프로그램을 그 프로그램이 실행될 컴퓨터에서 직접 실행가능하도록 어셈블리어나 기계어 프로그램으로 된 목적 프로그램으로 변환해 주는 변환기로서 프로그램의 실행 측면에 주안점을 두는 반면, 정적 분석기는 실제로 프로그램을 실행하지 않고 원시 프로그램의 구조적 측면, 즉 프로그램 논리상의 전반적 특성을 분석하여 주는 도구이다. 따라서 컴파일러의 여러 단계들중 목적 프로그램으로 변환해 주는 코드 생성 단계는 불필요하고, 단지 원시 프로그램을 읽고 논리 분석만을 하게 된다. 여기서는 먼저 컴파일러의 각 단계에 대해서 기술한 후 정적 분석기에 대해 이들각 단계들 중 정적 분석을 위해 필요한 단계를 정의하여 정적 분석기와 비교하여 보고자 한다.

가. 컴파일러

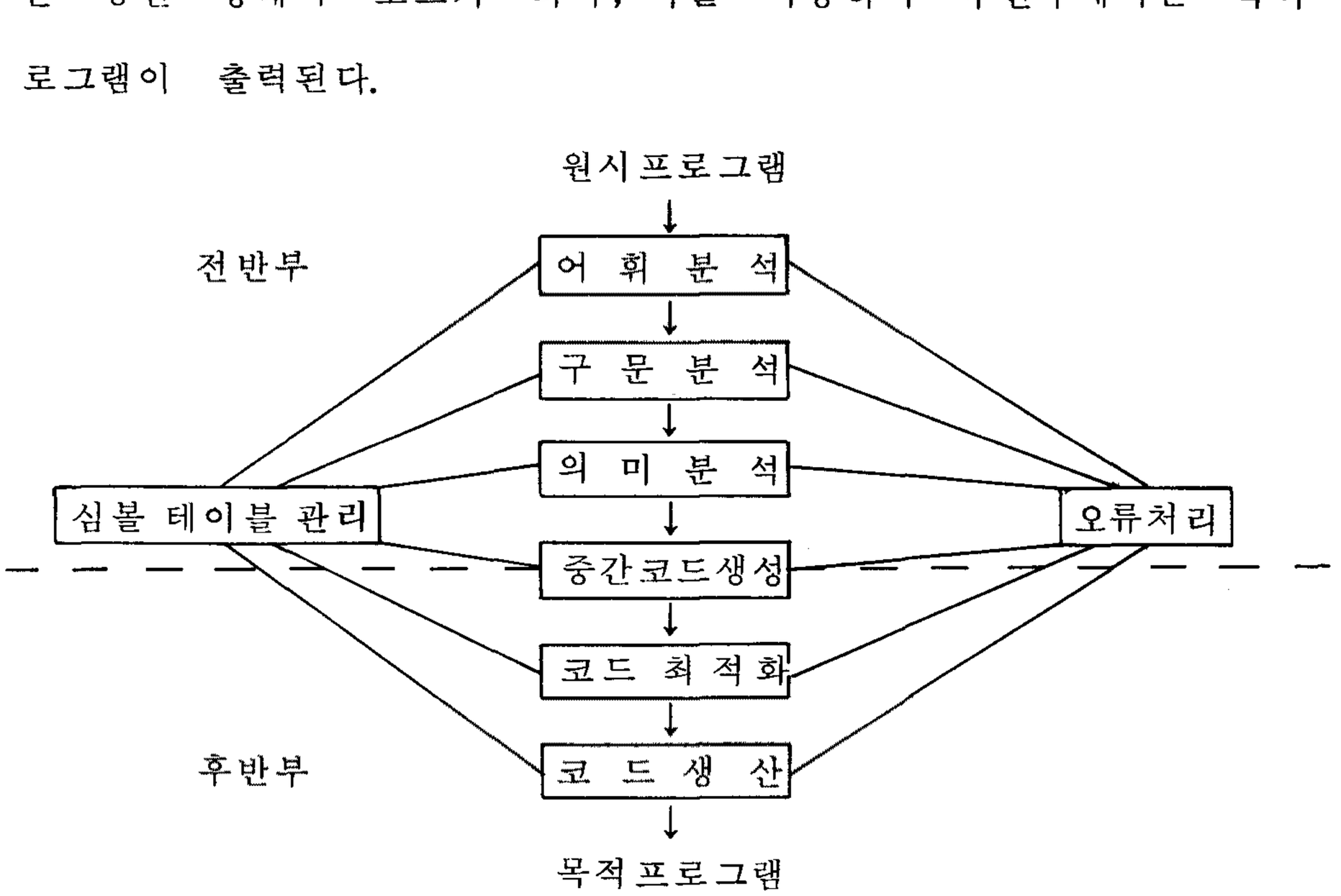
컴파일러는 포트란, 코볼 등의 고급언어로 쓰여진 프로그램을 어떤 특정한 컴퓨터에서 직접 수행가능한 형태의 어셈블리어나 기계어 프로그램으로 번역해 주는 컴퓨터 프로그램이다. 이는 [그림 2-9]와 같이 크게 전반부와 후반부로 나눌 수 있는데 전반부는 원시 언어에 관계되는 부분으로 원시 프로그램을 분석하고 중간 코드를 생성하는 부분이며, 후반부는 전반부의 원시 프로그램에 대한 부분을 종합하여 특정 기계에 대한 목적 프로그램을 생성하는 부분이다. 전반부는 문법 이론에 의해 잘 정립된 반면, 후반부는 아직도 경험적인 방법을

통하여 구현하며 계속적인 연구가 진행되고 있는 부분이다.



[그림 2-9] 컴파일러의 개략적 구조

이를 기능별로 세분하면 [그림 2-10]과 같이 6단계로 구성되며 전반부는 원시 프로그램을 읽어들이는 어휘 분석 단계로부터 중간 코드를 생산하는 중간 코드 생성 단계까지이다. 따라서 전반부의 출력은 중간 형태의 코드가 되며, 이를 이용하여 후반부에서는 목적 프로그램이 출력된다.



[그림 2-10] 컴파일러의 각 단계

(1) 어휘 분석 단계 (Lexical Analysis Phase)

컴파일 과정의 첫번째 단계인 어휘 분석 (Lexical Analysis) 단계는 검조 (Scanning) 단계라고도 부르며, 원시 프로그램 (Source Program) 을 한자씩 읽어서 각 변수, 상수, 연산자들을 토큰 (Token) 이라는 문법적 단위들로 원시 프로그램을 변환한다. 원시 프로그램은 긴 문자들의 열로 생각되며 어휘 분석기의 역할은 이 문자들을 차례로 읽어서 토큰 단위로 묶어내게 된다. 이 때 각 토큰의 형태는 프로그래밍 언어 설계자나 컴파일러 설계자에 의해 결정되는데 대개의 프로그래밍 언어는 다음과 같은 토큰 종류를 갖는다.

- 일반형태

  1. 명칭 (Identifier)-FIRST, SECOND, SUM, MUL, M, N 등

  2. 상수 (Constant)-15, 9, 'abc' 등

- 특수형태

  3. 지정어 (Keyword)-if, for, begin, goto 등

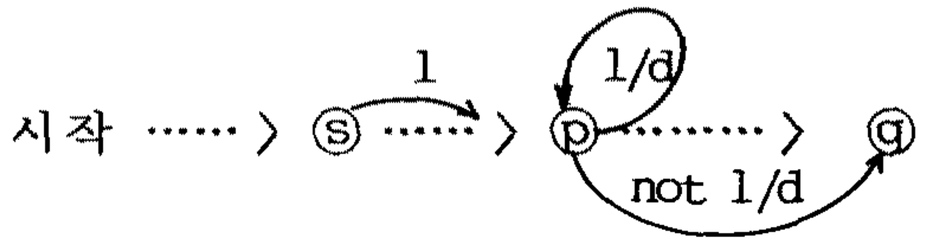
  4. 연산자 기호 (Operator symbol)-+,−,*,/,〈,〉,:=등

  5. 구분자 (Delimiter)-(,[,;,:,} 등

이들은 효율적인 처리를 위해서 고유의 내부 번호 (Token Number) 를 가지며 일반 형태의 토큰은 프로그래머가 사용한 값을 갖는데, 이 값을 토큰값 (Token Value) 이라 부른다. 이러한 토큰들은 유한 오토마타 (Finite Automata) 에 의해 인식되기 위해 정규 표현 (Regular Expression) 으로 표현되어야 하는데 이것은 우선 오토마타를 정규 문법으로 바꾸고 그 정규 문법을 풀음으로써 얻어진다.

이 과정을 명칭에 대하여 간단히 나타내면 다음과 같다. 먼저 프로그래머가 사용한 명칭을 길이에 관계없이 인식하기 위한 상태 전

이도는 [그림 2-11]와 같이 되며, 정규 문법을 거쳐 정규 표현으로 바꾸는 과정을 함께 나타내었다.



정규 문법 :s→lp　p→lp | dp | ε

s = lp

p = lp+ dp+ε =(l+d)p+ε=(l+d)*

∴ s = l(l+d)*

[그림 2-11] 명칭의 인식

처음에 상태 s에서 출발하여 원시 프로그램 속에서 Letter 를 읽고 p상태로 가며 다음 문자를 읽고 Letter이거나 Digit 이면 다시 p의 상태가 되어 이것을 반복하다가 Letter 또는 Digit 를 제외한 문자가 나오면 그 명칭을 인식하게 된다. 모든 가능한 토큰들을 결정한 후 각 토큰들에 대해서 각각의 정규 표현 형태에 따라 원시 프로그램 속에서 이것들을 식별해 내기 위한 인식기가 필요한데, 컴파일러의 어휘 분석기는 철회없는(Nonbacktracking) 프로그램으로 만들어야 하며, 이는 상태 전이도가 한 심볼에 의해 여러 곳으로 분기하는 비결정적 유한 오토마타(Nondeterministic Finite Automata:NFA)방식을 지양하고 한 방향으로만 가는 결정적 유한 오토마타(Deterministic Finite Automata:DFA)를 구성하며 프로그래밍하면 된다. 이 방식은 정규 표현으로부터 비결정적 유한 오토마타로, 비결정적에서 결정적 유한 오토마타로, 그리고 이것으로부터 상태수가 최소화된 결정적 유한 오토마타를 얻어 프로그래밍한다. 이렇게하여

인식된 토큰들을 어휘 분석기는 다음 단계의 구문 분석기 (Parser)에게 그 토큰의 해당 토큰 번호와 토큰값의 순서쌍을 넘겨준다. 예를 들어 다음과 같은 문장을 생각해 보자.

A : = B + 10;

어휘 분석 단계에서는 앞에서 설명한 오토마타에 의해 이 문장을 A라는 명칭, 치환 기호인 :=, B라는 명칭, 덧셈기호 +, 상수 10, 구분자 ;의 6개의 토큰으로 분리한다. 즉 어휘 분석 단계의 출력은 이러한 일련의 토큰들이다.

이러한 토큰들 중 A나 B와 같은 명칭들과 상수 10은 [표2-5]와 같은 심볼 테이블에 등록되어 의미 분석과 코드 생성 과정에서 중요하게 이용되며, 심볼 테이블에서의 위치가 이들의 토큰값이 되어 구문 분석기로 넘겨주게 된다.

[표 2-5] 심볼 테이블

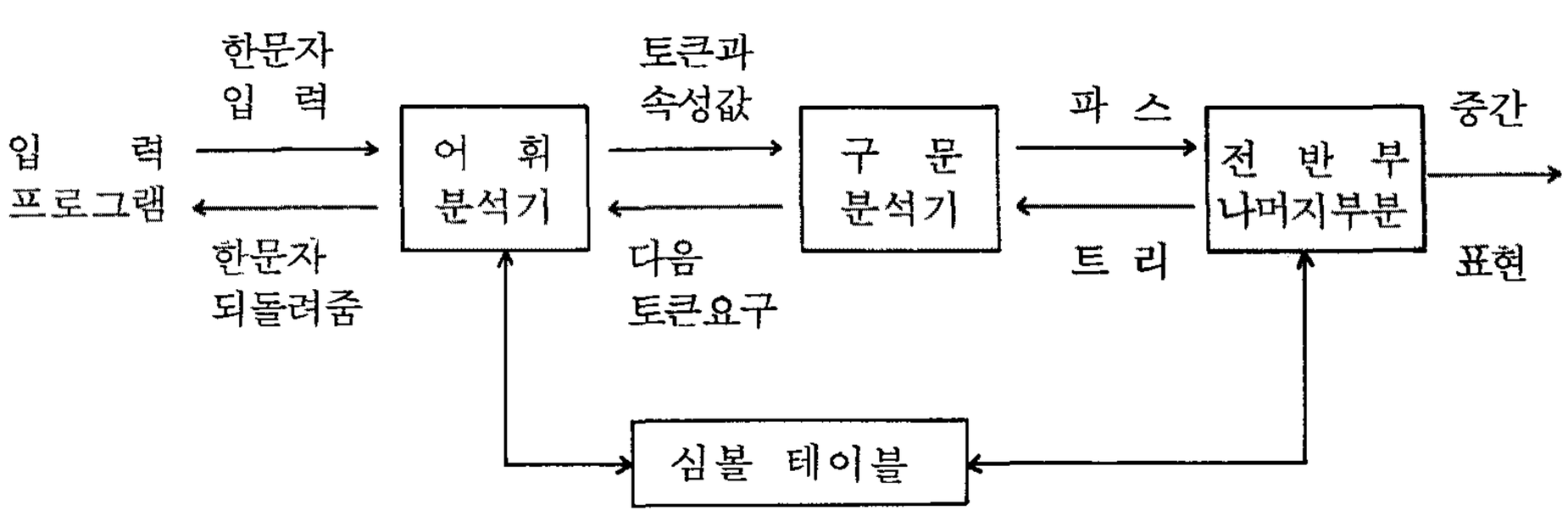
위 치	명 칭	종 류	값
	:		
4	A	id	
5	B	id	
	:		
20	10	Constant	
	:		

만일 연산자들의 해당 토큰 번호가 [표 2-6]과 같다면, [표 2-5]와 [표 2-6]을 이용하여 위의 문장은 〔4,4〕, 〔1,0〕,〔4,5〕,〔2,0〕,

[표 2-6] 토 큰 표

토 큰	코 드	값
:=	1	
+	2	
;	3	
id	4	변수 심볼테이블 위치
Constant	5	상수 심볼테이블 위치

〔5,20〕,〔3,0〕와 같은 토큰들의 행렬로 구성되어 구문 분석 단계의 입력이 된다. 즉 어휘 분석기와 구문 분석기와의 관계는 [그림 2-12]와 같이 도식화할 수 있다.



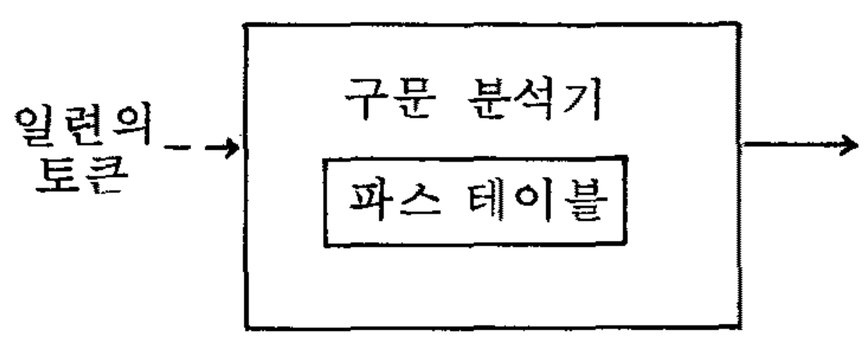
[그림 2-12] 어휘 분석기와 구문 분석기와의 관계

이와 같은 심볼 테이블 운영은 구문 분석이나 의미 분석에서도 행해져 그때마다 중요한 정보를 제공한다. 그리고 다른 기호들도 각각의 고유한 해당 토큰 번호에 대응되어 다음과정인 구문 분석 단계로의 입력이 된다. 그 밖에도 어휘 분석기는 원시 프로그램의 줄 번호를 기억하여 필요할 때 원시 프로그램을 줄별로 출력할 수 있도록 해주며, 프로그래머가 프로그램의 설명을 위해 쓴 주석 (Comme-

nt) 문과 공백은 이 과정에서 모두 처리되어 다음 단계에서는 무시된다. 또한 어휘 분석기는 토큰 인식 작업뿐만 아니라 간단한 오류 처리도 한다. 예를 들어, A:= 1st+2nd; 라고 쓴 문장에 대해 1st나 2nd는 정당한 변수명이 될 수 없으므로(숫자로 시작되었으므로) 오류를 내주게 된다.

(2) 구문 분석 단계 (Syntax Analysis Phase)

이 단계는 흔히 파싱 (Parsing) 이라고 하며 이 단계를 수행하는 컴파일러의 부분을 파서 (Parser) 라고 부른다. 파서가 하는 일은 어휘 분석 단계의 출력인 토큰들을 받아 원시 프로그램에 대한 오류를 검색 (Error Checking) 하고, 정의된 문법에 대해 올바른 문장에 대해서는 그 문장의 구조를 나타내는 구문 구조 (Syntatic Structure) 를 만든다. 이런 구문 구조는 종종 토큰들을 단말 노드 (Terminal Node) 로 하는 트리 (Tree) 형태로 표현되는데, 이 트리를 파스 트리 (Parse Tree) 라 한다. 이 파스 트리는 엄밀하게 정의된 문법 (Grammar) 에 의해 이루어진다. [그림 2-13] 은 이와 같은 과정을 나타낸다.



[그림 2-13] 구문 분석기의 기능

이렇게 생성된 파스 트리에 대해 나중에 그 트리를 따라가면서 의미 분석과 코드 생성을 하게 된다. 이렇게 문법에 기초한 파스 트리를 통해 컴파일하는 과정을 문법 지시적 변환 (Syntax-directed

Translation)이라고 한다. 그러면 다음에서 파싱의 과정에 대해 좀 더 자세히 알아보자.

프로그래밍 언어의 문법적인 표현으로 Context-free 문법은 효율적이고 잘 정의된 구문 분석 알고리즘을 가지기 때문에 프로그래밍 언어의 표현에 가장 널리 사용되어 왔다. Context-free 문법에서의 언어는 문장 형태(Sentential form)의 스트링에서 생성 규칙을 반복적으로 적용하여 Nonterminal을 확장함으로써 얻을 수 있다. 예를 들어, 산술식에 대한 아래와 같은 문법 G를 고려해 보자.

$$G : E \rightarrow E+E \mid E * E \mid (E) \mid -E \mid id$$

여기서 E는 식(expression)을 나타내는데, 생성 규칙 $E \rightarrow -E$는 -가 식앞에 나오는 경우도 식이 될 수 있다는 것을 나타낸다. 즉 문장을 얻기 위해 시작 심볼 E에서 부터 반복적으로 생성 규칙을 적용하여 문장을 유도하여 보면,

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(id)$$

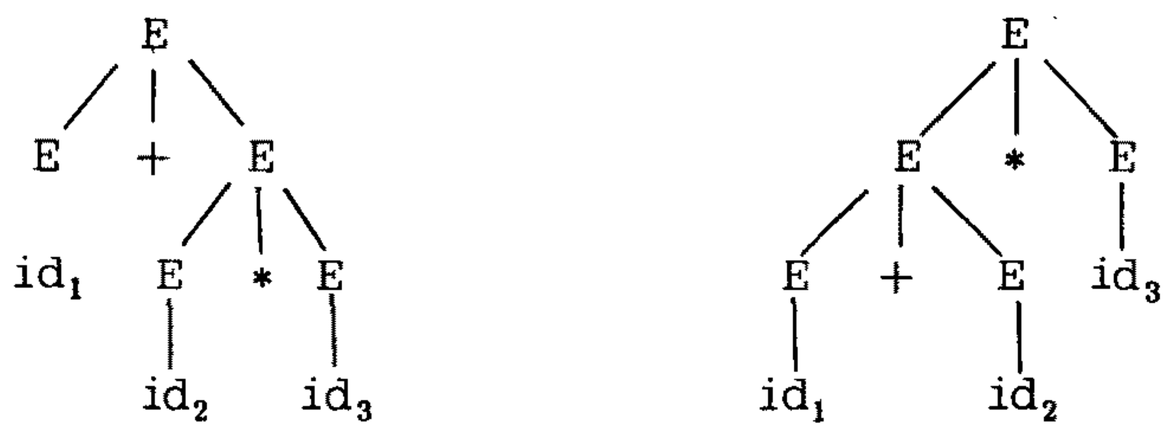
로 되어 -(id)가 문장이 될 수 있음을 보여 준다. Context-free 문법에서는 생성 규칙의 오른쪽에 Nonterminal이 여러개 나올 수 있으므로 같은 문장을 유도하는 데는 여러 방법이 있을 수 있는데, 유도 과정의 각 단계에서 문장 형태의 가장 왼쪽에 있는 Nonterminal을 먼저 대치하는 경우를 좌단유도(left most derivation)라 하고, 가장 오른쪽에 있는 Nonterminal을 대치하는 것을 우단유도(right most derivation)라 한다. 또 이러한 유도과정에서 적용되는 생성 번호들의 나열을 각각 좌파스, 우파스라 하며, 이들에 의해 생

성되는 트리를 유도트리라 하고, 이 때 나타나는 문장형태를 각각 좌문장 형태 (left-sentential form), 우문장 형태 (right-sentential form) 라 한다. 그런데, 어느 유도 방법을 이용하더라도 구성되는 유도 트리가 두 개 이상이 되는 경우가 존재한다. 예를 들어, 문장 $id_1 + id_2 * id_3$ 를 위의 문법 G에 의해 좌단유도를 하여 보면,

$E \Rightarrow E + E$
$\Rightarrow id_1 + E$
$\Rightarrow id_1 + E * E$
$\Rightarrow id_1 + id_2 * E$
$\Rightarrow id_1 + id_2 * id_3$

$E \Rightarrow E * E$
$\Rightarrow E + E * E$
$\Rightarrow id_1 + E * E$
$\Rightarrow id_1 + id_2 * E$
$\Rightarrow id_1 + id_2 * id_3$

와 같이 두가지의 유도 과정이 생기며, 이에 대응되는 유도 트리는 [그림 2-14]과 같다.



[그림 2-14] $id_1 + id_2 * id_3$ 에 대한 유도 트리

이와 같이 어떤 문법은 주어진 스트링에 대한 유도 트리를 두개 이상 그릴 수 있는데 이런 문법을 "모호하다"고 하며, 이런 문법은 여러가지 방법들에 의해 모호하지 않은 문법으로 변환되어 구문 분석된다. 뿐만아니라 주어진 문법에 대하여 효율적인 구문 분석을 하기에 적합한 형태로 문법을 변환하여 구문 분석한다. Context-free

문법을 위한 구문 분석 방법은 파스 트리를 어떤 순서로 만들어가느냐에 따라 Top-down 방식과 Bottom-up 방법으로 구분할 수 있다.

(가) Top-down 구문 분석

Top-down 방식은 시작 심볼(S)로부터 정의된 문법의 생성 규칙을 적용하여 유도 과정에 의해 주어진 스트링(w)과 같은 문장을 찾는 과정이다. 유도 과정중 적용되는 생성 규칙의 번호를 출력하거나 또는 부분트리 (Subtree)를 만들어 나간다. 스트링 (id+ id)를 Top-down 방식에 의한 파스 트리의 생성 예는 [그림 2-15]과 같다.

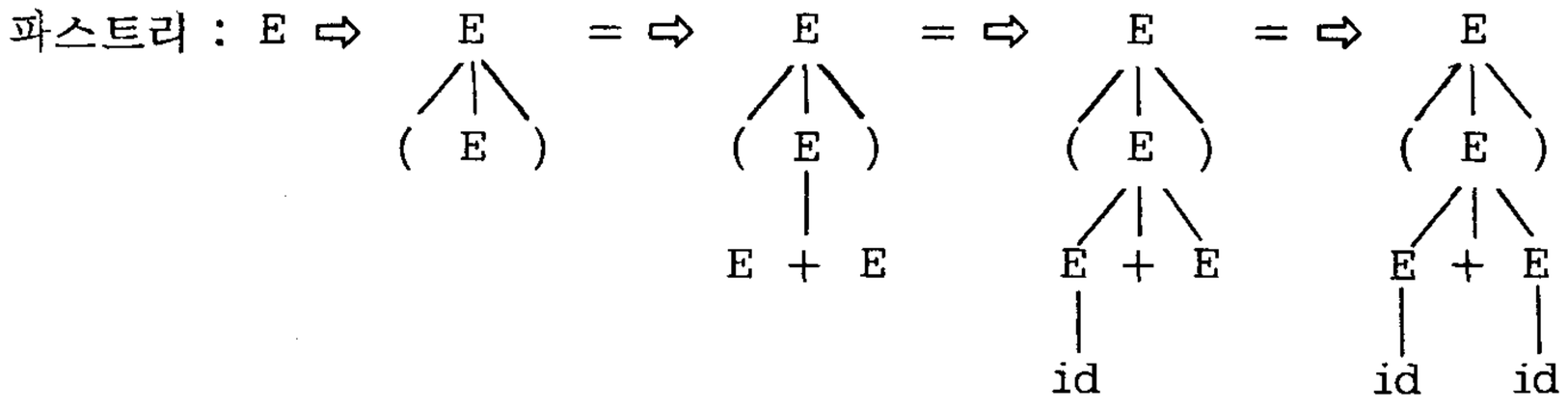
G : 1. E → E+E

2. E → E*E

3. E → (E)

4. E → -E

5. E → id



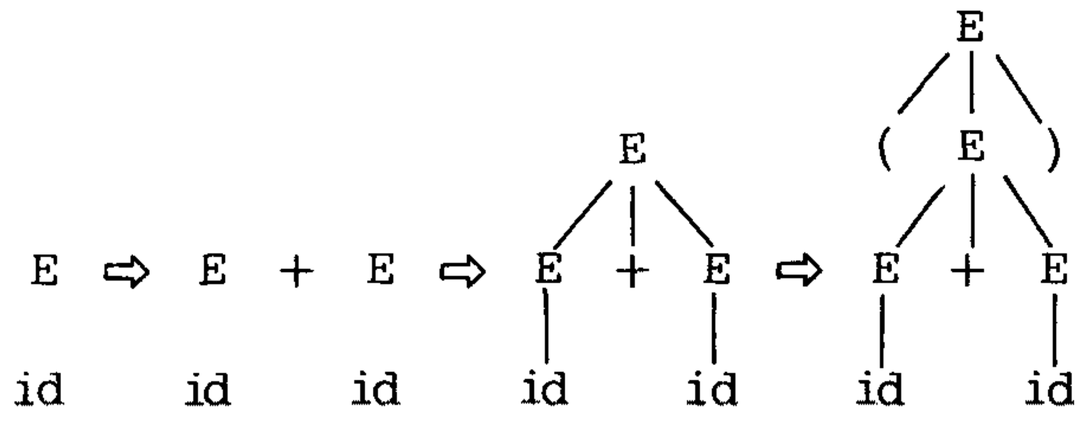
[그림 2-15] Top-down 파싱에 의한 파스 트리 생성

즉, Top-down 방법은 입력 스트링에 대한 좌측 유도 과정을 찾는 시도로 간주될 수 있으며, 이 과정에서 생성 규칙이 잘못 적용됐으면 그 생성 규칙에서 보았던 스트링을 다시 검조하기 위해 입력으로 보내고 다른 생성 규칙을 갖고 유도를 시작한다. 이와 같은 과

정을 Backtracking이라 부르며 한 입력 심볼을 여러번 반복하여 조사하므로 시간이 매우 많이 걸린다. 따라서 Backtracking을 하지 않고 결정적으로 파싱할 수 있는 방법, 즉 LL파싱이 필요하다. LL은 입력 스트링을 왼쪽에서 오른쪽으로 검조하며 (Left to right scanning) 좌파스 (Left parse)를 생성하는 파싱 방법이다. 즉, LL은 주어진 스트링과 같은 문장을 생성하기 위해 이제까지 본 심볼과 현재의 입력 심볼을 보고 적용될 생성 규칙을 결정적으로 선택하여 유도한다. 그리고 현재의 입력 심볼과 생성된 terminal심볼이 같지 않으면 주어진 스트링을 틀린 문장으로 간주하는 방법으로, Recursive-descent 파서와 Predictive 파서가 이에 속한다. Recursive descent 파서는 생성 규칙에 대한 프로시쥬어를 작성함으로써 구문 분석기를 구현한 프로그램으로 생성 규칙이 바뀔 때마다 구문 분석기를 구현한 프로그램을 다시 고쳐써야 하는 단점을 갖는 반면, Predictive 파서는 스택을 이용하여 구현한 방법으로 생성 규칙이 바뀌더라도 구문 분석기는 고치지 않고 단지 구문 분석기의 행동을 나타내는 파스 테이블만 고쳐서 구문 분석기를 구현할 수 있는 방법이다.

(나) Bottom-up 구문 분석

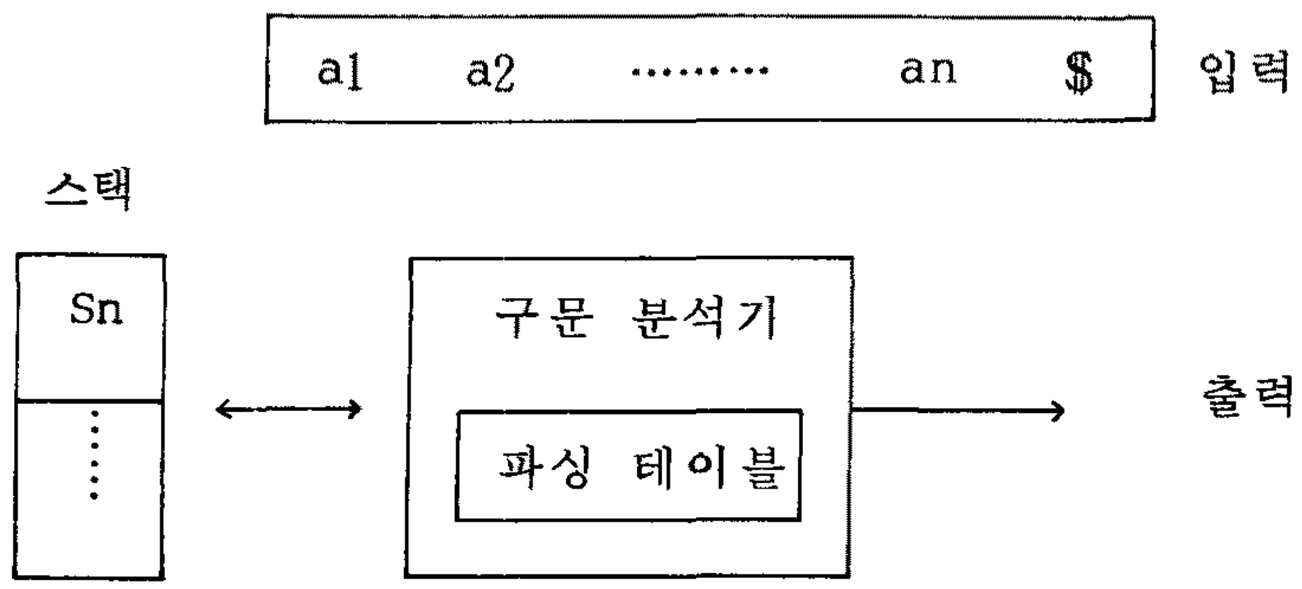
Bottom-up 방식은 주어진 스트링으로부터 Reduce를 계속하여 문법의 시작 심볼로 갈 수 있으면 올바른 문장이고, 그렇지 않으면 틀린 문장으로 간주하는 방법으로 reduce될 때마다 부분트리를 만들어 전체적인 파스트리를 구성한다. 문법 G에서 스트링 (id+id)에 대한 Bottom-up 방식의 파스트리 생성의 예는 [그림 2-16]와 같다.



[그림 2-16] Bottom-up 파싱에 의한 파스 트리 생성

Bottom-up 구문 분석 방법의 주안점은 유도 과정에서 나타난 문장 형태에서 reduce되는 부분(:handle)을 어떻게 찾아서 reduce할 생성 규칙을 어떻게 결정적으로 선택할 것인가 하는 점이다. 이 방식에서 Precedence Parsing은 문법 심볼들간의 순위 관계에 의해 reduce되는 부분을 찾아 구문 분석을 하는 방법인데, 문법의 제약점이 너무 많기 때문에 일반적인 프로그래밍 언어의 구문 구조를 표현하기에는 적당하지 못하므로 요사이는 대부분 LR 구문 분석 방법을 사용한다. LR 구문 분석 방법은 입력 스트링의 검조를 왼쪽에서 오른쪽으로(Left to right scanning)하며 파싱때 우파스(Right parse)를 생성한다. LR 구문 분석기는 [그림 2-17]과 같이 구성되며 스택의 Top과 입력 심볼에 따라 파싱 테이블내에서 Shift, Reduce, Accept 또는 Error와 같은 정보를 얻어 다음 파싱 행동을 결정한다.

이 LR 파서는 모호하지 않은 Context-free 문법의 모든 프로그래밍 언어에 대하여 구성이 가능하며 파싱 방법도 가장 보편성을 띠고 있으며, Top-down 파서에 비하면 실행 과정에서 여러 면으로 우월하며, 특히 입력 심볼을 읽는 과정에서 오류가 발생하면 바로 오류를 찾을 수 있어 강력한 구문분석 능력을 가지므로 컴파일러의 파



[그림 2-17] LR 파서

싱 알고리즘으로 가장 많이 쓰인다. 그런데 LR 파서는 자신의 파싱 테이블을 구성하는 방법에 따라 그 효율등에 약간의 차이가 있다. 가장 간단한 파싱 테이블은 SLR(Simple LR) 인데 이는 다른 방법보다 약하며, CLR(Canonical LR) 은 효율적인 방법이지만 실제로 작업을 하는데 어려운 부분이 있으며, LALR(Look Ahead LR) 은 SLR 과 CLR 의 중간 성격을 띠고 있는데, 이는 모든 프로그래밍 언어에 효율적으로 적용되는 것이 특징이다. 그러나, 실제로 정의된 문법에서 SLR, CLR 또는 LALR 방법에 의해 파싱 테이블을 손으로 직접 계산하여 얻는 것은 매우 복잡하고 어려우므로 대부분의 경우 자동 파서 제작 시스템을 갖고 이것을 이용하여 파싱 테이블을 구하여 구문 분석기를 구현하고 있는데, 이러한 파서 생성기로는 Unix 하에서의 파서 자동 생성기 YACC 가 있다.

(다) YACC(Yet Another Compiler-Compiler)

YACC 는 1975 년 Bell 연구소에서 Stephen C, Johnson 을 중심으로 개발된 파서 생성기이다. Unix 환경에서 수행되며, 생성 규칙이 LALA(1) 문법이면 구문 분석기를 자동으로 생성한다. YACC 의 사용자는 입력 자료 처리에 관한 표현을 제공해야 하는데, 입력 표현은

입력 구조를 나타내는 규칙과 규칙들이 인식되었을 때 호출되는 의미 수행 코드들, 그리고 입력을 처리하는데 필요한 기본적인 프로그램들로 구성된다. 그러면 YACC 는 입력 처리를 제어하는 기능을 가지는 프로그램을 생성한다. YACC 의 입력 표현 화일의 세가지 구성요소는 다음과 같다.

```
선언  부분 (Declarations)
%%
생성  규칙  부분 (Translation Rules)
%%
사용자  프로그램  부분 (User Program)
```

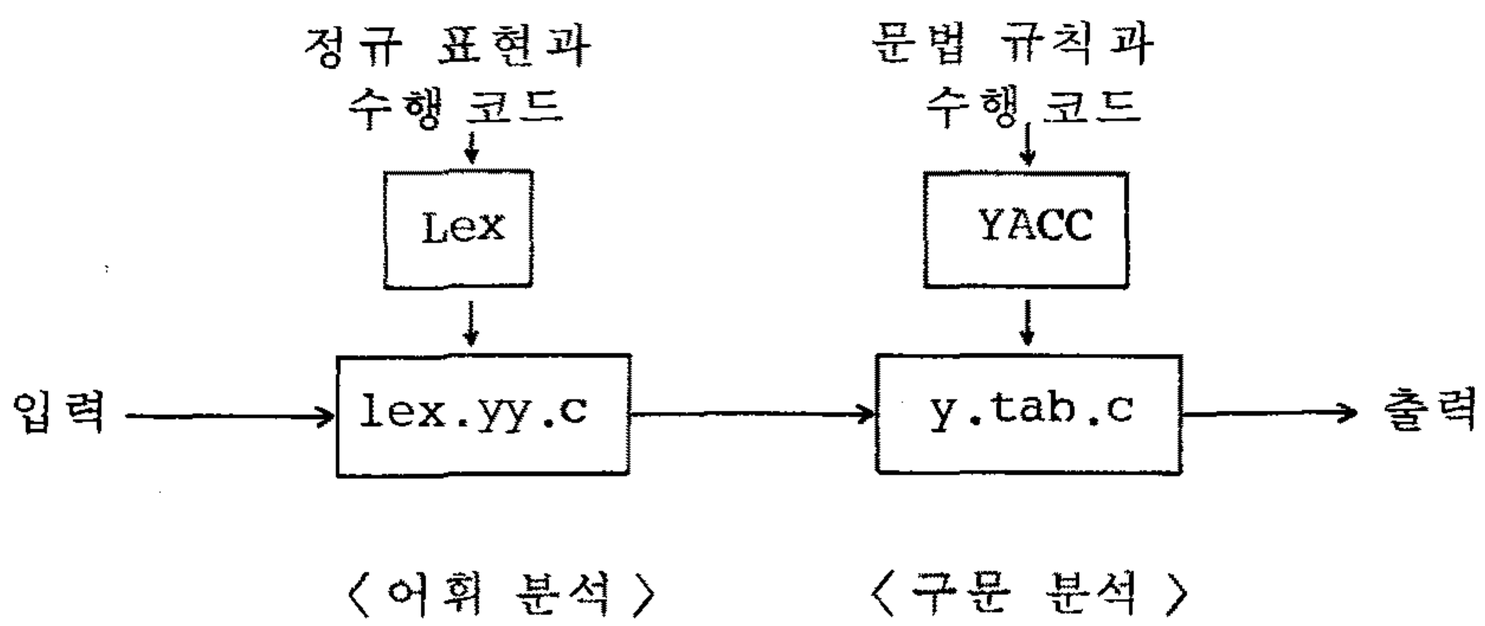
선언 부분은 문법 규칙의 토큰에 대한 선언과 변환 규칙 부분과 프로그램 부분에서 사용될 임시 변수들에 대한 선언을 하며 생성규칙 부분은 하나 이상의 문법에 대한 규칙들로 구성되는데 각 생성규칙은 Nonterminal 이름인 LHS 와 문법 규칙의 오른편에 나오는 심볼들로 이름이나 일련의 상수 문자 (Literal) 들로 구성된 Body의 형태를 취한다.

```
LHS: Body ;
```

YACC 의 각 규칙에 대한 의미 수행 코드들이 여기서 C 언어로 작성되며, 그 생성 규칙이 파서에 의해 인식되어 reduce 될 때 실행된다. 하나의 의미 수행 코드는 하나 이상의 문장들이 { } 안에 나타난다. 예를 들면, 다음과 같다.

```
X : YZ{printf("a message\n");
                flag= 25 ; }
```

또, 사용자 프로그램 부분은 YACC를 보조하는 C 루틴들로 오류처리 루틴등에 첨가된다. 또 YACC는 [그림 2-18]과 같이 어휘 분석기 자동 생성기 (Lex)와 연결하여 구문 분석을 자동으로 할 수 있다.



[그림 2-18] Lex와 YACC를 이용한 구문 분석

(3) 의미분석 단계 (Semantic Analysis Phase)

구문 분석에 의하여 입력 스트링에 대한 파스 트리가 생성되면 이 파스 트리를 운행하며 해당 생성의 의미에 따라 코드로 변환하게 되는데 이러한 코드를 중간 코드라 한다. 컴파일 도중에 구문 분석과 의미 분석을 하게 되는데, 의미 분석 단계에서 하는 일들은 대략 다음과 같다.

① 상수 정의 (Constant Definition)

상수의 명칭이 심볼 테이블에 들어가고 각 상수는 정의된 값을 심볼 테이블의 임의의 위치에 보관한 후 적당한 때에 기억 장소를 배정받고 초기값을 갖도록 해준다.

② 수형 정의 (Type Definiton)

주어진 문법의 생성들의 의미에 의해 수형의 해당 자료구조가 구성되며, 이 구성된 자료 구조를 보관하여 이후 이 수형의 구조를 인

용할 수 있도록 해 준다. 이를 위해 수형 정의 과정에서 구성된 자료 구조는 수형설명자를 만들어 구성된 수형의 자료 구조의 크기, 성격 등을 나타내주어 이 후 수형들이 기억될 때 필요한 정보를 제공해 주게 된다. 변수의 수형 선언에서는 선언되는 변수들이 선언된 수형에 의해 설명자를 얻어 변수 명칭과 수형설명자의 쌍이 심볼 테이블에 기억되며, 이 때 각 변수는 기억 장소를 할당받고 해당 수형을 구성하여 해당 변수들에게 기억 장소를 배정한다.

③ 프로시쥬어 선언 (Procedure Declaration)

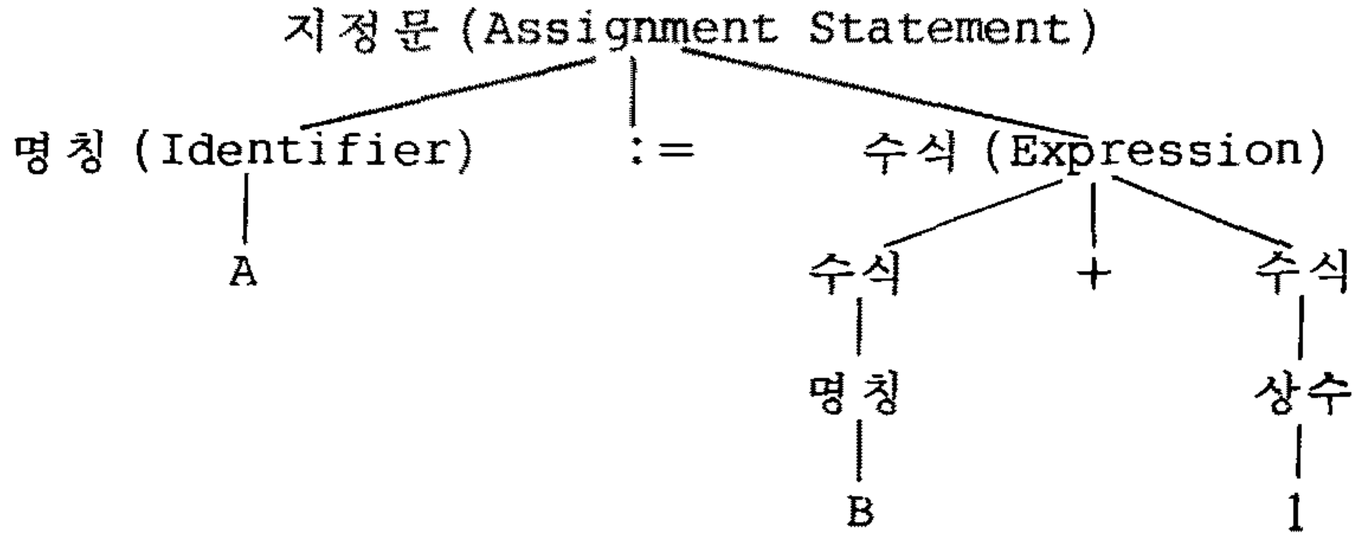
프로시쥬어가 나타나면 새로 시작된다는 뜻과 레벨을 하나 더 깊은 것을 나타내어 블럭구조 프로그램을 허용하게 된다. 즉,레벨의 변화를 관찰하여 변수나 인자 (Parameter) 들 사이의 변수 범위 (Scope Rule) 의 적용이 능률적이 되도록 하며, 블럭구조화된 프로그램을 지원해 준다.

④ 실행문의 의미 분석

선언문 이외의 실행문에 대해서는 심볼 테이블들을 이용하여 중간 코드를 생성해주게 된다. 즉 다양한 형태의 실행문들이 원래의 의미에 따라 해당 동작을 해 줄 수 있도록 중간 코드를 생산한다.

의미 분석 단계에서 하는 중요한 일 중의 하나인 수형 검사 (Type Checking) 에 대해 좀 더 자세히 설명하여 보면, 이 관계에서는 각 명칭의 수형에 따라 기억 장소를 배정해 주고, 그 명칭이 나오는 연산문에서의 각 연산자가 원시 언어의 정의에 맞는 피연산자 (Operand) 를 가지는가를 검사한다. 예로써, 대부분의 프로그래밍 언어에서는 실수가 배열의 첨자로 사용되었을 때 오류로 간주되어 진다. 또 어떤 언어에서는 실수와 정수의 연산을 허용하는 데 이때는 연산

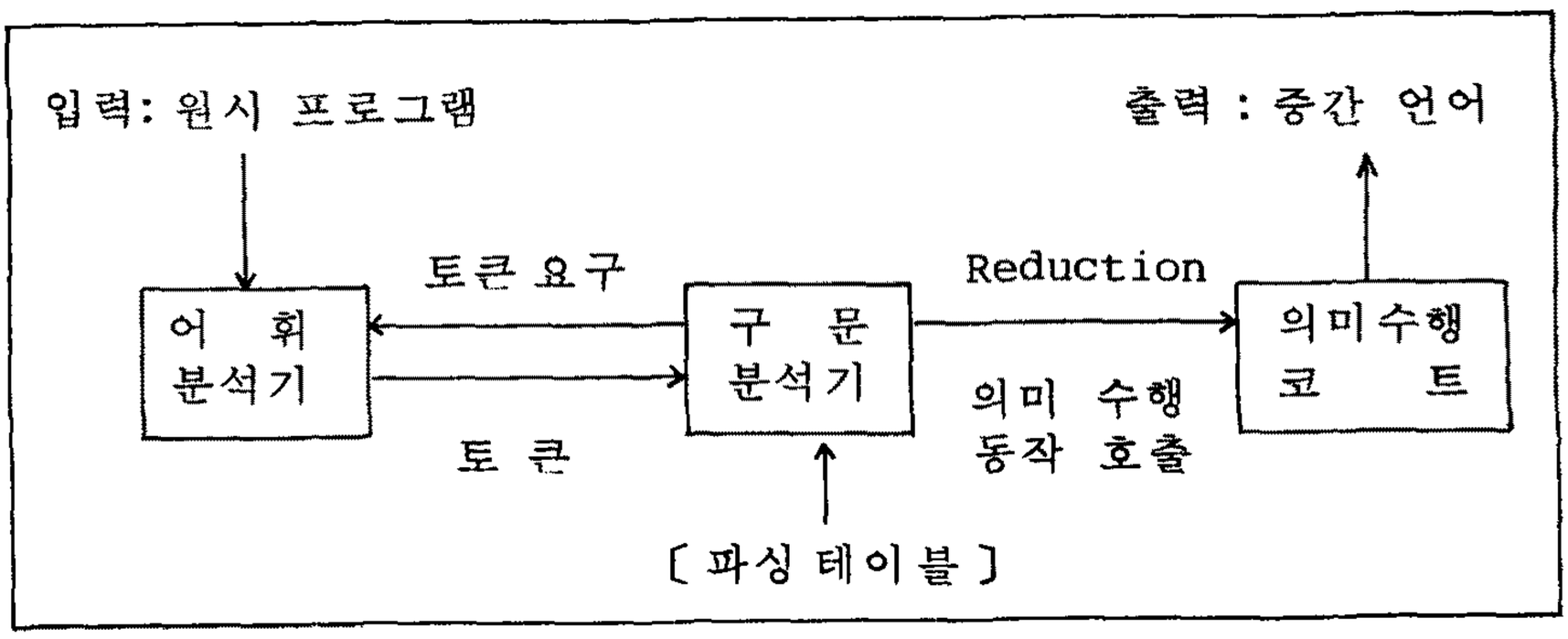
을 수행하기 전에 정수를 실수로 바꾸어 주는 작업이 필요하다. 예를 들어 A:=B+1과 같은 문장에서 만약 명칭 A와 B가 실수라고 하면 이에 대한 파스 트리는 [그림 2-19]와 같이 나타난다.



[그림 2-19]지정문 A:=B+1의 파스 트리

이러한 파스 트리에 대해 의미 분석 단계에서는 먼저 상수 1 값을 실수 값으로 바꾸어 주는 프로시쥬어를 호출하여 변환한 후 실수B에 더하여 실수A에 배정하도록 하는 의미 수행 코드를 생성한다. 또한 의미상으로 맞지 않는 것을 찾아 오류를 출력한다. 예를 들면, C:=1이라는 문장에 대해서 어휘 분석 단계나 구문 분석 단계에서는 명칭 C는 정당한 문자열이므로 오류가 발생되지 않으나 의미 분석 단계에서는 명칭 C에 1을 넣으라는 의미임을 알 수 있으므로 명칭 C가 심볼 테이블에 있는 지를 살펴본 후 만약 C가 존재하지 않으면 이전에 선언되지 않은 변수를 사용한 것으로 오류를 발생시킨다. 이처럼 문법에 기초한 파스 트리를 통해 컴파일하는 과정을 문법 지시적 변환(Syntax directed Translation)이라 하는데, 이 방법은 중간언어의 생성을 위해 Context-free문법의 각 생성 규칙들에 대한 의미있는 부프로그램이나 의미 수행 코드들로 구성된다. 이 부프로그램들은 파서에 의해 적당한 시간에 호출될 때 수행되어

중간 언어를 생성해내게 된다. 이러한 역할을 하는 변환기는 [그림 2-20]과 같은 구조를 갖는다.



[그림 2-20] 문법 지시적 변환기의 구성

(4) 중간 코드 생산 단계 (Intermediate Code Generation Phase)

이 단계에서는 파스 트리로 나온 결과를 해당 동작을 기술하는 중간 코드로 이루어진 프로그램으로 바꾼다. 중간 코드는 가상적인 기계의 기계어 명령어로 간주되는 것으로 대개 보통의 어셈블리 언어 형식과 비슷하지만 좀 더 일반적이고, 하드웨어에 독립적이다. 초창기의 컴파일러는 이론의 정립이 없는 상태에서 설계된 단일 패스 (One-pass) 컴파일러로서 원시 프로그램을 중간 코드 생산없이 직접 목적 코드로 번역하였다. 이 후 컴파일러 설계에 대한 이론 정립에 따라 점차 번역 단계가 세분되어 컴파일러를 여러 모듈들로 나누어서 설계할 수 있게 되었다. 이에 따라 각 모듈들을 연결시켜 주는 중간 코드들이 필요하게 되었으며, 특히 포터블 컴파일러와 같은 이식성 높은 소프트웨어를 만드는 데는 중간 코드 도입은 필수적이다. 중간 언어를 사용함으로써 얻어지는 장점은

① 컴파일러를 기능적으로 독립된 여러 모듈들로 구성할 수 있게

되며,

② 원시 프로그램의 이식성(portability)을 높일 수 있고,

③ 고급 원시 언어와 저급의 목적 언어간의 의미차를 이어주는 교량 역할을 해 줌으로써 컴파일 과정이 좀 더 쉽게 표현되고 효율적으로 처리될 수 있으며,

④ 중간 코드를 이용하여 최적화를 수행함으로써 기계와 독립적인 최적화가 가능해졌으며, 기계어 코드로 바뀐 상태에서 행하는 것보다 훨씬 효율적인 최적화를 수행할 수 있다.

반면에 결점으로는 중간·코드를 생성하여 다시 최종 목적 코드로 번역하는데 있어서 중간 언어를 거치지 않고 바로 목적 코드로 생산하는 것보다는 비효율적일 수 있다는 점이다. 그러나 이 문제는 중간 언어 단계에서 간단한 최적화를 통해 쉽게 극복될 수 있다.

상수의 정의 및 다른 여러 가지 정의나 선언의 경우는 의미 분석 단계에서 심볼 테이블의 정보를 갱신하거나 자료의 구조등을 정의하게 되지만, 수식을 비롯하여 실제로 실행을 하게 되는 문장들은 중간 코드를 얻는 것이 상례다. 이때의 중간 코드는 흔히 3주소 코드로 많이 생산되는데, 3주소중 2주소는 연산에 사용되며 나머지 주소는 연산 결과를 기억하게 된다. 따라서 각 연산문에는 하나의 연산자만이 허용되므로

$$A + B * C$$

와 같은 수식은 컴파일러의 임시 변수 T_1, T_2를 이용하여 두 개의 3주소문으로 나누어

$$T_1 = B * C,$$

$$T_2 = A + T_1$$

과 같이 3주소문으로 변환하여 중간 코드를 생산하게 된다.

(5) 코드 최적화 단계 (Code Optimization Phase)

이 단계에서는 전단계까지의 코드를 분석하고 같은 의미를 유지하면서 코드를 좀 더 효율적으로 만들어 코드 수행시 기억 공간이나 수행 시간을 절약하도록 해 주는 과정으로써 선택적인 단계로 생략될 수도 있다. 이 단계에 의해서

① 컴파일 시간 상수 연산 (Constant folding)

② 중복된 load, store 명령문 제거,

③ 식의 대수학적 간소화 (Algebraic simplification),

④ 좀 더 빠르게 계산 가능한 연산자로의 대치 (: Strength Reduction),

⑤ 불필요한 코드 블럭 삭제,

⑥ 루프 무변환 계산식의 고정,

⑦ 중복된 계산 제거

등의 효과를 얻을 수 있다.

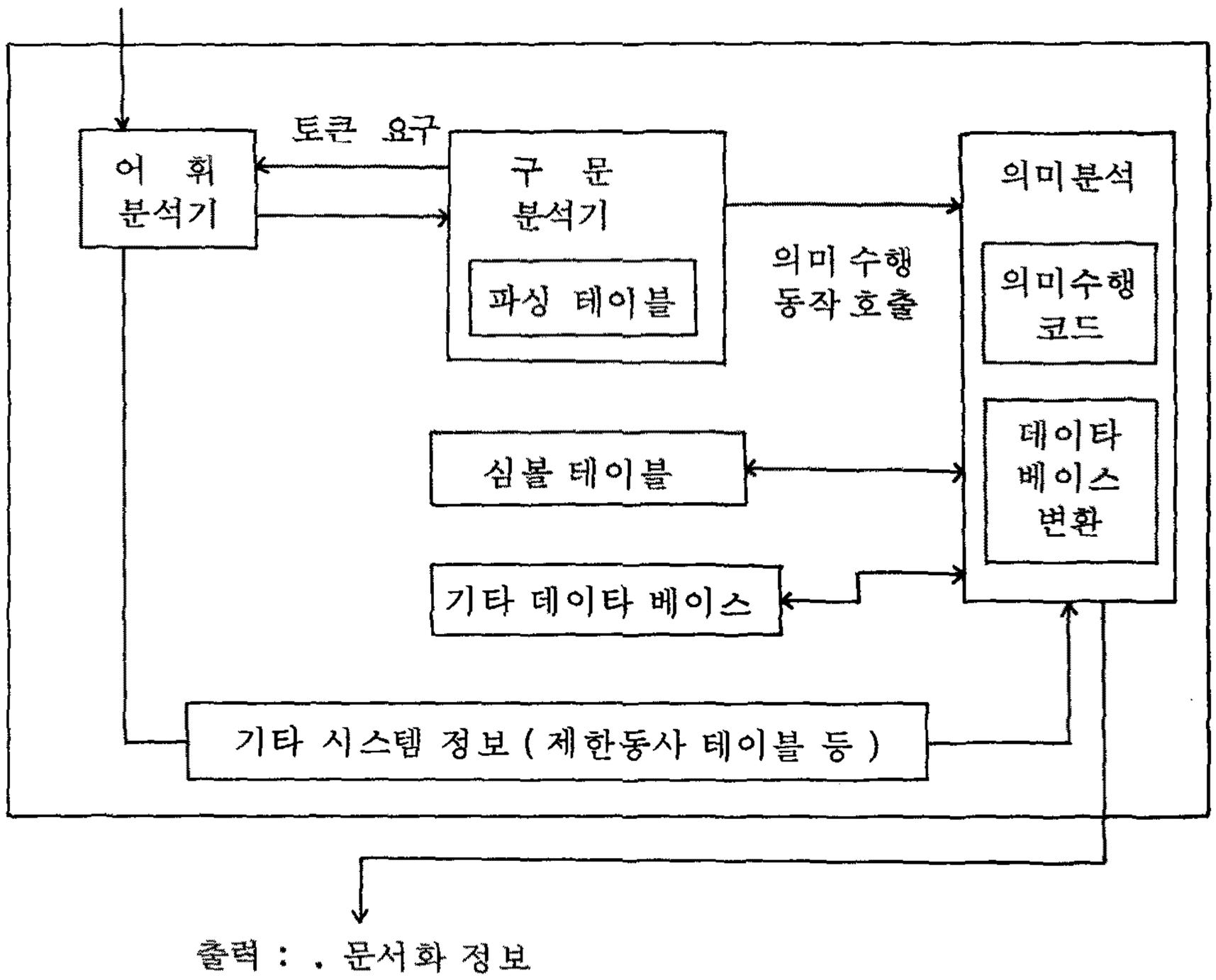
(6) 코드 생성 단계 (Code Generation Phase)

컴파일러의 최종적인 목적은 어셈블리어 또는 기계어로된 목적 프로그램을 만들어 내는데 있으므로 이 단계는 가장 기계에 종속된 단계로써 어셈블리 프로그램만 생성해 내는 경우도 있지만 대개는 재배치 가능한 목적 기계어 코드를 생성하여 시스템의 링커에 의해 라이브러리 루틴들과 연결되어 수행 가능한 프로그램 모듈로 만들어지게 된다.

나. 정적 분석기

앞 절에서 정의한 바와 같이 정적 분석은 실제로 프로그램을 수행하지 않고 프로그램의 구조적 측면, 즉 프로그램의 논리상의 전반적 특성을 분석해 내는 작업으로써, 이러한 정적 분석을 통하여 실제 프로그램 실행전에 오류를 유발시킬 코드에 대한 정보의 발견, 수형 검사, 변수들의 비정상적인 사용, 각 모듈간의 상호접속 관계, 자료흐름 정보, 프로그램의 구조 분석 및 복잡도 측정 등의 중요한 정보를 얻음으로써 그 프로그램에 대한 테스트 계획수립, 테스트 결과에 대한 신뢰도 증가 등의 효과를 얻을 수 있으며 개발된 프로그램의 유지 보수 단계에서 중요한 자료를 제공해 주게 된다. 이와 같은 작업은 물론 수작업으로도 가능하지만 수작업으로 인한 번거로움을 감소시키고, 방대한 크기의 프로그램에 대한 수작업 분석의 한계점을 해결하며 프로그래머의 생산성을 향상시키기 위해 정적 분석의 자동화가 이루어지게 되었다. 자동화 도구로서의 정적 분석기는 현재 다수 존재하나 이들이 어떤 통합된 시스템 형태로서 존재하고 있지는 않으므로 정적 분석을 위한 시스템들의 구조는 아직 정형화되지 못한 상태에 있다. 그러나 컴파일러나 정적 분석기 둘다 원시 프로그램을 입력으로 받아서 그 프로그램을 분석하는 단계까지는 동일하다는 입장에서 보면 정형화된 컴파일러의 여러 단계들 중 전반부의 어휘 분석 단계와 구문 분석 단계까지는 둘 다 동일한 형태를 취한다고 볼 수 있으며, 문법의 각 생성에 대해 컴파일러의 의미 분석 단계에서는 명칭의 수형 검사 및 기억장소 할당과 중간 코드를 생성하는 등의 동작을 하는 반면, 정적 분석기의 의미 분석 단계에서는 각 생성들로부터 정적 분석의 결과 정보를 검

입력 : 원시 프로그램, 규칙 테이블



출력 : . 문서화 정보

프로그램 통계 정보
프로그램 논리 구조
시스템 복잡도 측정
모듈 상호접속 정보 분석
변수의 참조상황 분석
자료 흐름 분석
코드 감사 정보
입출력 명세 정보

. 오류정보

[그림 2-21] 정적 분석기의 전반적 구조

출하기 위한 동작을 하므로 의미 분석 단계로부터 서로 다른 형태를 취하게 된다. 따라서, 정적 분석기는 [그림 2-21]와 같은 전반적인 구조를 갖는다. [그림 2-21]는 앞에서 설명한 컴파일러의 의미 분석 단계의 문법 지시적 변환기와 거의 동일한 구조를 하고 있음을 알 수 있다. 그 이유는 정적 분석기에서도 정적 분석 정보의 검출을 위해 문법의 각 생성 규칙에 따라 그 규칙이 구문 분석기에 의해 Reduction될 때 해당 의미 수행 행동을 호출하여 수행하는 문법 지시적 변환 방법을 사용하고 있기 때문이다. 그리고 단지 그들 각각의 출력 형태만 다르게 생성될 뿐이다.

여기서 본 연구 개발의 결과인 COBOL의 정적 분석기에 촛점을 맞추어 정적 분석의 각 단계에서 하는 작업들을 살펴보면 다음과 같이 이루어진다.

(1) 어휘 분석 단계 및 구문 분석 단계

정적 분석기의 어휘 분석 단계와 구문 분석 단계에서 하는 일은 근본적으로는 앞에서 기술한 컴파일러의 어휘 분석 단계 및 구문 분석 단계와 동일한 일을 한다. 즉 어휘 분석 단계에서 입력 원시 프로그램을 받아들여 이들을 토큰 단위로 묶어내어 구문 분석 단계로 넘겨주면 구문 분석 단계에서는 파싱 테이블을 이용하여 Shift 또는 Reduce를 함으로써 구문을 분석해 내는 작업을 한다. 이들은 모두 컴파일러의 단계들에서의 방법을 그대로 적용하여 구현된다. 그리고 컴파일러와 약간 다른 점은 정적 분석을 위한 초기화 과정을 거치게 된다는 것이다. 즉, 원시 프로그램이 입력 되기 전에 정적분석을 위한 테이블 입력 및 기억 장소들을 할당해 주는 역할을 한다. 예를 들면, YACC의 출력인 구문 분석 테이블(파싱 테이블)을

초기화 단계에서 읽어들여 구문 분석 단계에서 쓰일 수 있도록 준비를 해준다. 본 연구 개발에서는 COBOL의 구문 분석을 위한 파싱 테이블을 파서 자동 생성 도구인 YACC로부터 구하여 이 테이블을 초기화 단계에서 읽어 들여 테이블 형태로 저장하도록 하였다.

(2) 의미 분석 단계

컴파일러가 의미 분석 단계에서 목적 프로그램 생성을 위한 중간 코드들을 생성해내는 것과는 약간 다르게 정적 분석기의 의미 분석 단계에서는 원시 프로그램 언어 문법의 각 생성들에 대해서 정적 분석 정보를 검출해 내게 된다. 즉 입력 문이 문법의 하나의 규칙에 의해 Reduction될 때 그 규칙에 대한 해당 의미 수행 동작을 하는 프로시쥬어를 호출함으로써 정적 분석 정보를 검출한다. 그러면 여기서 다음과 같은 문법 P에 대하여 문법 지시적 변환 방법을 이용할 경우의 컴파일의 의미 분석 단계에서의 작업과 정적 분석기의 의미 분석에서의 작업의 차이를 비교해 보자. 문법 P는

P : S → E$

E → E + E

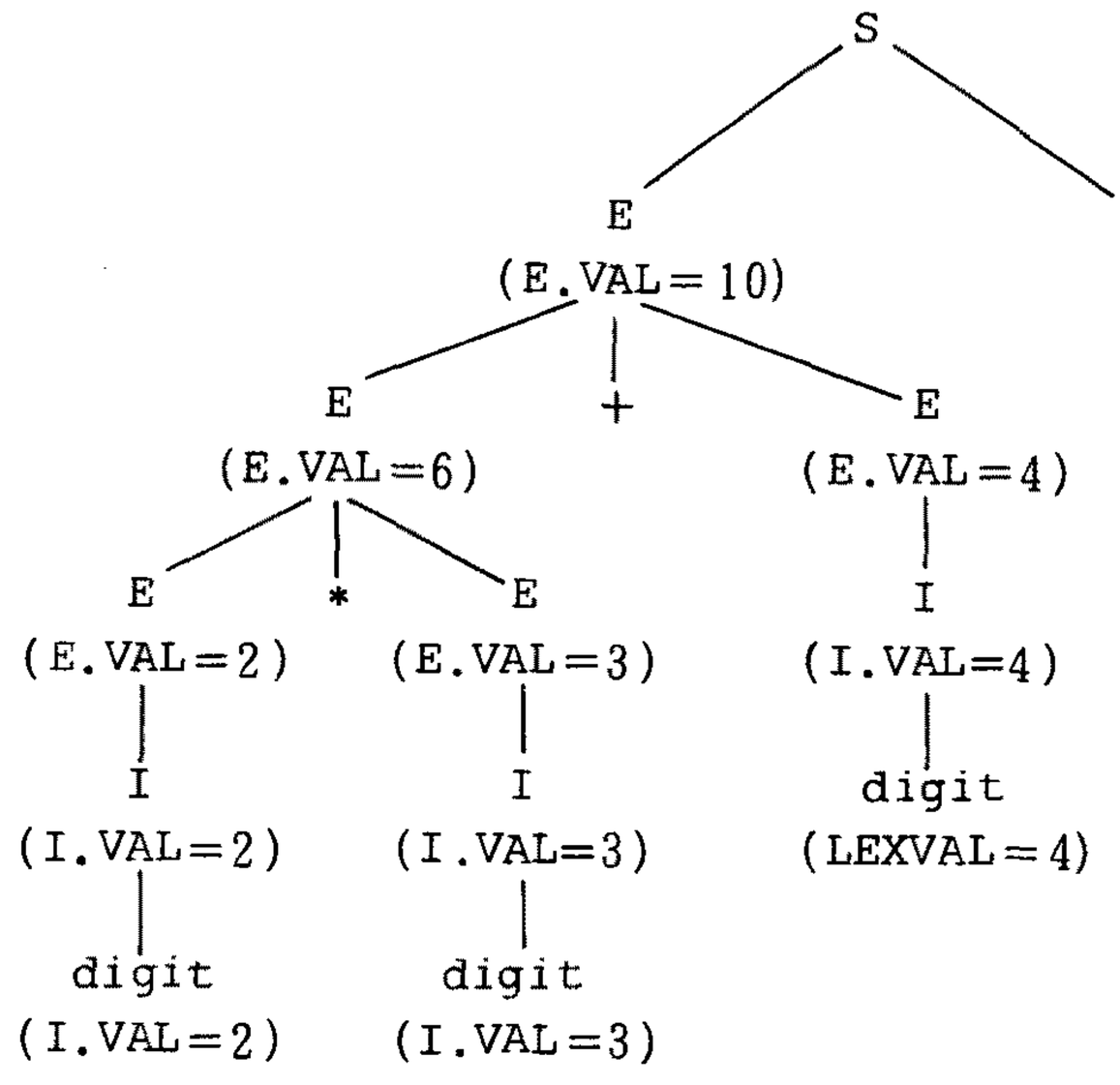
E → E * E

E → I

I → digit

와 같으며 digit는 0부터 9까지의 숫자다. 이러한 문법의 각 생성 규칙들에 대해 2 * 3 + 4$의 파스 트리는 [그림 2-22]와 같고, 컴파일러의 의미 수행 단계의 각 생성에 대한 의미 수행 코드는 [그림 2-23]과 같이 이루어진다. 여기서 각 Nonterminal의 값을

Nonterminal 이름. VAL 형태로 나타내며, Terminal digit의 변환은 LEXVAL로 나타내었다.



[그림 2-22] 2*3+4$의 파스 트리

생 성 규 칙	의 미 수 행 코 드
$S \rightarrow E\$$	{ Print E.VAL}
$E \rightarrow E^1 + E^2$	{ E.VAL:= E^1 .VAL + E^2 . VAL }
$E \rightarrow E^1 * E^2$	{ E.VAL:= E^1 .VAL * E^2 . VAL }
$E \rightarrow I$	{ E.VAL:= I.VAL }
$I \rightarrow$ digit	{ I.VAL:= LEXVAL}

[그림 2-23] 컴파일러의 의미 수행 코드

컴파일러의 의미 수행 단계에서는 [그림 2-26]과 같은 의미 수행 동작을 수행하는데, 이것은 원시 프로그램의 실행면에 관한 코드인 반면, 정적 분석기의 의미 수행 단계에서는 원시 프로그램의 전반적인 논리적 구조의 분석에 주안점을 두므로 위와 같은 생성 규칙들에 대해서 다른 의미 수행 동작을 실행하게 된다. 이러한 문법 지시적 변환 방법을 구현하는데 있어서 파서는 각 생성 규칙이 Reduction될 때 해당 의미 수행 코드를 호출하여 수행되게 되는데, 위의 생성들에 대한 정적 분석기의 의미 수행 동작을 위한 프로그램들은 [그림 2-24]과 같이 이루어진다.

생 성 규 칙	정적 분석기의 규칙 단위 수행 작업
S → E $	정적 분석 정보 출력
E → E^1 + E^2	E^1 변수와 E^2 변수가 참조되었음을 변수테이블에 기억시킴 {Update Variable Usage}
E → E^1 * E^2	E^1 변수와 E^2 변수가 참조되었음을 변수테이블에 기억시킴 {Update Variable Usage}
E → I	E.VAL := I.VAL {StackCopy}
I → digit	VAL[TOP] :=LEXVAL

[그림 2-24] 정적 분석기의 의미 분석 동작

(3) 출 력

위와 같이 정적 분석된 정보들은 TEXT형태로 바로 출력될 수도

있으나 통합 환경하에서는 사용자 접속환경의 지원을 받아 사용자들이 분석된 결과를 알아보기가 쉽고 Reporting에 용이한 여러가지 형태들로 출력된다.

(4) 설계시 고려사합

실제로 정적 분석정보를 추출해 내는 단계인 의미 분석 단계에서는 구문 분석 단계와의 연결관계가 가장 중요하게 고려되어야 할 사항이다. 의미 수행코드의 작성은 파서에 의해 생성 규칙이 Reduction 될 때 파스 트리를 구성한 후 트리를 운행하면서 정보를 추출해 내는 방법과 문법지시적 변환 방법에 의해 각 생성 규칙의 Reduce 시에 의미수행 코드를 호출하여 실행하며 정보를 추출하는 방법이 있다. 각 방법에 따른 규칙 분석의 정도 및 의미 수행 코드와 구문 분석기간의 Information Hiding을 고려하여 연결 구조를 정의한다.

파스트리를 이용하는 방법은 구현에 있어서 그 조작방법이 복잡할 뿐만 아니라 실제 의미있는 행동을 하기 전까지 파스트리를 유지하기 위한 기억장치 관리 문제등이 존재한다.

파스트리를 HEAP이라는 동적 기억장소로 유지하는 대신 스택(Stack)을 이용하여 파스트리의 형태를 기억하고 있다가 정적 분석의 행위를 하도록 변형하거나, 이보다 좀더 간단하고 구조화된 설계를 위해 문법 지시적 변환 방법을 도입하여 문법의 규칙을 세분화하여 정적분석을 단위 기능별 처리를 하도록 구현하는 것이 바람직하다.

그러면, 여기서 실제 이러한 설계 과정시의 고려사항들의 구현 형태를 차례로 설명하여 비교분석해 보자.

㈏ 파스트리를 이용한 정적 분석 행위 설계 방법론

① 트리 운행을 이용한 정적 분석

규칙(Rule)은 어떤 계층 구조를 갖는다. 우리가 얻고자하는 정적 분석 정보는 개개의 최하의 규칙에서 얻어지는 것이 아니며 일정 단위의 규칙 부분 집합이 모여 정적 분석에서 의미있는 정보가 얻어질 수 있는 단위가 되면 비로소 실제적 정적 분석 행위가 이뤄지게 된다. 실제적 정적 분석 행위를 야기시킬 만한 정도의 규칙을 편의상 Head Rule로 지칭하기로 하면, 정적 분석기의 정적 분석 행위는 이러한 Head Rule을 찾아 그에 달린 Sub Rule들을 운행하면서 정보를 추출하여 가는데, Sub Rule에서 모든 정보가 추출되면 Head Rule은 이 정보를 조직화하여 정적 분석 데이타베이스에 담아주는 것으로써 정적 분석의 한 행위가 종료된다. 파스트리를 운행하며 분석을 하는 이 방법은 이처럼 Head Rule과 정적 분석 데이타베이스가 일대일로 관계를 가지며 정적 분석행동(또는 의미수행 행동)을 하여 우리가 원하는 정보를 얻어낸다.

㉮ 알고리즘 설계

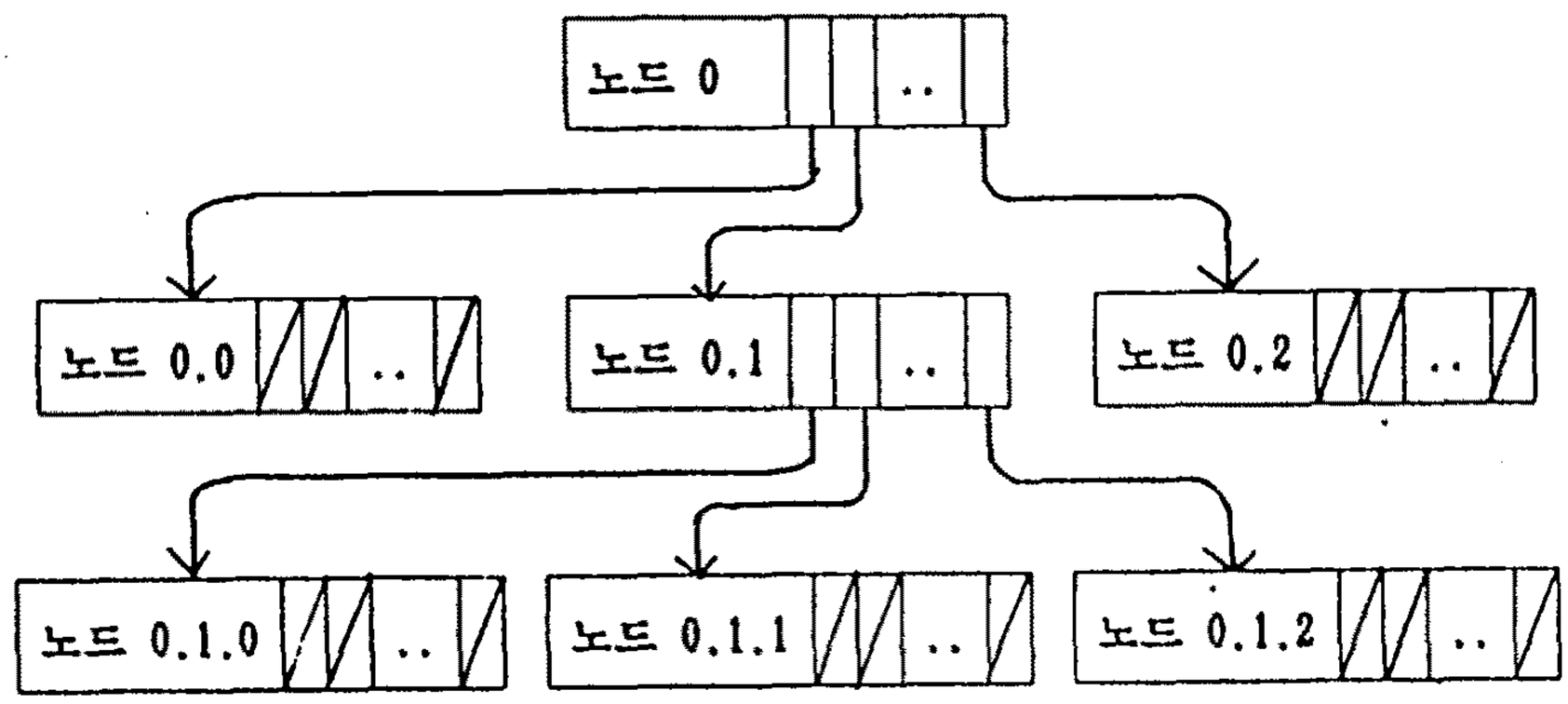
이러한 방법의 알고리즘 설계를 위해서는 먼저 BNF로 표시된 문법을 트리구조로 표시한 후 트리로 표시된 문법 규칙으로부터 정적 분석 데이타베이스와 일대일 관계를 갖는 Head Rule을 찾아내고 그의 행동을 프로시쥬어로 작성하여 이를 Head Procedure라 한다.

Head Procedure는 필요한 데이타를 얻기위해 하위 프로시쥬어(하위규칙)를 호출하도록 작성한다. 호출된 하위 프로시쥬어는 자신의 하위 프로시쥬어를 계속하여 호출하며 자신이 얻은 정보를 상위 프로시쥬어(상위 규칙)로 넘겨주게 된다. 상위 규칙이 여러 개의 하위

규칙을 선택적으로 가질 경우, 하위 규칙을 구성하는 노드를 검사하여 새로 호출할 하위 규칙을 결정한다. 상위 규칙의 하위 규칙에 대한 계속적인 호출은 호출된 규칙이 Terminal 규칙일 때까지 계속된다. 이런 과정을 계속하여 정보가 Head Procedure(Head Rule)로 모여지면, 모여진 정보를 해당 데이타베이스에 맞게 조직화하여 저장하는 프로시쥬어를 작성함으로써 정적 분석 알고리즘이 설계된다.

