C++ 중간 코드를 이용한 CHILL96 컴파일러의 설계 및 구현

금 창 섭† · 이 준 경†† · 이 동 길††† · 이 병 선†††

요 약

본 논문에서는 ITU-T에서 통신시스템 구현을 위해 제안된 CHILL96 언어를 C++ 언어로 변환하는 컴파일러의 설계 및 구현에 관하여 기술하였다. C++ 코드를 생성하기 위해서 CHILL96 언어에서 C++ 언어로의 변환 규칙을 고안하였다. CHILL96 컴파일러는 심볼 테이블과 추상구문트리와 밀접한 관계를 갖는 구문 분석기, 가시성 제어기, 의미 분석기, 코드 생성기로 이루어져 있다. 또한 본 논문에서 기술한 CHILL96 컴파일러의 유용성을 알아보기 위해 성능 평가를 수행하였다. 성능 평가 결과 C++ 코드를 중간코드로 사용하는 CHILL96 컴파일러는 이전에 개발된 다른 CHILL 컴파일러들에 비해 우수한 성능을 보여주었다. 이 논문에서 개발된 CHILL96컴파일러는 성능과 이식성의 향상 이외에도 기존에 CHILL로 개발된 통신 소프트웨어들을 C++로 변환함으로써 신규 기능의 추가나 유지보수에서 편의성을 높였다.

Design and Implementation of a CHILL96 Compiler Using C++ Intermediate Code

Chang-Sup Keum† · Joon-Kyung Lee†† · Dong-Gill Lee††† · Byung-Sun Lee†††

ABSTRACT

CHILL96 is recommended as development language for telecommunication systems by ITU-T. In this paper, we describe the design and implementation of CHILL96 compiler using C++ intermediate code. Translation rules from CHILL96 to C++ are designed for code generation. The CHILL96 compiler is composed of four parts such as syntax analyzer, visibility checker, semantic analyzer and code generator, and each part has very close relationship with symbol table and abstract syntax tree. Performance evaluation has been performed for feasibility study. After performance evaluation, we conclude the CHILL96 compiler using C++ intermediate show good performance compared with other CHILL compilers. In addition to high performance and portability, the CHILL96 compiler using C++ intermediate code helps application developers to maintain and enhance telecommunications software by translating CHILL96 program to C++ program.

1. 서 론

80년대에 걸쳐 ITU-T(Telecommunication Standardization section of the International Telecommunication Union)에서 제안, 수정, 권고된 CHILL[1] 언어는 어셈블리 언어로 방대한 시분할 교환기용 프로그램을 작성하던 한 시대를 마감하고 고급 언어로 교환기용 프로그램을 작성할 수 있도록 하였다. 교환기용 프로그램에 사용되는 CHILL 언어는 일반 프로그래밍 언어와는 달리 실시간 처리가 가능해야 하고, 신뢰성이 대단히 필요한 언어이다. 이러한 목표에 부합하면서 프로그램의 개발 시간과 경비를 줄이고 품질의 향상을 도모할 수 있도록 CHILL 언어는 기존의 프로그래밍 언어인 C,

† 정 회 원 : 한국전자통신연구원 연구원
†† 정 회 원 : 한국전자통신연구원 선임연구원
††† 정 회 원 : 한국전자통신연구원 책임연구원
논문접수 : 1999년 8월 9일, 심사완료 : 2000년 5월 16일

Modula-2, Ada 등의 특징을 대부분 포함하고 몇 가지 부가적인 특성을 가지고 있다. 특히 다양한 모드의 제공을 비롯하여 가시성 제어라는 독특한 구조를 제공하고, 병행 수행의 허용과 병행 프로세스간의 상호 배제(mutual exclusion)가 제공되었다 CHILL은 명령형 언어를 기반으로 설계되었으므로 다양하고도 강력한 기능을 제공함에도 불구하고 구조적으로 유사한 시스템을 일일이 재개발하는 경우가 자주 발생하였다. 이에 대한 해결책으로 프로그램의 재사용과 확장을 효과적으로 지원하는 객체지향 프로그래밍을 위한 기능을 CHILL에 추가하기 위한 연구가 ITU-T에 의해 진행되었고, 이 연구의 결과로 제안된 것이 CHILL96[2]이다. CHILL96은 CHILL 언어에 객체지향성, 중복성, 포괄성 등의 고급 기능을 추가한 것이다.

Bjarne Stroustrup은 AT&T Bell 연구소에서 C에 Simula와 유사한 클래스를 추가하여 C++[3]를 개발하였다. C++는 그 사용자들의 요구로부터 개발되었으며 효율적이고 실용적인 언어가 되도록 설계되었다. 1985년에 소개된 이후로 몇 년 안에 이 언어는 가장 많이 사용되는 객체지향 언어가 되었다. 현재 많이 사용되는 C++ 컴파일러는 AT&T C++ 컴파일러와 Free Software Foundation에서 만든 GNU C++[4] 컴파일러가 있다. GNU C++는 공용 도메인(public domain)에서 무료로 사용이 가능하며 소스 코드가 공개 되어서 필요에 따라 추가 기능 개발 및 수정이 가능하다.

CHILL96 to C++ 컴파일러를 개발하는 이유는 다음과 같다. 첫째, GNU C++ 컴파일러를 후단부(backend)로 사용함으로써 이식성과 성능이 향상된다 이전에 개발된 CHILL96 컴파일러는 EM(encoding machine) 코드를 중간 코드로 사용함으로써 자체적으로 계속된 후단부 개발이 필요하였다. GNU C++ 컴파일러는 거의 모든 H/W 플랫폼을 지원하고 있으며 막강한 최적화 기능을 갖추고 있다. 특히 GNU C++ 컴파일러를 사용함으로써 공개된 소스 코드를 확보하여 안정화 및 확장 개발할 수 있다. 둘째, CHILL96으로 이미 개발된 통신 소프트웨어의 유지 보수와 확장 개발이 쉽다. 국내 교환기 소프트웨어들은 대부분 CHILL 언어로 구현되어 있으며, 그 규모는 백만 라인을 상회한다. 그러나 대부분의 개발자들은 CHILL96 언어보다 C++ 언어를 잘 알고있다. 따라서 CHILL96 언어로 작성된 통신 시스템 소프트웨어를 C++로 변환할 경우 CHILL96을 모르는 개발자들에게 소프트웨어의 이해와 유지 보수에 많은 도움을 줄 것이다. 셋째, C++ 언어를 지원하는 강력한 상용 개발도구와 공용 도메인 소프트웨어의 사용이 가능하다. CHILL96은 특정 분야에만 사용되는 언어이므로 이를 지원하는 도구들이 많지 않다. 반면 C++는 많은 상용 개발도구와 공용 도메인 도구들이 존재하여 고품질의 소프트웨어 개발을 기대할 수 있다

2. CHILL96 의 주요 개념 및 특성

2.1 CHILL96의 3가지 주요 개념

객체지향 패러다임(paradigm)에 기반한 CHILL96은 다음과 같은 세 가지 주요 개념을 갖는다.

