C++ 컴파일러 및 프로그래밍 환경 개발

장 천 현[†] · 오 세 만[††] · 유 재 우[†††]

요 약

본 논문에서는 가장 널리 사용되고 있는 객체지향 언어인 C++를 위한 컴파일러 및 대화식 프로그래밍 환경을 제안하고 개발하였다. C++ 언어를 위한 컴파일러를 개발하기 위해 컴파일러를 Front-End와 Back-End로 나누고 가상기계인 EM을 사용하여 연결하는 모델을 사용하였다. Front-End 개발 시에는 C++ 문법을 문법분석 도구, 어휘 및 구문분석기 생성도구를 이용하여 구문분석 방법과 문맥에 연동된 문법 처리기술과 AST 클래스 라이브러리를 개발하였다. Back-End 에서는 목적기계 표현기술과 트리 코드 최적화 방법, 트리 패턴 매칭에 의한 재목적 코드 생성 기법을 제안하고 이를 이용한 재목적이 용이한 SPARC 기계 Back-End를 개발하였다. C++를 위한 대화식 프로그래밍 환경은 언어의 다양한 특성을 효과적으로 표현하기 위하여 AST를 이용하고, 점진적 분석 기술과 시각 기호를 제안하였다. 대화식 환경의 일반화에 의한 자동생성 방법과 프로그램의 정형화된 표현방법을 위한 Unparsing 체계를 제안하였다. 개발된 C++ 컴파일러와 대화식 프로그래밍 환경은 통합된 C++ 프로그래밍 환경을 구성하게 된다. 본 연구를 통해 얻어진 기술들은 새로운 고급언어 및 기계에 대한 컴파일러의 개발은 물론 병렬 및 분산 환경을 위한 컴파일러 개발에 활용될 수 있을 것이다.

Development of C++ Compiler and Programming Environment

Chunhyon Chang[†] · Seman Oh[††] · Chae-Woo Yoo[†††]

ABSTRACT

In this paper, we proposed and developed a compiler and interactive programming environments for C++ which is mostly worth of notice among the object-oriented languages. To develope the compiler for C++ we took front-end/back-end model using EM virtual machine. In developing Front-End, we formalized C++ grammar with the context sensitive tokens which must be manipulated by lexical scanner and designed a AST class library which is the hierarchy of AST node class and well defined interface among them. In developing Back-End, we proposed model for three major components: code optimizer, code generator and run-time environments. We emphasized the retargetable back-end which can be systematically reconfigured to generate code for a variety of distinct target computers. We also developed tree pattern matching algorithm and implemented target code generator which produce SPARC code. We also proposed the theory and model for constructing interactive programming environments. To represent language features we adopt AST as internal representation and propose incremental analysis algorithm and visual diagrams. We also studied unparsing scheme, visual diagram, graphical

※본 연구는 '93년도 한국과학재단 연구비 지원에 의한 결과임
(과제번호:93-0100-01-01-3)

† 종신회원:건국대학교 컴퓨터공학과
†† 종신회원:동국대학교 컴퓨터공학과
††† 정 회 원:숭실대학교 컴퓨터학부
논문접수:1997년 1월 28일, 심사완료:1997년 3월 4일

user interface to generate interactive environments automatically. Results of our research will be very useful for developing a compiler and programming environments, and also can be used in compilers for parallel and distributed environments.

1. 서 론

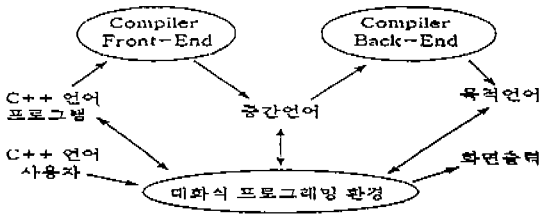
컴퓨터의 응용범위가 확대되고 다양해짐에 따라 컴퓨터 사용이 보편화되고 많은 작업들이 컴퓨터를 통하여 이루어지게 되면서 컴퓨터 사용자들은 편리하면서도 다양한 기능을 수행할 수 있는 소프트웨어를 요구하게 되었다. 이에 따라 개발되는 소프트웨어는 규모가 대형화되고 복잡성이 증가하게 되었다. 이러한 소프트웨어 위기 상황에서 신뢰성 있고 효율적으로 프로그래밍할 수 있으며 프로그램의 유지보수가 용이한 새로운 프로그래밍 언어의 설계 및 구현과 효과적으로 프로그래밍하기 위한 프로그래밍 환경 구축은 매우 중요한 문제로 등장하였다.

이와 같은 문제를 해결하기 위하여 시스템을 구성하고 있는 객체가 중심이 되는 객체 지향 방법(object-oriented paradigm)에 의한 개발방법이 제안되고 있으며, 이에 적합한 객체지향 프로그래밍 언어가 여러 가지 제안되었다. 이러한 객체지향 언어 중 현재 가장 주목을 받고 있는 프로그래밍 언어가 C++ 언어다. C++ 언어는 가장 많이 사용되어 오던 C 언어에 객체지향 개념을 추가하여 설계하였다. C++는 클래스 개념을 도입하여 자료의 추상화를 통한 객체의 표현을 지원하며 클래스간의 유도 클래스 정의를 통하여 객체의 상속성 및 가상 함수를 통한 다형성들을 제공함으로써 객체지향 프로그래밍 방법을 제공한다. 여러 가지 객체지향 언어 중 특히 C++ 언어가 가장 많이 쓰이고 이에 대한 연구가 활발히 되어 온 것은, 이 언어가 시스템 프로그래밍이나 일반 사무처리 및 계산 외에도 다양한 방면에서 현재 가장 널리 사용되고 있는 C 언어와 호환성을 갖고 있어서 기존의 소프트웨어를 그대로 사용할 수 있으면서 C 언어 사용자에게 친근감을 주며, C 언어가 갖고 있는 효율성을 거의 그대로 유지하면서 객체지향 프로그래밍을 지원하기 때문이다[1, 2]. 이러한 고급 프로그래밍 언어를 사용하기 위해서는 그 컴파일러와 프로그래밍 도구가 필요한데, 특히 컴퓨터 성능의 발달과 그래픽 사용자 인터페이스 기술의 급속한 신장으로 널리 개발되고 있으나, 국내의 경우 모두 외국에서 개발된 것을 그대로 사용하고 있는 실정이다. 이러한 새로운 패러다임의 프로그래밍언어의 컴파일러와 그 프로그래밍 환경에 대한 연구는 이 분야에서 지금도 가장 주요한 연구과제 중의 하나이며 효과적인 컴파일러와 프로그래밍환경 기술을 개발하는 것은 큰 의의가 있다고 하겠다.

C++ 언어를 위한 컴파일러와 프로그래밍 환경에 대한 연구는 외국의 경우 몇몇 연구소와 소프트웨어 업체를 중심으로 이루어지고 있으며, 최근에는 개인용 컴퓨터의 성능이 향상됨에 따라 윈도우를 기반으로 한 프로그래밍 환경이 소개되어 널리 사용되고 있다. Microsoft 사의 Visual C++[12]는 윈도우 프로그래밍을 위한 기본 도구로서 사용되고 있으며, Sun Micro Systems사에서는 프로그램 편집과 컴파일, 실행시 디버깅 도구를 합친 Sparc Works[13]를 제공하고 있다. 특히 Unix 시스템을 기반으로 한 시스템의 경우, 개발도구를 시스템에 기본적으로 설치하던 관례에서 벗어나 별도의 팩키지로 분리하여 판매하고 있는 추세이다. 현재 국내에서도 C++가 널리 사용되고 있으며 그 중요성 및 활용방안에 대한 연구가 점차 확산되고 있다[22]. 그러나 최근 소프트웨어 기술의 저작권 보호로 인하여 소프트웨어의 불법 복제 사용이 금지되고 있어서 새로운 프로그래밍 언어의 사용에 대한 비용이 급증하고 있어 그 확산이 어려워짐에 따라 선진국과의 소프트웨어 개발 기술 격차가 더욱 심화 될 전망이다. 따라서 C++ 컴파일러와 프로그래밍 환경과 같은 핵심 소프트웨어 개발의 필요성이 크게 대두되고 있다.

본 논문의 목적은 프로그래밍 언어를 중심으로 여러 가지 도구를 통합하기 위한 여러 가지 방법과 효과적인 사용자 인터페이스를 개발하여, C++ 언어를 중심으로 한 컴파일러와 대화식 프로그래밍 환경을 구축하는 것이다. 본 연구에서는 (그림 1)과 같이 컴파일러의 구현 모델 중 최근에 가장 보편적으로 이용되는 형식[4, 7, 12]으로 언어종속 부분인 front-end와 기계종속 부분인 back-end로 분리하고 이 두 부분을

중간 가상 기계의 중간 코드로 연결하는 방법을 따르고 있으며, 이와 같은 모델을 기본으로 하여 C++ 언어의 어휘분석, 구문 분석, 의미분석을 위한 front-end와 중간코드로부터 최적화 기술과 기계어 코드 생성을 위한 back-end를 개발하여 종합함으로써 C++ 언어의 컴파일러를 개발하였다. 또한 최근 연구경향으로 프로그래밍 언어의 구현은 컴파일러의 개발은 물론 사용자에게 편리한 종합적인 프로그래밍 환경을 제공하여야 하므로[14, 15] 사용이 용이한 그래픽 인터페이스 기반의 대화식 프로그래밍 환경을 제안한다.