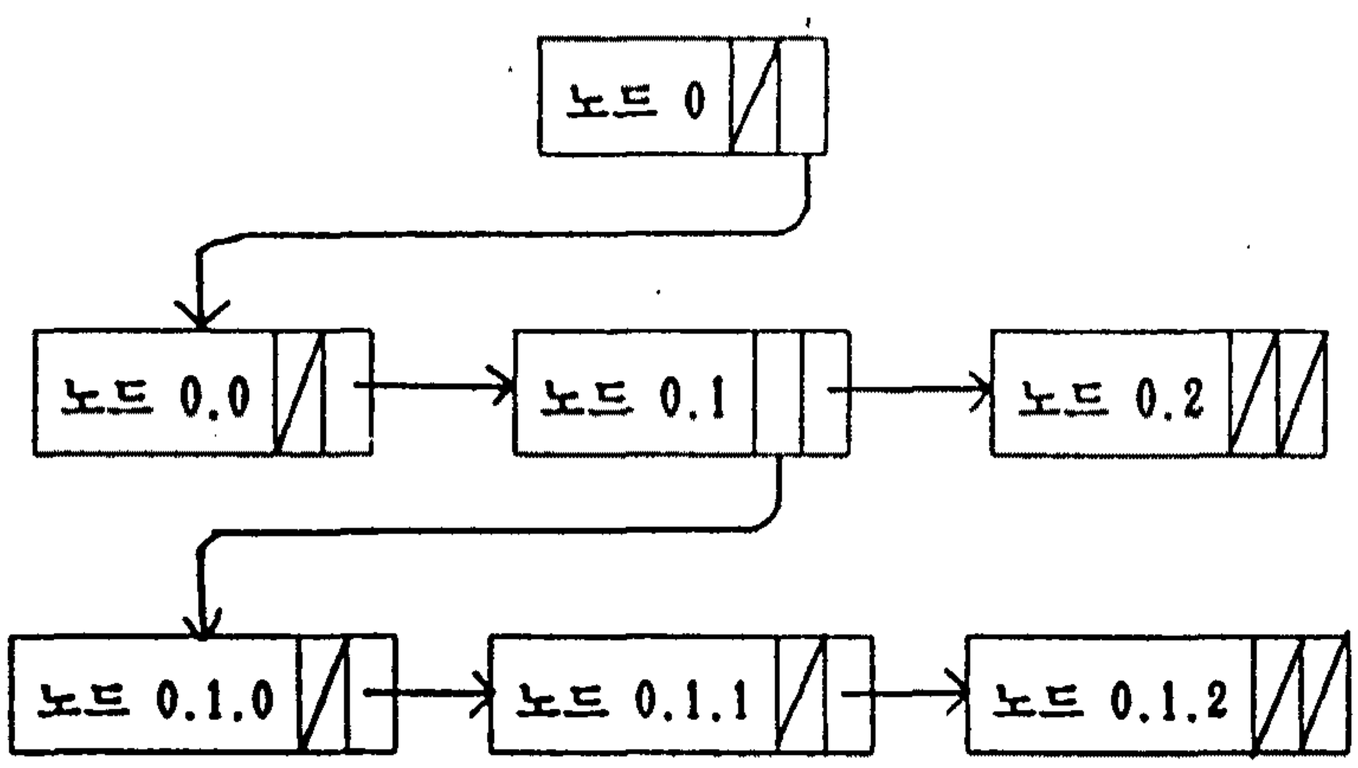
㈏ 파스트리 구현을 위한 List Pointer의 구조 설계

파스트리 구현에서 [그림 2-25]과 같이 각 노드가 모든 자식 노드에 대한 Pointer를 가지고 있으려면 각 언어의 문법표현(BNF)마다 최대 자식 노드의 갯수가 다르므로 일반적인 표현에는 한계가 있으며 최대 자식 노드 수를 추측하여 구현한다 해도 사용되지 않는 부분이 많아져 기억 공간의 낭비를 초래하므로 이와 같은 문제점을 해결하기 위해 각 노드는 두가지의 Pointer 부분을 가지도록 설계하였는데, 하위 노드에 대한 Pointer는 자신이 거느리는 자식 노드



[그림 2-25] 파스 트리의 구현 형태

중 가장 왼쪽 것에 대한 링크를 갖는다. 또, 동위 노드들(Siblings)에 대한 Pointer는 하나의 부모 노드에 딸린 자식 노드들을 왼쪽에서 오른쪽으로 연결하여 주는 Pointer로서 예를 들면 [그림 2-26]과 같이 표현되며, 이런 자료 구조의 특징으로서는 모든 종류의 트리를 표현할 수 있는 자료 구조이며 트리의 깊이가 깊어질수록 검색 효율은 증가하고 너비가 넓어질수록 검색 효율은 감소하지만 기억 공간의 절약이 그만큼 증가한다는 특징이 있다.



[그림 2-26] List Pointer를 이용한 파스트리의 개선된 구현

㉰ 구현된 의미 분석 부분 트리의 예

다음과 같은 예제 문법에 대하여 파스 트리를 이용한 각 노드에서의 정적 분석 과정을 아래에 나타내었다.

예제문법규칙 : VARIABLE_DES→LEVEL_NUMBER ident DEF_CLAUSES tdot

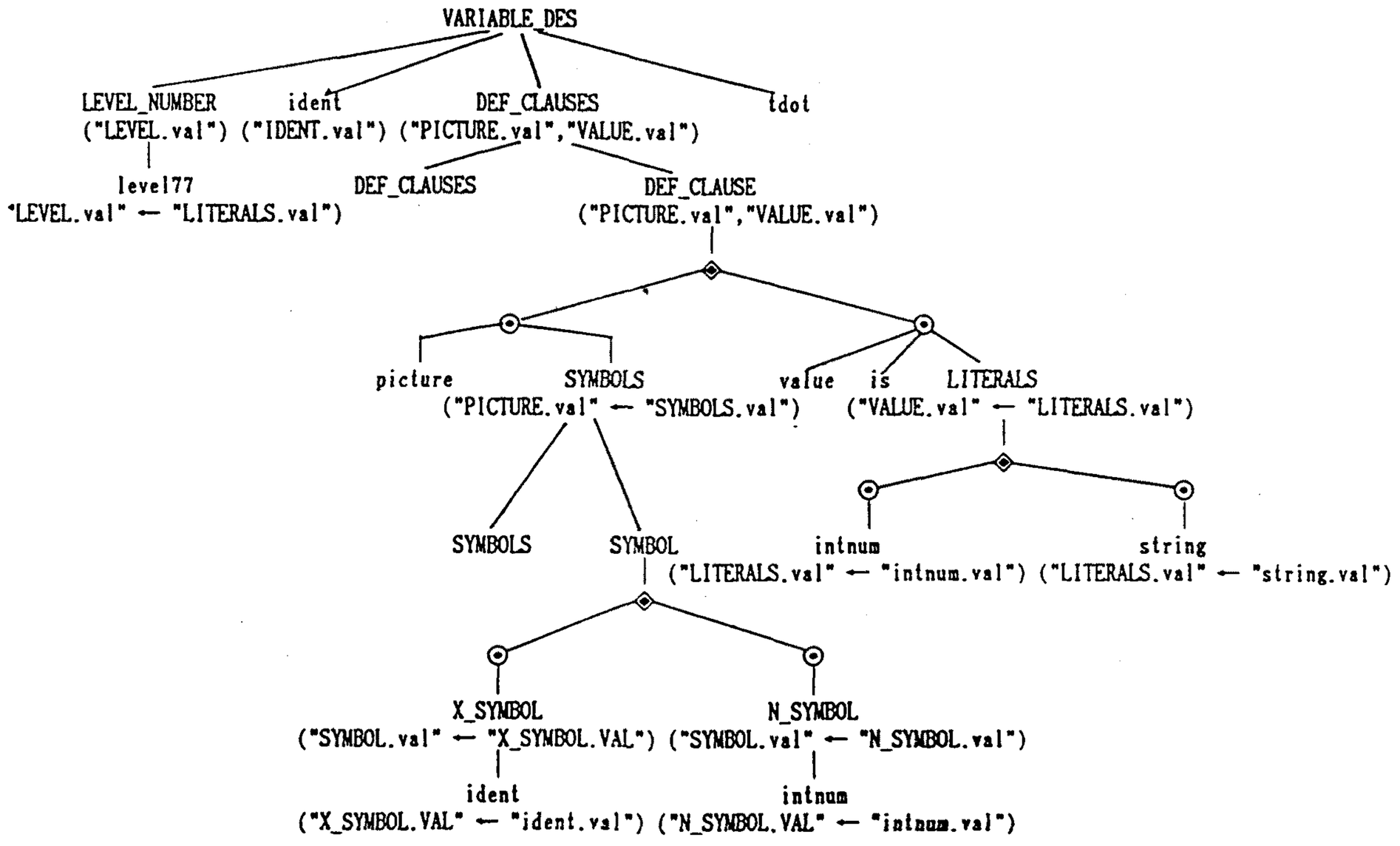
LEVEL_NUMBER→leve l77

DEF_CLAUSES→DEF_CLAUSE | DEF_CLAUSES DEF_CLAUSE

DEF_CLAUSE→picture SYMBOLS | value is LITERALS

SYMBOLS→SYMBOL | SYMBOLS SYMBOL

⊙ 파스 트리에서의 정적 분석 과정의 예



SYMBOL→X_SYMBOL | N_SYMBOL

X_SYMBOL→ident

N_SYMBOL→intnum

LITERALS→intnum | string

② 스택을 이용한 정적 분석

위의 트리 운행을 통한 정적 분석방법이 매우 복잡하고, 조작 및 구현이 어려우므로 이를 좀 더 쉽게 해결하기 위해 파서에 의해 Reduction될 때 파스트리를 List Pointer의 트리 구조를 만드는 대신 이러한 정보를 스택을 이용하여 구현한다. 즉 파싱의 과정중에 부분적으로 형성된 트리의 루트 노드 번호들을 Forest 스택에 기억해 두었다가 실제적인 정적 분석 행위시의 생성의 자식 노드들이 되도록 한다. 이 방법에 사용되는 스택들의 구조는 다음과 같다.

㉮ 스택구조

- STATE STACK
  - Shift/Reduce 행위를 할 때의 Action State를 저장
  - State Stk:array[1..Max State Stack Size] of integer
- FOREST STACK
  - Parsing 과정중에 부분적으로 완성된 트리의 노드 번호들을 기억했다가 Reduce 행위시에 그 Production의 자식 노드들이 되도록 한다.
  - Forest Stk:array[1..Max Forest Stack Size] of integer

- STACK(Parse Trees Stack)
  - 생성되는 트리의 노드 정보를 차례로 기억시킨 배열구조
  - 노드는 토큰 노드이거나 생성 노드이다.
  - 생성 노드의 자식 노드들은 이미 Parse Trees Stack에 생성되어 있으며, 그들의 위치 번호는 Forest Stack에 기억되어 있으므로 그 생성의 길이만큼 POP하여 자식 노드들로 기억하게 된다.
  - 스택의 자료 구조

```
● Stack : array    [1..Max_Size]    of
            record
              case  SYMBOLTYPE : integer  of

                  0 : {* Production Node :
                         . 생성 번호, 생성 길이,
                           자식 노드들의 index *}
                     (PRODUCTION_NUMBER,
                      PRODUCTION_LENGTH : integer;
                      CHILDREN : array [1..10] OF integer);

                  1 : {* Token Node  :
                         . 명칭인 경우 : (1,0,명칭테이블 index);
                         . 상수인 경우 : (1,1,상수테이블 index);
                         . 기 타       : (1,토큰 코드) *}
                     (TOKEN_PART : Tokentype);

            end; {* record *}
```

㉯ 스택을 이용한 정적 분석 알고리즘

파스 트리를 동적 기억장소에 유지함으로써 발생되는 기억장치 조작의 문제점 및 알고리즘의 복잡성을 해결하기 위해 트리 운행 부분을 스택을 이용하여 구현할 수 있다. 즉 파싱의 과정중에 부분적으로 형성된 트리의 루트 노드 번호들을 Forest 스택에 기억해 두

었다가 실제적인 정적 분석 행위시의 생성의 자식 노드들이 되도록 한다. 여러 규칙들에 대해 이와 같은 방법으로 스택에 트리 형태의 정보를 계속 쌓아두었다가 정적 분석 정보를 실제로 추출해낼 수 있는 규칙이 Reduction될 때 스택으로부터 하나씩 POP하여 그 때의 스택 노드가 생성 노드이면 그 생성의 자식 노드들로부터 정보를 추출해 내며 Token 노드이면 그 Token의 정보를 전달해 주게 된다. 필요한 정적 분석 정보를 이와 같은 방법으로 추출해 내는 과정은 위의 List Pointer를 이용하는 과정과 동일하며, 그 구현 형태 및 조작 방식에 있어서는 그보다 훨씬 간단해 진다는 잇점이 있다

㉰ 구현된 의미 분석 부분 트리의 예(Stack을 활용한 경우) 다음과 같은 문법 규칙에 대해 COBOL의 입력 스트링

```
77 COSTAR picture 999.
```

에 대한 파싱과정에서의 스택의 상태는 [그림 2-27]과 같으며 파스트리에서의 정적 분석과정은 [그림 2-28]와 같다. 그림에서 문법 규칙에서의 번호는 해당 생성번호를 나타내고, 토큰들의 번호는 다음과 같다고 가정한다.

토 큰	코 드
77 LEVEL	1
picture	2
intnum	3
tdot	4

문법 R : (1) VARIABLE_DES→LEVEL_NUMBER ident DEF_CLAUSE tdot

(2) LEVEL_NUMBER→leve177

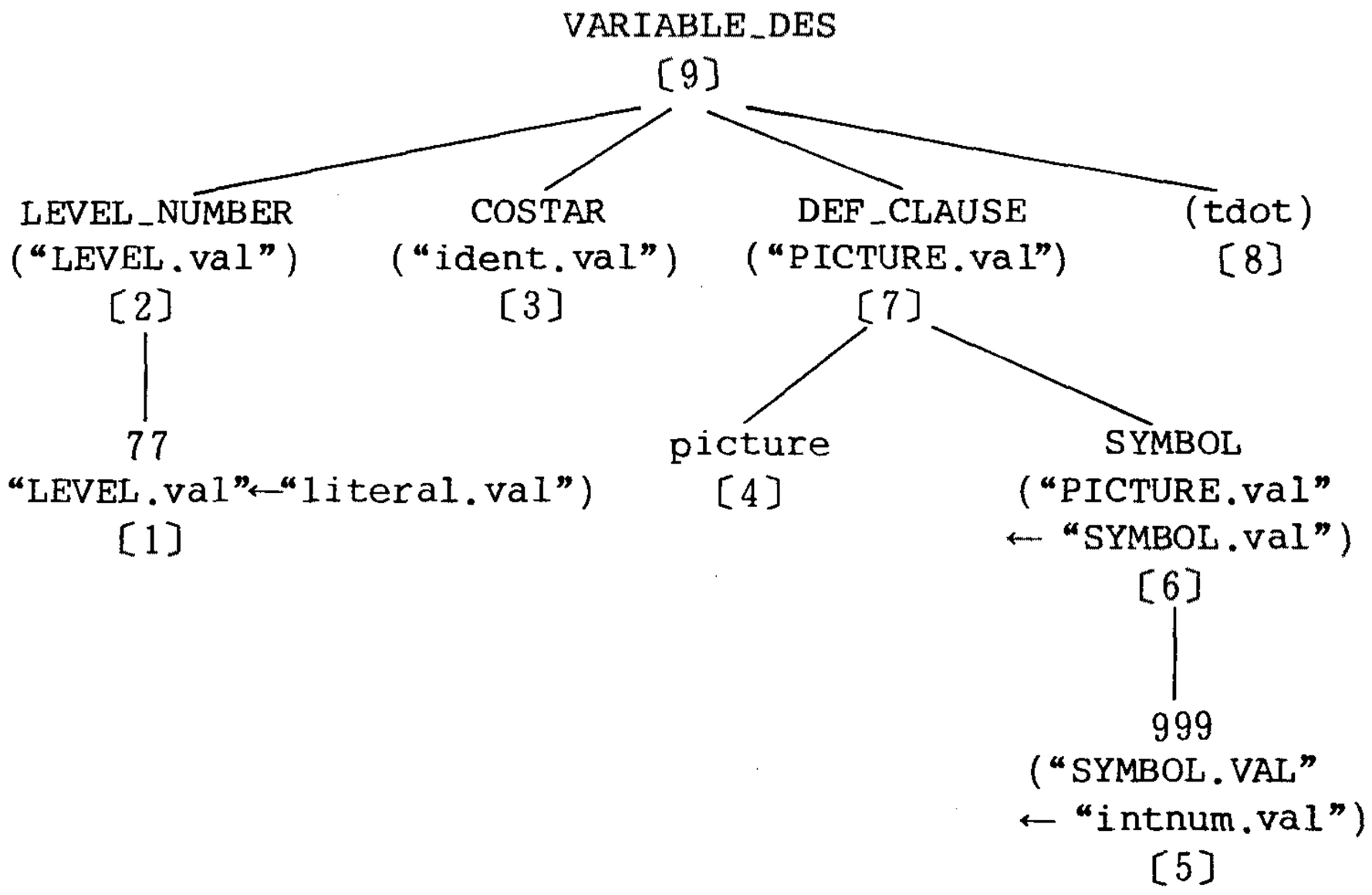
(3) DEF_CLAUSE→picture SYMBOL

(4) SYMBOL→intnum

INDEX	Tree_Node
1	ParseStack[1]=(1,1)
2	ParseStack[2]=(0,2,1)(1)
3	ParseStack[3]=(1,0,1)
4	ParseStack[4]=(1,2)
5	ParseStack[5]=(1,3)
6	ParseStack[6]=(0,4,1)(5)
7	ParseStack[7]=(0,3,2)(4,6)
8	ParseStack[8]=(1,4)
9	ParseStack[9]=(0,1,4)(2,3,7,8)

[그림 2-27] 파싱 과정의 스택 상태

파싱 과정의 스택에서 처음에 Level 값을 나타내는 77이 입력으로 들어올 때 토큰 77Level의 토큰값은 1이므로 스택에는 토큰 노드를 나타내는 1값과 그 토큰의 토큰값 1이 쌍으로 기억되므로 스택[1]에는 (1,1)이 기억된다. 다음에는 명칭 COSTAR가 입력으로 들어올 때 이전에 스택에 기억된 값 77이 문법 규칙 (2)에 의해 Reduction 되므로 스택[2]의 값이 생성 노드임을 나타내는 0값과 생성 번호 2, 생성 규칙 (2)의 길이가 1이라는 값들의 쌍으로 기억되며, 이 생성의 자식 노드가 스택의 [1]의 위치에 있음이 자식 노드를 기억해 두는 부분에 저장된다. 그리고 입력된 명칭에 대해서는 그 명칭을 심볼테이블에 기억해 두고 그 위치를 Return하여 스택에 저장하게 되므로 스택[3]의 값은 (1,0,1)이 된다. 이와 같은 파싱 과정을 계속하여 생성된 문법 R의 파스트리는 [그림2-28]와



[그림 2-28] 파싱 스택을 이용한 정적분석

같이 가상적으로 생성되며, 이의 형태의 파싱 스택에서 유지되는 상태가 [그림 2-27]에 나타나 있다.

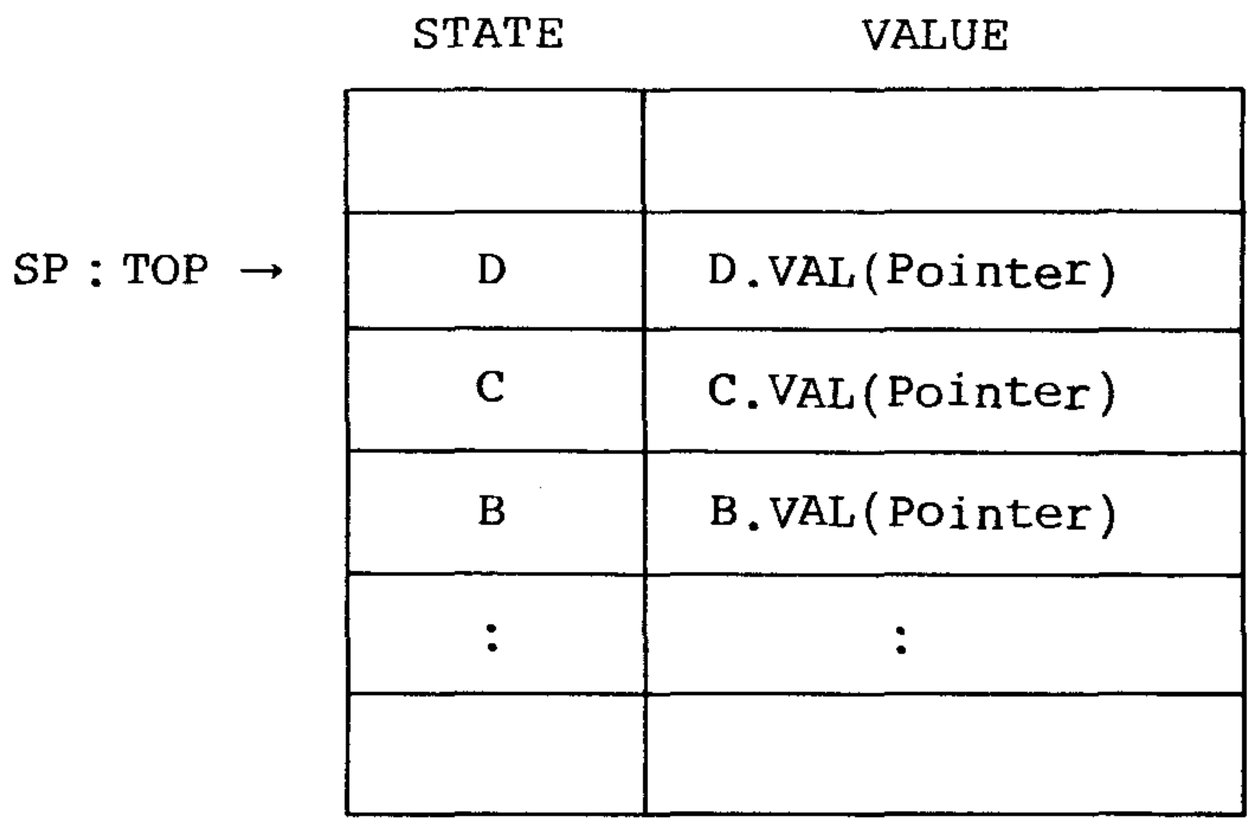
(나) 문법 지시적 변환을 이용한 정적 분석 행위 설계 방법론

문법 지시적 변환의 장점은 컴파일러 설계자가 원시 언어의 구문 구조에 따라 직접 중간 언어 생성을 위한 표현을 할 수 있다는것이다. 따라서 앞에서 설명한 파스 트리를 이용한 컴파일러의 의미분석 및 정적 분석 방법론보다 구현하기가 훨씬 용이할 뿐만 아니라 트리를 운행하면서 정보를 추출해내지 않으므로 효율면에서도 많은 잇점이 있기 때문에 요즈음의 변환기 및 분석기 설계에 있어서 널리 적용되고 있다. 또한 정적 분석 정보를 추출해 내는데 있어서도 여러 정적 분석기 설계자가 각 문법 규칙별로 추출 가능한 정보를 미리 예측하여 규칙별 단위 행동을 코딩함으로써 쉽게 통합하여 구

현할 수 있으며, 문법의 규칙이 바뀌었을 때 프로그램의 수정이 용이하다는 잇점이 있으므로 본 연구개발에 있어서도 최종적으로 이 방법을 채택하였다. 따라서 본 연구개발에서 개발된 COBOL의 정적 분석기를 좀 더 확장된 규칙들을 갖는 COBOL 프로그램의 분석을 위해서는 새로운 생성 규칙들과 의미 수행 코드들이 기존의 정적 분석 코드를 크게 바꾸지 않고 추가될 수 있다.

본 연구개발에서 구현한 정적 분석의 의미 수행 단계는 일반적인 컴파일러의 문법 지시적 변환방법을 그대로 이용하고 있으며, 한가지 특기 할 만한 사항으로는 정적 분석을 위해 다음과 같이 스택을 이용하고 있다. 즉, 문법 심볼에 대한 파싱 스택 원소에 특수한 부분을 추가하여 정적 분석을 할 때 각 문법 규칙에 대응되는 정보의 값이 그 장소에 임시로 저장되어 다음 규칙으로 전달되어 연쇄적인 정보를 얻을 수 있도록 하였다. 이런 스택을 이용하여 정적 분석과정이 어떻게 이루어지는 지를 간단히 살펴보자. 아래의 〔그림2-29〕과 같이 스택은 STATE와 VALUE(Node Pointer)의 쌍으로 이루어지는 배열로 구성되어 있으며 각 STATE 원소는 LR 파싱 테이블에서의 상태를 나타내며, 만약 i번째 STATE 심볼이 E라 할 때 VALUE〔i〕는 심볼 E의 심볼 테이블이나 상수 테이블의 위치를 나타내고, Stack Pointer(:SP) TOP은 스택의 현재 위치를 나타내는 포인터이다.

파서에 의해 해당 규칙이 Reduction될 때 의미 수행 코드들이 수행되므로 생성 규칙 A→B C D에 대하여 B C D가 A로 Reduce 될 때, D의 변환된 값은 스택의 VAL〔TOP〕에 있고, C와 B의 변환된 값은 VAL〔TOP-1〕과 VAL〔TOP-2〕에 각각 존재한다. 생성 A→B



[그림 2-29]파싱 스택의 구조

CD가 Reduction될 때는 TOP이 2만큼 감소되고 VAL[TOP]은 이 생성에서 얻어진 정적 분석 정보를 임시로 보관 하였다가 다음 생성으로 전달되어, 만약 X → STA와 같은 생성이 존재하여 Reduction될 때 이 정보는 중요하게 활용된다. 즉 만약 심볼 C의 심볼 테이블에서의 위치를 생성 X → STA에서 알아야 한다면, 이 생성에서는 바로 C의 위치를 알 수 없으므로 A → BCD가 Reduction 될 때 C의 심볼 테이블에서의 위치를 A에 넘겨주어 다음 단계의 생성 X → STA에서는 C의 위치가 저장된 A의 Pointer주소를 참조하여 해당 정적 분석 정보를 저장하게 된다. 복잡한 규칙이나 순환 구조를 갖는 규칙들에서는 이런 방법을 쉽게 적용하기가 용이하지 않으므로, 모든 규칙들에 대해 이런 방법을 적절히 활용할 수있도록 규칙 자체를 좀 더 미세하게 세분화하여 되도록 규칙 단위행동이 간단 명료해지게 한다.

예로써 문법 R에 대한 문법 지시적 변환을 이용한 각 생성별 정적 분석용 의미 수행 코팅의 예를 아래에 나타내었다.

문법 R : (1) VARIABLE_DES → LEVEL_NUMBER ident DEF_CLAUSE tdot

(2) LEVEL_NUMBER → level77

(3) DEF_CLAUSE → picture SYMBOL

(4) SYMBOL → intnum

```
Procedure Define_Level_Variable(Val : SymbolStackPointer);
Procedure Update_Level_Number(Val : SymbolStackPointer);
Procedure Picture_Symbol_Analysis(Val : SymbolStackPointer);
Procedure Insert_In_Constant_Table
          _and_Return_the_Value(Val : SymbolStackPointer);

 CASE                          Rule_No                          OF

    1 : Define_Level_Variable(Val(TOP-2));

        /* VAL(TOP-2)의 위치에 있는 변수를 심볼테이블에
           넣고 그 변수의 Level을 정한 후 규칙 (3)에서
           •기억된 변수의 길이를 해당 Field에 기억시킴 */

    2 : Update_Level_Number(Val(TOP));

        /* 77 Level 임을 기억시킴. 다른 Level에 대해서는
           Level_Number_Tree를 구성하여 Level별로 정보를
           Handling할 수 있도록 해 준다. */

    3 : Picture_Symbol_Analysis(Val(TOP));

        /* Picture 심볼의 형태를 분석함으로써
           변수의 길이등을 계산하여 규칙 (1)로 넘겨줌.
           이를 위해 계산된 값을 LHS(Left-Hand Side)에
           기억시켜 둠. */

    4 : Insert_In_Constant_Table_and_Return_the_Value(Val(TOP));

        /* Picture 심볼을 상수테이블에 넣고 그 값을
           위의 규칙 (3)으로 넘겨줌. 이를 위해 그 값을
           LHS로 Copy하여 기억시켜 둠 : (StackCopy(Val(TOP)) */

END CASE ;
```

이와 같이 문법 지시적 변환 방법을 이용한 정적 분석에서는 여러 규칙에서 동일한 작업을 하는 부분, 예를 들면 심볼테이블을 조작하는 부분등을 하나의 프로시쥬어로 만들고 각 규칙이 Reduce 될 때 각 규칙별로 그 프로시쥬어를 호출하여 실행토록 구성된다.

여 백

제 3 장　동적 지원 시험 시스템의 개발 방향

여 백

제 3 장　동적 시험 지원 시스템의 개발 방향

제 1 절 동적 시험 지원 시스템의 특징

동적 시험 지원 시스템은 다음과 같은 시험 활동을 지원하고 각 정보를 관리하여, 도구를 통한 시험 과정의 체계화를 추구한다.

1. 시험 항목의 정의
2. 시험 기능의 설계
3. 시험 사례의 설계
4. 사례의 수행
5. 수행결과 분석 및 오류분석

각 절차별 동적 시험 지원 시스템의 기능은 [표 3-1]과 같다.

각 시험 활동 과정중에 생성되는 시험 문서는 시험의 목적을 달성하기 위한 수단이며, 관련 팀간의 의사 소통 도구라고 할 수 있다 개발 공정의 요구분석, 설계 및 코팅 단계에 비하여 시험 관련 문서는 개념과 표준이 잘 정의되어 있지 않다. 동적 시험 지원 시스템에서는 IEEE 소프트웨어 공학 표준에 근거하여 표준 시험 문서를 정의하고 이를 지원한다.

[표 3-1] 시험 절차별 동적 시험 지원 시스템의 기능

시험 절차	내　　용
시험 항목 선정	• 시험 대상 소프트웨어의 등록, 조회 • 정적 품질 평가 및 분석

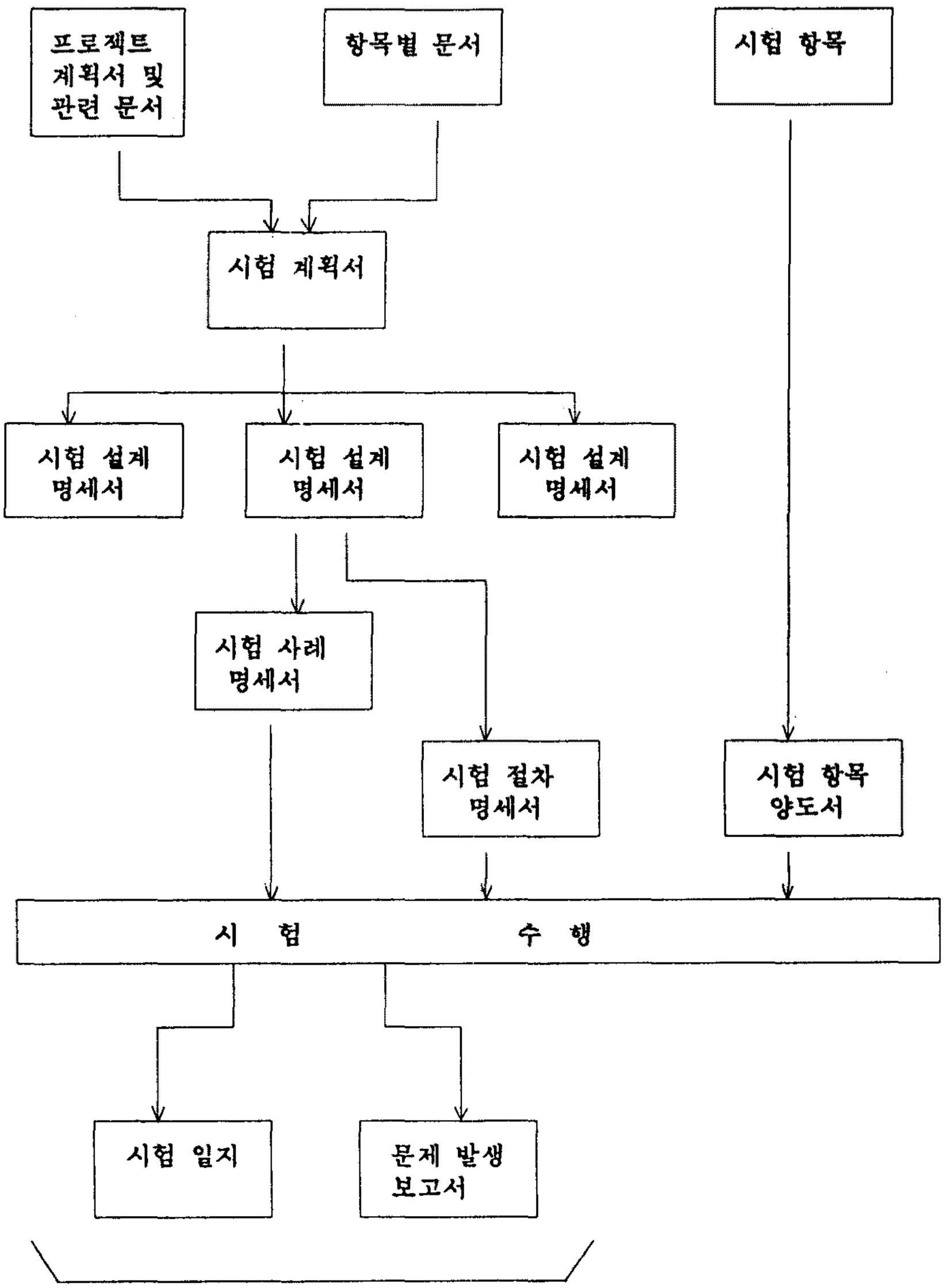
시 험 절 차	내 용
시험 기능 설계	• 시험 단계별로 시험할 기능을 정의, 등록 • 시험 기능의 완전성 점검
시험 사례 설계	• 시험 기능 수행 입력 데이타, 예상 출력을 정의 • 기능, 사례 범주 조회를 통해 최소한의 사례 작성 • 각 사례의 수행 순서 기술
사례의 수행	• 계기 삽입 프로그램 (Instrumented Code) 을 수행 • 수행 경로에 대한 결과 확인 • 미 시험 코드에 대한 사례 분석과 추가 사례 작성 • 시험 진행 점검 - 항목, 기능, 사례 등
수행결과분석 및 오류 분석	• 결과 확인 및 문제 발생 보고서 작성 • 오류 분석 및 개선점 분석 • 시험 수행 활동 및 결과 요약

시험 활동 과정중에 작성되는 문서는 시험 계획서, 시험 설계 설명서, 시험 사례 명세서, 시험 절차 명세서, 시험 항목 양도서, 시험 일지, 문제 발생 보고서, 시험 요약 보고서 등으로 주요한 기술 내용은 [표 3-2] 와 같다.

[표 3-2] 표준 시험 문서 내용

시험 문서	내 용
시험 계획서	• 시험 대상 항목 및 시험 대상 기능 • 기법, 도구등 시험 접근 방법 기술 • 산출물, 종결기준
시험 설계 명세서	• 시험 기능 상세화 • 사례 설계 기법, 결과 확인 기법 상세화
시험 사례 명세서	• 시험 사례 ID. • 입력조건 및 출력 • H/W,S/W,절차 및 사례 의존 관계
시험 절차 명세서	• 일련의 사례 수행 순서 • 시험 항목의 기능 확인 절차
시험 항목 양도서	• 시험 항목 라이브러리, 책임자, 상황등
시험 일지	• 시험 작업 및 시험 상황 등 기록
문제 발생 보고서	• 시험 항목, 수행 사례, 수행 결과, 재현 조건 등
시험 요약 보고서	• 시험 항목 종합 평가, 작업 및 투입노력 요약

각 시험문서의 연관 관계를 도식화 하면 [그림 3-1]과 같이 나타낼 수 있다.



[그림 3-1] 시험 절차와 표준 시험 문서

제 2 절 동적 시험 지원 시스템 환경

동적 시험 지원 환경등 시험 설계 정보 및 수행 정보의 관리는 구현 언어에 무관하나 시험 범주 분석기는 원시 프로그램에 계기 삽입을 하여 구현되므로 언어에 의존적이다.

도구가 분석하는 언어는 현재 국내 소프트웨어 개발 조직의 70% 정도가 사용하고 있는 코볼로 한정하였다. 코볼의 종류를 살펴보면 ANSI COBOL, COBOL 74, COBOL 85(COBOL II) 등이 있는데, COBOL II는 기존의 코볼에 부족한 EVALUATE, INITIALIZE문등의 추가와 IF~END-IF 와 같은 블록의 개념을 추가한 것이다. 현재 국내의 개발기관은 COBOL 74에서 점차 COBOL II로 변경을 해 나가는 추세이다. 이러한 점을 고려하여 COBOL 74 와 COBOL II를 동시에 지원할 수있도록 하였다. 하드웨어 기반은 PC로 하였는데 그 이유는 도구의 정적 분석 및 메트릭 계산등의 과정에서 많은 컴퓨팅 자원을 필요로 하나, 결과의 출력은 즉시성을 요구하지 않으므로 컴퓨터 자원을 분배하여 사용할 필요가 있기 때문이다. 또한 정적 분석 과정에서 산출된 메트릭은 정보의 이용 대상자에 따라 다양하게 가공되거나 그래픽화할 필요가 있는데 이 작업은 PC의 여러가지 소프트웨어를 이용하여 수행 할 수 있다는 장점이 있다. 개발된 소프트웨어의 이식성을 고려하여 운영체제는 UNIX, 개발 언어는 C, 사용자 접속은 X를 사용하였다.

[표 3-3] 주로 사용되는 컴퓨터 언어

순 위	컴 퓨 터 언 어	1) 사용기관수	2) 사용율(%)	3) 88.2 조사
1	COBOL	108	77.7	74.4
2	C-LANGUAGE	46	33.1	41.0
3	ASSEMBLER	41	29.5	41.0
4	FORTRAN	35	25.2	25.6
5	BASIC	35	25.2	-
6	PL/1	26	18.7	17.9
7	DBASE	24	17.3	-
8	PASCAL	11	7.9	7.7
9	RPG	10	7.2	-
10	기 타	8	5.8	

주1) '89.5, '국내 소프트웨어 형상 및 변경관리 실태조사', 시스템 공학센터

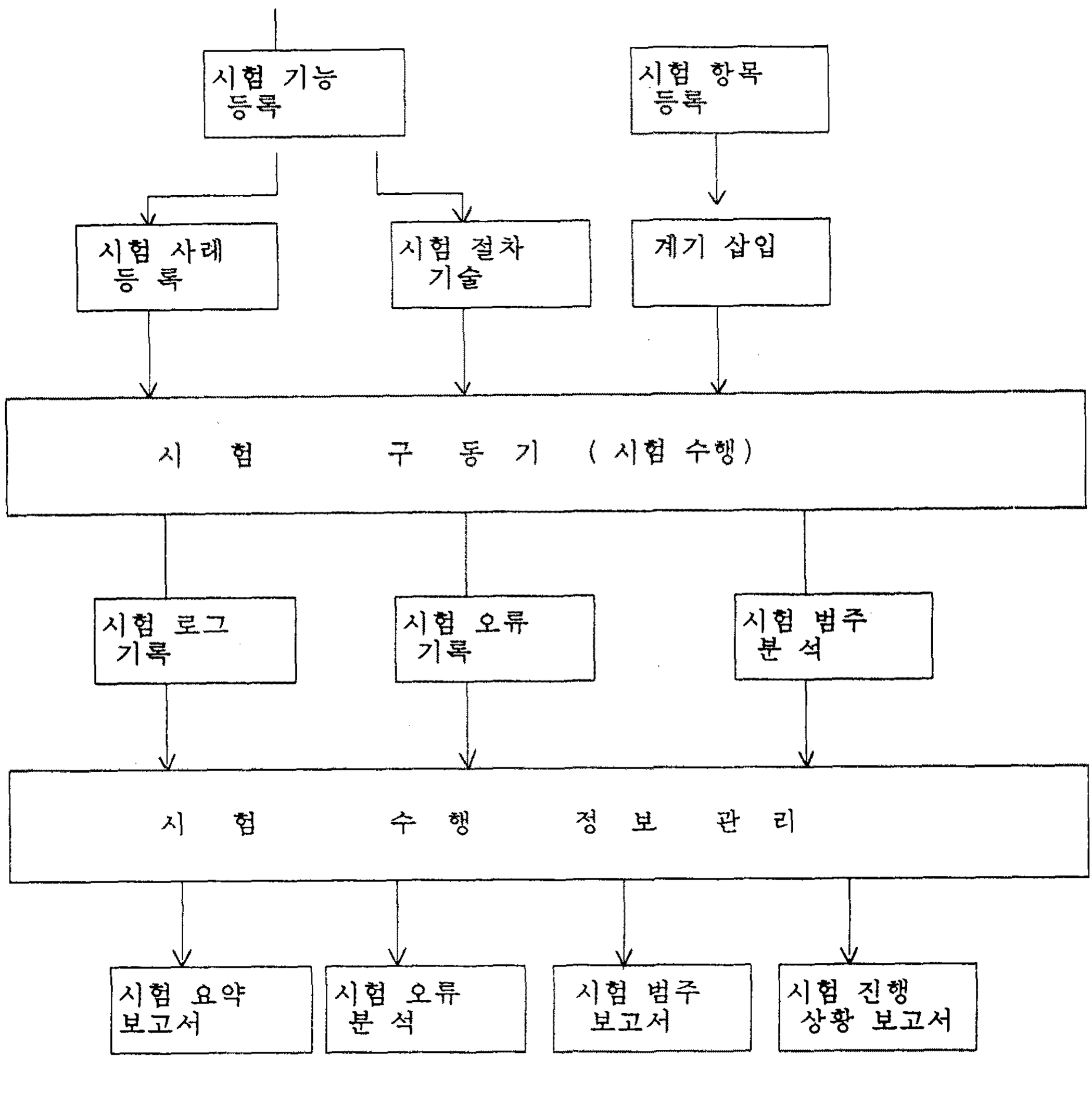
2) 응답기관별로 사용언어 모두를 중복 기재한 수치임.

3) '88.2 조사, '국내소프트웨어 개발환경조사', 시스템공학센터

제 3 절 동적 시험 지원 시스템의 기능 및 구조

동적 시험 지원 시스템은 크게 시험 절차 관리기와 시험 결과 처리기로 나누어지며 각 도구의 특징과 기능은 아래와 같다.

구조를 도식화하면 [그림 3-2] 와 같다.



[그림 3-2] 동적 시험 지원 시스템의 구조

1. 시험 절차 관리기

가. 도구의 특징

ㅇ 단위 기능의 독립적 수행과 통합 도구로서의 기능

• 소프트웨어 품질 평가도구, 설계 정보 생성 시스템 인터페이스

ㅇ 시험 항목의 선정 및 기능 설계등 시험 설계 작업의 체계화

ㅇ 시험 사례 설계 지원

- 시험 기능을 수행하기 위한 입력 데이타와 예상 출력을 정의
- 기능, 사례 범주 분석을 통하여 최소의 사례 작성
- 시험 수행전 준비 화일 등 준비 절차와 시험 수행시 유의사항 및 절차를 시험 절차에 기술

ㅇ 수행 결과의 분석 및 오류 기록

- 시험중 발견된 오류는 기록되어 유형별, 원인별 분석
- 시험 진행을 통제하기 위한 통계, 추이분석 정보 제공
- 시험 투입 노력과 효과의 분석

ㅇ 시험 정보의 관리

- 문서의 표준화
- 시험 설계 정보의 재이용 환경 구성

나. 기 능

ㅇ 시험 항목 및 기능 관리

- 항목 등록 및 조회

- 항목 시험 추이 분석
- 단계별 시험 기능의 등록 및 조회
- 시험 기능, 사례 범주 분석
- 기시험 기능 및 미시험 기능 출력
- 기능 시험 추이 분석

○ 시험 사례 및 절차 관리

- 시험 사례의 작성 및 조회
- 시험 절차의 작성 및 조회
- 시험 사례 작성 상황과 수행 상황 출력
- 사례 수행 추이 분석

○ 시험 오류 및 로그 관리

- 문제 발생 보고서 작성 및 조회
- 시험 오류 분석 및 통계
- 오류 발생 추이 분석
- 시험 로그의 입력 및 조회
- 시험 요약 보고서 생성

• 시험 투입 노력 및 효과 분석

2. 시험 결과 처리기

가. 도구의 특징

○ 시험 추적 결과를 이용 미시험 부분에 관한 정보를 제공하여 시험의 완전성을 추구

○ 시험 프로그램을 정적 분석하여 시험 범주 기록 계기 삽입

ㅇ 시험 구동기는 시험 사례의 입력 자료를 수행되는 프로그램의 입력으로 전달하고 수행된 결과를 기록

ㅇ 프로그램이 수행되면서 삽입된 계기는 레이블, 분기, 문, 호출 관계 등에 관한 수행 정보를 기록하는 프로그램 호출

나. 계기 삽입기 기능

ㅇ 통계자료 및 기본속성 저장

- 호출 관계
- 프로그램 정보
- 프로그램내 패러그래프, 분기 및 문 구조와 정보

ㅇ 코볼 원시 프로그램내 계기 삽입

- 어휘, 구문 분석
- 코볼 예약어 및 생성 규칙 유지
- 통계 누적화일 형태 정의
- 수행 추적 기록 및 입출력 기록 계기 삽입

다. 범주 분석기 기능

ㅇ 코드의 최초 수행 부분 지적

ㅇ 미시험 부분 지적

ㅇ 범주별 일부 수행 부분 지적

ㅇ 상세 범주 전이 기능

ㅇ 제공 보고서

- 시스템 범주

- 시스템 호출 범주
- 금회 범주 요약
- 금회 레이블 범주
- 금회 호출, 분기, 문 범주
- 사례별 누적 범주

라. 사용자 접속 기능

o 사용법과 제공 결과의 해석이 용이
o 시험 정보 선택 종류
  - 시스템 분석 정보
  - 프로그램 분석 정보
  - 시험 항목, 기능, 사례 등
o 시험 정보의 조작
  - 질의, 첨가, 갱신, 삭제, 인쇄
o 시험 항목의 조작
  - 항목 조회 및 편집
  - 계기 삽입
  - 정적 분석
  - 컴파일 및 실행
o 기타 선택 및 도움말
  - 저장 디렉토리 지정
  - 컴파일, 실행 옵션 지정
  - 계기 삽입 범위 지정
  - 호출 정보 관리 범위
  - 호스트 시스템과 PC간 통신
  - 도움말 제공

여 백

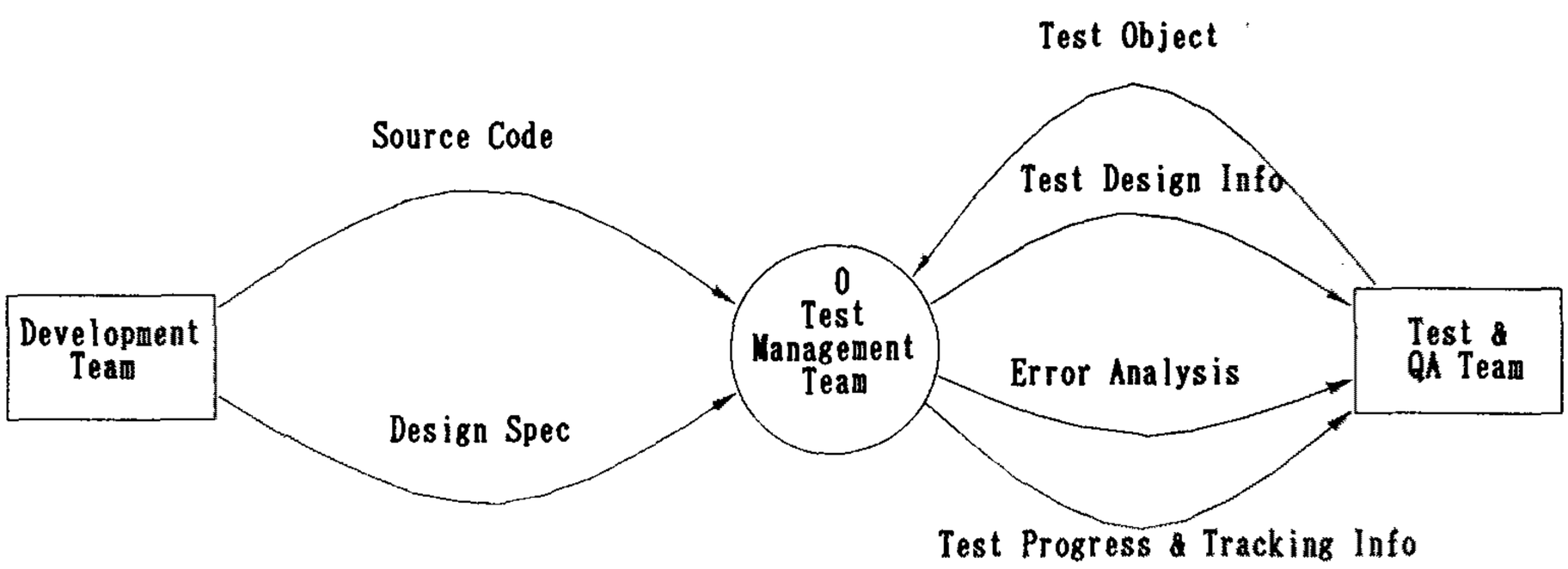
제 4 장　동적 시험 지원　시스템의 구현

여 백

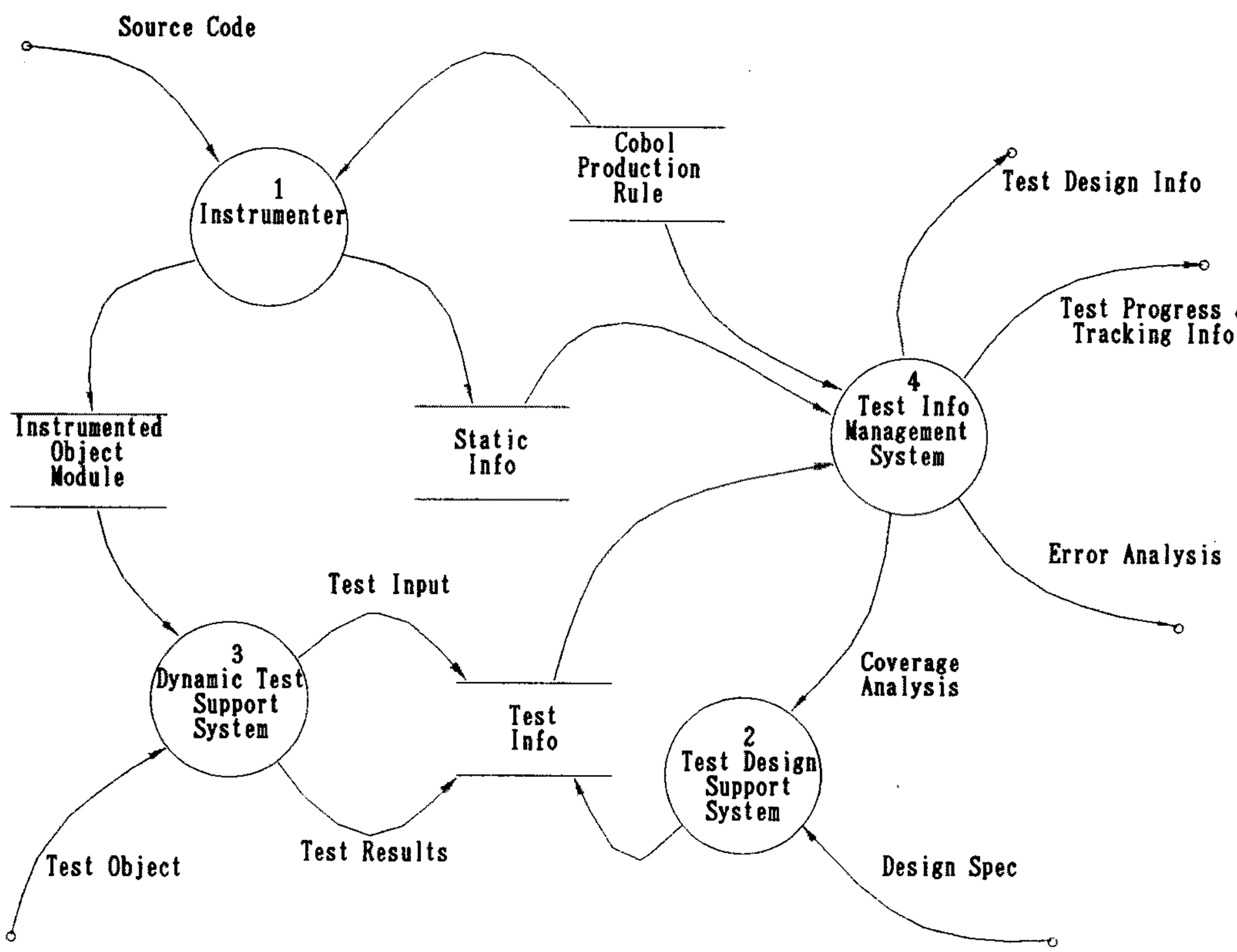
제 4 장　동적 시험 지원 시스템의 구현

제 1 절 자료 흐름도 및 자료 사전

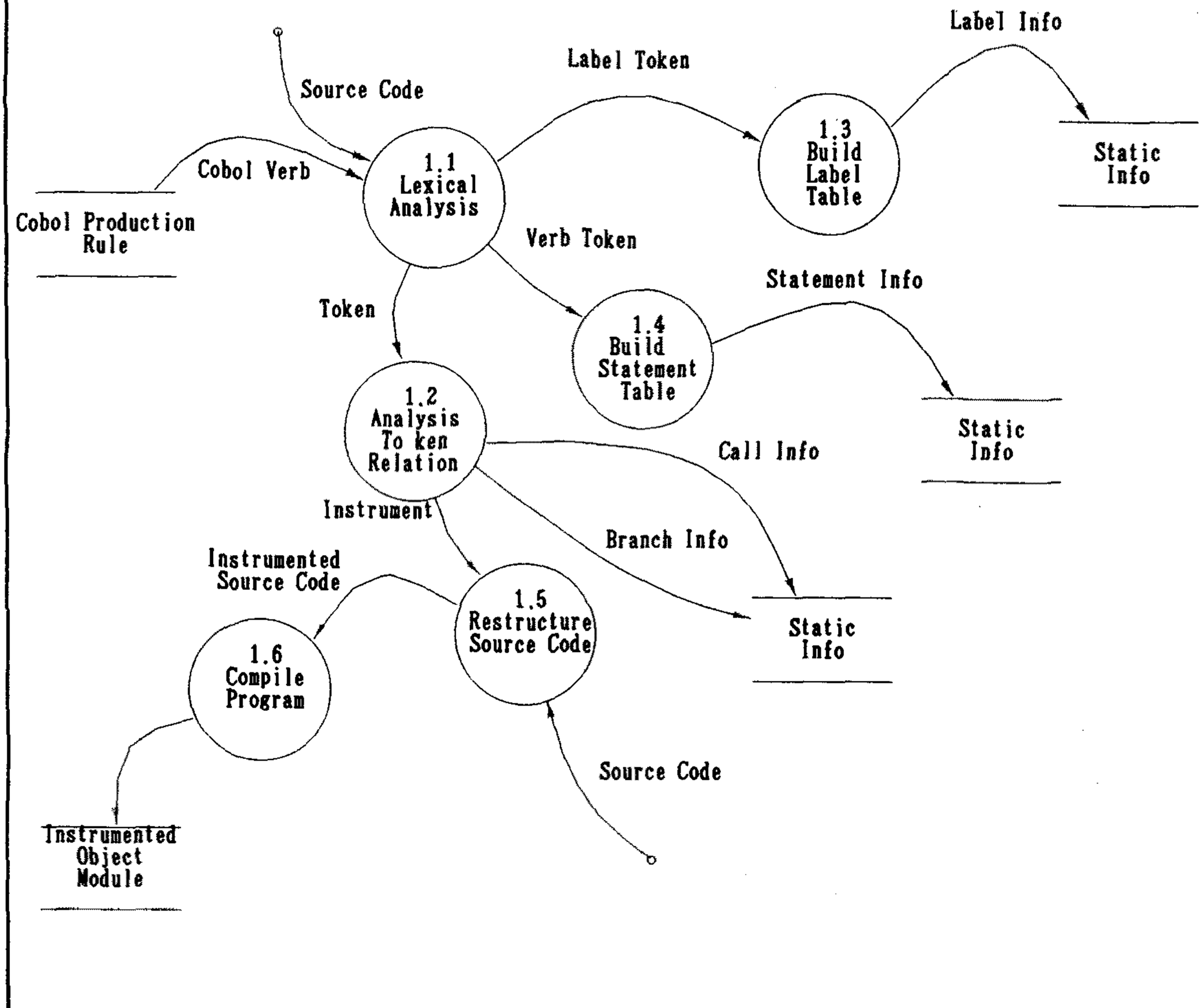
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목	Context Diagram				



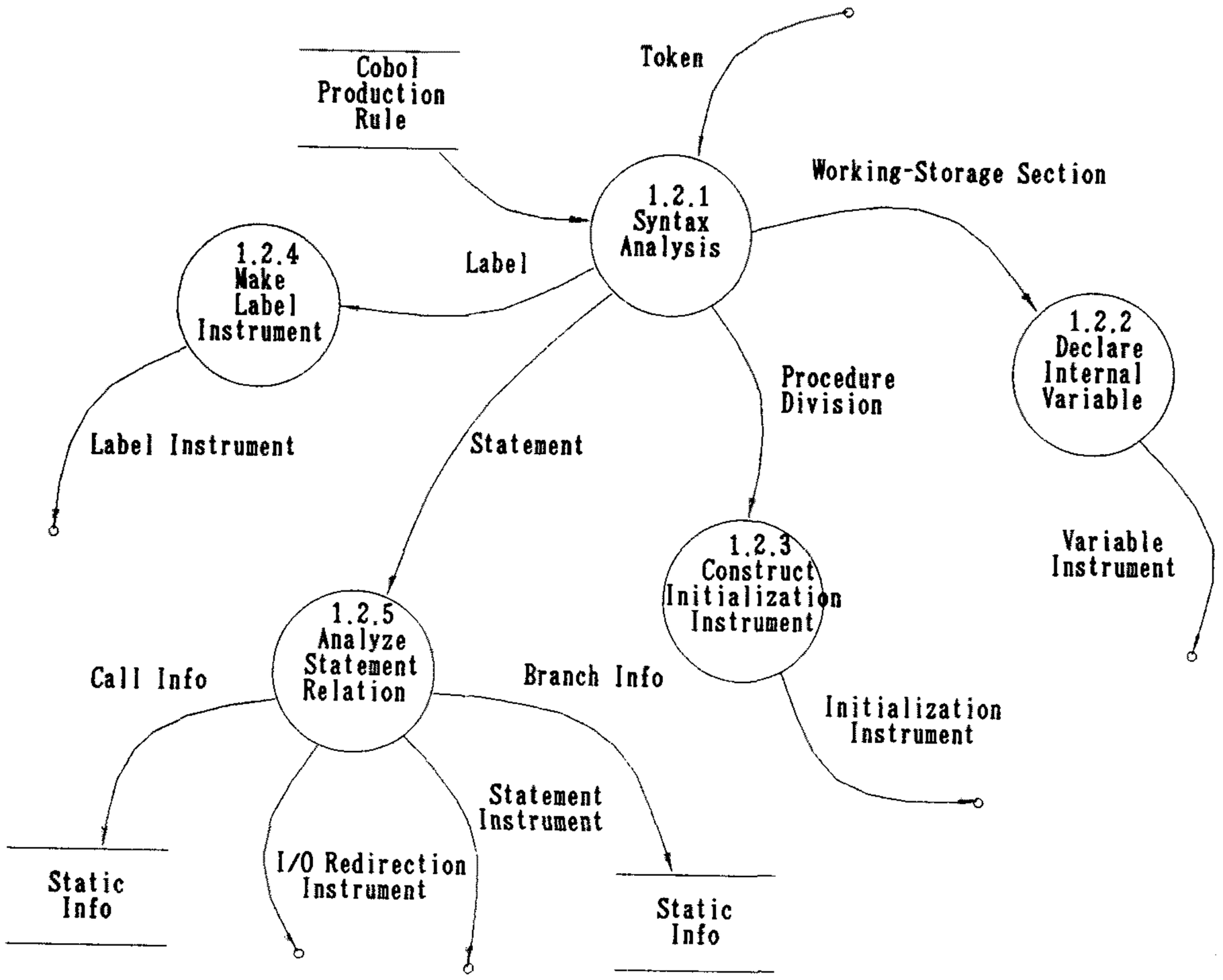
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목	Diagram O				



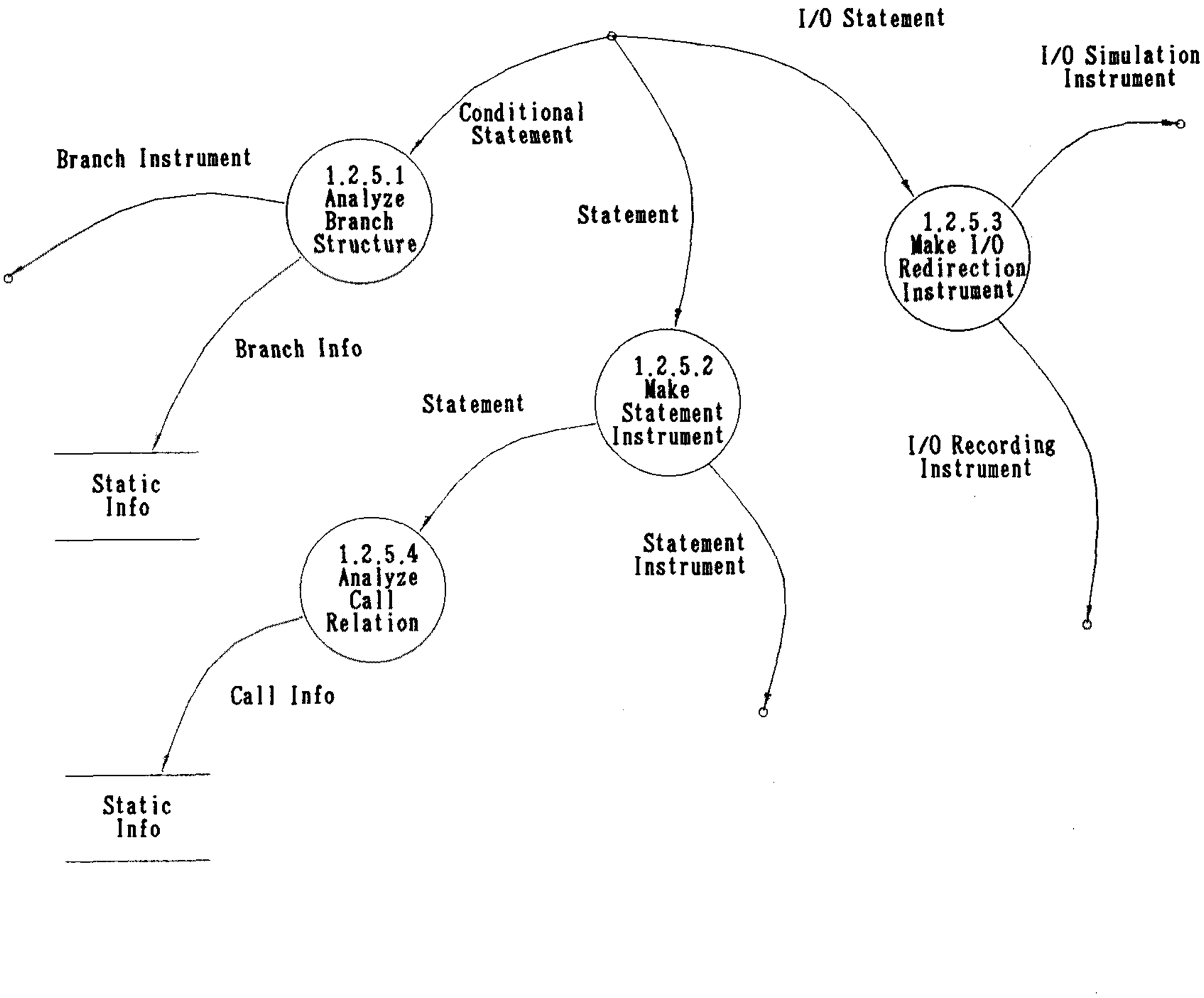
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목	1 Instrumenter				



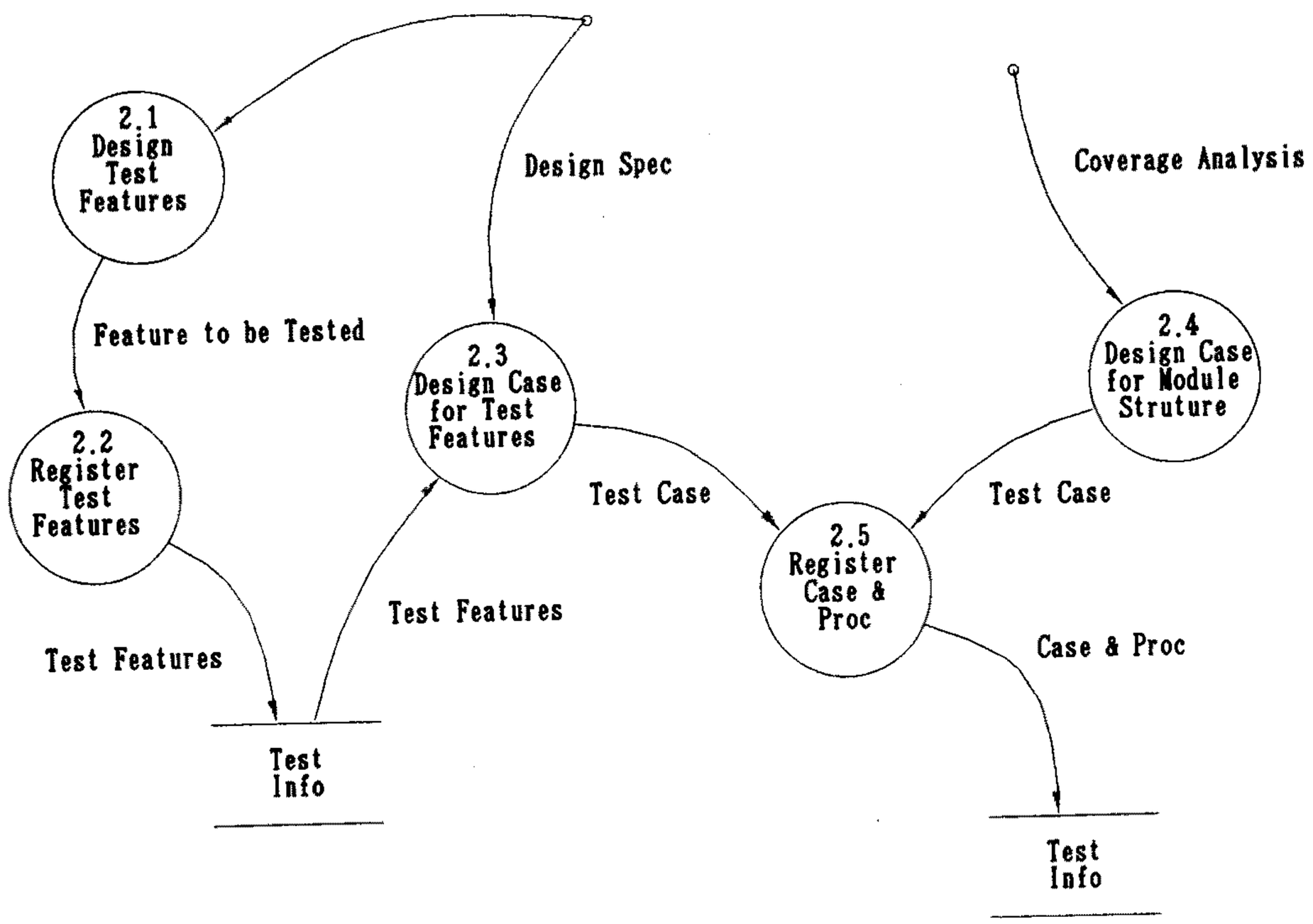
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목	1.2 Analyze Token Relation				



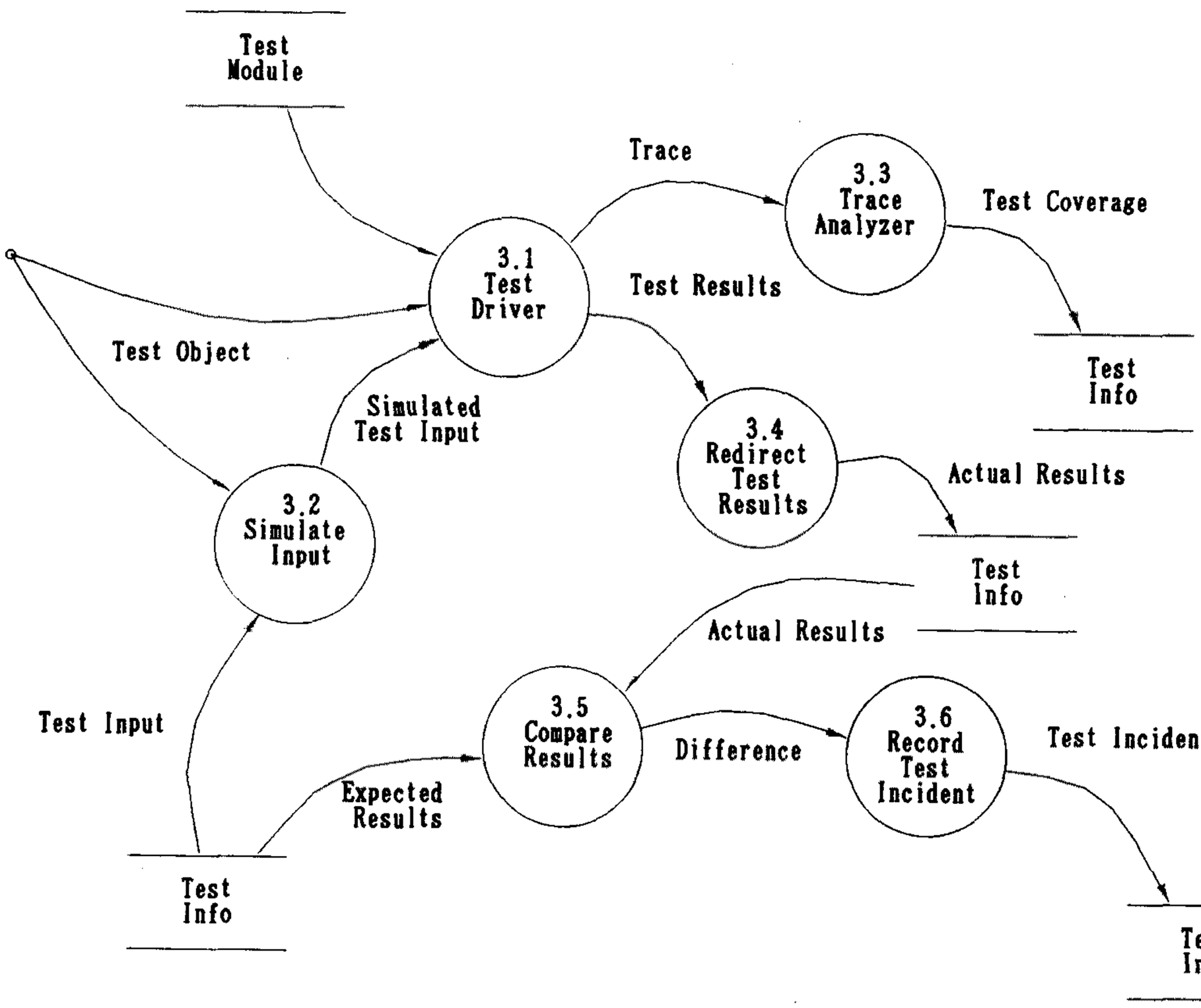
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목					



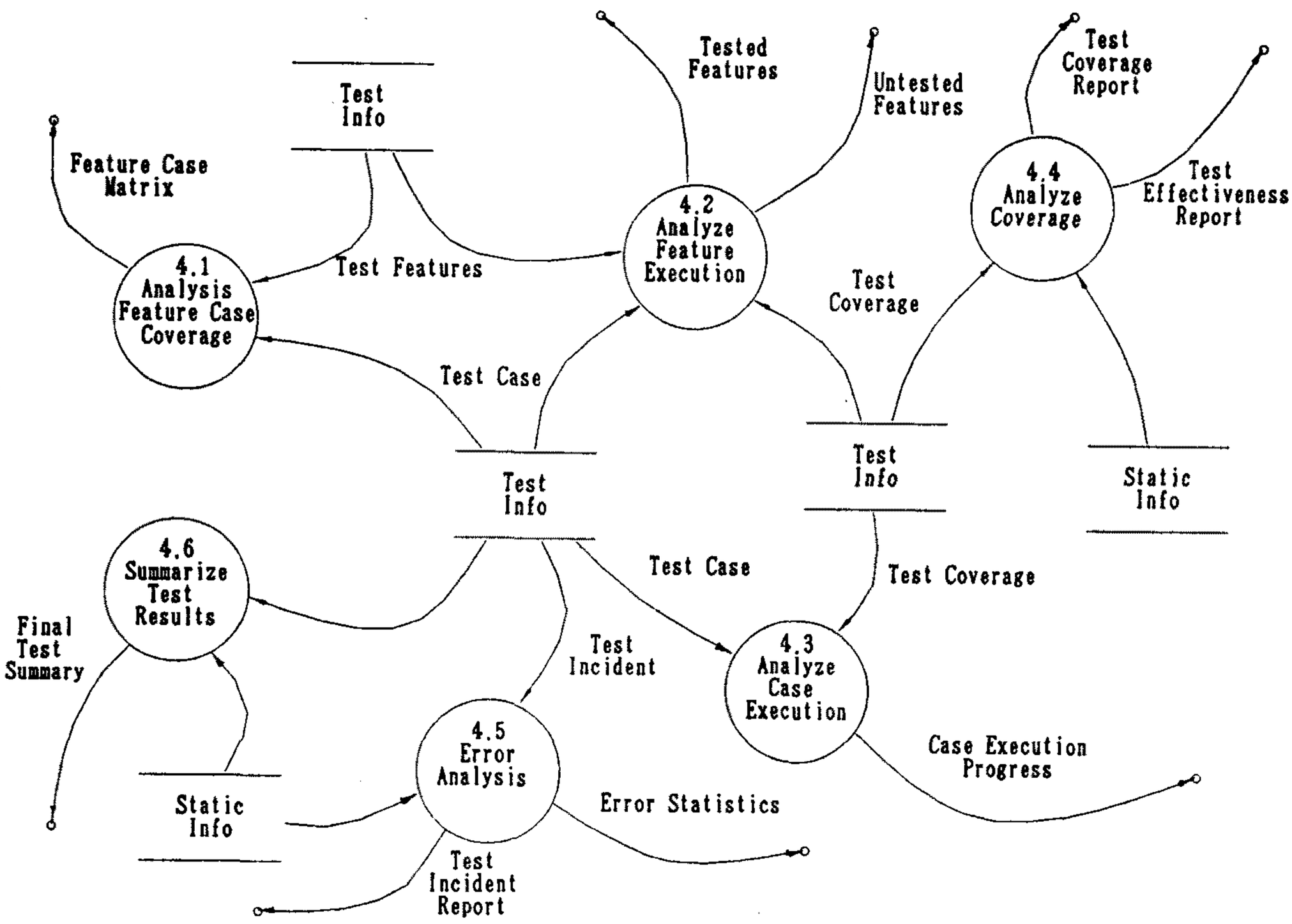
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목	2 Test Design Support System				



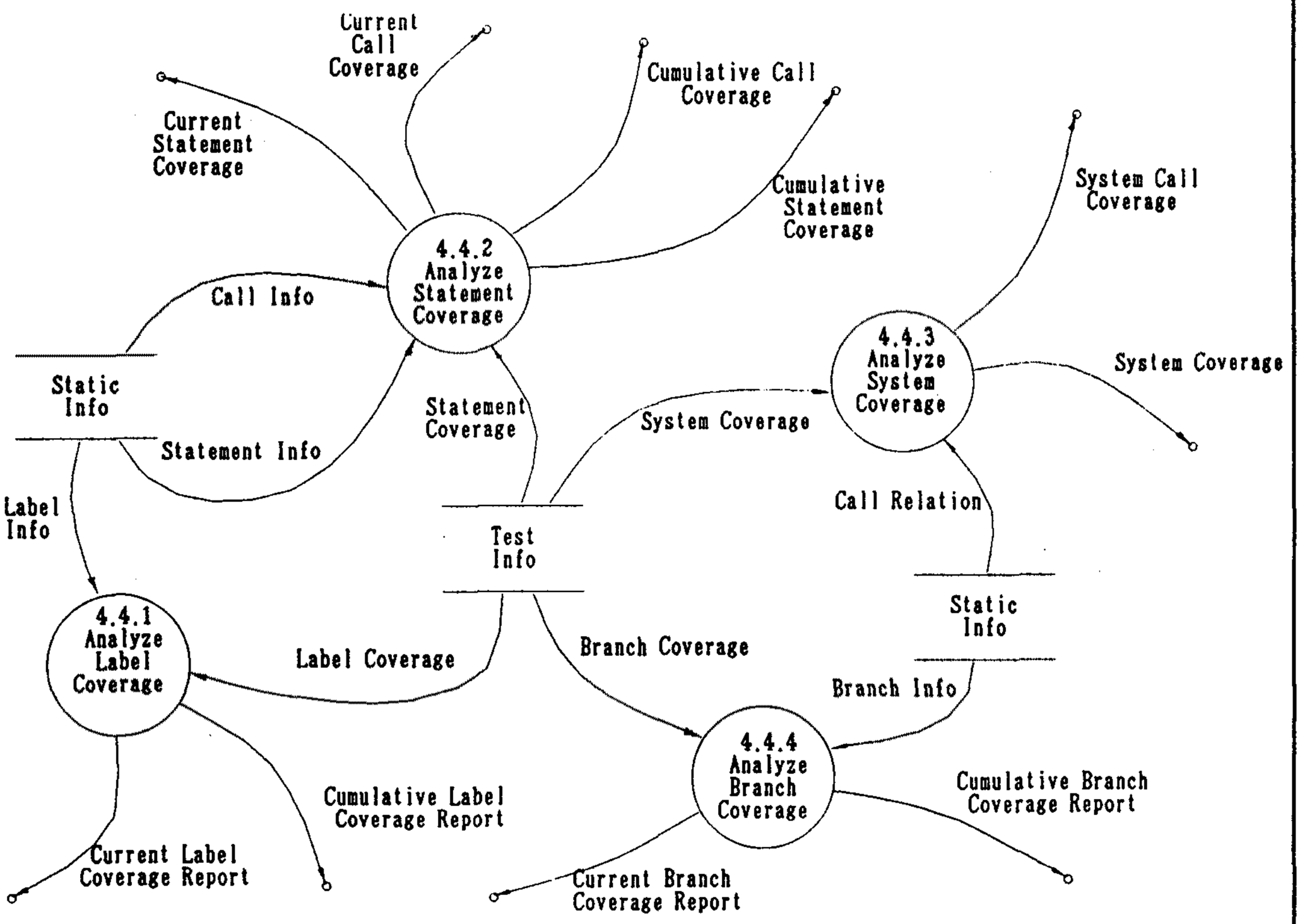
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목	3 Dynamic Test Support System				



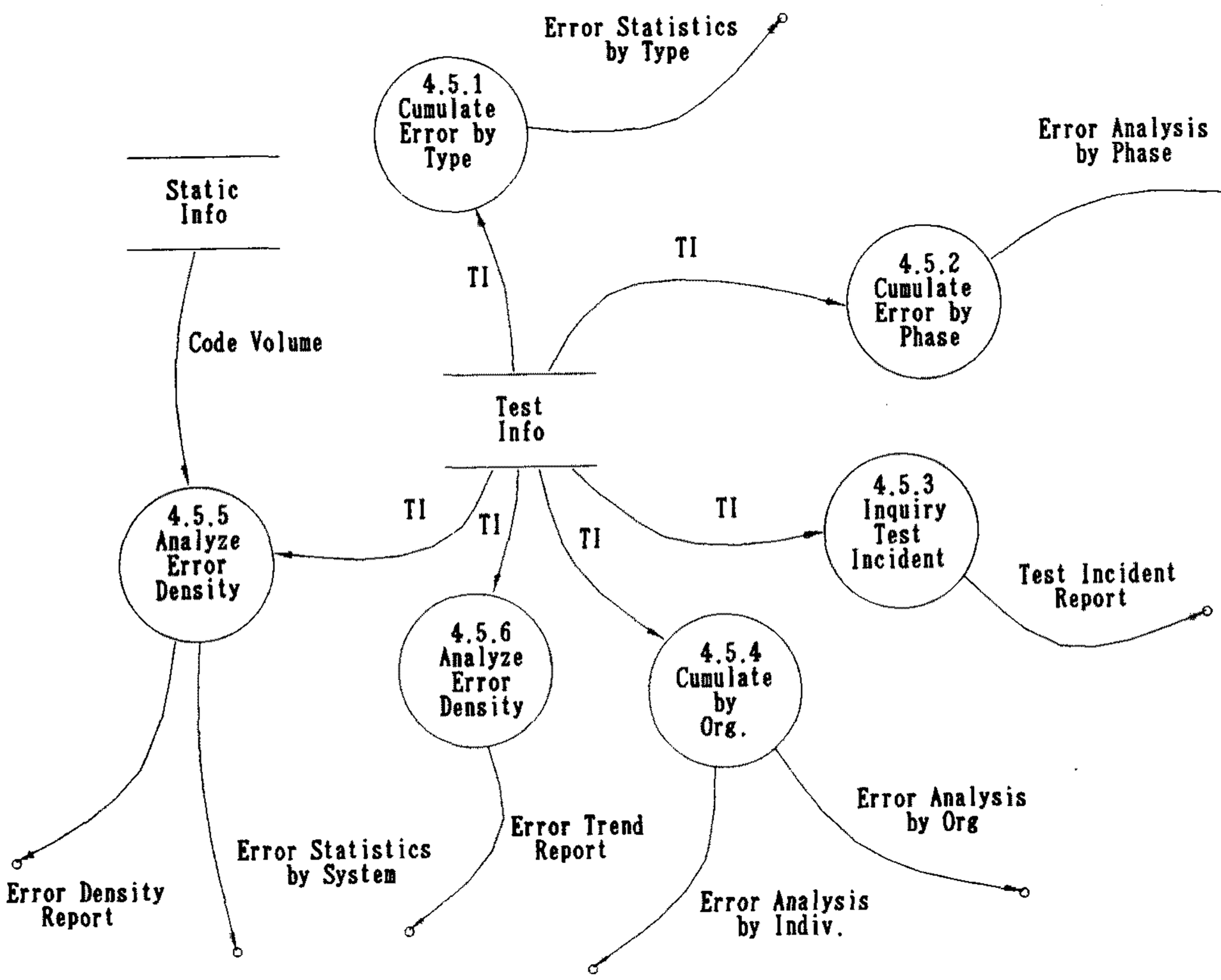
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목	4 Test Info Management System				



