규칙기반 한글 자동 프로그램을 위한 프로그램 변형기법

홍 성 수† 이 상 락†† 심 재 홍†††

요 약

초고급 언어에 의한 자동 프로그래밍은 프로그램의 자료구조 이외에 많은 부분을 시스템이 관장함으로써 프로그램 명세의 표현이 추상적이지만 프로그램 의미소가 술어논리, 집합, 사상, 혹은 제한된 자연언어를 사용 하기 때문에 초고급 구조에 익숙하지 않은 프로그래머들이 이를 이용하여 프로그램을 작성하는 경우 상당한 어려움이 따르고, 이들 초고급언어 구조에 익숙하기까지 많은 시간이 요하게 된다. 왜냐하면 초고급 언어는 프로그램 명세의 표현이 추상적이지만 프로그램 의미소가 술어 논리, 집합, 사상, 혹은 제한된 자연언어를 사용하기 때문이다. 본 논문에서는 기존의 자동 프로그램의 어려움을 줄이기 위해서 한글로 구성된 선언적 구문, 절차적 구문, aggregate구문으로 구성된 광역언어를 설계하고 구현한다. 본 논문에서 제안하는 한글 자동 프로그래밍 시스템(Hangul Automatic Programming)은 입력으로 순수한 한글로 구성되어 있으며 추상 알고리즘(Abstract Algorithm)과 자료형(Data Type) 혹은 절차적 구문을 받아서 출력으로는 C 언어 프로그램을 만들어 낸다. 자동 프로그래밍 접근 방식은 프로그램 변형기법과 규칙기반에 바탕을 두고 문제 영역은 일반적인 프로그램으로 한정 하였다. 시스템 제어구조는 한글 프로그램을 입력으로 받아서 지식베이스로부터 적절한 규칙을 선택해서 이것을 변형한 다음 전체 데이타 베이스에 넣는데 이 과정을 프로그램이 완성 될때까지 반복한다.

A Program Transformational Approach for Rule-Based Hangul Automatic Programming

Sung Soo HONG†, Sang Rak LEE†† and Jae Hong SIM†††

ABSTRACT

It is very difficult for a nonprofessinal programmer in Koera to write a program with very High Level Language such as, V,REFINE, GIST, and SETL, because the semantic primitives of these languages are based on predicate calculus, set, mapping, or testricted natural language. And it takes time to be familiar with these language. In this paper, we suggest a method to reduce such difficulties by programming with the declarative, procedural constructs, and aggregate constructs. And we design and implement an experimental knowledge-based automatic programming system, called HAPS(Hangul Automatic Program System). HAPS, whose input is specification such as Hangul abstract algorithm and datatype or Hangul procedural constructs, and whose output is C program. The method of operation is based on rule-based and program transformation technique, and the problem transformation technique. The problem area is general problem. The control structure of HAPS accepts the program specificaton, transforms this specificaton according to the proper rule in the rule-base, and stores the transformed program specification on the global data base. HAPS repeats these procedures until the target C program is fully constructed.

†종신회원 : 호서대학교 컴퓨터공학과 부교수
††정 회 원 : 인천대학교 전자계산학과 조교수
†††정 회 원 : 광운대학교 전자계산학과 교수
논문접수 : 1994년 2월 8일, 심사완료 : 1994년 4월 9일

1. 서 론

현대 사회는 정보화 사회로 컴퓨터가 정보의 매개체로서 사용되어지고 있으며 정보의 종합체로서

컴퓨터의 역량은 대단히 중요하여 중대 일로에 있다. 현대 사회가 정보화 사회로서 급변하는 정보에 부응하기 위해서는 이에 상응하는 신뢰성이 높고, 관리 유지의 가능성이 우수하며, 적시에 필요한 곳에 사용되어 크나큰 효과를 거둘 수 있는 이러한 소프트 웨어의 계발이 더 한층 요청되고 있다. 기계어에서 출발한 프로그래밍은 어셈블리어, 고급언어를 지나면서 점점 더 인간에게 사용하기 편리한 프로그래밍으로 발전 했으며 이러한 프로그래밍의 발전은 자동 프로그래밍(Automatic programming)[3,15] 이라는 프로그래밍의 자동화[2,12]를 추구하게 되었다. 자동 프로그래밍은 프로그램 언어, 소프트웨어 언어 공학, 및 인공지능의 한 응용분야[10]로서 현재 사람에 의해 수행되는 프로그램 과정의 일부를 컴퓨터가 수행하게 하는 것으로서 주어진 문제에 있어서 그 프로그래밍에 맞는 한정된 규칙기반이나 지식기반 [18,21]을 이용하여 그 문제를 해결 하는 것이다.

자동프로그램은 고급언어에 의한 프로그램이 아니라 자연언어나 제 4 세대 언어 수준인 초고급언어(Very High Level Language)인 명세를 입력으로 받아서 출력으로는 고급언어 수준의 목적 프로그램을 생산해 내며 생산된 프로그램은 기존의 컴파일러나 인터 프리터에 의해서 처리된다.[1,8] 초기의 자동 프로그램 시스템은 Stanford 대학에서 C. Green등이 지식기반을 이용한 지식기반 합성 시스템인 PSI[4] 시스템이 있는데 이것은 자동 프로그램의 여러기법을 종합적으로 사용한 방대한 자동프로그램 시스템이다. 그후 D.R.Barstow에 의해서 PSI 시스템를 확장시켜서 PECOS 라는 지식기반 자동 프로그래밍 시스템으로 확장시켰고 다시 C.Green은 PSI 시스템의 개념을 바탕으로 하여 지식기반 프로그래밍 환경인 CHI 시스템을 개발 하였는데 이 시스템은 초 고급 언어인 V 언어로 작성된 프로그램 받아서 지식 프로그램 기법과 알고리즘 설계법을 사용하여 프로그래머에게 보다 편리한 프로그램 환경을 제공한 자동 프로그래밍 시스템을 만들었다. 국내에서는 논리적 구문에 대한 규칙기반 프로그램 구성[20], 지식기반 시스템이 설계와 구축[19,21,22], 교육용 한글 파스칼, C[25,26,27] 등이 있다. 본 논문에서 제시하는 한글 자동프로그램 시스템(Hangul Automatic Programming System)은 입력으로 순수한 한글로 구성되어 있으며, 초 고급 언어에 해당되는 추상 알고리즘 (Abstract Algorithm)과 자료형 (Data Type) 혹은 절차적구문을 받아서 출력으로는 C언어 프로그램을 만들어내고 제어구조의 운용방법은 프로그램 변형기법과 규칙기반에 바탕을 두고 문제영역은 일반적인 프로그램으로 한정 하였다.

2. 자동 프로그래밍의 명세언어

자동 프로그래밍 명세에는 초고급어에 해당하는 CHI 의 V[4]언어 REFINE의 REFINE[7]언어, SETL[11]언어, 그리고 GIST[1,5]언어등 이밖에도 여러 가지 언어를 사용하여 프로그램 명세를 한다. 이들 언어를 살펴보면 V 언어는 PSI시스템에 바탕을 두어 Kestrel 연구소에서 개발한 CHI 시스템의 초고급언어로 프로그램을 입력으로 받아서 지식 프로그램 기법과 알고리즘 설계법을 써서 보다 편리한 프로그램 환경을 제공한 언어이다. V언어는 선언적 구문에 의한 프로그램에 주가 되고 있으므로 논리를 전문으로 하는 전문 프로그래머에게는 유용할지 모르지만 비전문 프로그래머나 일반 사용자가 사용하기에는 많은 어려움이 있으며, 특히 CHI 시스템의 자료구조합성기(Data Structure System Thesizer)에서는 프로그래머가 여러 개의 자료규칙 가운데 특정규칙을 선택하여 주어야 하는 등 프로그래머가 상당량의 프로그램 지식을 가지고 있어야 한다.[6,12] REFINE 언어도 CHI시스템을 바탕을 둔 언어로서 Reasoning Systems 사에서 최초의 상업적인 스시템으로서 지식기반 프로그래밍 환경을 명세할 수 있도록 개발 하였는데 REFINE언어도 V 언어에 바탕을 두었기 때문에 초고급 구조인 선언적 구문[7]에서 저급 구조인 절차적 구문까지 다양하게 프로그램 할 수 있는 광역 언어이며 여러 종류의 프로그래밍 기법에 바탕을 두고 있으므로 multi-paradigm 언어라고도 할 수 있다. REFINE 언어는 1차 술어 논리, 집합이론,