(그림 1) C++ 프로그램 개발환경의 구성
(Fig. 1) Structure of C++ program development environment.

Front-end는 어휘분석기, 구문분석기 및 의미분석기 등으로 구성하고, back-end 부분과의 연결을 위한 중간언어로서 EM 코드를 생성하도록 하였다. 입력된 C++ 프로그램은 어휘분석기를 거쳐 C++ 정형문법[3]을 적용한 구문분석기에 입력된다. 구문분석 결과는 의미분석 단계에서 이용하기 위해서 적당한 내부표현이 필요하다. 원시 프로그램의 구문적 의미를 잘 반영할 수 있는 형태는 트리 구조인데 종래의 파스 트리는 불필요한 중복되는 정보를 많이 갖고 있기 때문에 이를 제거한 추상구문트리(abstract syntax tree; AST)를 본 연구에서 사용하였다. AST는 원시 프로그램의 의미를 잘 표현할 수 있도록 언어의 특성과 분석의 용이성, 처리의 효율성을 고려하여 설계하였다. 본 연구에서 정의된 AST 노드는 객체지향 개념을 도입한 클래스 계층으로 AST의 각 노드를 분류하여 설계하였다[23]. 이러한 구조의 장점으로는 클래스의 다형성에 의해 AST가 관리되므로 제어구조가 간단해지고, 노드의 종류에 따른 정보를 쉽게 저장하고 관리할 수 있으며 언어의 특성이 추가되더라도 새로운 노드를 클래스로 정의하면 되므로 변화에 쉽게 적응할 수 있으며 비슷한 계열의 언어를 위한 컴파일러의 구성 시에 클래스 계층의 대부분을 재 사용할 수 있는 장점이 있다. 의미분석기에서는 구문분석의 결과인 AST를 운행하면서 선언문에 의해 심벌테이블을 구성하고 그 정보를 이용하여 AST의 각 노드에 대한 처리를 수행하며 이 결과를 이용하여 EM 코드를 생성한다.

Back-end에서는 객체 지향 특성을 지원하는 C++ 언어를 위한 컴파일러를 구현하기 위해Front-End에서 생성한 EM 중간코드를 입력으로 받아 SPARC 기계에서 실행할 수 있는 목적 코드를 생성한다. 이를 위해 중간코드로부터 다양한 목적기계에 대한 목적코드를 생성할 수 있도록 재목적 가능한 컴파일러 Back-end를 설계하고 구현하였다. 재목적 가능한 컴파일러 Back-end는 중간코드 최적화, 목적코드 생성, 목적코드 최적화 과정으로 구성되어 있으며 생성된 목적코드의 효율적인 수행을 위한 실행 환경 및 실행시간 지원 라이브러리를 구현하였다.

대화식 프로그래밍 환경에서는 원시 프로그램을 어휘 분석기로 입력하여 토큰으로 분리한 프로그램을 구문 지향 편집기에서 보여주고, 편집 작업 후에 점진적 구문 분석기에서 구문 분석을 수행한다. 이 때 프로그램은 텍스트의 형태가 아닌 프로그램의 구조에 따른 트리의 형태로 보관된다. Front-End 개발에서 연구된 AST를 내부표현으로 이용하여 프로그램을 저장하고, 이를 화면으로 정형화하여 출력하기 위한 Unparsing 방법, 목적코드의 점진적 생성과 실행 방법, 구문트리의 대화식 변경과 속성의 변경에 따른 점진적 그래프 평가 방법, 프로그램의 자료 및 실행상태의 확인을 위한 인터페이스 기술을 이용한 시각화 방법을 제안한다.

2. Front-End 구현

Front-End는 크게 어휘 분석기, 구문 분석기의 설계와 구현, AST 클래스 라이브러리의 계층 설계, 의미 분석기의 설계와 구현, 그리고 back-end 부분과의 연결을 위한 EM 코드 생성 부분으로 나누어져 있다.

2.1 어휘 분석기 구현

C++에서의 토큰(token)은 식별자, 키워드, 리터럴

(literal), 특수 심볼의 4가지로 구분된다. 이중에서 키워드, 리터럴, 특수 심볼은 쉽게 처리할 수 있으나 식별자의 경우에는 추가적인 처리가 필요하다. 즉 식별자의 경우에는 문맥에 따라 타입 이름인지 혹은 일반적인 식별자인지 구분하는 것과 같은 문맥에 연관된 처리를 필요로 하게 된다[6].

문맥에 연관된 주요 처리 부분은 다음과 같다.

①단순한 어휘 토큰을 읽어 이를 처리하고 보관하는 토큰 버퍼 관리

②선언부(declaration)를 조사하기 위한 상태 관리

③선언부를 조사하면서 필요한 토큰의 삽입과 변경 관리

④파싱중의 스코프 관리

토큰 버퍼는 실제 토큰들이 저장되는 토큰 버퍼와 이 토큰 버퍼를 관리하는 함수들에 의해 관리된다. 이들 함수에는 토큰을 읽어 오는 함수, 토큰 버퍼에서 원하는 위치의 토큰을 읽어 오는 함수, 토큰 버퍼에 어떤 토큰을 삽입하는 함수, 토큰 버퍼에서 원하는 위치의 토큰을 새로운 토큰으로 바꾸어 주는 함수, 그리고 예견(lookahead) 처리를 위한 함수들이 있다. 실제로 예견의 처리는 식별자, 템플릿, 연산자, 외부, 클래스 등에 따르는 예견 함수를 통하여 이루어진다.

선언부 처리는 크게 선언부 상태를 관리하는 부분과 선언부 검색을 처리하는 부분으로 나뉘어진다. 선언부 상태를 관리하는 부분은 현재의 상태, 다음에 나올 수 있는 상태 같은 데이터와 이 상태를 조정하거나 현재의 선언부 상태에 따라 선언부 부분의 검색을 검사하고 시작하는 함수들로 구성되도록 설계하였다. 여기서 말하는 선언부 상태란 다음에 어떤 블록({...})이 나올 것인가 하는 것이다. 즉, 예를 들어 함수를 정의하는 부분에서는 다음 블록은 함수 몸체가 될 것이라는 것이다. 이것은 이 상태를 관리하는 클래스의 데이터 중에서 현재의 상태를 나타내는 변수의 타입을 살펴보면 쉽게 알 수 있다. 토큰 버퍼에서 예견 처리가 끝난 토큰들은 현재의 선언부 상태와 함께 선언부 검색을 시작할 지 검사하고 타당하면 선언부 검색을 수행하도록 설계하였다. 타당성 검사 결과는 적당한 위치의 토큰에서부터 선언부 검색을 시작하거나 아니면 다음 토큰을 읽고 타당하면 선언부 검색의 시작을 알리게 된다.

선언부 검색 처리 부분은 변화된 부분을 보관하기 위한 버퍼와 이 버퍼 관리를 위한 데이터들과 재귀적 하향분석(recursive descent parser)을 수행하는 함수들과 변화된 부분을 처리하는 함수들로 구성되도록 설계하였다. 이 부분은 선언부 부분을 재귀적 하향분석 하면서 필요한 가상 토큰(실제 원시 프로그램에는 해당하는 문자열이 없는 토큰)을 삽입하여 실제 파싱을 할 때, 비단말(nonterminal) 심볼을 구분하는 역할을 하도록 하고, 또 필요에 따라 현재의 토큰을 다른 토큰으로 바꾸어 준다. 이때 변화된 토큰은 당장 변하는 것이 아니라 클래스의 데이터 부분에 배열 형태로 보관되어 있다가 그렇게 변화되는 것이 옳을 경우에만 실제로 토큰 버퍼의 내용을 변화시키도록 한다.

2.2 AST 설계

구문 분석 결과를 의미 분석 단계에서 이용하기 위해서는 적당한 중간 표현이 필요하다. 원시 프로그램의 구문적 의미를 잘 반영할 수 있는 형태는 트리 구조인데 종래의 파스 트리는 불필요한 중복된 정보를 많이 갖고 있기 때문에 이를 제거한 AST(Abstract Syntax Tree)를 본 연구에서는 사용한다. AST는 원시 프로그램의 의미를 잘 표현할 수 있도록 언어의 특성과 분석의 용이성 및 효율적인 처리성 등을 고려하여 설계하였다. AST 설계시 정형 문법과 같이 형식론에 맞도록 기술하므로써 컴파일러 제작 도구를 통한 분석 단계 구현의 신뢰성을 보장하도록 한다[23].

본 연구에서 AST 노드는 객체지향 개념을 도입한 클래스 계층으로 AST의 각 노드(node)를 분류하여 설계하였다. 이렇게 함으로써 아래와 같은 잇점을 얻을 수 있었다.