시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목	4.4 Analyze Coverage				



시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 흐름도	작 성 자		작성일	1991. . .
문서제목	4.5 Error Analysis				



시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 사전	작 성 자		작성일	1991. . .
문서제목	0				

원시 코드 (Source Code) = 프로그램 원시코드 + COPY 라이브러리

코볼 규칙 (Cobol Production Rule)
= 예약어 + 코볼 생성 규칙 + 동사 정보

정적 정보 (Static Info.) = 프로그램 정보 + 레이블 정보 + 분기 정보 + 문 정보 + 호출명

설계 명세 (Design Spec.) = 외부 설계 명세서 + D/B 화일 설계 명세서 + 내부 설계 명세서

시험 대상 (Test Object) = 시험 항목 + 시험 사례 ID

시험 정보 (Test Info.) = 시험 설계 정보 + 시험 결과

시험 결과 (Test Result) = 화면 출력 + 화일 출력 + 수행 추적 + 오류 정보

범주 분석 정보 (Coverage Analysis)
= 레이블 범주 + 분기 범주 + 문 범주 + 시스템 범주
+ 호출 범주

시험 설계 분석 (Test Design Info.)
= 시험 기능 + 시험 사례 + 시험 절차

시험 진행 정보 (Test Progress & Tracking Info.)
= 시험 범주 + 시험 효과 분석 + 기능 시험 상황 + 사례 시험 상황

오류 분석 정보 (Error Analysis)
= 오류 명세 + 오류 통계 + 오류 추이 분석

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 사전	작 성 자		작성일	1991. . .
문서제목	1 계기 삽입				

코볼 동사 (Cobol Verb.) = 동사 명 + 동사 번호

토큰 (Token) = 토큰 번호 + 토큰 값

호출 정보 (Call Info.) = 프로그램 + { 문 번호 + 호출 명 }

분기 정보 (Branch Info.) = 프로그램 정보 + { 분기 번호 + 분기 유형 + 분기 라인 +
레이블 번호 }

계기 삽입 코드 (Instrumented Source Code)
= 원시 코드 + 계기

레이블 정보 (Label Info.) = 프로그램 정보 + { 레이블 번호 + 레이블 유형 + 레이블 명 +
레이블 라인 }

문 정보 (Statement Info.) = 프로그램 정보 + { 문 번호 + 문 유형 + 문 라인 +
레이블 번호 }

계기 (Instrument) = 변수 계기 + 초기화 계기 + 레이블 계기 + 문 계기 +
입출력 전항 계기

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 사전	작 성 자		작성일	1991. . .
문서제목	1.2 토큰 관계 분석 및 계기 삽입				

레이블 (Label) = 레이블 명 + 레이블 출현 위치

문 (Statement) = [호출 문 | 분기 문 | I/O 문 | 기타 문]

초기화 계기 (Initialization instrument)
= 화일 처리 계기 + 변수 초기화 계기

입출력 전향 계기 (I/O Redirection Instrument)
= 입력 모사 계기 + 출력 기록 계기

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 사전	작 성 자		작성일	1991. . .
문서제목	2 시험 설계 지원기				

시험 기능 (Test Features) = 시스템 명 + 단계 번호 + 시험 항목 명 + 기능 번호 + 기능 명

시험 사례 (Test Case) = 시스템 명 + 사례 ID + 목적 + 작성자 + 작성일 +
입력 데이타 + 예상 결과 + 절차 번호

시험 절차 (Test Proc) = 절차 번호 + 준비 사항 + 수행 절차

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 사전	작 성 자		작성일	1991. . .
문서제목	3 시험 구동기				

수행 위치 (Trace) = 라인 + 수행 위치

시험 오류 (Test Incident) = { 시스템 명 + TIR 번호 + 사례 ID + 단계 번호 + 작성자 + 작성일 + 심각도 + 오류 상황 + 오류 원인 번호 + 오류 단계 번호 + 책임 조직 + 해결 방안 + 해결 여부 + 오류 번호 }

수행 결과 (Test Results) = { 시스템 명 + 프로그램 명 + 사례 ID + 수행 시간 }

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 사전	작 성 자		작성일	1991. . .
문서제목	4 시험 관리기				

사례 수행 상황 (Case Execution Progress)
= 사례 진행 상황 + 사례 수행 추이

사례 수행 추이 (Case Execution Trend)
= 사례 수 + 수행 사례 수 + 오류 발생 사례 수

사례 진행 상황 = 사례 ID + 사례 명 + 절차 작성 여부 + 수행 여부 +
오류 발생 여부 + 오류 해결 여부

시험 요악 보고서 = 시험 항목 수 + 시험 기능 수 + 시험 사례수 + 오류 발생 수 +
시험 인원 수

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 사전	작 성 자		작성일	1991. . .
문서제목	4.4 시험 범주 분석				

레이블 범주 (Label coverage)
= 수행 결과 + { 레이블 명 + 수행 횟수 }

문 범주 (Statement Coverage)
= 수행 결과 + { 문 명 + 수행 횟수 }

분기 범주 (Branch Coverage)
= 수행 결과 + { 분기 명 + 수행 횟수 }

시스템 호출 범주 (System Call Coverage)
= 시스템 명 + 프로그램 명 + 작성자 + 문 번호 + 호출 명 +
수행 횟수

시스템 범주 (System Coverage)
= 시스템명 + 프로그램 명 + 프로그램 수 + 수행된 수 +
수행 안된 수 + % 범주 + 라인 + 비고

수행 결과 = 시스템명 + 프로그램 명 + 사례 ID + 수행 시간 + 총 갯수 +
수행 갯수 + 수행 %

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	자료 사전	작 성 자		작성일	1991. . .
문서제목	4.5 오류 분석				

유형별 오류 통계 (Error Statistics By Type)
= 오류 원인 + 오류 수 + 총 오류 수에 대한 백분율

단계별 오류 통계 (Error Analysis By Phase)
= 오류 발생 단계 + 오류 수 + 총 오류 수에 대한 백분율

개인별 오류 통계 (Error Analysis By Indiv.)
= 작성자 + 오류 수 + 총 오류 수에 대한 백분율

조직별 오류 통계 (Error Analysis By Org.)
= 책임 조직 명 + 오류 수 + 총 오류 수에 대한 백분율

시스템별 오류 통계 (Error Statistics By System)
= 시스템 명 + 오류 수 + 총 오류 수에 대한 백분율

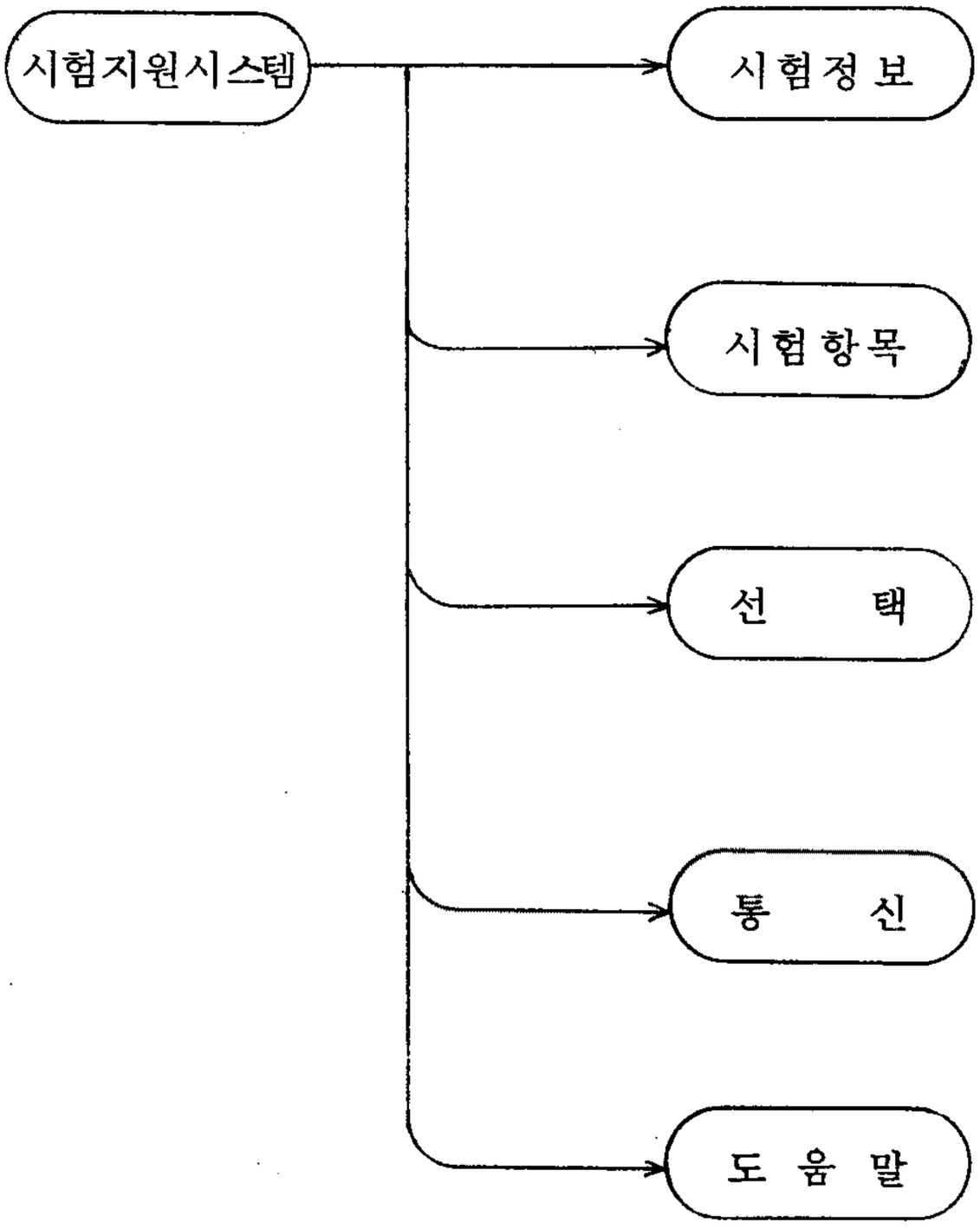
오류 밀도 분석(Error Density Report)
= KLOC당 오류 수

오류 발생 추이 분석(Error Trend Report)
= 시간 + 누적 오류 수

제 2 절 사용자 접속 설계

1. 화면제어 흐름도

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	화면 제어 흐름도	작 성 자		작성일	1991. . .
문서제목	주 메뉴				



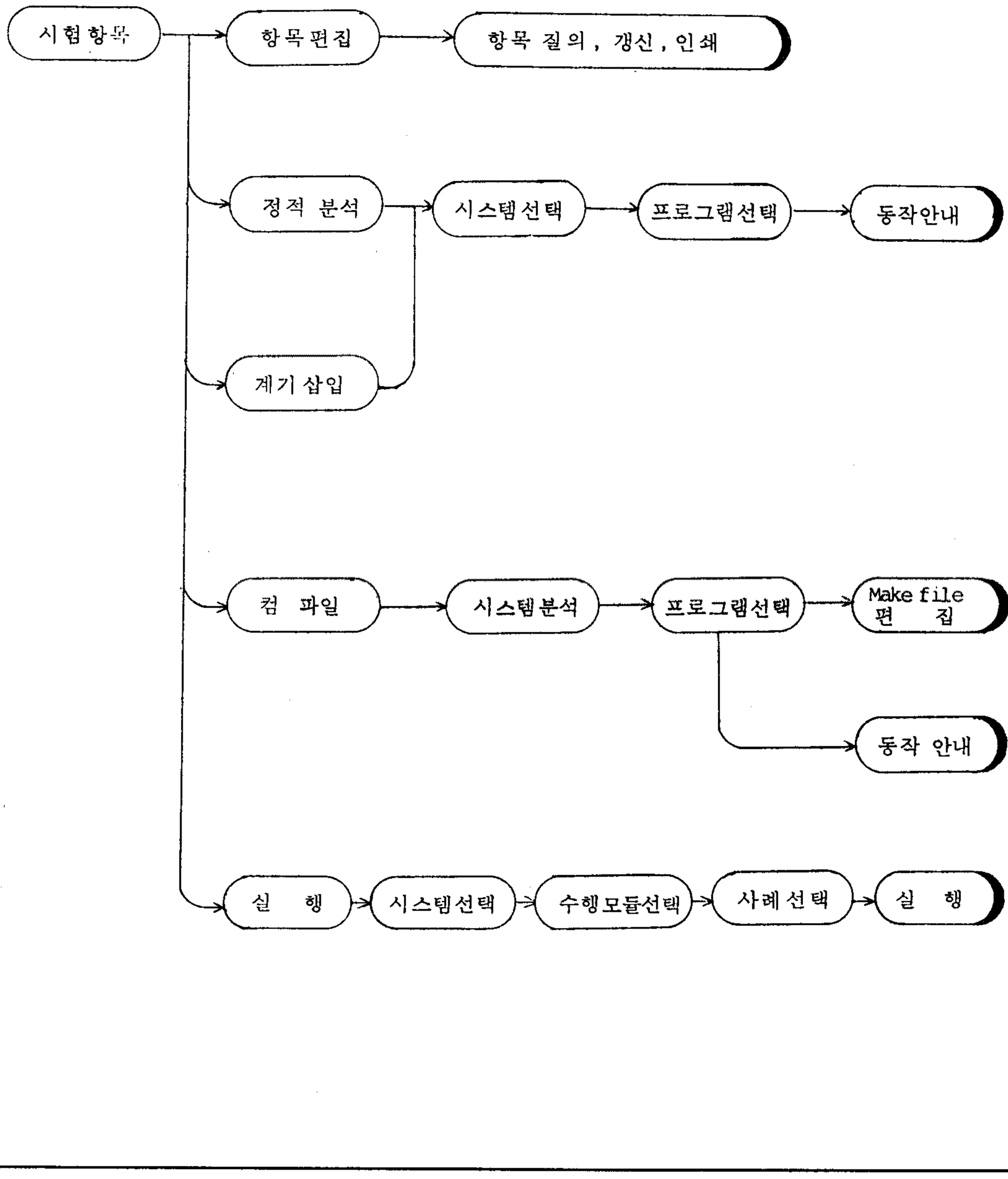
화면 제어 흐름도 표기법

중간 단계 화면 : 상위 메뉴, 하위 메뉴로 전이 가능

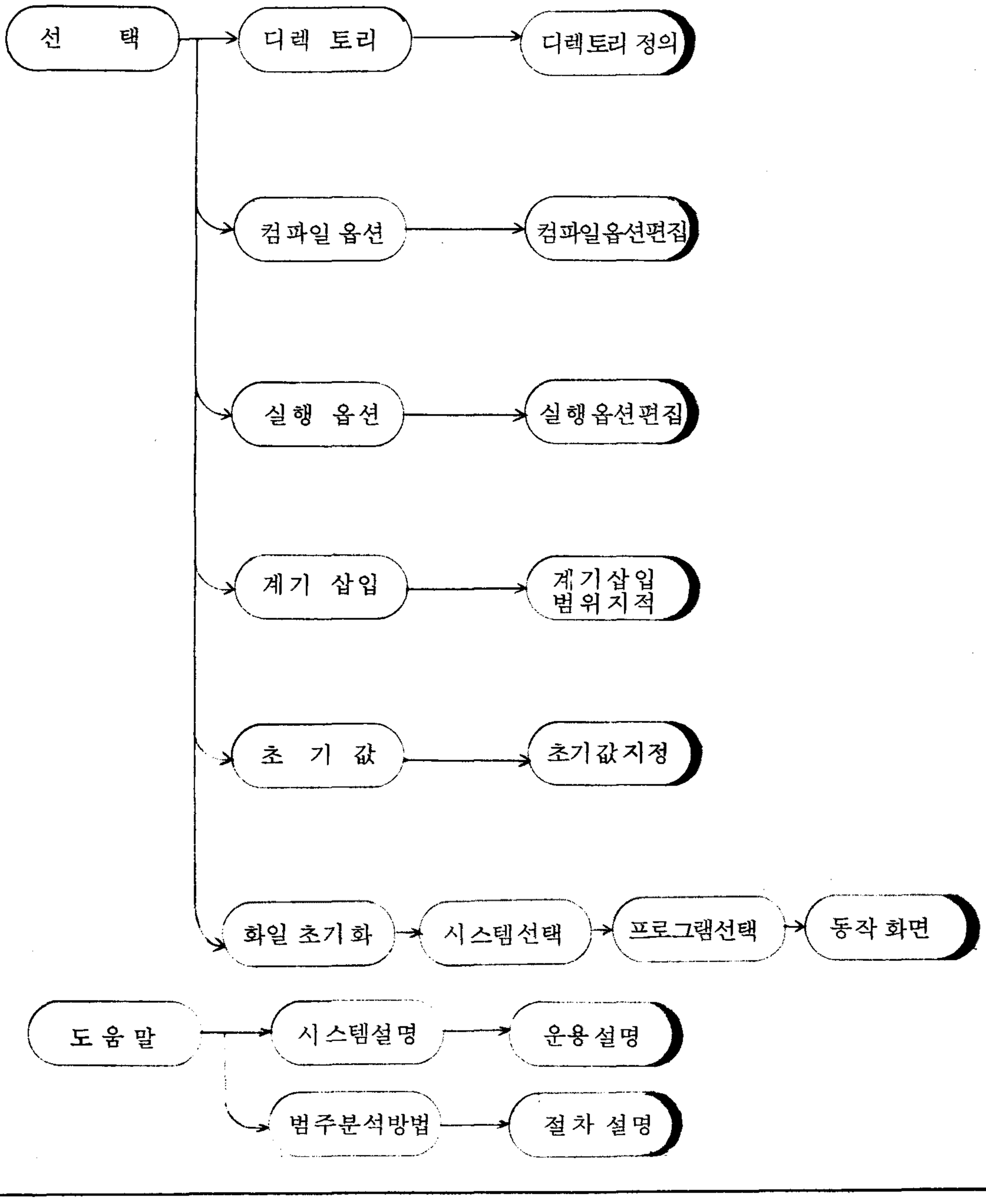
최종 화면 : 상위 메뉴로 전이 가능

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	화면 제어 흐름도	작 성 자		작성일	1991. . .
문서제목	시험 정보				

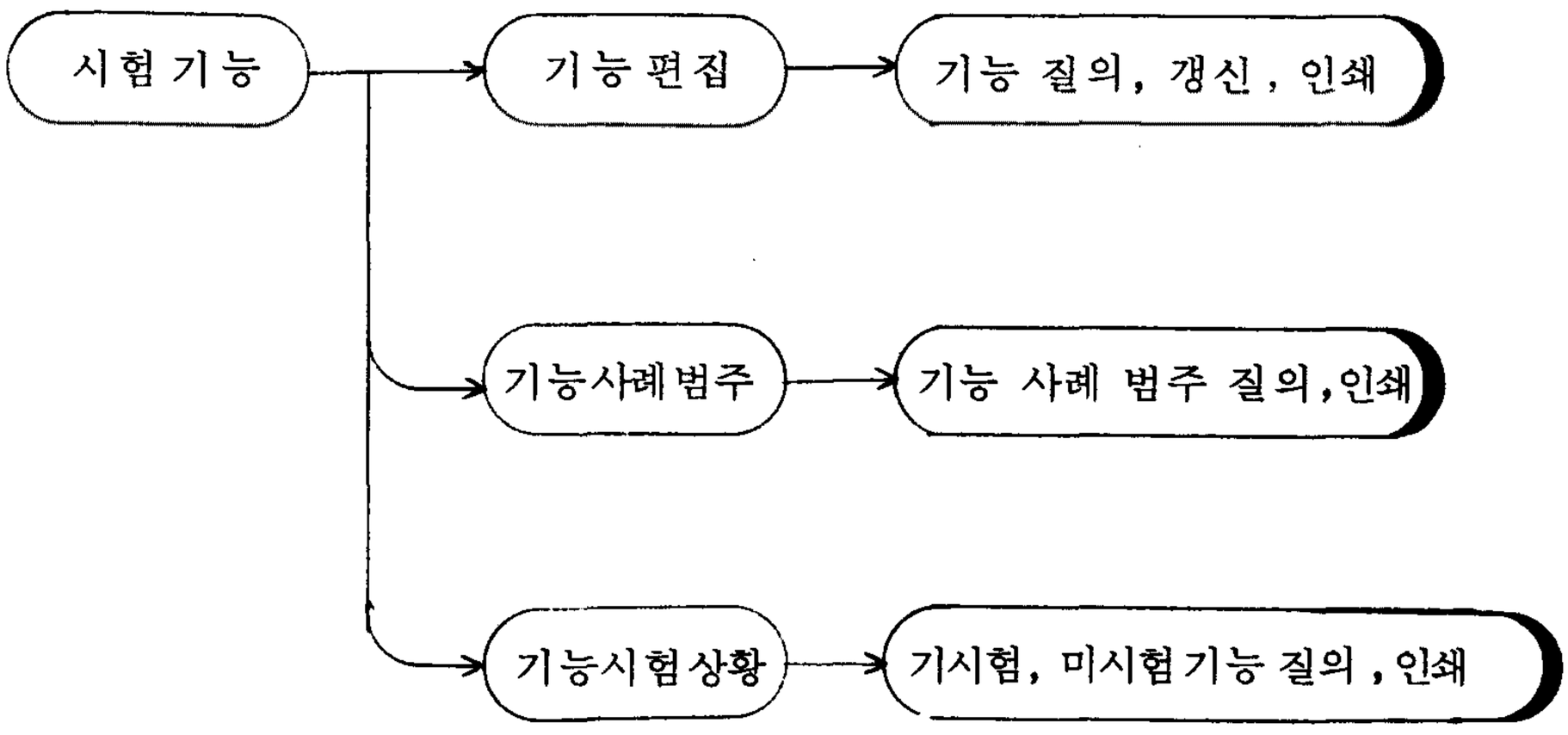
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	화면 제어 흐름도	작 성 자		작성일	1991. . .
문서제목					

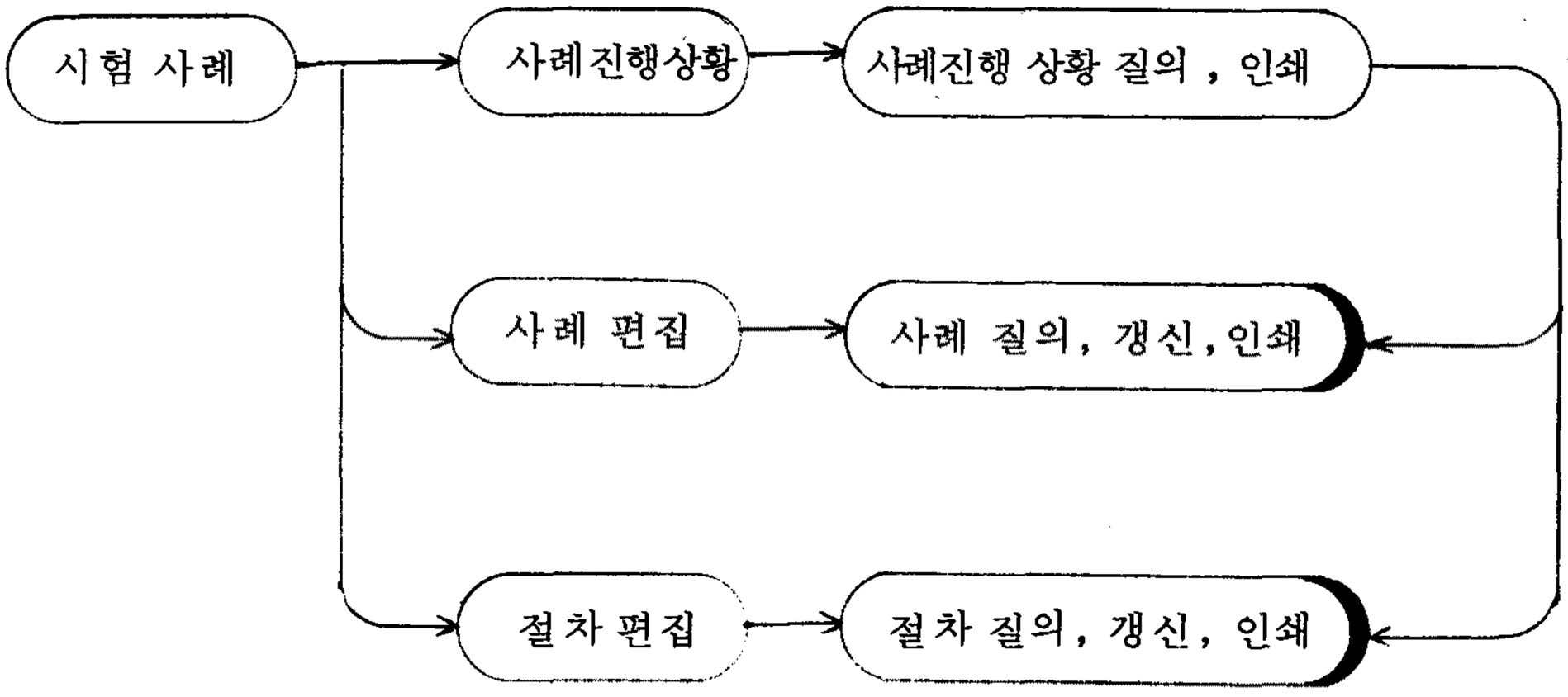


시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	화면 제어 흐름도	작 성 자		작성일	1991. . .
문서제목	선택, 도움말				

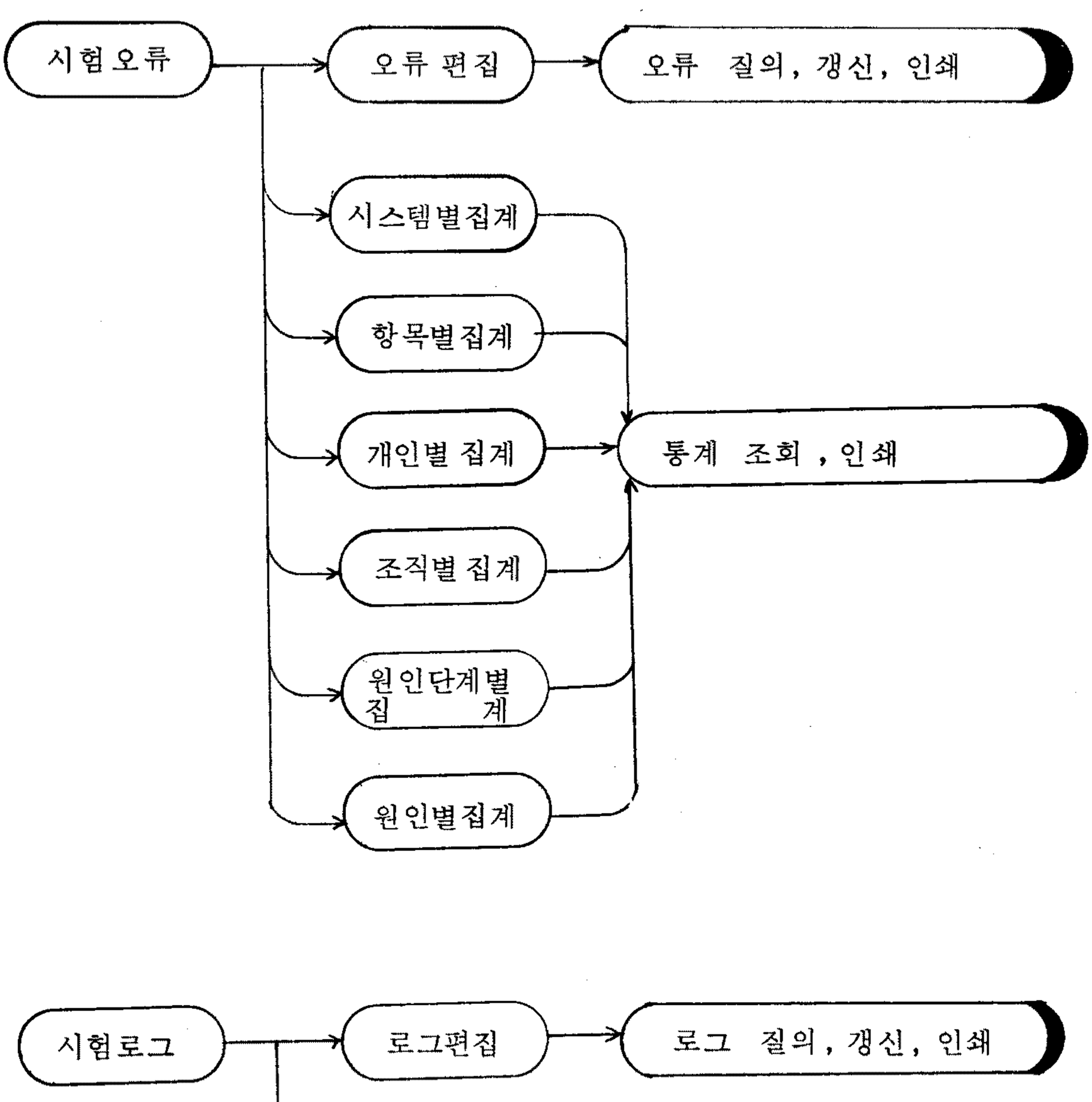


시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	화면 제어 흐름도	작 성 자		작성일	1991. . .
문서제목	시험기능, 시험사례				

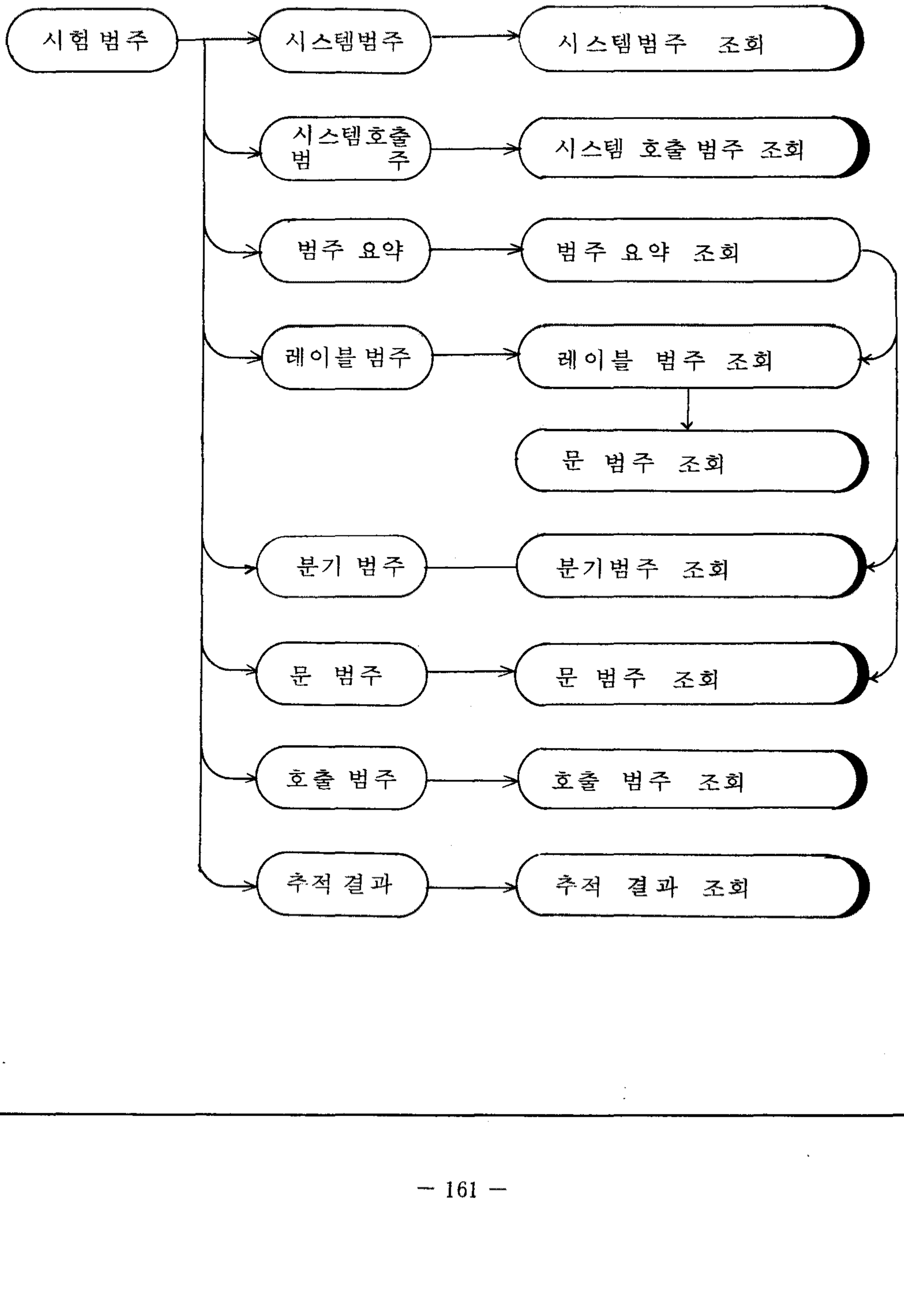




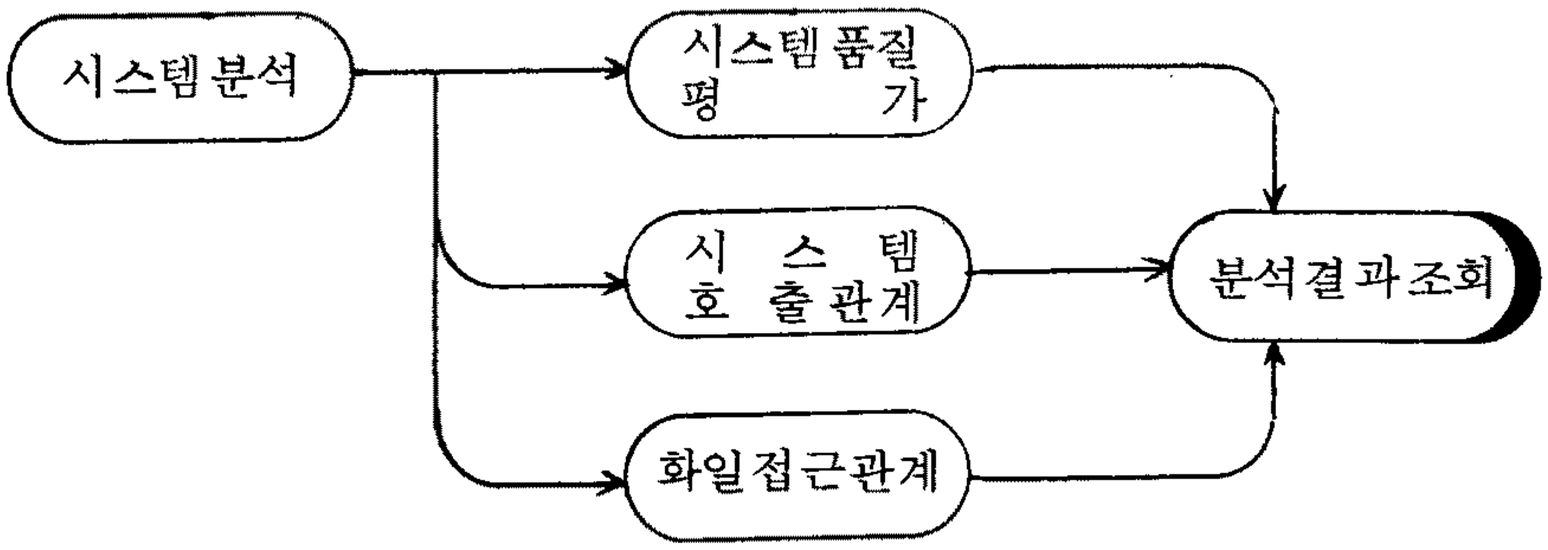
시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	화면 제어 흐름도	작 성 자		작성일	1991. . .
문서제목	시험오류, 시험로그				

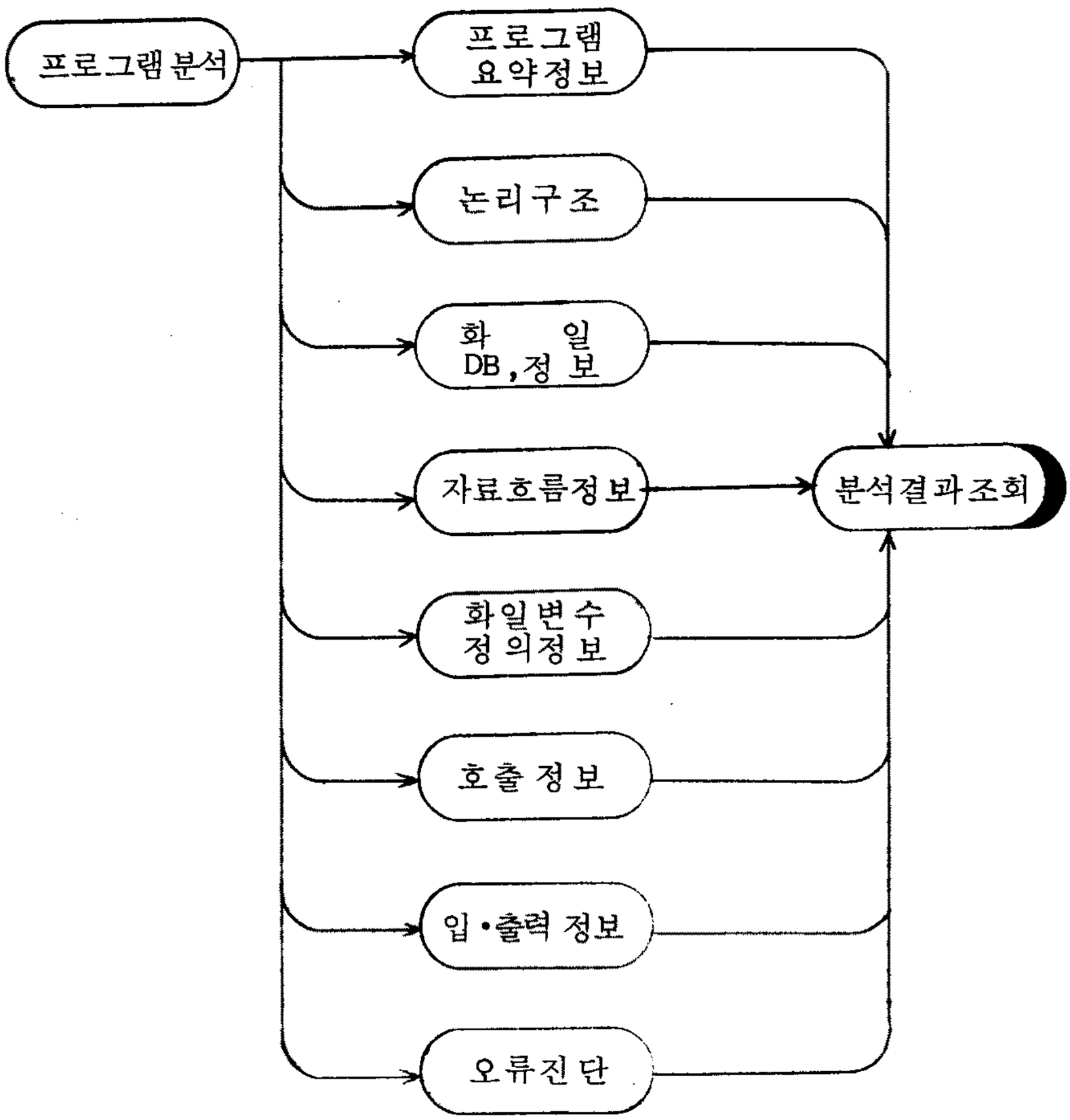


시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	화면 제어 흐름도	작 성 자		작성일	1991. . .
문서제목	시험 범주				

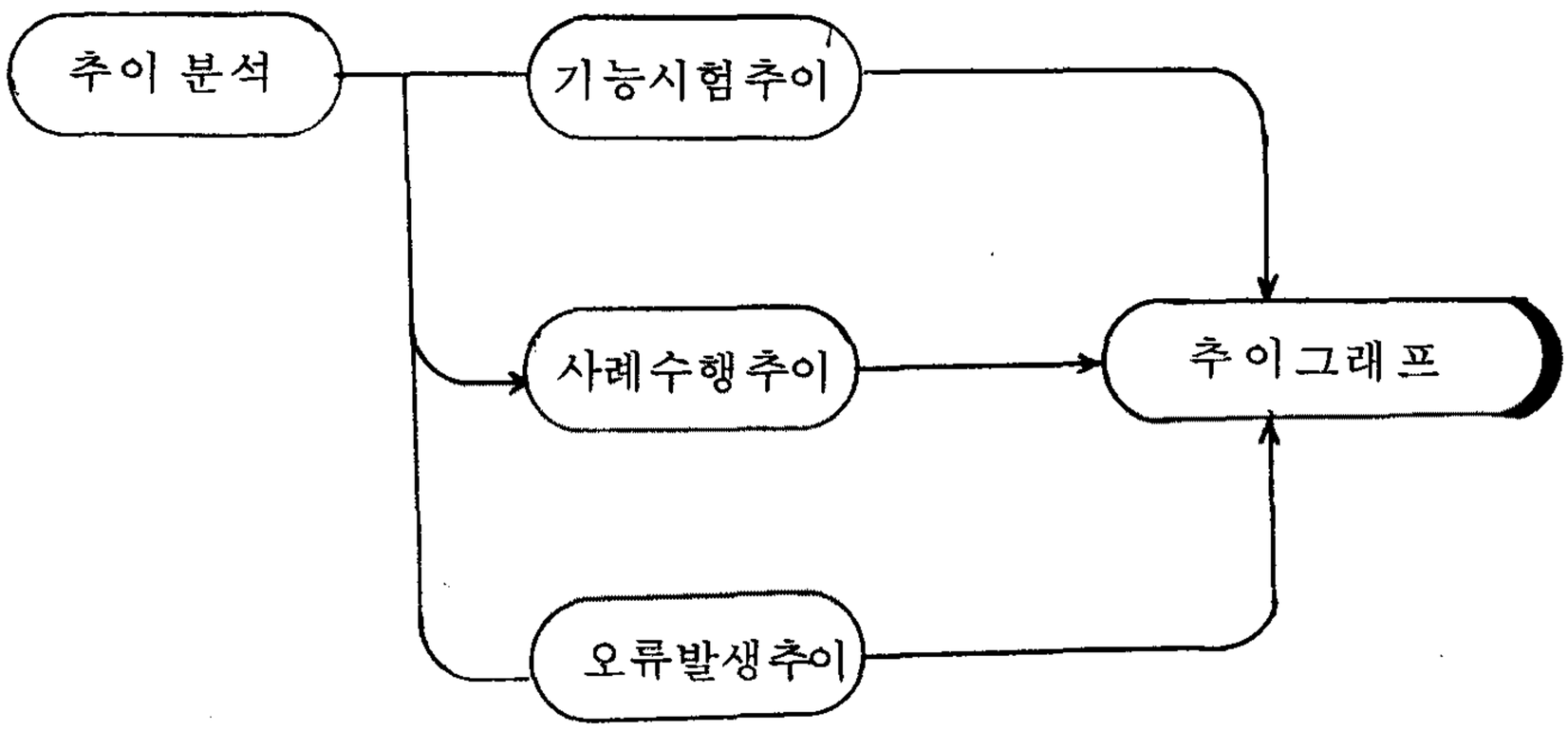


시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	화면 제어 흐름도	작 성 자		작성일	1991. . .
문서제목	시스템 분석, 프로그램 분석				





시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	화면 제어 흐름도	작 성 자		작성일	1991. . .
문서제목	추이 분석				



2. 입력 목록

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	입력 목록	작 성 자		작성일	1990. . .
문서제목					

번 호	화 면 이 름	입 력 자 료 요 소 이 름	출 력 자 료 요 소 이 름
1.	시험 기능	시스템	
		시험단계	
		시험항목	
		기능번호	
		시험기능명	
2.	시험 사례	시스템	
		사례번호	
		시험항목	
		목적	
		작성자	
		작성일자	
		입력데이타	
		예상출력	
3.	시험 절차	시스템	
		시험사례	
		시험절차	
		준비항목	
		수행절차	
4.	시험 오류	시스템	
		시험단계	
		TIR번호	

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	입력 목록	작 성 자		작성일	1990. . .
문서제목					

번 호	화 면 이 름	입 력 자 료 요 소 이 름	출 력 자 료 요 소 이 름
		사례번호	
		작성자	
		일자	
		시간	
		심각도분류	
		해결여부	
		오류상황	
		오류원인	
		원인프로그램	
		원인단계	
		책임조직	
		해결방안	
		관련모듈	
5.	시험 로그	시스템	
		시험단계	
		로그번호	
		수행팀	
		수행인원	
		작성자	
		작성일	
		수행시간	
		시간	
		발생상황	
		조치상황	

3. 출력 목록

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	출력 목록	작 성 자		작성일	1990. . .
문서제목					

번 호	화 면 이 름	입 력 자 료 요 소 이 름	출 력 자 료 요 소 이 름
1.	시험사례진행	시스템	계획된 수
			작성된 수
			작성된 수(%)
			수행된 수(%)
			사례번호
			목적
			작성상황
			시험절차
			수행횟수
			오류발생수
			해결한수
2.	시험로그상황	시스템	시험로그
		시험단계	일자
			작성자
3.	시험로그요약보고서	시스템	일자
		시험단계	수행사례수
		기간	TIR작성수
			시험인원
			시험시간
4.	시험오류분석	시험단계	
		기간	
	시스템별		시스템

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	출력 목록	작 성 자		작성일	1990. . .
문서제목					

번 호	화 면 이 름	입 력 자 료 요 소 이 름	출 력 자 료 요 소 이 름
			횟수
			도표
			백분율
	책임조직별		책임조직
			횟수
			그래프
			백분율
	개인별		작성자
			횟수
			도표
			백분율
	원인단계별		원인단계
			횟수
			도표
			백분율
5.	범주요악	구분	레이블 갯수
		시스템	레이블 수행된수
		프로그램	레이블 수행안된수
		시험사례	레이블범주(%)
		수행시간	분기 갯수
			분기 수행된수
			분기 수행안된수
			분기 범주(%)
			문 갯수
			문 수행된수

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	출력 목록	작 성 자		작성일	1990. . .
문서제목					

번 호	화 면 이 름	입 력 자 료 요 소 이 름	출 력 자 료 요 소 이 름
			문 수행안된수
			문 범주(%)
6.	레이블 범주	구분	레이블수
		시스템	수행된수
		프로그램	수행안된수
		시험사례	범주(%)
		수행시간	라인
			레이블
			횟수
			도표
			비고
7.	호출 범주	구분	호출 수
		시스템	수행된수
		프로그램	수행안된수
		시험사례	범주(%)
		수행시간	라인
			호출
			횟수
			도표
			비고
8.	분기 범주	구분	분기 수
		시스템	수행된수

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	출력 목록	작 성 자		작성일	1990. . .
문서제목					

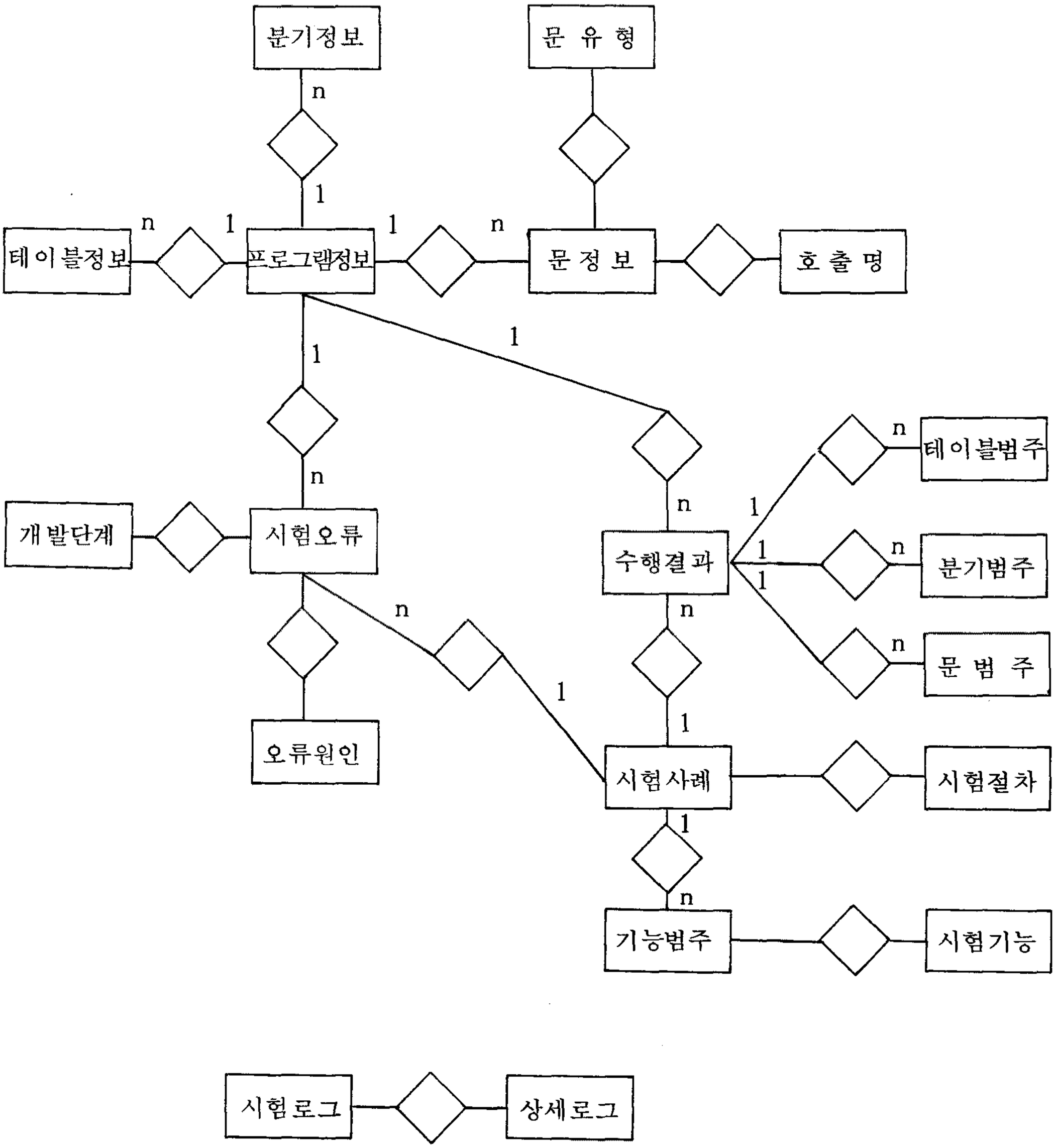
번 호	화 면 이 름	입 력 자 료 요 소 이 름	출 력 자 료 요 소 이 름
		프로그램	수행안된수
		시험사례	범주(%)
		수행시간	라인
			조건
			횟수
			도표
			비고
9.	문 범주	구분	문 수
		시스템	수행된수
		프로그램	수행안된수
		시험사례	범주(%)
		수행시간	라인
			문
			횟수
			도표
			비고
10.	시스템 범주	시스템	프로그램수
		수행시간	수행된수
			수행안된수
			범주(%)
			라인
			프로그램
			횟수
			도표

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	출력 목록	작 성 자		작성일	1990. . .
문서제목					

번 호	화 면 이 름	입 력 자 료 요 소 이 름	출 력 자 료 요 소 이 름
			비고
11.	호출 범주	시스템	프로그램수
		수행시간	수행된수
			수행안된수
			범주(%)
			라인
			프로그램
			호출
			횟수
			도표
			비고
12.	프로그램수행추적	시스템	라인
		프로그램	파라그랩
		시험사례	
		수행시간	

제 3 절 데이타베이스 설계

1 . 자료 요소 관계도



2. 자료 저장 목록

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	수행범주 테이블				

1. 저장자료명 : 수행결과

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	system_name	시스템명	시험대상 시스템명	8	Char
2	pgm_name	프로그램명	시험 프로그램	8	Char
3	case_id	사례id	시험사례번호	4	Char
4	run_time	수행시간	시험수행 시작시간		Time
5	run_no	수행번호	시험번호(기본키 일련본호)		Smallint

3. 저장자료의 색인

(1) 기본키 시스템명 + 프로그램명 + 사례id

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	수행범주 테이블				

1. 저장자료명 : 레이블 범주

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	run_no	수행번호	시험 번호		Smallint
2	label_no	레이블번호	프로그램내의 레이블 일련번호		Smallint
3	run_freq	수행횟수	레이블 수행 횟수		Smalint

3. 저장자료의 색인

(1) 기본키 수행번호

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	수행범주 테이블				

1. 저장자료명 : 분기 범주

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	run_no	수행번호	시험 번호		Smallint
2	branch_no	분기번호	프로그램내의 분기 일련번호		Smallint
3	run_freg	수행횟수	분기 수행 횟수		Smalint

3. 저장자료의 색인

(1) 기본키 수행번호

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. : .
문서제목	수행범주 테이블				

1. 저장자료명 : 문 범주

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	run_no	수행번호	시험 번호		Smallint
2	statement_no	문번호	프로그램내의 문 일련번호		Smallint
3	run_freg	수행횟수	문의 수행 횟수		Smalint

3. 저장자료의 색인

(1) 기본키 수행번호

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	정적정보 테이블				

1. 저장자료명 : 프로그램 정보

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	system_name	시스템명	시험대상 시스템명	8	Char
2	pgm_name	프로그램명	시험 프로그램	8	Char
3	author	작성자	프로그램 작성자	8	Char
4	pgm_function	프로그램기능	시험프로그램의 기능	20	Char
5	program_no	프로그램번호	시험프로그램 일련번호		Smallint

3. 저장자료의 색인

(1) 기본키 시스템명 + 프로그램명

(2) 선택키 프로그램번호

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	정적정보 테이블				

1. 저장자료명 : 레이블 정보

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	program_no	프로그램번호	시험프로그램의 번호		Smallint
2	label_no	레이블번호	프로그램내의 레이블일련번호		Smallint
3	label_type	레이블유형	디비전,섹션,패러그래프 구분	1	Char
4	label_name	레이블명	시험프로그램 레이블명	30	Char
5	label_line	레이블라인	프로그램 내의 레이블위치		

3. 저장자료의 색인

(1) 기본키 프로그램번호 + 레이블번호

(2) 선택키 레이블유형

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	정적정보 테이블				

1. 저장자료명 : 분기 정보

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	program_no	프로그램번호	시험프로그램의 번호		Smallint
2	barach_no	분기번호	프로그램 분기일련번호		Smallint
3	branch_type	분기유형	True,False 경로 구분	1	Char
4	branch_line	분기라인	프로그램 내의 분기문위치		Smallint
5	label_no	레이블번호	분기가 속한 레이블번호		Smallint

3. 저장자료의 색인

(1) 기본키 프로그램번호 + 분기번호

(2) 선택키 분기유형 , 레이블번호

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. -. .
문서제목	정적정보 테이블				

1. 저장자료명 : 문 정보

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	program_no	프로그램번호	시험프로그램의 번호		Smallint
2	statement_no	문번호	프로그램의 문일련번호		Smallint
3	statement_type	문유형	문의 종류(ADD,DELETE,....)	1	Char
4	statement_line	문라인	프로그램내의 문위치		Smallint
5	label_no	레이블번호	문이 속한 레이블번호		Smallint

3. 저장자료의 색인

(1) 기본키 프로그램번호 + 문번호

(2) 선택키 문유형 , 레이블번호

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	정적정보 테이블				

1. 저장자료명 : 호출명

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	program_no	프로그램번호	시험프로그램의 번호		Smallint
2	statement_no	문번호	호출문 일련번호		Smallint
3	called_pgm_name	호출명	호출되는 부 프로그램명	8	Char

3. 저장자료의 색인

(1) 기본키 프로그램번호

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	정적정보 테이블				

1. 저장자료명 : 문유형

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	statement_type	문유형	시험프로그램의 문유형	1	Char
2	statement_name	문명	ADD,IF 등 문명	20	Char

3. 저장자료의 색인

(1) 기본키 문유형

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. : .
문서제목	기능,사례 테이블				

1. 저장자료명 : 시험기능

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	system_name	시스템명	시험대상 시스템명	8	Char
2	phase_no	단계번호	시험단계번호		Smallint
3	test_item	시험항목명	모듈,프로그램명등	8	Char
4	feature_no	기능번호	기능 번호	4	Char
5	feature_name	기능명	항목내의 시험대상기능	30	Char

3. 저장자료의 색인

(1) 기본키 시스템명 + 단계번호 + 시험항목명

(2) 선택키 기능번호

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	기능,사례 테이블				

1. 저장자료명 : 시험사례

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	system_name	시스템명	시험대상 시스템명	8	Char
2	test_item	시험항목명	시험기능들을 포함하는항목	8	Char
3	case_id	사례id	시험사례번호	4	Char
4	case_purpose	목적	시험사례의 목적	15	Char
5	author	작성자	시험사례 작성자	10	Char
6	date_written	작성일	시험사례의 작성일자		Date
7	input_data1	입력데이타1	시험 입력데이타	60	Char
	input_data2	입력데이타2		60	Char
	input_data3	입력데이타3		60	Char
	input_data4	입력데이타4		60	Char
8	expected_result1	예상결과1	입력데이타에 대한 예상결과	60	Char
	expected_result2	예상결과2		60	Char
	expected_result3	예상결과3		60	Char
	expected_result4	예상결과4		60	Char

3. 저장자료의 색인

(1) 기본키 시스템명 + 사례id + 시험항목명

(2) 선택키 절차번호

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	기능,사례 테이블				

1. 저장자료명 : 시험절차

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	run_proc_no	절차번호	시험 절차번호		Smallint
2	setup	준비사항	시험 시작전 준비사항	250	Char
3	run_proc	수행절차	시험수행에 필요한 절차 및 방법	250	Char

3. 저장자료의 색인

(1) 기본키 절차번호

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	기능,사례 테이블				

1. 저장자료명 : 기능범주

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	system_name	시스템명	시험대상 시스템명	8	Char
2	case_id	사례id	시험사례번호	4	Char
3	feature_no	기능번호	사례가 시험할수 있는 기능		Smallint

3. 저장자료의 색인

(1) 기본키 시스템명 + 사례id

(2) 선택키 기능번호

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	오류,로그 테이블				

1. 저장자료명 : 시험오류

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	system_name	시스템명	시험대상 시스템명	8	Char
2	TIR_no	TIR번호	문제 발생 보고서 번호		Smallint
3	case_id	사례id	시험사례번호	4	Char
4	phase_no	단계번호	시험단계번호		Smallint
5	author	작성자	시험오류를 기록한사람	10	Char
6	date_written	작성일	시험오류를 작성한 일자		Date
7	severity	심각도	오류정도	1	Char
8	error_status	오류상황	오류발생시의 시스템상황	30	Char
9	error_cause_no	오류원인번호	오류유형 번호		Smallint
10	error_phase_no	오류단계번호	오류가 만들어진 개발단계		Smallint
11	resposible_org	책임조직	시스템을 개발한 조직	5	Char
12	resolution_ method	해결방안	발생한 오류의 해결방안	250	Char
13	resolve_flag	해결여부	오류의 해결여부	1	Char
14	error_no	오류번호	오류 일련번호		Smallint

3. 저장자료의 색인

(1) 기본키 시스템명 + TIR번호

(2) 선택키 오류번호

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	오류,로그 테이블				

1. 저장자료명 : 오류범주

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	error_no	오류번호	오류 일련번호		Smallint
2	related_module	관련모듈명	오류에 관련된 모듈명	12	Char

3. 저장자료의 색인

(1) 기본키 오류번호

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	오류,로그 테이블				

1. 저장자료명 : 오류원인

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	error_cause_no	오류원인번호	유형 번호		Smallint
2	error_cause_name	오류원인명	오류유형 명	10	Char

3. 저장자료의 색인

(1) 기본키 오류번호

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	오류,로그 테이블				

1. 저장자료명 : 개발단계

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	phase_no	단계번호	시험단계번호		Smallint
2	phase_name	단계명	시험단계명	10	Char

3. 저장자료의 색인

(1) 기본키 단계번호

(2) 선택키

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	오류,로그 테이블				

1. 저장자료명 : 시험로그

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	system_name	시스템명	시험대상 시스템명	8	Char
2	phase_no	단계번호	시험단계번호		Smallint
3	log_no	로그번호	시험로그번호		Smallint
4	test_team	수행팀	시험사례를 수행한 팀	10	Char
5	num_tester	시험인원	시험에 참가한 인원수 * 시험시간		Smallint
6	author	작성자	시험로그를 작성한 사람	10	Char
6	date_written	작성일	시험로그의 작성일자		Date
7	hr_testing	시험시간	시험을 수행소요시간		Time
8	detail_log_no	상세로그번호	상세로그번호		Smallint

3. 저장자료의 색인

(1) 기본키 시스템명 + 단계번호 + 로그번호

(2) 선택키 상세로그번호

시 스 템	동적 시험 지원 도구	단위시스템			
문서유형	저장 자료 목록	작 성 자		작성일	1991. . .
문서제목	오류,로그 테이블				

1. 저장자료명 : 상세로그

2. 저장자료 구성요소

번 호	구성 요소 이름		정 의	길 이	유 형
1	detail_log_no	상세로그번호	상세로그번호		Smallint
2	event_time	사건시간	사건이 발생한 시간		Time
3	event_desc	사건상황	사건의 상황을 기술	20	Char
4	event_treat	사건조치	사건에 대한 조치사항	20	Char

3. 저장자료의 색인

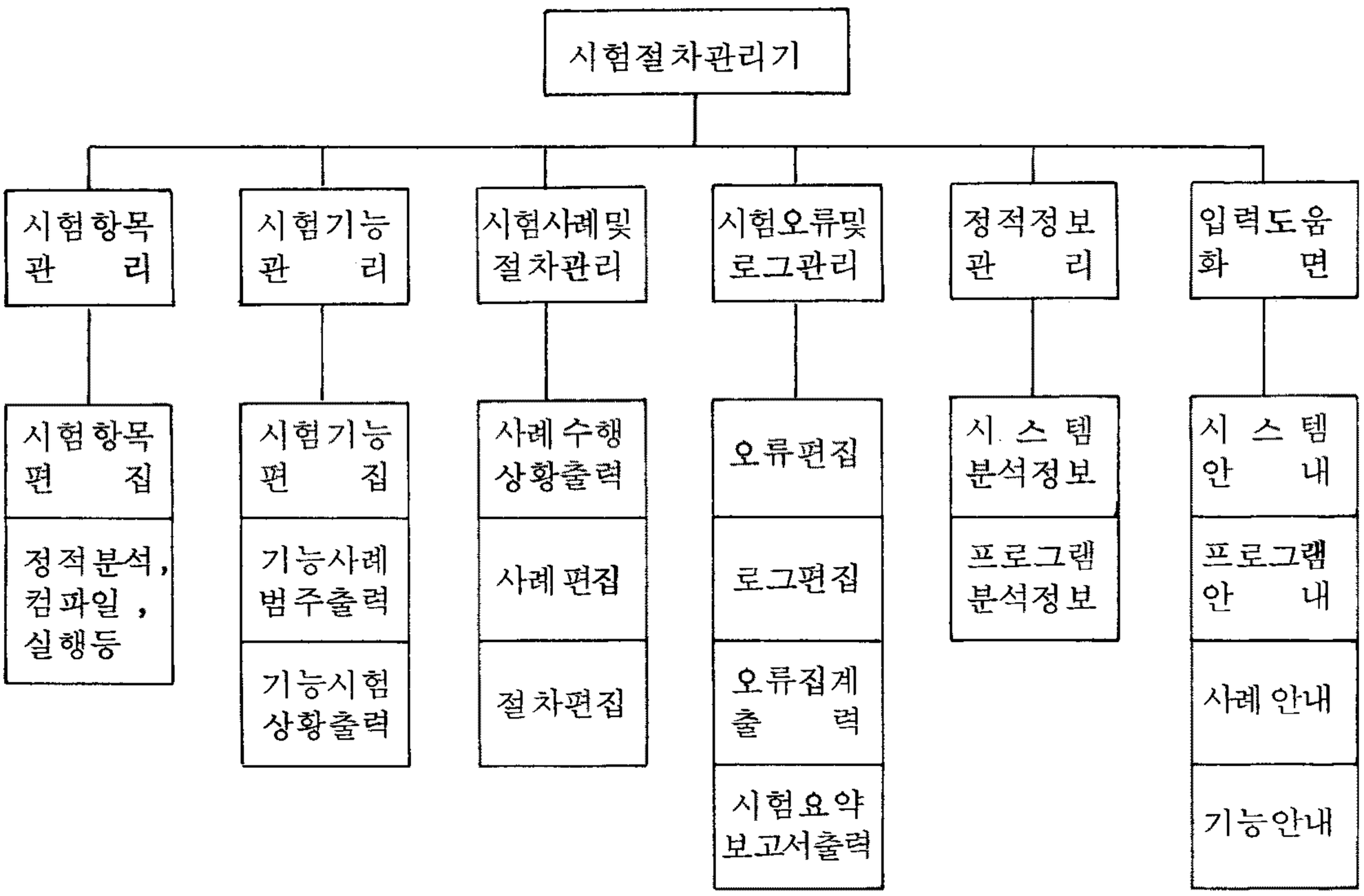
(1) 기본키 상세로그번호

(2) 선택키

제 4 절 기능별 구현 내용

1. 시험 절차 관리기

시험 절차 관리기는 시험 설계, 수행 및 결과 분석 작업 및 각 작업에서 작성되는 문서를 관리한다. 시험 절차 관리기의 구조는 아래와 같다.



시 스 템	동적 시험 지원 도구	단위시스템	시험 절차 관리기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	시험 항목 편집				

1. 기 능 : 시스템에 대한 시험 항목을 생성, 관리한다.

2. 입력 자료 : 시스템, 프로그램명, 작성자, 프로그램 기능

3. 출력 자료 : 시스템, 프로그램명, 작성자, 프로그램 기능

4. 처리 내용 :

가. 시스템명을 입력받는다.

나. 시스템에 해당되는 시험 항목을 등록, 수정, 삭제, 검색 한다.

다. 시스템에 대한 시험 항목들을 인쇄한다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 절차 관리기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	시험 기능 편집				

1. 기 능 : 시험 항목에 해당 되는 시험 기능을 생성, 관리 한다.

2. 입력 자료 : 시험 기능 (Features to be Tested)

3. 출력 자료 : 시스템명, 단계 번호, 시험 항목별 기능 번호 및 기능명

4. 처리 내용 :

가. 시스템, 단계 번호, 시험 항목을 입력 받는다.

나. D/B의 프로그램 정보, 시험 단계 테이블에서 조건에 맞는 시스템명, 단계명, 항목명을 검색한다.

다. 시험 기능 테이블을 검색하여 해당하는 기능 번호, 기능명을 화면상에 스크롤이 되도록 보여준다.

라. 시스템, 단계 번호, 시험 항목에 대한 기능 번호, 기능명을 등록, 수정, 삭제, 검색한다.

마. 시스템 단계 번호, 시험 항목에 해당되는 기능 번호와 기능명을 인쇄한다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 절차 관리기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	기능 사례 범주 출력				

1. 기 능 : 시험 기능에 관련된 시험 사례를 보여준다.

2. 입력 자료 : 시스템명

3. 출력 자료 : 시험 기능 번호, 시험 기능명, 시험 사례

4. 처리 내용 :

가. 시스템을 입력받는다.

나. D/B 테이블인 프로그램 정보에서 시스템의 존재 유무를 검색한다.

다. D/B에 입력받은 시스템이 없는 경우에는 오류 메시지를 띄워준 후 다시 시스템을 입력 받도록 한다.

라. D/B에 입력 받은 시스템이 존재할 경우 해당되는 시스템에 대하여 기능 범주 테이블을 검색하여 기능명과 기능에 해당되는 시험 사례 목록을 보여준다.

마. 인쇄를 선택하면 내용을 Matrix 형태로 인쇄한다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 절차 관리기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	사례 수행 상황				

1 . 기 능 : 시험 사례의 진행 상황을 보여주고, 시험 사례, 시험 절차를 생성, 관리한다.

2 . 입력 자료 : 시스템명

3 . 출력 자료 : 계획된 수, 작성된 수, 수행된 수, 사례 번호, 사례 목적, 작성 상황, 시험 절차 유무, 수행 횟수, 오류 발생 수, 해결한 수

4 . 처리 내용 :

가. 시스템을 입력받는다.

나. D/B내의 시스템 존재 유무를 검색한다.

다. D/B테이블인 시험 사례를 검색하여 시험 사례 수를 누적하여 계획된 수, 작성된 수를 보여준다.

라. D/B테이블인 수행 결과를 검색하여 유일한 사례를 누적하여 수행된 수를 보여준다.

마. 시험 사례 테이블의 작성일자를 기준으로 검색하여 작성 상황을 보여준다.

바. 시험 절차 테이블을 검색하여 시험 절차 유무를 보여준다.

사. 시험 오류 테이블을 검색하여 오류 수와 해결 수를 누적하여 화면에 보여준다.

아. 누적한 내용을 화면에 보여 준 후 원하는 시험 사례, 시험 절차를 등록, 수정, 삭제, 검색한다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 절차 관리기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	시험 사례 편집				

1. 기 능 : 시스템, 시험 항목에 대한 시험 사례를 생성, 관리한다.

2. 입력 자료 : 시스템명, 시험 항목명, 사례 번호, 목적, 작성자, 작성일자, 입력 데이타, 예상 출력

3. 출력 자료 : 시스템명, 시험 항목명, 사례 번호, 목적, 작성자, 작성일자, 입력 데이타, 예상 출력

4. 처리 내용 :

가. 시스템명, 항목명, 사례 번호를 입력 받는다.

나. D/B의 프로그램 정보, 시험 사례 테이블을 검색하여 시스템명, 시험 항목명, 시험 사례의 존재 유무를 검색한다.

다. 시험 사례의 등록, 수정, 삭제, 검색을 한다.

라. 시험 사례 테이블 검색 내용을 인쇄한다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 절차 관리기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	시험 절차 편집				

1. 기 능 : 시험 사례에 대한 시험 절차를 관리한다.

2. 입력 자료 : 시스템, 시험 사례, 준비 항목, 수행 절차

3. 출력 자료 : 시스템, 시험 사례, 준비 항목, 수행 절차

4. 처리 내용 :

가. 시스템, 시험 사례를 입력 받는다.

나. D/B의 시험 사례 테이블을 입력 조건으로 검색한다.

다. 시험 사례를 수행하기 위해 준비할 사항과 수행 절차를 등록, 수정, 삭제, 검색 한다.

라. 시험 절차 테이블을 검색하여 시스템명, 시험 사례, 준비 항목, 수행 절차를 인쇄한다.

2. 계기 삽입기

가. 개 요

계기 삽입기는 2개의 서브시스템으로 나눌 수 있다.

첫번째 서브 시스템은 어휘분석기이고 두번째 서브시스템은 구문 분석기이다.

어휘분석기는 코볼의 원시코드를 입력으로 받아서 토큰을 생성하고 구문 분석기는 어휘분석기가 생성한 토큰을 받아서 토큰과 토큰의 관계를 파악하여 계기 삽입을 한다. 계기 삽입을 하기전에 필요한 정보들을 정적 정보 테이블에 저장한다.

코볼의 프로그램은 4개의 Division으로 구성되어 있다.

원시 코드를 계기 삽입하려면 DATA Division과 PROCEDURE Division에 대해서 고려해야 한다.

PROCEDURE Division의 레이블, 분기, 문, 호출에 대해 계기 삽입을 할 때 필요한 변수를 DATA Division에 변수로서 선언을 해야한다.

계기 삽입은 PROCEDURE Division에서만 이루어 지지만 필요한 변수들을 선언해야 하기 때문에 DATA Division과 관계가 있게 된다.

각 서브 시스템의 자세한 사항은 다음 절에서 다룬다.

나. 어휘분석기

코볼 원시 프로그램을 입력받아 토큰을 산출하는 서브 시스템이다

테스팅 도구의 동적 분석기에 맞는 스캐너를 만들기 위해 LEX 를 이용한다.

입력은 코볼의 원시코드이고 출력은 토큰들을 출력한다.

토큰을 생성하는 방법은 Unix 환경에서 LEX를 이용하여 스캐너를

생성한다. 생성된 스캐너를 이용하여 코볼 원시 코드를 읽는다.

스캐너는 여러가지 종류의 토큰들을 생성한다.

토큰들의 유형을 보면 상수, 연산자, 식별자, 예약어 등으로 구분할 수 있다. 스캐너는 코볼원시코드를 일련의 스트림으로 간주하여 토큰들을 산출한다. 스캐너가 원시코드를 읽으면서 예약어 테이블과 동사 테이블을 참조하여 식별자중에서 예약어와 동사를 구분한다. 식별자가 예약어도 아니고 동사도 아닌 경우는 일반 식별자로 간주한다.

원시 코드에서 동사 토큰이 발견될 때는 그 동사의 출현 위치, 동사 테이블 번호를 정적 정보 테이블에 저장한다. 그리고 구문 분석기에 토큰번호를 넘겨준다.

식별자(identifier)가 발견될 때에는 그 식별자가 예약어인지 동사인지를 확인하기 위하여 예약어 테이블과 동사 테이블을 탐색한다 예약어 테이블과 동사 테이블에 속하지 않은 식별자는 테이블 명, 변수명, section 명으로 간주된다. 상수, 연산자, 정수형, 실수형 등에 대해서 토큰 번호를 나타내준다. 상수, 연산자, 정수형등 자료들을 정적 정보 테이블에 저장하지 않는다.

(1) 예약어 테이블 (Reserved word table)

코볼의 예약어는 약 535가 되고, 예약어는 알파베트 순서로 정렬되어 있다고 가정한다. 식별자가 예약어 테이블에 존재하는지를 알기 위해서 2진 탐색 (Binary search) 방법으로 한다.

예약어 테이블은 다음과 같다.

(2) 동사 테이블 (Verb table)

코볼의 동사는 약 45개가 되고 동사도 예약어 테이블과 마찬가지로 알파베트 순서로 정렬되어 있다고 전제한다. 식별자가 동사인가를 검사할 때는 동사 테이블을 2진 탐색 (Binary search) 방법으로 한다.

동사 테이블은 다음과 같다.

01 ACCEPT
02 ADD
03 ALTER
04 CALL
05 CANCEL
06 CLOSE
07 COMPUTE
08 CONTINUE
09 DELETE
10 DISPLAY
11 DIVIDE
12 ENTER
13 ENTRY
14 EVALUATE
15 EXIT
16 GOBACK
17 GOTO
18 IF
19 INITIALIZE
20 INSPECT
21 MERGE
22 MOVE

23 MULTIPLY

24 MEXT

25 NOTE

26 OPEN

27 PERFORM

28 READ

29 RELEASE

30 RETURN

31 REWRITE

32 RUN

33 SEARCH

34 SEEK

35 SET

36 SORT

37 START

38 STOP

39 STRING

40 SUBTRACT

41 TRANSFORM

42 UNLOCK

43 UNSTRING

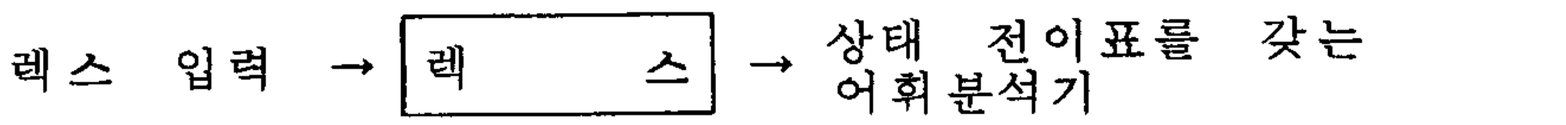
44 USE

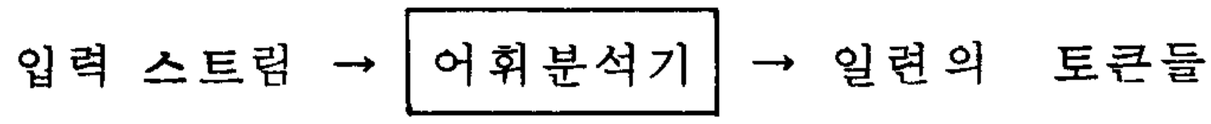
45 WRITE

(3) 렉스 (Lex)

렉스는 어휘분석기 생성기이다. 이것은 입력 스트림 (Stream) 에서 정규표현으로 표현된 토큰들을 찾아내는 프로그램을 작성하는데 유용한 도구이다. 렉스는 사용자가 정의한 정규표현과 수행 코드 (action) 를 입력으로 받아 일반 범용 언어로 쓰여진 프로그램을 출력한다.

이 프로그램은 입력 스트림에서 정규 표현에 해당되는 토큰을 찾았을 때 그에 결합된 수행코드를 수행한다.





렉스의 입력(Source)은 다음과 같이 세 부분으로 구성되어 있다.

〈정의 부분〉

%%

〈규칙 부분〉

%%

〈사용자 부프로그램 부분〉

〈정의 부분〉은 다음과 같은 형태를 갖는다.

```
% {
% }
D1 = R1
D2 = R2
   :
Dn = Rn
```

여기서 %{ 와 %} 사이에는 규칙 부분에서 사용하는 변수 및 상수를 선언한다. D_i는 이름이고, R_i는 D_i를 정의하는 정규 표현으

로 입력문자와 D_1 부터 D_{i-1} 에 나오는 이름으로 구성될 수 있다. 이 D_i 는 정의 부분과 규칙 부분에서 사용되는 정규 표현들에 대한 마크로 정의로 간주할 수 있다.