(1) 객체지향성 : 실세계의 현상을 살펴보면 하나의 독립적인 개체들이 독자적이고 병행적으로 수행된다. 이들 독립적인 개체들간에 상호관련 작업이 필요하면 메시지를 주고 받는다[5] 즉 송신자는 수신자의 상태에 관계없이 메시지를 보내고 자신의 작업을 계속 수행하며, 수신자는 받은 메시지를 살펴보고 자신의 현재 상태에 만족한 메시지를 선별적으로 하나씩 받아 해당 작업을 수행한다[6, 7]. 이기서 독자적으로 수행하는 개체를 객체(object)로 간주하고, 이 객체는 내부적으로 기억 장소뿐만 아니라, 이를 연산할 메소드(method)도 함께 갖는다. 이때 독자적으로 실행하기 위해서는 내부에 쓰레드(thread)를 가져야 하는데 이를 병행(concurrent) 객체라 한다[6-8] 그러나 내부의 쓰레드가 없으면 이를 순차(sequential) 객체라 하는데 외부의 병행 개체(프로세스 혹은 병행 객체)들이 순차 객체에 메소드 호출을 동시에 요청하면 순차 객체의 내부 데이터에 대한 동시 접근을 막을 수 없다. 이러한 객체 내부의 데이터를 임계 구역(critical region)으로 안전하게 보장하기 위해 병행적 접근에 대해 상호배제 동기화 기능을 제공한 객체를 모니터(monitor) 객체라 한다[9]. 이러한 세가지 종류의 객체를 생성시키는 일종의 형판(template)을 클래스라 하는데, 이 클래스를 CHILL96에서는 강한 정적 타입 점검을 위해 타입인 모드(mode)에 포함시켰다.

(2) 중복성(overloading) : 중복의 의미는 서로 다른 행위를 갖는 개체에 대해 같은 이름을 부여하는 것으로써 하나의 이름이 여러 개체를 지칭하므로 다형성(polymorphism)에 포함된다[10] 정확히 말하면 중복적 다형성이라 하는데, 이의 적용 가능성은 프로그램내의

모든 식별자(identifier)에 해당될 수 있다. 예를 들면, 같은 이름의 변수들이 다른 타입들을 갖는 경우, 같은 이름의 프로시듀어들이 다른 행위(혹은 구현 본체)를 갖는 경우, 같은 이름의 집합 원소가 다른 순서 값을 갖는 경우 등이 있다[11]. 그런데 이들 경우 중에는 프로그램의 구문과 어의 및 컴파일러를 복잡하게 만들며, 실제적으로 사용자에게 효용성이 없으므로 제한된 경우만을 프로그래밍 언어들이 제공한다. CHILL96에서는 프로시듀어와 프로세스 중복성에 한정하여 제공하고 있다.

(3) 포괄성(genericity) : 포괄의 의미는 복합적인 구조를 갖는 한 개체가 있을 때 그 개체 내에서 사용되는 값이나 타입이 결정되지 않고, 그 개체의 실제적인 개체 생성시 매개인자-전달 방식에 의해 값이나 타입을 결정하여 사례화(instantiation)하는 것을 지칭한다[3]. 이때 컴파일러는 생성할 개체의 정보에 대해 값이나 타입을 모르면 정확한 사용 메모리를 생성할 수 없으며, 이는 실행의 오류를 유발시킨다. 이와 같이 하나의 이름을 갖는 복합 개체가 실제 작용시 매개인자에 의해 여러 값과 여러 타입을 갖는 다중의 복합 개체들로 사례화하는 것을 매개적 다형성이라 하는데 이도 또한 소프트웨어의 재사용성을 제공한다[3, 11].

2.2 CHILL96의 주요 언어적 기능

(1) 클래스의 타입 적용 : 객체를 생성시키는 일종의 형판 역할을 하는 것을 클래스라 하는데, 과거에는 클래스를 기존 타입과 별개로 취급하였다. 즉 클래스를 기존 타입의 확장으로 생각하지 않고 클래스 구조 형태를 띠로 설정함으로써 객체 생성시에는 이 클래스에 의한 사례화로 적용하였다[12]. 그러나 교환기 시스템 상각상 실행의 안전성을 강한 정적 타입 검증으로 보장하고 기존 CHILL의 구문과 어의를 가급적 최소화한다는 원칙에 의해 클래스를 기존 CHILL의 복합 모드(타입)에 통합하였다.

(2) 클래스의 종류 : 앞서 언급한 바와 같이 클래스를 세가지로 분류하였다. 즉 기존 CHILL의 모듈에 근거한 순차(module) 객체, 모니터에 기반한 임계(region) 객체 및 모듈과 프로세스를 결합한 타스크(task) 객체로 구성하였다. 특히 병행 객체는 자신의 쓰레드를 가지므로 일련의 실행문을 가질 수 있지만 이를 사용자에게 은폐하도록 암시적인 병행적 쓰레드 실행문을 갖는다.

(3) 객체의 구성 원소 : 객체를 생성시키는 모든 클래스를 CHILL96에서는 모레타(morcta : MOdule, REgion, TAsk) 모드라 지칭하였다. 왜냐하면 기존 CHILL의 타입 접점에 대한 어의 용어로 이미 클래스가 존재하고 있기 때문이다[2] 그리고 모레타 모드의 구성 원소를 살펴보면 내부 데이터 메모리에 필요한 DCL-데이터 선언문, 이를 운용하는 프로시듀어인 메소드 정의문, 내부의 병행성을 갖는 프로세스 정의문 및 이들에 대한 값이나 타입 혹은 외부에서 가져온 식별자 정보 등을 포함하고 있다.