①이전에 C로 쓰여진 프로그램에서는 AST의 각 노드의 형태는 노드의 종류와 자식에 대한 정보만 가지는 구조여서 처리시 노드의 종류를 구별한 후 수행할 일이 결정되므로 조건문이 많이 들어가게 된다. 본 연구에서는 각 노드를 클래스 계층으로 구성하여 객체지향의 특징인 다형성(Polymorphism)을 이용하여 노드의 종류를 구별하지 않고 일을 수행할 수 있게 하였다. 그럼으로써 AST에 대한 모든 작업이 같은 접속에 의해 이루어짐으로써 이해하기 쉬워지게 되었다.

②클래스로 노드를 표현함으로써 각 노드에 필요

한 정보들을 자식 노드를 만들지 않고 저장 할 수 있게 되었다.

③클래스의 이름은 생성규칙과 연관지어 정함으로써 문법으로부터 생성되는 AST 노드 클래스를 알기 쉽게 하였다.

④AST 노드의 추가, 삭제, 변경이 쉽다. 일반적인 언어를 사용했을 경우에 AST노드의 처리는 컴파일러전역에 걸쳐 일어나게 되며 노드의 추가, 삭제 또는 변경을 위해서는 일부분이 아닌 소스 대부분이 바뀌어야 한다. 하지만 본 연구의 AST의 구성은 클래스로 되어 있으므로 추가를 위해서는 새로 상속된 클래스를 생성하면 되고 삭제나 변경의 경우에도 소스의 변화가 일부 클래스만으로 국한되게 된다.

⑤AST 클래스 계층이 언어의 의미를 유지하기 때문에 새로운 언어로의 변화에 적응하기 쉽다. 본 연구에서 구현하고 있는 C++ 언어는 현재에도 ANSI 등의 기관에서 계속적으로 표준화를 하고 있으므로 언어의 의미가 언제 바뀔지 모른다. 일단 의미가 변한다면 AST의 구성도 변경되어야한다. 이 경우에도 객체지향 프로그래밍 기법을 이용한 AST 계층에서는 변화가 국소적으로 일어난다.

⑥비슷한 계열의 언어를 위한 컴파일러 구성시 클래스 계층의 대부분이 재사용 가능하다.

2.3 구문 분석기 구현

본 연구에서 새로 고안된 C++ 정형문법을 적용하여 구문분석기를 YACC을 이용하여 구현하였다. 본 구문 분석기에서 주로 하는 일은 다음에 수행되어야 할 AST를 생성하는 것과 어휘 분석 과정에서 필요한 정보(파싱시의 스코프에 대한 정보)를 관리하는 일이다.

AST 생성은 YACC을 이용하여 구문 지시적으로 처리하는데, AST의 구성을 위한 수행 코드들이 YACC 입력중 생성규칙 부분의 행위(Action) 부분에서 이루어진다. C++ 언어로 쓰여진 원시 프로그램을 어휘분석기에 의해 분석된 토큰을 이용해 정의된 문법에 맞게 기술되었는지를 검사하면서 의미분석시에 사용될 AST의 각 노드를 생성하고, 연결해 전체 AST를 구성해 나간다.

파싱시 환원(reduce) 되는 생성규칙을 제한하게 하기 위해서는 어휘 분석기가 예견을 통해서 토큰에 대한 세부적인 구분을 할 수 있도록 파싱시의 스코프에 대한 정보가 어휘 분석기에 주어져야 한다. 어휘분석기에서 예견 처리하기 위해 앞에서 소개된 파스 스코프(구문분석 과정중의 스코프를 관리)를 참조하는데, 이를 유지하기 위해 생성규칙의 행위 부분에서 AST 노드를 생성, 연결하는 일 외에 적절한 위치에서의 스코프의 생성과 파괴가 이루어지도록 해야한다. 이와 같은 정보에 의해 어휘 분석기에서 주어지는 세부적인 정보로 문자열에 대한 환원(reduce) 되는 생성규칙을 제한함으로써 매우 효율적인 구문 분석기를 얻을 수 있었다.

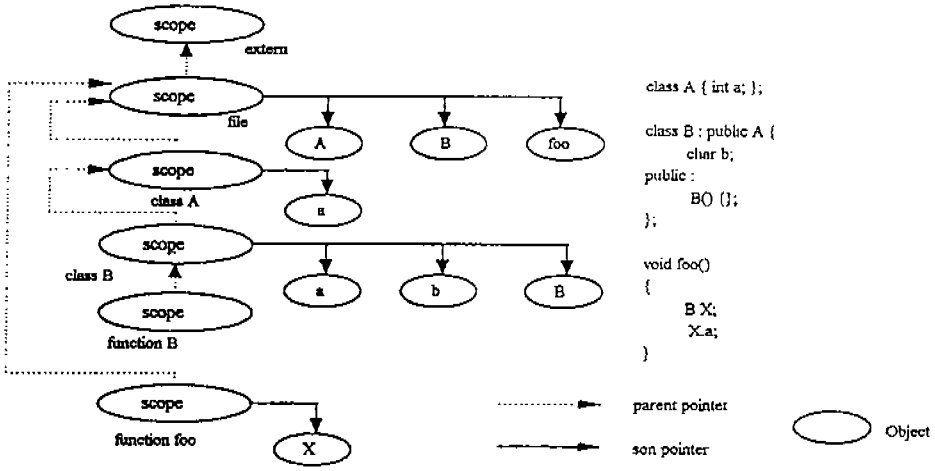
2.4 의미 분석기 구현

의미 분석기에서는 구문 분석기의 결과인 AST를 운행하면서 선언문에 의해 심볼테이블을 구성하고 그 정보를 이용하여 AST 각 노드에 대한 처리를 해준다. 이때, 중간 코드 생성을 위해 만들어지는 AST를 DAST(Decorated AST)라 하겠다[23]. AST의 루트 노드에서부터 각 노드의 클래스에 정의된 의미 처리 루틴을 호출하면서 깊이 우선 순서로 운행한다. 의미분석 과정을 통해서 AST의 구조가 변하면서 심볼테이블의 정보를 가지는 트리 구조가 생성된다. 여기에 필요한 노드의 종류에는 스코프를 표현하는 스코프 클래스, 스코프 내의 오브젝트를 표현하는 객체 클래스, 오브젝트의 타입을 표현하는 타입 클래스가 있다.

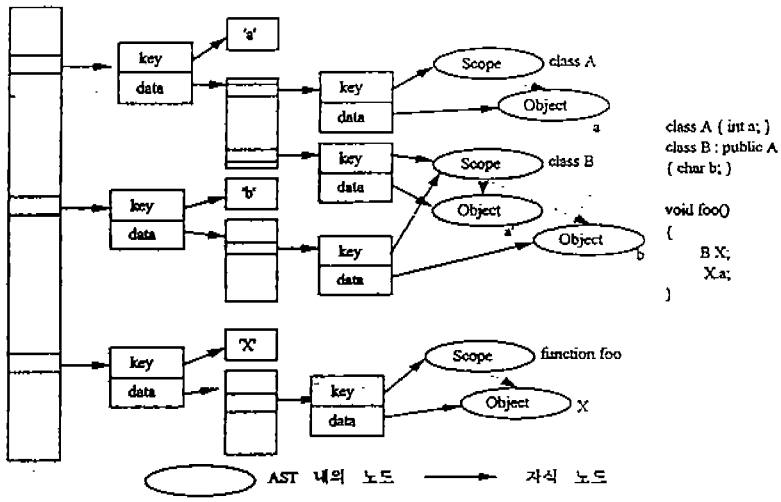
스코프 생성은 구문 분석기를 통해 얻어진 AST를 운행하면서 클래스의 선언, 함수의 선언, 복문 등을 만났을 때 일어난다. 스코프의 종류에는 함수, 클래스, 전역, 지역, 외부, 매개변수 스코프 등이 있다[5]. 각 스코프를 위해 유일한 스코프 노드를 만들어 줌으로써 심볼 테이블내의 네임 검색시 키(key)가 될 수 있게 한다. 변수, 함수, 클래스, 유니온, 열거, 타입 재정의 선언이 나타나면 오브젝트 노드를 생성해 해당 스코프의 자식 노드로 연결하여 준다(그림 2).

본 연구에서 구현된 심볼 테이블은 네임과 스코프 정보를 가지고 찾으려 하는 형태의 오브젝트를 찾을 수 있다. 효과적인 네임 검색을 위해 네임 테이블을 해쉬 테이블로 구성하였다. 네임 테이블의 생성과 네임 검색은 다음과 같이 이루어진다(그림 3).