예를 들어, 다음과 같이 명칭을 정의할 수 있다.

```
%{  int  LE=1φ ;
%}
letter = [ a..z ]
digit  = [ 0..9 ]
identifier = letter (letter | digit)*
```

규칙 부분의 일반형태는 다음과 같다.

```
%%
P1    { A1 }
P2    { A2 }
  ⋮
Pn    { An }
```

여기서 %%는 정의 부분과 경계를 나타내는 기호이며, P_i는 패턴(pattern)으로 불리며 입력 스트림에 나오는 문자와 정의 부분의 이름들로 구성될 수 있는 정규표현이다. 이 패턴들로 입력 스트림이 나올 수 있는 모든 토큰들을 표현하며, Ai는 입력 스트림으로부터 일치되는 P_i를 찾았을 때, 어휘분석기가 해야 할 행동을 표현하는 수행코드로 C프로그램으로 기술된다.

```
%{
/*
* lexical analysis of COBOL
*/
# include <stdio.h>

enum {plus = 256, minus = 257, integer = 258, real = 259, strconst = 301,
      id = 300, divide = 302, exp = 303, equal = 304, comma = 305,
      semicolon = 306, period = 307, quote = 308, lp = 309, rp = 310,
      gt = 311, lt = 312};
%}
sign          [+-]
d             [0-9]
l             [A-Z]
h             [-]
dl            [0-9A-Z]
comment       [*/]
%%
{comment} {   printf ("comment start \n");
              while (input() != '\n')
                    ;
              if ((input() == '-') && (blk_no == 6))
                  while (input() != '\n')
                        ;
              printf ("\n");
              blk_no = 0;
              printf ("comment end \n");
          }
"+"       return plus;
"-"       return minus;
"/"       return divide;
"**"      return exp;
"="       return equal;
","       return comma;
";"       return semicolon;
"."       return period;
"'"       return quote;
"("       return lp;
")"       return rp;
">"       return gt;
"<"       return lt;
\"(\\.|[^\"])*\"                    return strconst;
"'"(\\.|[^\"])*"'"                  return strconst;
{sign}?{d}+                         return integer;
{sign}?{d}*"."{d}*                  return real;
{dl}+({h}{0,1}{dl}+)*               return id;
[ \t]                               ;
[\n]                                ;
.                                   return yytext[0];
%%
```

다. 구문 분석기

구문 분석기는 어휘분석기가 넘겨준 토큰 번호를 가지고 각 토큰에 대해 계기 삽입을 한다.

각 토큰에 대해 계기삽입을 위한 모듈은 다음과 같다.

동 사 토큰 — Instrument-Verb( )

레이블 토큰 — Instrument-label( )

분 기 토큰 — Instument-branch( )

호 출 토큰 — Instrument-call( )

구문 분석기는 어휘 분석기의 yylex( )로 부터 하나의 토큰을 받아서 계기 삽입을 한다.

동사의 토큰을 계기 삽입하기 전에 동사의 출현위치, 동사 테이블 번호 등을 저장한다.

동사로 시작하는 문장(Statement)의 끝은 다음 동사를 보는 경우, 레이블을 만나는 경우, if문, call문 경우가 된다.

레이블 토큰을 계기 삽입하기 전에 레이블의 출현 위치, 레이블일련번호, 레이블명을 정적 정보 테이블에 저장한다.

레이블 계기 삽입은 레이블이 수행된 횟수를 기억하는 변수와 레이블 일련번호를 삽입하기 위하여 코볼의 서브 프로그램의 형식으로 계기를 삽입한다. 레이블명을 계기 삽입하는 위치 레이블을 주사하고 Period "."을 보고 동사가 출현한 이전 라인에 계기를 삽입한다.

분기 토큰에 대해서 분기의 true와 false로 구분할 수 있다.

정적 정보 테이블에도 IF문의 true,false 각각에 대해 IF문의 출현번호, 프로그램 일련번호등을 저장한다.

호출 토큰(Call)에 대해서는 Call의 출현번호, 프로그램 일련 번호를 정적 정보 테이블에 저장하고 계기 삽입의 내용은 Call의 수행횟수를 기억하는 변수와 "Call" 토큰을 기억하는 변수를 삽입한다.

(1) Instrument_verb( )

동사 토큰을 계기 삽입하는 모듈이다.

동사들을 계기 삽입하기 전에 정적 정보 테이블에 동사의 출현위치, 동사 테이블번호등을 저장한다.

동사 토큰으로 시작하는 문장의 끝은 새로운 동사, IF, Call의 경우가 된다. 계기 삽입은 동사 토큰의 끝에서 이루어진다.

삽입되는 것은 동사의 수행 횟수를 저장하는 변수와 동사를 기억할 수 있는 변수를 집어 넣는다.

(2) Instrument-label( )

레이블 계기 삽입은 먼저 레이블 토큰을 인식해야 하는데 레이블 토큰을 인식하는 방법은 코볼 프로그램의 특성상 열 중심이기 때문에 7 열까지 blank검사를 한다.

즉 레이블은 8열부터 시작하는 식별자이고 식별자 다음 “ . ”이 유무, 다음 동사나 예약어의 시작으로 레이블명을 식별한다. 예외적인 DECLARATIVES 문과 END DECLARATIVES 문은 예외로 처리한다. DECLARATIVES 문은 레이블로 취급하지 않는다.

정적 정보 테이블에 레이블의 출현 위치, 프로그램 일련 번호 등을 저장한다. 레이블의 계기 삽입은 레이블의 수행 횟수를 저장할 변수와 레이블 토큰을 기억하는 변수를 삽입한다.

(3) Instrument-branch( )

If문의 대해서 계기 삽입을 하는 모듈이다.

if문은 단순 if와 내포(nested) if로 구분할 수 있다.

단순 if인 경우에는 조건식이 끝난 다음 true에 대한 계기 삽입을 하는데, 계기 삽입을 할 때는 동사 토큰이 시작되기 전에 계

기 삽입을 한다. false에 대해 계기를 삽입하려면 예약어 else 다음에 동사 토큰이 시작되기 전에 삽입한다. else 토큰이 없는 경우는 계기를 삽입하지 않는다.

내포 IF(nested if)인 경우에는 IF문이 두번 이상 나오는 경우이다.

첫번째 IF가 시작되고 조건이 끝난 다음 동사 token이 시작되기 전에 단순 IF의 true와 같은 계기 삽입을 하고 스텍에 출현된 IF와 내포 IF의 일련번호를 push한다. 스텍을 사용하는 이유는 IF와 연관된 else를 찾아 계기 삽입하기 위해서이다. IF의 true에 대해 계기 삽입이 끝나고 else 토큰이 시작되면 스택의 한 내용을 pop한다.

false에 계기 삽입을 동사 토큰이 시작되기 전에 삽입한다.

else 토큰을 전부 주사하면 스택을 완전히 빈 상태로 한다. 정적 테이블 정보는 IF의 True, false의 출현위치와 프로그램 일련번호, IF문의 일련번호등을 저장한다.

계기 삽입의 내용은 IF의 각 True, false의 수행 횟수를 저장할 수 있는 변수와 IF의 출현 위치를 기억하는 변수를 삽입한다.

(4) Instrument-Call( )(Call계기 삽입)

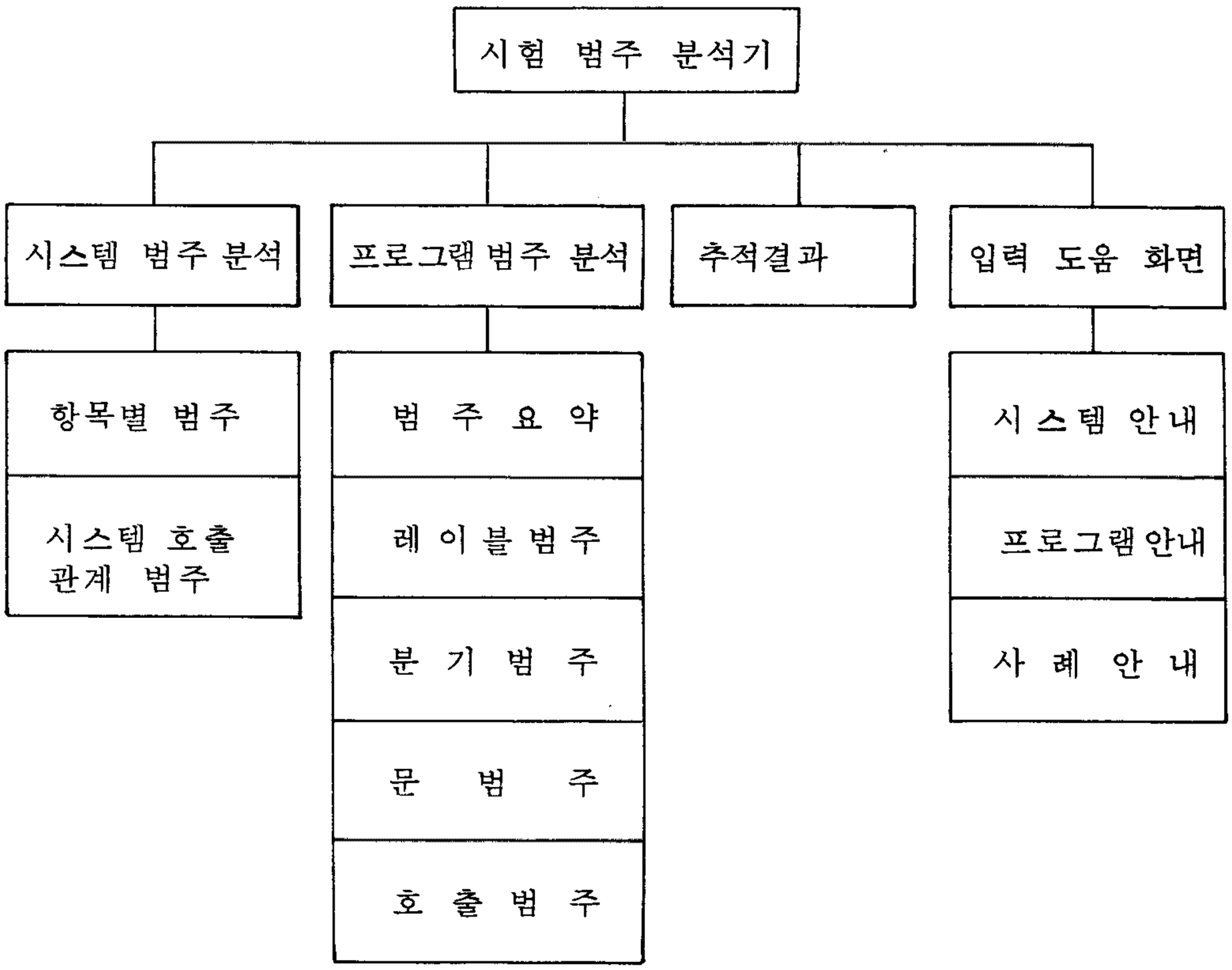
"Call"토큰을 보고 그 다음 토큰이 호출되는 프로그램명이 된다. 프로그램 일련번호, Call의 출현번호, 호출되는 프로그램명을 정적 테이블에 저장한다.

"Call"토큰을 계기 삽입할 때는 "Call"문 전체를 보고 그 다음 문이 시작될 때 삽입한다.

Call 토큰의 계기 삽입은 Call의 수행 횟수를 저장할 변수와 Call 토큰을 기억하는 변수를 삽입한다.

3. 시험 범주 분석기

시험 범주 분석기는 시험 사례 수행 결과를 분석하여 시험, 미시험 부분을 출력하고 시험된 부분의 전체에 대한 범주를 계산한다. 각 범주 결과는 관심 부분을 선택하여 상세 범주로 전이가 가능하다. 시험 범주 분석기의 구조는 아래와 같다.



시 스 템	동적 시험 지원 도구	단위시스템	시험 범주 분석기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	범주 요약				

1. 기 능 : 시험된 시스템, 프로그램, 시험 사례의 테이블, 분기, 문 등에 대한 갯수, 수행된 수, 수행안된 수, %범주를 분석할 수 있는 정보를 제공한다.

2. 입력 자료 : 수행 정보(수행 결과, 레이블 범주, 분기 범주, 문 범주)

3. 출력 자료 : 범주 분석, 수행 추적

4. 처리 내용 :

가. 시스템을 입력 받는다.

입력받은 시스템을 프로그램 정보 Table에 등록된 시스템명과 비교하여 등록되어 있지 않으면 에러 메세지를 나타내고 시스템을 재입력 받는다.

나. 프로그램을 입력 받아

프로그램 정보 Table에 등록된 프로그램과 비교하여 존재하지 않으면 에러 메세지를 나타내고 프로그램을 재입력 받는다.

다. 구분이 1(급회)인 경우에 한하여 시험 사례를 입력받는데 입력받은 내용과 수행 결과 Table을 비교하여 존재하지 않으면 에러 메세지를 나타내고 시험 사례를 재입력 받는다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 범주 분석기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	범주 요약				

라. 입력받은 시스템, 프로그램을 참고하여 프로그램 정보 Table의 프로그램 번호를 찾아낸다.

마. 프로그램 번호와 맞는 레이블 정보 Table의 레이블 수를 계산하여 레이블 갯수를 나타낸다. 같은 방법으로 분기 정보 Table, 문 정보 Table에서 분기 갯수, 문 갯수를 계산하여 나타낸다.

바. 시스템, 프로그램, 시험 사례를 참조하여 수행 결과 Table의 수행 번호를 찾아낸다.

사. 수행 번호를 참조하여 해당되는 레이블 범주 Table, 분기 범주 Table, 문 범주 Table의 수행 횟수가 0 (Zero)이 아닌 것을 찾아 누적 계산하여 각각의 수행된 수를 나타낸다.

아. 전체 갯수에서 수행된 수를 차감하여 수행안된 수를 나타낸다.

자. 선택된 기능을 호출하여 상세한 정보를 나타낸다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 범주 분석기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	레이블 범주				

1. 기 능 : 시험된 시스템, 프로그램, 시험 사례의 레이블에 대한 레이블 수, 수행된 수, 수행안된 수, %범주, 레이블별 수행 횟수를 분석할 수 있는 정보를 제공한다.

2. 입력 자료 : 수행 정보(레이블)

3. 출력 자료 : 범주 분석, 수행 추적

4. 처리 내용 :

가. 프로그램 정보 Table을 참조하여 시스템, 프로그램을 입력받는다.

나. 구분이 1(금회)인 경우 수행 결과 Table을 참조하여 시험사례를 입력받는다.

다. 프로그램 정보 Table과 레이블 정보 Table을 검색하여 레이블 수를 계산한다.

라. 수행 결과 Table과 레이블 범주 Table을 검색하여 수행된 수를 계산한다.

마. 선택된 레이블에 대한 정보를 가지고 문 범주 기능을 호출한다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 범주 분석기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	프로그램 호출 범주				

1. 기 능 : 시험된 시스템, 프로그램, 시험 사례의 호출 프로그램에 대한 호출 수, 수행된 수, 수행안된 수, %범주, 호출별 수행 횟수를 분석할 수 있는 정보를 제공한다.

2. 입력 자료 : 수행 정보 (수행 결과, 호출 범주)

3. 출력 자료 : 호출 범주 분석, 수행 추적

4. 처리 내용 :

가. 프로그램 정보 Table을 참조하여 시스템, 프로그램을 입력받는다.

나. 구분이 1 (금회) 인 경우 수행 결과 Table을 참조하여 시험 사례를 입력 받는다.

다. 프로그램 정보 Table과 호출명 Table을 검색하여 호출 수를 계산한다.

라. 수행 결과 Table, 문 범주 Table, 호출명 Table을 검색하여 수행된 수를 계산한다.

마. 검색된 호출명에 대한 사항을 나타낸다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 범주 분석기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	분기 범주				

1. 기 능 : 시험된 시스템, 프로그램, 시험 사례의 프로그램별 분기에 대한 분기 수, 수행된 수, 수행안된 수, %범주, 레이블별 분기의 수행 횟수를 분석할 수 있는 정보를 제공한다.

2. 입력 자료 : 수행정보 (수행결과, 분기범주), 정적정보 (분기정보)

3. 출력 자료 : 분기범주 분석, 수행추적

4. 처리 내용 :

가. 프로그램 정보 Table을 참조하여 시스템, 프로그램을 입력 받는다.

나. 구분이 1 (금회)인 경우 수행 결과 Table을 참조하여 시험 사례를 입력 받는다.

다. 프로그램 정보 Table과 분기 정보 Table을 검색하여 분기 수를 계산한다.

라. 수행 결과 Table, 분기 범주 Table, 분기 정보 Table, 레이블 정보 Table을 검색하여 수행된 수를 계산한다.

마. 검색된 분기 정보에 대한 사항을 레이블별로 나타낸다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 범주 분석기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	문 범주				

1. 기 능 : 시험된 시스템, 프로그램, 시험 사례의 프로그램별 문에 대한 문 수, 수행된 수, 수행안된 수, % 범주, 레이블별 문의 수행 횟수를 분석할 수 있는 정보를 제공한다.

2. 입력 자료 : 수행 정보(수행 결과, 문 범주), 정적 정보(문 정보, 문 유형)

3. 출력 자료 : 문 범주 분석, 수행 추적

4. 처리 내용 ;

가. 프로그램 정보 Table을 참조하여 시스템, 프로그램을 입력 받는다.

나. 구분이 1(금회)인 경우 수행결과 Table을 참조하여 시험 사례를 입력 받는다.

다. 프로그램 정보 Table과 문 정보 Table을 검색하여 문 수를 계산한다.

라. 수행 결과 Table, 문 범주 Table, 문 정보 Table을 검색하여 수행된 수를 계산한다.

마. 검색된 문에 대한 사항을 레이블 별로 나타낸다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 범주 분석기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	시스템 범주				

1. 기 능 : 시험된 시스템의 프로그램 종류, 프로그램 갯수, 수행된 수, 수행안된 수, %범주를 분석할 수 있는 정보를 제공한다.

2. 입력 자료 : 수행 정보 (수행결과), 정적정보 (프로그램 정보)

3. 출력 자료 : 시스템 범주

4. 처리 내용 :

가. 프로그램 정보를 참조하여 시스템을 입력 받는다.

나. 프로그램 정보 Table을 검색하여 프로그램 수를 계산한다.

다. 수행 결과 Table, 문 범주 Table, 프로그램 정보 Table을 검색하여 수행된 수를 계산한다.

라. 수행 결과 Table, 문 범주 Table을 검색하여 프로그램별 수행 횟수를 계산한다.

마. 검색된 시스템별 프로그램에 대한 수행 사항을 나타낸다.

시 스 템	동적 시험 지원 도구	단위시스템	시험 범주 분석기		
문서유형	모듈 명세서	작 성 자		작성일	1991. . .
문서제목	시스템 호출 범주				

1. 기 능 : 시험된 시스템의 프로그램 수, 수행된 수, 수행안된 수, 범주, 프로그램별 호출 수행 횟수를 분석할 수 있는 정보를 제공한다.

2. 입력 자료 : 수행 정보 (수행 결과, 문 범주)
정적 정보 (프로그램 정보, 문 정보, 호출명)

3. 출력 자료 : 호출범주분석

4. 처리 내용 :

가. 프로그램 정보 Table을 참조하여 시스템을 입력 받는다.

나. 프로그램 정보 Table을 검색하여 프로그램 수를 계산한다.

다. 수행 결과 Table, 문 범주 Table, 문 정보 Table, 호출명 Table을 검색하여 수행된 수를 계산한다.

라. 수행 결과 Table, 문 범주 Table, 문 정보 Table, 호출 명 Table을 검색하여 수행 횟수를 계산한다.

마. 검색된 프로그램별 호출명에 대한 수행 사항을 화면에 나타낸다.

4. 품질 측정 및 사례 설계 지원 시스템

가. 개요

품질 측정 시스템과 사례 설계 지원 시스템은 시범 단계에서 구축된 도구로 시험 지원 통합 시스템의 구성 요소이며, 현재 동적 시험 지원 시스템에 연결되어 있다. 품질 측정 시스템은 정적인 방법으로 원시 프로그램을 분석하여 품질 정보와 프로그램 추상화 정보(설계 정보)및 오류 정보를 추출한다.

나. 품질 측정 시스템

(1) 도구의 개발 방향

(개) 도구의 특징

품질 측정 시스템은 정적 분석 기법을 이용한다.

정적 분석이란 시스템을 수행하지 않고 시스템의 구조와 특성을 분석하는 것이다. 이 과정은 컴파일 단계의 확장이라고도 볼 수 있다. 정적 분석 도구는 70년대 중반부터 실험실 프로젝트로 또는 상품화 가능한 도구로 개발이 되기 시작하였으며 현재 코볼과 C언어 등에 대한 도구가 나와 있으나 그 수효는 많지 않다. 구현된 도구는 다음과 같이 기존 도구의 단점을 보완하고자 하였다.

- 현재 개발된 메트릭 측정 시스템이 복잡도의 측정에 집중되어 있는 점을 보완하여, 다양한 품질 기준에 대한 척도를 제공 한다.
- 메트릭 측정 과정의 정보를 이용하여, 오류 발생 가능성이 있는 부분을 상세히 지적한다.
- 출력의 해석이 용이하고, 조작을 간편하게 한다.

- 유지보수 과정에서 변경 관련 부분에 관한 정확한 정보를 제공한다.

(내) 도구의 기능 및 구조

도구의 주요 기능은 다음과 같다.

• 사용자 접속 기능

- 분석대상 프로그램의 선택

- 다수 프로그램의 일괄 분석

- 각 분석 결과의 조회

• 프로그램 정적 분석 기능

- 어휘 및 구문 분석

- 의미 수행

· 변수 및 상수 테이블의 유지

· 프로그램 제어 구조의 추출

· 정보 정의 참조 흐름 정보의 저장

· 기타 품질 측정 및 설계 복구를 위한 정적 정보의 추출 및 저장

• 소프트웨어 품질 측정 기능

- 정적 정보를 이용하여 다음과 같은 품질 기준을 측정한다.

정밀도, 완전도, 모듈화, 복잡도 등

- 복잡도 측정은 기존의 검증된 메트릭을 사용한다.

측정하는 복잡도는 다음과 같은 종류가 있다.

• Line of Code

• Effective Line of Code

• Cyclomatic Complexity

· Software Science

- 기타 다음과 같은 복잡도 메트릭을 제공한다.

· 화일의 갯수

· 패러그래프의 갯수

· 프로시듀어 디비젼의 라인 수

· 주석의 수

· 설계 복구 정보 제공 기능

시스템 단위로 제공되는 정보는 다음과 같다.

· 프로그램간의 호출관계

· 프로그램간의 화일 접근 관계

프로그램 단위로 제공되는 정보는 다음과 같다.

· 모듈 구조도

· 논리 구조

· 화일 명세

· 입 출력 정보

· 호출 정보

· 변수 및 화일의 자료 흐름

· 코딩 표준에 대한 검사

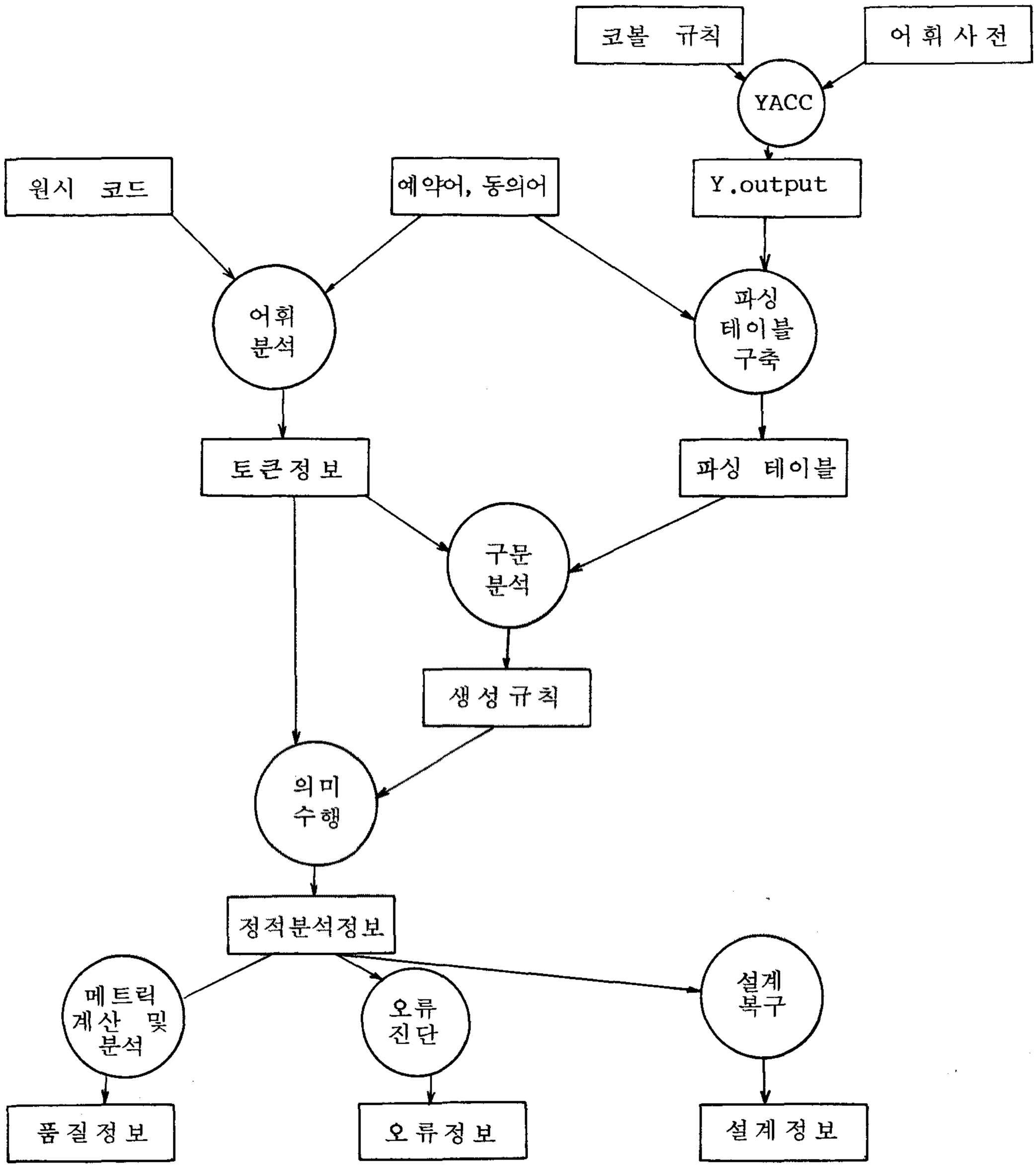
도구의 구조를 도식화 하면 [그림 4-1]와 같다.

(2) 도구의 설계 및 구현

㈎ 어휘 및 구문 분석

① 어휘분석기의 구현

정적 분석 과정의 첫번째 단계인 어휘분석 단계에서는 원시 프로그램을 읽어 토큰이라는 문법적 단위로 변환한다. 어휘 분석 단계에



[그림 4-1] 정적 분석기의 구조

서 구분하는 토큰의 종류는 다음과 같다.

1. 코볼 예약어
2. 숫자
3. 변수
4. 문자열
5. 기타 + - * / = 〉〈 . ,( )" '등

코볼의 경우 문자가 나타나는 열의 위치에 따라 토큰의 의미가 달라진다. 1 - 6열과 73 - 80열은 주석을 위한 것이며 7열은 주석라인과 페이지 배출 및 스트링과 문의 연결을 위한 기호를 기입한다. 프로시듀어 디비젼내에서는 8열 부터 시작되는 문자열은 레이블이고 12열부터는 문이 시작된다. 어휘 분석기에서는 코볼의 이와 같은 특성을 고려한다. 코볼의 픽쳐 심볼은 토큰을 구성하는 알파벳이 일반 토큰과 구분되어 '.','$', 문자, 숫자, 점 등이 사용될 수 있으며 이의 구분을 위해 어휘 분석 구현시 사용되는 오토마타를 픽쳐 심볼 인식용과 일반 토큰 인식용으로 구분하였다.

일반 토큰용 오토마타와 픽쳐 심볼용 오토마타 간의 전이는 'PICTURE'라는 토큰을 인식한 후에 일어난다. 어휘분석시 주석열의 문자는 무시하며 주석기호와 페이지 배출 기호가 7열에 나타나는 경우는 전체 라인을 무시한다. 어휘 분석을 하기 전에 코볼의 예약어를 정의하고 각 예약어에 번호를 부여하는데 예약어 테이블은 어휘 분석전에 정렬되어 준비한다. 어휘분석 단계에서 COPY토큰이 인식되면 COPY대상 화일을 디렉토리에서 탐색하여 대상화일이 완료될 때까지 어휘 분석을 한후 다시 원래의 프로그램을 주사한다.

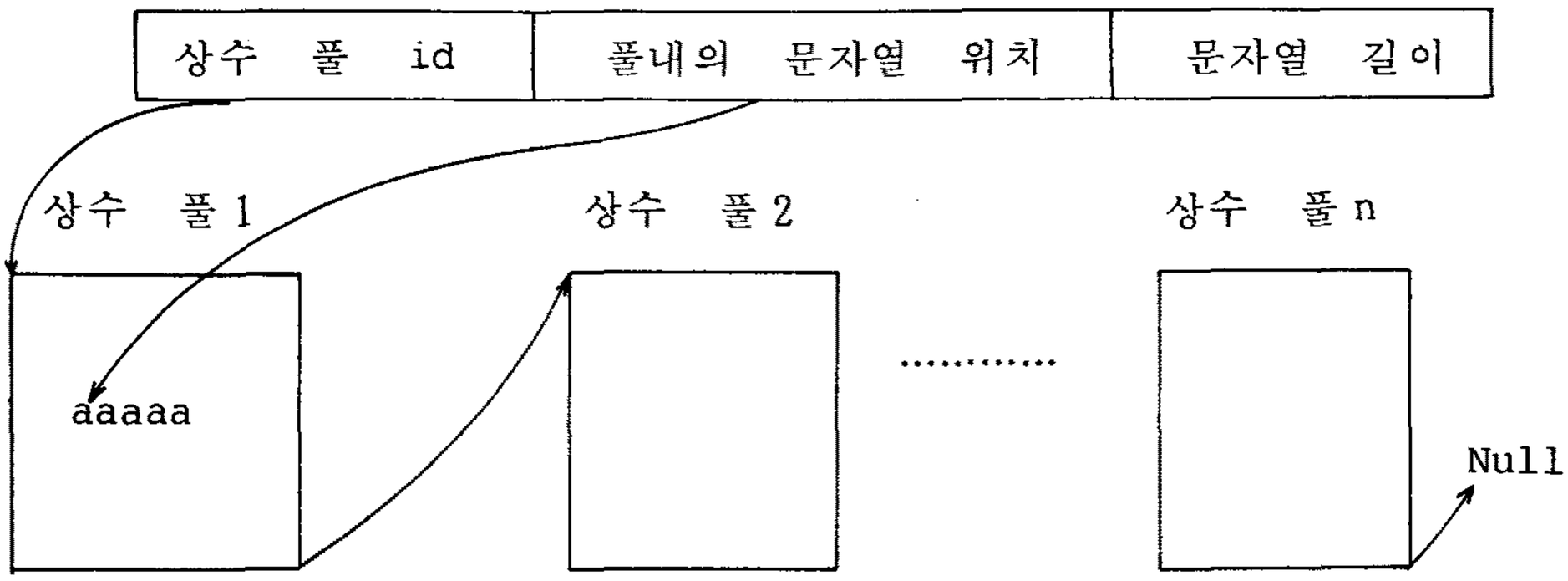
[그림 4-2]은 픽쳐 심볼을 인식하기 위한 유한 오토마타와 기타

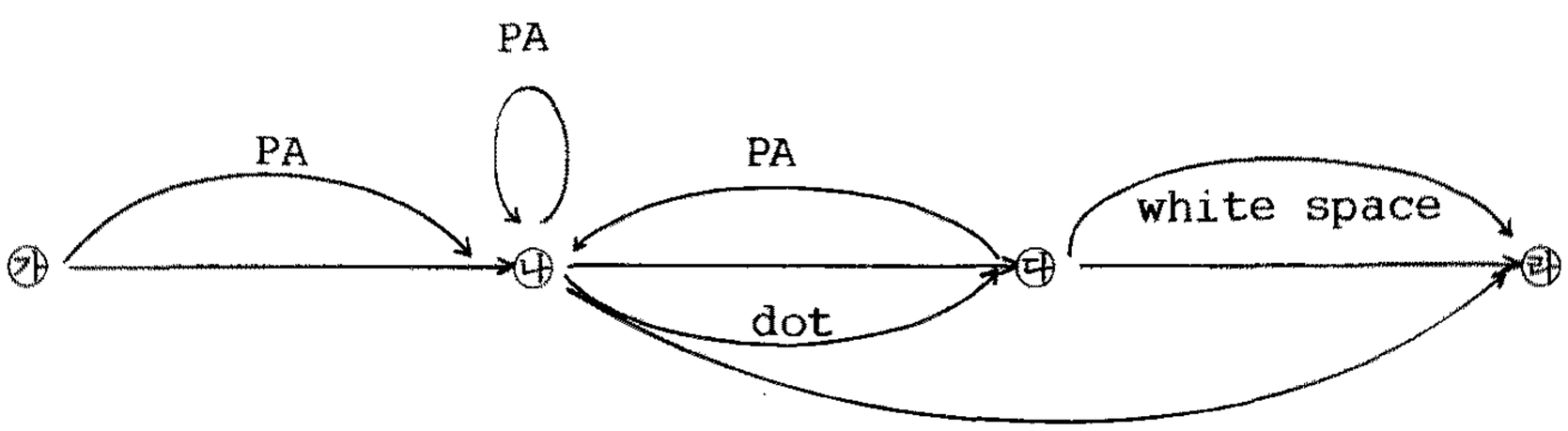
의 토큰을 인식하기 위한 유한 오토마타이다. 어휘 분석 후에 생성되는 토큰의 자료 구조는 다음과 같다.

```
typedef struct symbol_stack{
        int token_id;        예약어 번호
        int start_line;      토큰 시작위치
        int end_line;        토큰 완료 위치
        int token_usage;     변수, 상수, 초기상태
        variant_part vp;     변수, 상수, 초기상태에 따른 토큰 값
        }                    저장위치
typedef union{
        symbol_table symbolptr;   변수 테이블의 위치
        const_table constprt;     상수 테이블의 위치
        word_node *wordptr;       초기 문자열 보관 장소
        } variant_part
```

변수 테이블은 해상 함수를 이용하여 접근하는 linked list로 구현되었으며 코볼에서 허용되는 변수는 30자리로 해싱함수는 1번째, 15번째 문자의 아스키 값을 더하여 버켓의 위치를 구한다. 상수테이블은 각 132개의 문자를 보관할 수 있는 상수 풀의 id와 각 풀내에서의 문자열 시작 위치와 문자열 길이를 보관하는 linked list이며 문자열의 발생에 따라 풀을 계속 증가 시킨다.

상수 테이블 및 상수 풀의 구성



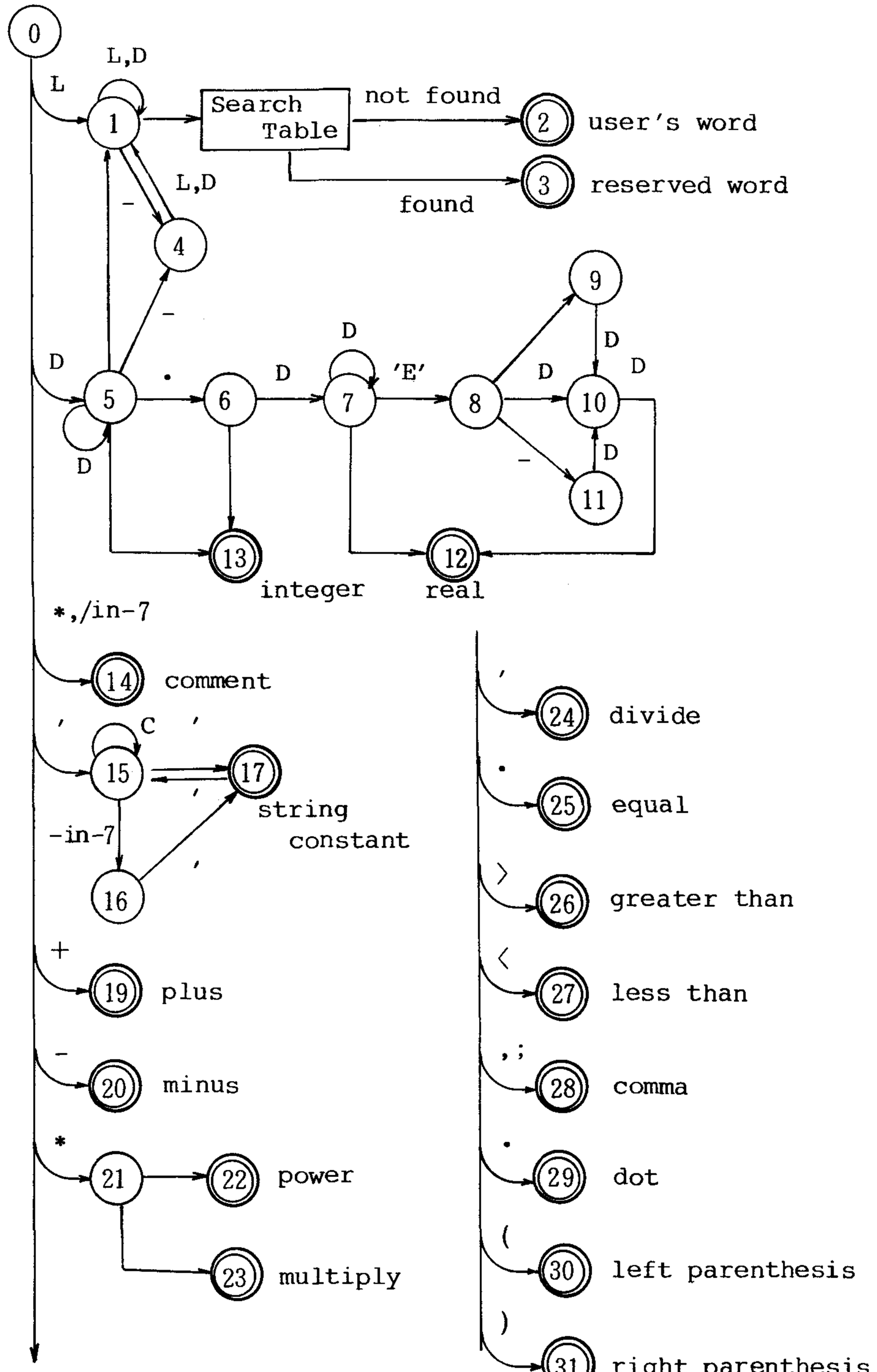


white space

PA : A - Z, 숫자, 괄호, 컴마, $, ₩, -, 등 픽쳐 심볼 알파벳

② 구문분석기의 구현

구문분석기의 파싱 테이블은 코볼 언어를 BNF형식으로 기술한 후 YACC를 사용하여 생성되는 Y.OUTPUT을 이용하였다. 코볼 생성 규칙 수가 많아 파싱테이블은 데이타 디비젼과 프로시듀어 디비젼을 구분하여 2개로 구축하였으며, 데이타 디비젼이 인식된 후 전반부용 파싱 테이블을 없애고, 후반부용 파싱 테이블을 로드하였다. 전반부와 후반부의 파싱 테이블 규모는 다음과 같다.



L : A-Z
D : 숫자
C : A-Z, 숫자, 특수기호
IN-7 : 행 7에 나타남

[그림 4-2] 일반 토큰을 인식하기 위한 오토마타

- 전반부

  • 생성 규칙 : 291

  • 터미날 심볼 : 294

  • 논 터미날 심볼 : 94

  • 상태의 수 : 517

- 후반부

  • 생성 규칙 : 547

  • 터미날 심볼 : 294

  • 논 터미날 심볼 : 158

  • 상태의 수 : 947

심볼 스택과, 상태 스택은 가변 길이로서 의미분석과정에서 속도를 향상 시키기 위하여 배열을 이용하였다. 파싱테이블은 sparse matrix로서 인덱스 테이블을 유지하여 SHIFT 테이블과 GOTO 테이블 검색 시작점을 유지하였다. 각 테이블은 2개의 정수로 구성되며 이러한 방식은 sparse matrix의 유지보다 기억장소를 20분의 1정도로 절약한다. 파싱 테이블의 구조는 다음과 같다.

• 인덱스 테이블

상태 번호	파싱 테이블 인덱스	GOTO 테이블 인덱스

• 규칙 테이블

규칙 번호	LHS의 논터미날 번호	규칙 길이

• 파싱 테이블

일련 번호	토큰 번호	동작

동작은 SHIFT 동작의 경우 상태번호가 REDUCE 동작의 경우 규칙번호가 음수로 저장되며 오류의 경우 0이 저장된다.

- GOTO 테이블

일련 번호	스택 상부의 논터미날	REDUCE후의 상태 번호

구문 분석의 과정을 자료 구조와 자료 구조에 의한 operation, 각 operation을 위한 조건 및 operation 결과로 나타내면 [그림 4-3]와 같다. 구문 분석의 최초 상태에서 커서는 0, 심볼 스택은 empty, 상태 스택은 스택의 원소가 1개(초기 상태값 0)를 가진다. 구문 분석 과정에서 하나의 규칙이 reduce가 되는 경우 규칙번호에 해당되는 의미수행 코드를 호출한다. 의미수행 코드에서는 규칙번호와 현재의 심볼 스택의 정보를 이용하여 정보를 추출한다. 심볼스택은 어휘 분석과정에서 설명한 토큰의 자료구조 배열이다.

- 테이블 검색 방법의 예

스택의 현재 상태가 3이고 현재 인식한 token id가 157이라면 인덱스 테이블의 일련번호 3의 파싱 테이블 인덱스 6을 추출하여 파싱 테이블 일련번호 6부터 아래로 내려가면서 token id 157 이 출현할 때까지 검색한다. 현재 인식된 token id 157이 파싱 테이블의 157과 일치하므로 SHIFT 행동이후에 스택에 PUSH할 다음 상태가 9라고 하는 것을 알 수 있다. 만약 파싱 테이블에 token id 157이 출현하지 않는다면 DEFAULT-1을 인식하여 0을 PUSH한다. 상태가 0인것은 에러에 해당되므로 파싱행동을 중단하고 여러 메세지를 출력한다.

[표 4-1] 문법 테이블 (1)

• 인덱스 테이블

일련번호 상태번호	파싱 테이블의 인덱스	GOTO 테이블의 인덱스
0	1	1
1	3	5
2	4	6
3	6	9
4	8	11
5	10	12
6	12	15
7	14	18
8	16	19
9	23	20
10	25	21
11	27	22
12	28	24
13	29	26
14	31	27
15	32	28
16	34	30
17	36	31

• RULE 테이블

일련번호 RULE 번호	LHS 의 NON- TERMINAL	RULE 의 길 이
1	500	4
2	501	2
3	502	3
4	503	4
5	503	4
6	503	4
7	503	4
8	503	4
9	503	4
10	503	4
11	504	0
12	505	2
13	506	3
14	507	2
15	508	0
16	508	4
17	509	3
18	510	0
19	510	4
20	511	0

[표 4-1] 문법 테이블 (2)

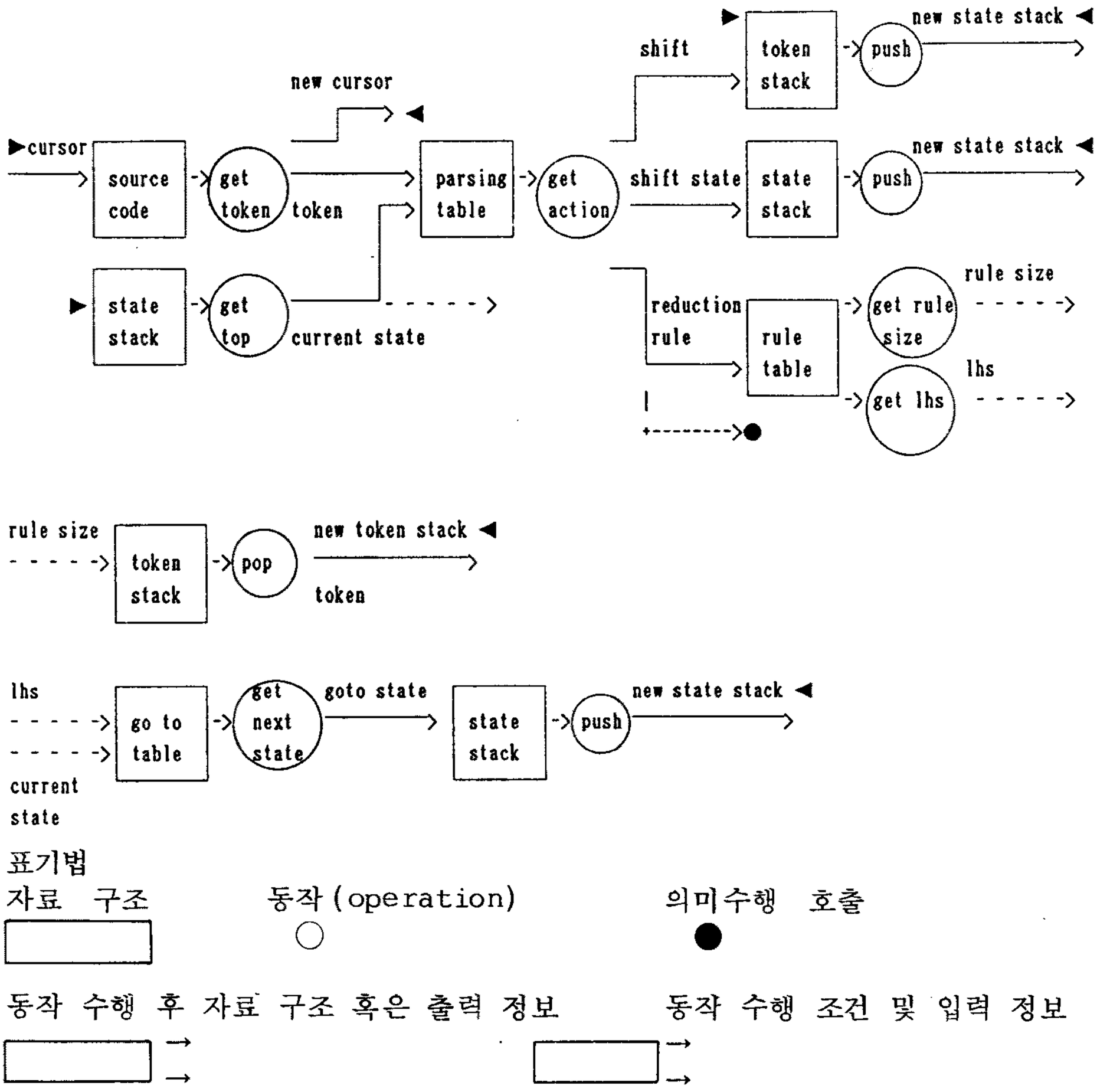
• 파싱 테이블

일련번호	현재 인식된 토큰의 ID	SHIFT ACTION 번 호
1	104	4
2	-1	0
3	-1	0
4	86	7
5	-1	0
6	157	9
7	-1	0
8	78	10
9	-1	0
10	67	13
11	-1	0
12	63	16
13	-1	-15
14	78	17
15	-1	0
16	46	18
17	69	21
18	70	20
19	111	19
20	167	23

• GOTO 테이블

일련번호	STACK상부의 NON-TERMINAL	REDUCE후의 상태 번호
1	500	1
2	501	2
3	502	3
4	-1	0
5	-1	0
6	505	5
7	506	6
8	-1	0
9	503	8
10	-1	0
11	-1	0
12	522	11
13	523	12
14	-1	0
15	507	14
16	508	15
17	-1	0
18	-1	0
19	-1	0
20	-1	0

REDUCE된 후에 상태 스택의 상부에 6이 있고 심볼 스택의 상부에 508이 있는 경우에 먼저 인덱스 테이블의 일련번호 6에서 GOTO 테이블 인덱스 15를 추출하고 GOTO 테이블에서 일련번호 15부터 508이나 -1 출현 할때까지 검색한다. NON-TERMINAL 508일 때 상태 번호 15를 얻어서 상태스택에 상태 15를 PUSH 한다.



[그림 4-3] 구문 분석 과정

(나) 의미 수행

의미 수행 코드의 작성은 파서에 의해 생성 규칙이 Reduction 될때 파스 트리를 구성한 후 트리를 운행 하면서 정보를 추출해내는 방법과 문법 지시적 변환 방법에 의해 각 생성 규칙의 Reduction시에 의미 수행 코드를 호출하여 실행하며 정보를 추출하는 방법이 있다. 본 시스템의 초기 프로토타입 시스템은 파스 트리를 이용하여 구현되었으나, 기억 장치의 관리와 트리의 운행, 규칙에 따른 트리 구조의 변경등의 문제점이 발견되어 문법 지시적 방법으로 최종 시스템을 구현 하였다.

① 자료 구조 및 흐름 정보의 구성

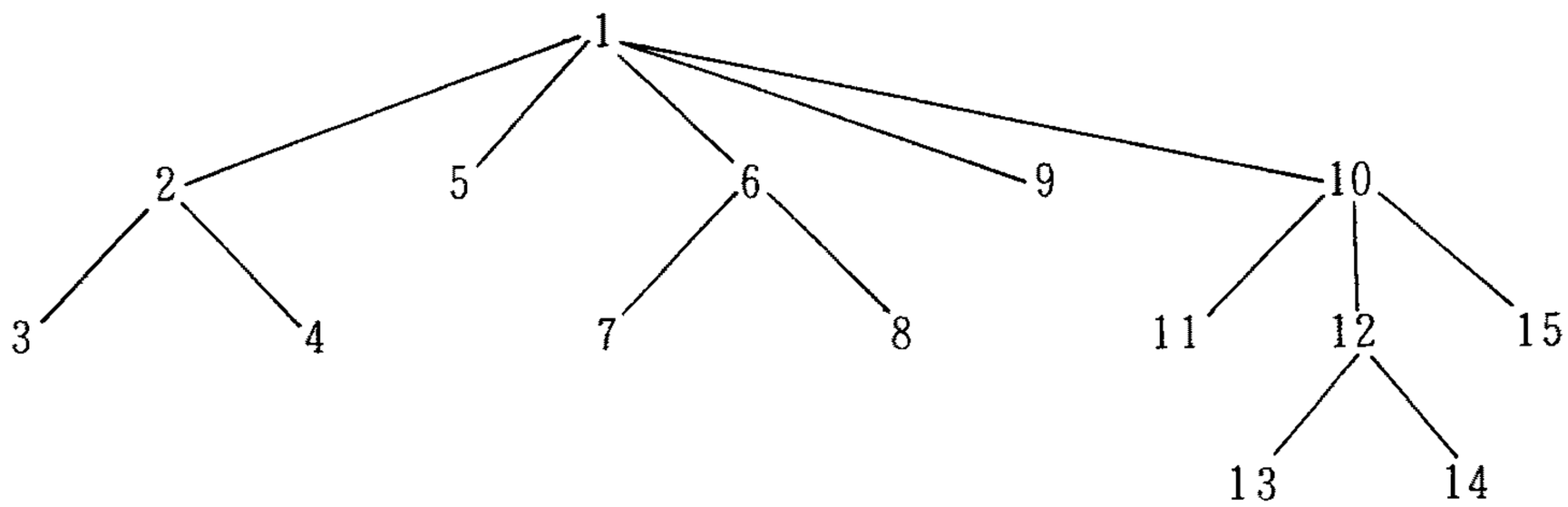
자료 구조 및 흐름 정보를 구성하기 위해서 먼저 데이타 디비젼과 화일 섹션의 변수 선언 내용을 심볼 테이블에 기록한다.

심볼 테이블은 심볼값, 레벨번호, 변수형, 변수 길이, 배열의 차원등에 관한 정보를 저장하고 있다. 코볼은 레코드의 선언이 가능하므로 레코드내에서 변수가 트리구조를 가진다. 특정 변수가 정의되거나, 참조될 때 (direct define, direct reference), 부모로부터 루트노드에 이르는 모든 노드와, 모든 자식 노드는 간접 참조 및 정의 (indirect define, indirect reference) 되므로 관련 노드에 대해서도 정의참조를 기록할 필요가 있다. 변수의 정의 참조 기록은 의미 수행 단계에서 이루어지며 "MOVE identifier1 TO identifier2"라는 구문이 인식되며, identifier 1은 참조, identifier 2는 정의된 것으로 의미수행이 이루어 진다. 정의, 참조의 정보는 레이블 단위로 이루어지며, 이 정보를 이용하여 패러그래프간의 변수 흐름 정보가 생성된다. 트리구조의 생성은 변수의 레벨 번호에 의하여 코볼에서 변수는

1부터 65까지 가능하다. 다음은 하나의 레코드 구조에 대한 트리 표현이다.

변수 선언

```
1          01 PAYROLL-RECORD.
2             03 PAYROLL-ID.
3                   05 PAY-PREFIX PIC XX.
4                   05 PAY-SUFFIX PIC XX.
5             03 PERSON-NAME PIC X(10).
6             03 DEPT-CODE.
7                   05 DEPT-PREFIX PIC XX.
8                   05 DEPT-PREFIX PIC XX.
9             03 ..................
10            03 ..................
11                  05 ..................
12                  05 ..................
13                        07 ..................
14                        07 ..................
15                  05 ..................
```



② 품질 메트릭 정보의 측정

㉮ 품질기준별 평점계산

품질 목표 정의의 다음 단계 작업으로 정적 측정기가 각 품질 점검항목에 대해 측정한 자료를 기초로 품질 기준별 평점과 인자별 평점을 산정한다. 먼저 품질 기준별 평점은 다음 식을 이용하여 계산된다.

$$S(Ci) = \sum_{i=1}^{n} \frac{Si}{n}$$

S(ci) : 품질 기준 Ci의 평점

Si : 품질 기준 항목 i가 획득한 점수

n : 측정된 품질 점검 항목 수

품질 기준별 평점 계산은 측정된 점검항목에 대해서만 평균 값을 구한다. 예를 들면 어떤 품질 기준에 대해 7개의 점검항목이 설정되어 있으나 실제 5개의 품질 점검항목만 측정되었다면 이 품질 기준에 대한 평점은 5개 기준 점수의 합계를 5로 나눈다. 품질 인자별 평점은 해당 품질 인자에 영향을 미치는 품질기준에 대한 가중 평균이다.

측정되는 품질 인자와 각 인자의 기준별 측정 방법은 다음과 같다. 각 기준은 정적 분석 과정에서 측정되며 최대 1점 최소 0의 값을 가진다.

• 구조화 정도

- 하향식 제어

1 - 상향식 PERFORM 문의 수 / 총 PERFORM 문의 수

• 프로그램 단순도

- 복합 논리 연산의 최소화

  1 - (AND, OR, NOT 의 수) / ELOC

- 단일 입구와 출구를 가진 loop

  단일 입구, 출구를 가진 LOOP 의 수 / 총 LOOP 의 수

- loop 내에서의 첨자의 변경

  1 - 변경된 LOOP INDEX 수 / 총 LOOP 수

- 프로그램 내에서 코드의 변경

  1 (ALTER 문 없음), 0 (ALTER 문 사용)

- 레이블의 비율

  1 - 레이블 수 / ELOC

- 단일 입구와 출구를 가진 모듈

  단일 입출구 모듈 수 / 총 모듈 수

- 고유한 변수명 사용

  1 (고유명 사용), 0 (중복명 사용)

- 혼합 연산의 최소화

  1 - 혼합 연산문 수 / 총연산문 수

- 형 변환의 최소화

  1 - 형 변환 문의 수 / MOVE 문 수

- 중첩 레벨의 최소화

  1 / 최대 중첩수

- 분기문의 최소화

  1 - IF 문의 수 / ELOC

- GOTO 문의 최소화

  1 - GOTO 문의 수 / ELOC

· 수행되지 않는 코드의 최소화

1 (No DC), 0 (DC)

· 변수 밀도

1 - 변수의 수 / (변수의 수 + ELOC)

· 사용되지 않은 변수의 정의

1 - 사용되지 않은 변수의 수 / 총 변수의 수

· 참조되지 않는 변수의 정의

1 - 참조 되지 않은 변수의 수 / 프로시듀어 디비젼내의 총 변수 수

· 초기화 되지 않는 변수

1 - 초기화 되지 않은 변수의 수 / 프로시듀어 디비젼내의 총 변수 수

점검 항목에 대한 측정은 구문 분석시 관련 규칙이 reduce 될 때 의미수행코드에서 수행된다. 예를 들어 정밀도, 처리효율도라는 품질 기준인 '형 변환문 최소화'라는 항목은 다음 각각의 구문이 인식되었을 때 변수의 형을 검사함으로써 이루어진다.

MOVE 문의 형 변환 검사

```
문장                MOVE  var1  TO  var2 var3  var4.
생성 규칙  1   move_statement : move_to
           2                  | MOVE CORRESPONDING identifier1
                                                TO identifier2;
           3   move_to       : MOVE identifier1 TO identifier2
           4                  | MOVE literal TO identifier
           5                  | move_to identifier;
```

의미 수행

```
reduce 3 : if (SSP(2) -> var_type != 
                SSP(4) -> var_type ) {
                RecordViolation(SSP(2), SSP(4)
                }
           StackCopy(SSP(2), lhs);
reduce 5 : if (SSP(1) -> var_type != 
                SSSP(2) -> var_type ) {
                RecordViolation(SSP(1), SSP(2)
                }
           StackCopy(SSP(2), lhs);
```

위의 코드에서 StackCopy라는 함수는 첫번째 인수(토큰 스택의 위치)의 속성을 규칙이 reduce된 후 토큰 스택이 pop된 후에도, lhs에 해당되는 논터미날에 유지될 수 있도록 복사하는 기능이며, SSP함수는 토큰 스택에서, 인수 위치 변수의 심볼 테이블 위치를 return하는 함수이다. RecordViolation함수는 형 변환 내용을 저장하는 함수이다.

㉰ 복잡도 계산

- Cyclomatic 복잡도

프로그램의 제어 흐름도 G를 기초로 하여 측정하며, 다음식에 의한다.

여기서, n은 제어흐름 그래프의 노드수, e는 에너지를 말하며 p는 strongly connected component수를 의미한다. Cyclometic 복잡도

의 계산은 아래와 같이 단수화 될 수 있다.

V(G) = predicate 수 + 1

복잡도 계산은 코볼내의 PREDICATE 수로 계산된다.

• 기타 계산 되는 복잡도 메트릭은 다음과 같다.

- 화일의 수
- 레이블의 수
- 연산자, 피연산자의 수 및 Software Science
- ELOC
- 레이블당 평균 ELOC
- 주석의 수
- LOC

③ 설계 정보의 복구

코드에서 설계 복구 과정에 의해 추출될 수 있는 구조도는 모듈간의 호출관계 및 Sequence, Selection과 Iteration이다. 이것은 설계 단계에서 사용되는 모듈 구조도상의 모듈간의 호출관계, 모듈간의 정보 및 제어 흐름과 비교해 볼 때 모듈간의 정보와 제어흐름 정보가 누락되는 것이다. 그 이유는 코볼 프로그램내에서 선언된 변수들이 거의 전역 변수로서 모듈간의 정보 흐름을 표현하기 어렵기 때문이다. 모듈 구조도를 구성하는 과정의 기본 정의는 다음과 같다

• 코볼 프로그램의 한 패러그래프를 한 모듈로 간주한다.

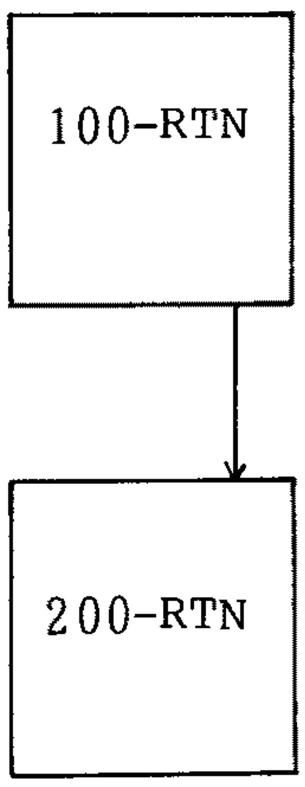
• 구조도는 패러그래프간의 PERFORM 문에 의한 관계를 나무 구조로 나타낸 것이다.

• SORT 문의 INPUT PROCEDURE 와 OUTPUT PROCEDURE 는 INPUT

PROCEDURE 와 OUTPUT PROCEDURE 를 PERFORM 한 것으로 간주한다

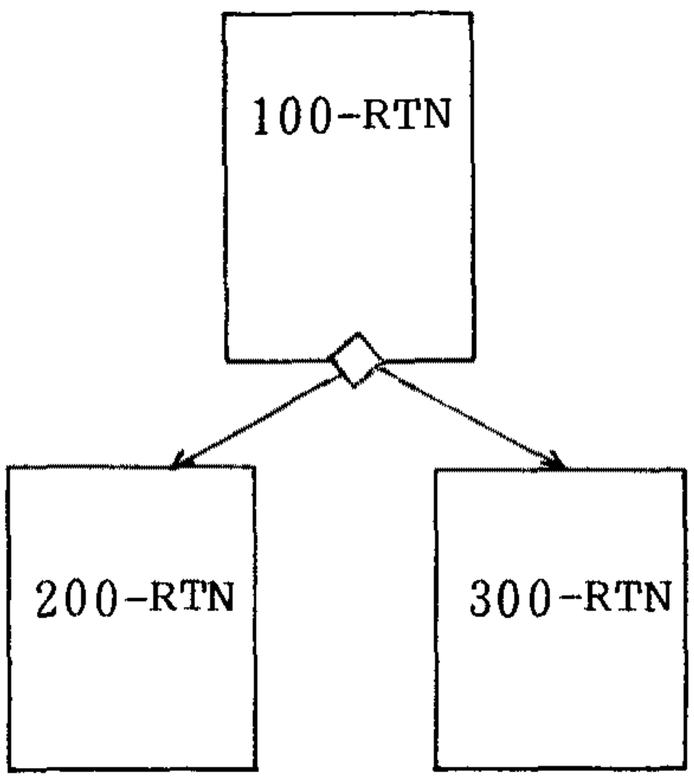
- GOTO 에 의한 제어 전이는 구조도 상에서 표현하지 않는다.
- EXIT PARAGRAPH 는 관계에서 제외한다.
- 구조도에서 표현되는 Notation 의 종류는 다음과 같다.
  - PERFORM 문에 의한 패러그래프의 호출
  - 조건 PERFORM 문에 의한 패러그래프의 호출
  - 중첩된 IF 문 내의 PERFORM 구조의 selection 에 의한 표현 (중첩 구조내에서의 관계는 고려하지 않는다)
  - 어휘적 근접의 표현 - 어휘적 근접에 의해 순차 흐름이 형성될 때만 표현하며 어휘적 근접에 의한 순차 흐름이 발생되는 경우는 SECTION 내 패러그래프간과 (SECTION 이 프로그램 내에서 호출되는 경우) PROCEDURE DIVISION 에서 시작하여 STOP RUN 이 출현하기 까지이다.
  - 여러개의 패러그래프를 걸쳐서 PERFORM 하는 경우에 패러그래프 군에 의한 CHILD 를 생성하고 상세 나무는 별도로 구성된다.

- PERFORM 문에 의한 패러그래프의 호출

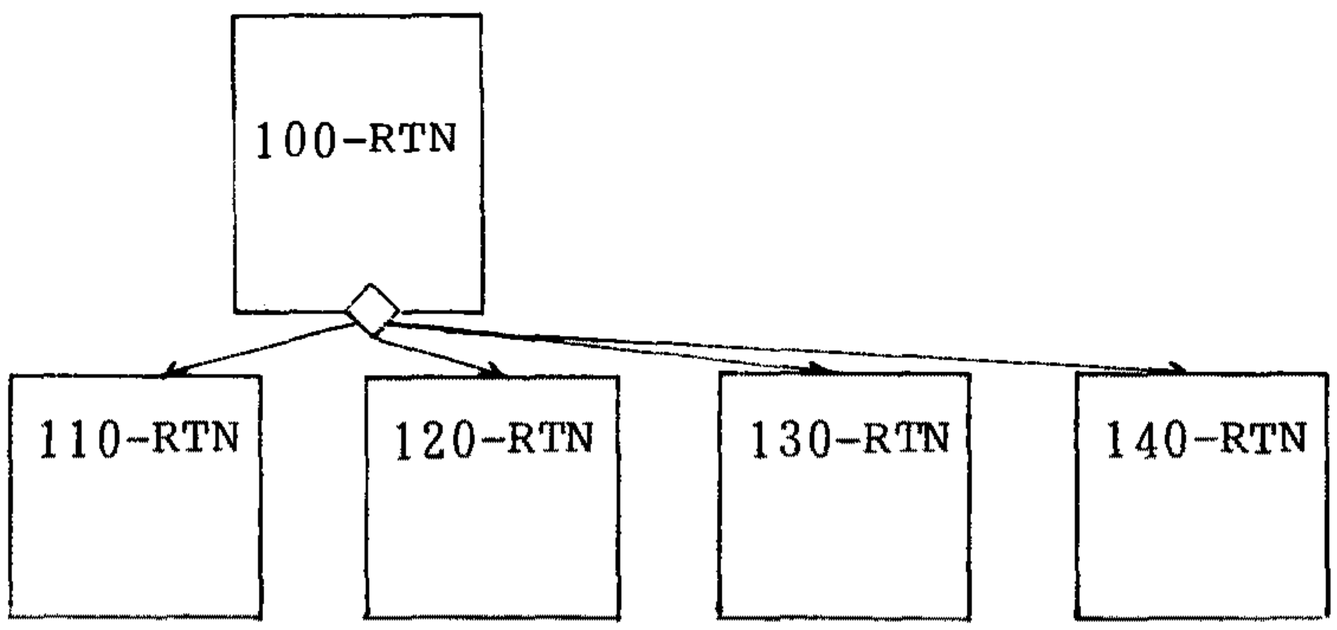



```
100-RTN.
.....
.....
.....
PERFORM 200-RTN.
```

• 조건 PERFORM 에 의한 패러그래프의 호출



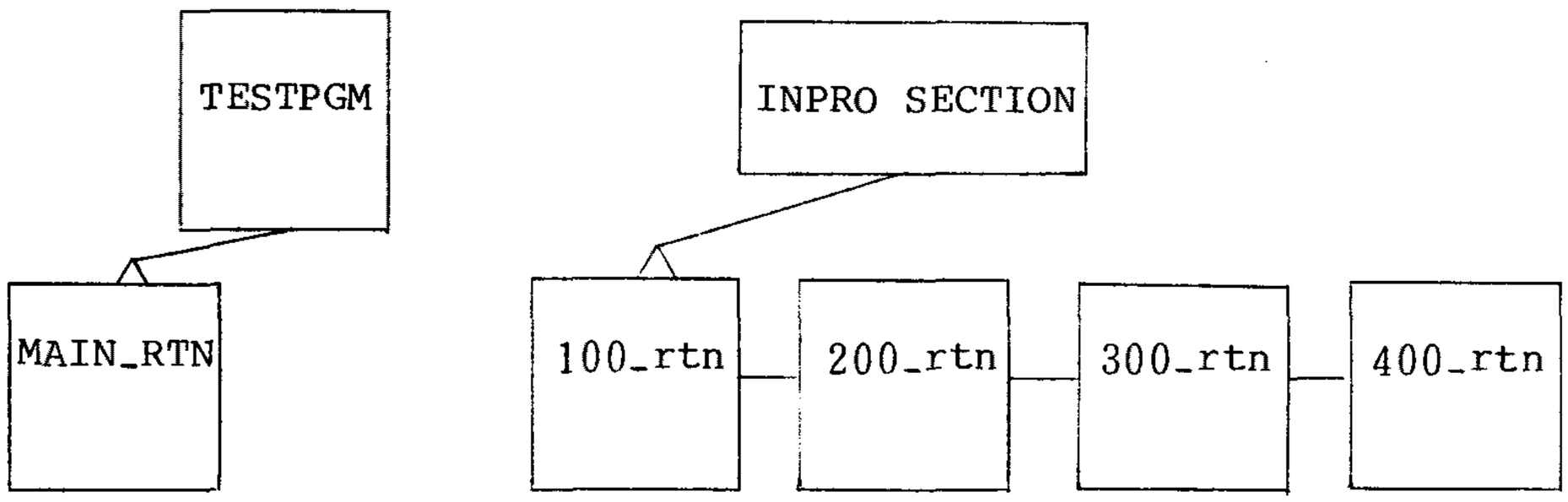

```
100-RTN.                                100-RTN.
    ......                                  ......
    ......                                  ......
    EVALUATE A>3                            READ AT-F INVALID KEY
    WHEN TURE PERFORM 200-RTN               PERFORM 200-RTN
    WHEN FALSE PERFORM 300-RTN.             PERFORM 300-RTN.
```



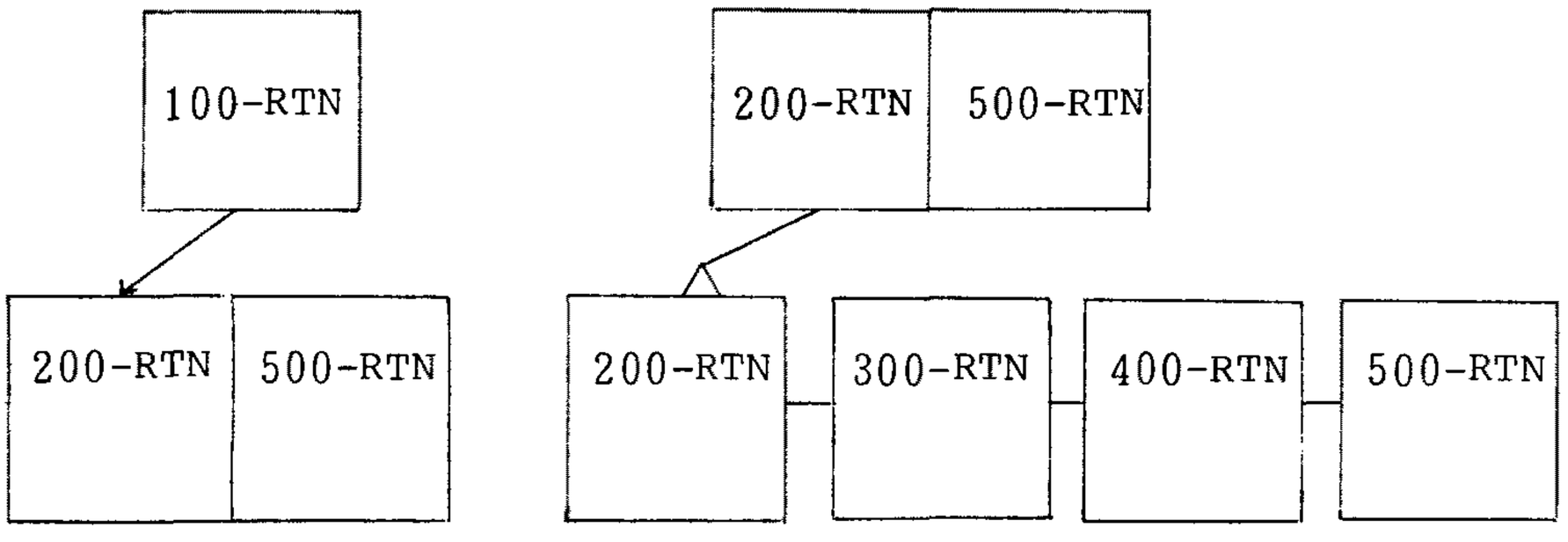

```
100-RTN.
    IF CASE1 PERFORM 110-RTN
    ELSE IF CASE 2 PERFORM 120-RTN
```

```
ELSE
   EVALUATE A>3
     WHEN TRUE PERFORM 200-RTN
     WHEN FALSE PERFORM 300-RTN.
```

• 어휘적 근접의 표현



• PERFORM 200-RTN THRU 500-RTN 의 경우



(다) 출력별 제공 결과

〈프로젝트 요약 및 통계 (Program Summary and Statistics)〉

- ㅇ 프로그램명
- ㅇ 저자, 작성일, 최종 수정일
- ㅇ TLOC

o ELOC

o 코멘트 라인 수

o copy 라인 수

o 멕케이브의 복잡도

o 최대 교차 수

o 홀 스테드 소프트웨어 사이언스 메트릭

o 패러그래프의 갯수

o 섹션의 갯수

o 화일의 갯수

o 문 유형별 사용횟수

멕케이브의 복잡도는 프로그램의 구조를 근거로 복잡도를 측정하는 방법으로 상세한 설명은 [시89]에 기술되어 있다. 복잡도 메트릭의 값은 프로그램을 시험할 수 있는 basic path의 수와 같다.

모듈이나 프로시듀어의 Cyclomatic complexity가 10이상인 경우에, 그 모듈은 오류가 발생할 가능성이 많다고 간주된다. 또한 모듈을 시험하거나 이해하고 디버깅하는데, 그리고 유지보수하는데 어려움이 있다고 생각될 수 있다. 이러한 모듈들은 단순한 구조로 분할되는 것이 바람직하다. 구조적 시험 관점에서 본 적당한 규모의 모듈이란 시험에 투입되는 노력이 통제 가능해야 하며, 쉽게 읽고 이해할 수 있어야 하고, 다른 시스템에 재이용이 가능하도록 응집도가 높고 유지보수하기 쉬어야 한다.

〈변수 참조(Variable Reference)〉

o 변수명

o 변수 레벨 번호

ㅇ 변수형 및 길이

ㅇ 변수 초기값 여부

ㅇ 각 패러그래프별 정의 참조 구분

〈화일 변수 참조(File Variable Reference)〉

ㅇ 화일명

ㅇ 화일길이

ㅇ 변수 참조 내용

ㅇ 레코드 키 구분

ㅇ Alternate Key

〈화일 사용 내역(File Usage)〉

ㅇ 화일명

ㅇ 화일 Organization

ㅇ 화일 액세스 모드

ㅇ 레코드 키

ㅇ Alternate Key

ㅇ 화일 암호

ㅇ 라벨 레코드

ㅇ 블록킹 정보

ㅇ 화일 길이

ㅇ 패러그래프별 참조 여부

- 입력 Open

- 출력 Open

- 입출력 Open

- Close

- 읽기
- 쓰기
- 수정
- 삭제
- 소트
  - 소트 입력
  - 소트 출력

〈 Accept 정보 (Accept Information)〉

- o 패러그래프명
- o Accept 필드
- o Accept 소스필드
- o 라인 번호

〈프로그램 접속 상황 (Program Interface)〉

- o 패러그래프명
- o 접속구분
  - CALL
  - ENTRY
- o 호출 프로그램명
- o 매개 변수
- o 라인 번호

〈논리 구조 (Hierarchical Structure)〉

- o 패러그래프명 및 섹션명
- o 각 패러그래프에서의 PERFORM 내역 (조건여부)
- o 각 패러그래프에서의 GOTO 내역 (조건여부)

o 각 패러그래프내에서의 STOP RUN 내역 (조건여부)

o 각 정보의 발생 라인 번호

o 상향 (Upward) GOTO 의 사용여부

o 상향 (Upward) PERFORM 의 사용여부

〈오류 진단 (Diagnostics)〉

o 변수 anomaly

- 프로시듀어 디비젼에서 사용되지 않은 변수

- 프로시듀어 디비젼에서 정의되었으나 참조되지 않은 변수

- 프로시듀어 디비젼에서 정의가 되지 않고 데이타 디비젼에서 초기치가 정의되지 않은 변수

o Dead Code

도달 불가능한 코드의 부분

- 패러그래프명 및 라인번호

o 형의 불일치

MOVE 문에서의 source 변수와 destination 변수의 형의 불일치

IF 문에서의 비교 변수간의 형의 불일치

- 각 변수의 형 및 길이

- 패러그래프명 및 라인번호

o 혼합연산

연산에서 코볼의 9 mode 와 X mode 의 혼용

- 각 변수의 형 및 길이

- 패러그래프명 및 라인번호

o 복잡한 부분

- 부정조건이 사용된 패러그래프명 및 라인번호

- 복합조건이 사용된 패러그래프명 및 라인번호
- Large Nesting Level

○ GOTO 및 ALTER 동사의 사용부분

- 패러그래프명 및 라인번호

○ GOTO runaway path

PERFORM 범위에서의 범위밖으로의 GOTO에 의한 Branch

○ GOTO에 의한 endless loop

○ PERFORM에 의한 endless loop

〈확장 원시 코드〉

○ copy문 확장

○ 라인번호 및 패러그래프 번호

시스템 총괄 정보는 다음과 같다.

〈프로그램별 Metrics〉

한 시스템내의 모든 프로그램의 품질 정보를 비교 제공한다. 비교 제공되는 품질 정보는 다음과 같다.

○ Total Lines of Code

○ Executable Lines of Code

○ Cyclomatic Complexity

○ Halstead Software Science

○ Maximal Intersection Number

○ Number of Files Referenced

○ Number of Paragraphs Defined

〈프로그램 호출 정보〉

한 시스템내의 프로그램간의 호출관계를 matrix형태로 표현한다.

종축은 호출 프로그램 횡축은 피호출 프로그램이 된다.

〈화일 참조 정보〉

시스템내의 모든 화일에 대해서 어떤 프로그램이 어떤 형태로 화일을 사용하고 있는지를 나타낸다.

화일의 사용형태는 읽기, 쓰기, 수정, 삭제로 구분되어 표시된다.

〈 Program Structure Chart 〉

프로그램 모듈간의 호출관계를 Structure Chart로 표현 Structure Chart 구성에서 사용되는 Notation

- o module
- o 모듈 호출 관계
- o lexical adjacency
- o sequence
- o selection

(래) 사용자 접속

도구가 로드되면 코볼 원시 프로그램 저장 디렉토리의 프로그램들이 출력되고 메뉴바에 조회, 분석, 결과, 선택, 인쇄, 종료등이 나타난다. 조회를 선택하여 프로그램을 조회할 수 있고 정적분석 결과는 결과를 먼저 선택한 후 서브 메뉴가 나타나면 원하는 출력을 선택하도록 되어 있다. 분석을 선택하는 경우 하나의 프로그램 또는 해당 디렉토리 내의 모든 프로그램을 분석하여 정적정보를 추출한다.

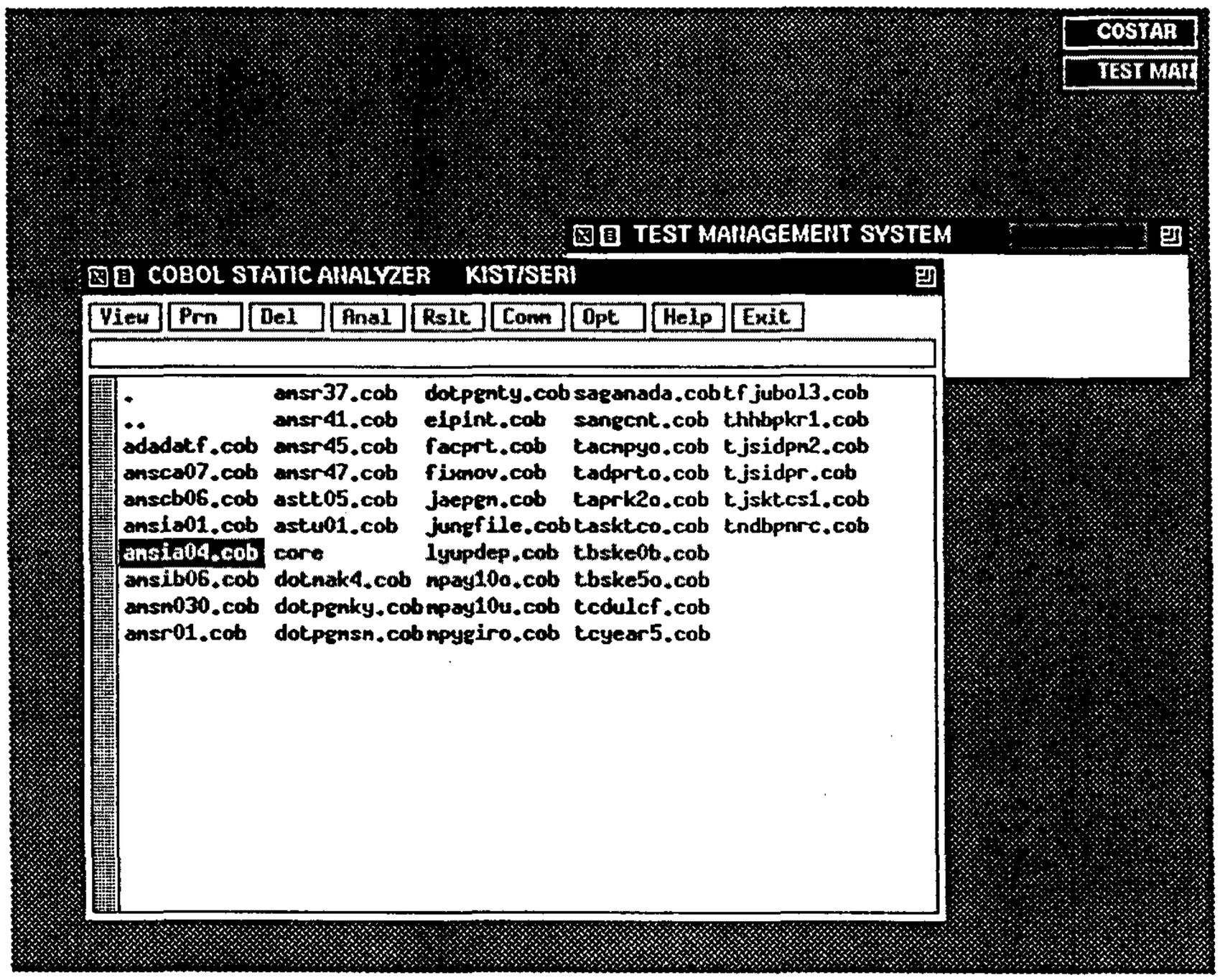
[그림 4-4]은 초기 사용자 화면에서 분석하고자 하는 프로그램을 선택한 경우이다. 해당 프로그램이 이미 분석된 경우에는 바로 메뉴바에서 결과를 선택한다.