객체지향언어(Object Oriented language), 패턴 매칭, 규칙기반 프로그래밍, Constaints, 그리고 기존의 절차적 구조에 바탕을 두고 있으므로 생산성 면에서는 고급언어보다 많이 앞서지만 REFINE언어가 높은 생산성을 내기 위해서는 절차적 구문보다는 선언적 구조에 중점을 두어야 함으로 선언적 구조에 익숙 하지 못한 사람은 프로그램 작성시에 많은 어려움이 따른다. STEL언어는 집합이론에 바탕을 둔 언어로서 임의길이를 ORDERED Sequence로 구성된 tuple, 집합, 그리고 길이가 2인 tuple의 집합인 map 으로 프로그램 명세를 표현하고 자동적으로 자료구조 표현을 해주는 것을 목적으로 한다. [11] STEL 언어의 제어구조는 기존언어의 제어구조인 IF문, case문, while순환문등과 고급제어구조인 집합에 대한 순환과 전칭 한정사(univwrsal quantifier), 존재 한정사(existential quantifier)등이 있다. STEL언어를 사용하여 프로그램 하는 것은 프로그램 명세의 크기는 줄일 수 있으나 집합 이론에 중점을 둔 언어이므로 전문 프로그래머가 아니면 프로그램 작성이 힘들고 프로그램 크기가 커지면 프로그램의 이해도 또한 어렵다. GIST 언어는 SAFE 시스템의 결과인 초고급 형식 명세를 고급언어 프로그램으로 변형하는 작업으로 개발된 언어로서 프로그램 명세의 표현 방법을 개선함으로써 프로그램에게 원하는 명세를 쉽게 표현할 수 있도록 한 언어이다. 표현 방법의 개선에서는 historical reference, constaints, 데이타의 관계모델(relational model)과 associative model, 추론, demon 등을 포함 하고 있다.[1] 하지만 GIST 언어는 제한된 형의 자연언어를 이용하여 자연언어 명세의 표현력을 형식 구문과 의미를 통해서 나타냄으로써 자연언어에 의한 프로그램과 형식 구문에 의한 프로그램에 익숙하지 않은 프로그래머가 프로그램하기에는 어려움이 따른다. 본 논문에서는 이러한 명세 언어의 어려움을 감안하여 명세언어 뿐만 아니라 절차적 구문에 의한 프로그램 명세를 제안하며 순수한 한글을 사용함으로써 외국어에 익숙하지 않은 사람이나 컴퓨터에 관해 비 전문가들도 프로그램을 작성 할 수 있게 4 세대언어의 구조적 설계기법을 사용하였다

우리나라 컴퓨터 역사는 그리 길지 않지만 이미 2백 만대이상의 크고 작은 컴퓨터가 보급되었고 90년도 부터 통신 공사가 교육용 컴퓨터를 초. 중. 고에 무상으로 보급함으로써 컴퓨터 산업의 미래 지향적인 교육을 지도하며 96년도에는 각급 학교에 컴퓨터 교육을 할 수 있는 교육 환경을 제공 할 것이다. 이러한 정부의 컴퓨터 교육 활성화로 인해서 컴퓨터와 보다 친숙하게 컴퓨터를 대할 수 있는 길이 열리게 되었으나 컴퓨터 상에서 사용하는 거의 모든 언어가 주로 외국어로 되어 있기 때문에 컴퓨터를 처음으로 대하는 이들에게는 자국어가 아닌 외국어 이므로 컴퓨터를 사용하는데는 언어의 많은 장애가 초래 되어왔다. 본 논문에서는 순수한 한글로 구성된 한글 자동 프로그래밍을 설계하고 구현하여 한국 사람이 손쉽게 사용할 수 있는 한글 자동 프로그래밍을 제안하는데 그 목적이 있다.

3. 한글 자동프로그래밍의 설계

3.1 선언적 구문에의한 한글 자동 프로그래밍 설계

선언적 구문에 의한 한글 자동 프로그램 명세구성은 다음과 같이 <자료형>과 <알고리즘>으로 구성되어 있으며 <자료형>은 프로그램 명세에서 사용되는 자료형에 관한 내용이 기술되어 있고 <알고리즘>난은 선언적 구문, 절차적 구문, aggregate 구문으로 기술된다.

<자료형>은 <알고리즘> 에서 사용될 변수들의 자료를 정리한 것으로 모든 변수는 변수 선언부에서 선언 되어야만 사용 할 수 있는 strongly type[17] 형태를 취하고 있으며 (그림 1)의 형태를 취하고 있다.

```
        자 료 형
< 변수1 > :    < 선언1 >
< 변수2 > :    < 선언2 >
      .
      .
< 변수n > :    < 선언n >
```

(그림 1) 한글 자동 프로그램의 자료형
(Fig. 1) type of Hangul Automatic program

한글 자동 프로그램은 선언적 구문, aggreate 구문으로 구성되어 있으며 (그림 2)와 같이 주 프로그램과 부 프로그램으로 표현된다.

프로그램	프로그램 <형식인자>
{	{
<복합문>	<복합문>
}	}

(그림 2) <알고리즘>의 주 프로그램과 부프로그램 구성
(Fig. 2) Main-program and Sub-program

(그림 2)에서 형식인자는 매개변수 전달시 사용되며, <복합문>은 다시 <문장> 되고 <문장>은 선언적 구문, aggregate 구문으로 구성된다.

선언적 구문의 구성은 (그림 3)과 같이 BNF (Backus Naur Form) 방식을 택했으며, 논리연산자의 우선 순위는 알골과 파스칼의 구문 정의를 참조했다.

```
<선 언 문> ::= < 단순 선언문 > | < 함축 >
<단순선언문> ::= < 선 언 부 > | < 단순 선언문 > OR < 선언부 >
<선 언 부> ::= < 선 언 요 소 > | < 선 언 부 > AND < 선언요소 >
<선 언 요소> ::= < 술 부 > | { 프리 픽스 } [ < 선 언 문 > ] |
                                NOT < 선 언 요 소 >
<술 부> ::= < 상 태 > | < 술부 변수 > | < 술부지정자 >|
            < 한정사 변수 > | < 집합연산자 > < 집 합 변 수 >
<상 태> ::= < 연 산 >
<술어 변수> ::= < 변 수 >
<술부지정자> ::= < 술 부 변 수 > ( < 연 산 > [ < 연 산 > ] )
<한정사변수> ::= < 변 수 >
<집합연산자> ::= ∈
<집합 변수> ::= < 변 수 >
<프리 픽스> ::= | < 한 정 사 > < 한정사 변수 > | | < 한 정 사 >
                < 한정사 변수 > < 집합연산자 > < 집합변수 > |
<함 축> ::= < 선 언 문 > | < 상 태 >
<선 언 문> ::= <선 언 문 > | < 상 태 >
<한 정 사> ::= ∀ | ∃
```

(그림 3) 선언적 구문의 형식 정의
(fig. 3) BNF form of declarative constructs

aggregate 구문은 비 절차적 구문으로 관계형 데이타 베이스 시스템의 질의어의 aggregate 구조를 바탕으로 하고 있다. 최대값, 최소값, 합, 선택정렬, 이분검색, 힙정렬, 퀵정렬 등의 aggregate 구문은 각각 해당하는 절차적 구문의 프로그래밍의 지식을 지식 베이스내에 규칙을 저장해 두며, 사용자가 특정 aggregate 구문을 사용하여 프로그램을 작성하는 경우 한글 자동 프로그램 시스템은 이 구문과 지식 베이스를 비교해서 매칭되는 지식을 해당 aggregate 구문으로 변환 시킨다.