(4) 객체의 정보 은닉 외부에서 객체에 서비스를 요청하기 위해서는 객체에 대한 추상적인 정보가 필요하다 이때 그 객체의 정보는 모레타 명세 모드가 가지고 있는데, 모레타 모드는 외부 사용자나 하위 객체 생성시 컴파일러에게 필요한 모레타 명세 모드와 객체 생성시 컴파일러에게 필요하며 외부에 구현 정보를 보여줄 필요가 없는 모레타 본체 모드로 분류하고 있다. 따라서 모레타 본체 모드 속의 정보는 C++처럼 private 가시성을 가지며, 명세 모드 속의 정보는 파생된 하위 모레타 모드들에만 보여주는 internal 가시성 혹은 모레타 객체 외부에서도 볼 수 있는 public 가시성 등 3종류의 은폐성을 갖는다

3. CHILL96 to C++ 변환 규칙

CHILL96의 프로그램의 단위는 가시성을 제어하는 모듈리인(모듈과 리전)과 장소의 존재성을 결정하는 블록(프로시듀어, 프로세스, begin-end)이 된다. 각 프로그램 단위는 크게 세 부분으로 구성되어 있다. 각 프로그램 단위의 첫 번째 부분은 프로그램의 구조를 결정하는 문장들이다 두 번째 부분은 자료 객체를 기술하는 부분으로써 모드, 값, 장소, 가시성 제어 등이 포함된다 세 번째 부분은 프로그램 실행문에 해당하는 부분으로 치환문, 제어문, 반복문 등이 있다. CHILL96에서 C++로의 변환은 CHILL96과 C++ 언어의 유사성으로 인하여 대부분의 변환은 쉽게 이루어지지만 CHILL96의 모듈구조, 비트스트링 치환, 병행처리 구문 등의 변환에 대해서는 유의가 필요하다.

3.1 프로그램 구조의 변환

모듈과 리젼은 데이터 추상화를 제공하는 프로그램 구조로서 모듈리언 내에 선언된 이름은 그 모듈리언 내에서만 가시성을 가진다. 모듈리언 내에 선언된 이름을 밖으로 보여주기 위해서는 CHILL의 GRANT선언문을 사용하고 모듈리언 밖에 있는 이름을 사용하기 위해서는 SEIZE선언문을 사용해야 한다. 그러나 C++에서는 모듈리언에 대응하는 프로그램 구조가 존재하지 않는다. 따라서 CHILL96을 C++로 변환할 때는 모듈리언 구조는 없어지며 서로 다른 모듈에서 사용되던 같은 이름들의 중복을 피하기 위해서 모듈리언 내의 이름들은 재명명하게 된다. 재명명 규칙은 모듈리언 내의 모든 식별자 이름 뒤에 모듈리언의 이름을 붙이는 것이다 그러나 GRANT된 이름은 프로그램내에서 유일한 이름이므로 재명명하지 않는다. SEIZE 문은 모듈리언 바깥의 이름을 사용할 수 있도록 하는 기능을 제공하는데 C++에서는 모듈리언이 존재하지 않으므로 사용될 필요가 없으므로 C++로 변환될 때 SEIZE문은 무시된다.

〈표 1〉 프로그램 구조 변환 규칙

CHILL96	C++
module/region	*vanish*
process	function
procedure	function
moreta mode	class
begin-end block	{… }

CHILL96의 프로시듀어는 C++의 함수로 변환된다 중첩 프로시듀어는 중첩 함수로 변환한다. 프로세스는 프로시듀어와 마찬가지로 함수로 변한다. Begin-end 블록은 {… }로 변환된다. 표 1은 프로그램 구조 변환 규칙을 요약한 것이다. (그림 1)은 <표 1>의 변환 규칙을 사용하여 CHILL96의 모듈 구조를 변환한 예이다 (그림 1)에서 모듈 M에서 선언된 변수 a,b는 a_main, b_main으로 이름 변환 규칙에 따라 재명명 되었고 main 프로시듀어는 main 함수로 변환되었다.

CHILL96	C++
M:MODULE *DCL a INT;* *DCL b INT;* *main: proc()* *a := a + b;* *end;* *END*	*int a_main;* *int b_main;* *main()* *{* *a_main = a_main + b_main;* *}*

(그림 1) 모듈 구조 변환 예

(그림 2)는 리젼 구조의 변환 예이다 C++에서는 상호 배제를 위한 리젼 구조가 없으므로 리젼 내의 있는 함수의 시작과 끝에 _enterregion( ), _exitregion( ) 함수 호출을 삽입하여 상호 배제 기능을 수행하도록 하였다.

CHILL96	C++
boolean_semaphore. *REGION* *DCL busy BOOL := FALSE;* *acquire:: PROC().* *…* *END acquire;* *release PROC().* *. .* *END release*	*unsigned boolean_semaphore,* *unsigned char busy_boolean_* *semaphore=FALSE;* *acquire()* *{* *_enterregion (&boolean_* *semaphore) ;* *. .* *_exitregion (&boolean_* *semaphore),* *}* *release()* *{* *_enterregion (&boolean_* *semaphore) ,* *.* *_exitregion (&boolean_* *semaphore),* *}*

(그림 2) 리젼 구조 변환 예

3.2 데이터 선언부의 변환

CHILL96의 데이터 선언부는 C++ 언어와 유사하여 대부분 일대일로 변환된다 CHILL96의 int 모드, 기술된 범위 내의 있는 값들의 집합을 나타내는 range 모드, 수학적 집합을 나타내는 powerset 모드, 비트 스트링을 나타내는 bit 모드는 모두 내부적으로 정수 값으로 표현되므로 C++의 경우 타입으로 변환된다. 논리적 진위값을 가지는 boolean 모드는 unsigned char 타입으로 변환된다 CHILL96의 set 모드는 enumeration 타입으로 변환한다. 임의의 장소를 가르키는 PTR 모드는 C++언어의 'void *'로 변환된다 프로시듀어 장소 값을 저장하여 동적 프로시듀어를 처리할 수 있는 PROC 모드는 함수에 대한 포인터로 변환된다. 문자 스트링 모드 char(n)은 C언어의 문자 배열로 변환된다. CHILL96의 array 모드는 C++의 배열 모드로 그대로 변환되지만 C에서 배열은 색인이 항상 0에서 시작하므로 배열 장소를 접근할 경우에는 반드시 색인 계산을 하여 올바른 장소가 접근되도록 해야한다 구조체를 나타내는 struct 모드는 C++의 struct로 변환되고 가변