[그림 4-5]는 초기 화면에서 결과에 마우스를 위치하고 버튼을 누

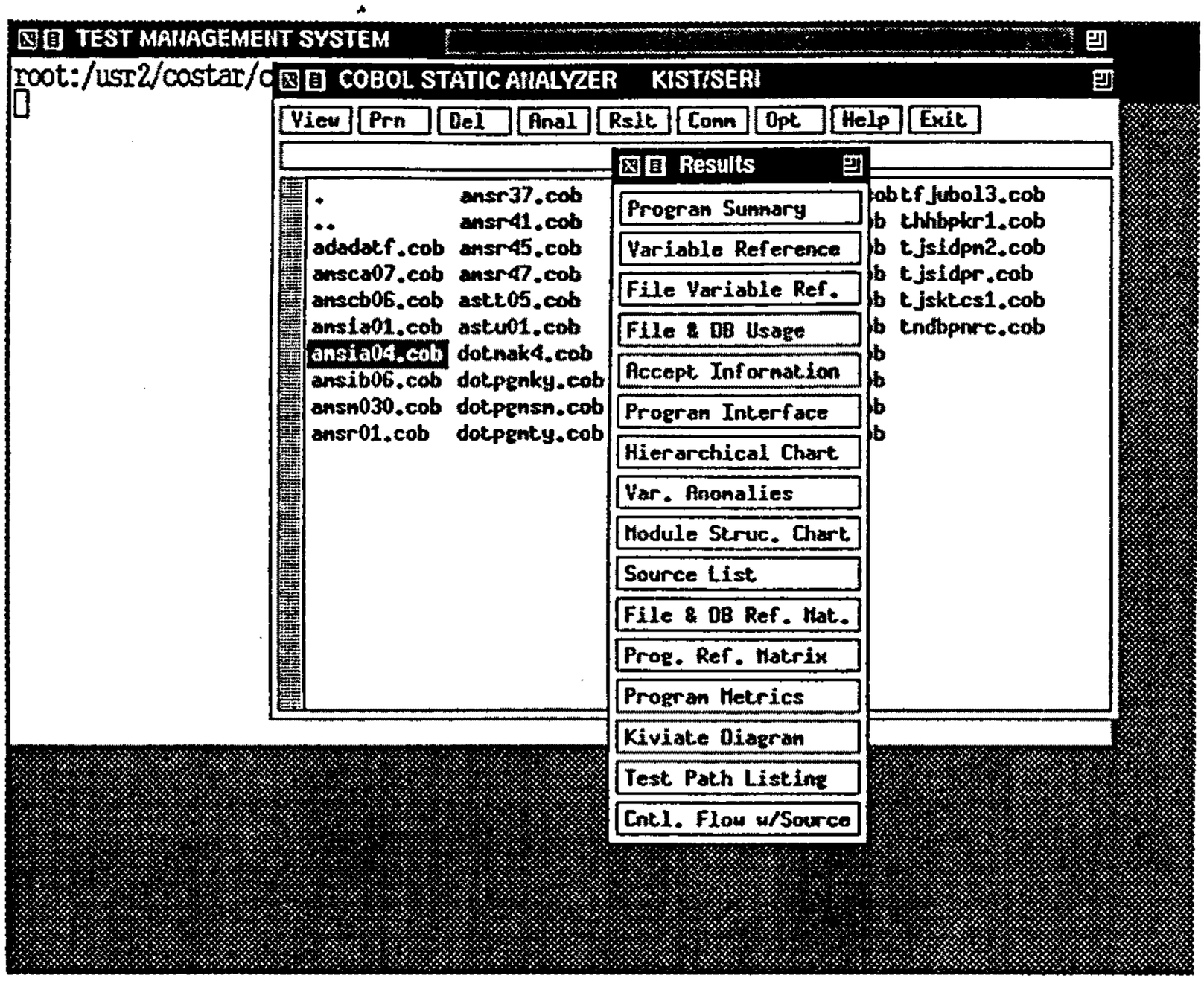
르면 나타나는 정적 분석 결과 안내 화면이다.

[그림 4-6], [그림 4-7], [그림 4-8]은 각 화면에서 프로그램 논리 구조, 변수 참조, 프로그램 메트릭 정보를 선택한 화면이다. 각 화면의 결과는 인쇄할 수 있다.

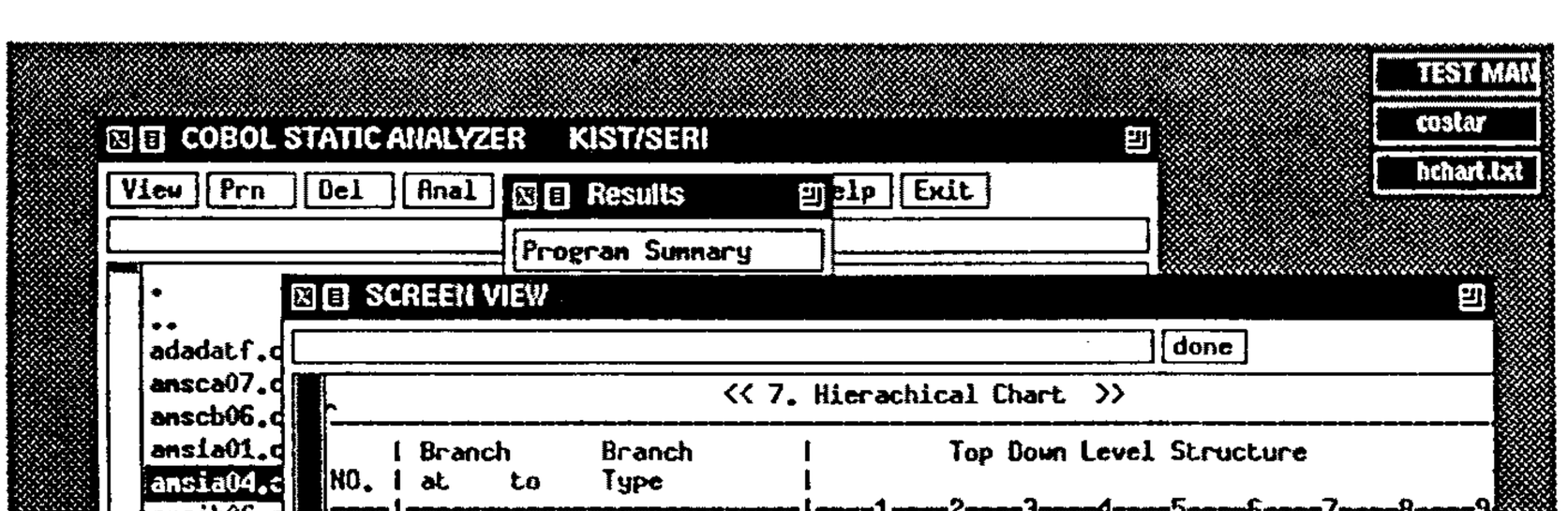
각 정적 분석 결과의 제공 정보는 다음과 같다.



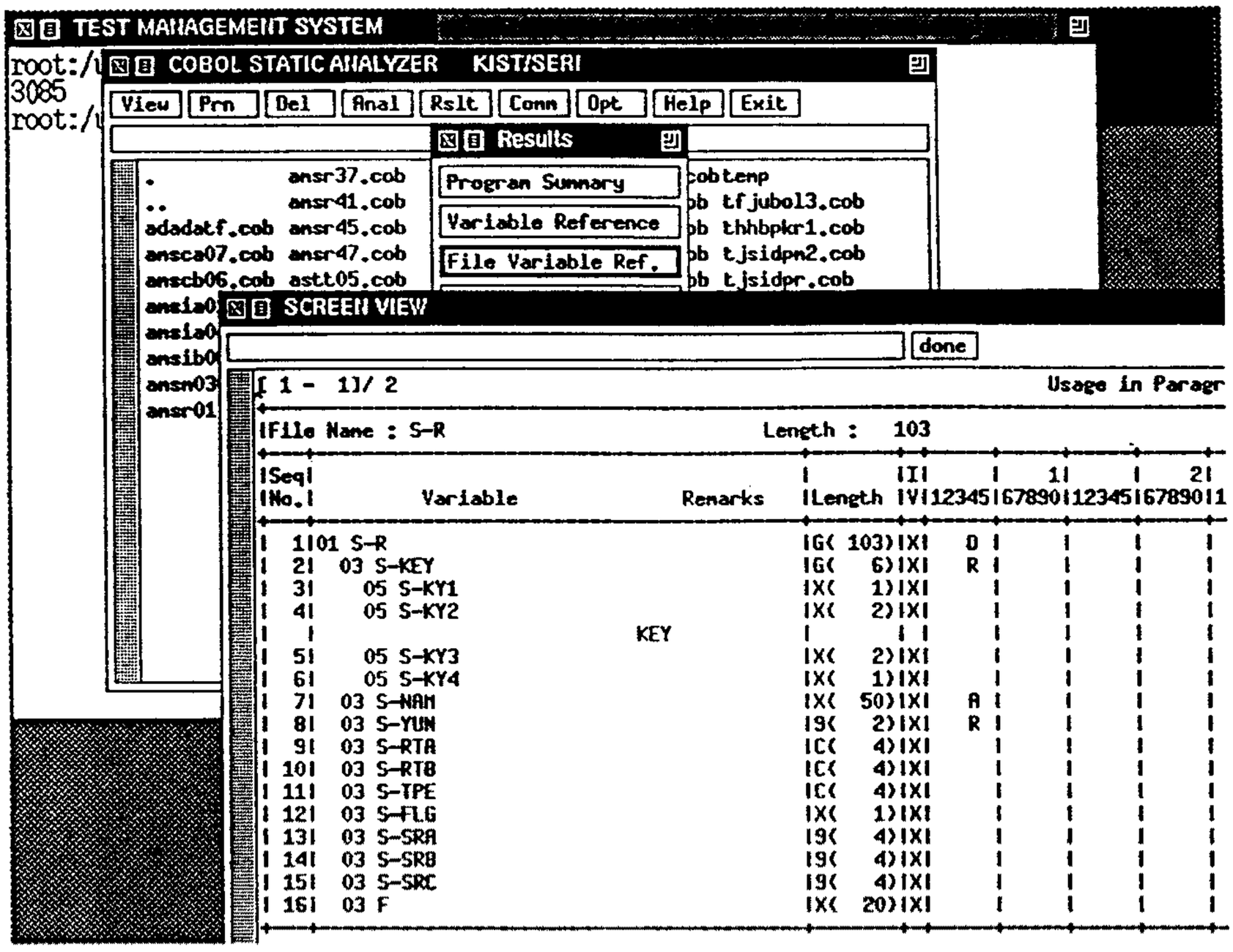
[그림 4-4] 분석 또는 조회를 위한 프로그램 선택



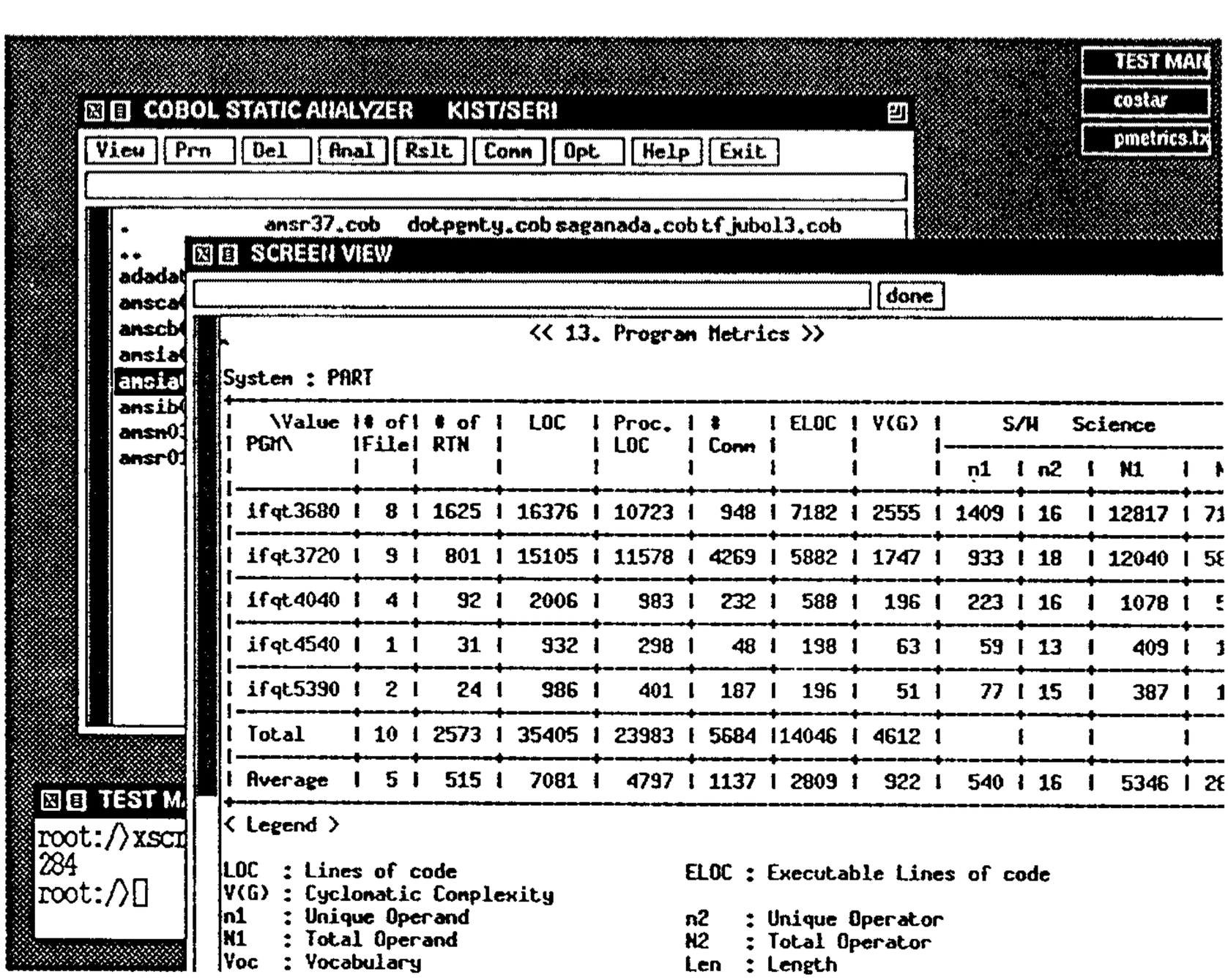
[그림 4-5] 정적 분석 결과 선택 화면



[그림 4-6] 프로그램 논리 구조의 화면 출력



[그림 4-7] 변수 참조 정보의 화면 출력



[그림 4-8] 품질 메트릭 정보의 화면 출력

다. 시험 사례 설계 지원 시스템

(1) 도구의 개발 방향

(개) 도구의 특징

동적 시험 작업은 노력 집약적인 작업으로, 특히 주어진 시험 기능을 수행하기 위한 데이타의 생성 작업에 많은 노력과 숙련된 전문가를 필요로 한다. 사례 생성 지원기는 이러한 데이타 생성 작업을 자동화하여 시험 작업을 감소하고 범주에 의해 시험 과정을 철저화 하고자 개발되었다. 데이타의 생성은 형식 언어로 기술된 요구 분석이나 설계 명세로 시험 데이타를 생성하는 방법과 프로그램의 구조에 의한 방법등이 있다. 전자의 경우 형식적 요구 사항 기술언어의 개발이 선행되어야 하나, 사례가 모든 요구 기능을 시험하는지 추적하기 쉬운 장점이 있다. 궁극적인 시험 사례 지원 도구는 정형적 명세에 의한 데이타와 예상 출력을 생성할 수 있어야 한다.

시스템의 산출물의 구조에 의한 방법은 위의 방법에 대한 차선책으로 그리고 위의 방법과 병행하여 보완적인 방법으로 사용될 수있다. 이 연구에서는 산출물의 구조에 의한 시험 데이타의 생성을 지원할 수 있도록 시스템의 구조를 추출하고 구조내의 특정 경로를 수행하기 위한 조건을 분석하여 동적 시험을 지원한다.

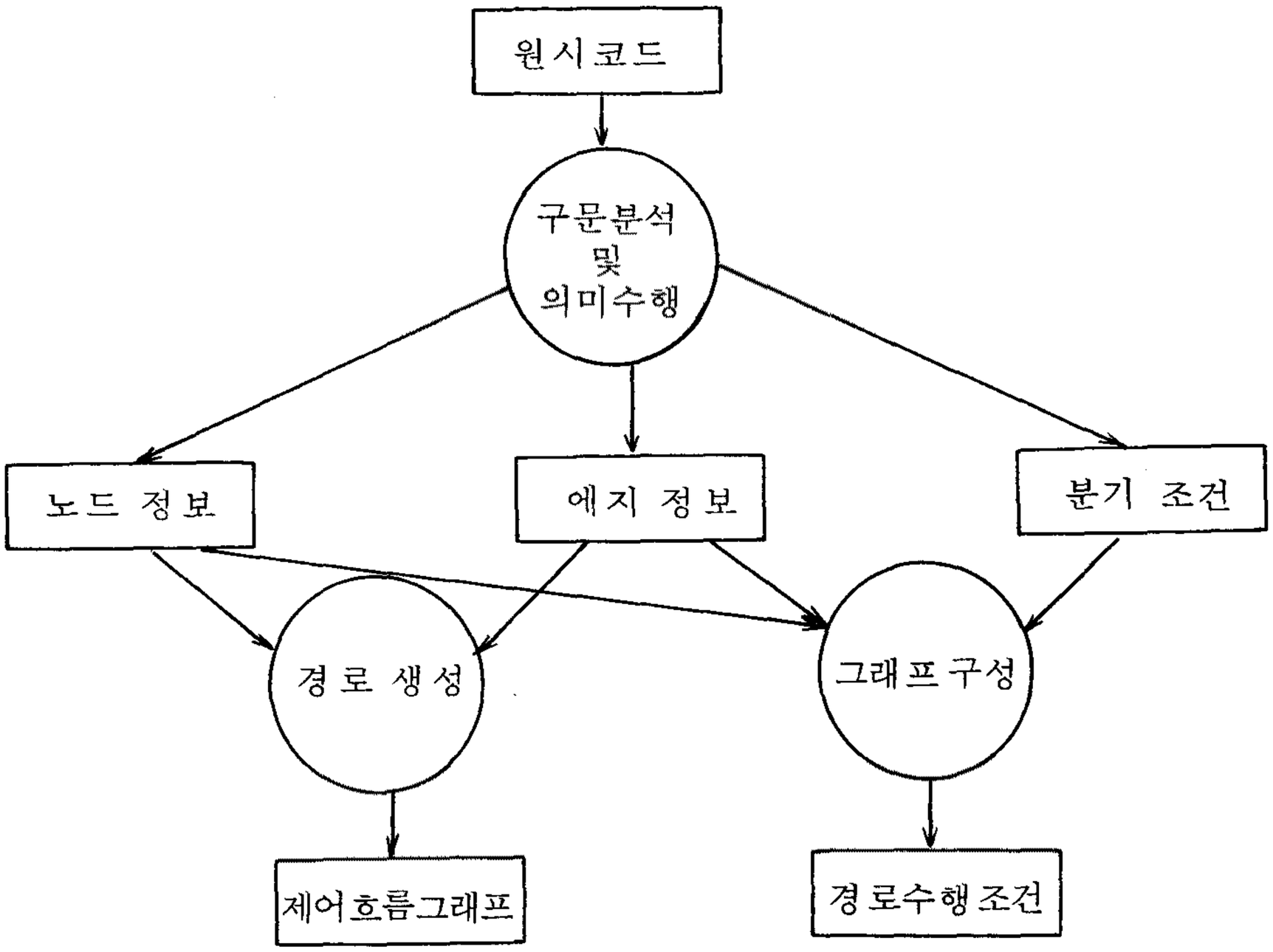
(내) 도구의 기능 및 구조

국외에서 개발된 도구의 경우 프로그램 제어 구조와 각 분기 조건을 추출하는 기능을 수행하고 있다. 개발된 도구는 다음과 같은 기능을 가진다.

- 프로그램 제어구조의 생성
- 제어 그래프의 출력

• 제어 그래프 상의 분기 조건 출력

도구의 구조를 도식화 하면 [그림 4-9]와 같다.



[그림 4-9] 데이타 생성 지원기의 구조

(대) 구조적 시험 데이타 생성의 배경

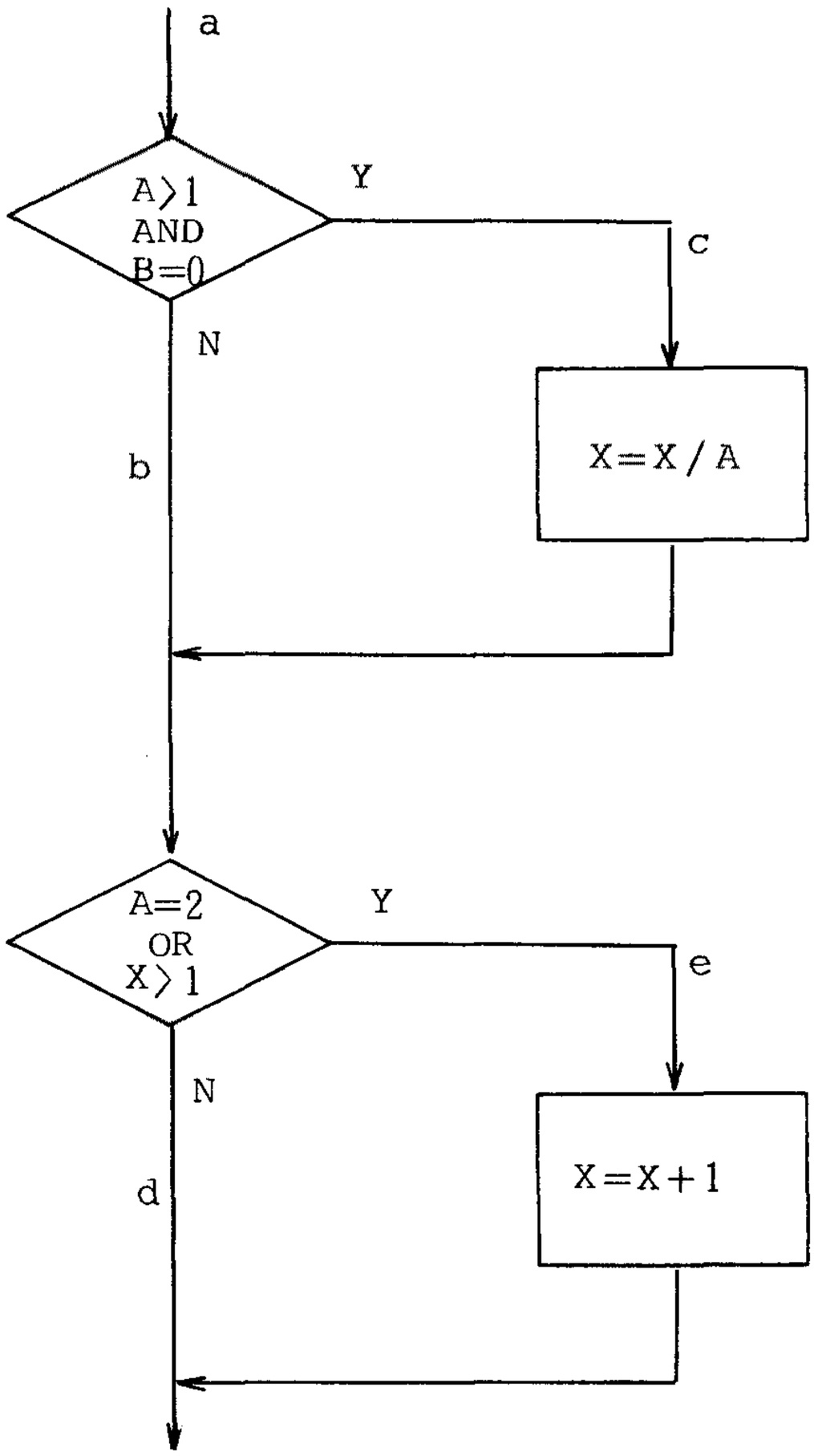
프로그램 시험은 요구 명세에 기술된 행동을 확인하여 정확히 작동하는가를 검증하는 것이지만 실제로 원시 프로그램 내에 모든 에러를 찾아내는 것은 불가능하다. 이상적인 테스트를 수행하기 위한 방법론들이 제시되어 왔으며, 이 중 프로그램의 구조를 근간으로 하는 시험 방법론들이 최근 활발히 연구되고 있다. 이들 방법론중에는 논리 커버리지 (Logic coverage), 자료 흐름 커버리지 (data flow coverage)가 있으며 최근들어 함수적 커버리지 (functional coverage)이

론이 제시되었다.

프로그램 내에는 반복문이 존재하기 때문에 모든 경로를 실행하는 것은 불가능하다. 따라서 구체적 테스트 목표를 설정하고, 그 목표에 도달했을 때 테스트를 종료하는 기준이 마련되어야 한다. 이를 커버리지 (coverage) 라고 한다.

가장 간단한 커버리지가 문장 커버리지 (statement coverage) 인데, 이는 모든 문장이 최소한 한번은 실행되어야 한다는 것을 요구한다. 그러나 모든 문장을 실행하기 위한 시험 사례가 합리적인 시험의 기준으로 볼때 결코 충분하지 못하다. [그림 4-10] 에서 보듯이 단 한 개의 시험 사례로써 모든 문장을 수행할 수는 있지만, 이 시험 사례가 합당하다고 보기는 어렵다. 이와 같은 이유로 문장 커버리지를 시험 커버리지로 적용하는 예는 드물다. 다음으로 결정 커버리지(decision or branch coverage) 가 있는데, 이 커버리지는 프로그램 내의 모든 분기가 최소한 한번은 참과 거짓이 될 수 있도록 충분한 시험 사례를 작성할 것을 요구한다. 결정 커버리지를 만족한다. 모든 문장은 분기문 또는 분기점에서 시작하기 때문에, 모든 분기문의 참 거짓이 실행된다면 모든 문장이 실행된다.

다음은 조건 커버리지 (conditional coverage) 인데 이는 분기문 내에 모든 가능한 조건을 검증할 수 있어야 한다. 조건 커버리지는 모든 문장이 실행하는 것을 보증하지 못한다. 또 다른 커버리지는 경로 커버리지 (Path coverge) 가 있는데, 이는 모든 경로가 실행되야 할 것을 요구하고 있다. 그러나 실제로 무한히 많은 경로에 대해 시험을 설계하는 것은 불가능하다.



[그림 4-10] 제어 흐름의 예

(2) 도구의 설계 및 구현

(개) 프로그램 제어구조의 생성

데이타 생성 지원 도구는 그래프 이론을 바탕으로 하고 있다. 프로그램 제어 그래프는 정점(노드) 집합 V와 에지 집합 E로 이루어진 방향그래프 G=(V,E)이다. 그래프의 정점 G는 프로그램 코

드 세그먼트를 나타내며, 이러한 코드 세그먼트는 항상 함께 수행되는 일련의 문의 모임이다. 에지는 세그먼트간의 제어 전이를 나타낸다. 코볼 프로그램에 대한 노드 생성 기준을 다음과 같이 정의한다

1. 레이블 출현시 새로운 노드를 생성한다.

   - PROCEDURE DIVISION도 레이블과 같은 경우로 간주한다.

2. 하나의 패러그래프 내에서 조건문을 제외한 함께 수행되는 일련의 문들을 하나의 노드로 구성한다.

3. PERFORM문은 항상 별도의 노드로 구성한다.

4. 조건문과 GOTO 문, PERFORM 문 후에는 항상 새로운 노드를 생성한다.

5. 한 패러그래프의 마지막 문이 조건문인 경우에는 패러그래프가 한개의 노드로 끝나게 하기 위하여 더미 노드를 생성한다.

조건문의 종류는 다음과 같다.

Decision	IF	
	EVALUATE	
Arithmatic	ADD	····· ON SIZE ERROR
	COMPUTE	····· ON SIZE ERROR
	DIVIDE	····· ON SIZE ERROR
	MULTIPLY	··· ON SIZE ERROR
	SUBTRACT	····· ON SIZE ERROR
Data Movement	STRING	····· ON OVERFLOW
	UNSTRING	··· ON OVERFLOW
Odering	RETURN	····· AT END
Input/ Output	DELETE	····· INVALID KEY

READ ····· AT END

READ ····· INVALID KEY

RECEIVE ··· NO DATA

REWRITE ··· INVALID KEY

START ····· INVALID KEY

WRITE ····· AT END-OF-PAGE

WRITE ····· INVALID KEY

Procedure Branching

CALL ····· ON OVERFLOW

Table Handling SEARCH ··· WHEN

조건문의 노드 생성 기준은 다음과 같다.

〈 IF 문 〉

문법 : IF conditional expression THEN statements ELSE statemen-
ts

기준 : IF conditional expression 에서 하나의 노드를 생성하며
후행 노드는 then statements 의 첫번째 노드와 else
statements 의 첫번째 노드가 된다. statements 는 조건문의
중첩이 가능하다.

〈 EVALUATE 문 〉

문법 : EVALUATE conditional expression

WHEN condition imperative_statements

{WHEN condition imperative_statements } ···

WHEN OTHER imperative_statements

기준 : EVALUATE conditional expression 에서 하나의 노드를 생성한다.

후행 노드는 when part 들과 when other part 의 첫번째 노드들이다.

〈 SEARCH 문 〉

문법 : SEARCH identifier { WHEN conditional expression impera-
tive_statements} …
AT END imperative statements

기준 : SEARCH indentifier 에서 노드를 생성하고 후생 노드는 when part 들과 at end part 의 첫번째 노드들이다.

〈 ADD 문등 조건문 (CALL, DIVIDE, ADD, MULTIPLY, SUBTRACT, 등) 〉

문법 : ADD identifier to identifier ON SIZE ERROR imperative
statement

기준 : ADD 문등에서는 ADD 문의 전에 나타난 문이 노드생성기준 4번 (조건문, GOTO 문, PERFORM 문) 에 해당되는 경우에만 새로운 노드를 생성한다.

노드의 정보 저장 형태는 다음과 같다.

프로그램의 정적 분석 과정에서, 에지 정보는 linked list 를 이용하여 구현되나 정적분석이 완료된 후 다음과 같은 정규화 된 형태로 저장한다.

• 노드 정보 저장 구조

```
struct node_record {
    int node_para_seq;              /* 패러그래프 위치 */
```

```
int node_number;                    /* 노드번호  */

char node_type;                     /* PERFORM, EVALUATE 등
                                       노드 유형  */

int node_start_line;

int node_end_line;

int node_goto_number;               /* GOTO 제어  점  */

int node_perform_number1;           /* PERFORM 시작 위치 */

int node_perform_number2;           /* PERFORM 완료  위치 */

int node_succ_cnt;                  /* 후행  노드  수 */

int node_flow_start;                /* 에지  정보  시작점 */

}
```

• 에지 정보 저장 구조

```
struct flow_record {

    int  flow_pred_node;            /* 선행  노드 */

    int  flow_successor_node;       /* 후행  노드 */

    int  flow_path_type;            /* 참, 거짓  경로  구분 */

    }
```

• 레이블 정보

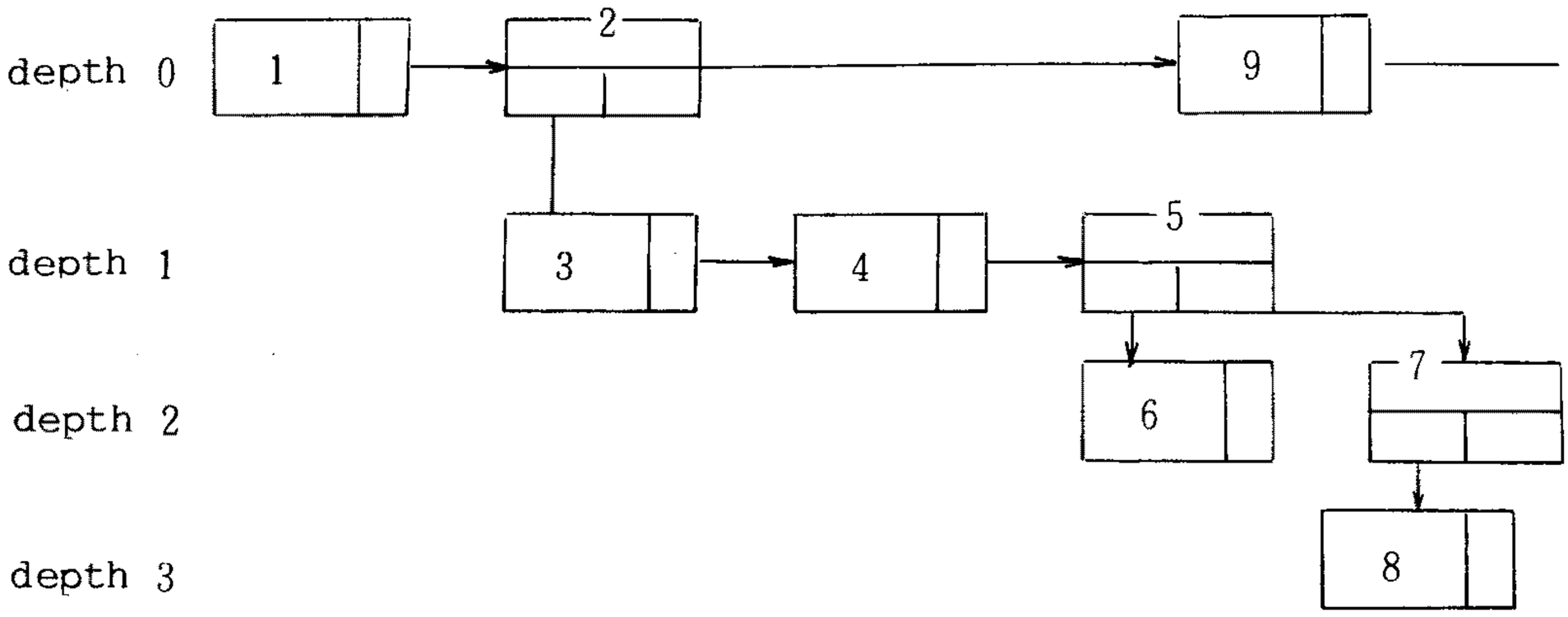
```
struct para_record {

    int para_seq_no;                /* 패러그래프  위치 */

char para_name[31];                 /* 노드  번호  */

int para_node_start;                /* 시작  노드  번호  */

int para_node_end;                  /* 끝  노드  번호 */

char para_fall_thru;                /* 순차  흐름  발생  여부 */

}
```

코볼의 경우 어휘적 근접 (Lexical adjacency) 이 발생할 때 순차적 흐름이 허용된다. 코볼 프로그램에서 순차 흐름이 형성되는 경우는 PROCEDURE DIVISION 부터 특정 패러그래프에서 STOP 이 발생하기 전과, 여러개의 패러그래프가 단위로 PERFORM되는 경우, 그리고 GOTO에 의해 제어지점 부터 순차 흐름이 발생하는 경우이다. 구문 분석시에는 노드간의 선후관계 연결이 패러그래프 단위로 이루어 진다. 구문 분석시에는 각 패러그래프간의 fallthru 여부를 판단하기 어렵기 때문에, 구문 분석후 fallthru 여부와 PERFORM,GOTO 문의 정보를 이용하여 fallthru에 의한 에지를 추가로 생성한다. 시작 노드와 끝 노드 번호는 이 시점의 노드 연결 관계 구성시 이용된다.

패러그래프내에서의 노드간 연결은 의미수행 과정에서 각 프로그램의 노드번호와 깊이를 부여한 후 [표 4-2], 선후 관계 연결이 재귀적으로 이루어진다. 노드가 저장될때 앞 노드의 형을 검사하여, 하나의 후행 노드를 가질 수 있는 경우에는 바로 선후 관계를 연결 하지만, 앞 노드가 여러개의 후행 노드를 가질 수 있는 경우에는 후행 노드를 일일이 검사하여 후행 노드번호에 따라 다시 형검사를 실시하고 후행노드번호가 없는 경우에는 노드를 연결하여 준다. 연결 순서가 [그림 4-11]에 나타나 있다.

[표 4-2]프로그램의 노드 구분후 노드 번호와 깊이를 부여한 형태

Line#	Node#	Depth	Type	
1	1	0	N	MOVE WORK-KEY TO MASTER-KEY
2	.	0		ADD INPUT-LENGTH TO TOTAL-LENGTH.
3	2	0	I	IF TOTAL-LENGTH > MAX-LENGTH
4	3	1	P	THEN PERFORM PRINT-NEW-PAGE
5	4	1	P	PERFORM PRINT-HEADER
6	5	1	I	IF CASE-A
7	6	2	N	THEN MOVE VARIABLE-A TO WORK-AREA1
8	.	2		MOVE VARIABLE-AA TO WORK-AREA2
9	7	2	I	ELSE IF CASE-B
10	8	3	N	THEN MOVE VARIABLE-B TO WORK-AREA.
11	9	0	N	MOVE WORK-CONTENT TO PRINT-CONTENT.
12		0		WRITE PRINT-R.



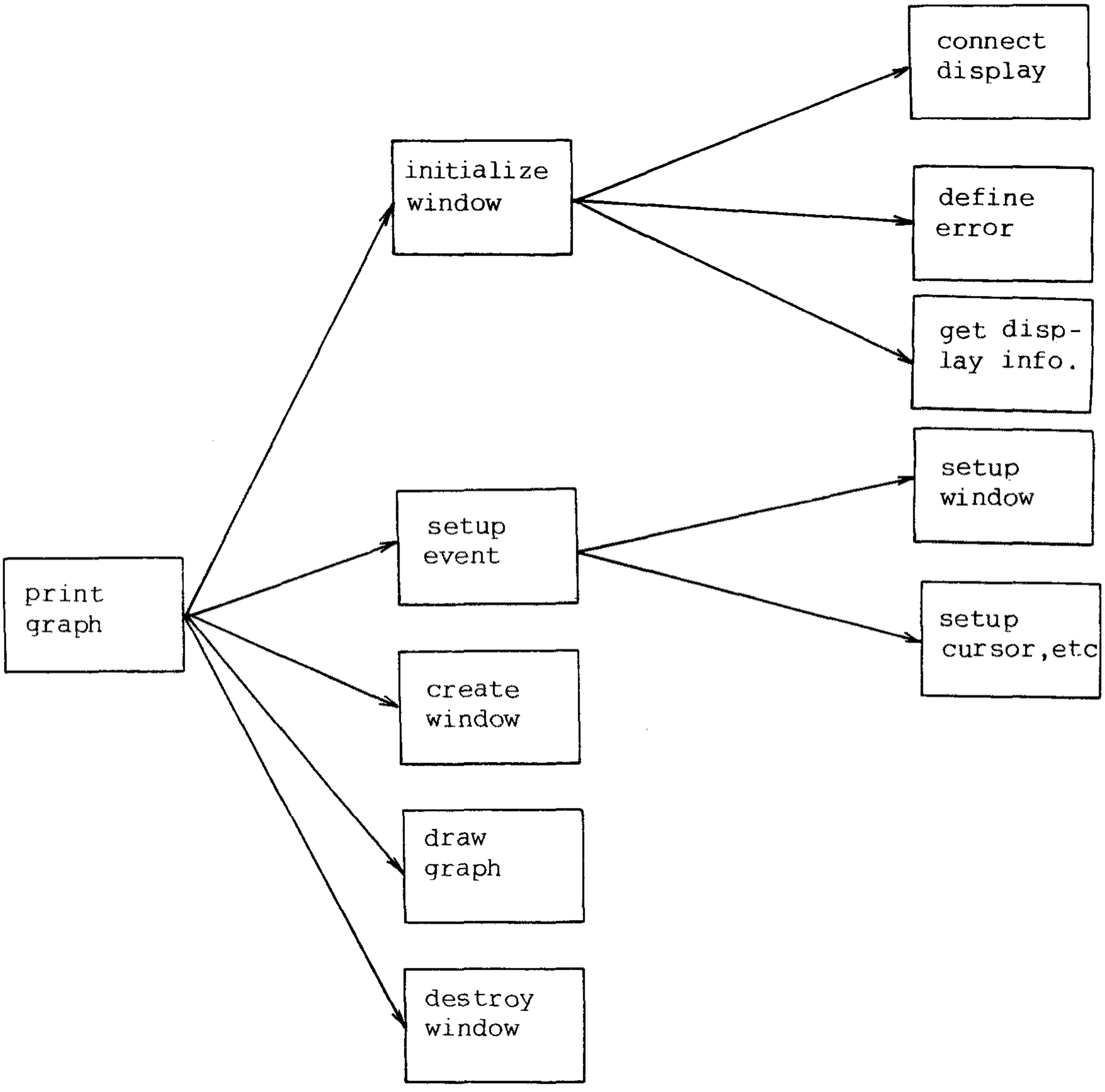
노드의 저장과 연결관계 구성

1. depth 0에 노드 1저장
2. 〃 노드 2 저장 …> 에지 1-2 발생
3. depth 1에 노드 3 저장
4. 〃 노드 4 저장 …> 에지 3-4 발생
5. 〃 노드 5 저장 …> 에지 4-5 발생
6. depth 2에 노드 6 저장
7. 〃 노드 7 발생 * …> 에지 5-6 발생 …> 노드 6 제거
8. 〃 노드 7저장
9. depth 3에 노드 8 저장 …> 에지 7-8 발생 …> 노드 8 제거
10. 5번 IF 문의 else part reduce …> 에지 5-7 발생 …> 노드 7제거
11. 2번 IF 문 then part reduce …> 에지 2-3 발생 …> 노드 3,4,5 제거
12. depth 0에 노드 9 저장 …> 노드 2번 검사
    에지 6-9 발생
    에지 8-9 발생
    에지 7-9 발생 (거짓 경로)
    에지 2-9 발생 (거짓 경로)

[그림 4-11] 노드간의 선후 관계 연결 동작

(나) 제어 구조의 출력

제어구조 출력시스템의 구조도는 [그림 4-12]와 같으며 각 모듈별 기능을 요약하면 다음과 같다.



[그림 4-12] 제어 구조 출력 시스템 구조도

1. print graph

• 제어 흐름 그래프출력의 주 프로그램으로 부 프로그램을 호출 한다. 호출 순서는 [그림 4-12]에 나타나 있다.

2. initialize window

• Display 를 open 하고, X 시스템의 default screen, screen depth, black pixel, white pixel 을 얻는다.

3. connect display

• display 를 connect 한다.

4. define error, get disp. infor

• error 가 발생하였을 때의 Display, Event type, Request, Resource 를 출력한다.

• 현재 font 의 attribute 를 구한다.

• 현재 X 의 Release, Version, Color plane depth, display 명, display width display height 를 출력한다.

5. setup window

• window 의 위치와 크기를 지정한다.

6. setup cursor, etc

• busy cursor 와 idle cursor 의 bitmap 을 생성하고, 생성된 bitmap 으로 Pixmap cursor 를 생성한 후. Pixmap cursor를 생성한 후, Pixmap cursor 로 cursor 를 정의한다. idle cursor 는 KIST 로고로 busy cursor 는 시계로 나타난다.

7. destroy window

• 작업 종료후 윈도우를 제거한다.

8. initialize data structure

• 노드의 depth, width, position 등의 계산 결과를 저장하기 위한 자료구조의 초기화

9. compute node width, depth

• 노드의 선후행 관계를 분석하여 Topological ordering 을 한 후 odering 결과에 따라 depth 를 부여하고 각 depth 의 node 갯수로 width 를 구한다.

10. fill nodes to node array

11. compute node position

• 노드의 depth 와 width 에 따라 화면상에서의 node 의 x,y 좌표를 구한다.

하나의 노드가 여러개로 분기하여, 다시 모일때 처음 노드와 마지막 노드의 x 좌표를 동일하도록 조정한다.

12. adjust node position

• selection 노드의 경우 then part 와 else part 중 한개가 나타나는 경우가 있다. 이 경우 노드 위치를 grow direction 모듈을 호출하여 조정한다.

13. grow direction

• selection 노드의 경우 then part 와 else part 중 한개가 나타나는 경우가 있다.

이때의 then(또는 else) 노드가 selection 노드의 왼쪽 (또는 오른쪽) 에 나타나도록 조정한다.

14. serialize node position

• 화면에서의 노드의 위치는 패러그래프 단위로 계산되며, 여러

개의 패러그래프의 노드들을 화면에 나타낼 때는 노드 위치의 조정이 필요하다.

각 노드의 위치에 패러그래프 시작점을 base로 더하여 실제 위치를 구한다.

15. draw graph

• draw edge와 draw node를 호출하여 그래프를 화면에 출력한다

16. draw edge

•두 노드 사이의 에지를 출력한다.

17. draw node

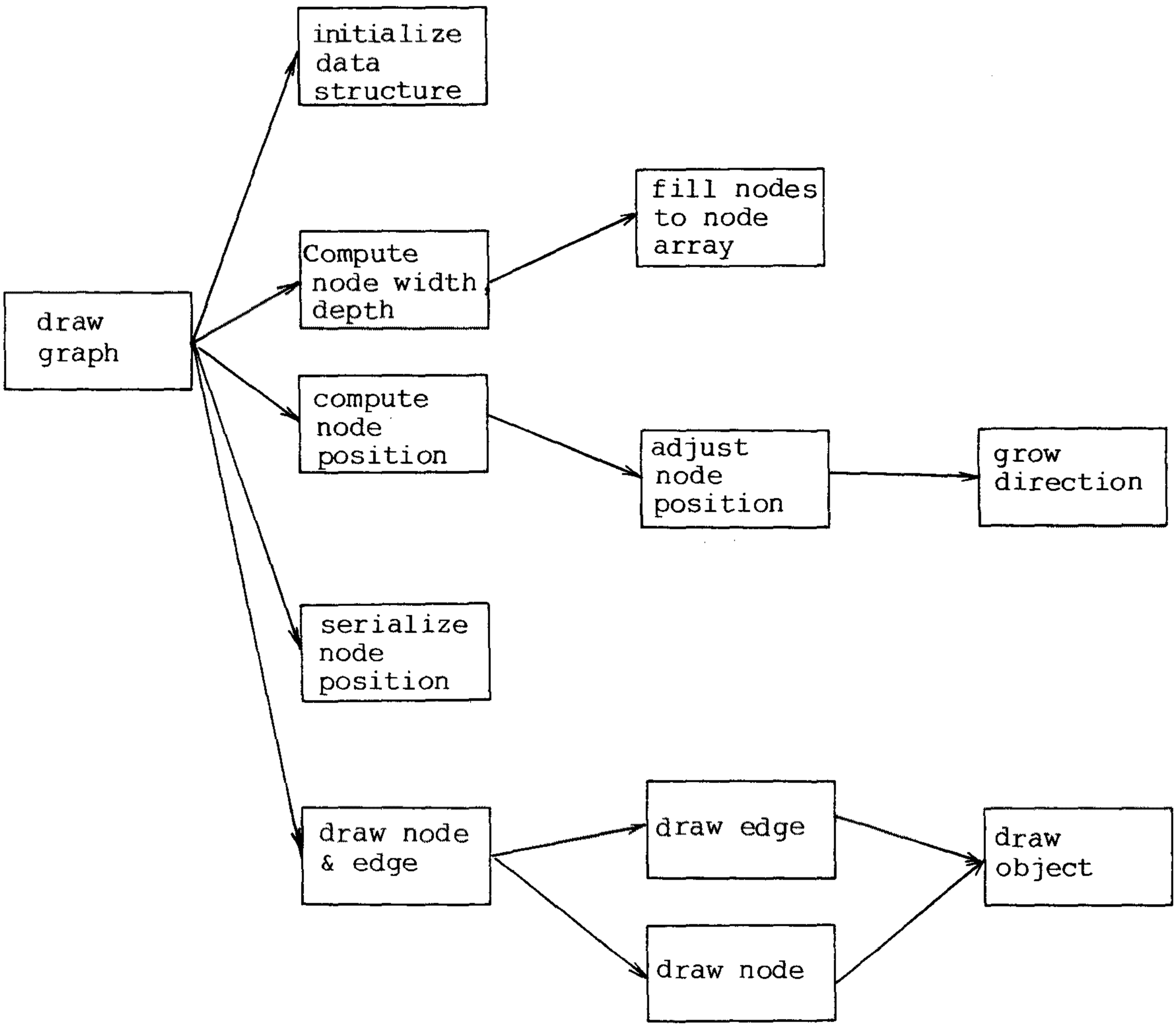
•에지의 유형에 따라 에지를 출력하고, 패러그래프 명을 출력하며, 에지 내부를 fill 한다.

18. draw object

• 인자 : object type(line, arc, rectangle, fill arc, fill rectangle), x1,y1,x2,y2, 시작 노드 정보 위치, 끝 노드 정보 위치

• x1,y1,x2,y2 좌표에 따라 object를 그린다. 화면 출력은 XDrawLine, XDrawRectangle, XFillRectangle, XFillArc를 이용하며, 지면 출력 fprintf함수를 이용한다. 시작 노드 정보 위치, 끝 노드 정보 위치는 지면 출력시 에지의 양 노드가 PERFORM 노드인 경우 에지 길이의 재계산에 사용된다.

draw graph 모듈의 구조도는 [그림 4-13]와 같다.



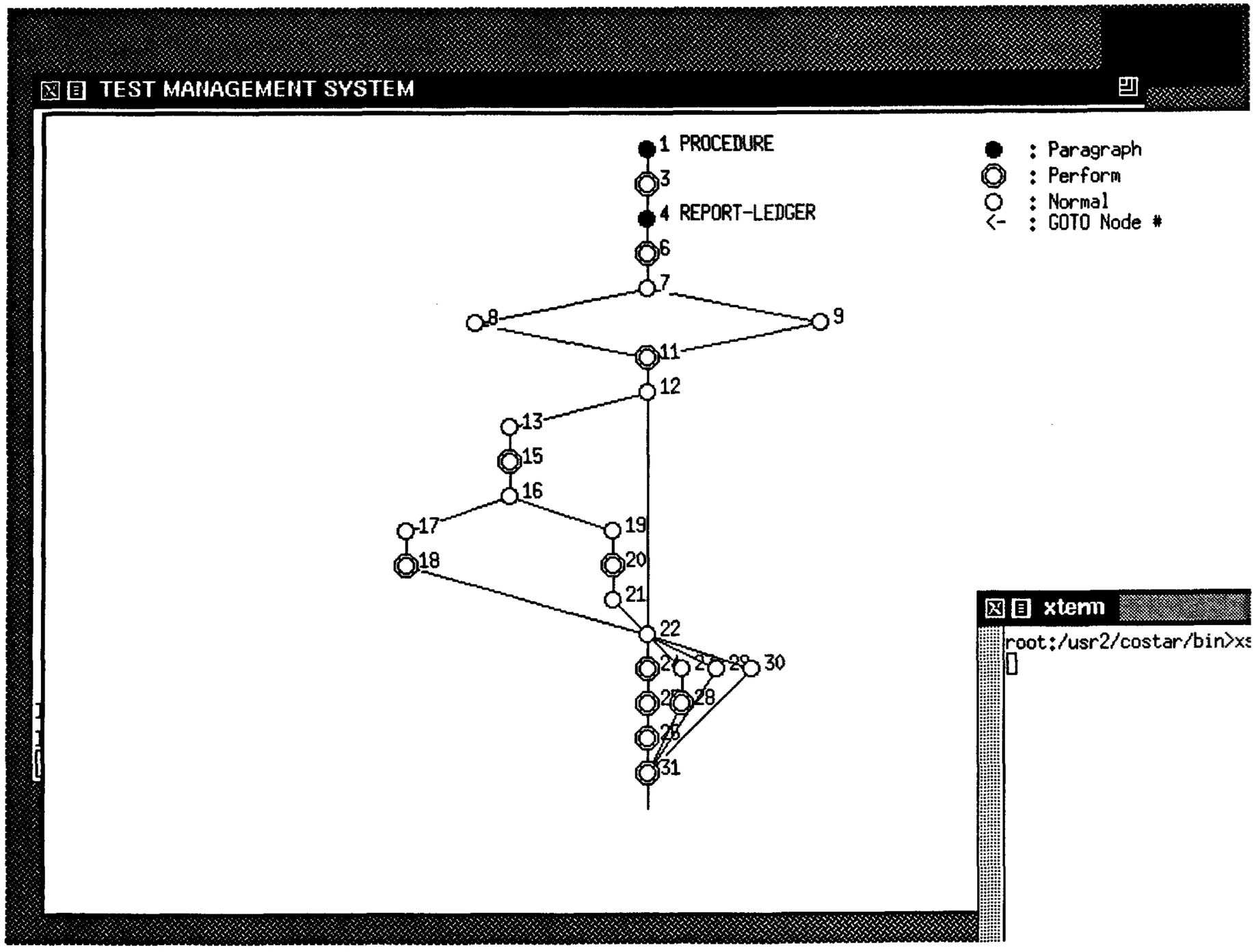
[그림 4-13] draw graph 모듈의 구조도

(다) 출력된 제어 그래프의 예

[그림 4-13]에 출력된 제어 그래프의 화면이 나타나 있다.

[그림 4-13]의 경우 PROCEDURE DIVISION에 "PERFORM"문이 있고 PROCEDURE DIVISION 다음 Paragraph는 Report Ledger이다.

Report Ledger 내에는 "PERFORM" 문에 이어 if-then-else 의 selection 구조가 있고 (node 7,8,9), "PERFORM" 문이 있으며 (node 11), 그 뒤에 depth 2 의 selection 구조가 나타난다. depth 1 에서는 false 경로가 없고 depth 2 에서는 true, false 경로가 나타난다. node 22 는 "EVALUATE" 문으로 When part 가 3 개, When other part 가 1 개인 경우를 나타낸다.



[그림 4 - 13] 제어 흐름 그래프 출력의 예

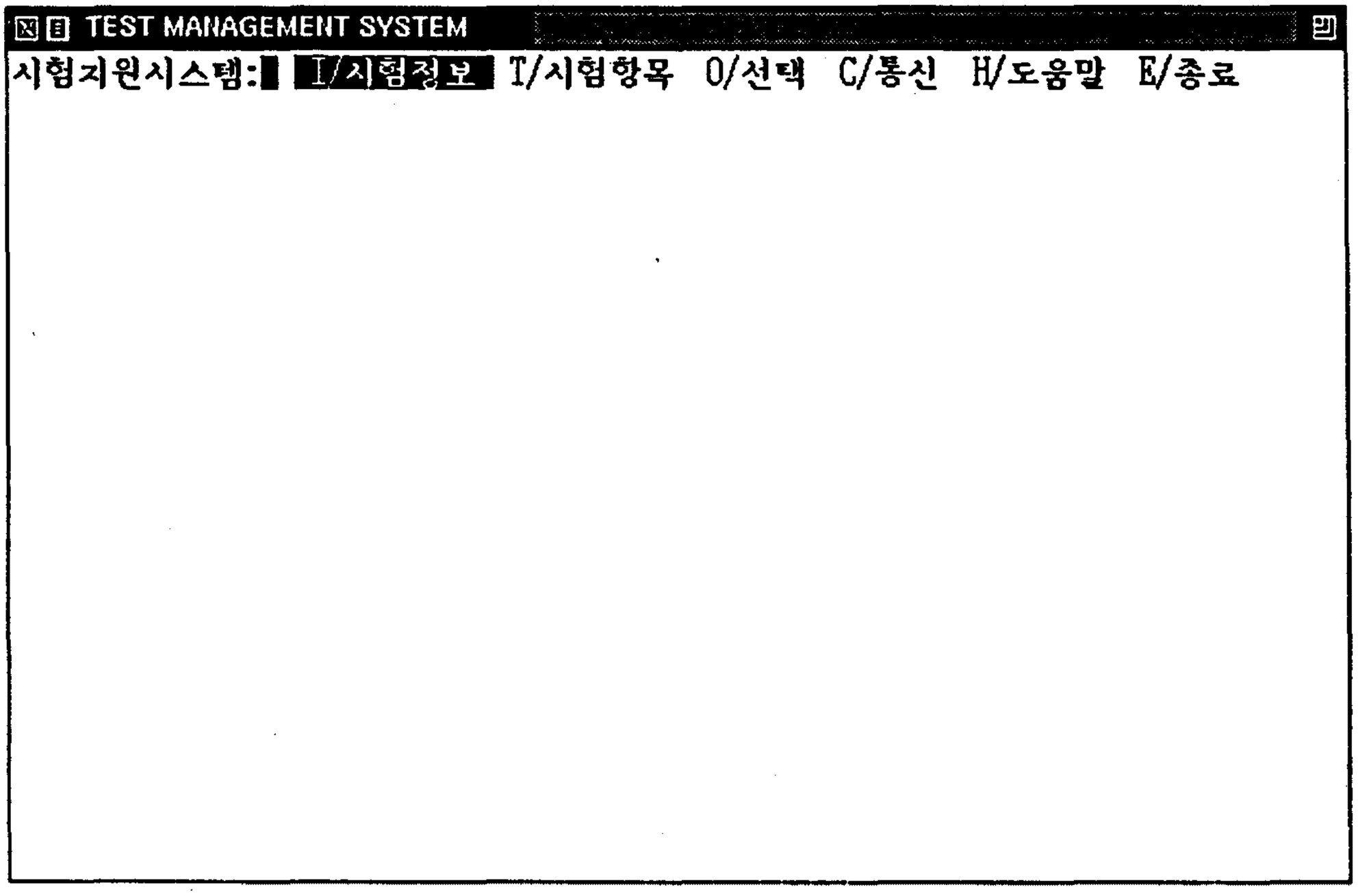
여 백

제 5 장　사용자 접속 설명

여 백

제 5 장　사용자 접속 설명

제 1 절 개요

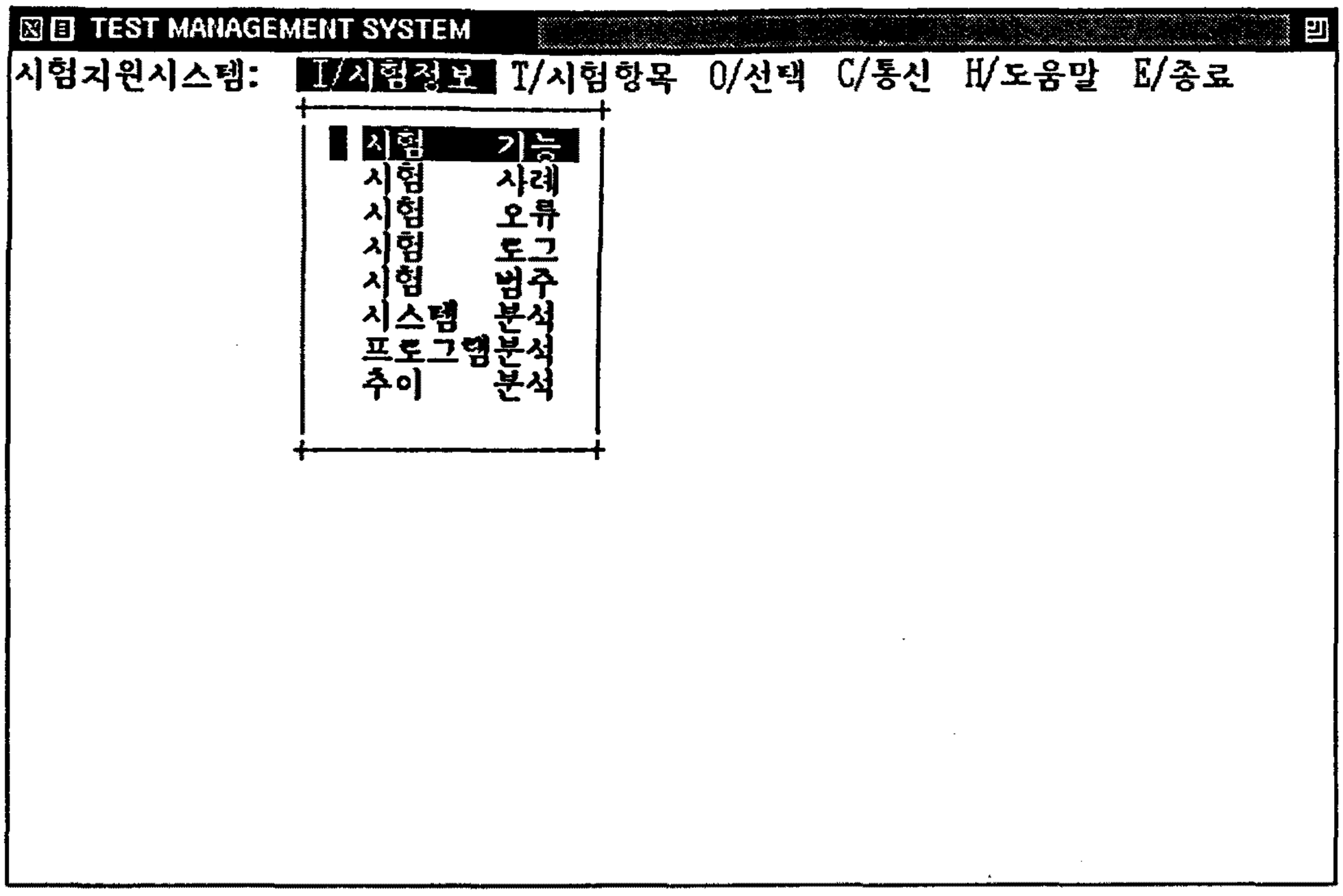


o 시험지원 시스템의 주 화면이다.

o 시험정보는 시험 기능, 사례, 절차, 범주 분석, 정적 분석 결과등을 조회하고 편집하는 메뉴이다.

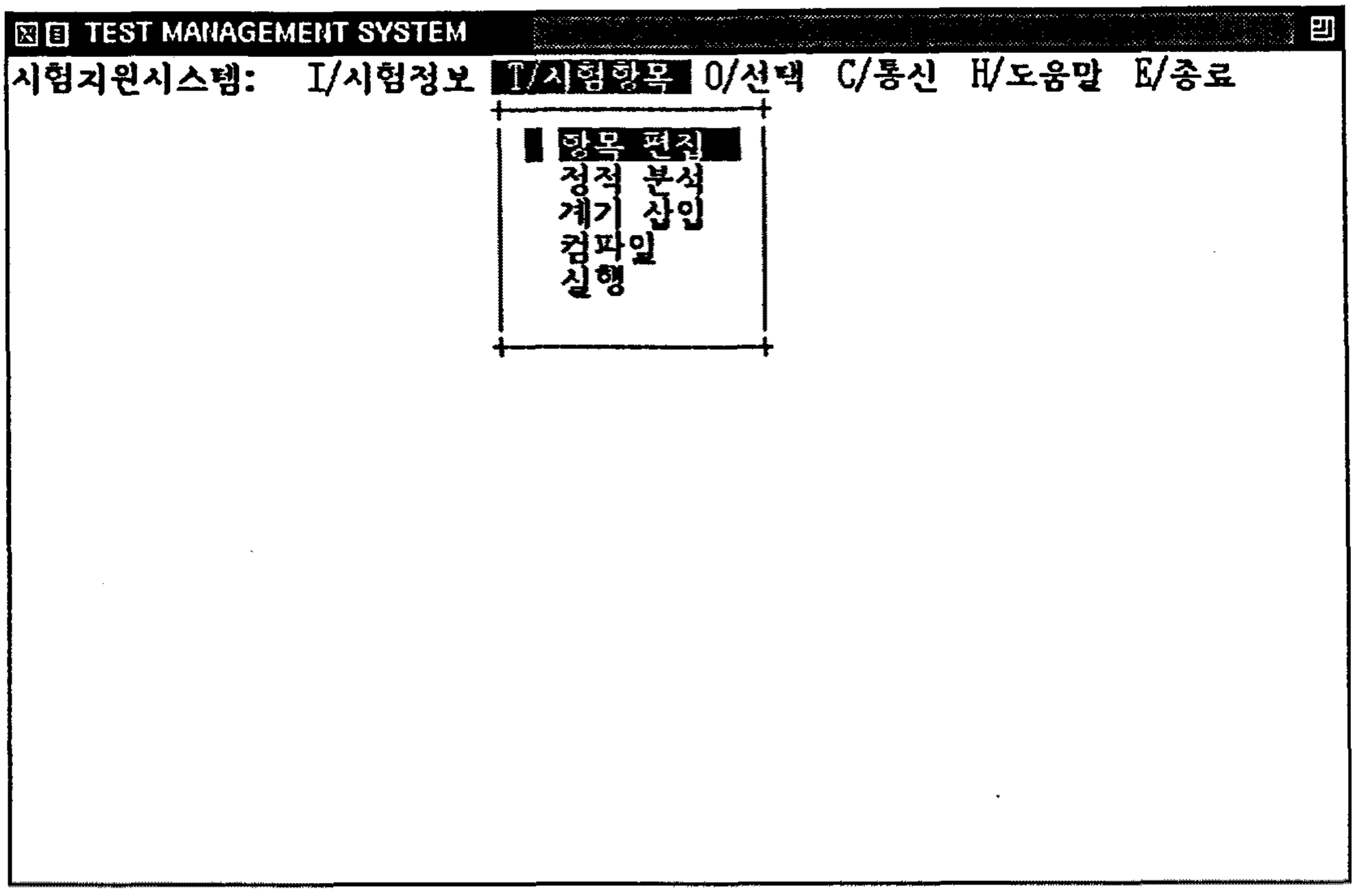
o 시험 항목은 시험 대상 프로그램을 조회, 편집할 수 있으며, 정적 분석, 계기 삽입 작업을 할 수 있다.
정적 분석 또는 계기 삽입이 완료된 프로그램을 이 메뉴에서 컴파일하거나 수행 가능하다.

o 선택에서는 각종 초기값을 지정한다.



o 시험 정보의 상세 메뉴이다.

o 각 항목을 cursor를 움직여 선택하면 부 메뉴가 나타난다.

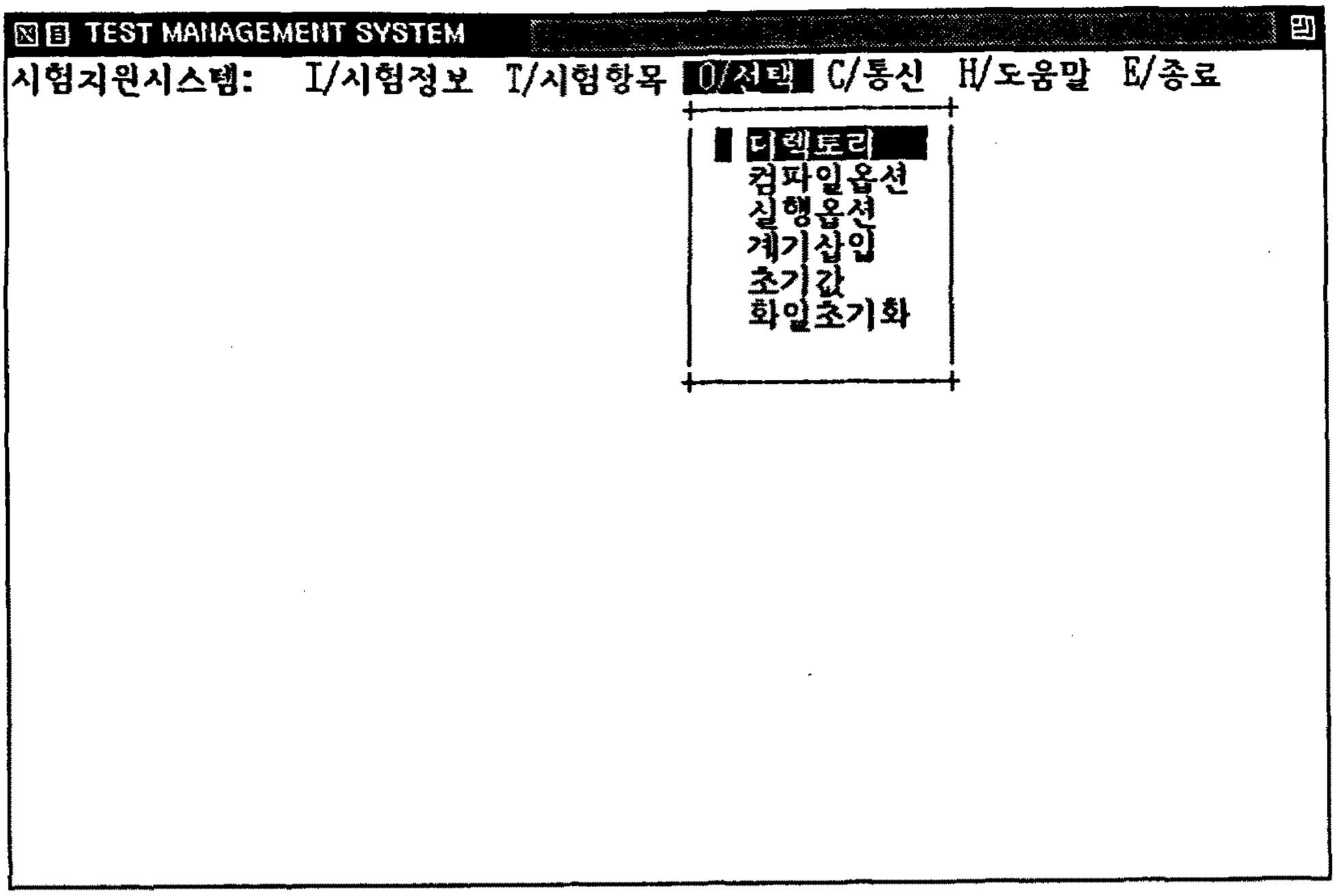


○ 시험 항목 관리 메뉴로 항목 편집을 선택하여 프로그램을 조회, 편집할 수 있다.

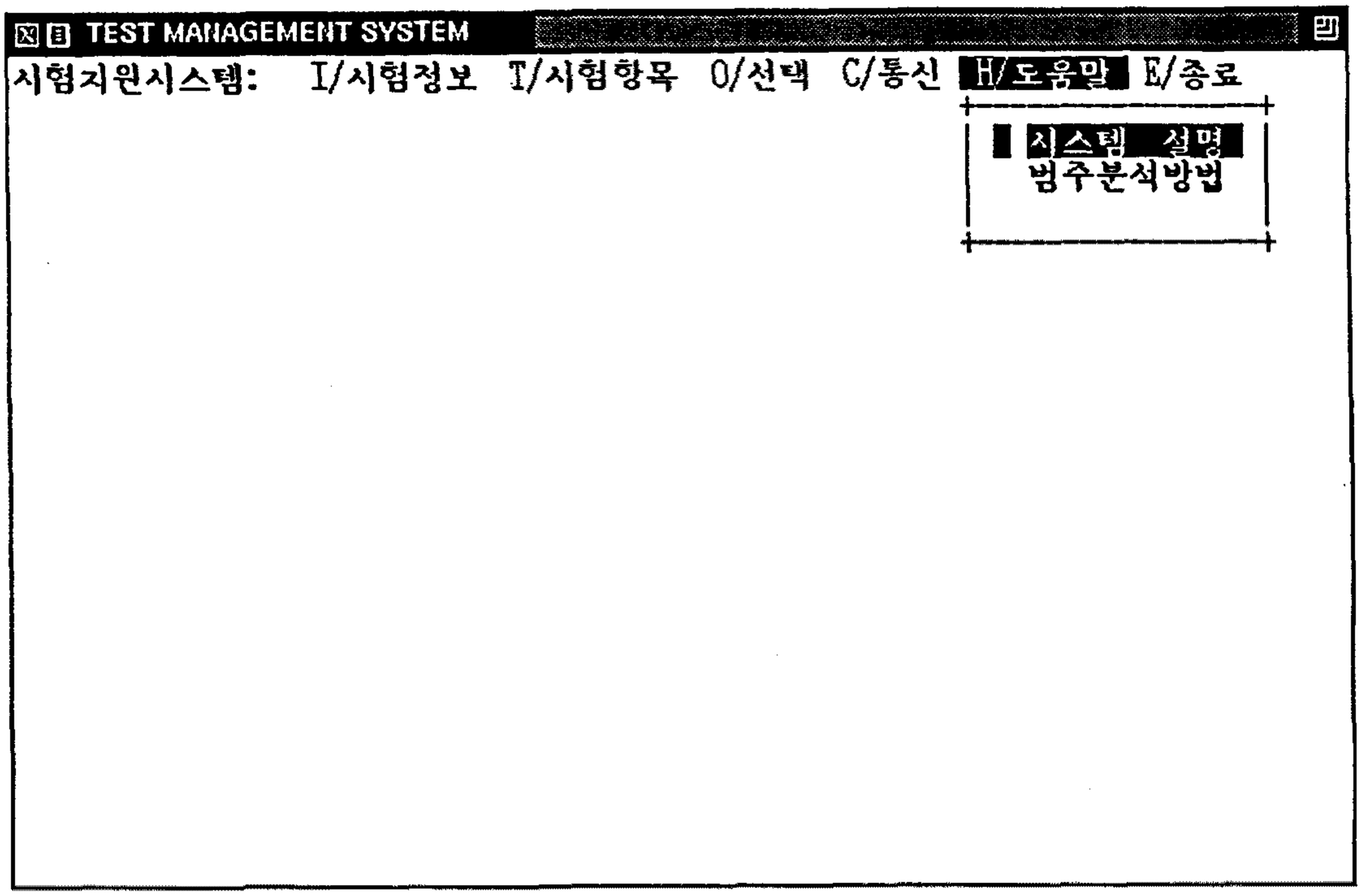
○ 정적 분석시에는 시스템 분석 정보와, 프로그램 분석 정보가 저장되어 주 메뉴의 시험 정보를 통해 조회 가능하다.

○ 범주 분석을 하기 위해서는 계기 삽입 후 프로그램을 컴파일 한다.

○ 정적 분석과, 계기 삽입, 컴파일을 수행하면 동작화면이 나타나고 실행을 선택하면 선택된 수행 모듈로 제어가 전이된 후, 프로그램 종료시 다시 이 화면으로 복귀된다.

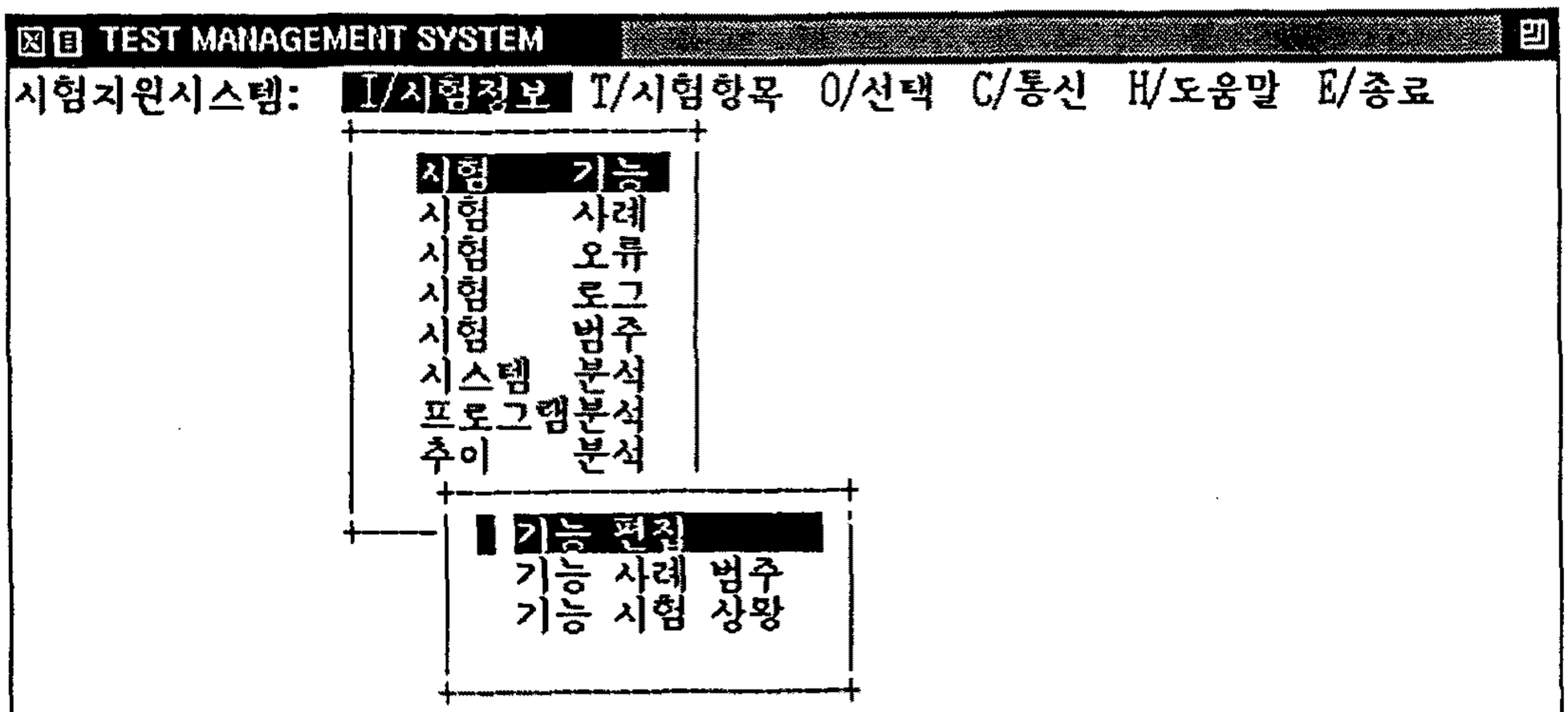


- 작업 수행시 필요한 옵션과 초기값을 지정해 주는 메뉴이다.
- 디렉토리 : 원시 프로그램, 수행 모듈 등의 디렉토리를 지정한다.
- 옵션 : 코볼 프로그램 컴파일, 수행시 공통 옵션을 지정한다.
- 화일 초기화 : 시험 도중, 프로그램이 수정되었을 때 누적 결과를 초기화 한다.



○ 시스템 개요와 범주 분석 절차를 안내하는 메뉴이다.

제 2 절 시험 절차 관리기



○ 이 화면은 시험 기능에 대한 선택 화면이다.