3.2 한글 자동 프로그래밍의 지식

한글 자동 프로그래밍에서 사용하는 지식은 선언적 구문을 위한 지식 aggregate 구문을 위한 지식, 절차적 구문을 위한 지식으로 분류 할 수 있으며 한글 자동 프로그래밍을 위한 이러한 규칙들은 술어논리의 추론에 이용되는 단순화 규칙과 명제논리의 추론에 이용되는 Wang 알고리즘의 변형 규칙을 사용한다.[13] 선언적 구문의 지식은 1차 술어 논리에 바탕을 둔 논리이며 전칭 한정사, 존재 한정사, 함축(Impliction),집합 등을 이용하여 프로그램 명세를 표현하며 프로그래밍에 필요한 전문 지식에 바탕을 두고 고수준의 프로그래밍을 저수준의 프로그래밍으로 변형하는 기법으로 의미규칙(semantic rule)을 사용하고 있다. 즉 의미규칙이란 처리순서를 지식에서 결정해줌으로써 필요한 규칙을 차례로 호출하여 선언적 구문을 절차적 구문으로 변형하는 것을 말한다. [알고리즘 3-1] 은 입력으로 받은 노드가 선언적 구문이라면 선언적 구문을 저수준의 절차적 구문으로 바꾸는 알고리즘이다.

입력 : 선언적 구문
출력 : 절차적 구문

```
CASE
  : ∃ X ∈ S[p] :
      enumerate X in S do
        begin
          H0 <- true;
          Apply[p];
          if NOT[C] then H0 <- false;
          if H0 then return(X);
        end
  : ∀ X ∈ S[p] :
      H0 <- true;
      enumerate X in S do
      begin
          Apply[p];
          if NOT [C] then H0 <- false;
      end
```

```
: p -> q:
    if p then Apply[p]:
: p ∨ q:
    if NOT p then Apply[q]:
: p ∧ q:
    begin
    Apply[p]:
    Apply[q]:
  end
endcase
```

[알고리즘 3-1] 선언적 구문을 절차적 구문으로 바꾸는 알고리즘

3-2 절차적 구문에 의한 한글 자동 프로 그래밍의 설계

절차적 구문에 의한 한글 프로그래밍 언어는 한국인이 보다 쉽게 대화 할 수 있게 하는 도구이므로 프로그래밍의 일반적인 성질을 고려하여 설계하여야 하며, 특히 누구나 이해하기 쉬운 키워드를 선택해야 한다. 또한 한글은 '조사' 라는 품사를 갖고 있으며 한글 문장 구조가 post-fix 라는 점을 감안 하여야 한다[15]. 지금까지 한글에 관한 많은 연구 들이 논의 되어 왔다. 이를테면 IMSAI용 베이직 언어를 대체하는 연구[23] 애플 기종에서 베이직언어를 한글 명령화하여 한글 도스 및 오류 메시지를 한글화하는 연구 교육용 한글 프로그래밍 언어 CELL의 설계에 관한 연구[24], 자연어 처리를 위한 신택스 구조에 관한 연구, 한국어 불규칙 활용 유형을 형태별로 구별한 연구, 교육용 한글 파스칼, C 설계 및 구현에 관한 연구[25,27], 영어-한글 번역 시스템[19], 기계 번역 시스템에 필수적으로 필요한 변환 사전기술에 관한 연구 등이 있다. 한글 자동 프로그래밍 언어는 프로그래밍 언어의 기본 요건을 만족하면서 한글의 구조적 특징을 반영하여, 읽기 쉽고, 사용하기 쉬우며 자연스러워야 한다는 점에 중점을 두었으며 또한 기존 컴파일러 기법을 사용하여 컴파일이 쉽게 될 수 있도록 하였다. 한글 자동 프로그래밍 언어의 문법 표현은 조사를 명사들과 같이 붙혀 사용하는 문법 규정언어 (Meta Language)를 원칙으로 하나 문법 규정 언어는 애매 모호 하기 쉽기 때문에 사용자가 쉽게 이해 할 수 있도록 일부 조사에 제한을 두었다.[17] 한글의 기본 문법 구성은 명령어가 뒤에 오는 Post-Fix라고 할 수 있는데, 한글을 Post-Fix 형태로 사용할 수 있으나, 이 경우 기존 파싱 방법을 사용할 경우 파싱이 쉽지 않고 사용하기가 어려움으로 될 수 있는 한 앞 부분에 키 워드를 주어 파싱과정에서 발생하는 문제점을 제거 하도록하며 한글 자동 프로그래밍에서 사용되는 일반적인 문장 구성에 관해서 기술하면 다음과 같다. 한글 문장의 기본구조는 <주부>+<서술부> 형태로 구성될 수 있다[14].그러나 한글 자동 프로그래밍 언어의 문장은 컴퓨터에게 명령하는 명령이므로 주부는 거의 대부분이 생략됨으로 이를 생략하며 서술부는 <관형어>+<목적어>+<부사어>+<동사>를 기본구조로 한다.[14,16] 서술부의 구성중 <목적어>는 <명칭>+<조사>의 구조로서 <목적어> 부분에 키워드가 나타나지 않으므로 기존의 파싱 방법을 사용할 경우 파싱 테이블 크기가 증가 할 뿐만 아니라 구성이 어렵고 복잡해지기 때문에 한글 자동 프로그래밍의 기본 문법 구조는 서술부의 구조를 변형해서 키워드가 맨 앞에 나오도록 하는 방법을 채택했다.[15] 그 방법중 하나로 컴퓨터가 행하는 행우를 일반적으로 실행[하다]라는 동사로 바꾸어 준다. 또한 기존 파싱방법을 그대로 사용하기 위해서 목적어의 <조사> 부분을 한글 자동 프로그래밍에 국한하여 '='에 (of) 에서 (to)으로 (with)까지 (until)까지 실행 (do)등으로 대체 함으로서 사용자가 사용하기에 자연스럽게 했다. 그러므로 이러한 문장구조는 <부사>+<목적어>+<부사어>+<동사> 형태가 되어 키워드가 문뒤에 나오게 됨으로 목적 하는 바를 이룰 수 있다. 복합문(conpound statement)는 문장의 앞부분에 <관형어>+<주어>의 형태로 한글 자동 프로그래밍 언어의 특이한 구조이나 한글 자동 프로그래밍 언어 복합문을 구성할 때 CFG 특징인 Self-embeded 한 특성을 이용하여 문장구조를 만들었으며, 복합문마다 시작과

닫음 키워드 '{}' 을 사용하여 문장의 범위를 알기 쉽게 했다.