구조체를 의미하는 varient struct는 union 모드로 변환된다 참조형 모드 ref는 pointer 형으로 변환된다. 새로운 모드를 나타내는 이름을 정의하는 newmode는 C++ 언어의 typedef로 변환된다. <표 2>는 데이터 선언부의 변환규칙을 요약한 것이다. (그림 3)은 데이터 변환부에 대한 예를 보여준다. 표에서 보는 바와 같이 set 모드는 enumeration 타입으로 변환되었고 newmode 정의는 typedef로 변환되었다. 그림 4는 단순한 모듈 모드를 C++의 클래스로 변환하는 예이다. 모듈 명세 모드(module spec mode)는 클래스 정의로 변환되고 모듈 본체 모드(module body mode)에 정의된 생성자와 소멸자의 구현부는 클래스 멤버 함수로 변환되었다

〈표 2〉 데이터 선언부의 변환 규칙.

CHILL96	C++
char	char
int	int
bool	unsigned char
set	enum
range	int
powerset	int
ptr	void *
proc	pointer to function
bit	int
char(n)	array of chars
array(lower : upper)	array of same length
struct	struct
varient struct	union
ref	pointer
newmode	typedef

CHILL96	C++
m1: MODULE *main: PROC()* *syn a int = 1,* *newmode s1 =* *set(one,two,three);* *newmode x3 =* *array[5:7] int;* *dcl i, res INT;* *dcl j CHAR;* *dcl s s1* *dcl arr array* *[1:2,2:10] int;* *end;* *END;*	*#include <stdio.h>* *void main()* *{* *const int a=1;* *typedef enum {one,two,three} s1;* *typedef int x3[3];* *int i;* *int res;* *char j;* *s1 s;* *int arr[2][9];* *}*

(그림 3) CHILL96의 데이터 선언문을 변환한 예

3.3 실행부의 변환

CHILL96에는 여러 종류의 실행 문들이 존재한다. 기본적인 치환문, 호출문, 제어문, 반복문에서부터 CHILL96의 병행성을 지원하기 위한 START 문, SEND 문, RECEIVE CASE 문 등과 같은 것들이 있다. 임의의 수식의 결과로 나온 값을 치환문이 지정하는 장소에 저장하는 치환문은 C++ 언어에서 그대로 변환된다. 그러나 비트 스트링에 대한 치환문은 C++ 언어에서 지원하지 않으므로 |, &, <<, >>, ~ 와 같은 bitwise 연산자를 이용하여 변환하는 방법을 고안하였다. (그림 5)는 비트스트링 변환 규칙을 정리한 것이다. (그림 5)에서 보는 바와 같이 비트스트링 치환문은 네 가지 유형을 가지며 유형에 따라 변환하는 방법이 다르다 여기서 n은 비트스트링의 길이이다.

CHILL96	C++
Main: MODULE *NEWMODE mm1 =* *MODULE SPEC* *DCL ii INT;* *mm1: SPEC PROC (INT)* *CONSTR* *END;* *mm1: SPEC PROC()* *DESTR END;* *END;* *NEWMODE mm1 =* *MODULE BODY* *mm1: PROC(x INT)* *CONSTR* *iii := x;* *printf("constructor\n");* *END;* *mm1: PROC() DESTR* *printf("destructor\n");* *END;* *END;* *END;*	*#include <stdio.h>* *class mm1_Main* *{* *protected* *int ii;* *public:* *mm1_Main(int);* *~mm1_Main();* *};* *mm1_Main mm1_Main(int x)* *{* *ii = x;* *printf("constructor\n");* *}* *mm1_Main::~mm1_Main()* *{* *printf("destructor\n");* *}*

(그림 4) 모듈 모드를 변환한 예

CHILL96

```
b(i)    := a          // A-pattern
b(i)    :=a(j)        // B-pattern
b(i, j)   := a        // C-pattern
b(i, j)   := a(k, l)  // D-pattern
```

C++

```
{ i = n-1-1; b = ( b &~(1 <<i) ) | a << i; } // A-pattern
{ i = n-1-1, j = n-j-1, b = ( b & ~(1 <<i) ) |
        ( a & 1 << j ? 1 : 0) << i ); } // B-pattern
{ i = n-1-1; j = n-j-1, k = n-k-1, l = n-l-1 ; \ // C-pattern
  b = ( b & ~ (~(~0 << j-1-1) <<j) ) | ( ( ( a>>l ) &
      ~(~0 <<(l-k+1) <<j ) ); }
{ i = n-1-1; j = n-j-1, b = ( b & ~( ~(~0 <<i-j+1) <<j) ) |
    a << i; } // D-pattern
```

(그림 5) 비트스트링 치환문 변환규칙

CHILL96의 제어문 if, case 문은 C++의 if 문과 switch 문으로 그대로 변환되고 반복문 do 문은 for 문으로 변환된다. (그림 6)은 CHILL96의 DO문과 CASE문을 for 문과 switch 문을 사용하여 변환한 예를 보여준다. CHILL96의 병행성을 지원하기 위한 START 문, SEND 문, RECEIVE CASE 문은 실행시간 라이브러리 호출로 변환한다. <표 3>은 실행부의 변환 규칙을 요약한 것이다.