○ 기능 편집

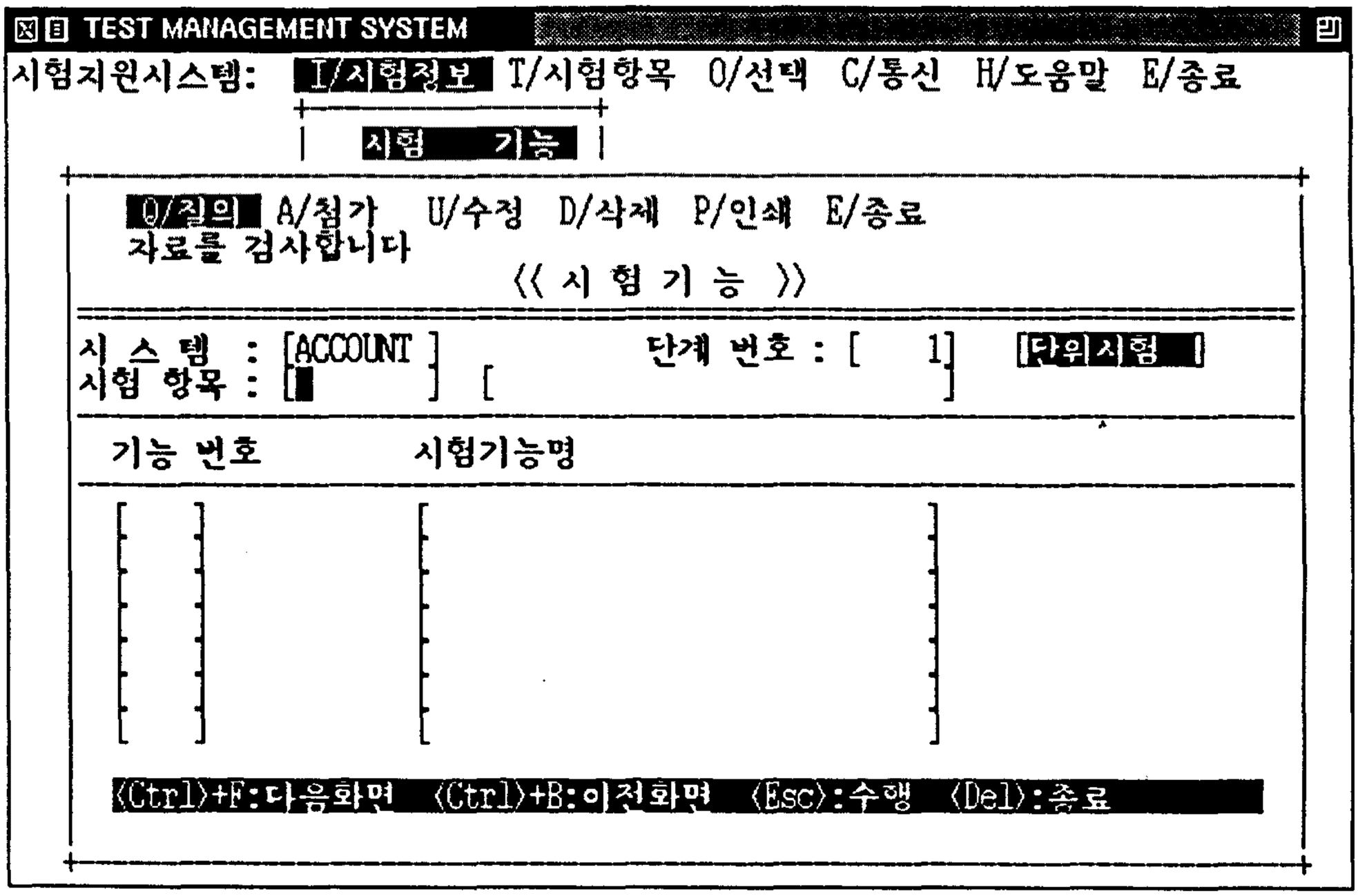
- 시험 기능을 등록, 수정, 삭제, 검색, 인쇄하는 관리 화면이다.

○ 기능 사례 범주

- 시험 기능의 수행 여부를 보여준 후 시험 기능에 해당되는 시험 사례들을 보여준다.

○ 기능 시험 상황

- 수행된 시험 기능, 미시험 기능을 보여준다.

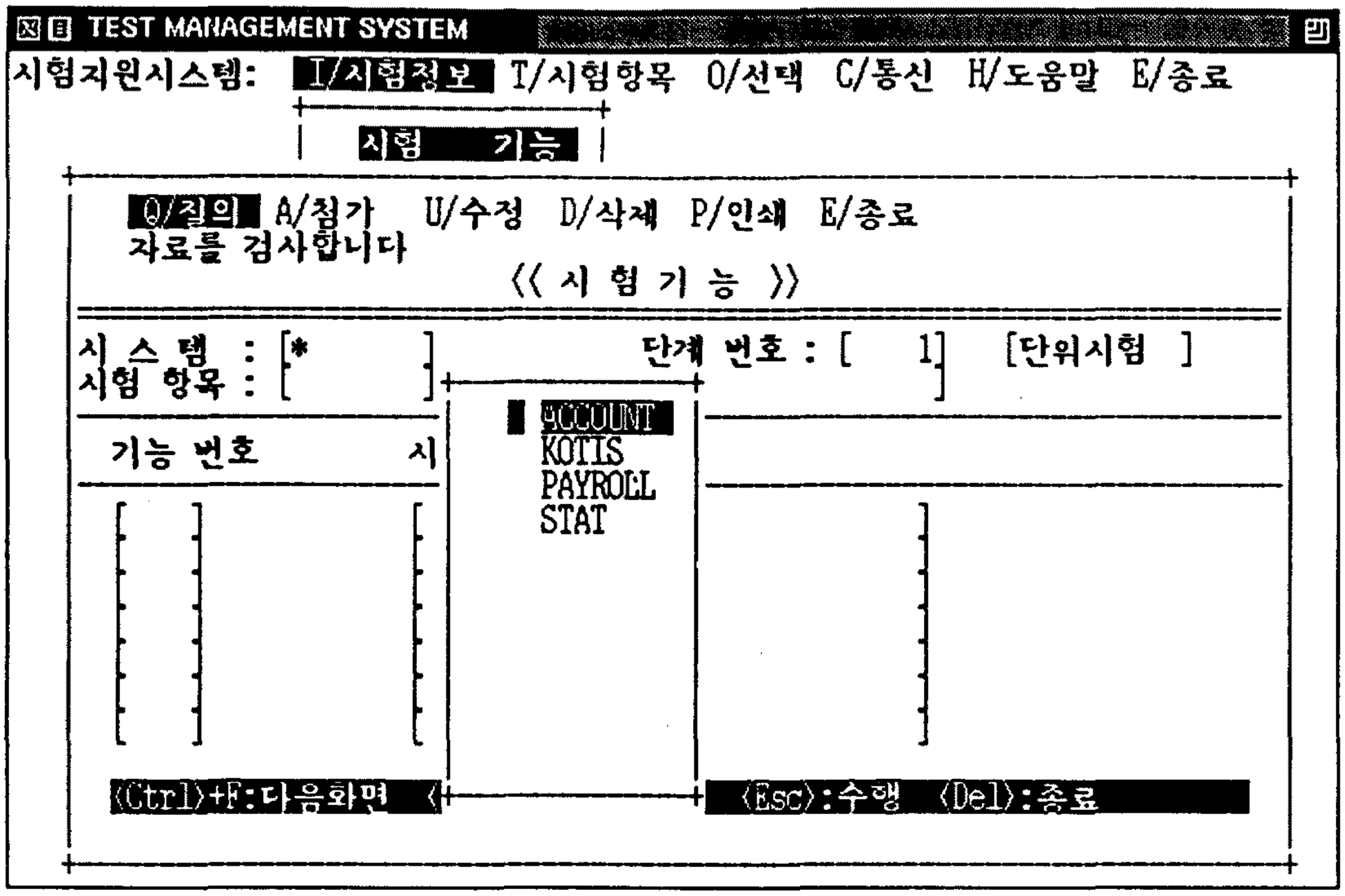


o 이 화면은 시험 기능 관리 화면이다.

o 메뉴 화면에서 기능 편집을 선택하면 이 화면이 나타난다.

o Cursor의 위치는 화면상의 메뉴중 "Q/질의"에 Reverse된다.

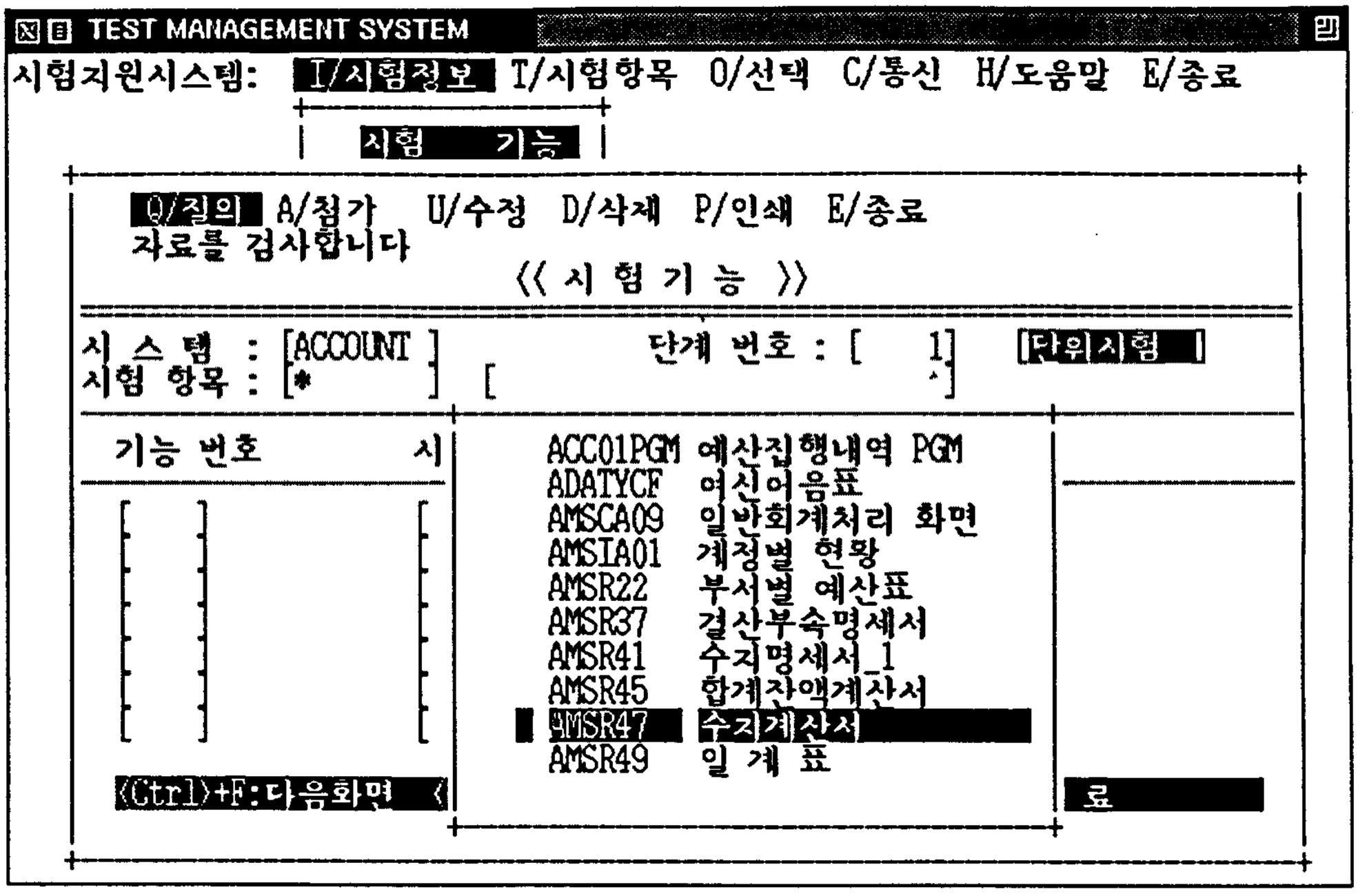
o 시스템 단계 번호는 주 메뉴 "O/선택"에서 입력된 초기값을 화면상에 보여준다.



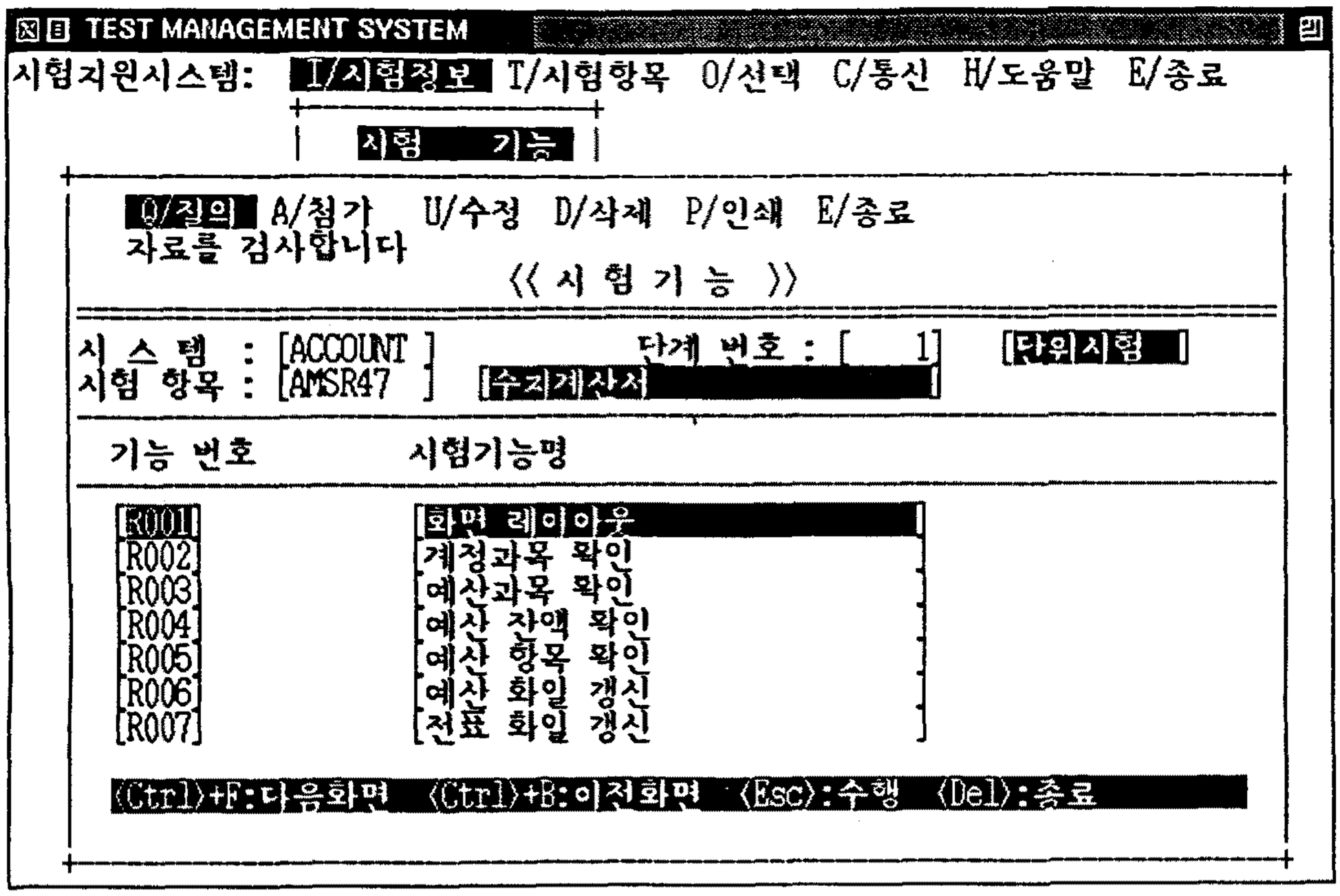
o 이 화면은 전 화면에서 시스템을 선택하는 화면이다.

o 시스템을 선택한 후 〈Return〉 Key 를 누른다.

- 초기값에서 지정된 시스템이 Default 값으로 나타난다.
- 다른 시스템을 선택할 경우
  - 시스템을 알고 있을 경우
    - 시스템을 직접 입력한다.
  - 시스템을 모르는 경우
    - 해당 필드에 "*"를 입력한 후 〈Return〉 Key 를 누른다.
    - 방향 Key 를 이용하여 원하는 시스템에 Cursor 를 이동시킨 후 〈Return〉 Key 를 누른다.
- 다음 화면 : 〈Ctrl + F〉 를 누른다.
- 이전 화면 : 〈Ctrl + B〉 를 누른다.



- 이 화면은 시험 기능 편집 화면중 시험 항목을 선택하는 화면이다.
- 시스템을 선택하는 화면과 같이 시험 항목을 직접 입력하거나 도움 화면을 통하여 시험 항목을 선택한다.



○ 이 화면은 시험 기능 편집 화면에서 "Q/질의"한 화면이다.

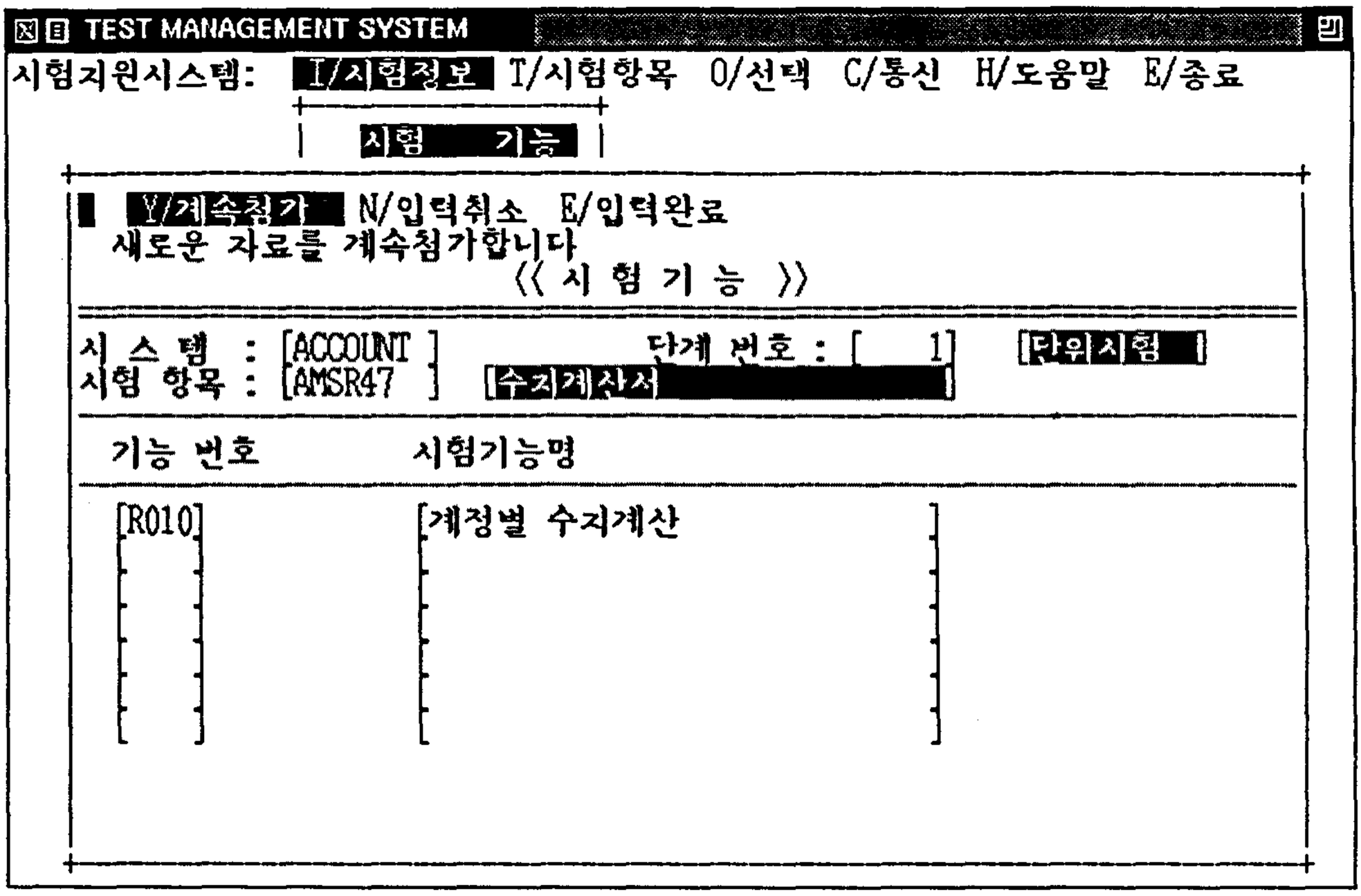
○ 시스템, 단계 번호, 시험 항목을 선택한 후 〈Esc〉Key를 눌러 관련된 시험 기능들이 화면에 질의된 상태를 나타낸다.

○ 검색된 자료

- 다음 화면 : 〈Ctrl + F〉Key를 누른다.
- 이전 화면 : 〈Ctrl + B〉Key를 누른다.
- 방향키 : 한행씩 이동한다.

○ 검색 작업을 종료할 경우

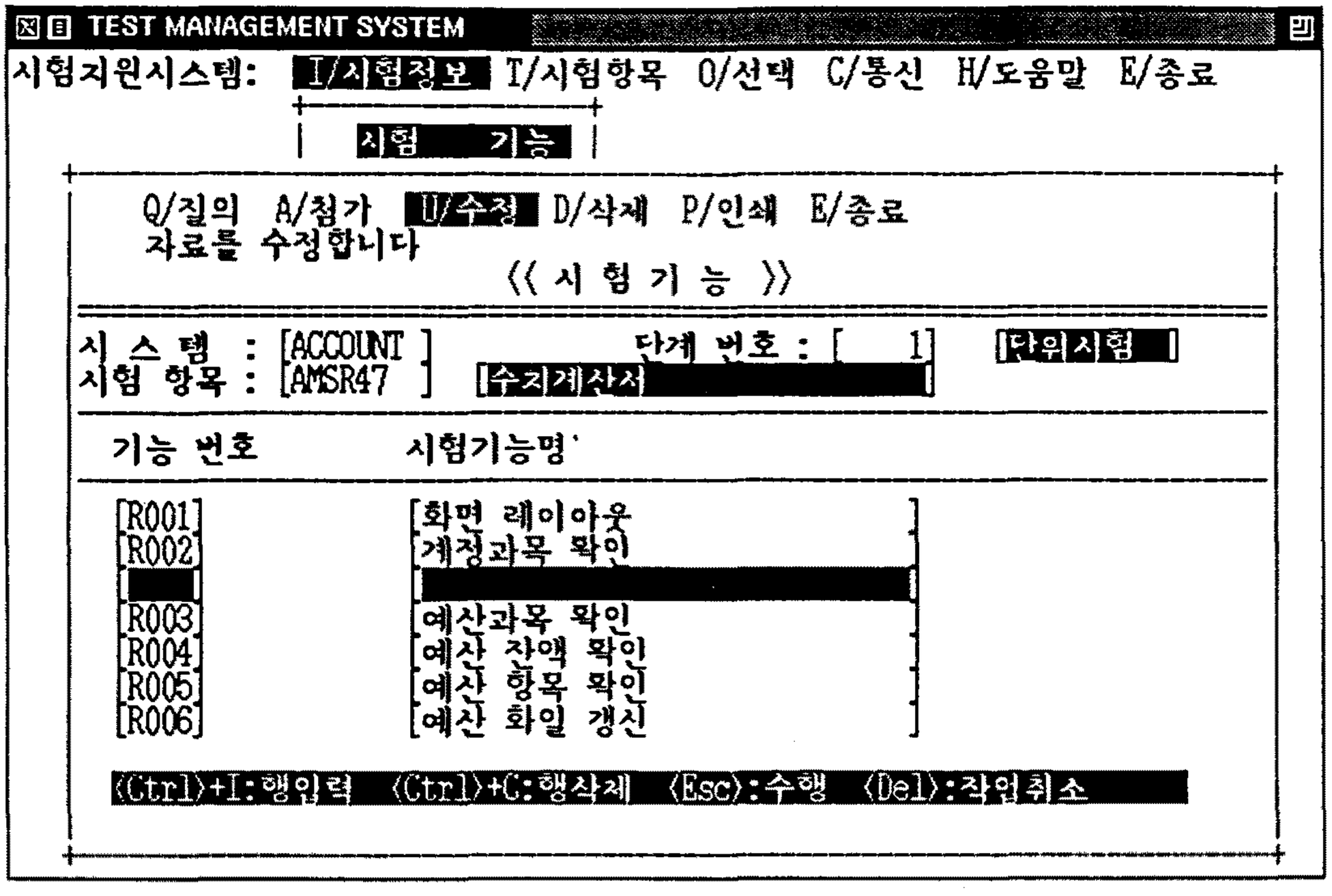
- 〈Del〉Key를 누른다.
- 검색된 내용이 지워지고 메뉴 "Q/질의"로 Cursor가 이동된다.



○ 이 화면은 시험 기능 편집 화면중 " A/첨가"를 선택한 화면이고 시스템, 단계 번호, 시험 항목에 새로운 내용들을 추가하는 화면이다.

○ 시험 기능을 입력했을 경우 부 메뉴가 보여진다.

- " Y/계속 첨가" : 새로운 기능을 계속 등록
- " N/입력 취소" : 새로 추가한 기능을 등록하지 않는다.
- " E/입력 완료" : 새로 추가한 기능들을 등록한다.



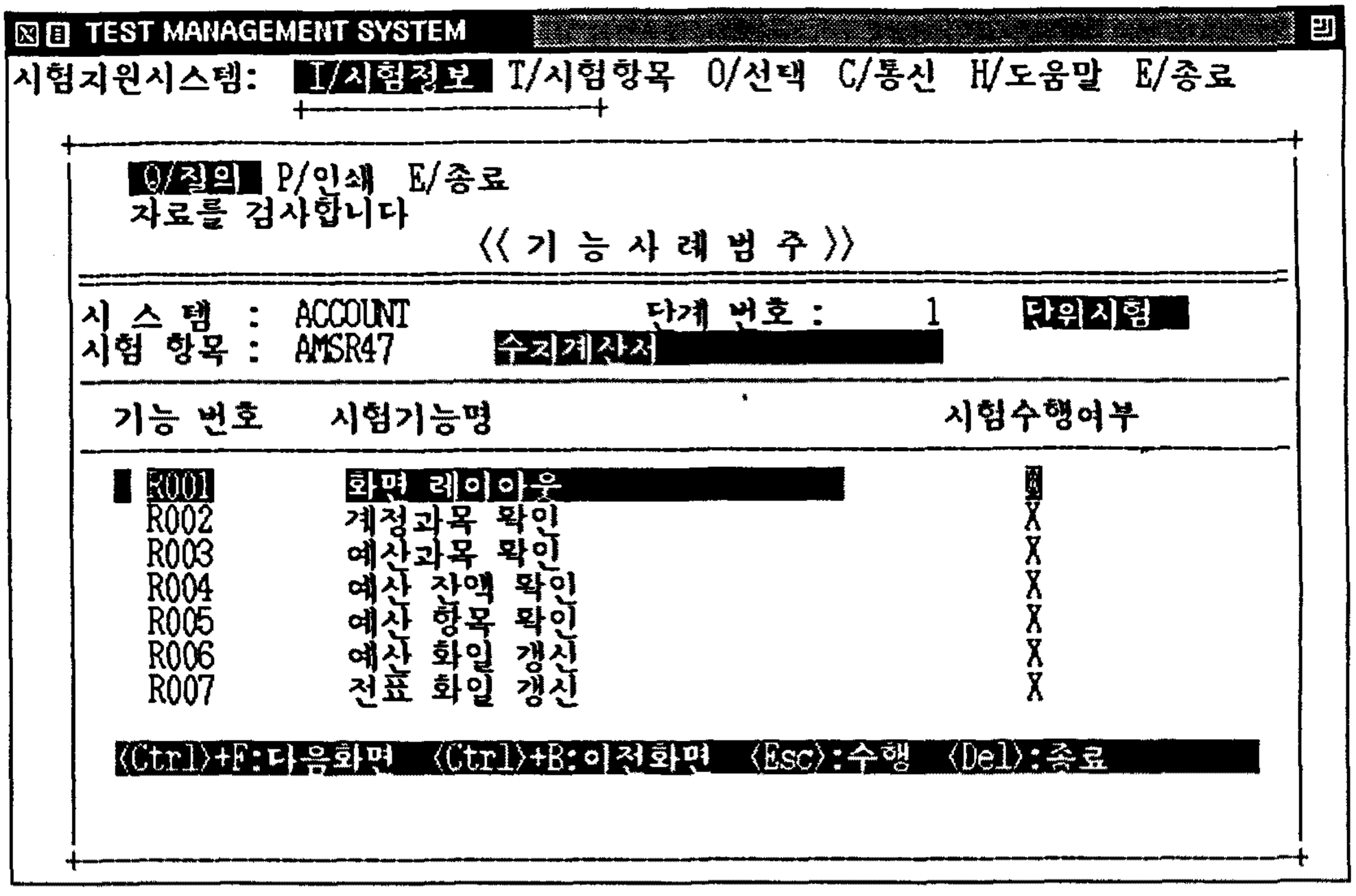
o 이 화면은 시험 기능 편집 화면중 "U / 수정 " 화면이다.

o 시스템, 단계 번호, 시험 항목에 대한 시험 기능의 각각을 수정 삭제, 추가를 할 수 있다.

- 새로운 기능을 추가 할 경우 : 〈Ctrl〉+I Key 를 누른다.
- 기능을 삭제할 경우 : 〈Ctrl〉+C Key 를 누른다.
- 기능들을 수정할 경우 : 시험 기능명을 수정한다.
- 추가, 삭제, 수정을 실행할 경우 : 〈Esc〉 Key 를 누른다.
- 추가, 삭제, 수정을 취소할 경우 : 〈Del〉 Key 를 누른다.

o 메뉴의 "D : 삭제 " 의 경우와의 차이점

- "D : 삭제 " 시 : 시스템, 단계 번호, 시험 항목에 대한 전 시험 기능을 삭제한다.
- "U : 수정 " 시 삭제 : 시스템, 단계 번호, 시험 항목에 대해 선택한 기능만 삭제한다.

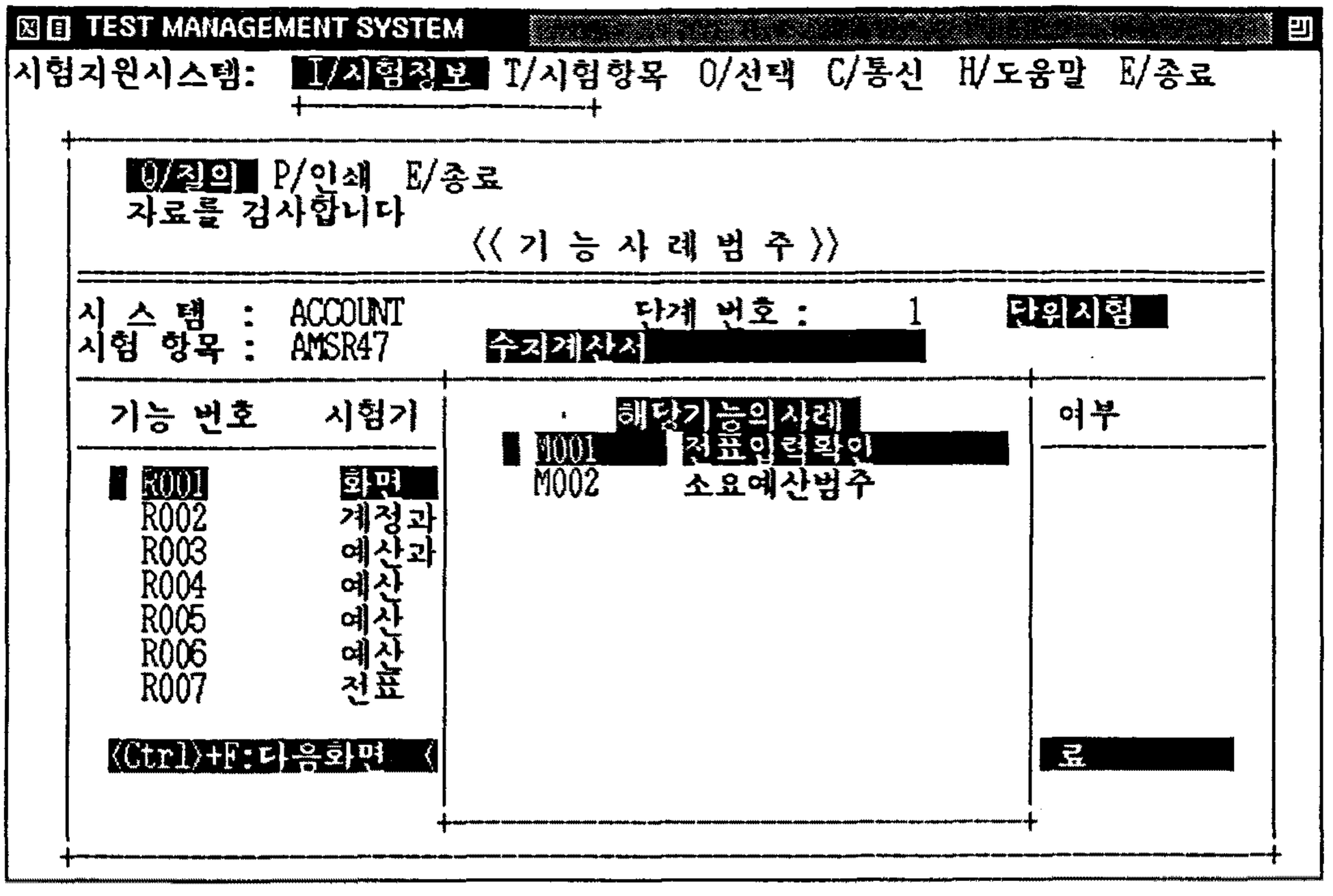


o 이 화면은 기능 사례 범주 화면이다.

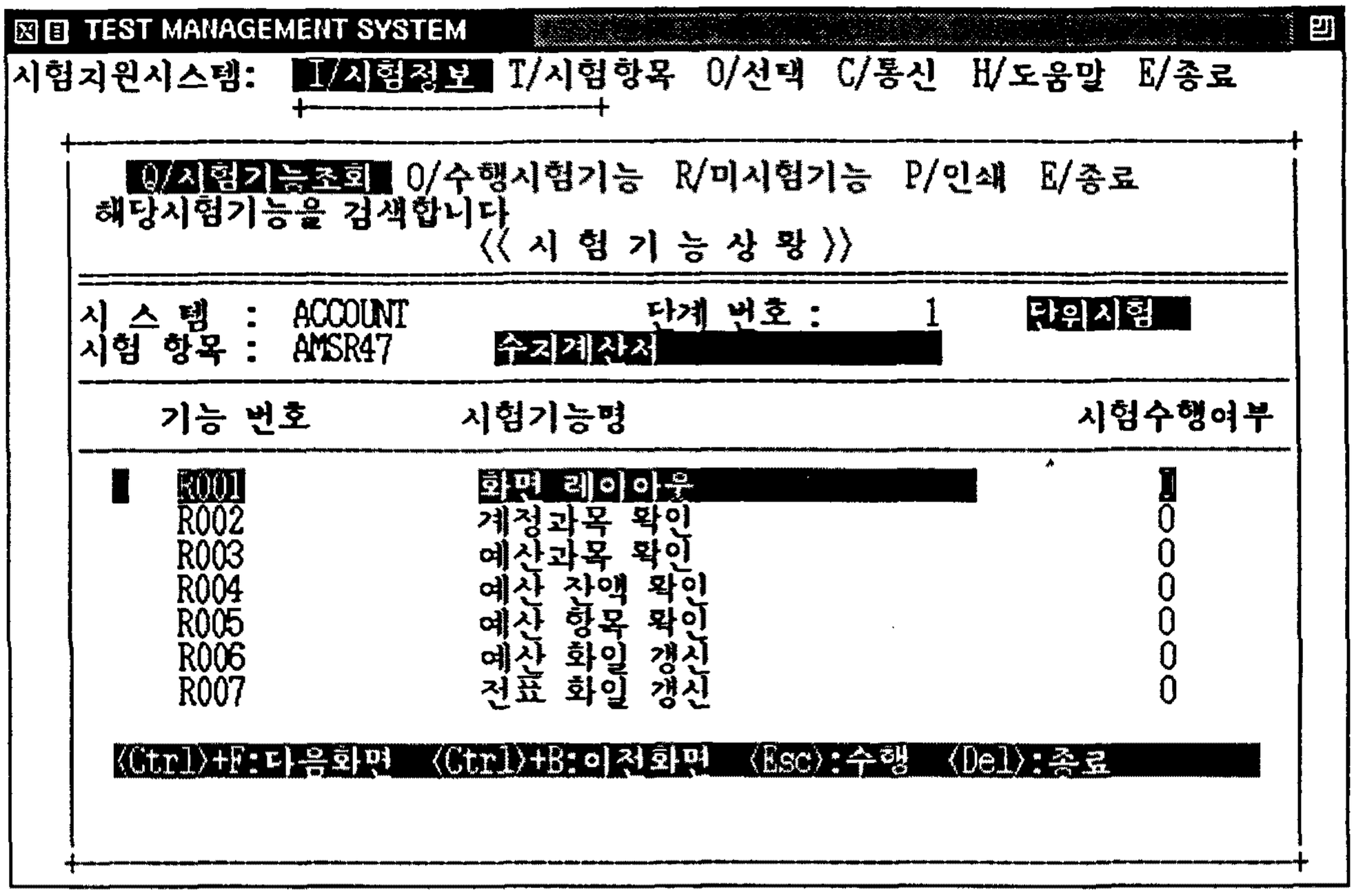
o 시스템, 시험 단계, 시험 항목에 해당되는 시험 기능들의 수행, 미 수행 여부를 보여준다.

o 시험 기능에 관련된 시험 사례를 보여준다.

- 원하는 시험 기능에 Cursor를 옮긴 후 〈Return〉 Key를 누르면 시험 기능에 관련된 시험 사례를 보여준다.

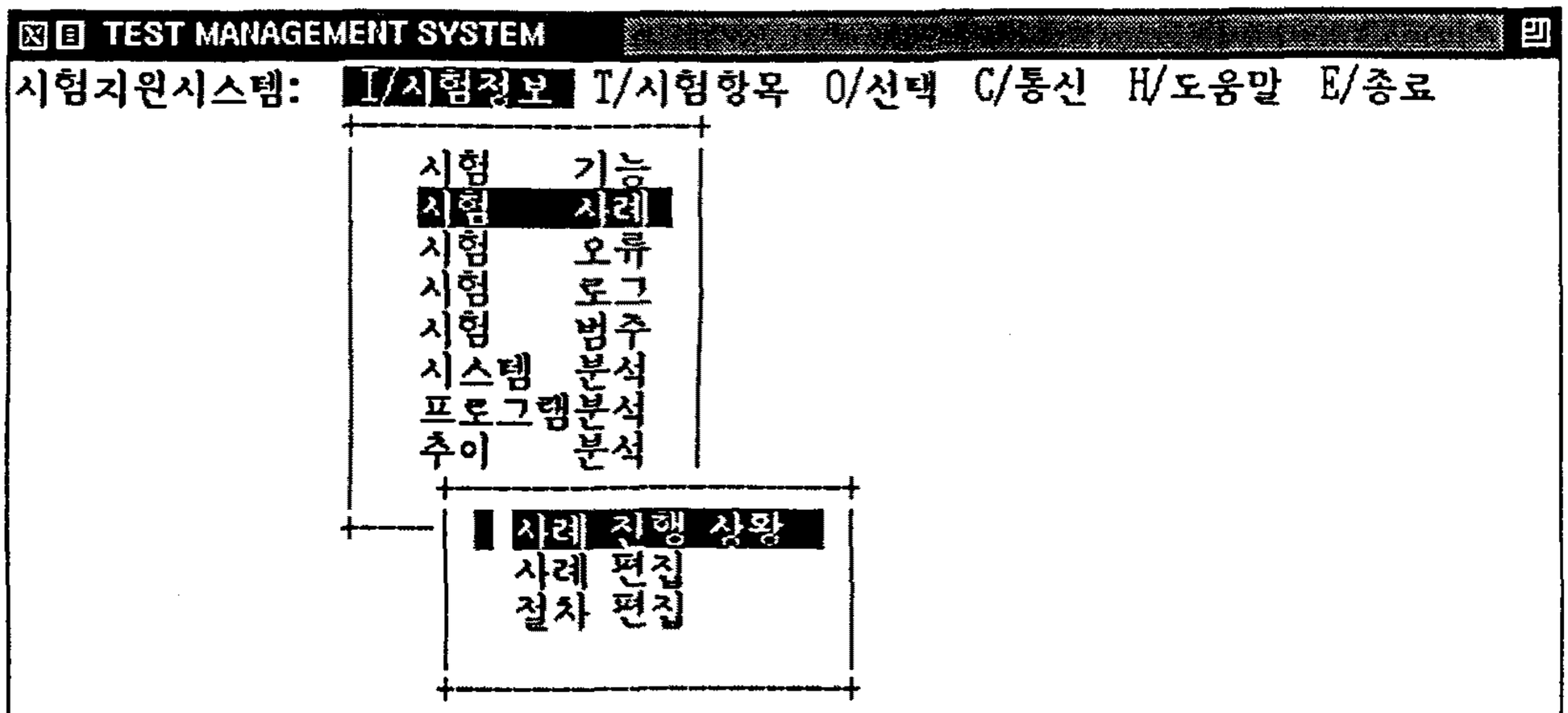


○ 이 화면은 기능 사례 범주 화면에서 기능을 선택한 후 기능을 시험하기 위한 사례를 보여주는 화면이다.



o 이 화면은 기능 시험 상황 화면이다.

- 전체 기능, 미 시험 기능, 수행된 시험 기능을 선택하여 검색한다.
- "Q/시험 기능 조회" : 모든 시험 기능을 검색한다.
- "O/수행 시험 기능" : 시험 기능이 수행된것만 검색한다.
- "R/미 시험 기능" : 미 시험 기능만 검색한다.
- "P/인쇄" : 시험 기능을 인쇄한다.
- "E/종료" : 이 화면을 선택한 전 메뉴 화면으로 돌아간다.



o 이 화면은 시험 정보 중에서 시험 사례를 선택한 화면이다.

o 사례 진행 상황

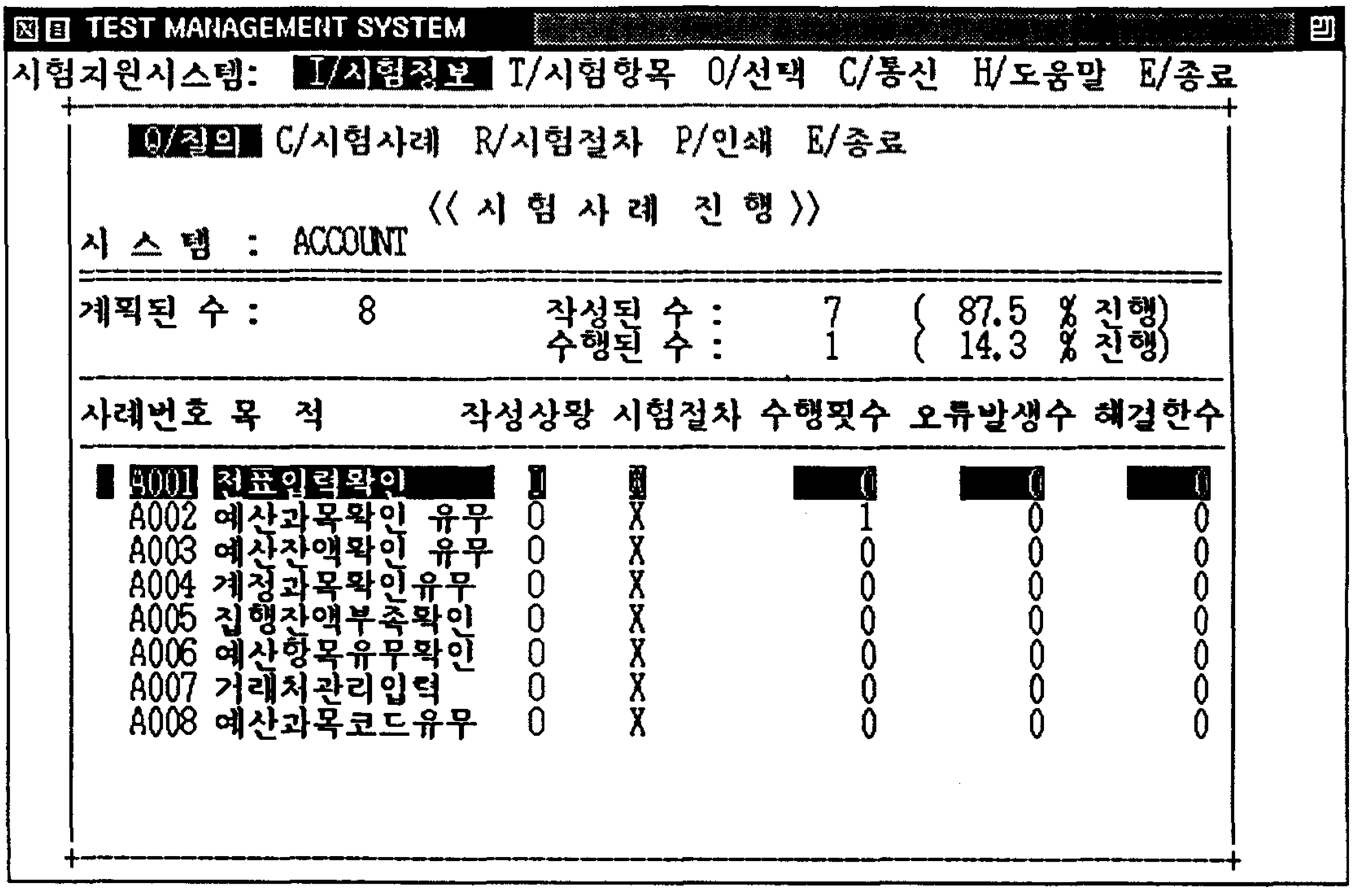
- 시험 사례에 대한 수행 및 오류 발생 상황을 보여주는 화면인다.

o 사례 편집

- 시험 사례를 등록, 수정, 삭제, 검색, 인쇄를 하는 관리 화면이다.

o 절차 편집

- 시험 절차를 등록, 수정, 삭제, 검색, 인쇄를 하는 관리 화면이다.



○ 시험 사례 진행 상황을 조회하는 화면으로 시스템을 선택한 후 〈ESC〉 Key를 누른다.

○ 시스템 안내 화면은 시험 기능과 같은 동작으로 접근한다.

○ 필드 설명

• 계획된 수 : 선택된 시스템에 대한 시험 사례를 계획한 수

• 작성된 수 : 선택된 시스템에 대한 시험 사례를 작성한 총수

• 수행된 수 : 선택된 시스템에 대한 시험 사례를 수행한 총수

• 사례 번호 : 시험 사례 번호

• 목 적 : 시험 사례에 대한 목적

• 작성 상황 : 시험 사례의 작성 유무

• 시험 절차 : 시험 사례에 대한 시험 절차 작성 유무

• 수행 횟수 : 시험 사례가 수행된 횟수

• 오류발생 수 : 시험 사례 수행중 발생한 오류수

• 해결한 수 : 오류발생 수 중에서 해결된 수

• 검색된 자료

- 다음 화면 : 〈Ctrl+F〉Key 를 누른다.

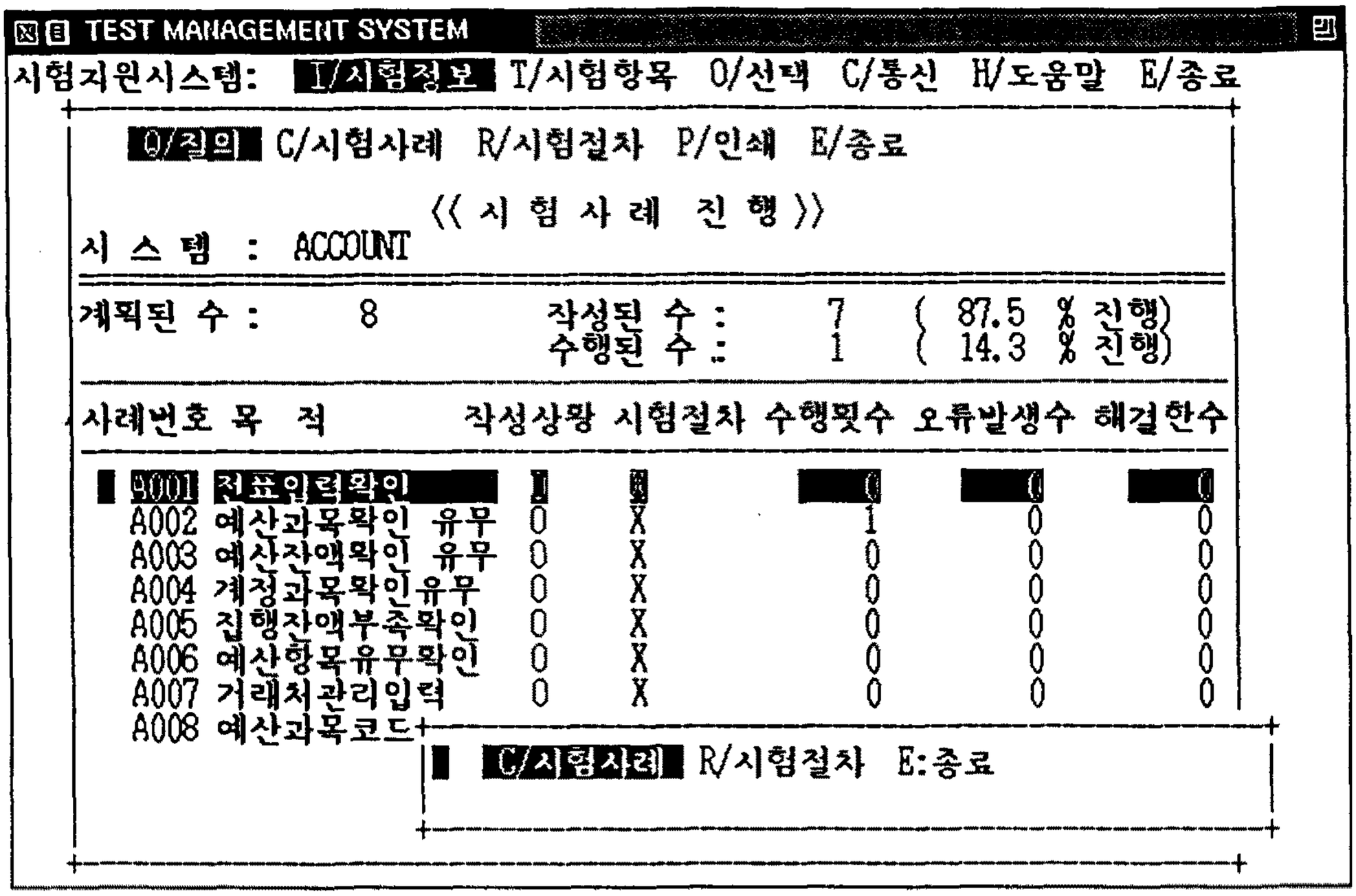
- 이전 화면 : 〈Ctrl+B〉Key 를 누른다.

- 방향키 : 한행씩 이동한다.

• 검색 작업을 종료할 경우

- 〈Esc〉 Key를 누른다.

- 검색한 내용을 지우고 메뉴로 Cursor가 이동한다.



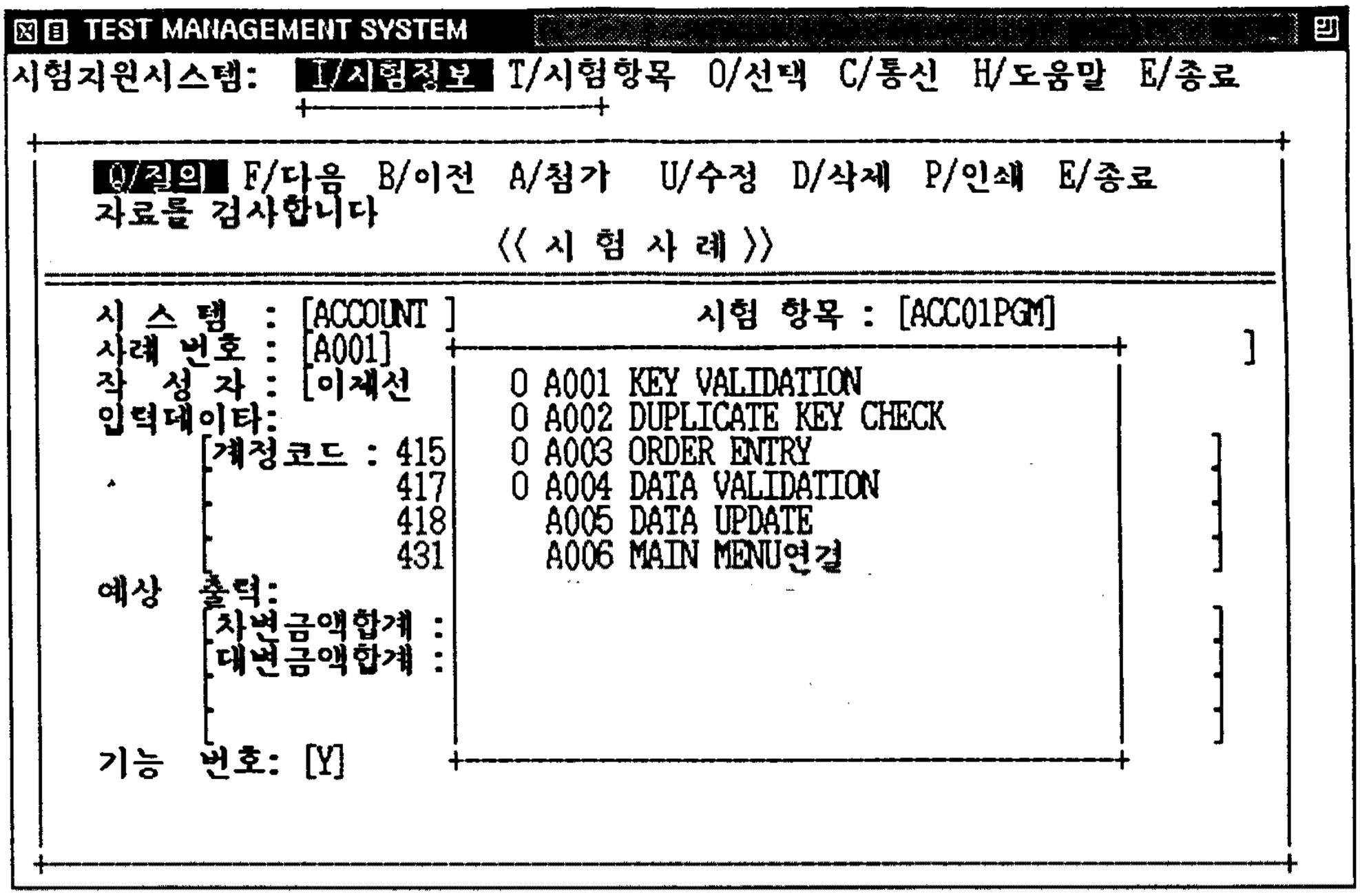
○ 이 화면은 시험 사례의 진행 화면에서 원하는 시험 사례를 선택 (〈Return〉 Key) 하여 시험사례나 절차를 편집, 조회하는 경우이다.

○ "C/시험 사례"선택할 경우 : 선택된 사례 번호에 대해 시험 사례 편집 화면이 나타난다.

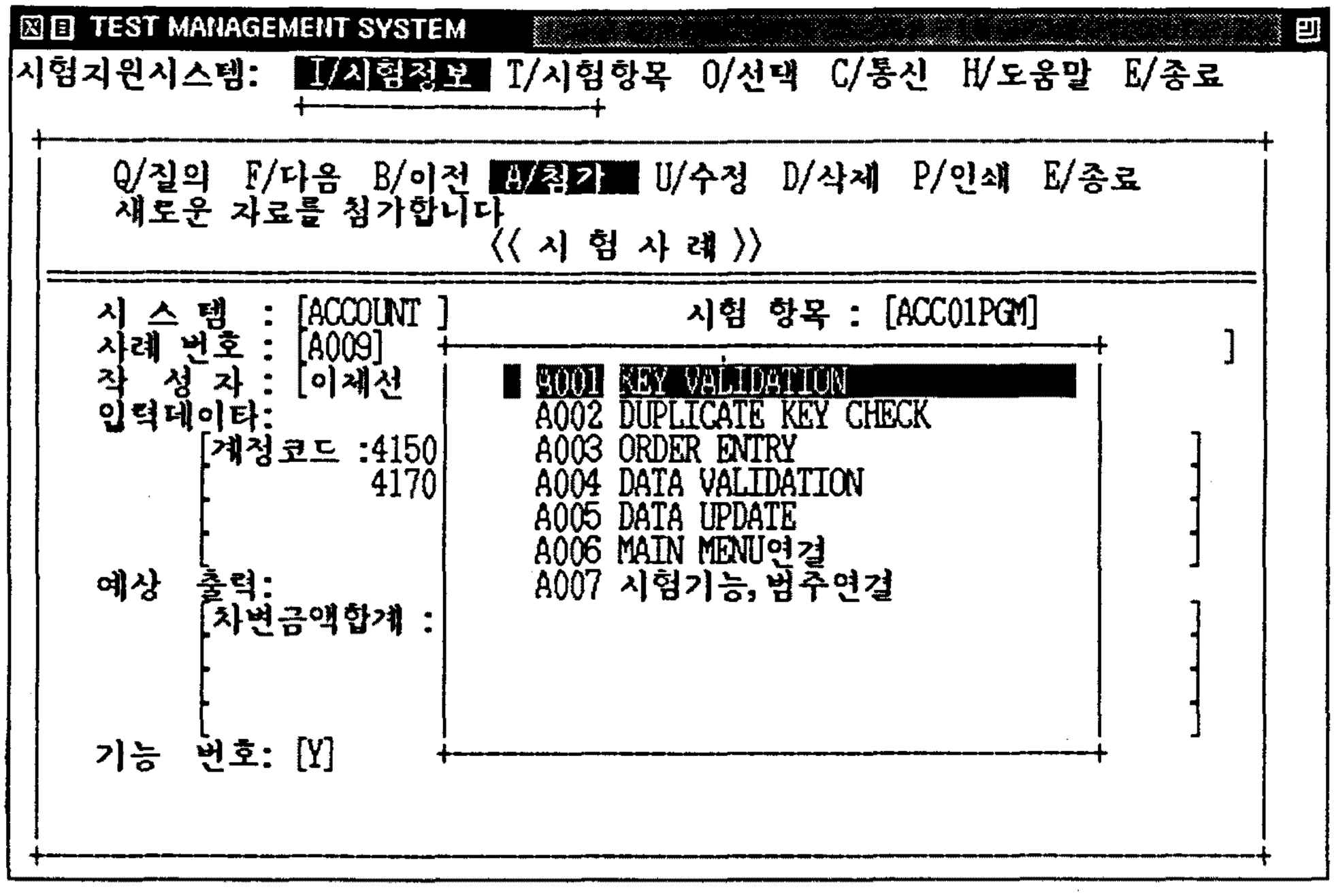
○ "R/시험 절차"선택할 경우 : 선택된 절차에 대한 시험 절차 편집 화면이 나타난다.

```
TEST MANAGEMENT SYSTEM
시험지원시스템: I/시험정보 T/시험항목 O/선택 C/통신 H/도움말 E/종료
+-----------+

 Q/질의 F/다음 B/이전 A/첨가  U/수정 D/삭제 P/프린트 E/종료
 자료를 검사합니다
                        <<  시 험 사 례  >>

 시 스 템  : [ACCOUNT ]              시험 항목 : [ACC01PGM]
 사례 번호 : [A001]                   목     적 : [전표입력확인          ]
 작 성 자  : [이재선    ]             작성 일자 : [1991/06/04]
 입력데이타:
      [계정코드 : 4150, 대차구분:1(차변)  금액:1,000,000                  ]
      [           4170, 대차구분:2(대변)  금액:1,000,000                  ]
      [           4180, 대차구분:1(차변)  금액:1,000,000                  ]
      [           4310, 대차구분:2(대변)  금액:1,000,000                  ]
 예상  출력:
      [차변금액합계 : 2,000,000                                           ]
      [대변금액합계 : 2,000,000                                           ]
      [                                                                   ]
      [                                                                   ]
 기능  번호: [N]
```

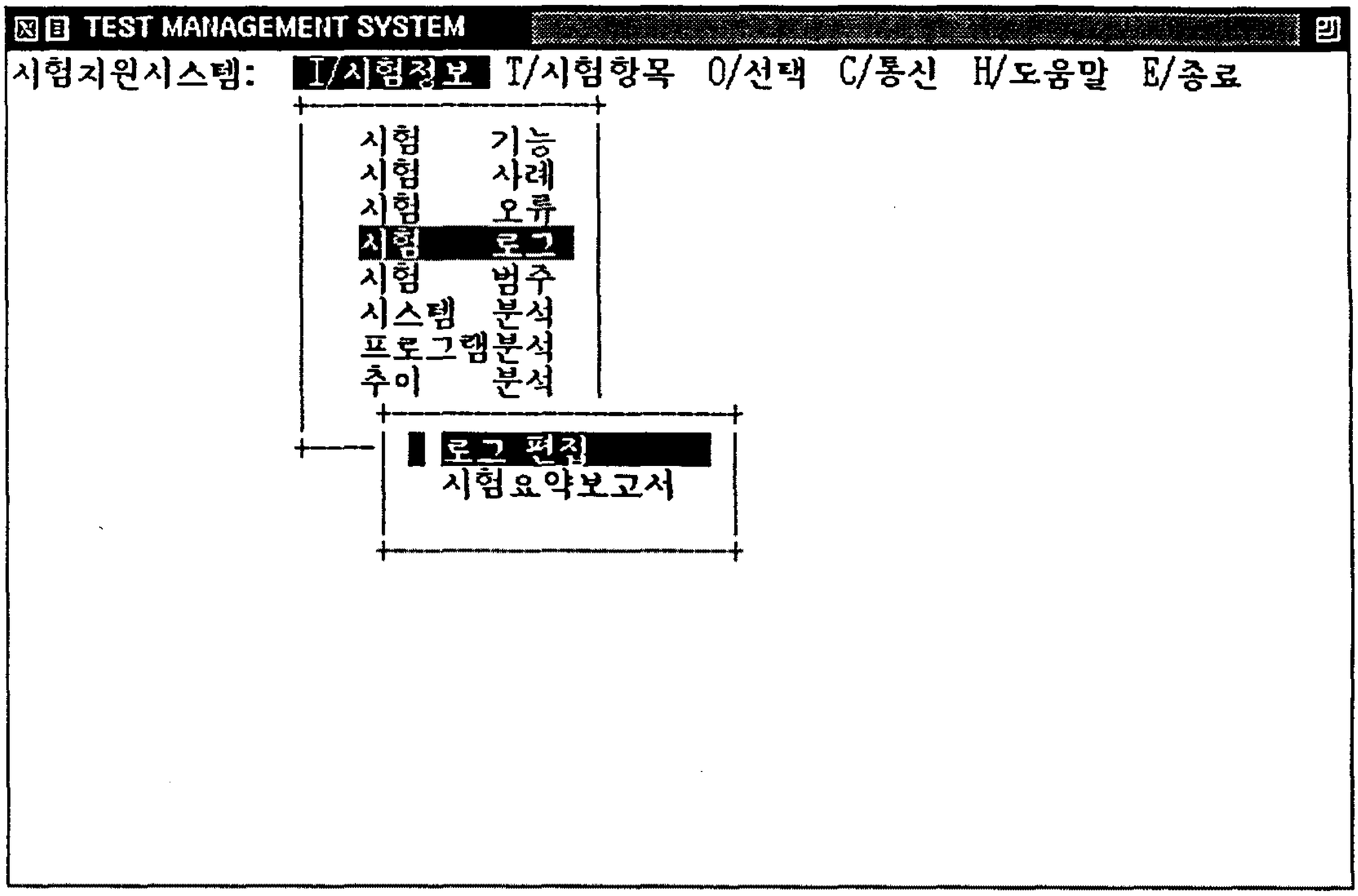
- 시험 사례 조회, 편집 화면이다.
- 시험 사례 조회, 편집 후 기능 번호에 'Y'를 입력하면 관련된 시험 기능을 조회, 편집할 수 있다.



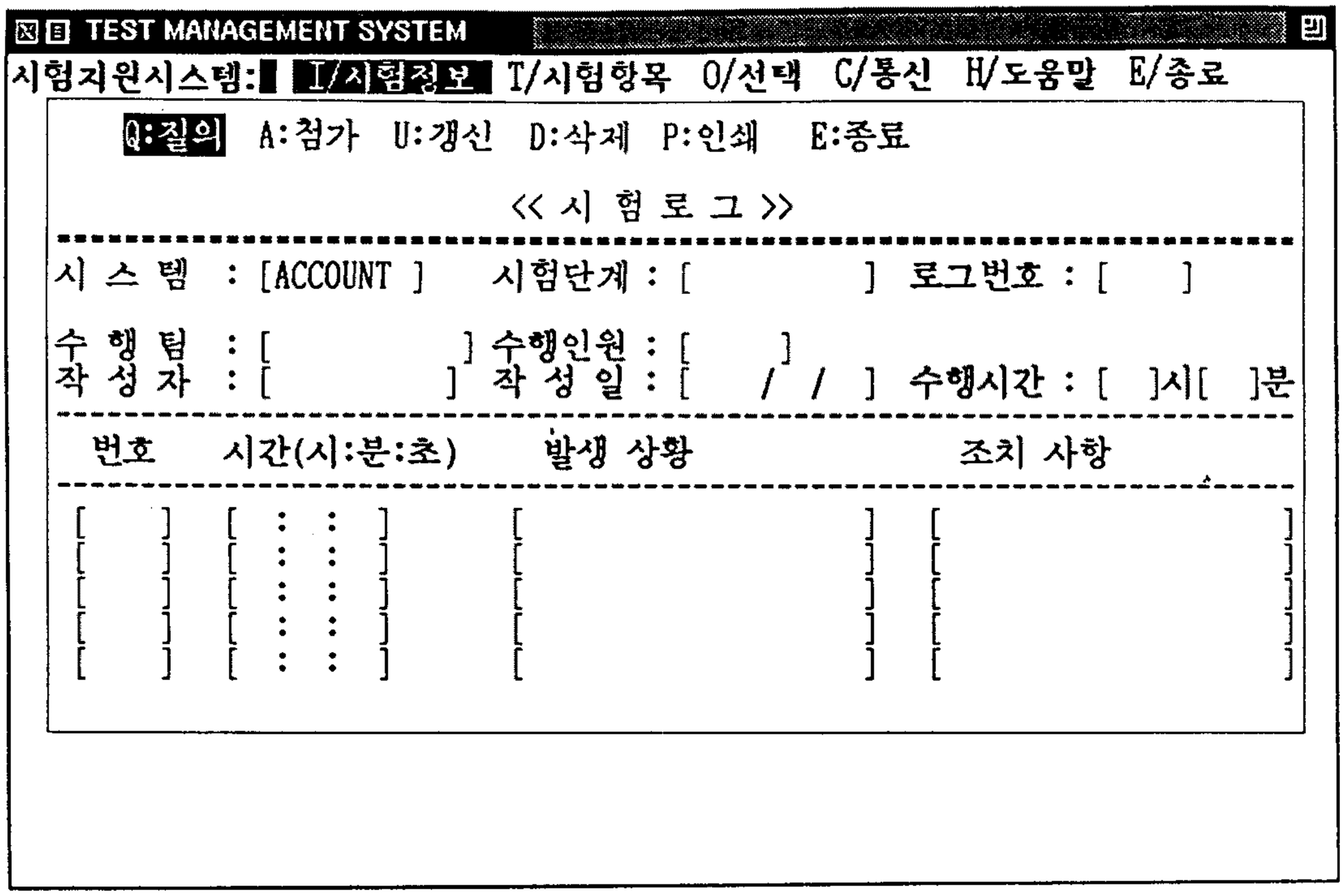
- 시험 사례 관리 화면에서 시험 사례에 해당되는 시험 기능을 조회하는 화면으로 각 기능앞에 관련 여부가 '0' 또는 공란으로 나타난다.



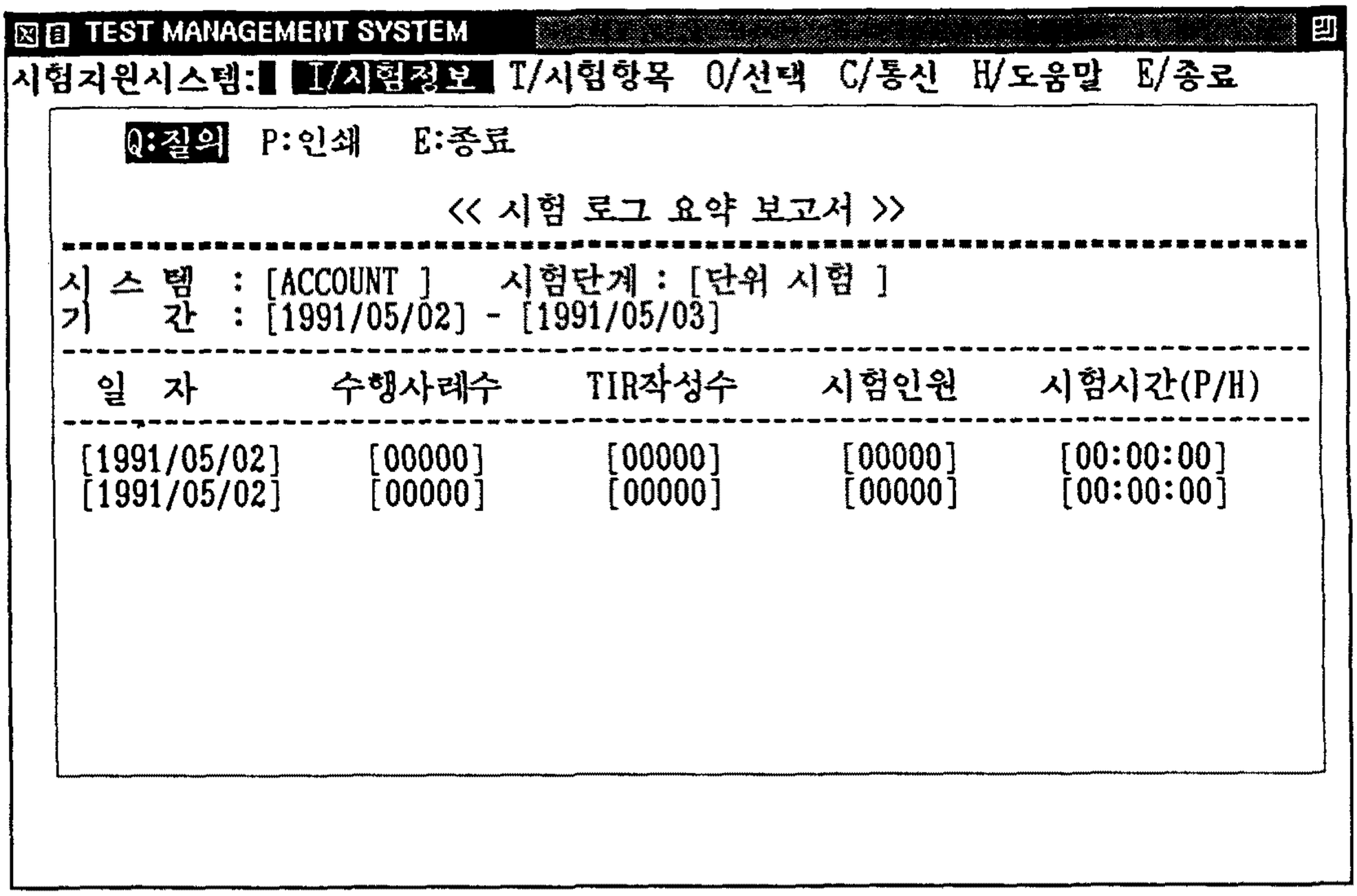
- 이 화면은 시험 사례를 "A : 첨가"하는 화면이다.
- 각 필드 입력
  - 시험 항목 : 시험 기능을 포함하는 항목
  - 목 적 : 시험 사례를 수행하는 목적
  - 작 성 자 : 시험 사례를 작성하는 자
  - 작성 일자 : 시험 사례를 작성한 일자
  - 입력 데이타 : 시험 사례에 대한 입력데이타
  - 예상 출력 : 입력 데이타를 수행한 후의 예상 결과
  - 기능 번호 : 시험 사례에 포함되는 시험 기능
- 모든 필드를 입력한 후
  - 〈Esc〉 Key를 누른다.
  - Message를 보여준 후 메뉴로 Cursor가 이동한다.
  - Interrupt Key〈DEL〉를 누르면 작업을 중단하고 메뉴로 Cursor가 돌아간다.
  - "U : 수정, D : 삭제"는 검색을 한 후 화면에 존재하는 시험 사례를 수정, 삭제한다.



• 시험 로그의 조회 및 편집, 요약 보고서 출력을 선택하는 메뉴이다.



- 이 화면은 시험 로그를 조회, 편집하는 화면이다.
- 사례 진행 상황을 경유한 경우에는 해당 로그 검색 내용이 바로 화면에 출력되며, 메뉴에서 직접 시험 로그를 선택한 경우에는 시스템, 시험 단계, 로그 번호를 입력하여 작업을 수행한다.



• 이 화면은 시험 로그 요약 보고서 화면이다.

• 시스템, 시험 단계, 기간을 선택한다.

• 검색된 자료

- 다음 화면 : 〈Ctrl + F〉 Key를 누른다.
- 이전 화면 : 〈Ctrl + B〉 Key를 누른다.
- 방향키 : 한행씩 이동한다.

• Interrupt Key〈DEL〉를 누르면 작업을 중단하고 메뉴로 Cursor 가 돌아간다.

• " Q : 질의 "

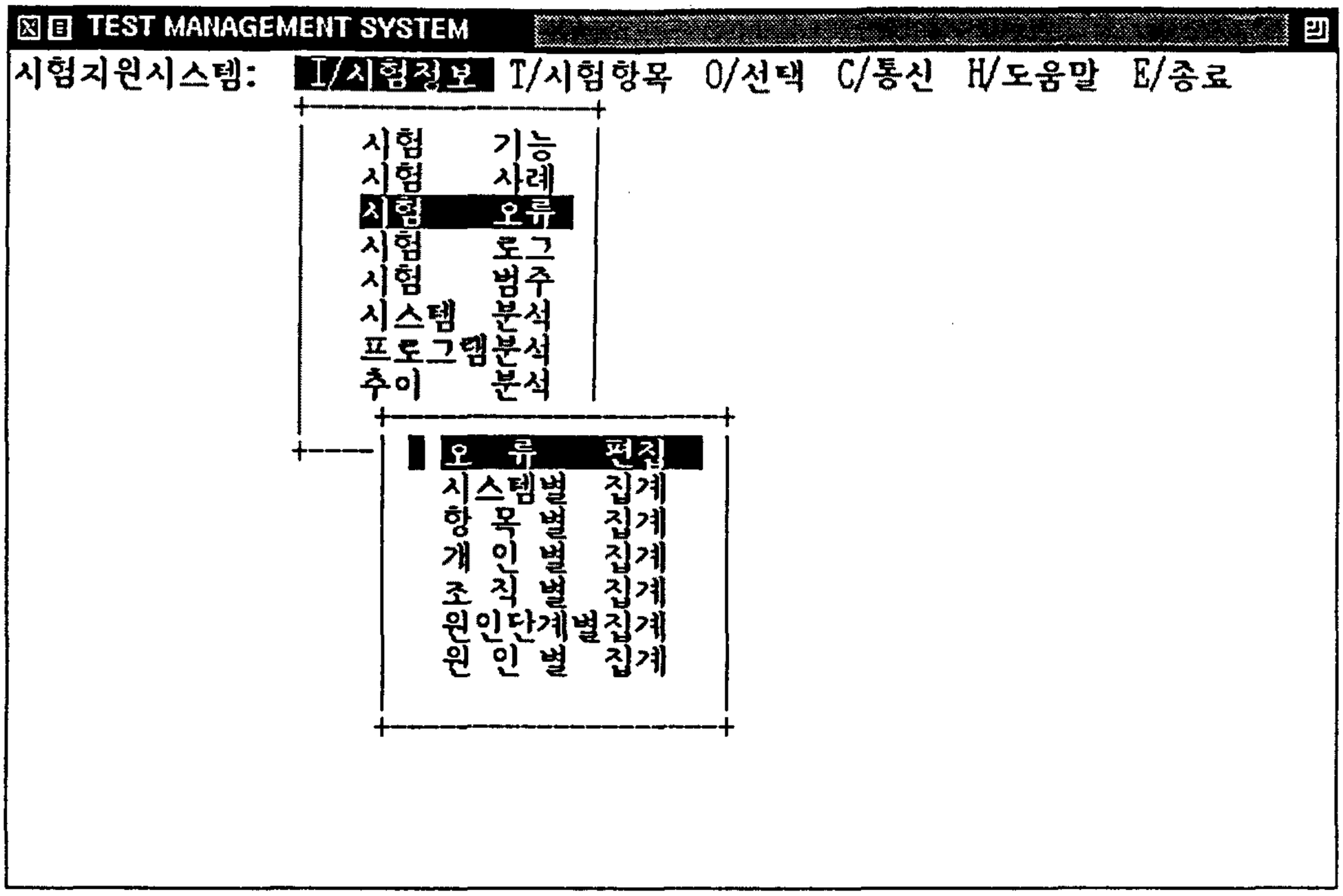
- 시스템, 시험 단계, 기간을 선택한 내용이 검색된다.