3.2.1 선언부

선언부는 데이타 형태를 규정하는 부분으로 변수의 범위, 정보, 연산 형, 데이타, 자료구조 등을 기술하는 부분이다. 절차적 구문에 의한 한글 자동 프로그래밍에서는 변수선언과 그 외 관련된 자료형을 선언 할 수 있게 하였다. 이에 따른 한글 자동 프로그래밍 변수 선언부의 BNF 형식 정의는 (그림 4)와 같다.

```
<변수 선언부>::= <공란> | 변수 {<변수선언>}
<변수 선언>  ::= <명칭> [<명칭> [<초기선언>]]
<초기선언>   ::= <초기치> | <좌대괄호><초기치>{<초기치>}<우대괄호>
<초기치>     ::= <식> | <부호없는 변수><식> {<식>} |<변수>=<식>
<변수>       = <일반변수> | <첨자변수> | <레코드변수>
<일반변수>   = <변수명칭> | <첨자변수> | <레코드변수>
<변수명칭>   ::= <명칭>
<첨자명칭>   ::= <배열명칭>(<식>{<식>})
<배열명칭>   ::= <일반변수>
<레코드변수> ::= <레코드명칭><장명칭>
<장명칭>     ::= <오른쪽 끝> | <왼쪽 끝>
<오른쪽 끝>  ::= <식> | <공란>
<왼쪽 끝>    ::= <식> | <공란>
```

(그림 4) 선언문의 BNF 형식 정의
(Fig 4) BNF form of variable statement

3.2.2 연산문

선언된 자료에 따라 수식, 문장, 배정문이 프로그램 내에서 사용되는 수식은 변수, 상수, 함수의 산출 등으로 구성된다. 산술문을 구성할 때 수학식에서 일반적으로 사용하는 연산자를 사용 하였으며 문장과 문장은 세미콜론으로 구별하였다. 연산문에 대한 구문형식을 BNF 형태로 나타냈으며 표시하면 (그림 5)와 같다.

```
<문장>     ::= <산술문> |<제어문> | <복귀문> | <일반문> | <입출력문>
<산술문>   ::= <명칭><식>
<식>       ::= <단순식>  |  <식> <관계연산자><단순식>
<관계연산자>::=< | > | < = | > = | == | ! = |
<단순식>   ::= <인자> | <부호> <인자> | <단순식> <가감 연산자><인자>
<가감연산자>::=  + |
<인자>     ::= <요인>|<승제 연산자><요인>|<요인> <이전연산자><요인>
<승제연산자>::= * |  / |  ^  | && | :: | & | |
<요인>     ::= <원소> |  <원소>^<요인>
<원소>     ::= <변수> |  <부호없는 정수> | <식>  | <함수 수행문>
<함수 수행문> ::= <함수명칭> (<실인자열>)
<함수명칭>::=  <명칭>
<실인자열> ::= <공란> |  <실인자> {<실인자>}
<실인자>   ::= <식>
```

(그림 5) 연산문의 BNF 형식정의
(Fig. 5) BNF form of arithmetic statement

3.2.3 입출력문

선언적 구문에 의한 프로그래밍언어의 입출력 설계는 표준 입출력과 화일 입출력을 중심으로 하되 화일 입출력의 경우 프로그램에서 화일을 열고 (open) 각 레코드 형에 대한 버퍼를 할당하고 이것을 통해서 입출력이 행해진다. 다음은 입출력 선언부를 BNF구문 형식으로 표현한 것이며 (그림 6)과 같다.

```
<입출력 선택선언>
<입출력 선언부>::=<공란> | <양식선언> <입출력 버퍼선언>
<공란>::=
<양식선언>::=<공란> | 양식 { <양식형> }
<입출력 레코드형>::=<명칭> { <명칭> } <입출력형>
                  | 여백 <여백선언>
                  | <명칭> { <명칭> [레코드] <입출력 레코드형> }
                  | <명칭> { <입출력 선택선언> } [ { <입출력 레코
                    드형> } ]
<입출력 선택선언>::=<상수> { <상수> } [ { <입출력 레코드형> } ]
<입출력형>::=<부호없는 정수> <단순형> [<초기선언>]
          | <부호없는 실수> <단순형> [<초기선언>]
<단순형>::=실수 | 정수 | 논리수 | 음절 | 문자 | <단순형명칭>
```

(그림 6) 입출력문의 BNF 형식정의
(Fig. 6) BNF form of input and output statement

3.2.4 제어구조

제어문은 기본을 복합문으로 두어 CFG의 특징인 Self-embeded한 구조와 일치시켜 구조적 프로그래밍이 쉽게 구성되게 했으며, 제어흐름을 보다 쉽게 이해하기 위해 중괄호를 이용하여 불필요한 분기문을 줄이도록 하였다. 선택문의 경우 '아니면'이라는 키워드를 두었으며 '~'이면 혹은 '~면'의 한글의미는 같으나 번역상에 애매모호함을 피하기 위해 불필요한 조사는 제외하고 '~ 이면'으로 한정 시켰다. 제어문 구문에서는 루프를 중간에서 끝낼수 있게 중단(break)을 사용할 수 있으며, 계속(continue)사용하면 루프 중간에 있어도 다음 반복문을 시작 할 수 있다. 다음은 제어문을 나타낸 BNF 형태로 나타낸 것이며 (그림 7)와 같다.

```
<제어문>  ::= <조건문> | <선택문>  | <순환문>  | <참이면 반복문>
            | <반복문> | <분기문>  | <탈출문>
<조건문>  ::= 만약<식> [이면<문장열>] {<식>이면<문장열>}
<문장열>  ::= {<문장>}
<선택문>  ::= <변수>{<선택열>}
<선택열>  ::= <선택값>{<선택값>}<문장열>
<선택값>  ::= <숫자>{<숫자>}
<반복문>  ::= <반복>{<반복식><반복식>[<반복식>]}
<참이면 반복> ::= 반복<반복식>
<반복식>  :: <변수> <식> <식>
```

```
<분기문>      ::= <돌아감>
<순환문>      ::= 순환<순환문장열>
<순환문장열>   ::= {<순환문장>}
<순환문장>    ::= <문장> | 탈출문
```

(그림 7) 제어문의 BNF 형식정의
(Fig. 7) BNF form of control statement

(예제)

```
참이면 반복(낮은 <= 최대
{
  count++;
  중간=(낮은+최대)/2;
  선택(크기비교(배열[중간], 키))
    경우'<':아래=중간+1; 중단;
    경우'>':위 =중간-1; 중단;
    경우'=':찾는데 걸린 반복수=count; 돌아감 중단;
}
```

3.2.5 함수문, 순환문, 리스트

절차적 구문에는 주 프로그램, 부프로그램, 함수문(function)이 있으며, 함수문 자체가 하나의 문으로 구성될 수 있으므로 반환된 값을 무시할 수 있다. 절차적 구문에서는 기본 형과 포인터만이 인수로 될 수 있어서 부작용을 줄일 수 있게 하였다. 또한 인수들을 call-by-value로 전해진다. 즉, 그 값은 스택에 복사되며 파라메타값이 바뀌어도 처음에 호출한 함수 값은 변하지 않는다. 따라서 호출시에 변화를 원할 때는 그 파라메타에 해당하는 형의 포인터를 선언해 주어야 한다. 또한 함수문이 호출되어 그 수행이 끝나기 전에 함수문을 호출할 수 있는 순환(recursion)기법을 사용할 수 있다. 다음은 한글 주 프로그램과 부프로그램, 함수문을 BNF구문 형태로 표현한 것이며 (그림 8)과 같다.

```
<주프로그램>::= <부프로그램> | <함수문>
<부프로그램>::= <부프로그램 머리> <부프로그램 구성부>
<부프로그램 머리>::= <명칭> [(매개변수열)]
<함수문>::= <함수문 머리> <프로그램 구성부>
<함수문 머리>::= <명칭> ( [ <함수매개변수열> ] ) | <함수매개변수>
<매개변수열>::= <함수매개변수> {, <함수매개변수> }
<복귀문>::= <함수복귀문> | <부프로그램복귀문>
<함수복귀문>::= <식> 돌아감
<부프로그램복귀문>::= 돌아감
```

(그림 8) 함수문의 BNF 형식 정의
(Fig. 8) BNF form of subprogram

(예제)

```
정수 최대공약수(갑, 을)
      정수 갑, 을;
      {
         돌아감 을? 최대공약수(을, 갑%을):갑;
      }
```

변화가 많은 문제를 다룰 때는 배열을 사용하기 보다는 동적 자료구조를 사용하는 것이 효율적으로 기억장소를 경영할수 있다. 동적 자료구조에서는 배열처럼 고정된 자료와는 달리 필요할때마다 각 노드에 대한 기억장소를 배정할 수 있다. 절차적 구문에서는 새로운 노드를 위한 기억장소 할당기법을 사용할 수 있으며 새로운 노드의 기억장소를 할당시키기 위해서는 새로운 노드를 가르킬 수 있는 방법이 필요한데 절차적 구문에서는 이를 위해 포인터형이라는 특별한 형이 사용된다.