<표 3> 실행부의 변환 규칙

CHILL96	C++
:=	=
if	if
case	switch
do	for
allocate	malloc
procedure call	function call
exit/goto	goto
return	return
signal send	_sendsig( )
receive case	_rcvsigcase( )
process start	_start( )
process stop	_stop( )

CHILL96	C++

m: MODULE
  main: proc()
  DCL score CHAR;
  DO FOR EVER ;
  scanf("%c", >score);
  CASE score OF
  ('A', 'B');
  printf("Good!\n");
  ('C');
  printf("Average!\n");
  ('D','F');
  printf("Poor!\n");
  ELSE
  printf(Invalid!\n);
  ESAC;
  OD;
  END;
END;
	#include <stdio.h>
	void main()
	{
	  char score;
	  while(1) {
	  scanf("%c", &score);
	  switch (score) {
	  case 'A':
	  case 'B'
	  printf("Good!\n);
	  break;
	  case 'C';
	  printf("Average!\n");
	  break;
	  case 'D' .. 'F';
	  printf("Poor!\n);
	  break;
	  default'
	  printf(Invalid\n);
	  }
	  }
	}

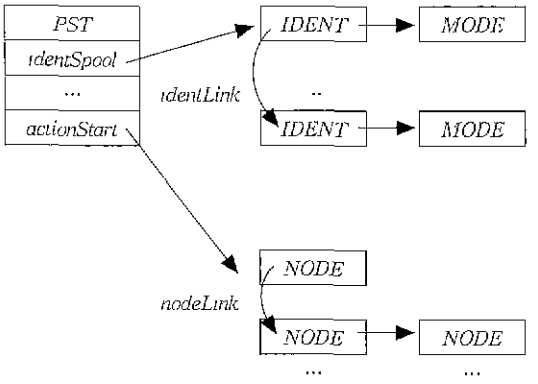
(그림 6) 실행문 변환 예

4. C++ 중간코드를 사용하는 CHILL96 컴파일러의 구현

C++ 중간코드를 사용하는 CHILL96 컴파일러는 다음과 같은 요구 사항에 따라 개발되었다. 첫째, CHILL 96에서 C++로 변환된 프로그램은 읽기 쉬워야 한다. 둘째, 생성된 C++ 프로그램은 효율적이어야 한다. 셋째, 대규모 CHILL96 프로그램도 변환되어야 한다. 넷째, CHILL96컴파일러 자체도 빠르고 이식성이 있어야 한다. 이와 같은 요구사항을 바탕으로 기존의 EM 코드를 사용하던 CHILL 컴파일러의 심볼 테이블과 추상구문트리(AST : Abstract Syntax Tree)[14, 15]를 대폭 개선하여 새롭게 심볼 테이블과 추상구문트리를 설계하였다.

4.1 심볼테이블과 추상구문트리

일반적으로 CHILL프로그램은 여러 개의 블록, 모듈, 리전으로 구성된다. 블럭은 변수의 수명을 제어하며, 모듈과 리전은 변수의 가시성을 제어한다. 모듈 내에는 선언문이나 정의문이 먼저 존재하고 그 다음에 실행문이 존재한다. 컴파일러 내에서 CHILL프로그램은 심볼테이블과 추상구문트리로 표현되어 처리된다. (그림 7)은 단위 프로그램이 어떻게 심볼테이블과 AST로 표현되는지 보여주고 있다. 프로그램 구조를 나타내는 PST에서 identSpool필드는 선언문이나 정의문에서 사용된 명칭을 가리킨다 명칭(IDENT)은 identLink로 연결되어 있으며 자신과 관련된 모드(MODE) 정보를 가질 수 있다 actionStart 필드에서 가리키는 실행문은 노드(NODE)들로 표현되어 AST를 구축한다.