우선, 선언문을 통해서 네임을 네임 테이블에 등록한다. 등록되는 이름에는 변수 이름, 함수 이름, 타입 재정의 이름, 내장 연산자 이름, 클래스 이름, 공용체



(그림 2) 의미 분석후의 DAST
(Fig. 2) DAST after semantic analysis



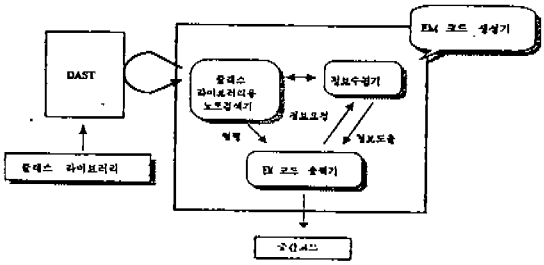
(그림 3) 해쉬 테이블 구조
(Fig. 3) Structure of hash table

이름, 열거 이름, 매개변수 이름 등이 있다. 'X.a;'에 대한 네임 검색은 우선 X를 네임 테이블에서 찾아 타입이 클래스 B라는 것을 알아낸 후, 클래스 B의 스코프를 현재 스코프값으로 준다. 네임 검색 시에 쓰이는 키중의 하나인 스코프 키가 현재 스코프이기 때문이다. 다음으로 클래스 'B'의 스코프와 네임 'a'를 키로 하여 오브젝트 'a'를 찾을 수 있다. 여기서 B의 스코프내에서 베이스 클래스의 멤버인 'a'를 찾을 수 있는 것은 상속시 파생 클래스에 베이스 클래스의 멤버를 가리키는 오브젝트를 가지기 때문이다.

2.5 EM 코드 생성

의미 처리가 완료된 DAST로부터 중간 코드(EM 코드)를 생성하기 위해서는 DAST를 운행하면서 많

은 정보를 검색하고 유지하면서 적절한 중간 코드를 생성하도록 해야 한다. 본 논문에서는 EM 코드 생성 과정을 (그림 4)에서 보듯이 DAST의 각 노드를 검색하면서 특정 노드의 특색에 맞게 처리를 해주는 노드 검색기, EM 코드 생성을 위해 지속적인 정보 및 일시적인 정보를 수집하고 필요에 따라 노드 검색기의 요청에 응답할 수 있는 정보 수집기, 그리고 노드 검색기의 명령에 의해 실행되며 정보 수집기의 정보를 받아 EM 코드를 출력하는 EM 코드 출력기로 구성하였다[24].



(그림 4) EM 코드 생성기 구조
(Fig. 4) Structure of EM code generator

노드검색기 구현은 보다 효율적인 검색을 전달하는 부분으로 DAST의 각 노드에서 다음 노드를 추적하는 과정을 삽입하여 시작점에서의 호출 과정만으로도 모든 검색이 지속적으로 이루어지게 되며, 검색 과정에서 노드로부터 취득한 정보는 정보수집기로 보내진다. 정보수집기는 노드검색기로부터 얻은 정보를 유지하며 출력기 요청에 응한다. EM 코드 출력기는 노드검색기의 명령에 의해서 기동되며 정보수집기에 정보요청을 하여 EM 코드를 출력하게 된다. 정보수집기는 위에서 정의되어진 것 외에도 일시적인 정보와 지속적인 정보를 담당하는 부분으로 분류되며, 이러한 정보들은 각각 정보에 대하여 정의되어진 함수에 의하여 값을 지정하거나 도출할 수 있도록 하였다. 노드검색기는 각 노드에 대하여 동일한 함수를 정의하여 노드에 상관없이 검색 함수를 호출하도록 함으로써 현재 노드에서의 검색과정에 대하여 일괄성을 유지할 수 있도록 하였다. 또한 특정 노드, 즉 for, if, switch와 같이 하나의 블록을 유지하는 구조로 이루어져 있는 부분에서는 정보 도출과 중간코드 생성을 용이하게 하기 위하여 검색함수 내부에 하위 검색을 처리하는 부분을 따로 정의하여 특성에 맞는 정보 도출을 행함으로써 일괄적으로 이루어지는 검색 과정에서 정보수집기가 가져야 되는 정보의 양을 최소화함으로써 정보질의와 도출과정을 최소화하였다.

3. Back-End 구현

Back-End를 구현하기 위해 컴파일러 제작 도구인 ACK를 분석하고 개선하여 재목적 가능한(Retargetable) Back-End 모델을 제시하고 그 모델로부터 코드 최적화기, 목적코드 생성기, 실행 환경으로 구성된 세 가지 컴포넌트를 구현하였다. 그리고 이와 같은 컴포넌트를 상호 연결하여 Front-End에서 생성한 EM 코드를 목적코드인 SPARC 코드로 변환하는 Back-End를 구축하였다.

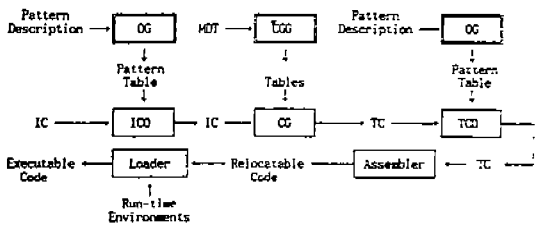
ACK에서 목적코드 생성은 코드-생성기 생성기를 통하여 코드 생성기를 생성하여 이용하는 방법과 코드 확장기를 이용하는 방법으로 구분된다. 코드 확장기를 이용한 방식에서는 각 EM 중간 코드를 해당되는 목적기계 코드로 마크로 확장(macro expansion)하는 기법을 이용하고 있다. 특히, ACK에서는 SPARC 목적기계 코드 생성 시에 코드 확장기 방식을 사용하고 있다.

3.1 재목적 가능한 Back-End

재목적 가능한 Back-End는 중간코드 최적화, 목적코드 생성, 목적기계 코드 최적화 과정으로 구성되며 다양한 목적기계에 대해 원시 프로그램의 중간코드를 목적기계 코드로 바꾸는 과정을 기계 정형화를 통하여 자동적으로 구성하는 것이다. 따라서 코드 생성 과정을 목적기계 의존적인 부분과 목적기계 독립적인 부분으로 나누어 정형화하는데 목적을 두고 있다. 목적기계 의존적인 부분에서는 목적기계 표현 테이블을 입력으로 받아 목적기계에 대한 정보 및 코드 생성 규칙을 저장하고 있는 테이블을 출력해 주며 목적기계에 대해 독립적인 부분에서는 원시 언어의 중간코드 입력과 테이블을 참조하여 하나의 공통된 코드 생성 알고리즘을 가지고 목적기계 코드를 생성하게 된다[11, 12].

본 연구에서 제안한 재목적 가능한 Back-End는

(그림 5)와 같이 중간 코드 최적화기(Intermediate Code Optimizer;ICO), 코드 생성기(Code Generator;CG), 목적기계 코드 최적화기(Target Code Optimizer;TCO)로 구성되어 있다.



(그림 5) 재목적 가능한 Back-End 모델
(Fig. 5) Retargetable Back-End model

중간코드 최적화기(ICO)는 EM 중간코드에 대해 트리 패턴 매칭 기법을 이용한 핍홀 최적화 동작을 수행하여 최적화된 EM 코드를 생성한다. 트리 패턴 매칭 기법을 이용한 핍홀 최적화 동작의 수행을 위해 패턴 표현 부분에서는 중간코드에서 나타날 수 있는 패턴과 최적화 동작 시에 대치되는 패턴이 트리 구조로 재구성된다. 또한 순차적인 구조를 갖는 EM 중간코드도 스택 운영 정보를 활용하여 트리 구조로 재구성되며 트리 구조를 가진 패턴 정보를 참조하여 최적화된 EM 코드를 생성한다.

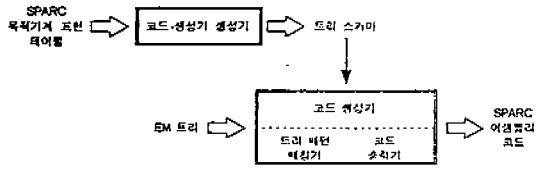
목적기계 코드 생성기는 목적기계 표현 테이블(Machine Description Table;MDT), 코드-생성기 생성기(Code Generator-Generator;CGG), 코드 생성기(Code Generator;CG)로 구성되어 있다. 코드-생성기 생성기는 목적기계 테이블을 읽어 목적기계의 정보 및 코드 생성 규칙을 트리 구조로 변환된 트리 스키마(Tree Schema)를 출력한다. 코드 생성기는 트리 스키마와 트리 패턴 매칭 코드 생성 알고리즘을 이용하여 중간코드에 해당하는 목적기계 코드를 생성한다. 중간코드에 대한 목적코드 출력시에 트리 스키마와 일치하는 패턴을 찾기 위해 트리 패턴 매칭 알고리즘을 적용한다[25].

목적기계 코드 최적화기는 중간코드 최적화기와 달리 목적기계 특성에 의존적이므로 목적기계 코드의 패턴 표현을 입력으로 받아 최적화기 생성기에 의해 패턴 테이블을 구성한다. 목적기계 코드 최적화기는 생성된 패턴 테이블과 패턴 매칭 알고리즘을 적용하여 최적화된 목적기계 코드를 생성한다.

본 논문에서는 이러한 재목적 가능한 Back-End 모델을 기반으로 하여 중간코드 최적화기, 목적코드 생성기, 목적코드 최적화기 및 생성된 목적코드의 수행을 위한 실행 환경을 구현했다. 특히 Back-End의 핵심 부분인 목적코드 생성기의 설계 및 구현 부분을 강조한다.