(예제)

```
참이면 반복(표준입력 "%d", 갑)
   {
     포인터=시작;
     시작=(구조노드 *)메모리 할당(구조노드);
     시작 -> 번호=갑;
     시작 -> 다음=포인터;
   }
```

3.2.6 배열과 구조체

배열과 동일한 자료형으로 취급되는 데이타들의 리스트이고, 배열의 각 원소는 하나의 변수로 처리할 수 있고, 일반적인 변수와 마찬가지로 선언할 수 있으며, 배열임을 나타내는 특별한 키워드를 갖지는 않는다.

3.2.7 오류처리

절차적 구문의 프로그래밍시에 오류처리는 한글 자동 프로그램 번역기 자체에서 표시하는 오류와 기존의 자동프로그램에서 나타나는 오류가 있다. 전자는 오류발생과 동시에 메세지를 화면에 출력하고 번역을 멈추거나 경우에 따라서는 오류메세지를 보낸체 번역을 계속 하도록 설계했다. 후자는 터보 C 에서 제공되는 오류메세지를 한글화 시켜서 출력시켜주며 (그림 9)와 같다.

```
한글 오류 메세지
"매개변수리스트가 틀림"
"(경우)문에 :이 빠졌습니다."
"{,}의 짝짓기가 잘못되었습니다.(복합문)."
"선언 문법이 잘못되었습니다."
"0으로 나눔"
"잘못된 부동 소수점 연산."
"포인터에 포인터가 아닌 데이타 치환"
"실수 상수 오류가 발생 했습니다."
"정의 되지 않은 식별자를 사용했습니다."
```

(그림 9) 한글 오류 메세지
(Fig. 9) Error Message of Hangul

3.2.8 기타

일반적으로 한글 프로그래밍언어는 명령어가 길어지기 쉽고, 한글 문자 입력의 번거로움이 발생되기 쉬우나 일반화되지 않은 약어나 다른 방법에 의한 한글 사용은 오히려 한글의 자연스러움을 저해할 수 있으므로 자연스럽게 번역하여 사용한다. 주요한 한글 C 명령어와 DOS명령어는 표준 함수표 〈표 1〉과 〈표 2〉, 〈표 3〉과 같다.

<표 1> 한글 명령어 일람표
<Table 1> Table of Hangul Instruction

한글 명령어	C	한글 명령어	C
중 단	break	만 약	if
경 우	case	돌 아 감	return
계 속	continue	구 조 체	struct
그 외 에	default	선 택	switch
실 행	do	형 선 언	type def
아 니 면	else	공 용 체	union
집 합	enum	형 없 는	void
반 복	for	참이면 반복	while

<표 2> 한글 함수 명령 일람표
<Table 2> Table of Hangul function

한글 명령어	C	한글 명령어	C
아크코사인()	acos()	소수나머지계산()	fmod()
아크싸인()	asin()	자연로그()	log()
아크탄젠트()	atan()	상용로그()	log10()
코사인()	cos()	싸인()	sin()
지수계산()	exp()	제곱근()	sqrt()
가장가까운값()	floor()	탄젠트()	tan()

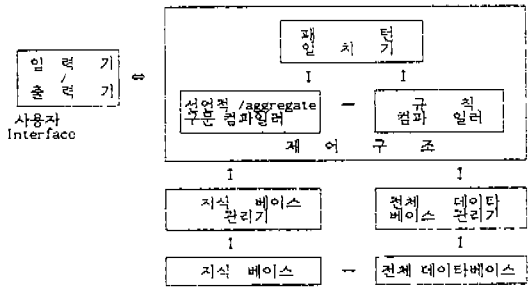
<표 3> 도스 명령 일람표
<Table 3> Table of Hangul DOS

한 글	C
디렉토리 작성	mkdir
시간 날짜 변환	unixtodos
시스템 시간 얻음	gettime
인터럽트	intr
복 사	copy

4. 한글 자동 프로그래밍 구현

4-1 선언적 구문에 의한 한글자동 프로그래밍의 구현

선언적 구문에 의한 한글 자동 프로그래밍 명세는 선언적 구문에 의한 프로그래밍 명세를 사용하고 목적 언어로는 C 언어를 쓴다. 자동 프로그래밍 접근 방법은 프로그래밍 변형기법과 지식공학 기법을 사용하고 시스템구조는 (그림 10)과 같다.



(그림 10) 명세를 입력으로 하는
자동 프로그래시스템의 전체 구성도
(Fig. 10) System strutures of HAPs
for Declarative statement

선언적 구문에 의한 시스템은 선언적 구문, aggreate 구문, 절차적 구문등으로 구성된 선언적 구문에 의한 한글 자동프로그래밍을 사용함으로써 초고급 구조인 선언적 구문, aggreate 구문에서부터 저급 구조인 절차적 구문에 이르기까지 문제의 성격과 프로그래머의 능력에 맞는 구문을 선택하여 활용할 수 있다. 선언적 구문에 의한 한글 자동 프로그래밍 시스템의 구성은 생성시스템 방법에 의하며, 제어구조는 선언적/aggreate 구문 컴파일러(Declarative/aggreate Statement Compiler), 규칙 컴파일러(Rule Compiler), 그리고 패턴 일치기(Pattern Matcher)로 구성하였다. 이 시스템에서의 각 기능들은 살펴보면 다음과 같다. 선언적/aggreate 구문 컴파일러는 선언적/aggreate 구문으로 작성된 명세를 받아서 등가의 절차적 구문으로 변형하며 자동프로그래밍 시스템에서 명세 변형의 pass 1에 해당한다. pass 1에 의한 프로그램 명세는 규칙 컴파일러에 넘어간다. 규칙 컴파일러는 지식 베이스에 저장된 규칙에 의해 입력으로 받아서 각 행의 내용에 따라 알맞는 지식 베이스를 택하여 프로그램 명세를 변형한 다음 전체 데이타 베이스에 변형된 프로그램을 저장한다.

패턴 일치기는 선언적 구문/aggreate 구문 컴파일러나 규칙 컴파일러가 입력으로 받은 프로그램 명

세를 각 행의 지식 베이스에 의한 규칙과 비교해서 2개의 pattern이 일치 하면 참을 넘기고 그렇지 않으면 거짓을 넘긴다.

사용자 Interface는 사용자 Interface인 입출력기(Reader/Writer)는 프로그램 명세를 읽어들이고, 목적 프로그램을 출력하거나 혹은 프로그래머와의 대화를 통해 필요한 정보를 프로그래머로 부터 얻는다.

지식 베이스는 프로그래밍에 관한 지식으로서 일반 프로그래밍 지식, 목적 프로그래밍 지식, 그리고 메타 지식 등이 모듈 별로 구성되며, 지식 베이스 내용은 선언적 구문에 관한 지식, aggreate구문에 관한 지식 그리고 절차적 구문에 관한 지식으로 구성되어 있다.

전체 데이타 베이스는 규칙 컴파일러가 주어진 프로그램 명세를 받아서 지식 베이스에 적절한 규칙을 택하여 변경한 다음 변형된 프로그램 명세를 저장하는 것이다. 이때 이 변형과정을 목적 프로그램이 완성될 때까지 반복하며 중간 결과를 반복적으로 전체 데이타 베이스에 저장한다.

전체 데이타 베이스 관리기 (Global data Base Manager)는 변형된 프로그램 명세를 전체 데이타 베이스의 내부 표현으로 바꿔 주는 일을 규칙 컴파일러와 전체 데이타 베이스 사이의 교신을 맡아서 한다.