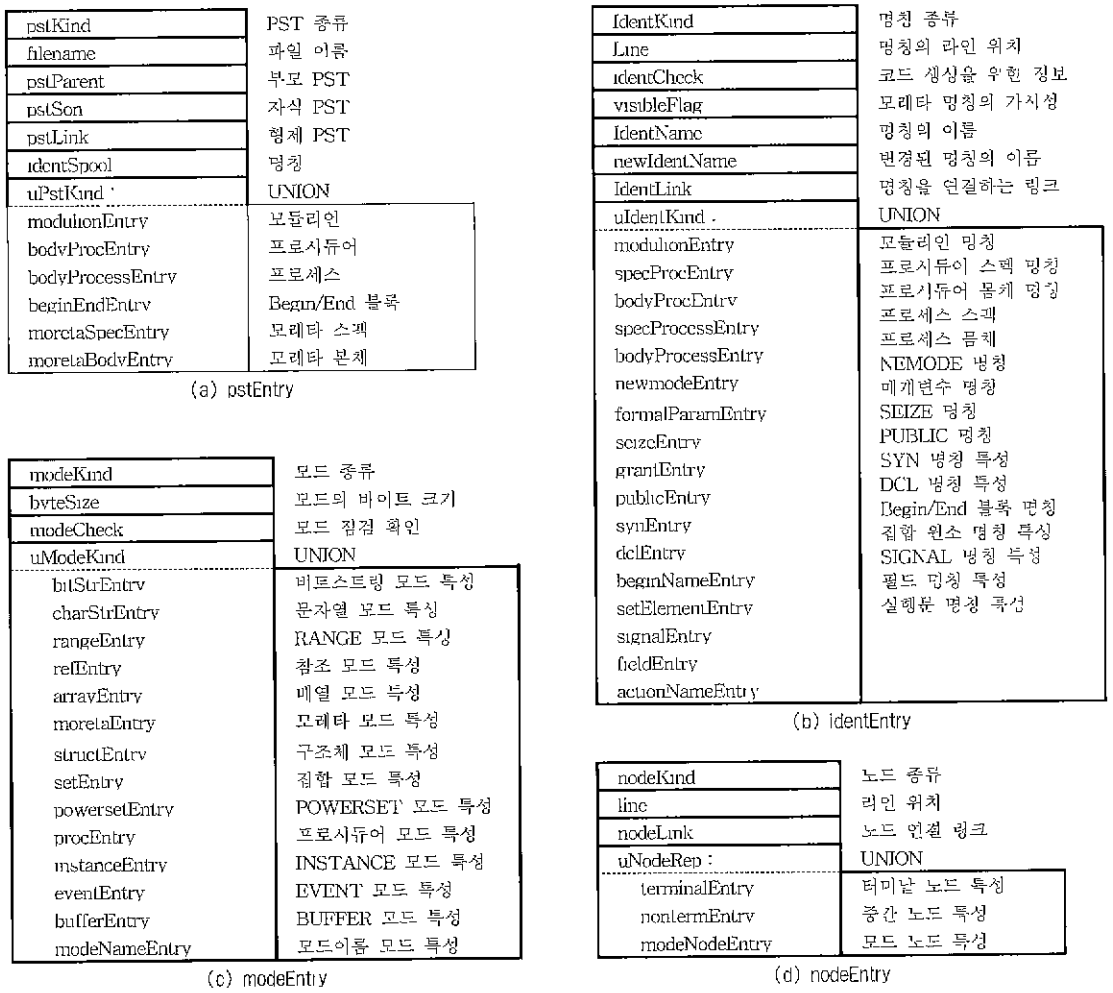


(그림 7) 단위 프로그램에 대한 심볼 테이블과 AST

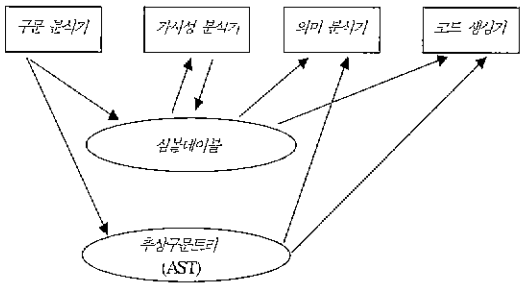
심볼테이블을 구성하는 주요 자료구조는 (그림 8)에 기술되어 있다 프로그램 구조를 표현하기 위한 자료 구조는 pstEntry이다. pstEntry는 PST (Program Structure Tree)종류, PST가 속해있는 파일이름, 부모, 자식, 형제 관계의 PST를 가리키는 포인터와 PST 내의 명칭을 가리키는 포인터 정보를 가지고 있다. 또한 PST의 종류에 따라 각각의 고요한 정보를 가지며 이는 uPStKind 형태의 union 자료형으로 구성되었다. 여기서 실행문을 가지는 프로세스나 프로시듀어 단위 프로그램은 actionStart필드를 정보로 포함한다 명칭을 표현하기 위한 자료 구조는 identEntry이며, 명칭 종류, 라인 위치, 코드 생성과 가시성 제어를 위한 정보, 명칭의 이름, 이름 변환을 위해 개명된 명칭의 이름, 명칭을 연결하는 링크를 정보로 가지고 있다. identEntry는 매우 다양한 종류의 명칭을 가지며, 명칭의 종류에 따라 필요한 정보가 서로 달라 이를 정보를 표현하기 위한 uIdentKind 필드를 가지고 있으며 union 자료형으로 이루어져있다. 명칭 중에서 모드 정보를를 가지는 것은 자신의 모드 정보를 찾을 수 있는 포인터 정보를 uIdentKind속에 기진다. 모드를 표현하기 위한 자료 구조는 modeEntry이며, 모드 종류, 바이트 크기, 모드 점검 확인을 위한 정보, 모드의 종류별로 필요한 정보를 구성한 uModeKind로 구성되어 있다 실행문을 표현하는 기본 구조인 노드 정보는 노드 종류, 라인 위치, 연결 정보를 가진 nodeEntry에 니타난다

4.2 전체구조와 알고리즘

CHILL96 to C++ 컴파일러의 전체 구조는 (그림 9)와 같다



(그림 8) CHILL96 컴파일러의 주요 자료구조



(그림 9) CHILL96 to C++ 컴파일러 전체 구조도

4.2.1 구문 분석기

구문 분석기는 Bell연구소에서 개발한 LALR(1) 파서 생성 시스템인 YACC[13]을 이용하였다. YACC을 이용하여 구문 분석기를 만들기 위한 CHILL96 문법은 "*ochgram.y*"에 정의되어 있다. 그리고 이 문법으로 생성한 구문 분석기는 "*y.tab.c*"화일에 *yyparse*()함수로 구현되어 있다. 다음은 구문 분석기 알고리즘을 요약한 것이다.

1 소스 파일을 읽음
2 PST 생성 함수를 호출하여 단위 프로그램의 pstEntry를 생성하고 pstEntry 간의 관계에 따라 관련 필드를 연결
3. PST내에 명칭이 있으면 명칭 생성 함수를 호출하여 identEntry를 생성하고 identLink로 연결한다. 단위 프로그램의 첫번째 명칭은 pstEntry의 identSpool에서 가리키도록 함
4. 명칭이 모드를 가지면 모드 생성 함수를 호출하여 modeEntry를 생성
5 PST내에 실행문이 존재하면 AST 구축 함수를 호출하여 AST nodeEntry를 만들고 첫번째 실행문은 pstEntry의 actionStart 필드에서 가리키도록 함
6 구문 오류가 없으면 다음 단계인 가시성 분석 단계로 넘어감

4.2.2 가시성 분석기

CHILL96 프로그램에서 모듈과 리젼은 각 이름의 경계 역할을 하기 때문에 모듈과 리젼의 범위를 넘어서 참조될 수 없다. 이 경계를 넘기 위해서는 GRANT와 SEIZE 선언문을 사용하여야 한다. CHILL 컴파일러의 가시성 분석 단계에서는 GRANT/SEIZE 선언문을 처리하여 다른 모듈리언에서 GRANT나 SEIZE된 이름을 참조할 수 있도록 만든다.

한 모듈리언 내에서 GRANT된 이름은 외부 모듈리언에서 사용할 수 있으며, 한 모듈리언에서 SEIZE 선언문을 사용하여 외부 모듈리언에서 선언된 변수를 사용할 수 있다. 이러한 기능이 가능하도록 하기 위해서 가시성 분석기는 심볼 테이블을 재구성해야 한다. 즉, GRANT된 이름은 외부 모듈리언의 이름 리스트에 추가해주어야 하고, SEIZE된 이름은 외부 모듈리언의 이름 리스트에서 찾아서 SEIZE 선언문을 사용한 모듈리언의 이름 리스트에 추가해 주어야 한다. 다음은 가시성 분석기 알고리즘을 요약한 것이다.