3.2 트리 패턴 매칭 코드 생성기

트리 패턴 매칭 코드 생성기(Tree Pattern Matching Code Generator)는 최적화된 EM 중간코드를 입력으로 받아 SPARC 코드를 생성하는 부분으로서 (그림 6)와 같이 코드-생성기 생성기와 코드 생성기 부분으로 구성된다. 코드-생성기 생성기는 SPARC 목적기계 표현 테이블을 입력으로 받아 목적코드 생성을 위한 정보를 트리 패턴 매칭에 적합한 구조인 트리 스키마(Tree Schema)를 생성한다. 코드 생성기는 EM 트리에 대해 트리 스키마 정보를 참조하여 SPARC 기계에 대한 어셈블리 코드를 생성하는 부분으로서 트리 패턴 매칭을 수행하는 트리 패턴 매칭기(Tree Pattern Matcher)와 실질적으로 SPARC 어셈블리 코드를 생성하는 코드 출력기(Code Emitter)로 구성된다[25].



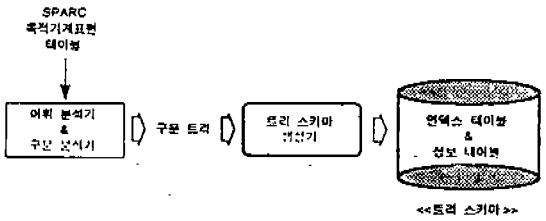
(그림 6) 트리 패턴 매칭 코드 생성 시스템
(Fig. 6) Tree pattern matching code generating system

SPARC 목적기계 표현 테이블에는 SPARC 기계의 특성 및 EM 중간코드에 대한 코드 생성 규칙이 정형화된 방법으로 기술한다. 실질적인 코드 생성 규칙은 지정어 PATTERN으로 시작되는 코드 규칙 절에 기술되는데 pat, with, kills, leaving, uses, yields, gen과 같은 지정어를 이용해서 EM 패턴, 스택 패턴, 레지스터 할당 등의 코드 생성 규칙을 다음과 같이 기술한다.

```
pat loc
    yields {Constants, $1}.
```

```
pat adi $1 == 4
    with L_Reg, RegCon13, Register
    gen add %1, %2, %3
    yields %3
```

코드-생성기 생성기는 SPARC 목적기계 표현 테이블을 입력으로 받아 코드 생성시에 참조되는 트리 스키마를 생성하는 부분으로서 (그림 7)과 같이 목적기계 표현 테이블에 대해 구문 분석을 수행하여 구문 트리를 구성한 후 구문 트리를 순회하면서 EM 명령어 패턴 및 수식 부분을 트리 패턴 매칭에 적합한 트리 스키마(인덱스 테이블, 정보 테이블)를 생성한다. 목적기계 표현 테이블로 부터 기계 정보 및 패턴 정보의 생성 과정은 ACK 코드-생성기 생성기의 구조를 2단계로 구성하였다[26].



(그림 7) 코드-생성기 생성기 구조
(Fig. 7) Structure of code-generator generator

코드-생성기 생성기에 생성된 정보는 코드 생성기가 효과적으로 이용할 수 있도록 인덱스 테이블(Index table)과 정보 테이블(Information table)로 구성된 트리 스키마에 저장된다. 인덱스 테이블은 정보 테이블에서 트리 패턴의 검색 시간을 줄이기 위한 것으로 트리 패턴 구성시에 루트 노드로 존재할 수 있는 명령어 및 실제로 트리 패턴이 저장되어 있는 정보 테이블의 인덱스에 대한 정보를 저장하고 있다. 또한 루트 노드를 중심으로 구성되는 트리 패턴의 노드 수를 제공하므로서 좀더 효과적인 트리 패턴 검색이 이루어지도록 하고 있다. 정보 테이블은 실제로 SPARC 목적기계 표현 테이블에 나타나는 EM 명령어 패턴에 대해 트리 패턴 및 트리 패턴에 나타날 수 있는 수식에 대해서도 트리 구조로 변환하여 저장한다.

코드 생성기는 실질적으로 SPARC 코드를 생성하는 부분으로서 순차적인 EM 중간 코드를 EM 명령어의 스택 운용 정보를 참조하여 구성된 EM 트리를 입력으로 받는다. EM 트리 구성시에 의사 명령어, 레이블은 피연산자를 갖지 않으므로 연속적인 의사 명령어에 대해서는 최상위 수준에서 형제 노드로 연결하며 연산자형 명령어와 피연산자형 명령어에 대해서는 스택 운용 정보를 활용해서 EM 트리를 구성한다.

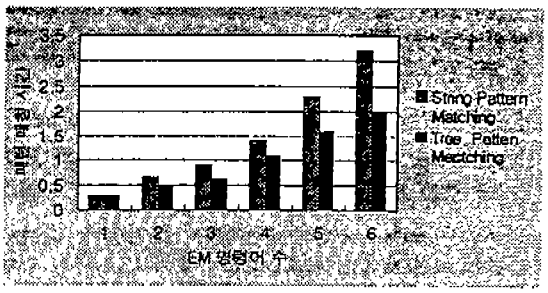
패턴 매칭기는 패턴 매칭 동작을 수행하기 위해 구성된 EM 트리를 먼저 하향(Top-down) 방식으로 순회하면서 루트 노드를 중심으로 구성될 수 있는 트리 패턴을 인덱스 테이블과 정보 테이블에서 검색한다. 정보 테이블에는 동일한 루트 노드를 중심으로 구성될 수 있는 트리 패턴이 노드의 개수에 따라 내림차순으로 정렬되어 있으며 보다 많은 개수를 가진 트리 패턴과 매칭이 이루어지는 것이 효과적이다. EM 트리에서 발견되는 트리 패턴을 정보 테이블에서 찾은 후에는 EM 트리의 피연산자 부분과 트리 패턴의 수식 부분에 매칭 여부를 검사한다. 트리 패턴과 식이 일치하는 패턴이 존재하면 패턴 매칭이 이루어지는 것으로 간주해서 정보 테이블의 인덱스를 EM 트리에 존재하는 패턴의 루트 노드에 저장한다. 또한 EM 트리상에서 매칭 결과와 매칭 범위를 표시하기 위해 플래그를 설정하여 동일한 매칭 범위를 갖는 노드는 동일한 플래그를 갖도록 한다. 하나 이상의 노드로 구성된 트리 패턴이 트리 스키마에 존재하지 않는 경우에는 하나의 트리 노드가 트리 스키마의 정보 테이블의 인덱스를 지정할 수 있도록 구성되어 있다.

실질적인 코드생성은 단말 노드까지 패턴 매칭을 완료한 후 EM 트리를 다시 상향(Bottom-up) 방식으로 순회하면서 정보 테이블의 내용을 참조하여 SPARC 코드를 바로 생성하거나 보다 효과적인 다른 EM 패턴 트리로 대치한다. 최종적으로 루트 노드에 대한 행위(Action) 부분을 수행하게 되면 트리 구조는 축약이 이루어져 코드 생성을 종료하게 된다. 이러한 트리 패턴 매칭을 이용한 목적코드 생성 기법은 ACK의 스트링 패턴 매칭 기법에 비해 보다 빠른 시간 내에 패턴을 발견할 수 있다. 또한 SPARC 코드 생성을 위한 목적기계 표현 테이블을 기술하므로서 재목적 가능한 방법으로 양질의 SPARC 코드를 생성한다.

3.3 성능 평가

본 논문에서 제안한 트리 패턴 매칭 알고리즘과 ACK

에서 사용한 스트링 패턴 매칭 알고리즘에 대한 성능을 측정하였다. 성능 측정을 위한 실험 환경은 SUN 워크스테이션의 SUN OS 4.1.3 환경에서 스텐포드 벤치마크 프로그램을 사용하여 (그림 8)과 같은 결과를 얻었다.



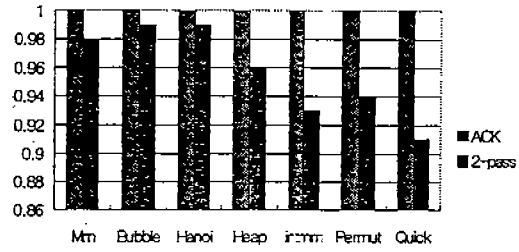
(그림 8) 패턴 매칭 시간 비교
(Fig 8) Comparison of pattern matching time

EM 명령어에 대한 패턴의 수가 증가될수록 스트링 패턴 매칭에 비해 트리 패턴 매칭 시간이 감소됨을 알 수 있으며 패턴을 구성하는 명령어의 수는 최대 6개로 제한하고 있다. EM 트리 구성시에 소비되는 메모리 양은 전체적으로 스트링 패턴 매칭 기법에 비해 많이 소비되지만 본 논문에서는 이를 고려하지 않고 있다. 그리고 생성된 코드의 질은 ACK의 코드 확장기를 이용한 코드의 질과 유사하다.