4-2 자동 프로그램의 구현

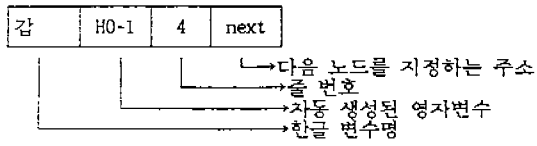
본 논문에서 제시하는 자동 프로그래밍 접근방법은 프로그래밍에 필요한 전문지식에 바탕을 두고 고 수준 프로그램에서 저 수준 프로그램으로 변형하는 기법을 사용하고 있다. 본 논문에서 제시하는 선언적 구문 방법은 문헌에서와 같이 의미 규칙의 처리 순서에 따라 메타지식을 결정함으로써 필요한 지식을 호출하는 방식을 택한다. [알고리즘 4-1]은 최대값을 구하는 한글 자동 프로그램을 선언적 구문을 사용하여 프로그램을 작성하였다.

[알고리즘4-1]에서 선언적 구문인 12행의 변형과정은 다음과 같다. 12행이 논리적 구조인 선언적 구문이라고 판별되면 존재한정사의 절차적 구조

```
1   프로그램
2   자료형
3   {
4   갑   : 배열
5   최대 : 정수
6   원소 : 정수
7   }
8   프로그램
9         {
10              {
11        입력 갑
12         ∃ 최대∈갑[∀원소∈갑[최대>= 원소]]
13          }
14             }
```

[알고리즘 4-1] 최대값을 구하는 한글 자동프로그램

알고리즘을 적용시켜같이 변형시킨다.이때 갑=H0-1, 최대=H0-2, 원소=H0-3로 한글에 대응되는 변수는 자동생성되며 자료구조는 (그림 11)와 같이 링크드 리스트로 되어 있다.



(그림 11) 사용자 변수 테이블
(Fig. 11) table of user variable

```
1  enumerate H0-3 in H0-1 do
2              {
3      H0-0 <- true
4       Apply [∀H0-3 ∈ H0-1[H0-2 >= H0-3]]
5         if NOT[C] then H0-0<-false
6         if max then return(H0-2)
7         }
```

(그림 12) 존재 한정사의 절차적 구조화
(Fig. 12) Procedural structuring for existential quantifier

여기서 4행의 Apply[]가 한정사 ∀를 포함하고 있으므로 Apply[]내의 술어식을 다시 변형한 다음 그술어식의 부울 변수를 5행의 조건 C에게 넘긴다. 따라서 (그림 12)sms (그림 13)과 같이 변형된다.

(그림 13)의 7행에서 Apply[]내 술어식을 더 이상의 한정사를 가지지 않으므로 이 술어식 자체를 9행의 조건 C에 대입시키다. 이 때 3행과 4행

의 H0-0는 같은 변수이므로 (그림 14)와 같이 간략하게 할 수 있다.

```
1  enumerate H0-3 in H0-1 do
2          {
3       H0-0 <- true
4       H0-0 <- true
5         enumerate H0-3 in H0-1 do
6            {
7              Applly [H0-2 >= H0-3]
8                 if NOT[C] then H0-0 <-false
9              }
10          if NOT[C] then H0-0 <- false
11      if H0-0 then return(H0-2)
12      }
```

(그림 13) 전칭 한정사의 절차적 구조화
(Fig. 13) Procedural structuring for universal quantifier

```
enumerate H0-3 in H0-1 do
          {
          H0-0 <- true
          enumerate H0-3 in H0-1 do
              {
                  Applly [H0-2 >= H0-3]
                  if NOT[C] then H0-0 <-false
              }
          if H0-0 then return(H0-2)
           }
```

(그림 14) 최대값을 구하는 한글 자동 프로그램의 변형결과
(Fig. 14) Modification of Hangul Automatic Programming for finding a maximum value

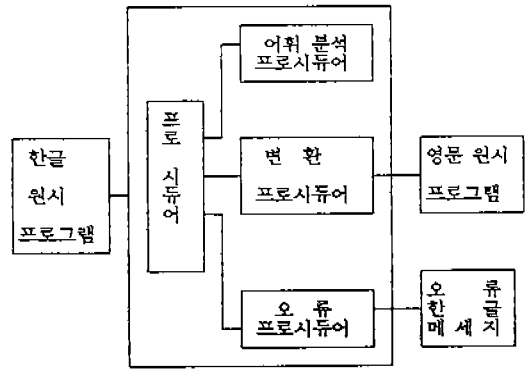
```
#define SIZE  50
main()
{
  int H0-1[SIZE], H0-2, H0-3, H0-4, H0-5;
  for( H0-4 = 0; H0-4 < SIZE; ++H0-4)
       scanf("%d", &H0-1[H0-4]);
  for( H0-4 = 0; H0-4 < SIZE; ++H0-4)
    {
     H0-3 = 1;
     for( H0-5 = 0; H0-5 < SIZE; ++H0-5)
     if(H0-1[H0-4] < H0-1[H0-5]) H0-3 = 0;
     if(H0-3) {
                printf("%d\n",H0-1[H0-4]);
                exit(0);
              }
    }
}
```

(그림 15) 최대값을 구하는 한글 프로그램을 목적 프로그램으로 합성한 결과
(Fig. 15) Result of Hangul Automatic Proramming for finding a maximum value

(그림 14)의 프로그램 명세를 받아 다음 단계에서는 최종적인 목적 프로그램을 (그림 15)과 같이 생산하게 된다. 여기서 SIZE와 자료는 사용자와 대화시스템에 의하여 결정된다.

4-2 절차적 구문에 의한 한글 자동 프로그래밍의 구현

절차적 구문에 의한 한글 자동 프로그래밍 언어는 한글로 프로그래밍 하는 것으로서 구현에 있어서 번역기법이나 인터 프리터기법을 고려할 수 있으나 여기서는 사전 번역기의(preprocessor)를 취했고 이것을 도시하면 (그림 16)과 같다.



(그림 16) 한글 프로그래밍의 사전 번역기
(Fig 16) Preprocessor of Hangul program

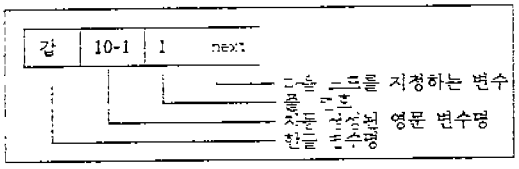
4.2.1 렉시칼 프로시듀어

렉시칼 프로시듀어는 번역기를 설계함에 있어서 매우 중요하다. 왜냐하면 이 프로시듀어 토큰이 잘못 나눠지면 주요 명령 및 단어들에 대한 올바른 번역 불가능하기 때문이다. 본 논문에서 제시하는 렉시칼 프로시듀어는 기존의 프로그래밍 언어에서 행해지는 방법과 같은 방법이나 한글 처리도 할 수 있게 되었다. 한글은 2바이트 완성형으로 채택했으며 렉시칼 프로시듀어 결과 분류되는 항목은 상수(CONSTANT), 명칭(IDENTIFICATION), 한글 지정어 (RESERED WORD), 연산자(OPERATION), 등이며 렉시칼 프로시듀어의 결과 각 단어들은 이름(NAME), 형(TYPE), 줄(LINE)을 순서대로 생성한다.

이때 <, ==, - 는 같은 의미로 사용되므로 바로 그자체를 출력시키며 만약 같음은 순서대로 변환 프로시듀어를 사용하여 해당 지정어나 변수 명령으로 바꿔준다.