1 각 모듈리언을 방문하면서 해당 모듈리언의 GRANT이름을 외부 모듈리언의 이름 리스트에 추가
2 각 모듈리언을 방문하면서 해당 모듈리언의 SEIZE이름 연결 리스트에 나타난 이름을 자신의 모듈리언의 이름 리스트에 추가
3. 각 단계에서 한 모듈리언에 이름을 추가할 때 같은 이름이 충돌하는 지를 검사
4 이름 충돌 오류가 없으면 다음 분석 단계로 넘어감

4.2.3 의미 분석기

의미 분석기는 구문 분석기에 의해서 생성된 심볼 테이블 내의 명칭들 사이에 모드 검사 등의 의미 분석을 수행한다. 객체지향 기능을 지원하기 위해 모레티 빙세/본체 모드 간의 구성 원소 선언과 정의가 일치하는지, 상속 계층간의 일관성이 유지되는지, 추상 모레타 모드들에 대한 의미 점검이 이루어지는지 등과 같은 다양한 의미 분석을 수행한다. 또한 실행문 의미를 분석하기 위해 AST를 순회하면서 객체의 이름 접근에 대한 정의/사용이 올바른지, 객체 간의 연산이 올바르게 되는지, 매개변수의 전달 및 치환문의 모드 일치성이 보장되는지 점검한다. 다음은 의미 분석기 알고리즘을 요약한 것이다

1 프로그램을 구성하는 모든 모듈과 리전에 대한 의미 분석을 구성하는 함수 호출
2. 각 모듈이나 리전에서 정의된 모든 명칭에 대한 의미 분석을 수행하는 함수를 호출하여 관련된 심볼 테이블 정보를 채우고 오류를 발견하면 메시지 출력
3. 명칭의 종류가 프로세스나 프로시듀어이면 블록 특성에 대한 의미분석 수행하고 오류를 발견하면 메시지 출력
4 블록 내에 실행문이 존재하면 실행문 별로 AST 노드를 순회하면서 각각의 의미 심김 함수를 호출하여 각 실행문에 대한 의미 분석 수행
   4.1. 실행문이 수식부를 포함하면 수식부에 대한 의미 분석을 수행하여 심볼테이블 정보를 채움
   4.2. 실행문이 상소부를 포함하면 객체의 이름 접근에 대한 정의/사용이 올바른지, 객체 간의 연산이 올바르게 되는지를 조사
5 명칭이 모드를 가지고 있으면 모드에 관련된 심볼 테이블 정보를 채우고 오류를 발견하면 메시지 출력
6 명칭의 모드가 모레타 모드이면 모레타 빙세/본체 모드 간의 구성 원소 선언과 정의가 일치하는지 조사하고 오류를 발견하면 메시지 출력

7. 명칭이 수식을 가지고 있으면 수식에 대한 의미 분석을 수행하고 심볼테이블 정보를 제공
8 모든 의미분석 단계에서 오류가 없으면 코드 생성 단계로 넘어감

4.2.4 코드 생성기

코드 생성기는 의미 분석 단계에서 정보가 추가된 심볼테이블 정보를 참조하고 추상구문트리를 순회하면서 C++ 코드를 생성한다. 이 부분은 크게 프로그램 구조에 대한 코드 생성, 객체의 정의에 대한 코드 생성, 실행문들에 대한 코드 생성으로 이루어져 있다. 다음은 코드 생성기의 알고리즘을 요약한 것이다.

1 프로그램에 정의된 모든 명칭 이름이 충돌하는 것을 방지하기 위해 이름 변환 함수를 호출하여 모든 명칭에 대하여 이름 변경
2 명칭의 생성 순서를 조정하는 함수를 호출하여 명칭의 정의 순서를 결정하여 재배열함
3 프로그램을 구성하는 모든 모듈과 리젼에 대한 코드 생성을 위해 프로그램 생성 함수를 호출
4. 프로그램 생성함수는 각 모듈이나 리젼에서 정의된 모든 명칭에 대하여 명칭 생성 함수를 호출
5. 명칭 생성함수는 명칭의 종류에 따라 심볼테이블 정보를 참조하여 C++ 코드 생성
6 명칭이 모드를 가지고 있으면 심볼테이블 정보를 참조하여 모드에 대한 C++ 코드를 생성
7 명칭의 종류가 프로세스나 프로시듀어이면 AST를 순회하면서 실행문에 대한 C++ 코드를 생성
  7.1 실행문이 수식부을 포함하면 수식의 종류에 따라 AST수식 노드를 순회하면서 수식부에 대한 C++ 코드 생성
  7.2. 실행문이 장소부를 포함하면 장소의 종류에 따라 AST 장소 노드를 순회하면서 장소부에 대한 C++ 코드 생성
8 명칭의 종류가 클래스 정의이면 심볼테이블에서 모레다 모드 정보를 참조하여 C++ 클래스 코드 생성
9. C++ 코드 생성에 오류가 없으면 GNU C++ 컴파일러를 호출하여 원시 코드 생성

5. 성능 평가

본 논문에서 기술한 CHILL96 컴파일러의 성능을 알아보기 위해 대표적인 프로그램 3가지를 통해 컴파일러 간의 성능을 비교하였다. 성능 평가의 대상 컴파일러는 EM 코드를 중간 코드로 사용하는 기존 CHILL96 컴파일러, CYGNUS CHILL 컴파일러, 그리고 본 논문에서 기술한 C++ 중간 코드를 사용하는 CHILL96 컴파일러이다.

성능측정 대상 프로그램의 대상이 된 프로그램은 quicksort, 수치계산, 함수호출 프로그램이다. quicksort 프로그램은 9999개의 정수 데이터를 정렬하도록 하였고 재귀적 호출이 많이 일어난다. 복잡한 수치 계산 프로그램은 다양한 연산자를 사용하여 루프 내에서 10000번 반복 계산을 수행하도록 하였다. 함수 호출은 크기가 10004 바이트, 20004 바이트인 구조체 2개를 매개변수로 전달하는 함수 호출로 10000번 호출이 일어나도록 하였다.

성능 측정은 다음과 같은 방법으로 수행하였다. 각각의 성능 측정 대상 프로그램을 UNIX시스템에서 실험 대상 컴파일러로 별도의 실행 프로그램으로 생성하였다 이때 C++를 중간 코드로 CHILL Compiler는 GNU C++ 컴파일러의 -O2 옵션을 사용하여 실행코드를 생성하였다 실행 시간 측정의 정확도를 높이기 위해 독립(stand-alone) 워크스테이션에서 timex 명령어를 사용하여 각 실행 파일에 대하여 20회 반복 수행하여 가장 큰 값과 작은 값을 버리고 평균 값(trimmed mean)을 구했다. <표 4>는 실험 결과를 요약한 것이다 표에서 보는 바와 같이 C++ 중간코드를 사용하는 CHILL96 컴파일러는 다른 CHILL컴파일러들에 비해 우수한 성능을 나타냈다. 특히 최적화의 여지가 많은 수치 계산에서는 성능 격차가 더욱 커졌다. 이와 같은 결과가 나타나는 이유는 GNU C++ 컴파일러의 최적화 기능이 우수하기 때문이다.