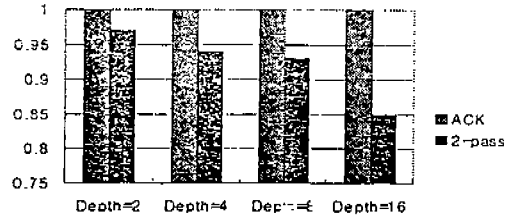
생성된 SPARC 코드는 본 연구에서 개발한 2-PASS 최적화기에 의해 최적화 된다. 2-PASS 최적화기는 ACK의 목적코드 최적화 동작을 수행한 후 파이프라인 조정과 같은 SPARC 기계의 특성을 고려한 최적화를 수행한다. 2-PASS 최적화를 수행한 결과에 대해서도 기존의 ACK 목적기계 최적화기와 성능을 비교 평가하기 위해 SPARC 워크스테이션의 Sun OS 4.1.3 환경에서 스텐포드 벤치마크 프로그램을 사용하였다. ACK와 비교하여 최적화 동작을 수행한 후의 실행 시간 대비는 (그림 9)와 같다.

또한 반복 횟수의 증가에 따른 프로그램 실행 시간 성능 비교는 다음과 같다.

여기서 Depth는 루프의 중첩된 횟수를 의미하며 중첩이 많을수록 생성된 SPARC 코드에 지연 슬롯이 많이 발생하므로 2-PASS 최적화기를 사용하여 실행 시간을 감소시킬 수 있다. 또한 위 실험 결과에서 보여 주듯이 ACK 방식과 2-PASS 방식을 비교해 볼 때 원시 프로그램에 제어문이 많거나 반복 횟수가 증가함에 따라 실행 시간이 감소되는 결과를 얻을 수 있다.



(그림 9) 프로그램 실행 시간 대비
(Fig. 9) Comparison of runtime



(그림 10) 반복에 따른 실행 시간 비교
(Fig. 10) Comparison of runtime for iteration

4. C++를 위한 대화식 프로그래밍 환경

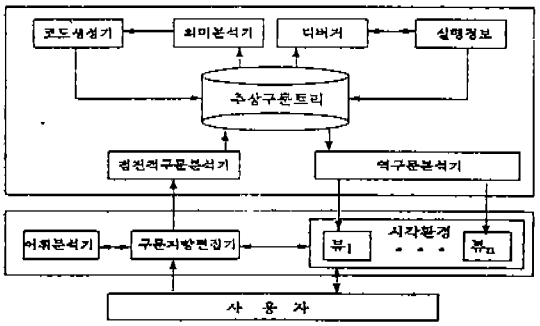
4.1 대화식 프로그래밍 환경의 구성

대화식 프로그래밍 환경은 프로그램 개발시의 일괄 처리 환경을 개선하여 원시 프로그램을 어휘 분석기로 입력하여 토큰화된 프로그램을 구문지향 편집기에서 보여주고, 편집 작업 후에 점진적 구문 분석기에서 구문 분석한다. 이 때, 수정 작업이 있으면 변경된 부분을 점진적으로 분석하여 추상 구문 트리를 구성한다. 추상 구문 트리로 변환된 프로그램은 의미 분석기를 거쳐 코드 생성기에 의해 중간 코드가 생성되고, 디버거에 의해 점진적인 실행을 수행하며 역구문 분석기에 의해 다양한 화면으로 출력된다.

객체지향 언어에서의 대화식 프로그래밍 환경은 클래스의 표현과 객체간의 실행 관계를 시각화하여

사용자에게 보여 주므로서 프로그램의 전체 구조에 대한 파악이 쉬우며, 프로그램의 오류를 쉽게 발견할 수 있으며, 프로그래밍 언어의 문법을 정확히 지킬 수 있는 장점이 있다[16, 17]. 여러 개의 화면을 통하여 프로그램의 여러 특성을 동시 파악할 수 있도록 다양한 그래픽 표현이 사용되고 있으며, 이를 통하여 프로그램 분석을 용이하게 할 수 있다. 또한 프로그램을 점진적으로 구성하고 실행하면서 프로그램을 작성할 수 있으므로 개발 시간을 줄일 수 있다.

본 연구의 대화식 프로그래밍 환경의 전체적인 구성은 (그림 11)과 같다. 여기서 제시된 각 모듈의 기능과 이에 사용되는 요소 기술들은 다음과 같다.



(그림 11) 대화식 시각 프로그래밍 환경
(Fig. 11) Interactive visual programming environment

• 어휘 분석기 : 입력된 원시 프로그램을 토큰으로 분리하여 구문지향 편집기로 전달한다. 또한 분리된 토큰으로 심벌 테이블을 만들고, 이 테이블은 구문 분석기와 역 구문 분석기에서 사용한다.

• 구문 지향 편집기 : 어휘 분석기에서 전달된 토큰을 이용하여 프로그램 작성자에게 익숙한 형태로 프로그램을 전개한다. 내부 표현 형태는 이미 토큰으로 변환되어 있으므로 프로그램의 들여 쓰기(indentation)나 주석에 관한 처리가 이루어진다. 역 구문 분석기에 의해 전달된 정보를 이용하면 프로그램이 정돈된 형태로 보여진다. 다른 시각환경에서 수정된 부분도 구문지향 편집기에 일차적으로 변경이 발생한 뒤에 구문 분석이 이루어진다.

• 점진적 구문 분석기 : 점진적 대화식 프로그래밍 환경에서는 프로그램의 수정이 발생하면 수정된 부분은 즉시 추상 구문 트리에 반영되어야 한다. 기존의 구문 분석 방법과 달리 점진적 구문 분석 방법은 프로그램에서 수정된 부분을 중심으로 수정에 의해 프로그램의 의미가 영향을 받는 부분에 대해서만 분석을 수행하므로 훨씬 효율적이다.

• 추상 구문 트리 처리기 : 추상 구문 트리와 심볼 테이블과 같은 시스템 정보를 관리한다. 노드의 정보가 변화될 때 뷰에서의 클래스나 메시지의 시각적인 표현이나 텍스트 그리고 추상 구문 트리 노드가 일치하도록 하고, 마우스 포인터에 의해 노드를 추적할 수 있도록 항상 최근의 상태를 유지하여야 한다.

• 의미 분석기 : 프로그램의 수정에 의해 추상 구문 트리가 변경되면, 수정된 추상 구문 트리를 이용한 의미분석이 이루어진다. 각 시각 환경에서 프로그램의 수정에 따라 변경된 추상 구문 트리에 따라 일치성을 확인하고 이를 다른 시각 환경에서 이용할 수 있도록 한다.

• 코드 생성기 : 대화식 프로그램 개발 환경에서는 프로그램의 수정이 이루어진 이후, 언제라도 바로 실행 가능한 상태이어야 한다. 이러한 과정을 대화식으로 처리하기 위하여 프로그램 작성과 구문 분석이 동시에 점진적으로 처리되어야 한다. 점진적인 코드를 갖는 노드를 추상 구문 트리에 첨가하여 점진적 수행이 가능하도록 한다.

• 디버거 : 코드 생성기에서 생성된 코드를 실행하면서 프로그램의 실행 경로와 수행결과의 확인이 가능하도록 한다. 이를 위해서는 추상 구문 트리에 생성된 코드를 검색하면서 실행한다. 또 프로그램 개발 과정 중에도 사용이 가능하도록 불완전한 구문과 프로그램에 대해서도 수행이 가능하도록 하였다.

• 역 구문 분석기 : 추상 구문 트리의 심볼 테이블을 입력으로 하여 들여 쓰기(indentation), 줄 바꿈(line break), 쪽 바꿈(page break) 등에 맞추어 추상 구문 트리에 해당하는 원래의 텍스트와 그래픽 심볼을 시각 환경의 각 뷰나 인쇄 장치에 보기 좋은 형식으로 출력한다. 역 구문 분석기는 AST를 입력으로 받고, 각 트리에 해당하는 원래의 텍스트 형태의 문장을 화면에 출력한다. 또한 프로그램을 시각기호로 표현하기 위한 적절한 정보를 시각환경에 제공하여 시각기호로 프로그램을 표현하도록 한다.

• 시각 환경 : 시각 환경은 여러 가지 뷰들로 구성되는

데 초기화면(Cool View)과 원시 코드 뷰(Source Code View), 클래스 뷰(Class View), 메시지 전달 뷰(Message Passing View)등이 제공된다. 이 시각 환경은 기본적으로 시스템과 개발자가 직접 접속되는 부분이다.