4.2.2 CONVERTOR 프로시듀어

변환 프로시듀어는 분석할 프로시듀어 결과 한 단어가 생성되면 그 단어를 한글 예약어 〈표 1〉과 표준 함수 테이블 표 2 에서 탐색하고 그 테이블에 존재하면 일대일 대응하는 언어를 출력하고 그렇지 않으면 (사용자 언어일 경우) 그에 대응하는 변수를 자동으로 생성하게 된다 이때 국부 테이블과 지역 테이블을 만들어 처리한다. 국부 지역 테이블에는 변수, 상수, 레이블, 형, 함수, 프로시듀어들이 정의되고 변수의 자료구조는 LINKED-LIST로 (그림 17)과 같이 구성 되어져 있다.



(그림 17) 변수의 자료구조
(Fig. 17) Data structure of variable

지역 테이블은 블록레벨에 따라 그 갯수만큼 자동으로 생성되어 사용된후 곧 소멸된다. 그리고 각 레벨에 따라 입구(entry)가 다르게 설계되어 있다.

4.2.3 오류 프로시듀어

한글 프로그래밍언어에서 발생하는 오류의 처리는 오류메세지에서 발생하는 오류 메세지와 사전 번역기에서 나타나는 2가지의 오류메세지가 있다.

사전 번역기에서 나타나는 경우 오류메세지는 다음과 같다.

(1) 잘못된 토큰을 사용한 경우
(2) 실수 표기를 잘못한 경우
(3) 문자열 길이가 정의된 한계를 넘는 경우
(4) 삽입화일을 오픈하지 않은경우
(5) 정의 되지 않은 명칭을 사용한 경우

원시화일과 삽입화일을 찾지 못하는 경우와 같은 치명적인 오류는 즉시 번역을 멈추고 나머지경우는 나머지 문장들을 끝까지 번역해 오류가 있는가를 조사한다.

4.2.4 절차적 구문에 의한 한글자동프로그래밍

3장에서 설계된 원리에 따라 구현된 절차적 구문에 의한 한글 자동 프로그래밍 개략적 알고리즘은 [알고리즘 4-1]와 같다.

입력 : 절차적 구문에 의한 한글 프로그램
출력 : 영문 프로그램

1>. 한글로 구성된 프로그램이 절차적 구문이면 프로그램을 받아들여 확장자 HC를 붙이고 번역된 영문 화일 이름 확장자 C를 보관 한다.
2>. 입력된 문장을 어절(string)으로 받아 들인다.
3>. 한 단어가 될 때까지 토큰을 받아들인다.
4>. 그 입력된 문자열을 반복, 실행, 선택등과 같이 하나의 품사로 된 것인가 검색
5>. 만약 4번에서 실패한다면 그 문자열은 동사와 다른 단어 혹은 명사와 조사가 유합 된 것을 조사 한다.
6>. 단계 5에서 명사+조사인 것을 분리해서 명사와 조사로 만들고 동사+다른 단어 는 동사와 다른 단어로 분리 시킨다.
7>. 단어를 한글 예약어 테이블과 표준 함수, 및 한글 DOS에서 검색하며 만약 존재하면 해당 영문으로 출력한다.
8>. 이때 단어가 블록구조일 경우 그것의 순서에 해당하는 SCOPE 값을 결정한다.
9>. 만약 단어가 사용자 정의어일 경우 먼저 사용자 테이블에 그 단어를 추가 시킨다.
  9-1) 사용자 테이블은 각 프로시져 단위와 블록구조 단위로 설계되어 있으며, scope에 따라 입구가 틀리게 설계 되었다.

9-2) 사용자 정의 테이블의 자료구조는 접속 리스트 (linked list)로 설계되어 있으며, SCOPE 값에 따라 자동 변수 들은 사용후 곧 소멸된다.

9-3) 사용자 정의 테이블은 사용후 곧 삭제됨으로 사용자 테이블 사용자 변수를검색하는 순서는 발생된 순서의 역으로 검색 한다.

10>. 단계 1에서 9까지 모든 경우에 오류가 발생하면 오류 메시지를 발생시킨다. 이때 치명적인 오류(원시 화일이 없음, 삽입 화일이 없음) 등은 오류메시지 발생과동시에 멈추고, 그렇지 않으면 단계 1에서 9까지 반복해 수행한다.

논문에서 제안된 한글 사전 번역기는 한글로 구성된 절차적 구문의 변수, 상수, 예약어, 라이브러리, 도스 명령어등 영자, 영한글 혼용 뿐만 아니라 순수한 한글 프로그래밍이 가능하게 구현였다. 또한 프로그래밍시 발생할 수 있는 오류 메세지도 한글로 표시했다. 이에 따른 선택정렬 프로그램은 프로그램 4-1과 같고, 번역된 결과는 프로그램 4-2와 같다.

```
# 화일포함<stdio.h>
# 정의 자료수 10
 주프로그램
{
   정수 자료[10] = { 5,6,2,4 8,9,12,2,3,1};
   정수 갑, 을, 정;
   화면지움();
   제목쓰기();
   반복(갑=0; 갑 < 자료수 - 1 ; ++ 값) {
     을 = 값;
     반복(정=을 + 1; 정 < 자료수 ; ++ 정)
       만약 (자료[을] < 자료[정]) 을 = 정;
     값바꾸기( &자료[갑], &자료[을]);
    }
     반복(갑 = 0;갑 < 자료수 ; ++갑) {
     표준출력 ("d",자료[갑]);
      }
       표준출력("\n");
}
제목쓰기 ()
{
    표준출력("한글 C 선택정력 예제\n");
    표준출력("\n\n");
}
값바꾸기(정수 *값, 정수 *을)
{
  정수 임시;
  임시= *갑;
  *갑 = *을;
  *을 = 임시;
}
```

프로그램 4-1 절차적 구문에 의한 선택정렬

```
#include <stdio.h>
#define HC_0 10
#main()
{
    int HC_1[10] = {5,6,2,4,8,9,12,-2,3,1};
    int HC_2,HC_3,HC_4;
    clrscr();
    FUNC_0();
    for ( HC_2 = 0; HC_2 < HW_0-1; +
    +HC_2) {
       HC_3=HC_2;
       for(HC_4 = HC_3+1;HC_4 < HC_0
       =1 ; ++HC_2){
         if(HC_1[HC_32] < HC_1[HC_4])
         HC_3 = HC_4;
       FUNC_1( &HC_1[HC_2], &HC_1[HC
       _3]);
       }
    }
    for(HC_2 = HC_2 < HC_0; ++HC_2)
{
      printf("%d",HC_1[HC_2]);
      }
        printf("\n");
}
FUNC_0()
{
  printf("한글 C 선택정렬 예제\n");
```

```
    printf(" n n  :
}
FUNC_1( int *HC_2, int *HC_3)
{
                                    ..
    int HC_5;
    HC_5  = *HC_2;
    *HC_2 = *HC_3;
    *HC_3 = HC_5;
```

프로그램 4-3 절차적 구문에 의한 선택정렬 프로그램 변역결과

4-3. 한글 자동프로그래밍의 비교

현재까지 개발된 대다수의 자동 프로그래밍 시스템은 실험적이거나 연구용에 국한하였다. 반면에 튼튼한 단계를 거친 즉 초고급 언어의 구문으로 작성한 명세를 자동프로그래밍 할 수 있는 시스템은 REFINE 시스템뿐이고 이 시스템도 계속해서 확장하고 있다.[7] 따라서 본 논문에서 제안하는 시스템도 여러가지로 미진한 부분이 많으므로 앞으로 계속해서 개선해 나갈 생각이다. 본 논문에서 제안한 한글 자동프로그래밍과 국내외 이와 비슷한 시스템과 비교한 그는 표 4와 같다.