• " P : 인쇄 "

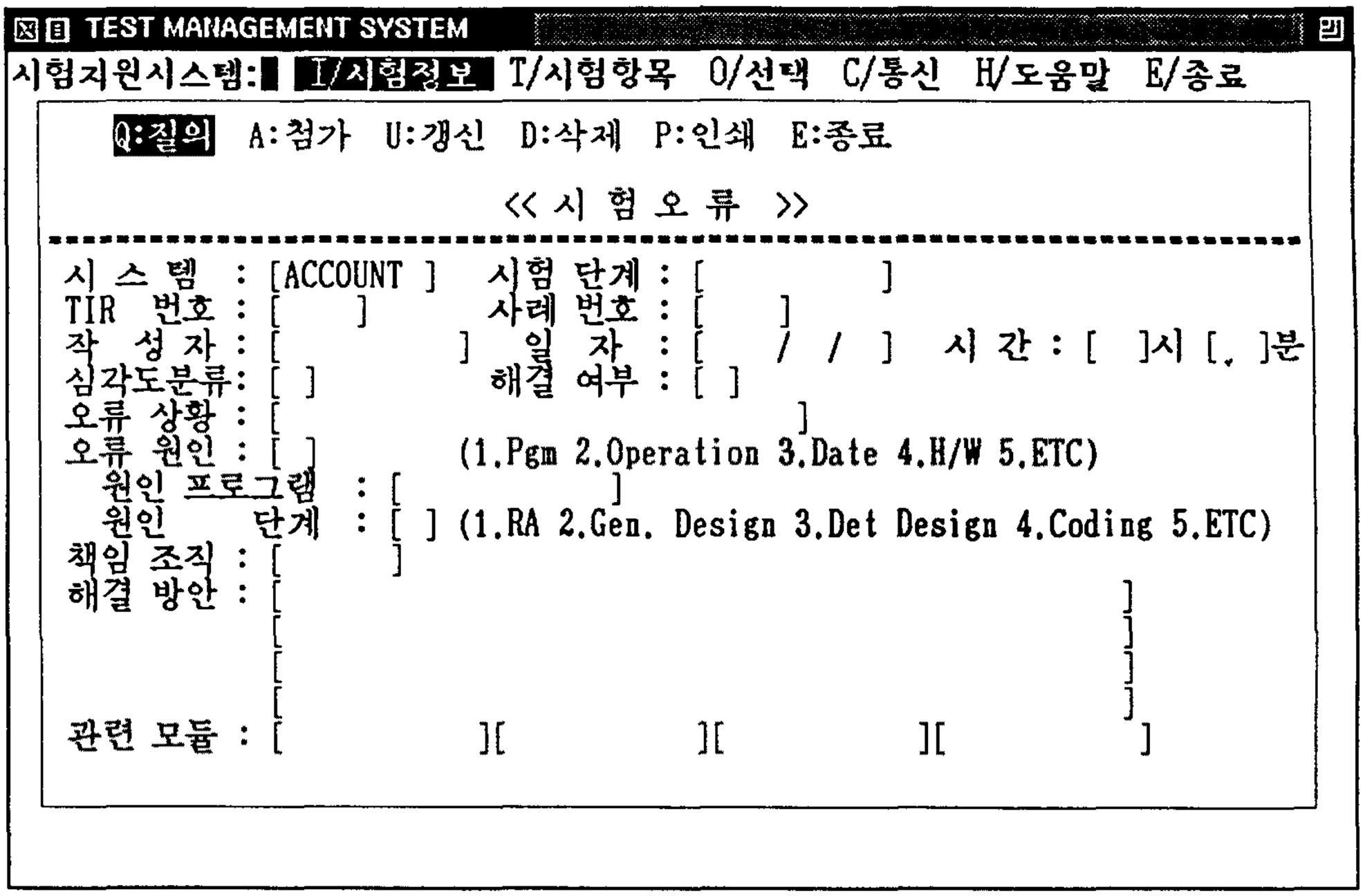
- 시험 로그 요약을 출력한다.

• E : 종료

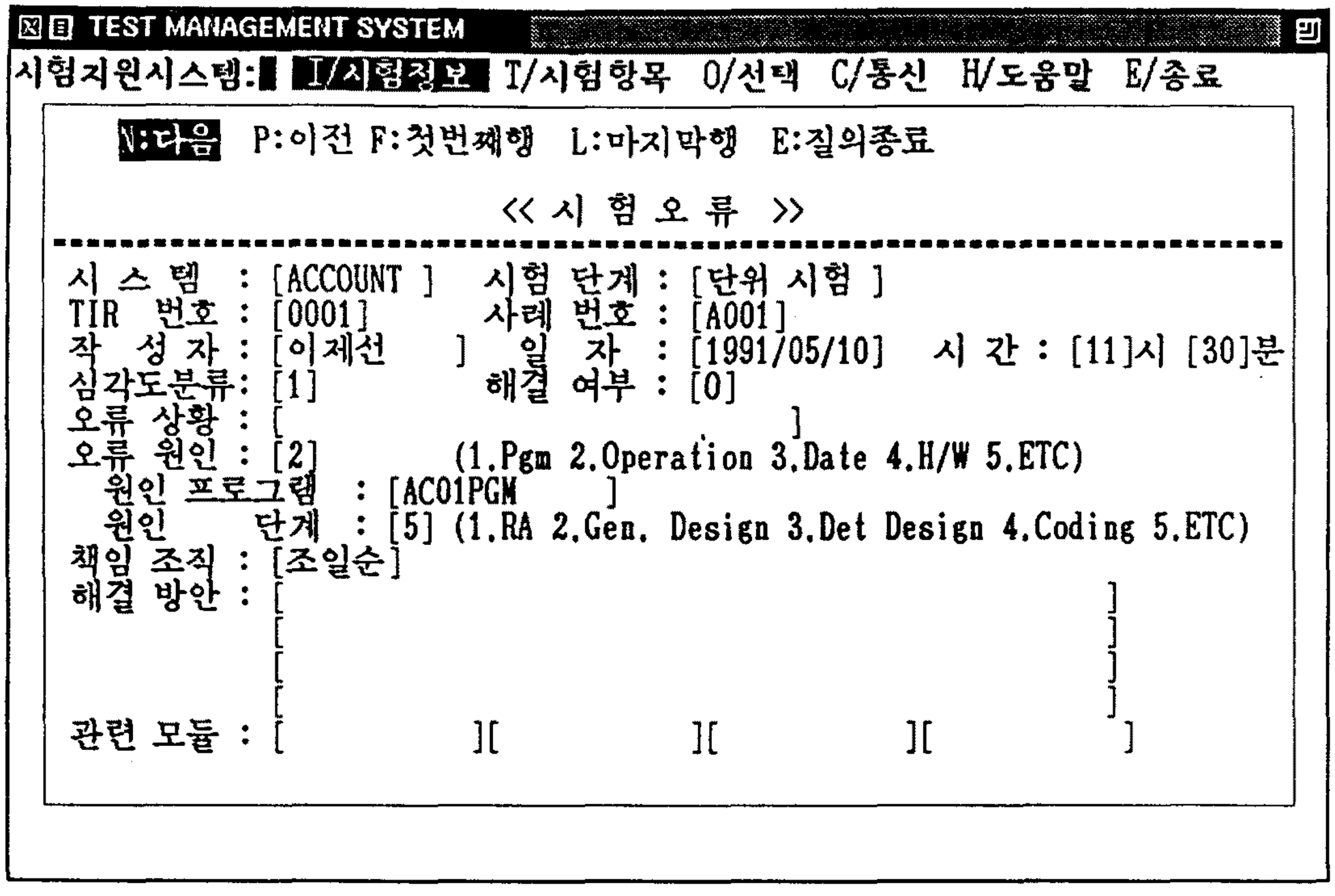
- 이 화면을 선택한 메뉴 화면으로 돌아간다.



- 이 화면은 주 메뉴 화면에서 오류 편집 작업을 선택하는 과정을 나타낸 화면이다.
- 작업 과정은
  - “ I/시험정보 ” 부분이 Reverse된 상태에서 〈Return〉 Key를 누르면 시험정보에 관한 메뉴가 나타난다.
  - 시험정보 메뉴가 나타난 상태에서 방향 Key를 눌러 원하는 작업을 선택한 후 〈Return〉 Key를 누르면 그에 따른 해당작업이 나타난다.
  - 원하는 작업을 선택하여 〈Return〉 Key를 누른다.



- 이 화면은 시험 오류를 질의, 첨가, 갱신, 삭제, 인쇄하는 화면이다. 질의를 선택하는 경우에는 "Q:질의"를 선택하여 각 필드에 조건을 입력한다. 갱신, 삭제는 현재 조회된 내용에 대해 수행된다.
- 시스템 시험 단계 등에 대해서는 *를 입력함으로서 안내 화면을 호출할 수 있다.
- 필드 설명
  - TIR번호 : Test Incident Report 번호
  - 해결 여부 : 시험 오류 해결 여부
  - 오류 상황 : 오류가 발생한 내용
  - 오류 원인 : 오류 원인의 Resource 일련 번호
    - 원인 프로그램 : 오류가 발생한 프로그램
    - 원인단계 : Life Cycle 단계
  - 책임 조직 : 시스템을 개발한 조직
  - 해결 방안 : 발생한 오류 해결 방안
  - 관련된 모듈 : 오류에 관련된 모듈명



- 이 화면은 검색 조건에 맞는 오류 내용들을 보여준다.
- 시험 오류 화면에서 메뉴만 변경된다.
- 검색된 오류 내용중 첫번째 내용을 화면에 보여준다.

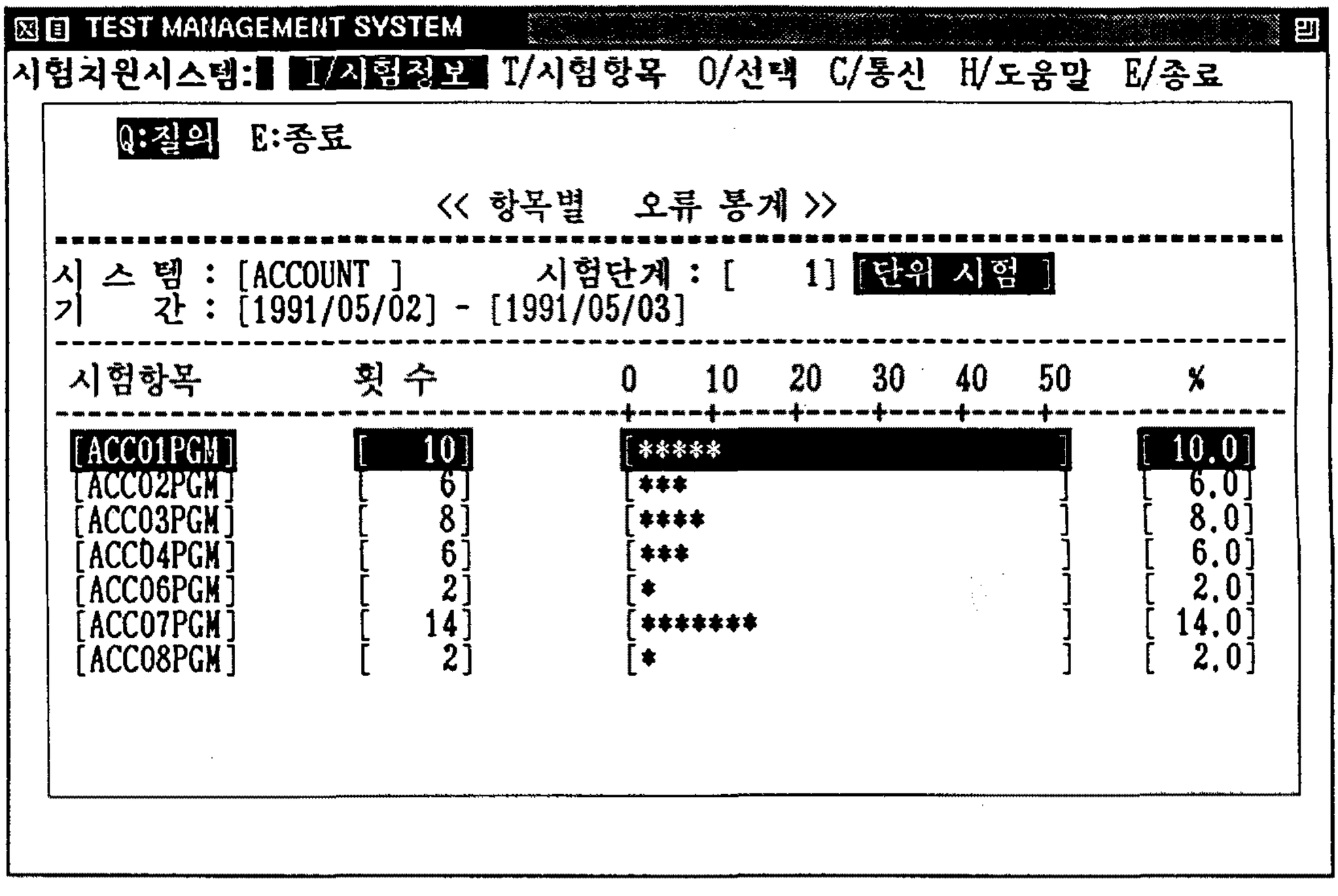
  “N : 다음” : 검색된 오류 내용중 화면에 보인것 다음 오류 내용을 보여준다.

  “P : 이전” : 검색된 오류 내용중 화면에 보인것 이전 오류 내용을 보여준다.

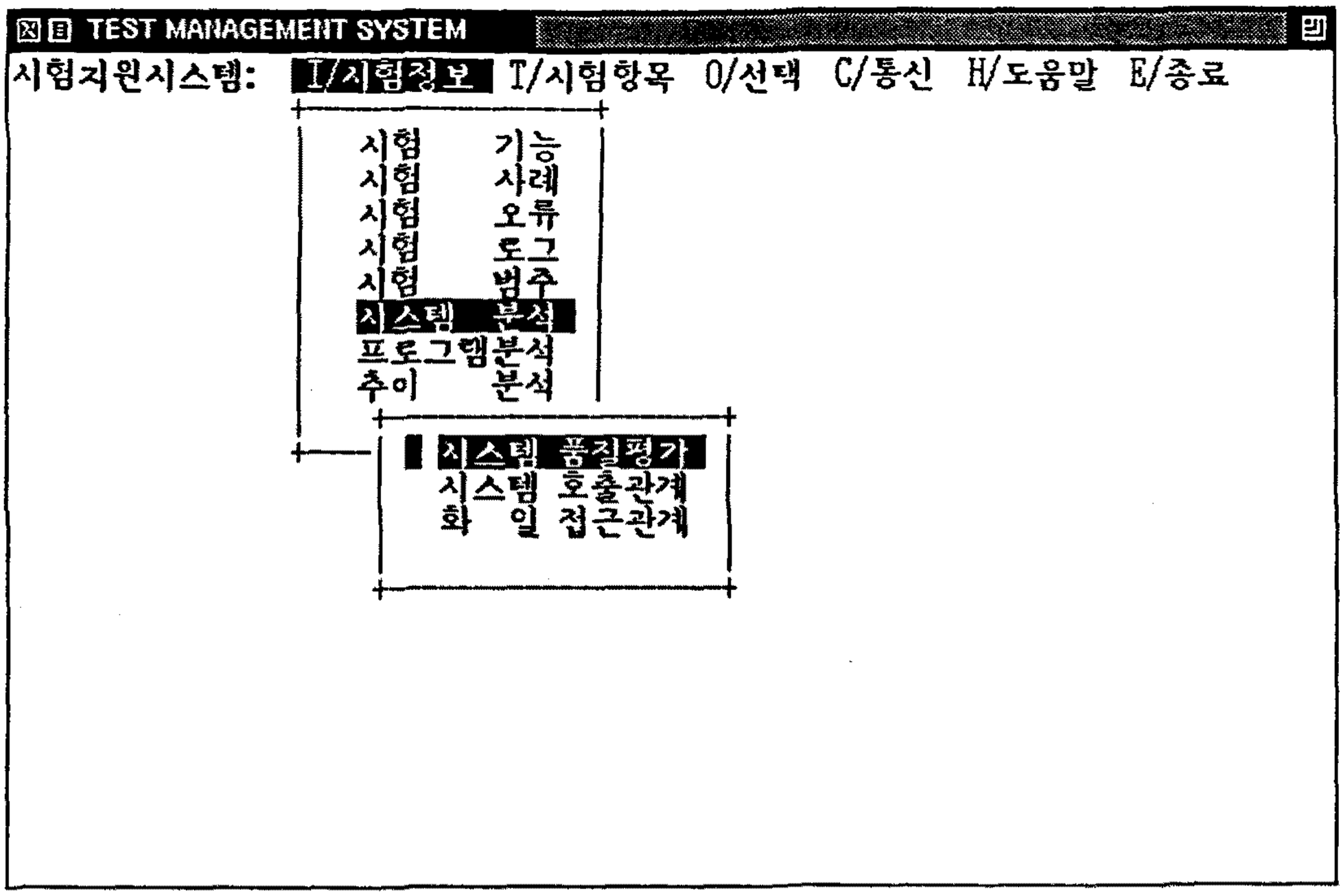
  “F : 첫번째행” : 검색된 오류 내용중 첫번째 행을 화면에 보여준다.

  “L : 마지막행” : 검색된 오류 내용중 마지막 행을 화면에 보여준다.

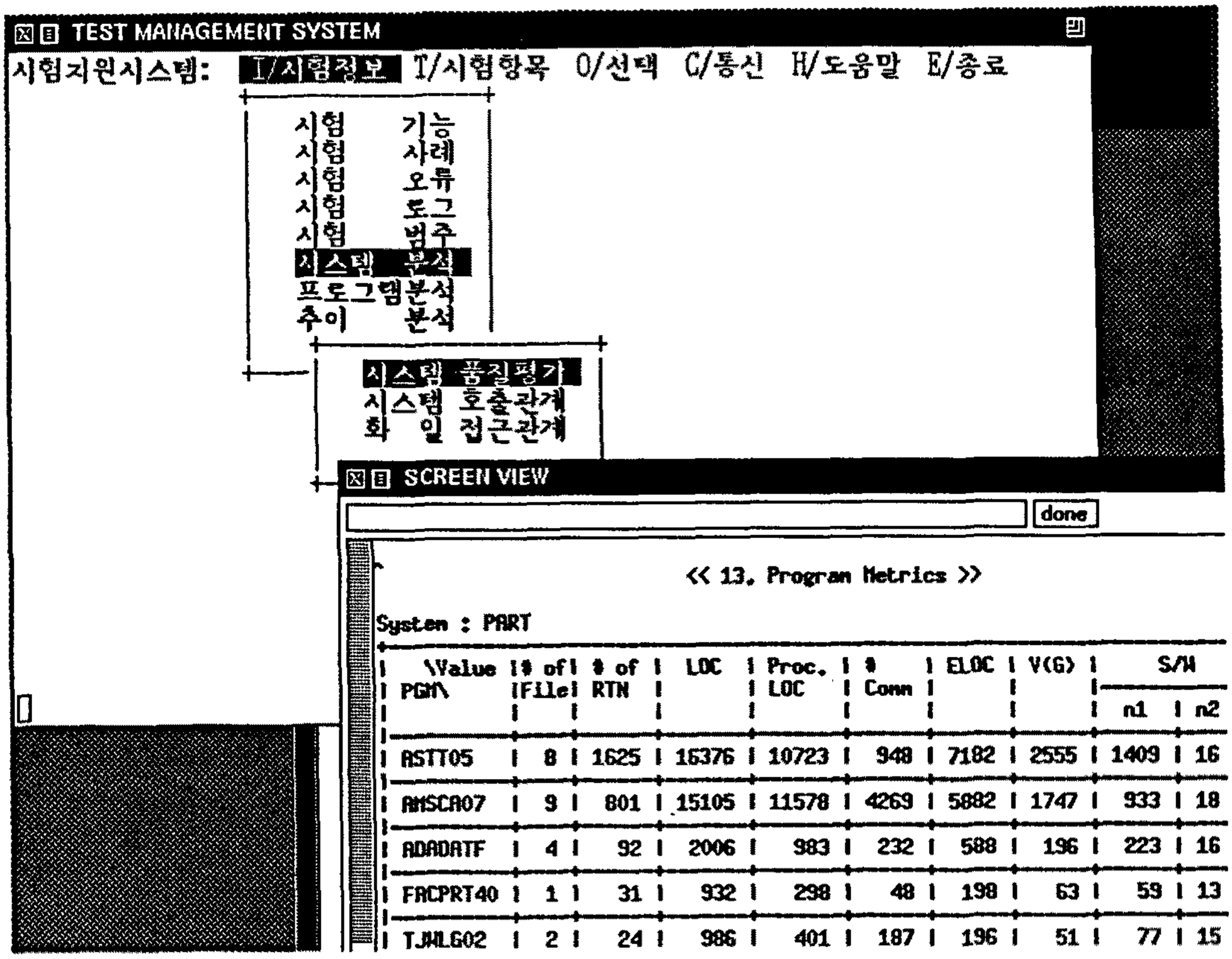
  “E : 질의종료” : 각 필드가 지워지면서 “Q : 질의” 메뉴에 Cursor가 이동된다.



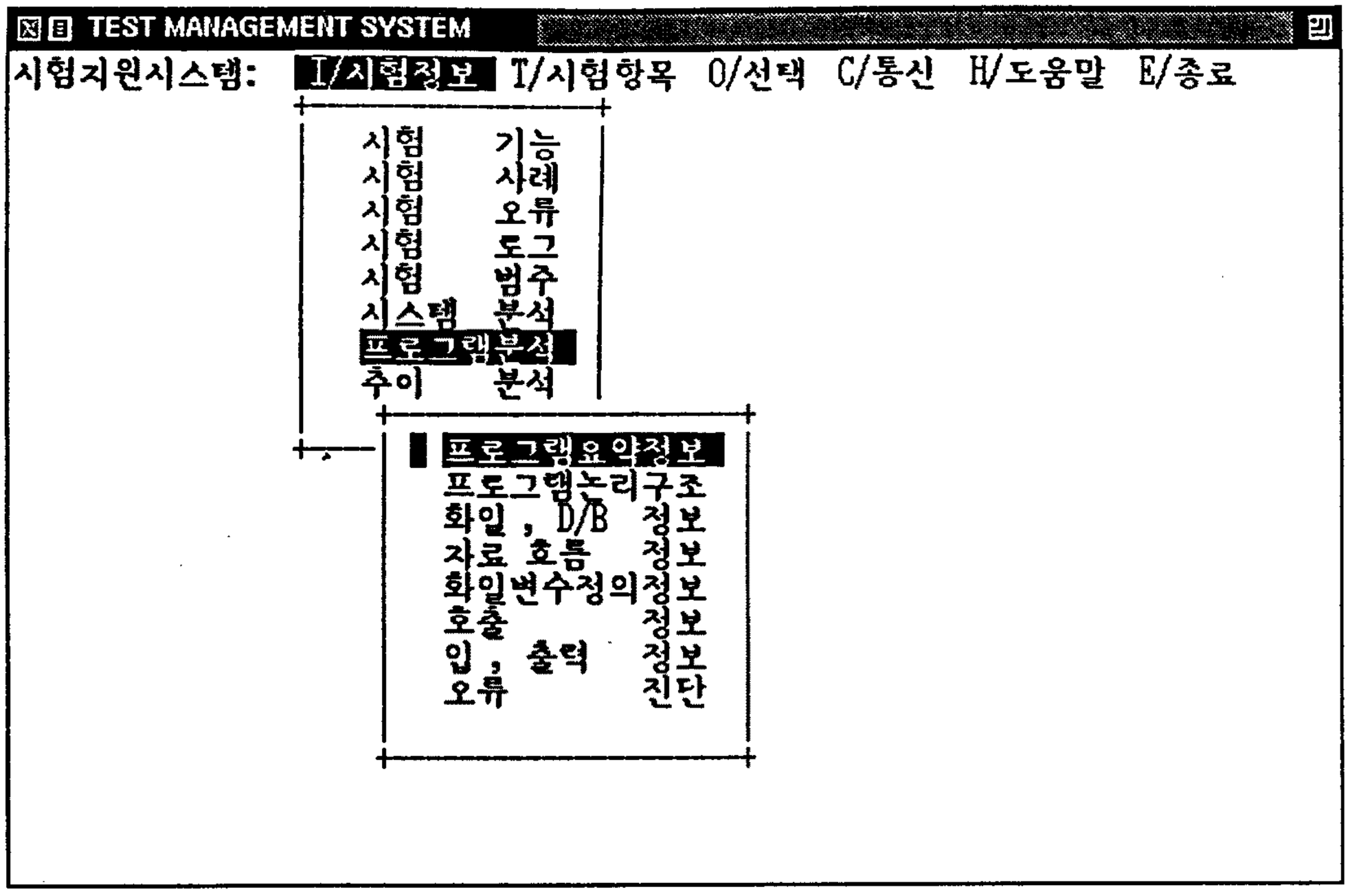
- 이 화면은 항목별 오류 통계 화면이다.
- 시스템, 시험 단계를 기준으로 하여 각 시험 항목에 대한 오류 발생 횟수, 비교표, 퍼센트를 보여 준다.



- 프로그램을 정적 분석한 후 생성되는 시스템 분석 결과 선택화면이다.
- 시스템 품질 평가 선택 시에 크기, 제어구조 등을 고려한 복잡도 측정 결과가 출력된다.
- 시스템 호출 관계는 주 프로그램과 부 프로그램간의 호출관계가 매트릭스 형태로 표현된다.
- 화일 접근 관계에서는 시스템내의 모든 화일에 대해서 매트릭스 형태로 쓰기, 읽기, 삭제 등을 구분 출력한다.

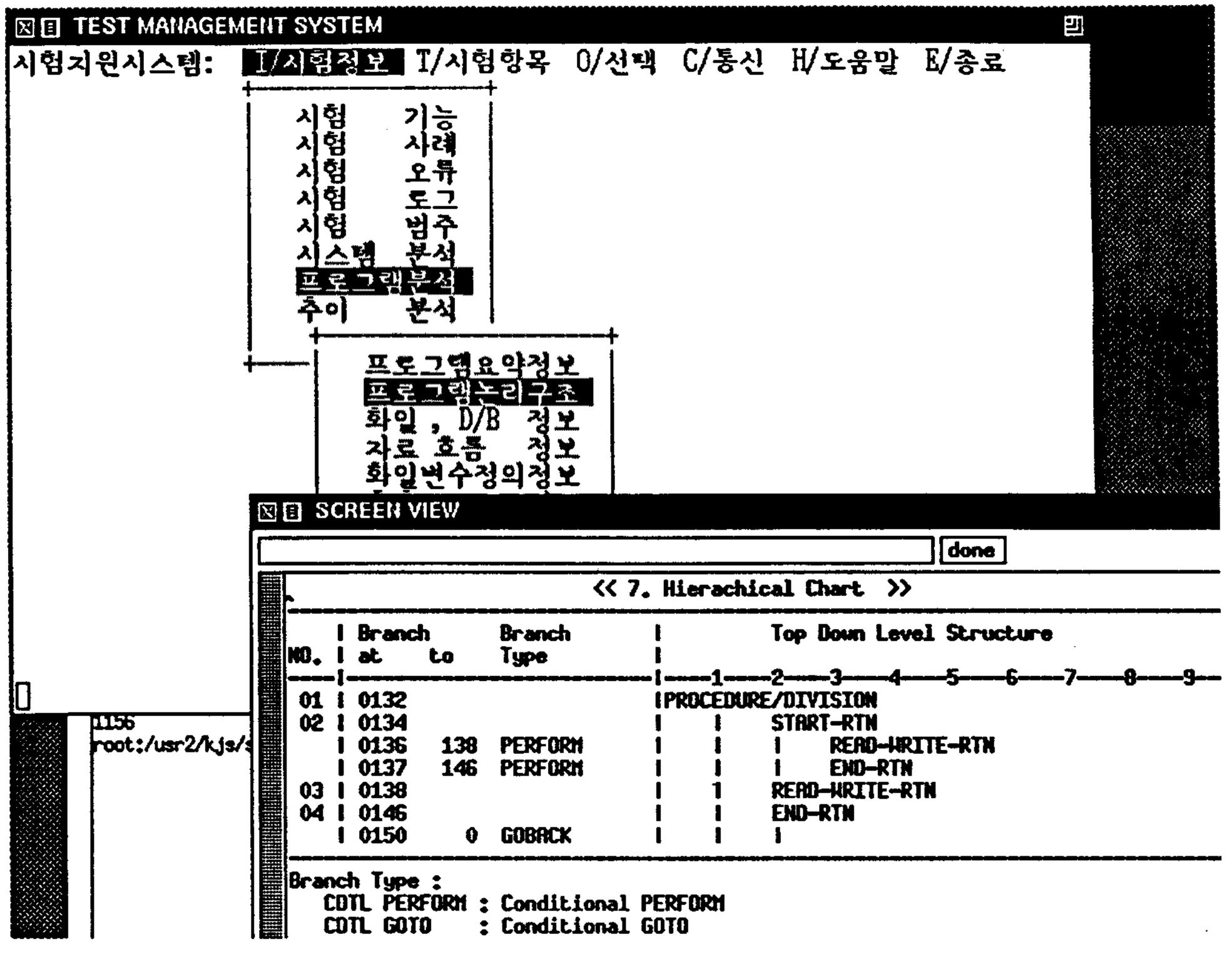


○ 시스템 품질 평가 출력 화면으로 LOC, Cyclomatic Complexity, Halstead Software Science, 화일 갯수등을 프로그램별로 출력한다.

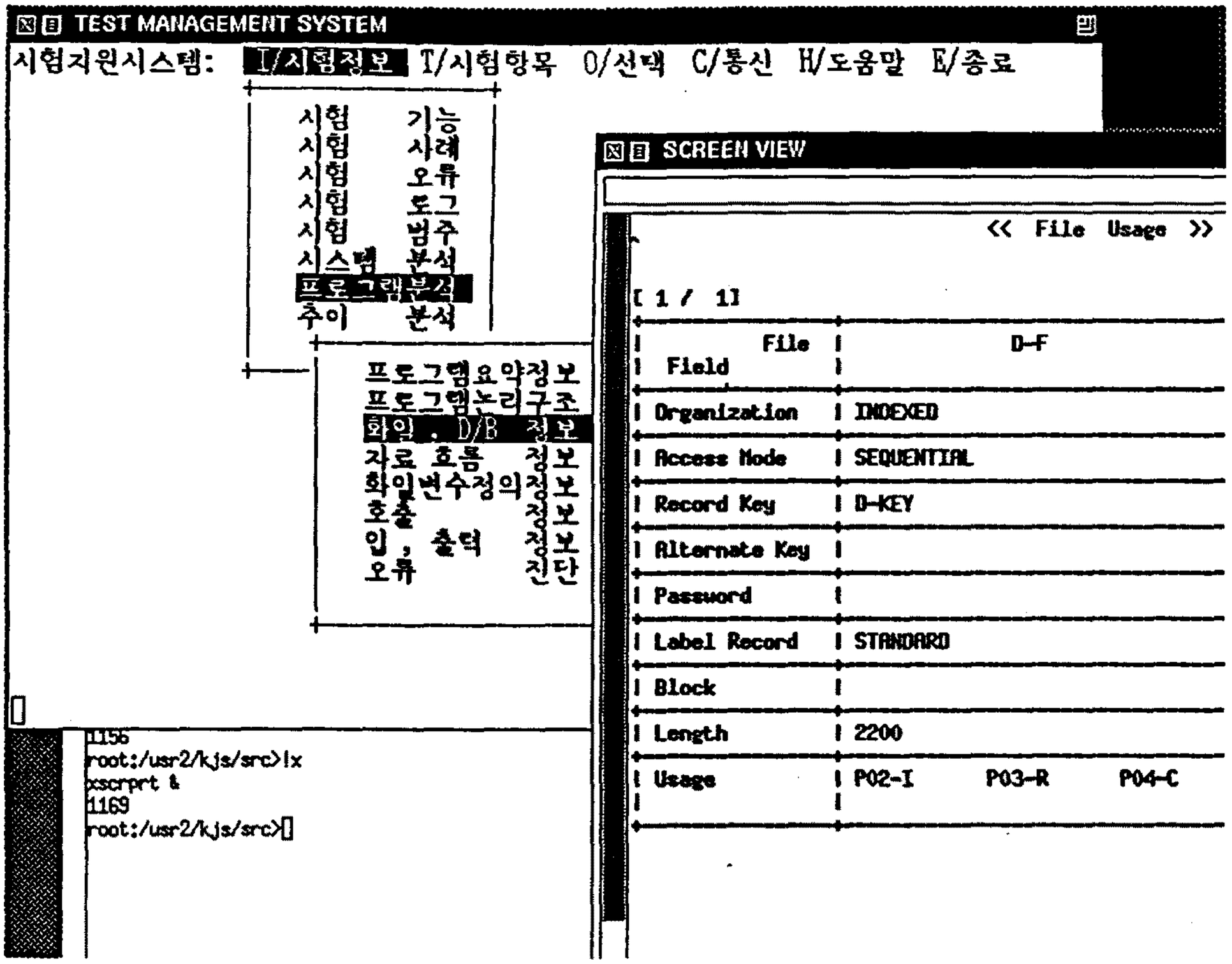


○ 프로그램별 분석 정보를 선택하는 화면이다.

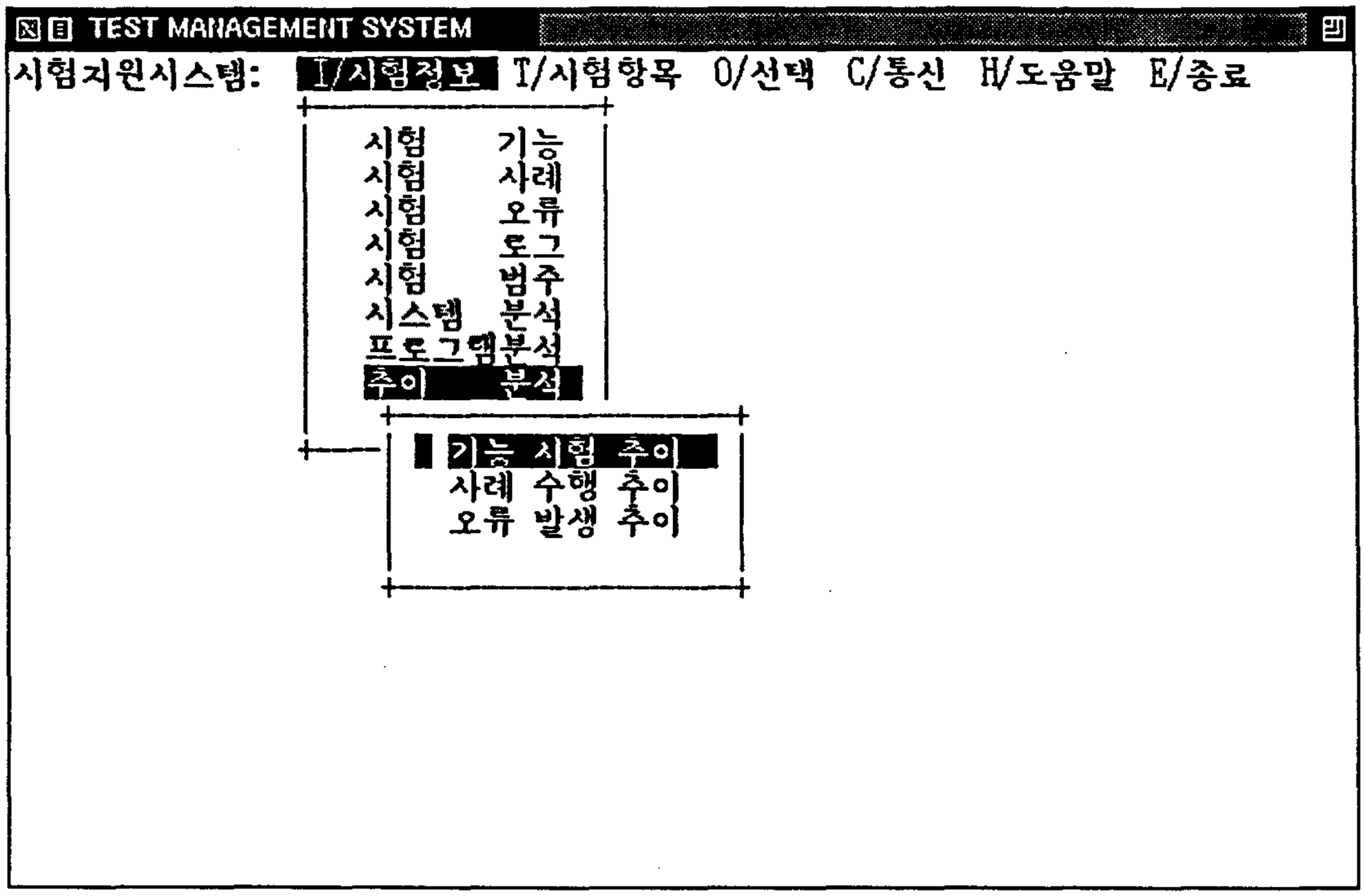
○ 항상 최근에 정적 분석한 프로그램의 결과를 보여준다.



○ 프로그램별 분석 정보중 논리 구조 출력 화면이다.

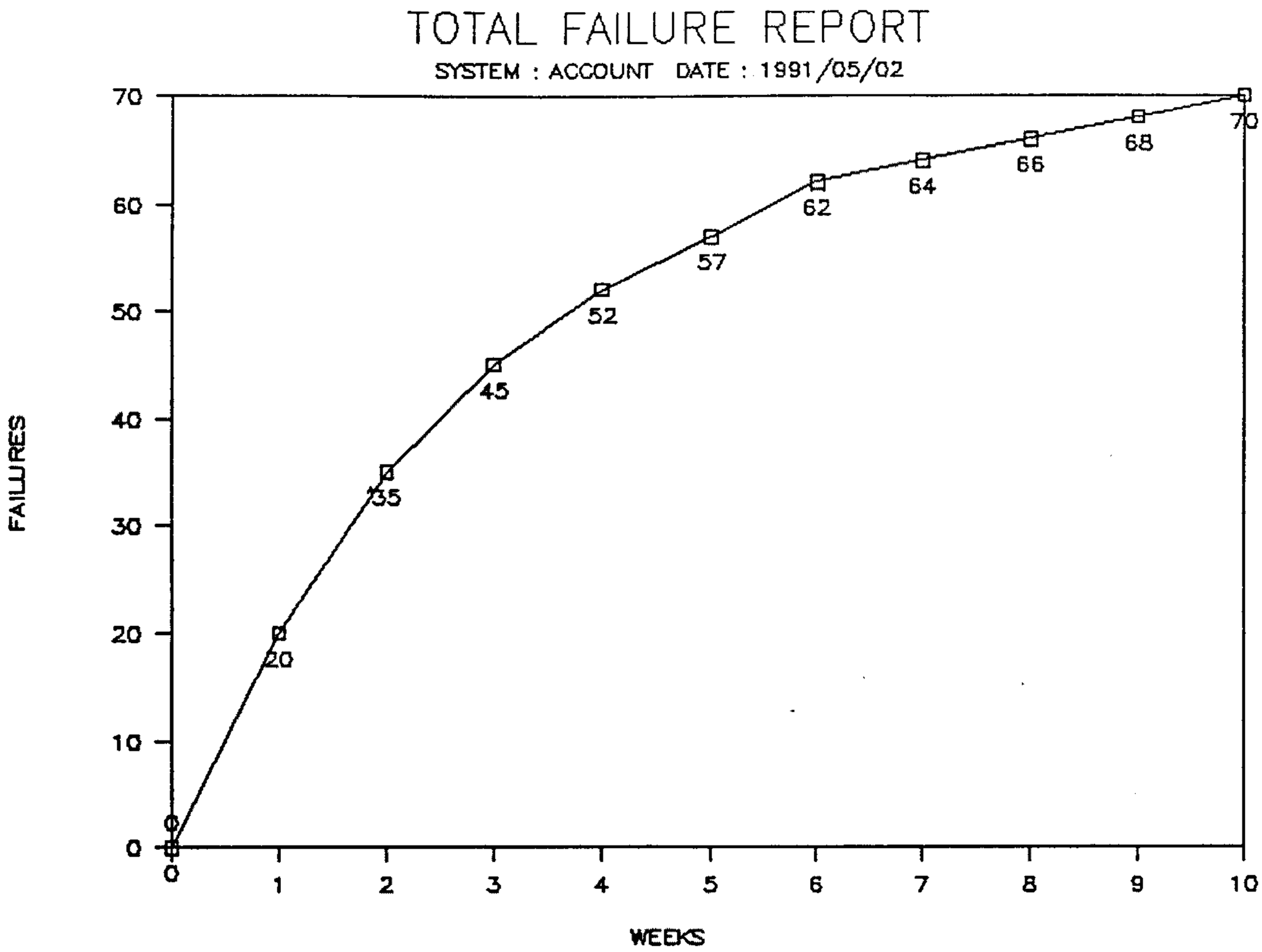


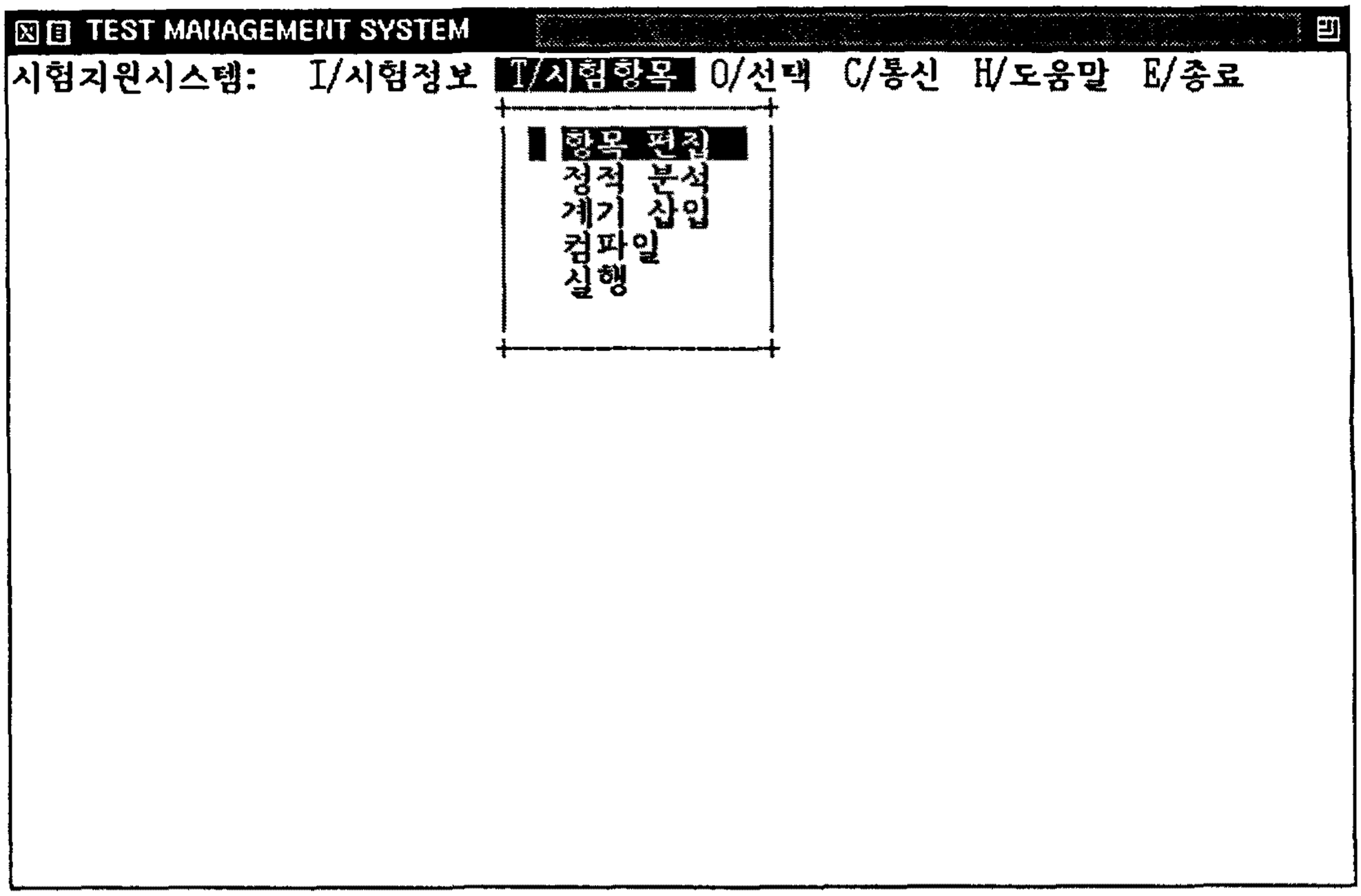
o 프로그램별 분석 정보 중 프로그램내의 정의된 화일에 대한 정보 및 Open, Close, 읽기, 쓰기 정보를 출력한다.



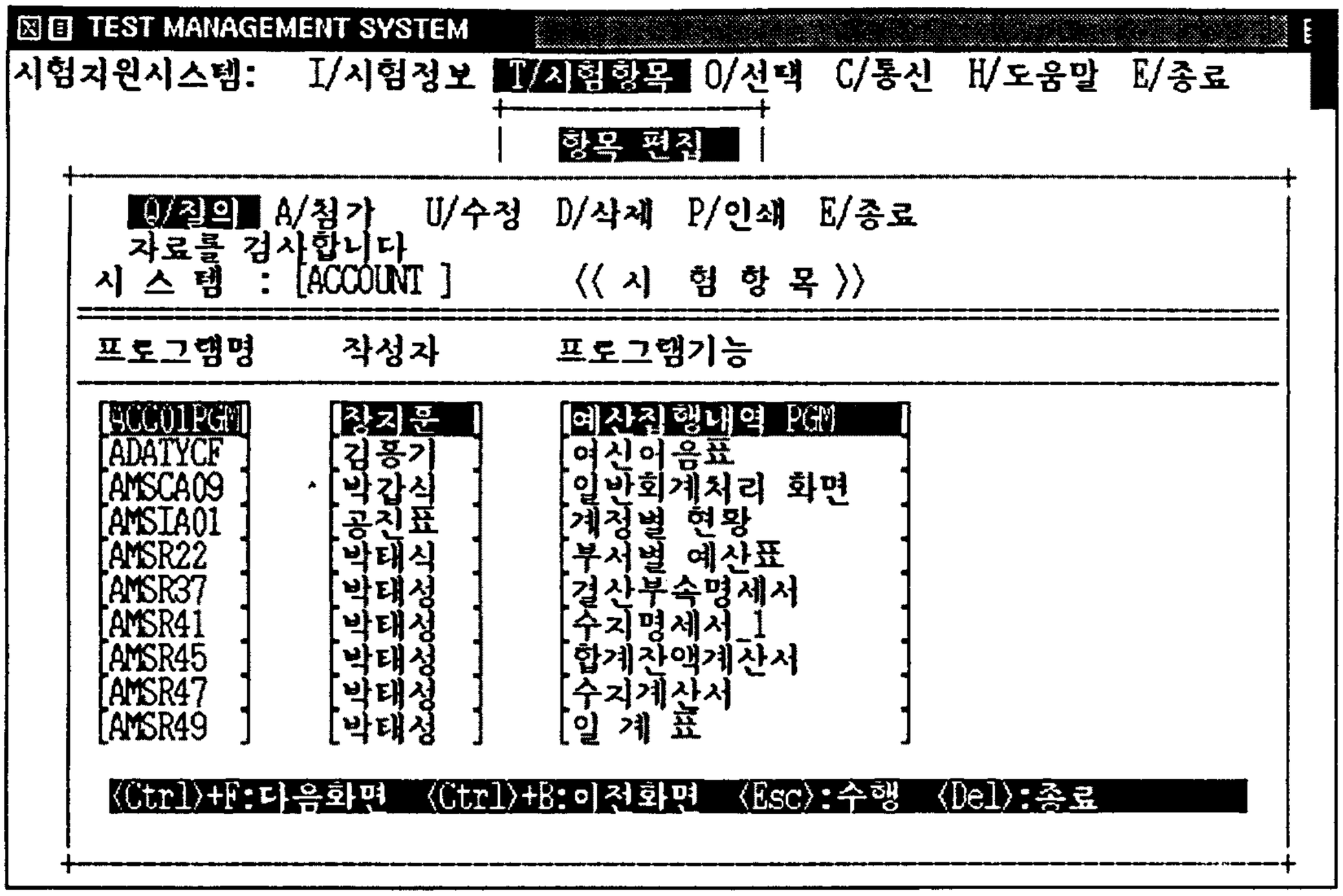
○ 추이 분석 결과 선택화면이다.

○ 각 항목 선택시 추이 분석 그래프가 주 단위로 출력된다.

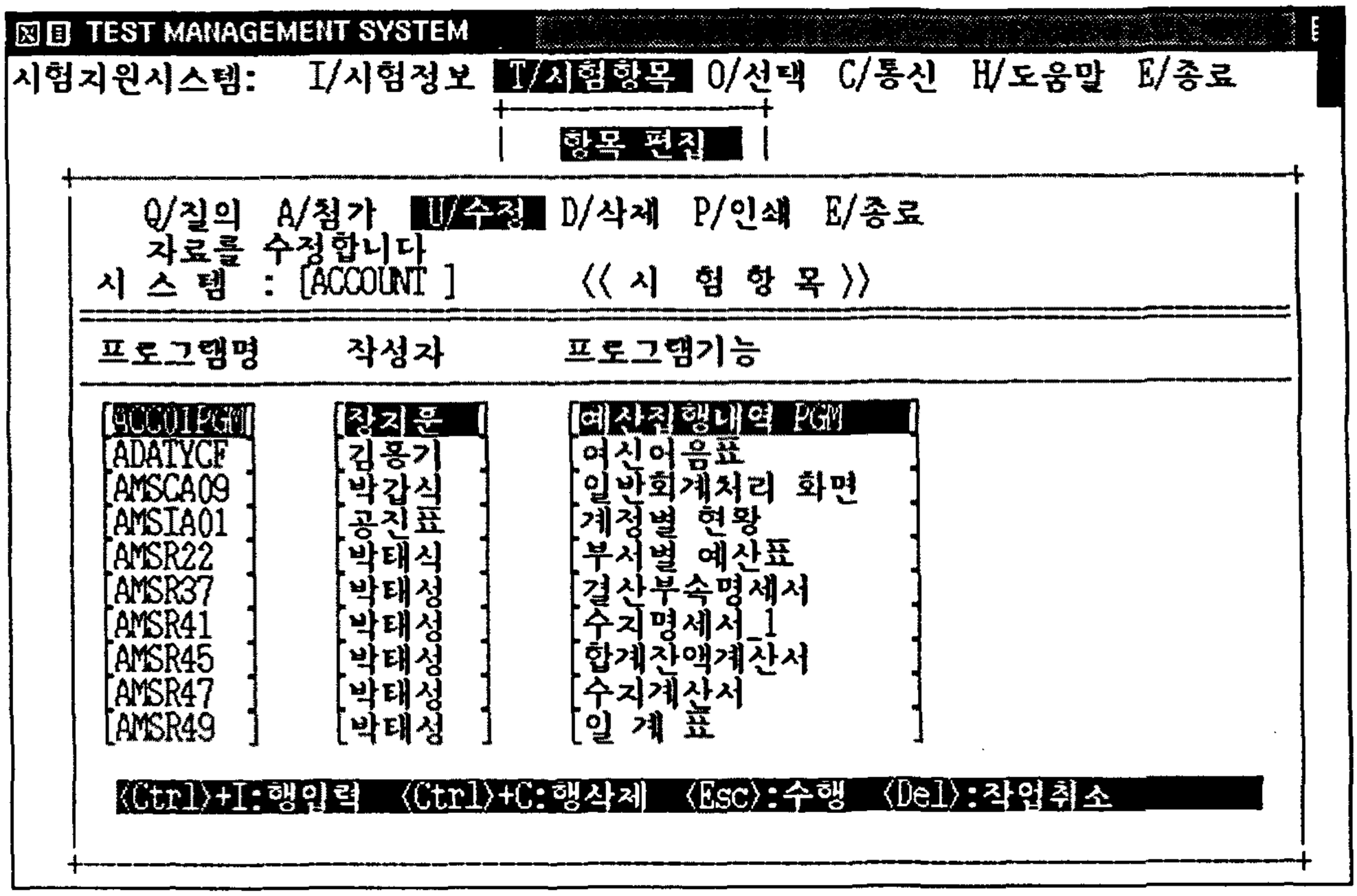

TOTAL FAILURE REPORT
SYSTEM : ACCOUNT DATE : 1991/05/02
FAILURES
WEEKS
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6
7
8
9
10
0
20
35
45
52
57
62
64
66
68
70



- 이 화면은 시험 항목을 선택한 화면이다.
- 항목 편집
  - 시스템 내의 프로그램 목록을 관리한다.
- 정적 분석
  - 프로그램을 분석하여 품질 평가 정보 및 프로그램 분석 정보를 생성한다.
- 계기 삽입
  - 원시 프로그램의 레이블, 분기, 문, 호출에 동적 분석 결과를 생성하기 위해 계기를 삽입한다.
- 컴 파일
  - 계기가 삽입된 프로그램을 선택하여 컴파일 한다.
- 실행
  - 계기가 삽입된 프로그램을 선택하여 실행시킨다.



- 이 화면은 항목 편집 화면이다.
- 시스템을 선택한 후 〈Return〉 Key를 누른다.
- 시스템 선택 방법은 시험 기능과 같다.
- Q/질의
  - 선택한 시스템에 대한 시험 항목들을 보여준다.
- A/첨가
  - 새로운 시험 항목을 등록한다.
- U/수정
  - 등록된 시험항목을 수정, 삭제, 추가한다.
- D/삭제
  - 시스템에 등록된 전체 시험 항목을 삭제한다.
- P/인쇄
  - 시스템에 대한 시험항목들을 인쇄한다.
- E/종료
  - 이 화면은 선택한 메뉴 화면으로 돌아 간다.



• 이 화면은 항목 편집 화면이다.

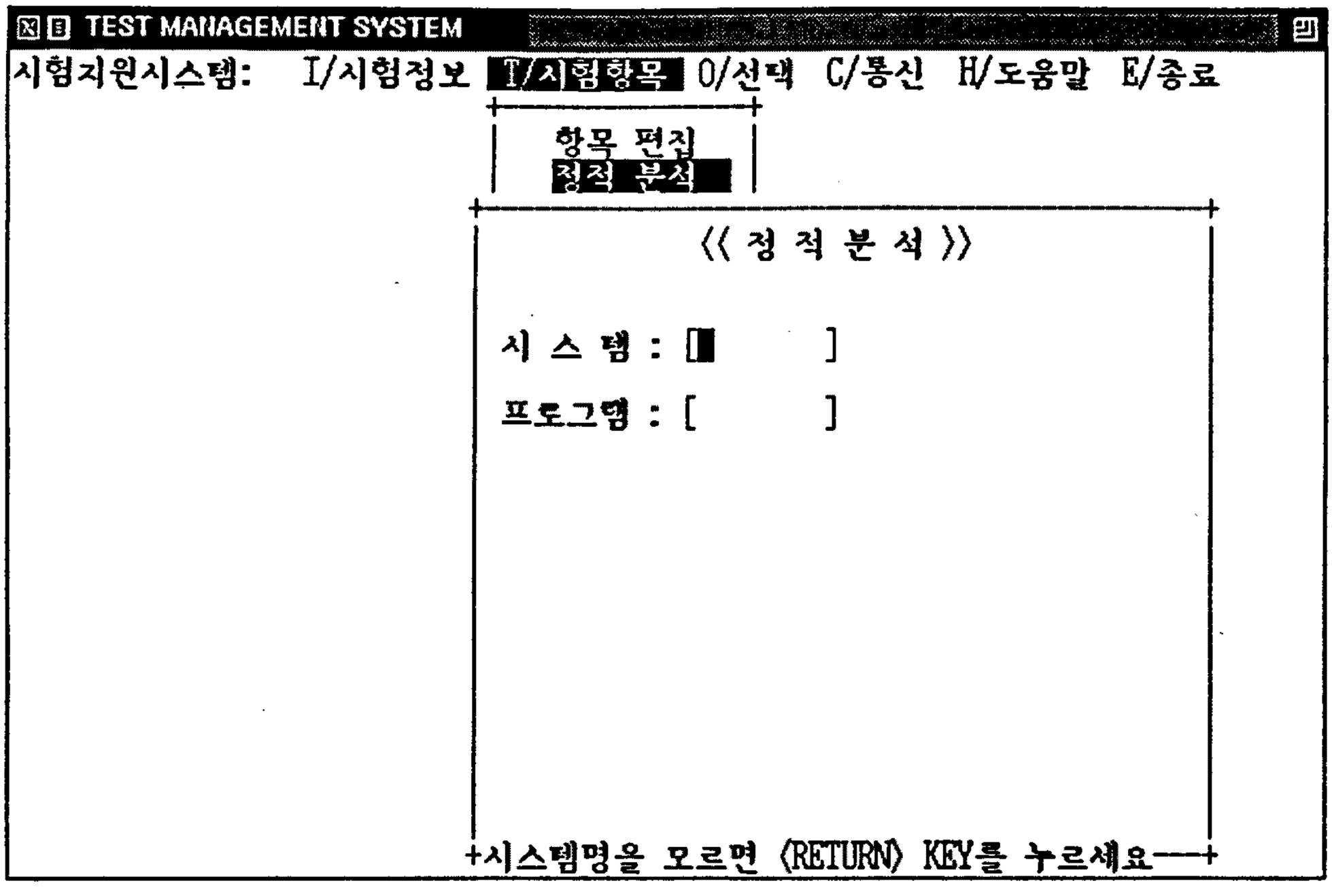
• 시스템에 해당되는 프로그램 목록을 수정, 삭제, 추가한다.

- 〈Ctrl〉+I : 새로운 프로그램을 추가 등록한다.

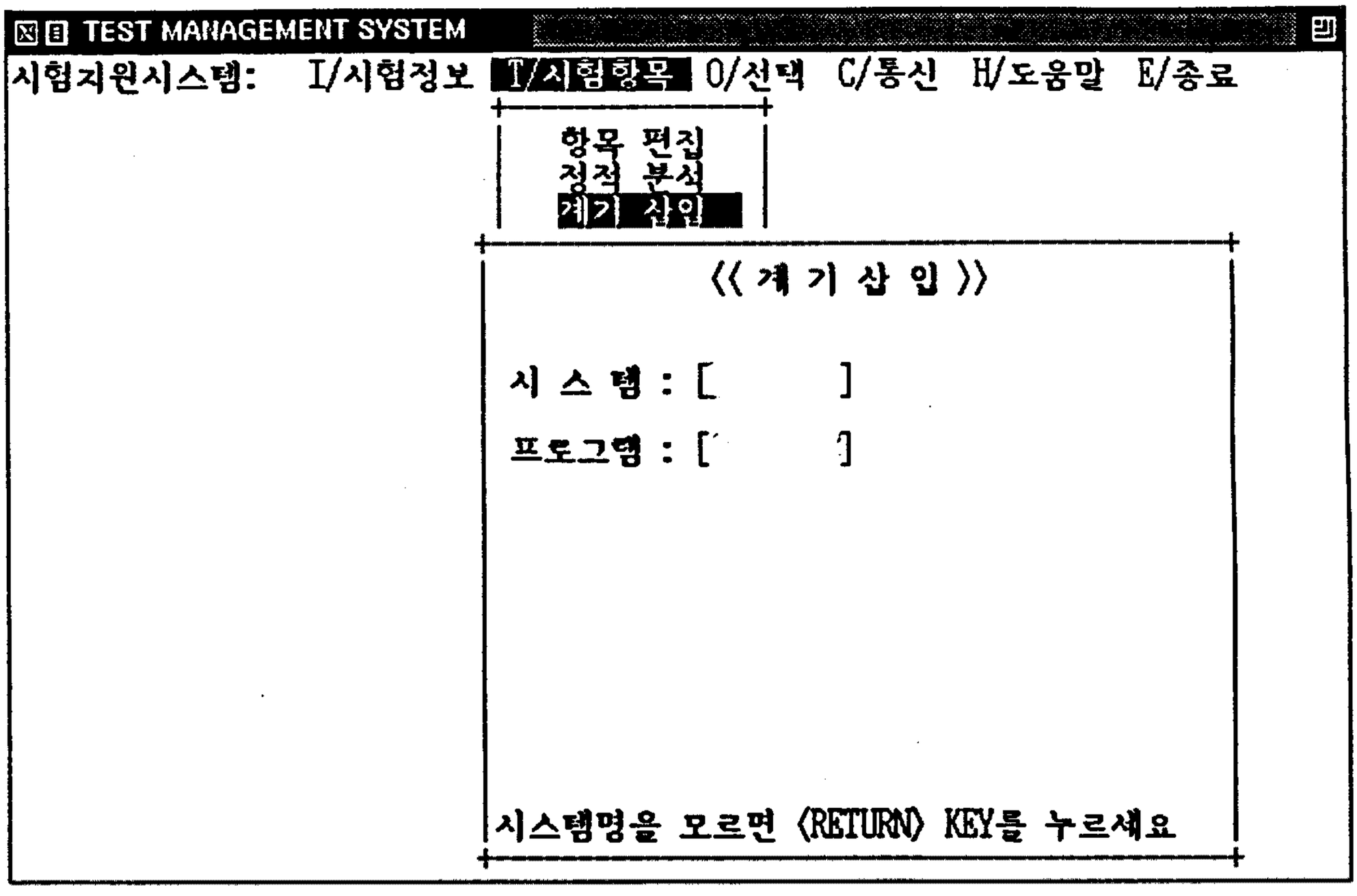
- 〈Ctrl〉+C : 하나의 프로그램을 삭제한다.

- 〈ESC〉 : 추가, 삭제, 수정작업을 실행시킨다.

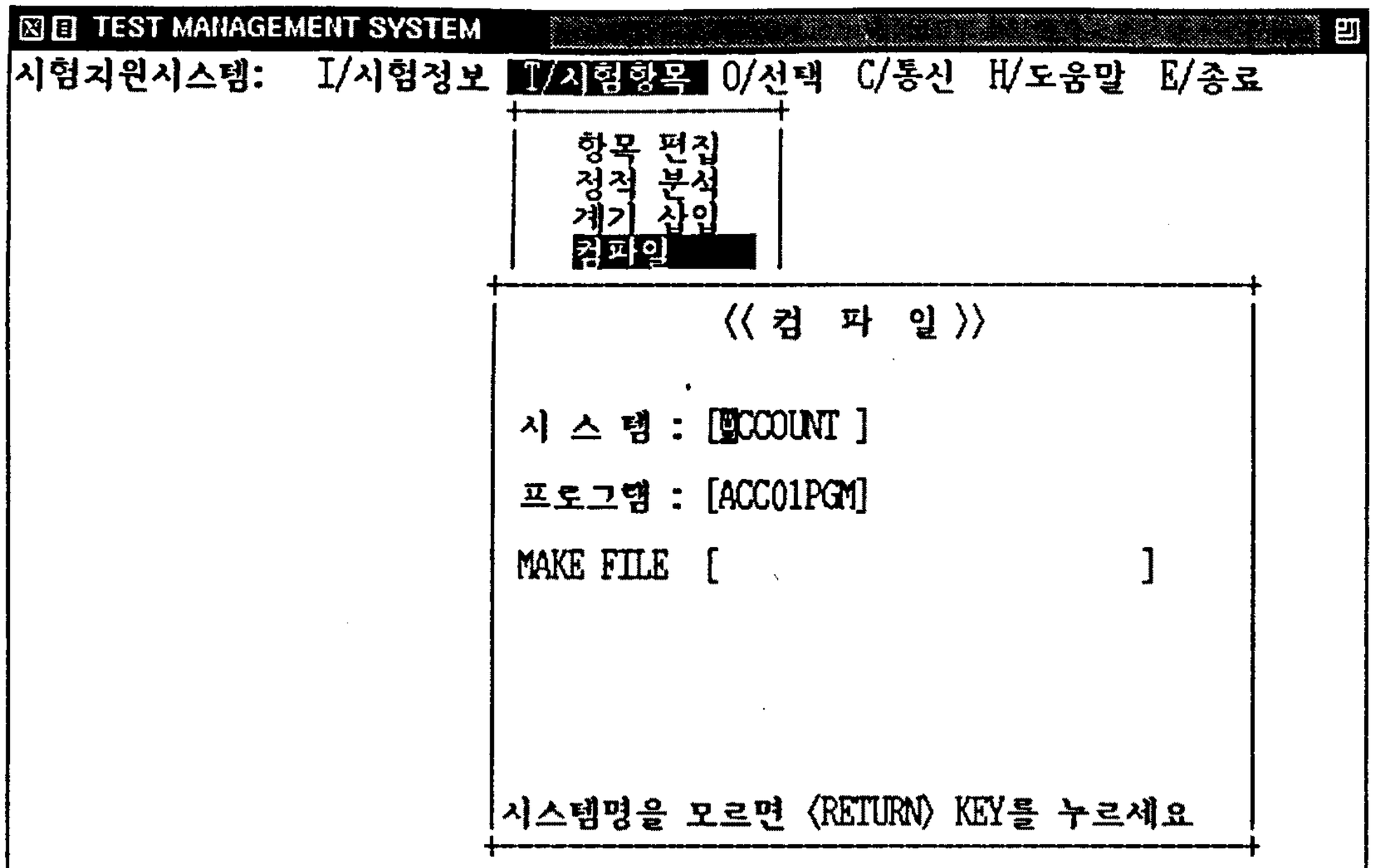
- 〈Del〉 : 추가, 삭제, 수정 작업한 이전의 상태를 유지한다.



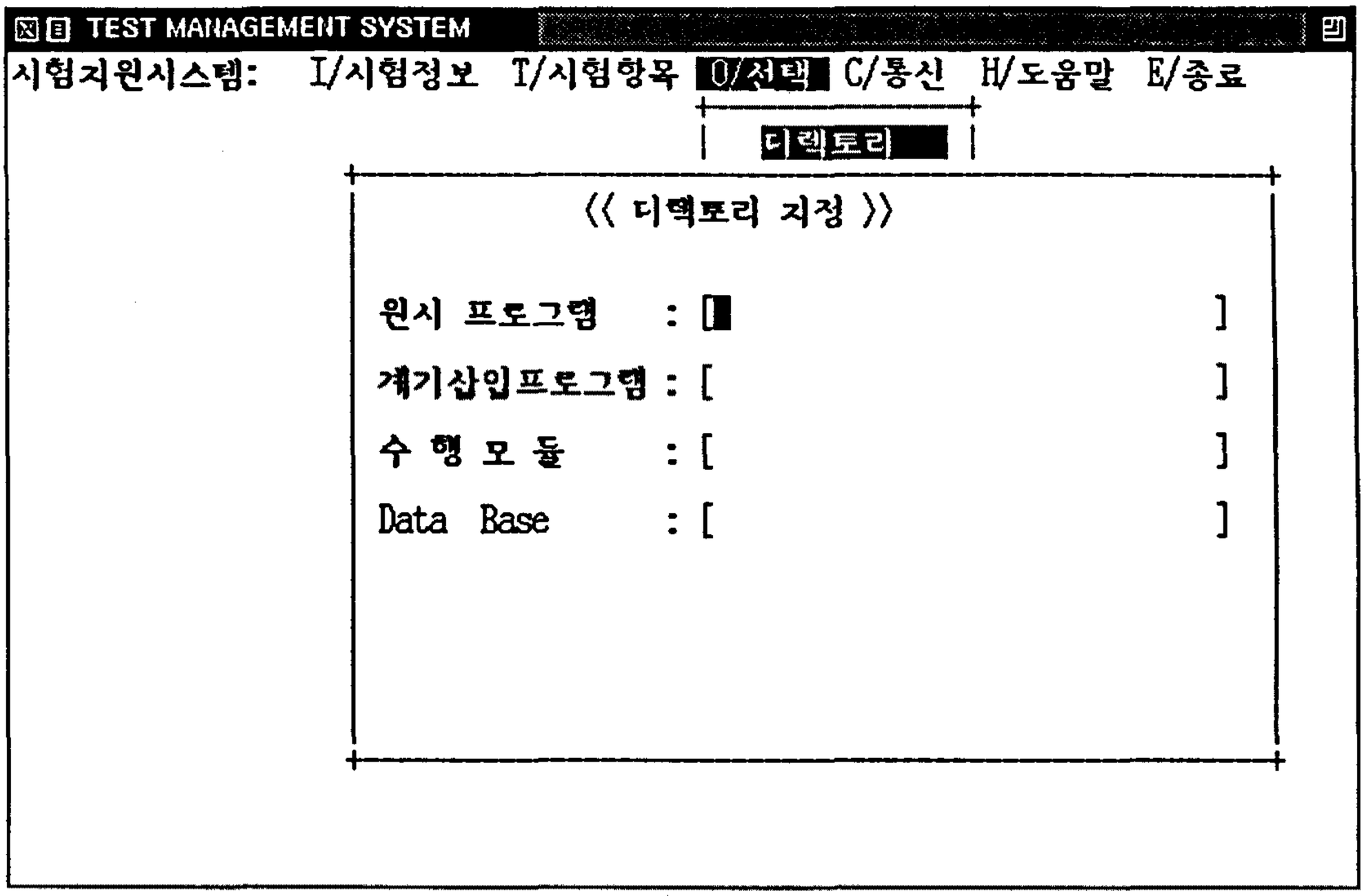
- 이 화면은 시험 항목의 정적 분석을 수행하는 화면이다.
- 시스템과 프로그램을 입력한 후 〈Return〉 key를 누르면 정적 분석 상황이 화면에 나타난다.
- 시스템명과 프로그램명에 '*'를 입력하면 각각에 대한 안내 화면이 나와, 원하는 항목을 선택할 수 있다.
- 종료하고 싶을 경우는 〈Del〉 Key를 누른다.
- 〈Del〉 Key를 누르면 이 화면을 호출한 시험항목 선택 화면이 나타난다.



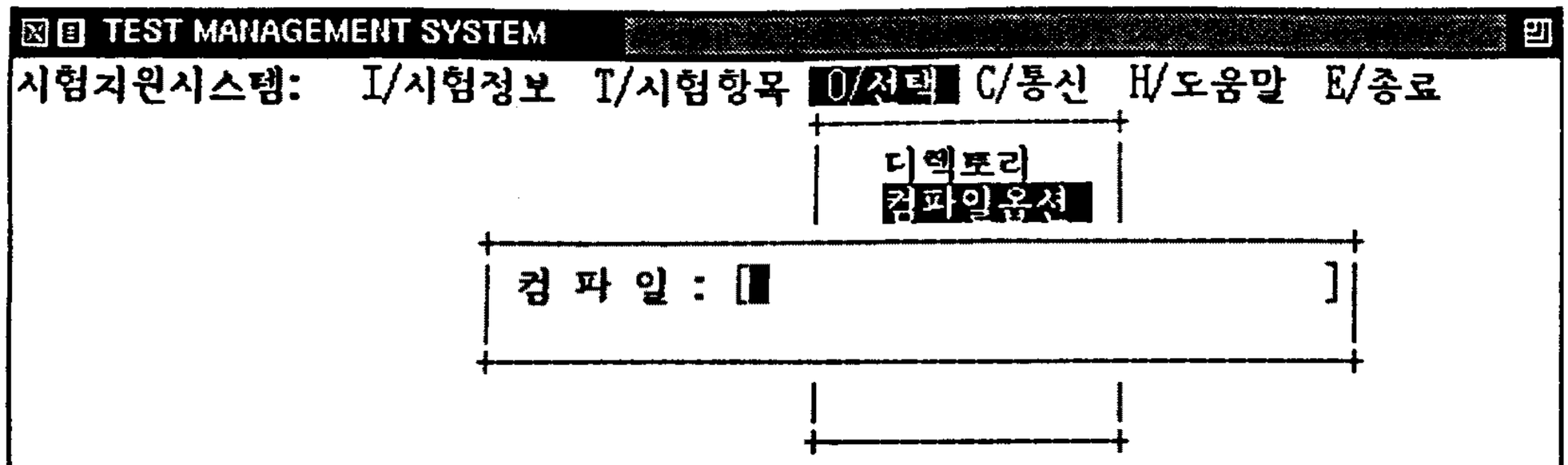
- 이 화면은 시험항목의 계기삽입을 수행하는 화면이다.
- 시스템명과 프로그램을 입력한 후 〈Return〉 Key를 누르면 실행된다.
- 시스템과 프로그램 안내는 정적 분석과 동일하다.
- 종료하고 싶을 경우는 〈Del〉 Key를 누른다.
- 〈Del〉 Key를 누르면 이 화면을 호출한 시험항목 선택 화면이 나타난다.



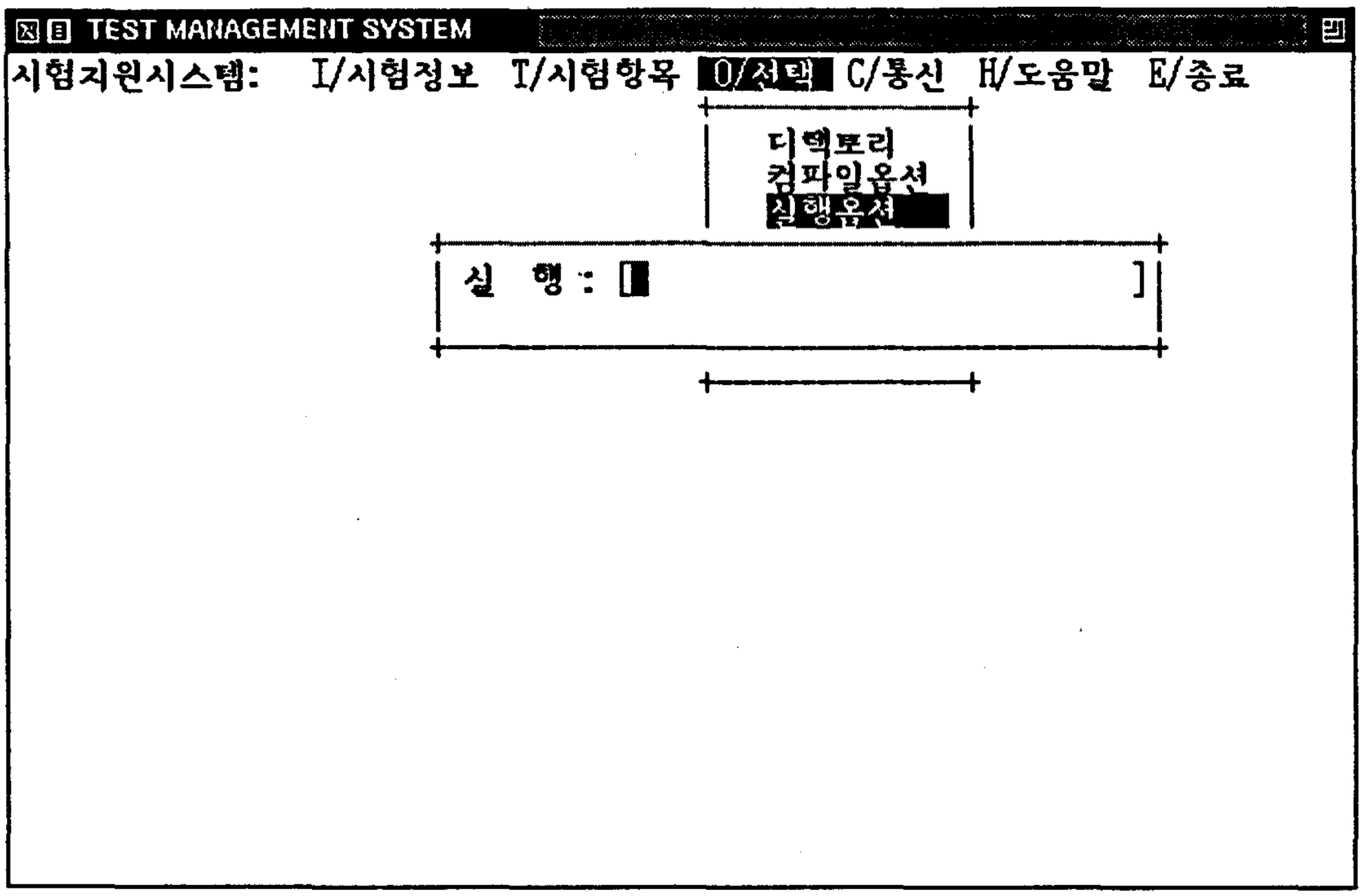
- 이 화면은 시험항목의 컴파일 화면이다.
- 시스템 프로그램, MAKE FILE을 입력한 후 〈Return〉 Key를 누르면 실행된다.
- 입력된 MAKE FILE은 저장되어 추후 재사용 또는 수정할 수있다.
- 종료하고 싶을 경우는 〈Del〉 Key를 누른다.



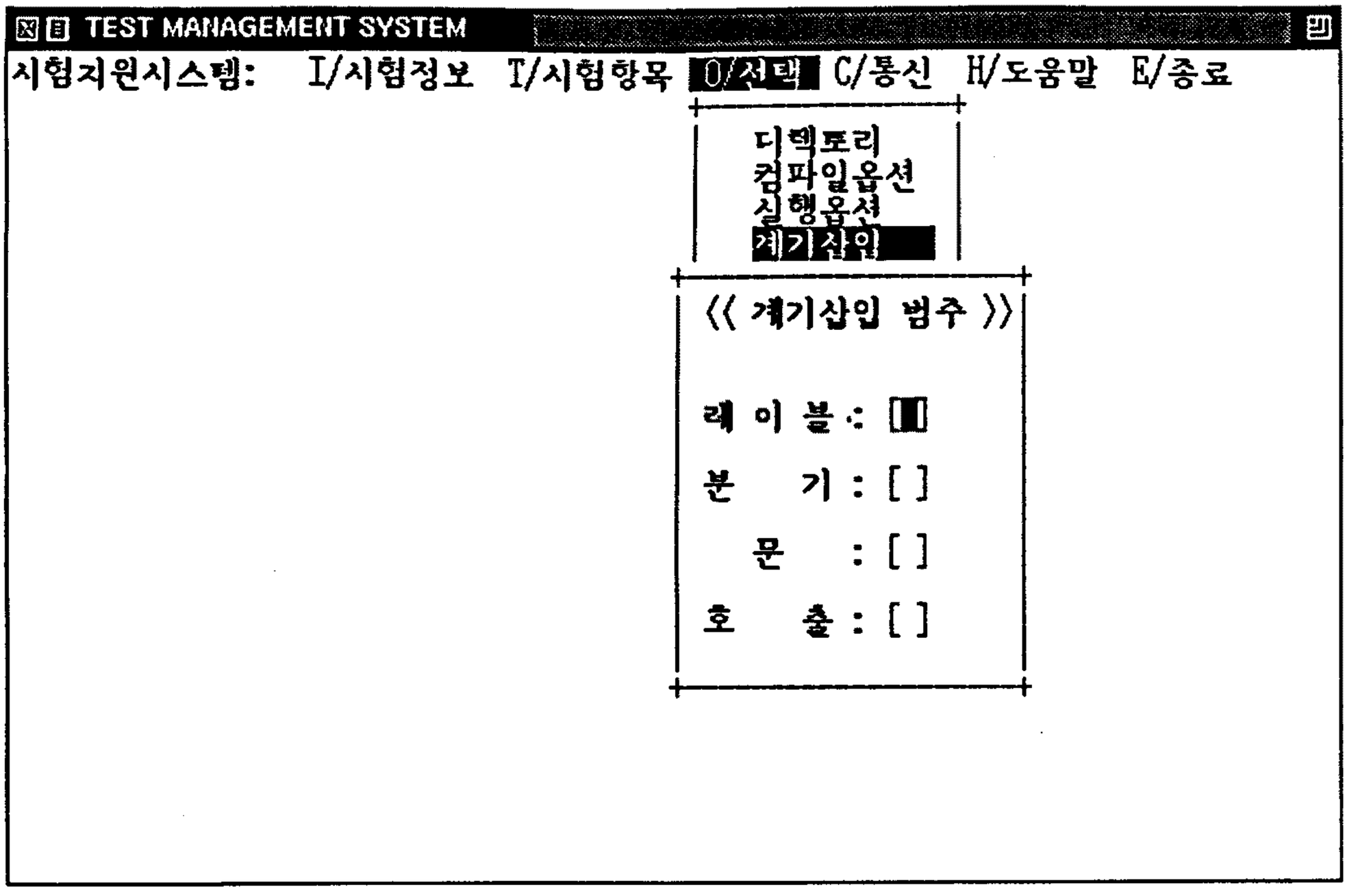
- 이 화면은 디렉토리 지정 화면이다.
- 각각의 항목을 입력한 후 〈Return〉 Key를 누르면 디렉토리가 세트되며 추후 변경 가능하다.
- 종료하고 싶을 경우는 〈Del〉 Key를 누른다.