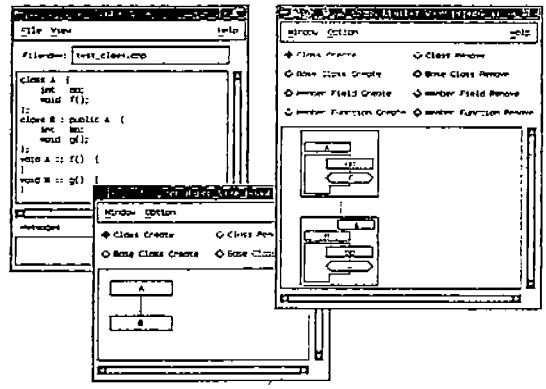
4.2 그래픽 사용자 인터페이스

본 연구에서는 다중 뷰를 이용한 C++ 언어 프로그래밍 개발환경을 위하여 클래스를 중심으로 시각화하였다. 따라서 클래스에 대한 상세하고 명료한 시각정보 전달이 가능한 시각기호를 설계하여야 한다. C++ 언어에서 클래스는 다중 상속이 허용되므로 상속성을 표현하기 위해 순환이 없는 방향 그래프를 사용한다. 그러나 이 상속성 그래프만으로는 사용자와 개발자 모두에게 충분한 정보를 제공치 못한다. 그러므로 C++ 언어의 클래스를 보는 관점을 사용자의 관점과 개발자의 관점 두 가지로 구분할 필요가 있다. 사용자의 관점에서는 허용된 멤버들에 대한 가시성과 클래스의 상속성 정도로 충분할 수 있을 것이다. 그러나 개발자의 관점에서는 클래스 내부의 모든 멤버들에 대한 상세한 정보가 반드시 필요하므로 이들 정보를 단계적으로 제공할 수 있는 방법이 고려되어야 한다.

본 연구에서는 Raimund의 시각기호[18]를 기본으로 하여 구성하였다. 시각환경은 다중 뷰로 구성되며 본 연구의 프로그래밍 환경에서는 클래스와 관련하여 원시 코드, 클래스 상속성, 클래스와 멤버의 세 가지 뷰를 제공하도록 하였다. 사용자에 의해 발생하는 이벤트들은 모두 시각 환경을 통해 해석되고 점진적인 속성 평가에 의한 프로그램의 파싱과 unparsing이 이루어지게 된다[19, 20, 21]. 제공되는 세 가지 뷰는 다음과 같다.

• 원시코드 뷰: 원시 코드 뷰에서는 새로운 원시 코드의 읽기와 직접적인 원시 코드의 수정이 발생되는 곳으로서 시각 환경의 가장 기본적인 뷰로 제공이 되고 다른 뷰를 활성화시킬 수 있다. 원시 코드 뷰는 기본적으로 문자 기반의 편집기 형식을 취하고 있으며, 각종 시스템의 메시지를 표시하는 메시지 상자도 포함하고 있다.

• 클래스 상속 뷰: 클래스 상속성 뷰에서는 클래스의 이름과 상속성을 그래프로 시각화하고 사용자는 이를 직접 수정할 수 있게 된다. 클래스는 이름을 갖는 상자모양의 정점으로 상속성은 이들 사이의 간선으로 표시된다. 여기서는 클래스의 생성 및 삭제와 상속성 즉 베이스 클래스의 생성 및 삭제가 가능하다. 이는 마우스의 조작을 통하여 이루어지며 메뉴의 선택 여부에 따라서 다른 의미로 해석된다. 클래스 생성의 경우에는 마우스 버튼의 클릭과 버튼을 누르고 끌어다 놓기, 두 가지 방식의 조작이 가능하다. 클래스가 표시되지 않은 영역에서 마우스 버튼의 클릭은 슈퍼 클래스의 생성으로 해석되고 클래스의 이름을 입력받기 위한 입력상자가 열린다. 다음으로 버튼을 누르고 끌어다 놓기는 버튼이 눌려진 위치에 있는 클래스를 베이스 클래스로 하는 클래스의 생성을 의미하며 마찬가지로 이름을 입력받는다. 이때 버튼의 놓기는 클래스가 존재하지 않는 영역에서 이루어져야 한다. 상속성을 생성할 경우에는 마우스 버튼을 누르고 끌어다 놓기가 클래스 생성 시와 유사한 의미로 해석되며 이때는 존재하는 두 클래스 사이의 상속성만을 생성하므로 추가적인 입력이 없다. 클래스와 상속성의 삭제 시에는 마우스 버튼이 클릭된 위치에 해당하는 클래스나 상속성이 존재하면 해당 요소의 삭제로 해석된다.



(그림 12) C++를 위한 다중 뷰를 이용한 시각환경
(Fig. 12) Multi-view visual environments for C++

• 클래스와 멤버 뷰: 클래스와 멤버 뷰에서는 클래스의 상속성과 클래스 내의 멤버들의 구성을 상세히 시각화하고 시각 프로그래밍 할 수 있게 한다. 여기서도 사용자의 명령은 마우스를 통하여 전달되고 클래

스와 상속성의 생성과 삭제는 클래스 뷰에서와 동일하게 작동한다. 그밖에 멤버 필드와 멤버 함수의 생성은 해당 클래스의 적당한 위치에 마우스 버튼의 클릭으로 수행하는데 이때 마우스 포인터의 위치가 클래스의 내부와 외부, 하단으로 구분되어 각 멤버의 접근 제어가 private, public, protected로 결정된다. 멤버의 삭제는 클래스나 상속성의 삭제와 동일한 방식으로 처리된다.

본 연구에서 구성된 대화식 프로그래밍 환경은 Sun Sparc 시스템에서 Gnu C/C++컴파일러와 X-window 시스템, OSF/Motif 위젯을 이용하여 구현하였다. 인터페이스를 위해 정의된 대부분의 클래스는 Motif 위젯을 C++ 언어에서 사용할 수 있도록 변한 Motif 위젯 클래스들이다. 일반적으로 사용되는 그래픽 사용자 인터페이스 요소들은 클래스로 정의하여 재 사용성을 높였다. 개발된 시스템을 현재 널리 사용되는 C++ 언어를 위한 프로그래밍 환경과 비교해 보면 다음과 같은 특징을 가진다. 현재 상용화된 C++를 위한 프로그램 개발 환경으로는 Microsoft사의 Visual C++와 Sun Microsystems 사의 Sparc Works 가 있다. 이들 제품들은 C++ 언어의 컴파일러를 기반으로 하여 프로그램의 편집과 실행, 디버깅 환경을 제공하고 있다. 프로그램의 이해를 도와주기 위한 다양한 시각 환경도 제공하고 있으나 프로그램의 실행과 디버깅은 일괄처리방식으로 이루어진다. 이들 제품을 몇 가지 기능 면에서 본 연구에서 구성된 대화식 프로그래밍 환경과 비교하면 〈표 1〉과 같다.

〈표 1〉 C++ 프로그래밍 환경의 비교
〈Table 1〉 Comparison of C++ programming environment

시스템 / 기능	Sparc works	Visual C++	본 연구에서 개발된 시스템
클래스 상속성 시각화	가 능	가 능	가 능
클래스 멤버 시각화	가 능	불가능	가 능
시각 프로그래밍	불가능	불가능	가 능
단계적 프로그램 표현	가 능	가 능	가 능
편집시 문법확인	불가능	불가능	가 능
점진적 분석	불가능	어휘분석수준에서 가 능	가 능

C++를 위한 대화식 시각 프로그래밍 환경은 클래스를 중심개념으로 클래스의 멤버 정보과 특성을 표현하기 위한 시각 기호가 제공되는 환경으로 대화적 처리가 가능한 여러 가지 뷰를 제공한다. 이는 C++ 프로그램에서 클래스와 객체간의 메시지 처리를 시각화하고 이를 이용한 프로그래밍이 가능한 시각 프로그래밍 환경을 제공한다. 객체간의 실행 시에 발생하는 메시지 전달을 시각적으로 표현함으로써 프로그램의 실행시 구조 파악이 용이하므로 프로그램의 테스트가 간편해 지므로 개발기간을 줄일 수 있게 된다. 점진적 프로그래밍 환경은 소프트웨어 재사용 및 사양 변경에 따른 전환이 용이하여 강력하고 효율적인 객체지향 프로그래밍 환경으로서의 기능을 충분히 발휘할 수 있을 것이다.

5. 결 론

프로그래밍 언어가 다양해지고 컴퓨터 하드웨어 기술이 급속히 발전함에 따라 컴파일러 및 프로그래밍 환경 개발에 대한 연구가 활발히 진행되고 있으며 그에 대한 자동화 이론도 상당히 진척되었다. 또한, C++ 언어는 기존의 C 언어를 기반으로 클래스, 상속성, 다형성과 같은 객체 지향 특성을 지원하고 있어 소프트웨어의 구현 및 응용 분야에서 널리 사용되고 있다. 본 논문에서는 최신의 컴파일러 개발 이론을 적용하여 C++ 컴파일러 및 대화식 프로그래밍 환경을 개발하였다. 컴파일러 개발에 있어서는 Front-End와 Back-End로 나누어 가상 기계를 이용하여 연결하는 모델을 사용하였으며 프로그래밍 환경 개발에서는 정형화된 방법을 적용하여 다양한 환경을 개방적으로 구성할 수 있는 기술을 사용하였다. 이렇게 개발된 C++ 컴파일러와 대화식 프로그래밍 환경을 결합하여 C++ 언어를 위한 통합 프로그래밍 환경을 구축하였다.