<표 4> HAPS와 그를 자동프로그램의 비교
<Table 4> Table of HAPS and other Automatic Program System

요점	프로그램 구문	입출 방법	특성
PSI	절차적 구문	자연어 인터	지식 기반 합성시스템
HAPS	절차적 구문	프로그램 변형기법	PSI 시스템을 확장한을
[illegible]	[illegible]	[illegible]	지식기반 프로그램환경
[illegible]	[illegible]	[illegible]	지식기반 프로그램환경
[illegible]	[illegible]	[illegible]	프로그램 변형 시스템
[illegible]	[illegible]	[illegible]	제한된 자연어 처리
[illegible]	[illegible]	[illegible]	알고리즘 기법
[illegible]	[illegible]		설계한 맵을(한글)
[illegible]	[illegible]	[illegible]	순수한 한글로 작성
[illegible]	절차적 구문 선언적 구문	[illegible] 프로그램 변형기법	순수한 한글로 작성

5. 결 론

본 논문은 한국 사람이 순수하게 한글을 사용하여 자동 프로그램 할 수 있도록 한글 명령어와 도스(DOS)명령어 그리고 오류메세지를 설계하고 구현 하였다. 한글 자동 프로그래밍 언어는 선언적 구문에 의한 방법, aggregate 구문에 의한 방법, 절차적 구문에 의한 방법으로 구성 되어 있다. 선언적 구문은 고 수준의 자료 구조인 집합, 함수, 형 등으로 구성 되어 있으며, 제어구조의 운용방법은 프로그램 변형기법과 지식공학기법중 규칙기반에 바탕을 두고 문제 영역은 일반적인 프로그램으로 한정하였다. 선언적 구문으로 작성된 한글 자동 프로그래밍 언어는 한글 지식 베이스에 의해서 저수준의 프로그램으로 변형된다. 절차적 구문은 프로그래밍 언어의 기본 요건을 만족하면서 한글의 구조적 특징을 반영하여 읽기 쉽고, 사용하기 쉽고, 자연스러움에 중점을 두어 설계하고 구현 하였다. aggregate 구문은 절차적 구문의 프로그래밍에 해당하는 프로그램 지식을 지식 베이스에 저장해 두었다가, 사용자 특정 aggregate 구문을 사용하여 한글 프로그램을 할 수 있게 했다. 앞으로의 연구 방향은 한국 사람이 좀 더 친숙할 수 있는 한글 표준 명령어를 설정해야 하고, 선언적 구문에 집합, 함수 등을 폭 넓게 응용하는 것이며 aggregate 구문을 폭 넓게 사용 할 수 있고 좀더 다양한 오류 처리를 해결하는 한글 자동 프로그래밍을 완성 하는 것이다.

참 고 문 헌

[1] Balzer, R. "A 15 Year Perspective on Automic Programming,"IEEE Trans. on Software Eng., Vol. SE-11, NO.11, PP. 1257-1267, Nov. 1985.

[2] Barr, A. and Feigenbaum, E.A. The Handbook of Artificial Intelligence, Vol. II, William Kaufmann Inc. CA. 1982.

[3] Biermann, A. W. et al., "An Overview of Automatic Program Construction Teachniques," in Automatic Program Construction Techniques (Ede.), Macmillan, NY, pp.3-30, 1984.

[4] Green, C. et al. "Research on Knowledge-Based Programming and Algorithm Design-1981," Tech. Rep. KES.U. 81.2, Kestrel Inst. Palo Alto, CA.Aug. 1981

[5] London, P. and Feather, M. "Implementing Specification Freedoms," in Readings in Artificial Intelligernce and Software Engineering (Eds.), Morgan Kaufmann, CA, pp. 285-305, 1982.

[6] Manna, Z. and Waldinger, R. "Synthesis : Dreadms => Programs," IEEE Trans. on Software Eng., Vol. SE-5, No.4, pp.294-328, July 1979.

[7] Markosian, L. et al., "Knowledge-Based Software Engineering Using REFINE," Reading Systems, 1988.

[8] Mattin, J. Fourth Generation Language, Vol.I, Prentice-Hall Inc., N.J. 1985.

[9] Rich and Richard C. Waters, (editors), Readings in Artificial Intelligence and Software Engineerings, Mongan Kaufmann Publishers, Inc 1986.

[10] Rich, E. Artifical Intelligence, McGraw-Hill, NY, 1983.

[11] Schonberg, E. et al., "An Automic Technique for Selection of Representations in SETL Programs,"ACM TOPLAS, Vol.3, No. 2, pp. 126-143, April 1981.

[12] Smith, D. R. et al., "Research on Knowledge-Based Software Environment at Kestrel Institute," IEEE Trans. on Software Eng. Vol. SE-11, No.11, pp. 1278-1295, Nov. 1985.

[13] Tanimoto, s. l. The Elements of Artificial Intelligence, Computer Science Press, MD, 1987.

[14] 김민수, "국어 문법론", 일조각, 1982

[15] 김영택, 김종상, 이석호, 조유근, 권혁철, "한글 프로그래밍 언어 설계에 관한연구", 한국 정보 과학회지 Vol. 11, No.2 pp81-101, May, 1984.

[16] 김영택, "프로그래밍 언어론", 홍릉출판사, 1982.

[17] 김영택, "컴파일러 구성론", 탑출판사, 1987.

[18] 김영택, 심광섭, "변환 사전 기술 언어", 한국정보과학회지 Vol.19, No.1, pp1-11 Jan. 1992.

[19] 김일곤, 유석인, "지식 베이스 구성 : 영역 분할과 혼합 추론을 이용한 통합 기법" 한국정보 과학회 논문지 Vol. 18, No.5, pp479-493, Sep. 1991.

[20] 박경환, 김영택, "논리적 구문에 대한 규칙기반 프로그램 구성" 한국 정보 과학회 논문지 Vol. 16, No.4, pp317-330, July, 1989.

[21] 박창현, 유석인, "지식기반 시스템에서의 추론 Browser의 설계" 한국 정보 과학 회지 Vol.19, No.1, pp80-92, Jan. 1992.

[22] 박창현, 유석인, "지식 기반 시스템의 효율적 구축을 위한 개발 환경", 한국 정보 과학 논문지 Vol.17, No1, pp40-51, Jan. 1990.

[23] 배명진, 안수길, "한글 명령 컴퓨터 설계", 마이크로 소프트 웨어, 1984.

[24] 원유헌, 이태욱, 김명렬, "컴퓨터 프로그래밍 언어 CELL의 설계에 관한 연구" 한국 정보 과학회 논문지 Vol.16, No.4, pp350-361, July. 1989.

[25] 김창희, 이상락, 홍성수, 심재홍, "교육용 한글 C 프로그래밍 언어 사전 처리기의 설계 및 구현" 한국 통신 학회 논문지 Vol.18. No.2, pp239-249, Feb. 1993.

[26] 주헌식, 홍성수, 심재홍, "선언적 구문에 의한 한글 자동 프로그래밍 시스템의 설계 및 구현" 한국 정보과학회 학술 논문 발표집 Vol.20. No.1, pp321-324, 1993.

[27] 홍성수, 김용성, 심재홍, "교육용 한글 파스칼 설계 및 구현" 한국 통신 학회 논문지 Vol 16, No.10, pp1009-1018, Sep. 1991.

홍 성 수

광운대학교 전산학과 졸업
쌍용 컴퓨터실 근무
1983년 광운대학교 전산학과 석사학위 취득
1990년 광운대학교 전산학과 박사학위 취득
현재 호서대학교 컴퓨터공학과 부교수 재직
관심분야 : 알고리즘, 자연어 처리, 병렬처리

이 상 락

1971년 서울대학교 사범대학 물리과 졸업
1983년 광운대학교 대학원 전산학과 졸업(공학석사)
1988년 광운대학교 대학원 박사과정 수료
1988~1994년 인천대학교 전자계산학과 조교수
관심분야 : 알고리즘, 그래픽스, 자연어 처리

심 재 홍

1967년 서울대학교 수학과 졸업
1980년 고려대학교 대학원(이학석사)
1981~1988년 경희대학교 대학원(이학박사)
1984~1986년 정보과학회 부회장 역임
1992년 광운대학교 총장
현재 광운대학교 교수로 재직중
관심분야 : 그래프알고리즘, 그래픽스, 수치해석