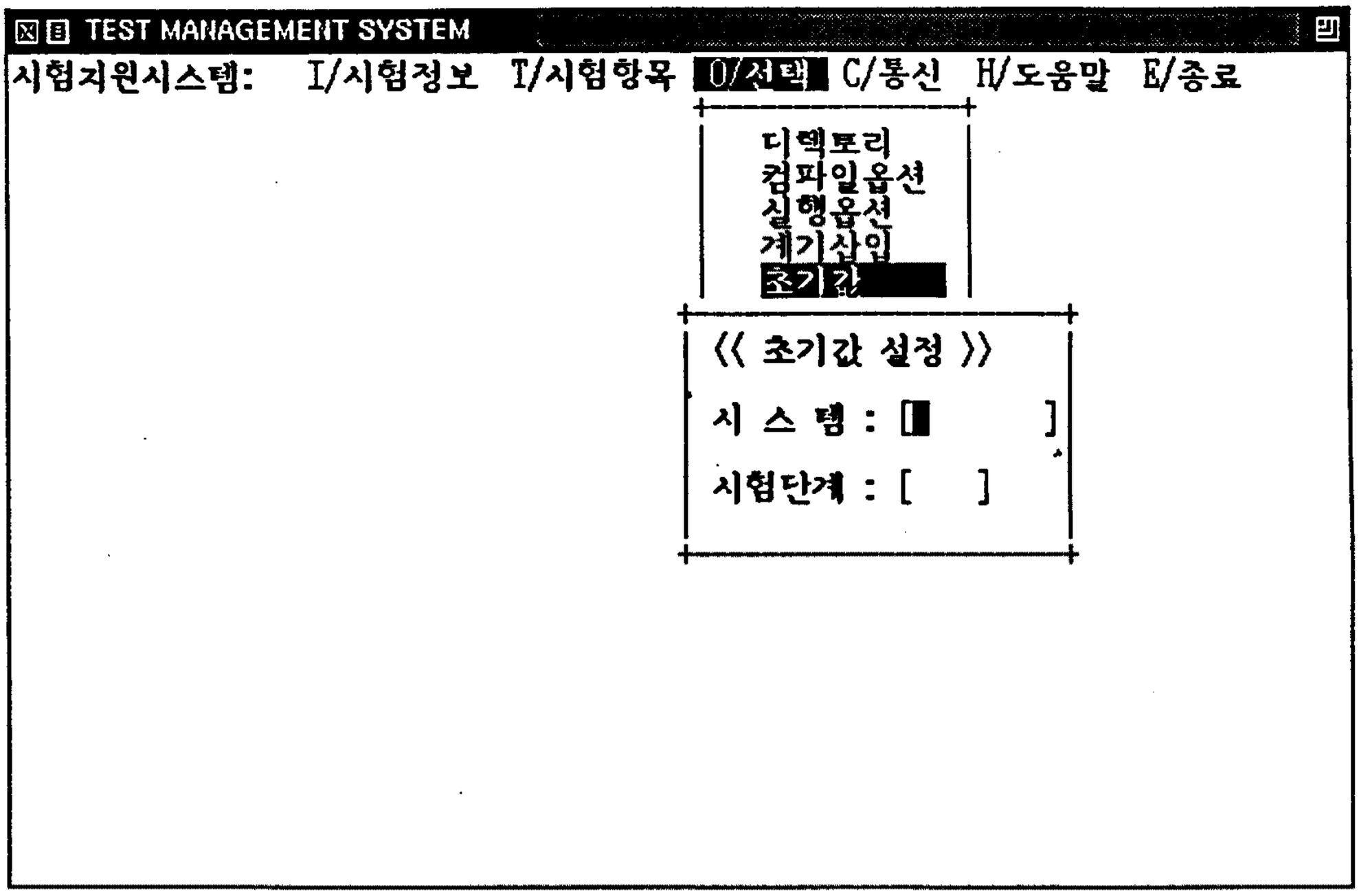
- 이 화면은 컴파일 옵션을 정의하는 화면이다.
- 컴파일 옵션을 정의한 후 〈Return〉 Key를 누르면 옵션이 저장된다.
- 종료하고 싶을 경우는 〈Del〉 Key를 누른다.



- 이 화면은 실행 옵션을 정의하는 화면이다.
- 실행 옵션을 정의한 후 〈Return〉 Key를 누르면 실행 옵션이 저장된다.
- 종료하고 싶을 경우는 〈Del〉 Key를 누른다.

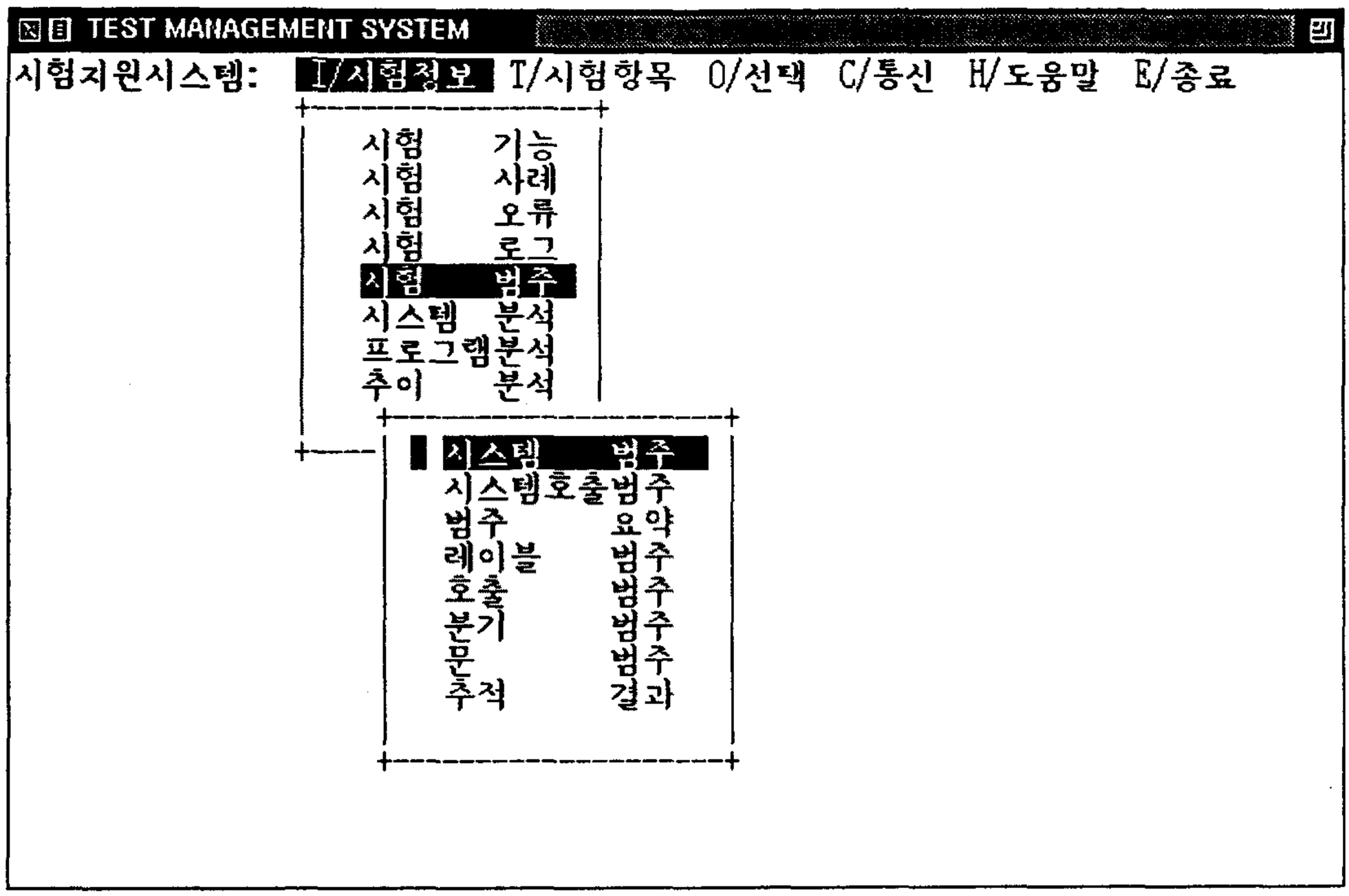


- 이 화면은 계기 삽입 범위를 지정하는 화면이다.
- Cursor가 위치한 장소에 “ y ”를 입력하면 해당 계기가 삽입되고 “ y ”가 아닌것은 수행을 안한다.
- 계기 삽입 범위는 시스템, 프로그램 단위별로는 지정할 수 없다.
- 종료하고 싶을 경우는 〈Del〉 Key를 누른다.



- 이 화면은 초기값 설정을 하는 화면이다.
- 시스템, 시험단계를 입력한 후 〈Return〉 Key를 누르면 초기값이 설정된다.
- 종료를 원할때는 〈Del〉 Key를 누르면 이 화면을 호출한 선택화면으로 돌아간다.

제 3 절 시험 범주 분석기



- 이 화면은 주 메뉴 화면에서 범주 분석 결과를 선택하는 과정을 나타낸 화면이다.
- 작업 과정은
  - " I / 시험정보 " 부분이 Reverse 된 상태에서 〈Return〉 Key 를 누르면 시험정보에 관한 메뉴가 나타난다.
  - 시험정보 메뉴가 나타난 상태에서 방향 Key 를 눌러 '시험 범주 ' 를 선택한 후 〈Return〉 Key 를 누르면 그에 따른 해당 작업이 나타난다.
  - 원하는 출력을 선택하여 〈Return〉 Key 를 누른다.

TEST MANAGEMENT SYSTEM

시험지원시스템: I/시험정보 T/시험항목 O/선택 C/통신 H/도움말 E/종료

〈〈 시스템 범주 〉〉

시 스 템 : account

프로그램수 : 10 수행된수 : 2 수행안된수 : 8 % 범주 : 20

라인	프로그램	횟수	20 40 60 80 100	비 고
1	ac03pgm	0		xxx
2	ac01pgm	3	*	
3	ac02pgm	1	*	
4	ac04pgm	0		xxx
5	ac05pgm	0		xxx
6	ac06pgm	0		xxx
7	ac07pgm	0		xxx
8	ac08pgm	0		xxx
9	ac09pgm	0		xxx
10	ac10pgm	0		xxx

- 이 화면은 시스템 범주 출력 화면으로 선택된 시스템 내의 프로그램들에 대한 수행 횟수와 미 시험 구분, 최초 수행 구분과 요약 범주를 출력한다.
- 시스템 선택시에 시스템에 대한 도움을 받고 싶은 경우에는 공란 혹은 '*'를 입력한 상태에서 Enter Key를 누르면 시스템 도움 화면이 출력되며 이 화면에서 시스템을 선택할 수 있다.
- 비고

  ×××: 수행안됨

TEST MANAGEMENT SYSTEM

시험지원시스템: I/시험정보 T/시험항목 O/선택 C/통신 H/도움말 E/종료

〈〈 호출 범주 〉〉

시 스 템 : account

프로그램수: 2 수행된수 : 1 수행안된수 : 1 % 범주 : 50

	라인	프로그램	호 출	횟수	20 40 60 80 100	비 고
■	1	ac01pgm	AMSCC01	10	**	
	2	ac01pgm	AMSR02	0		XXX
	3	ac01pgm	FEEDEL00	15	***	
	4	ac01pgm	COMB610E	2	*	
	5	ac01pgm	TASIL10	0		XXX
	6	ac01pgm	MAMD030	26	******	
	7	ac01pgm	MCTR080	15	***	
	8	ac01pgm	MDEL040	40	********	
	9	ac01pgm	PNPRT	23	*****	
	10	ac01pgm	QPOECI3	21	*****	
	11	ac01pgm	APOSAP1	14	***	

• 시스템내의 프로그램별 호출 관계에 대한 수행 범주 출력 화면으로, 하나의 프로그램 내에서 부 프로그램이 여러 위치에서 호출되는 경우에는 각각의 횟수를 누적하여 출력한다.

• 시스템을 선택하는 데 선택 방법은 시스템 범주와 같다.

• 회수 : 수행된 횟수이다.

• 도표(*) : 수행된 횟수에 대한 표시이다.

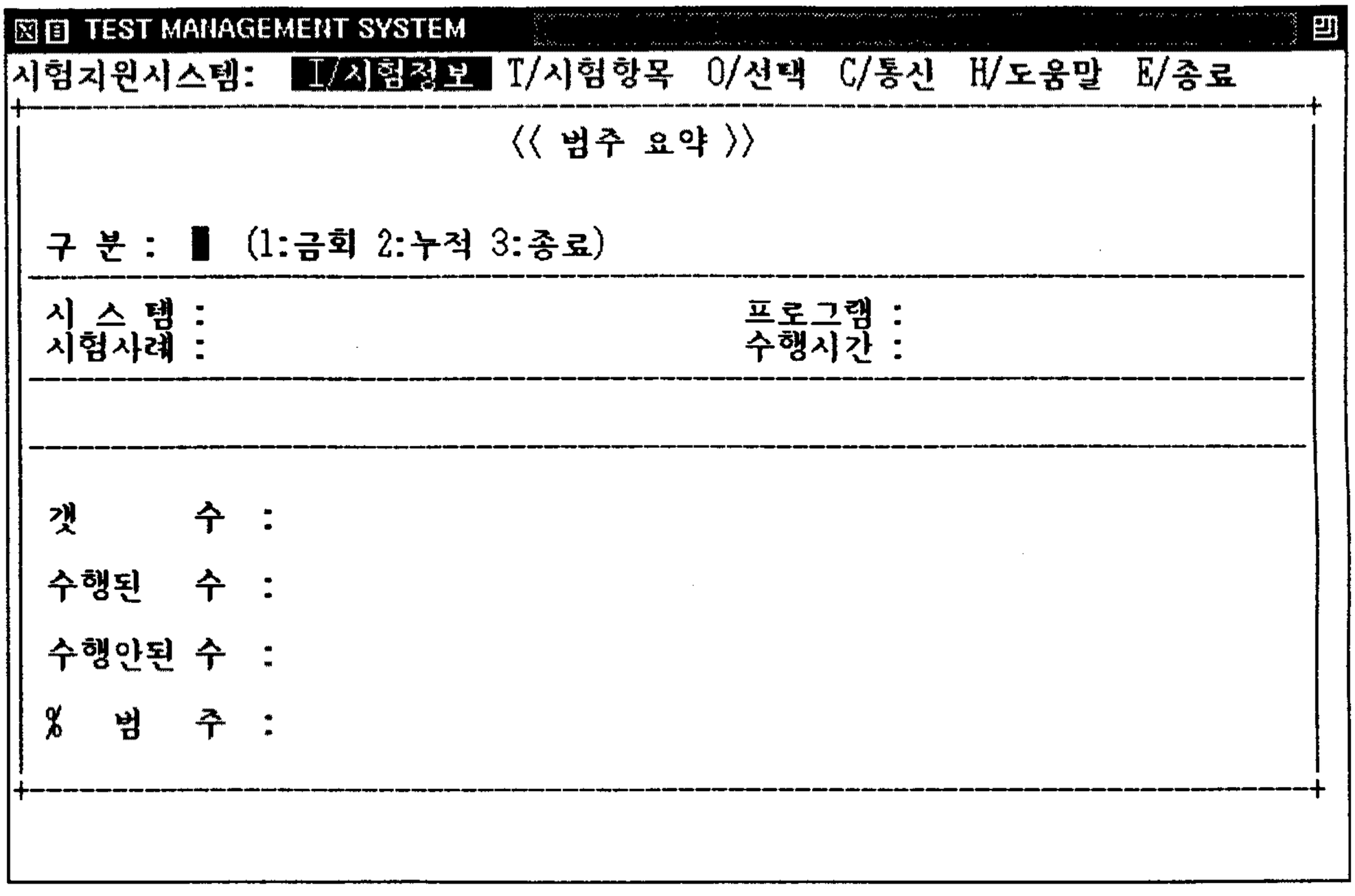
• 비고 : ××× 수행이 안됨.

• 검색된 자료들의

- 다음 화면을 보고 싶으면 〈Ctrl+F〉 Key를 누른다.

- 전 화면을 보고 싶으면 〈Ctrl+B〉 Key를 누른다.

• 작업을 끝내고 싶을 경우는 〈Esc〉 Key를 누른다.



• 이 화면은 범주 요약 화면이다.

• 메뉴화면에서 범주 요약을 선택하면 이 화면이 나타난다.

• Cursor의 위치가 구분에 있는 상태에서 구분을 선택한다.

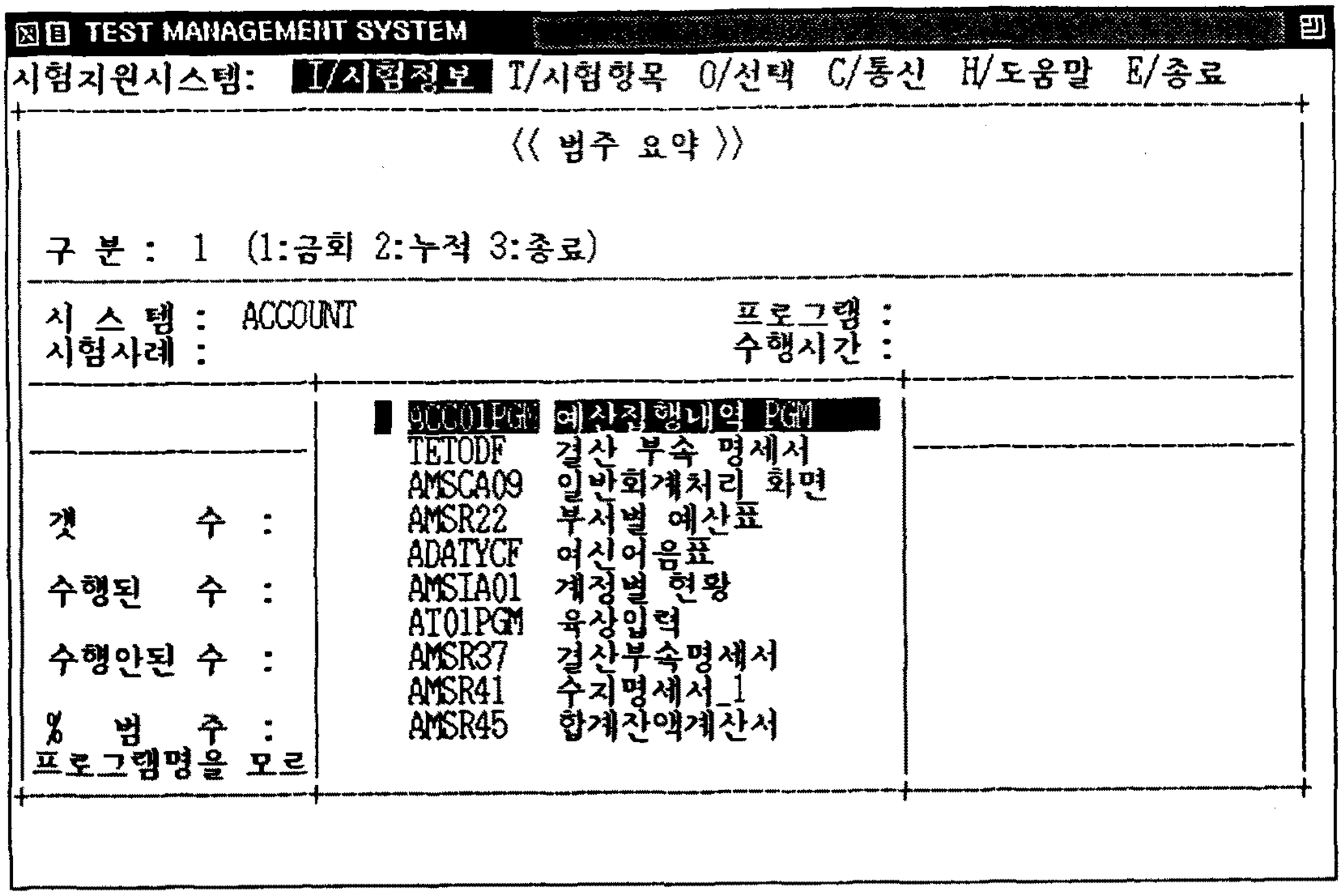
• 구분을 선택한 다음에 시스템을 선택 하는데

  - 처음에는 초기에 입력한 시스템이 나타난다.

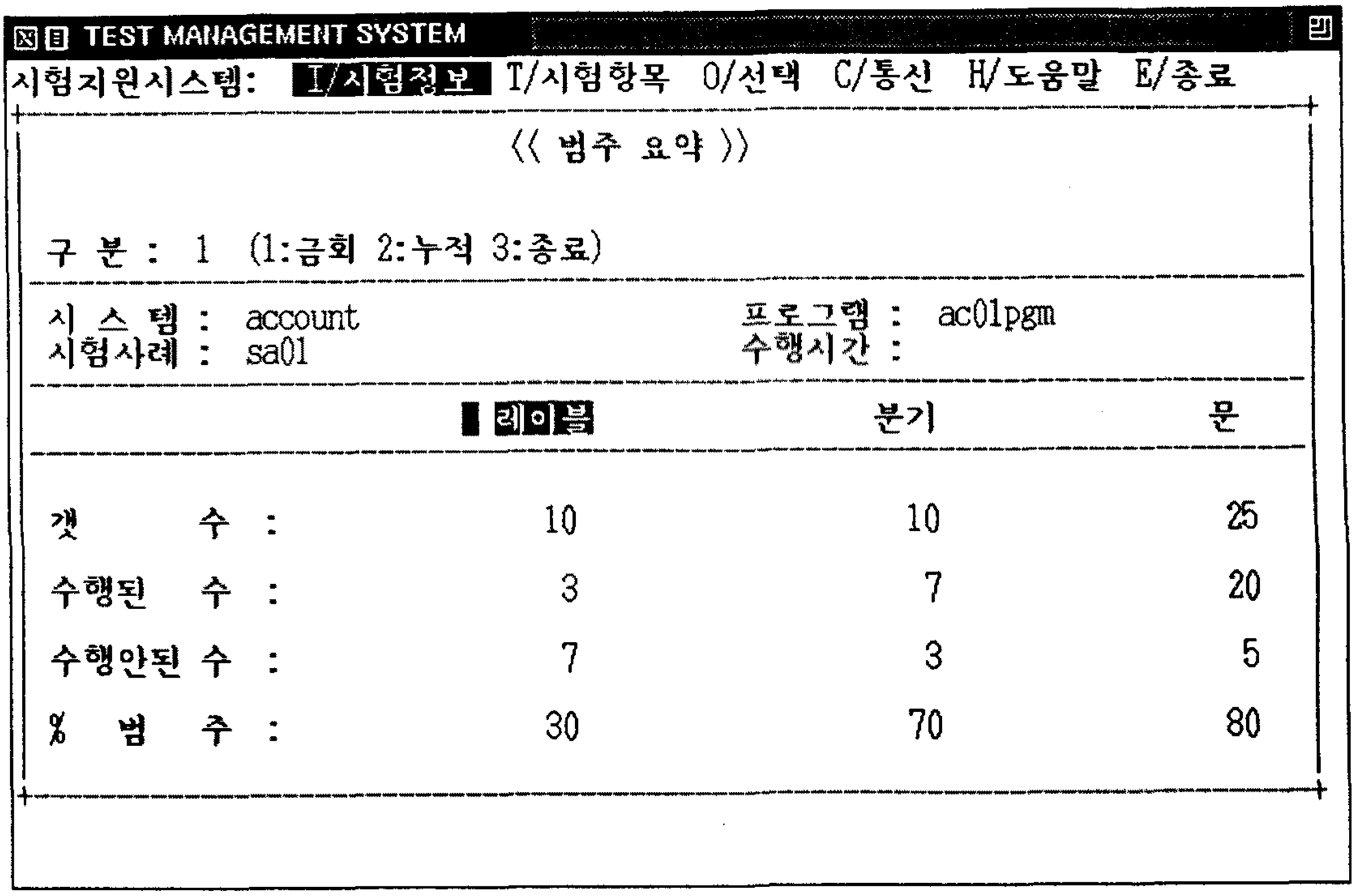
  - 시스템을 수정할 경우

    - 시스템을 알고 있을 경우는 시스템을 직접 입력한다.

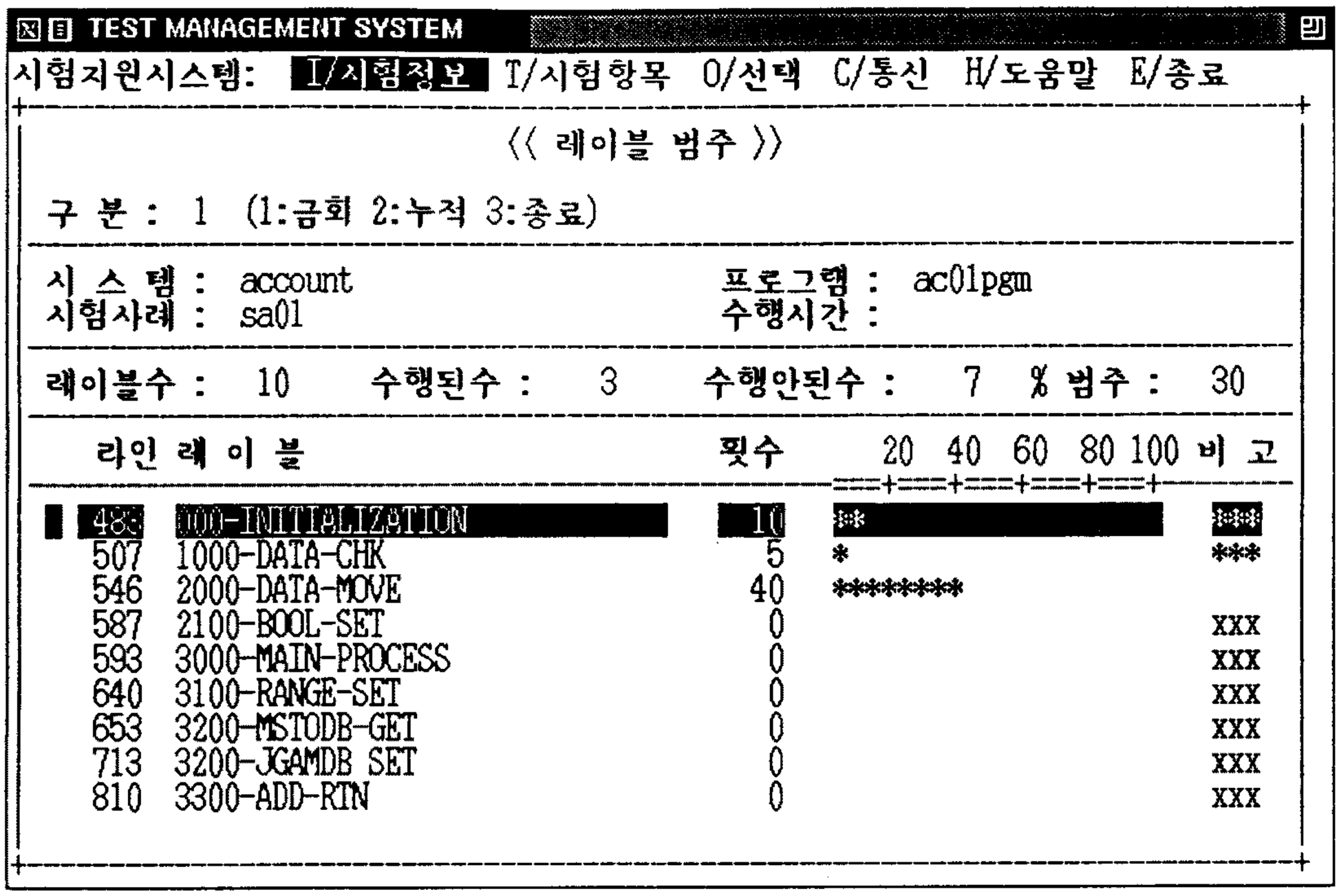
    - 시스템을 모르는 경우에는 ' * '를 입력한 후 〈Return〉Key를 누른다.



- 시스템을 선택한 다음에 프로그램을 선택하는 데 선택 방법은 시스템과 같다.
- 이 화면은 프로그램 선택 위치에서 '*'를 입력한 경우이다.
- 시험사례를 선택 했으면 〈Return〉 Key를 누른다.
- 〈Return〉 Key를 누르면 선택된 시험사례가 표시되고 위에서 선택한 조건들의 해당 자료들이 검색되어 화면에 나타난다.



- 이 화면은 프로그램별로 레이블, 분기, 문에 대한 요약 결과 출력 화면이다.
- 갯 수 : 레이블, 분기, 문 등의 전체갯수를 나타낸다.
- 수행된 수 : 〃 수행된 갯수를 나타낸다.
- 수행안된수 : 〃 수행안된 갯수를 나타낸다.
- % 범 주 : 전체갯수 대 수행된 수를 %로 나타낸다.
- 레이블, 분기, 문 등에 대해서 자세한 정보를 보고 싶을 경우에는 해당 필드가 Reverse된 상태에서 "Y"를 입력하고 〈Return〉 Key를 누르면 레이블 범주, 분기 범주 및 문 범주 화면으로 전이된다. 전이된 화면에서 작업을 종료하면 다시 이 화면으로 복귀한다.



• 이 화면은 레이블 범주 분석 화면이다.

• 구분을 선택한 다음에 시스템을 선택 하는데

- 처음에는 초기에 입력한 시스템이 나타난다.

- 시스템을 수정할 경우

- 시스템을 알고있을 경우는 시스템을 직접 입력한다.

- 시스템을 모르는 경우는 ' * '를 입력한 후 〈Return〉 Key를 누른다.

• 시스템이 화면에 나타난 상태에서 Cursor를 움직여 원하는 시스템을 선택한다.

• 시스템이 선택 되었으면 〈Return〉 Key를 누른다.

• 시스템을 선택한 다음에 프로그램과 시험사례를 선택하는 데 선택 방법은 시스템과 같다.

• 시험사례를 선택한후 〈Return〉 Key를 누른다.

• 〈Return〉 Key를 누르면 선택된 시험사례가 표시되고
위에서 선택한 조건들의 해당 자료들이 검색되어 화면에 나타난다.

• 회 수 : 수행된 횟수이다.

• 도표(*) : 수행된 횟수에 대한 표시이다.(1개당 5회 예 : 횟수가 8이면 **)

• 비 고 : ××× 수행이 안됨.

××× 일부수행

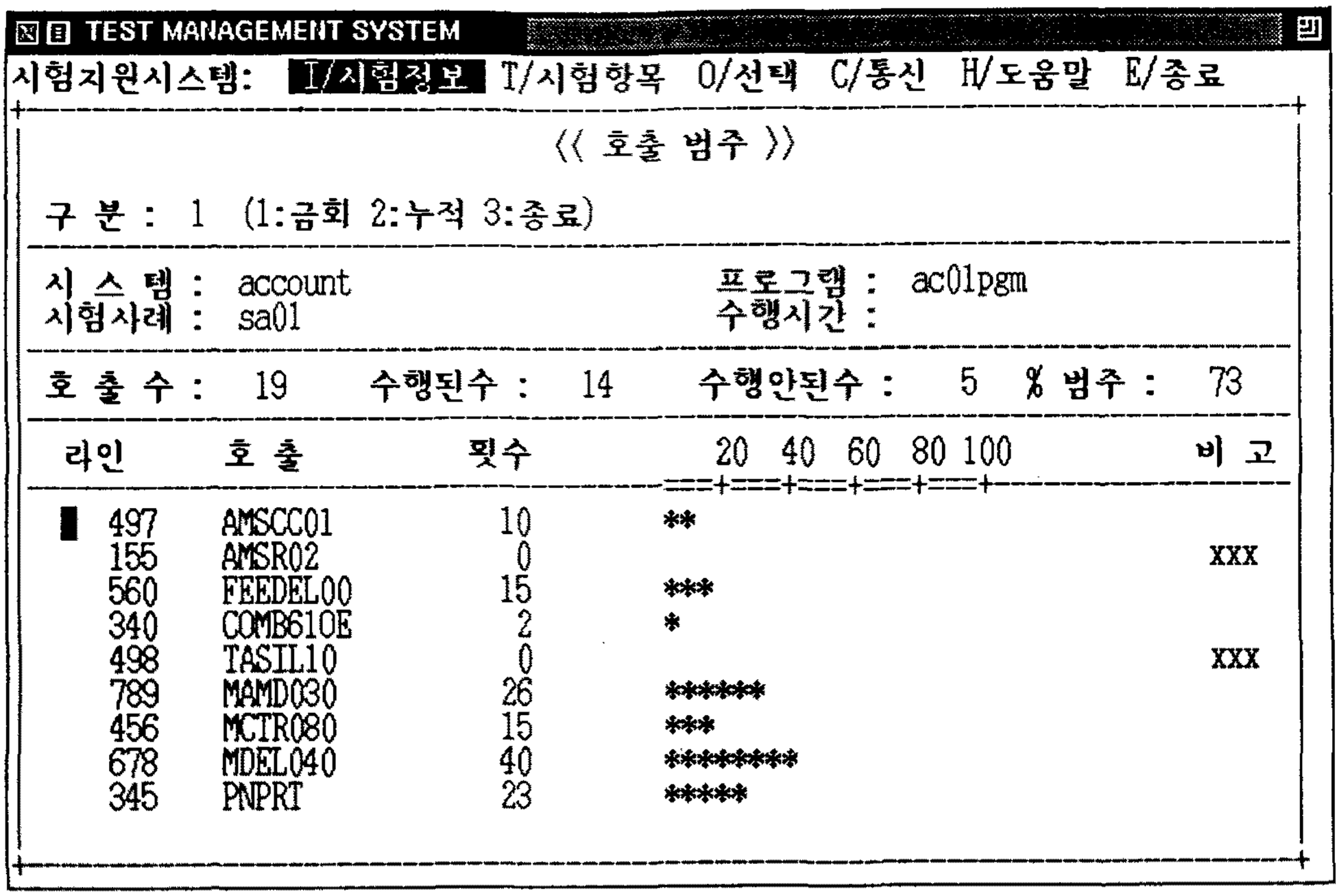
• 검색된 자료들의

- 다음 화면을 보고싶으면 〈Ctrl+F〉 Key를 누른다.

- 전 화면을 보고싶으면 〈Ctrl+B〉 Key를 누른다.

- 화면에 나타난 레이블에 대한 자세한 정보를 보고 싶으면 해당 레이블로 Cursor를 옮겨서 〈Return〉 Key를 누른다. 이 경우 해당 레이블에 대한 문 범주가 출력되며 작업 종료시 다시 이 화면으로 복귀한다.

• 해당 작업을 끝내고 싶을 경우는 〈Esc〉 Key를 누른다.



- 프로그램 내의 부 프로그램 호출별로 수행 범주를 출력하는 화면이다.
- 구분, 시스템, 프로그램, 시험사례를 선택 하며, 선택 방법은 범주 요약과 같다.
- 회 수 : 수행된 횟수이다.
- 도표(*) : 수행된 횟수에 대한 표시이다.
- 비 고 : ××× 수행이 안됨.
- 검색된 자료들의
  - 다음 화면을 보고 싶으면 〈Ctrl+F〉 Key를 누른다.
  - 전 화면을 보고 싶으면 〈Ctrl+B〉 Key를 누른다.
- 해당 작업을 끝내고 싶을 경우는 〈Esc〉 Key를 누른다.

TEST MANAGEMENT SYSTEM

시험지원시스템: I/시험정보 T/시험항목 O/선택 C/통신 H/도움말 E/종료

〈〈 분기 범주 〉〉

구 분 : 1 (1:금회 2:누적 3:종료)

시 스 템 : account 프로그램 : ac01pgm
시험사례 : sa01 수행시간 :

분 기 수 : 10 수행된수 : 7 수행안된수 : 3 % 범주 : 70

	라인	조 건	횟수	20 40 60 80 100	비 고
▌	483	000-INITIALIZATION			
	490	True	50	**********	
	495	False	30	******	
	507	1000-DATA-CHK			
	509	True	0		XXX
	520	False	67	**************	
	546	2000-DATA-MOVE			
	549	True	0		XXX
	560	False	12	***	

• 프로그램 내의 각 패러그래프 단위로, 분기 수행 범주를 출력하는 화면이다.

• 구분, 시스템, 프로그램, 시험사례를 선택하는 데 선택 방법은 범주요약과 같다.

• 회 수 : 수행된 횟수를 나타낸다.

• 도표(*) : 수행된 횟수에 대한 표시이다.

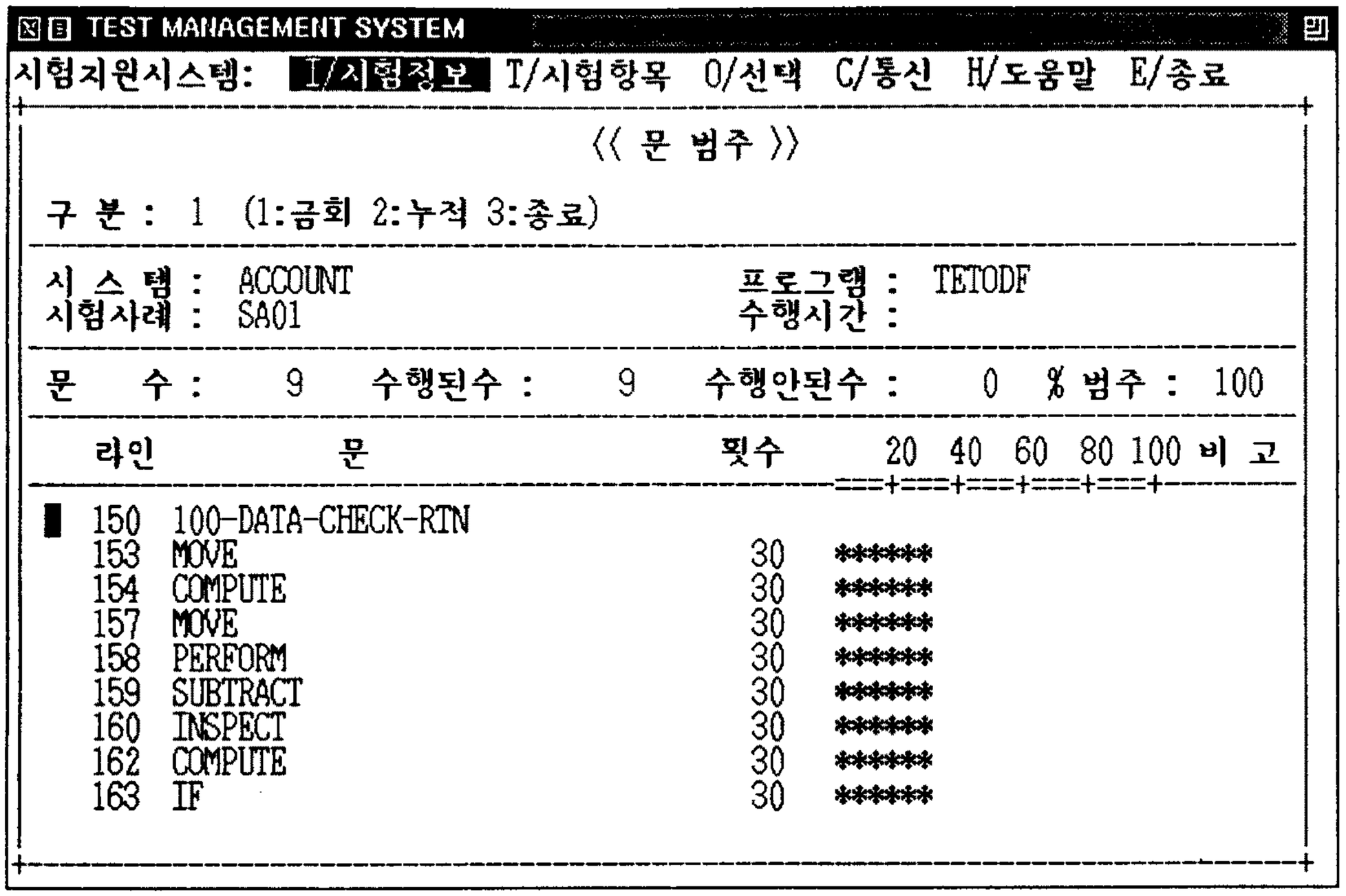
• 비 고 : ××× 수행이 안됨.

• 검색된 자료들의

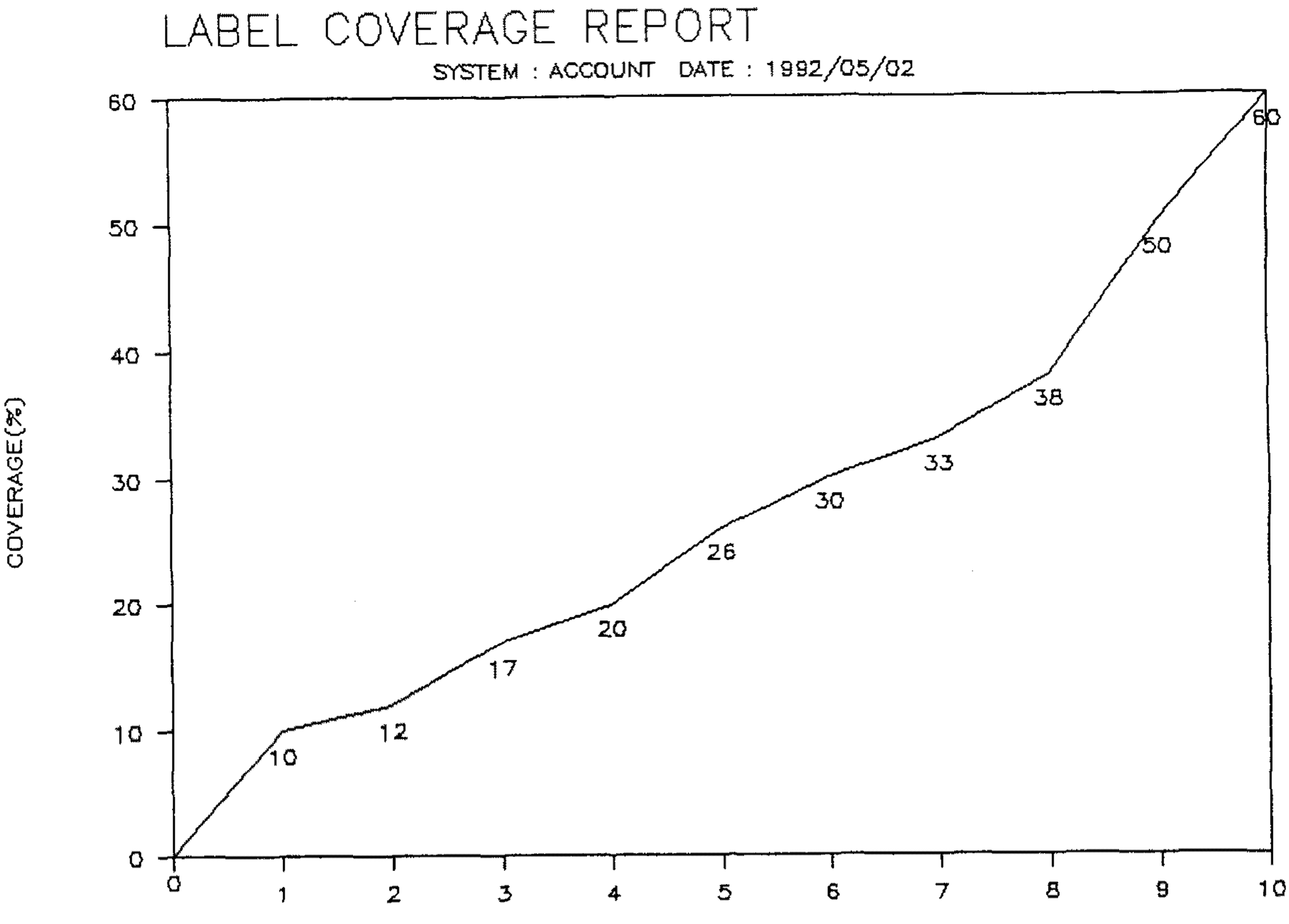
- 다음 화면을 보고싶으면 〈Ctrl+F〉 Key를 누른다.

- 전 화면을 보고싶으면 〈Ctrl+B〉 Key를 누른다.

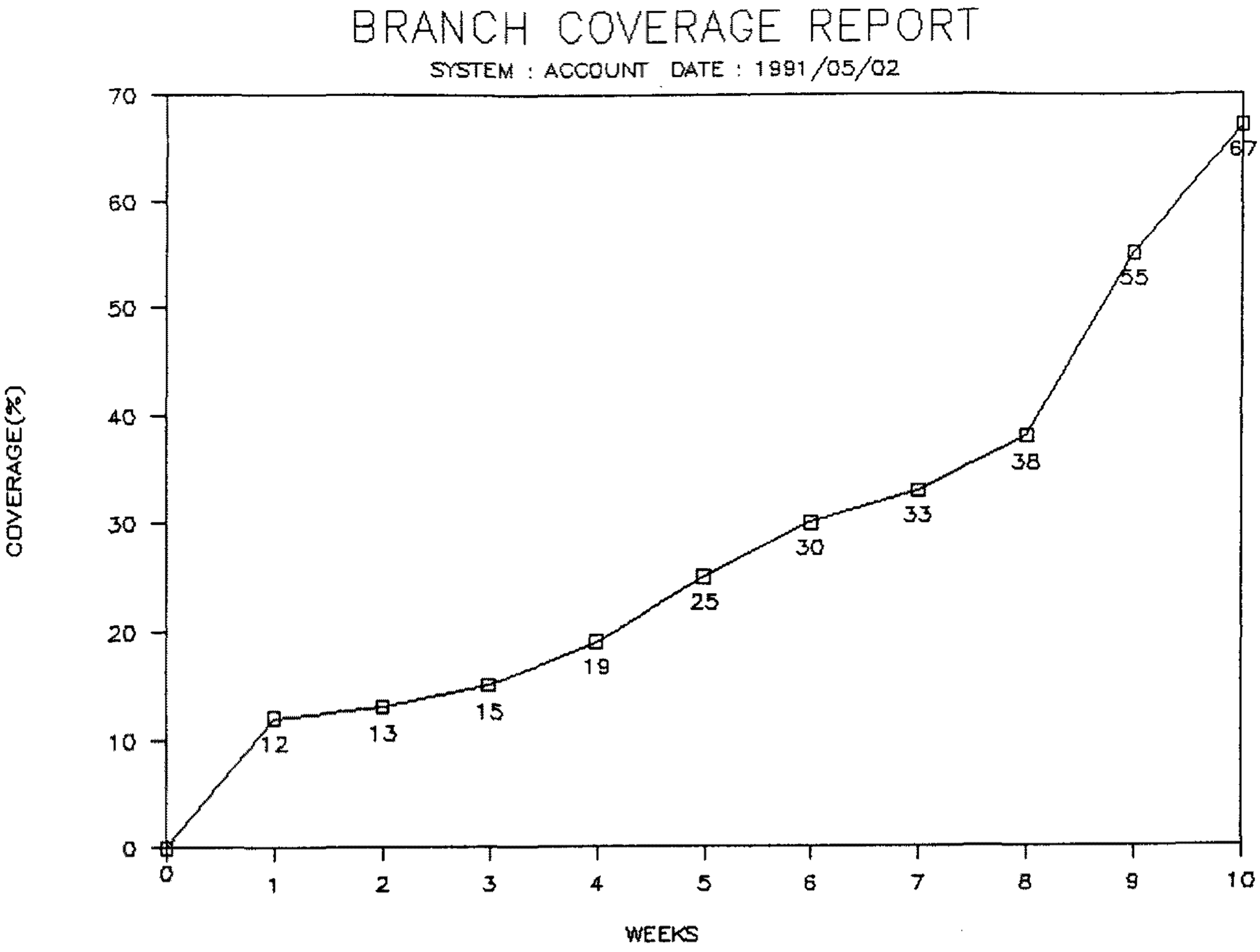
• 작업을 끝내고 싶을 경우는 〈Esc〉 Key를 누른다.



- 프로그램 내의 문별 수행 범주 출력 화면으로서, 문은 패러그래프 단위로 구분된다.
- 구분, 시스템, 프로그램, 시험사례를 선택 하는데 선택 방법은 범주요약과 같다.
- 문 ; 문의 종류를 나타낸다.
- 회 수; 수행된 횟수를 나타낸다.
- 도표(*); 수행된 횟수에 대한 표시이다.
- 비 고; ××× 수행이 안됨.
- 검색된 자료들의
  - 다음 화면을 보고 싶으면 〈Ctrl+F〉Key를 누른다.
  - 전 화면을 보고 싶으면 〈Ctrl+B〉Key를 누른다.
- 작업을 끝내고 싶을 경우는 〈Esc〉 Key를 누른다.

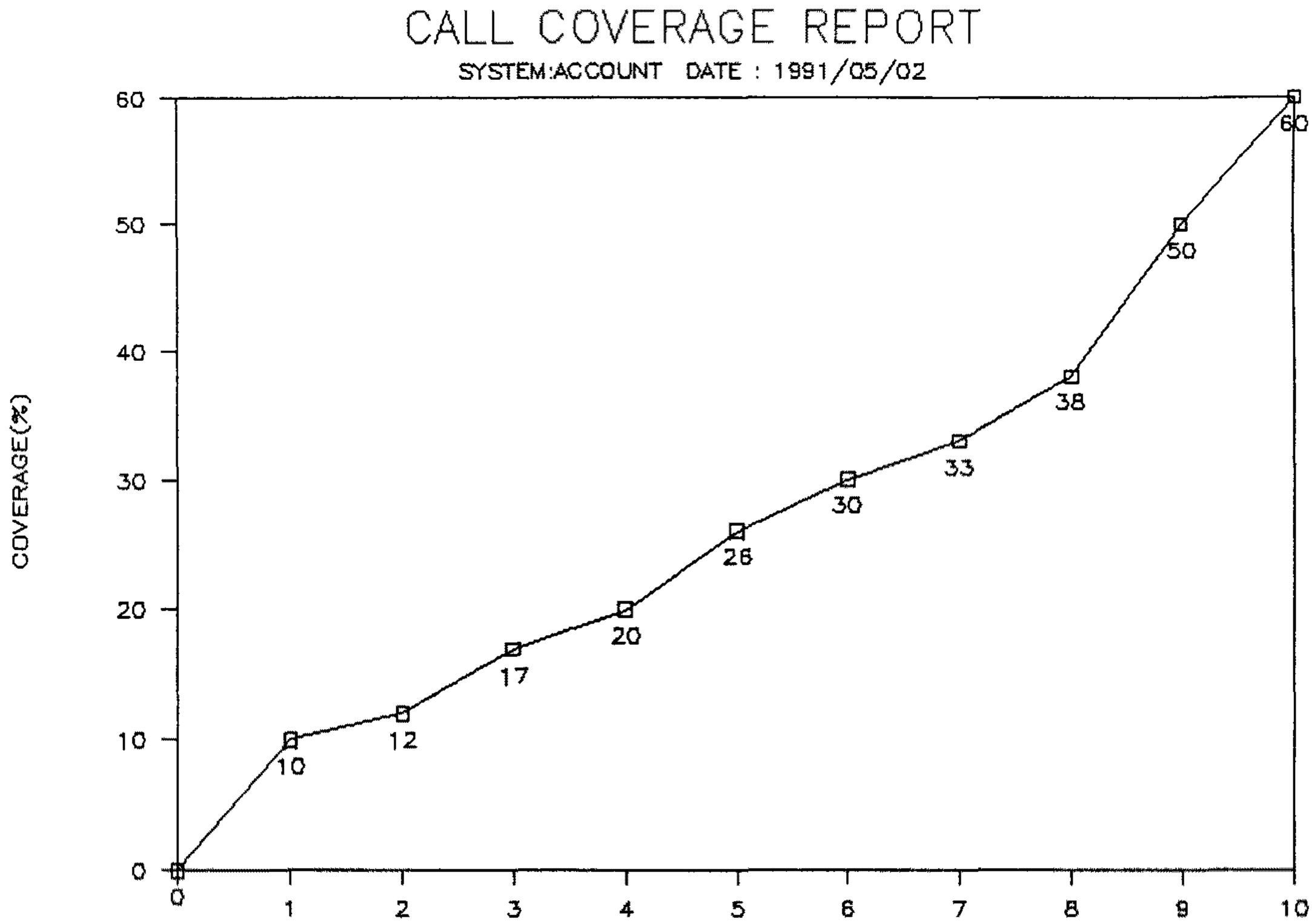


- 이 화면은 레이블 범주를 도표로 나타낸 화면이다.
- 시스템에 대한 레이블 범주 시험율을 주 단위로 보여준다.
  (시험된 레이블 수/총 레이블 수의 백분율)



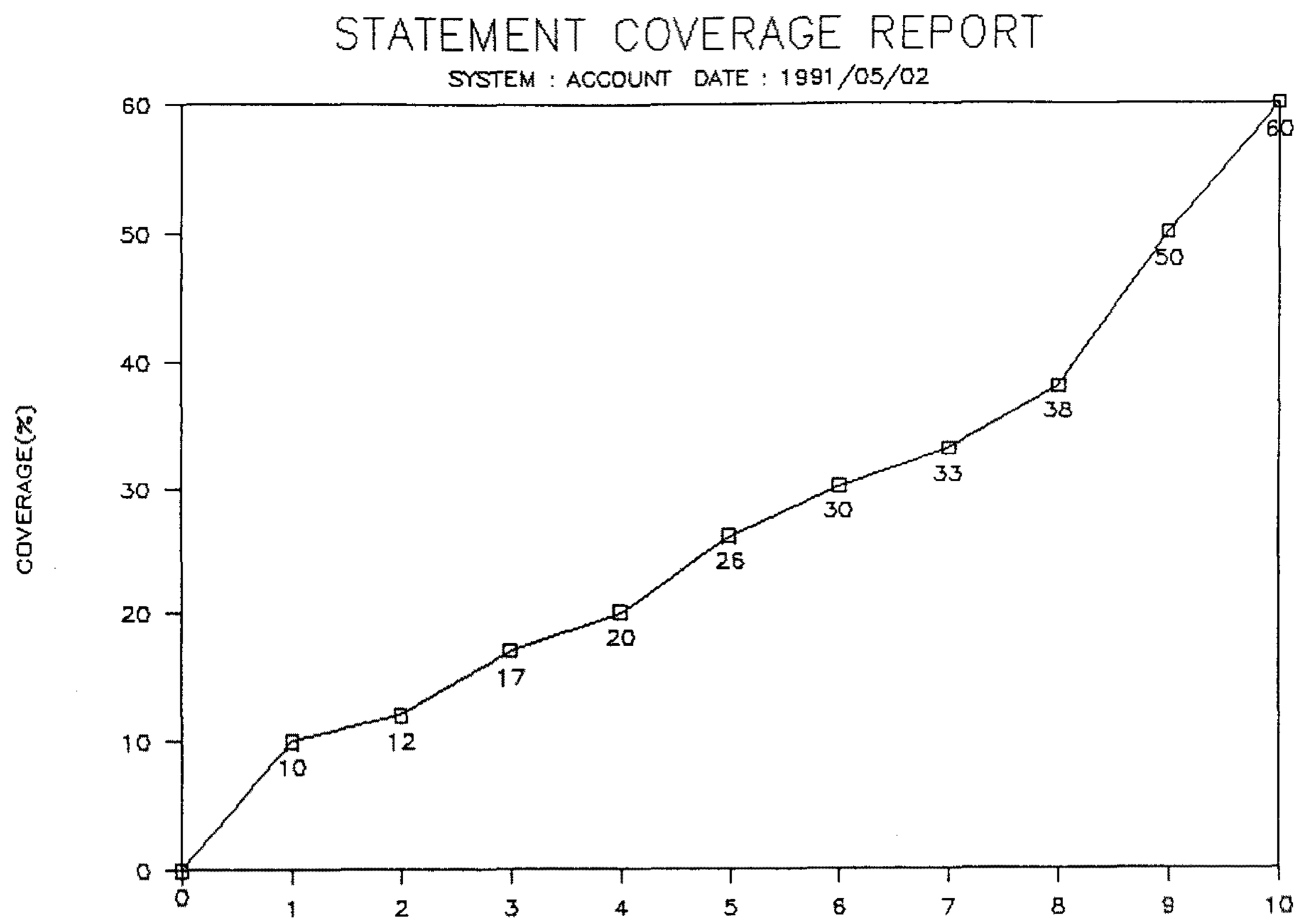
- 이 화면은 분기 범주를 도표로 나타낸 화면이다.
- 시스템에 대한 분기 범주 시험율을 주 단위로 보여준다.

  (시험된 분기 수 / 총 분기수의 백분율)



- 이 화면은 호출 범주를 도표로 나타낸 화면이다.
- 시스템에 대한 호출 범주 시험율을 주 단위로 보여준다.

  (시험된 호출 수 / 총 호출 수의 백분율)



- 이 화면은 문 범주를 도표로 나타낸 화면이다.
- 시스템에 대한 문범주 시험율을 주 단위로 보여준다.

  (시험된 문수/총 문수의 백분율)

제 6 장　도구 적용 사례

여 백

제6장 도구 적용 사례

제 1 절 대상 시스템 설명

시범 단계에서 개발된 품질 측정 및 문서 추출 시스템을 두 회사의 시스템 평가에 적용하였다. 대상 회사는 각각 제조회사(이하 갑회사)와 무역 관련회사 (이하 을회사)로서, 시스템의 대부분이 코볼로 개발 되었다.

도구의 주요 적용 목적은 시스템 품질 평가에 있었으며, 갑 회사의 경우 기술 시스템의 일부를 을 회사의 경우 20개 정도의 프로그램을 무작위로 선택하여 분석하였다.

[표 6-1]은 갑 회사의 시스템 품질 평가 결과를 [표 6-2]는 을 회사의 시스템 품질 평가 결과를 나타낸다.

[표 6-3], [표 6-4], [표 6-5]은 각각 갑 회사 시스템 분석 결과 중 프로그램 요약 정보, 프로그램 논리 구조, 변수 사용 이상 정보의 한 예이다.

[표 6-6], [표 6-7], [표 6-8], [표 6-9]는 각각 갑 회사 시스템 분석 결과 중 화일사용 정보, 화일 구조 및 참조 정보, 호출 정보 및 시스템내 호출 관계의 한 예이다.

제 2 절 적용 결과

도구의 적용 결과 기능적인 면에서 통상적인 10000 Line 이하의 프로그램 분석이 가능하였다. 문법 해석 면에서는 다음과 같은 보완 사항이 지적되었다.

- 동일 변수명 사용시 OF 혹은 IN의 사용
- Special Register 등 기종 의존적인 문법 확장

제공 정보에 대한 분석 결과는 아래와 같다.

- 품질 측정 결과가 시스템 개선 계획 수립등에 객관적 수치로 이용될 수 있었다.
- 각 품질 측정 항목별 기준이 제시되는 것이 바람직하다.
- 정적 오류 진단 기능의 확장이 필요하다.
- 구현 및 유지 보수 단계의 시스템 이해 및 문서 작성 노력이 대폭 감소되었다.

품질 항목별 기준 제시는 누적된 자료의 분석을 통하여 제시될 수 있도록 도구의 계속적 적용과 그 결과의 관리가 이루어 져야 한다고 생각된다.

[표 6-1] 갑 회사의 시스템 품질 측정 결과

<< 13. Program Metrics >>

System :

WValue / PGMW	# of File	# of RTN	LOC	Proc. LOC	# Comm	ELOC	LOC /label	V(G)	#goto	# if	%goto	%cntl	S/W nl	Science n2	N1	N2	Diff	Effort
bpeb3680	8	1625	16376	10723	948	7182	7(4)	2555	2532	1430	35	55	1409	16	12817	7182	72.8	15247551.0
bpeb3720	9	801	15105	11578	4269	5882	14(7)	1747	1421	1159	24	44	933	18	12040	5882	116.1	20592784.0
bpeb4040	4	92	2006	983	232	588	11(6)	196	86	85	15	29	223	16	1078	588	38.7	509042.0
bpeb4540	1	31	932	298	48	198	10(6)	63	51	40	26	46	59	13	409	198	45.0	168753.7
bpeb5390	2	24	986	401	187	196	17(8)	51	44	41	22	43	77	15	387	196	37.7	143362.3

< Legend >

LOC : Lines of code
ELOC : Executable Lines of code
V(G) : Cyclomatic Complexity
nl : Unique Operand
n2 : Unique Operator
N1 : Total Operand
N2 : Total Operator
Voc : Vocabulary
Len : Length
Diff : Difficulty
Effo : Effort

[표 6-2] 을 회사의 시스템 품질 측정 결과

<< 13. Program Metrics >>

System :

wValue PGMw	LOC	ELOC	V(G)	MIN	S/W Science n1	n2	N1	N2	S/W Science Voc	Len	Diff	Effo	# of File	# of RTN
Omsr01	902	341	136	0	184	18	785	341	202	1126	38.4	331100.7	2	27
amsr37	614	220	51	0	146	18	488	220	164	708	30.1	156702.6	2	19
amsr41	542	157	90	0	150	16	399	157	166	556	21.3	87259.1	2	15
amsr45	1481	983	290	0	211	19	2187	983	230	3170	98.5	2448890.0	2	80
amsr47	781	300	140	0	185	17	824	300	202	1124	37.8	325887.8	2	19
dotmak4	189	41	10	0	23	16	62	41	39	103	21.6	11740.0	3	13
eipint	186	32	7	0	30	12	62	32	42	94	12.4	6285.3	2	8
facprt	306	69	39	0	71	13	187	69	84	256	17.1	28015.3	2	5
fixmov	118	9	2	0	5	7	14	9	12	23	9.8	808.0	1	2
jungfile	153	25	5	0	17	14	39	25	31	64	16.0	5091.7	4	8
lyupdep	273	122	137	0	22	17	202	122	39	324	78.0	133650.5	2	9
mpay10u	521	230	159	0	65	15	401	230	80	631	46.3	184574.3	2	40
mpygiro	833	241	83	0	214	19	561	241	233	802	24.9	157072.8	6	18
sangcnt	226	57	96	0	15	15	134	57	30	191	67.0	62793.5	1	7
tbske0b	150	15	7	0	8	10	26	15	18	41	16.2	2778.2	2	4
tcdulcf	137	21	3	0	10	10	33	21	20	54	16.5	3850.8	2	4
tcyear5	246	81	23	0	30	15	126	81	45	207	31.5	35809.6	3	19
tfjubol3	212	50	11	0	32	11	72	50	43	122	12.4	8192.3	4	8
thhbpkr1	583	186	73	0	132	19	391	186	151	577	28.1	117529.0	6	18
tjsidpm2	697	202	116	0	149	15	498	202	164	700	25.1	129102.8	3	11
tjsidpr	782	275	151	0	158	18	597	275	176	872	34.0	221198.4	7	25
tjsktcs1	235	64	20	0	34	12	104	64	46	168	18.4	17030.8	3	8
Total	10167	3721	1649	0	1891	326	8192	3721	2217	11913	701.4	4475364.0	63	367

< Legend >

LOC : Lines of code
ELOC : Executable Lines of code
MIN : Maximal Intersection Number
V(G) : Cyclomatic Complexity
n1 : Unique Operand
n2 : Unique Operator
N1 : Total Operand
N2 : Total Operator
Voc : Vocabulary
Len : Length
Diff : Difficulty
Effo : Effort

[표 6-3] 프로그램 요약 정보의 예

<< 1. Program Summary and Statistics >>

Program Name : bpeb3680				
Author : RAINER MAHLINGER.				
Date-written : 15.10.1973				
Date-compiled :				
Total Line		Total	Comments	Executable
. Data Division : 5653 (copy : 0)		460		
. Proc Division : 10723		488		7182
16376 (16376)		948		
* Cyclomatic Complexity : 2555				
< Software Science >				
1. Unique Operand : 1409		2. Total Operand : 12817		
3. Unique operator : 16		4. Total Operator : 7182		
5. Vocabulary(1+3) : 1425		6. Length(2+4) : 19999		
7. Volume : 209524.4		8. Difficulty : 72.8		
9. Effort(7*8) : 15247551.0				
* No. of Paragraph : 1625				
* No. of Section : 0				
* No. of File : 8				

Command	Freq	Command	Freq	Command	Freq
< Arithmatic >	105	< Input/Output >	56	< Branching >	3923
ADD	103	ACCEPT Ident	0	ALTER	0
COMPUTE	2	CLOSE	8	CALL Literal	0
DIVIDE	0	DELETE	0	EXIT	246
INSPECT(Tally)	0	DISABLE,ENABLE	0	GO TO(Down,Up)	2206 326
MULTIPLY	0	DISPLAY	11	PERFORM(Down,Up)	455 690
SUBTRACT	0	OPEN	8		
		READ	1	< Ending >	1
< Data Movement>	2384	REWRITE	0	STOP RUN	0
ACCEPT	0	START	0	EXIT PROGRAM	0
INITIALIZE	0	STOP Literal	0	GO BACK	1
INSPECT(Replace)	2	WRITE	28		
MOVE	2382			< Linkage >	1
STRING,UNSTRING	0 0	< Ordering >	0	CALL	1
TRANSFORM	0	MERGE	0	CANCEL	0
		RELEASE	0	ENTRY	0
< Table Handling>	0	RETURN	0		
SEARCH	0	SORT	0	< Decision >	1430
SET	0			IF	1430
				EVALUATE	0

[표 6-4] 프로그램 논리 구조의 예

```
                                    << 7. Hierachical Chart >>
-------------------------------------------------------------------------------
    | Branch      Branch      |          Top Down Level Structure
NO. | at    to    Type        |
----|---------------------------|----1----2----3----4----5----6----7----8----9----0----1----2
 01 | 05654                   |PROCEDURE/DIVISION
 02 | 05655                   |    |    START1
 03 | 05695                   |    |    A001
    | 05696 05806 PERFORM     |    |    |    D001 THRU D001X
    | 05698 16323 CDTL GOTO   |    |    |    Z001
    | 05700 05932 CDTL PERFORM|    |    |    F001 THRU F001X
    | 05701 05695 CDTL GOTO   |    |    |    A001
    | 05702 06040 PERFORM     |    |    |    F100 THRU F100X
    | 05705 05695 CDTL GOTO   |    |    |    A001
 04 | 05715                   |    |    A002
 05 | 05718                   |    |    A003
 06 | 05720                   |    |    A004
    | 05721 05806 PERFORM     |    |    |    D001 THRU D001X
    | 05724 05745 CDTL GOTO   |    |    |    A005
    | 05726 05954 CDTL PERFORM|    |    |    F002A THRU F002X
    | 05727 06475 CDTL PERFORM|    |    |    F200 THRU F200X
    | 05733 05742 CDTL GOTO   |    |    |    A004A
    | 05736 14876 CDTL PERFORM|    |    |    G001 THRU G001X
    | 05737 16323 CDTL GOTO   |    |    |    Z001
    | 05740 14480 CDTL PERFORM|    |    |    F300-SERDE THRU F300-SERDEV
    | 05741 13228 CDTL PERFORM|    |    |    F300-SERDD THRU F300-SERDDX
 07 | 05742                   |    |    A004A
    | 05743 05869 PERFORM     |    |    |    E001A THRU E001U
    | 05744 16323 GOTO        |    |    |    Z001
 08 | 05745                   |    |    A005
    | 05746 06040 PERFORM     |    |    |    F100 THRU F100X
    | 05749 05720 CDTL GOTO   |    |    |    A004  ---> 6
    | 05750 05943 PERFORM     |    |    |    F002 THRU F002X
    | 05752 05718 CDTL GOTO   |    |    |    A003  ---> 5
    | 05755 06475 CDTL PERFORM|    |    |    F200 THRU F200X
    | 05757 05766 CDTL GOTO   |    |    |    A006
    | 05759 05715 CDTL GOTO   |    |    |    A002  ---> 4
    | 05763 05769 CDTL GOTO   |    |    |    A007
    | 05765 05774 CDTL GOTO   |    |    |    A008
 09 | 05766                   |    |    A006
    | 05767 05859 PERFORM     |    |    |    E001 THRU E001X
    | 05768 05715 GOTO        |    |    |    A002  ---> 4
 10 | 05769                   |    |    A007
    | 05772 05869 CDTL PERFORM|    |    |    E001A THRU E001X
    | 05773 05715 GOTO        |    |    |    A002  ---> 4
 11 | 05774                   |    |    A008
    | 05776 05782 CDTL GOTO   |    |    |    A009
    | 05778 05791 CDTL GOTO   |    |    |    A010
    | 05780 05715 CDTL GOTO   |    |    |    A002  ---> 4
    | 05781 05800 CDTL GOTO   |    |    |    A011
 12 | 05782                   |    |    A009
    | 05785 14480 CDTL PERFORM|    |    |    F300-SERDE THRU F300-SERDEX
    | 05786 05802 CDTL GOTO   |    |    |    A012
    | 05789 14480 CDTL PERFORM|    |    !    F300-SERDE THRU F300-SERDEX
```

[표 6-5] 변수 사용 이상 정보의 한 예

<< Variable Anomalies and Dead Code >>

Error	Variable/Paragraph	Line -- Line
Unused	S-FHL504	278
	ZI-07	339
	ZI-43	375
	ZI-87	419
	S-VHKF-4	644
	S-VHKF-5	645
	S-VHKF-6	646
	S-E20-VHKF-2	657
	S-E20-VHKF-3	658
	S-E20-VHKF-4	659
	S-E20-VHKF-5	660
	S-E20-VHKF-7	662
	S-E20-VHKF-8	663
	S-E20-VHKF-9	664
	S-FSGS-E53	690
	S-FSGS-E53-1	691
	S-FSGS-E53-2	692
	S-FSGS-E53-3	693
	S-FSGS-E53-4	694
	S-FSGS-E53-5	695
	S-FSGS-E53-6	696
	S-FSGS-E53-7	697
	S-FSGS-E54	700
	S-FSGS-E54-1	701
	S-FSGS-E54-2	702
	S-FSGS-E54-3	703
	S-FSGS-E54-4	704
	S-FSGS-E54-5	705
	S-FSGS-E54-6	706
	S-FSGS-E54-7	707
	KAL-E50-L-BLATT-KAL	5442
	KAL-E60-ZSTRSCHL	5496
	KAL-E60-ZSTRSCHL-NU	5497
	KAL-E60-ANZ-KSL-FOLGE	5503

<< Error Prone Area >>

Line #	Error	Command	Paragraph / Data Name	Mode	Len
11595	Type Inconsistency	MOVE	F359-RUECKS		
			SS-WW-E70	X	1
			S-VORHKA-E70	9	1
11597		MOVE	F359-RUECKS		
			SS-WW-E72	X	1
			S-VORHKA-E72	9	1

[표 6-6] 화일 사용 정보의 예

<< File Usage >>

[1 / 1]

File Field	P-F	C-F
Organization		
Access Mode		
Record Key		
Alternate Key		
Password		
Label Record	OMMITED	OMMITED
Block		
Length	132	80
Usage	P03-OW P13-WW P14-WW P16-WCR P17-WCR P19-CWR	P01-

File Usage :
P[XX] : Paragraph number XX

I : Input Open O : Output Open E : Extend Open C : Close
R : Read(Return) M : Rewrite W : Write(Release) D : Delete
S : Sort U : Using G : Giving

[표 6-7] 화일 구조 및 참조 정보의 예

[2 - 1]/ 2 Usage in Paragraph

File Name : C-R		Length : 80							
Seq No.	Variable	Remarks	Length	I V	1 12345	67890	2 12345	67890	3 12345	67890
1	01 C-R		G(80)	X D						
2	03 S-BUN		X(1)	X R						
3	03 FILLER		X(1)	X						
4	03 S-YER		9(2)	X R						
5	03 S-MON		9(2)	X R						
6	03 S-DAY		9(2)	X R						
7	03 FILLER		X(72)	X						

[1 - 1]/ 2 Usage in Paragraph

File Name : PREC		Length : 132							
Seq No.	Variable	Remarks	Length	I V	1 12345	67890	2 12345	67890	3 12345	67890
1	01 PREC		X(132)	X	A		AA	AA A		

[표 6-8] 호출 정보의 예

<< 6. Program Interface >>

Type	Program name / Routine	Paragraph / Parameters	Line #
CALL	DFHEI1	PROCEDURE DFHEIV0 DLZDIB	378
ENTRY	DLITCBL	PROCEDURE	379
CALL	HGBATCH	START-RTN FUNC1 RC1 H1 LE1 TH1 ...>> More Parameters	409

[표 6-9] 시스템내 호출 관계의 예

<< 12. Program Reference Matrix >>

System :

Main W sub	DFHEI1	DLITCBL	HGBATCH
Omsr01	0	0	0
amsr37	0	0	0
amsr41	0	0	0
amsr45	0	0	0
amsr47	0	0	0
facprt			0
mpay10u	0	0	
mpygiro	0	0	0
sangcnt			0
thhbpkr1	0	0	0
tjsidpm2			0
tjsidpr			0

제7장 결 론

여 백

제7장 결 론

제1절 연구 결과의 종합적 분석

• 1차년도 목표는 시험 절차 관리기 개발 및 시험 결과 처리기의 개발로, 현재 절차 관리기에는 시험 항목 관리, 시험 기능 설계 및 수행 관리, 시험 오류 작성 및 분석 등의 기능이 구현되어 있다. 결과 처리기 부분은 계기 삽입기와 범주 분석기가 구현 완료되어, 전체적으로 목표가 달성되었다고 본다.

• 현재 정적 품질 평가 시스템은 개발 완료되어 시범 적용 단계에 있으며 시험 정보 관리 시스템과 범주 분석기의 개발이 예정대로 완료되었으므로 2차년도의 설계 도구와의 연계 기능 정의에 의한, 3차년도의 통합 시스템 구현 목표는 무난히 달성되리라고 본다.

• 선진국의 질적 수준을 보면 범주 분석기와, 시험 구동기의 개발과 적용이 활발한 반면, 시험 정보 관리 시스템과 사례 설계 지원 시스템은 아직 만족할 수준이 못되고 있다. 시험 지원 도구들은 독립적으로 개발되어 도구간의 연계 기능이 부족하다는 점에서 시험 활동을 체계적으로 지원하는 통합도구의 필요성 대두되고 있다.

• 연구 결과의 질적 수준을 살펴보면 선진 기술국과 비교할 때 시험 범주 분석 기능과 시험 절차 관리 부분은 성숙 단계, 시험 구동기 개발 기술은 도입 단계라고 할 수 있다.

• 개발 기술의 전방 연결 체계로서 시험 및 품질 관리의 체계

정립을 위하여 1988년 부터 위탁 과제를 수행하였다.

1988 : 소프트웨어 품질 관리 체계 정립에 관한 연구

1989 : 소프트웨어 품질 평가 자동화 도구 개발

1990 : 소프트웨어 품질관리 환경 지원 시스템

• 연구 결과는

- 소프트웨어 시험 과정의 체계적 모델 설정 및 문서 표준 정립
- 시험 정보의 재이용 환경 구축
- 소프트웨어 개발 과정 및 품질의 가시성 부여
- 개발 조직의 유지보수 생산성 향상 및 시스템 품질 개선
  - 시험 정보의 재이용으로 유지보수 비용 감소
  - 시험 자동화 환경으로 시스템의 생산성과 품질 향상
  - 시험 정보 관리에 의한 시스템의 품질 개선
- 소프트웨어 개발 용역 업체의 경쟁력 향상
  - 시험의 완전성 추구로 인한 시스템 품질 향상
  - 객관적 품질 평가 기준 제시 등에 기여할 수 있다.

• 접근 방법을 볼 때

- 시험 지원 도구의 대상 언어는 국내에서 가장 많이 사용되고 있는 코볼로 정의하였고
- 시험 절차 및 문서의 표준은 IEEE 소프트웨어 공학 표준을 근거로 국내 개발 과정에 적합하게 정의하였으며
- 도구의 적용성을 고려하여 기존 상용화 도구들의 비교 분석을 통하여 시스템의 기능적인 모델을 정립하였고

- 소프트웨어는 다양한 환경에의 이식과 적용을 고려하여 UNIX 및 X-window를 이용하여 구현하였다.

제 2 절 연구의 한계점 및 향후 연구 방향

1. 연구의 한계점

- 동적 시험 지원 시스템은 운영 체제 및 언어에 종속적인 시스템으로 대형 기종 등 다양한 환경 지원용 시스템 개발이 프로젝트화 될 필요가 있다.
- 향후 동적인 방법과 정적인 방법을 통하여 포괄적인 품질 정보를 제공할 수 있도록 보완이 되어야 하며, 현재의 최종 산출물에 대한 품질 측정에서 중간 산출물과 개발 과정에 대한 품질 측정으로 그 범위가 확장되어야 한다.

2. 향후 연구 방향

- 시험 수행은 설계 명세와, 요구 분석 명세로부터 시험 기능을 추출하는 작업부터 시작된다. 현재의 기능 관리는 요구 사항의 기술과, 설계 내용의 기술이 규격화 되어 있지 않은 상황에서, 수작업으로 기능을 추출한 후 입력하는 것을 전제로 한다.

향후 요구 분석 명세나, 설계 명세서로부터 기능을 추출하는 작업을 지원할 수 있도록 요구분석 및 설계 도구와의 연계 기능이 확장되어야 한다.

- 장기적으로는

  규격화 된 명세로부터 시험할 기능과 그에 대한 사례가 자동적으로 추출될 수 있도록, 시험 가능한 명세서를 기술하는 방법과 품질 명세서 기술에 관한 연구가 진행되어야 한다.

- 코드 및 시험에 집중된 품질 평가 작업을, 설계 단계와 요구분석 단계로 계속적 확장이 필요하다.

참 고 문 헌

[오88] 오세만, 컴파일러 입문, 정익사 1988

[이 89] 이금석외, 소프트웨어 품질관리 체계 정립 및 지원도구 개발에 관한 연구, 과기처, 1989

[이 90] 이금석외, 소프트웨어 품질평가 자동화 도구 개발, 과기처, 1990

[시 89] 시스템 공학연구소, 소프트웨어 테스팅 도구 개발[Ⅰ], 과기처, 1989

[시 90] 시스템 공학연구소, 소프트웨어 테스팅 도구 개발[Ⅱ], 과기처, 1990

[최 87] 최광무 외, 컴파일러 개발에 관한 연구, 과기처, 1987

[형 90] 시스템 공학연구소, 국내 소프트웨어 형상 및 변경관리 실태조사, 과기처, 1990

[AHO86] A.V. Aho, R. Sethi, and J.D. Ullman, Compilers-Principles, Techniques and Tools, Addison Wesley, 1986.

[BOR84] Boris, Beizer, Software System Testing and Quality Assurance, Van Nostrand Reinhold Company, Inc., 1984.

[DEI90] Dieter Rombach, "Design Measurement: Some Lessons Learned", IEEE Software, 1990.

[DEM87] DeMillo, McCracken, Martin and Passafiume, Software Testing and Evaluation, The Benjamin/Cummings Publishing Company Inc., Inc., 1987.

[EDW81] Edward Miller, Software Testing & Validation Technique, IEEE Computer Society, 1981.

[FRA89] Frank Ackerman, "Software Inspections: An Effective Verification Process", IEEE Software, 1989.

[GLE79] Glenford. J. Myers, The art of software testing, A Wiley-

Interscience Publication, 1979.

[IEE87] Software Engineering Standards, IEEE, 1987.

[JOH90] John D. Musa and William W. Everett, "Software-Reliability Engineering: Technology for the 1990s", IEEE Software, 1990.

[LOR75] Lori A. Clarke, "A system to generate test data and symbolically execute program", IEEE Trans. on Software Engineering, March 1975.

[MAR88] Martin Shepperd, "An evaluation of software product metrics, Information and software technology", Vol 30, No. 3 April 1988.

[RIC90] Richard H. Cobb and H.D. Mills, "Engineering Software under Statistical Quality Control", IEEE Software, Nov. 1990, pp. 44-46.

[RIC87] R.W. Selby, V.R. Basili, and F.T. Baker "Cleanroom Software Development: An Empirical Evaluation", IEEE Trans. Software Eng. 1987, 1.027-1.307.

[SAN79] Sandra Rapps and Elaine J. Weyuker, "Selecting Software Test Data Using Data Flow Information", IEEE Trans. on Software Engineering, Jan. 1979.

[THO76] Thomas, McCabe, "A Complexity Measure", IEEE Trans. on Software Engineering, Apr. 1976.

[WIL87] William E. Perry, Effective Methods of EDP Quality Assurance, QED Information sciences, Inc., 1987.

[WIL84] William Hetzel, The Complete Guide to Software Testing, QED 1984.