〈표 4〉 컴파일러 간의 성능 비교 (단위 : sec.)

compiler / program	CHILL96 Compiler (EM)	Cygnus CHILL Compiler	CHILL96 Compiler (C++)
수치 계산	0.31	0.15	0.04
함수 호출	18.08	10.26	10.29
quicksort	8.18	14.60	3.91

6. 결 론

본 논문에서는 C++ 중간코드를 사용하는CHILL96 컴파일러를 설계 및 구현하였다. C++ 코드를 생성하기 위하여 CHILL96에서 C++로의 변환 규칙을 설계하였으며, 설계된 변환 규칙은 프로그램 구조, 데이터 선언부, 실행부로 나누어 정리하고 예제를 통해 설명하였다. CHILL96컴파일러의 구현에서는 구문 분석기, 가시성 제어부, 의미 분석부, 코드 생성기 등의 구성요소별로 간략한 알고리즘을 살펴보았다. 이들 구성요소들은 심볼 테이블과 추상 구문 트리를 이용하여 각각의 기능을 수행한다. 본 논문에서는EM 코드를 사용하던 CHILL 컴파일러의 자료구조와 AST를 버리고 4장에

서 기술한 요구사항에 맞도록 새롭게 설계하였다.

성능 평가 결과 CHILL96 컴파일러는 다른 CHILL 컴파일러에 비하여 상대적으로 우수한 성능을 보여주었다. 성능 향상의 주원인은 GNU C++ 컴파일러의 최적화 기능을 이용했기 때문이다. 이 논문에서 기술된 CHILL96컴파일러는 GNU C++ 컴파일러를 후단부로 사용함으로써 얻는 성능 향상과 이식성의 향상 이외에도 기존에 CHILL로 개발된 많은 통신 소프트웨어들을 C++ 언어로 변환하여, CHILL을 모르는 통신 시스템 개발자들이 변환된 C++ 코드로 통신 소프트웨어를 유지보수 및 확장 가능하도록 하였다. 또한 변환된 C++ 코드는 상용 환경에서 사용되는 소규모 통신 시스템에도 용도에 맞게 수정하여 이식 시킬 수 있으므로 경제적 파급 효과가 클 것으로 생각한다.

참 고 문 헌

[1] CCITT, 'Recommendation Z.200', CCITT High Level Language(CHILL)', COM X-R 34-E, Feb 1992.

[2] ITU-T, 'Recommendation Z.200, CHILL-The ITU-T Programming Language', Sept. 1999.

[3] B. Stroustrup, 'The C++ Programming Language', Addison Wesley, 3rd Edition, 1997.

[4] R. Stallman, 'Using GNU CC - 97R2', Free Software Foundation, 1997.

[5] P. Wegner, "Concepts and Paradigms of Object-Oriented Programming," ACM SIGPLAN OOPS Messenger, Vol.1, No.1, pp.7-87, Aug 1990

[6] A. Yonezawa, 'ABCL : An Object-Oriented Concurrent System-Theory, Language, Programming, Implementation and Application', The MIT Press, 1990.

[7] G. Agha, 'ACTORS A Model of Concurrent Computation in Distributed Systems', The MIT Press, Cambridge, Massachusetts, London, England, 1986.

[8] J. F. H. Winkler et al., "Object CHILL-An Object Oriented Language for Telecom Applications," XIV International Switching Symposium, pp.204-208, Oct. 1992.

[9] C. A. R. Hoare, "Monitors : An Operating System Structuring Concept," CACM, Vol.17, No.10, pp.549-557, Oct 1974.

[10] L. Cardelli et al., "On Understanding Types, Data Abstraction and Polymorphism," ACM Computing Surveys, Vol.17, No.4, pp.471-522, Dec. 1985

[11] J Ichbiah et al, 'Rationale for the Design of the Ada Programming Language', Cambridge University Press, 1991.

[12] G. R. Andrews, 'Concurrent Programming - Principles and Practice', The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.

[13] J. R. Levine et al., 'Lex & Yacc', O'Reilly & Associates, Inc, 1992

[14] A. V. Aho et al., 'Compilers - Principles, Techniques and Tools', Addison Wesley, 1986.

[15] C. N. Fisher, et al., 'Crafting a Compiler With C', The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991

금 창 섭

e-mail : cskeum@etri.re.kr

1992년 서울시립대학교 전산통계학과 졸업(학사)

1994년 서울시립대학교 대학원 전산통계학과 전산전공(이학석사)

1994년~현재 한국전자통신연구원 개방형서비스팀 연구원

관심분야 : 컴파일러, 개방형 네트워크, 소프트웨어 공학, 분산 데이터베이스

이 준 경

e-mail leejk@etri.re.kr

1985년 서강대학교 물리학과 졸업(학사)

1987년 숭실대학교 대학원 전산학과(공학석사)

1997년 숭실대학교 대학원 전산학과(공학박사)

1987년~현재 한국전자통신연구원 통신망기반팀 선임연구원

관심분야 : 프로그래밍 언어, 컴파일러, 객체지향 시스템 소프트웨어

이 동 길

e-mail : dglee@etri.re.kr

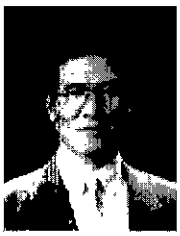
1983년 경북대학교 전자공학과 졸업(학사)

1985년 KAIST 전산학과 (공학석사)

1993년 KAIST 전산학과(공학박사)

1985년~현재 한국전자통신연구원 개방형플랫폼팀장

관심분야 : 프로그래밍 언어, 컴파일러, 소프트웨어 공학



이 병 선

e-mail : bslee@etri.re.kr

1980년 성균관대학교 수학과 졸업(학사)

1982년 동국대학교 전자계산학과 (공학석사)

1982년~현재 한국전자통신연구원 개방형서비스팀장

관심분야 : Fault-tolerant Computing, Component-based Software Engineering, Real-time Systems