컴파일러 개발에 사용한 Front-End/Back-End 모델은 컴파일러의 이식성 및 재목적성을 추구할 수 있었으며 개발의 편의성도 얻을 수 있었다. Front-End와 Back-End를 연결해 주는 중간 언어로는 ACK에서 제공하는 EM 코드를 사용하였다. 이로 인해, Front-End와 Back-End 접속을 명확히 할 수 있었으며 또한 개발의 독립성도 달성할 수 있었다.

Front-End 개발시 문법 분석 도구, 어휘 및 구문 분석기 작성 도구 등을 최대한 활용하여 효율성 및 신뢰성을 확보하도록 하였다. C++ 언어를 중간 코드인 EM으로 번역하는 과정을 C++ 언어로 구현함으로써 기존 도구의 사용을 그대로 유지하면서도 기존의 컴파일러 구현 과정에서의 복잡성과 해결책의 다양성을 줄이고 객체지향개념을 지닌 언어의 구현 방법에 대한 방향을 제시하였다. Back-End 개발과정에서는 컴파일러 제작 도구인 ACK를 분석하고 개선하여 재목적이 가능한 Back-End 모델을 구축하였고 모델로부터 제안된 세 가지 컴포넌트를 구현하였다:목적 코드 생성기, 최적화기, 실행시간 환경. 이와 같은 컴포넌트를 상호 연결하여 중간 코드인 EM을 목적 코드인 SPARC 코드로 변환하는 Back-End를 완성하였다. 대화식 프로그래밍 환경은 프로그래머의 생산성을 높일 수 있는 효율적인 프로그래밍 도구이다. 본 논문에서는 대화식 환경을 일반화하여 자동 생성이 가능하도록 분석된 프로그램을 시각기호로 표현하는 정형화된 unparsing 체계를 개발하였다. 또한 언어의 다양한 특성을 효과적으로 표현하기 위하여 추상구문트리를 정의하였고 신속한 구문 분석을 위해 점진적 파싱 기법을 사용하였으며 시각기호와 시각환경을 구성하였다.

본 논문에서는 컴파일러 및 프로그래밍 환경을 구현하는 기법을 기능별로 나누고 기능에 따른 모듈들을 구현하여 각 모듈의 재사용성을 높였다. 그리고 다음과 같은 구현 방법에 대한 이론을 적용하고 개선할 수 있었다. Front-End 구현시 문맥에 연동된 문법처리 기술과 AST 클래스 라이브러리, Back-End 구현시 적용한 트리 패턴 매칭에 의한 재목적 코드 생성 기법, 프로그래밍 환경 개발시 추상구문트리를 이용한 점진적 분석 기법과 정형화된 unparsing 체계 등이다. 이와 같은 연구 결과는 앞으로 다양한 프로그래밍 언어의 컴파일러 및 대화식 프로그래밍 환경 개발에 효과적으로 이용될 수 있을 것이다. 특히, 본 논문에서 적용한 컴파일러 개발 기법은 객체지향 특성을 갖는 언어를 효과적으로 목적 코드로 번역하는 언어 번역기와 그에 따른 프로그래밍 환경을 개발하는데 유익하게 사용할 수 있다. 본 논문에서도 그와 같은 기법을 이용하여 성공적으로 C++ 프로그래밍 환경을 구축할 수 있었다. 그러나, 완벽한 개발품이라기보다는 실험적으로 활용할 수 있는 프로토타입으로 개발되었다. 보다 완벽한 개발품이 되기 위해서는 앞으로도 많은 연구와 노력이 뒤따라야 한다.

본 논문에서 제시한 컴파일러 및 프로그래밍 환경 개발 기술은 새로운 고급 언어 및 기계에 대한 컴파일러 개발에 활용될 수 있으며 실제 상용화된 C++ 컴파일러 개발에 활용될 수 있다. 또한, 병렬 및 분산 환경에서 수행할 수 있는 컴파일러 개발에도 활용할 수 있을 것이다.

참 고 문 헌

[1] Coplin, J.O., "Advanced C++ Programming Styles and Idioms," Addison-Wesley, 1992.

[2] Bjarne Stroustrup, "The C++ Programming Language," 2nd edition, Addison-Wesley, 1991.

[3] Margaret A. Ellis, Bjarne Stroustup, "The Annotated C++ Reference Manual," Addison-Wesley, 1990.

[4] Stallman, R.M., "Using and Porting GNU CC," Free Software Foundation. 1992.

[5] Stephen C. Dewhurst, "Flexible Symbol Table Structure for Compiling C++," Software Practice & Experience Vol. 17, No. 8, pp.503-512, 1987.

[6] Bruce Hahne, Hiroyuki Sato, "Using YACC and LEX with C++," ACM SIGPLAN Notices, Vol. 29, No. 12, December 1994.

[7] Jim Holmes, "Object-Oriented Compiler Construction," Prentice-Hall, 1995.

[8] Alfred V. Aho, Mahadevan Ganapathi, Steven W. K. Tjiang, "Code Generation Using Tree Matching and Dynamic Programming," ACM TOPLAS, Vol. 11, No. 4., pp.491-516, Oct., 1989.

[9] R. G. G. Cattell, "Automatic Derivation of Code Generators from Machine Descriptions," ACM TOPLAS, Vol. 2, No. 2, pp.173-190. Apr., 1980.

[10] Mahadevan Ganapathi, Charles N. Fischer, John L. Hennessy, "Retargetable Compiler Code Generation," ACM Computing Surveys, Vol. 14, No. 4, pp.573-592, Dec., 1982.

[11] Hans van Staveren, "The table driven code generator from ACK 2nd. Revision," report-81, Netherlands Vrije Universiteit, 1989.

[12] Visual C++ Tutorials, Microsoft, 1995.

[13] Sparc Works Tutorial, Sunsoft, 1994.

[14] R. Medina-Mora and P.H.Feiler, "An Incremental Programming Environments," *IEEE Trans. On SE*, vol. 7, no. 5, pp.472-481, Sep. 1981.

[15] T. Teitelbaum and T. Reps, "the Cornell Program Synthesizer:A syntax-Directed Programming Environment," CACM, Vol. 24, no. 9, pp. 563-573, Sep. 1981.

[16] Adele Goldberg, Margaret Burnett and Ted Lewis "What is Visual Object-Oriented Programming?," in *Visual Object-Oriented Programming*, Manning Pub. Co, pp.3-20, 1995.

[17] Wim De Pauw, Richard Helm, Doug Kimelman, and John Vlissides, "Visualizing the Behavior of Object-Oriented Systems," *OOPSLA'93*, pp. 326-337, 1993.

[18] Raimund K. Ege, *Programming in an Object-Oriented Environment*, Academic Press, 1992.

[19] Ephraim P. Glinert, *Visual Programming Environments: Paradigms and Systems*, IEEE Computer Society Press, 1990.

[20] Margaret M. Burnett, Adele Goldberg, Ted G. Lewis, *Visual Object-Oriented Programming-Concept and Environments*, Manning Publications Co., 1994.

[21] Scott Meyers, *Representing Software System in Multiple-View Development Environment*, Ph.D. Thesis, Brown University, Department of Computer Science, 1993.

[22] 유재우, 류천열, 정근호, "C++를 위한 대화식 다중 뷰 시각 프로그래밍 환경", *한국정보처리학회 논문지* 제2권 제5호, pp.746-756, 1995. 9.

[23] 주상은, 김태완, 장천현, "AST 클래스 라이브러리를 이용한 C++ 컴파일러 front-end의 구현", 한국정보처리학회학술발표논문집, 2권 1호, pp. 479-482, 1995, 4.

[24] 김태완, 이정현, 장천현, "DAST를 이용한 중간 코드 생성", 한국정보처리학회 학술발표논문집, 제2권, 제2호, pp.185-189, 1995, 10.

[25] 고광만, 김정숙, 오세만, "트리 패턴 매칭 기법을 이용한 코드 생성 알고리즘의 개발", 한국정보과학회 학술발표논문집, 제22권, 제1호, pp.992-996, 1995, 4.

[26] 김성철, 오세만, "ACK의 코드 생성 시스템에 관한 연구", 한국정보과학회 학술발표논문집, 제21권, 제2호, pp.89-92, 1994, 10.

장 천 현

1977년 서울대학교 계산통계학과(학사)
1979년 한국과학기술원 전산학과(석사)
1985년 한국과학기술원 전산학과(박사)
1988년~1989년 Virginia 대학교 연구원
1984년~현재 건국대학교 컴퓨터공학과 교수
관심분야:컴파일러, 프로그래밍언어

오 세 만

1977년 서울대학교 사범대학 수학과(학사)
1979년 한국과학기술원 전산학과(석사)
1985년 한국과학기술원 전산학과(박사)
1988년~1989년 미국 USL 대학교 환교수
1985년~현재 동국대학교 컴퓨터공학과 교수
관심분야:컴파일러, 프로그래밍언어, 병렬 및 분산처리 언어

유 재 우

1976년 숭실대학교 전자계산학과(학사)
1978년~1985년 한국과학기술원 전산학과(석사, 박사)
1986년~1987년 Cornell Univ. Visiting Scientist
1983년~현재 숭실대학교 컴퓨터공학부 교수
관심분야:컴파일러, 프로그래밍 환경, HCI