응용통계연구 제 **12**권 *1*호, *1999*년, *pp. 29–43*

정적로딩 및 동적로딩을 통한 S-PLUS와 C 언어간의 인터페이스 구현 *

차경준[1] 박영선[2]

요 약

S-PLUS는 통계자료분석 및 모의실험을 행할 때 가장 많이 사용되는 통계패키지 중 하나로서 연산을 수행하는 내장함수(built-in function)와 그래픽 기능을 가지고 있다. 그러나, 이러한 기능이 모든 문제를 해결해 주는 것은 아니며 문제 해결을 위하여 함수를 복합적으로 사용하거나 C 또는 Fortran같은 언어로 구성된 프로그램을 S-PLUS와 연결시켜 사용해야 하는 경우가 발생한다. 본 논문에서는 많은 장점을 가지고 있음에도 불구하고 실제 구현단계에서 여러가지 어려움으로 인하여 널리 쓰이지 못하고 있는 S-PLUS와 C 언어간의 인터페이스 구현에 관한 것으로 정적로딩(Static Loading)과 동적로딩(Dynamic Loading)을 통한 구체적인 인터페이스 실행방법을 보이고 그 예를 실행하였다.

1. 서론

S 언어 계열의 패키지들(이하 S-PLUS로 통일 표기)의 다용성과 특징은 이미 널리 알려져 있으며 통계학 및 공학분야에서 널리 활용되고 있다. 현재 윈도우즈 플랫폼에서 사용되는 S-PLUS for Windows 버전은 운영체제에서 지원되는 그래픽기반을 중심으로 더 한층 강화된 사용자 중심의 환경과 그래픽 기능으로 자료분석, 통계분석, 모델링, 그래픽등의 분야에 그 활용도를 더욱 넓히고 있다.

또한 S-PLUS의 중요한 특징 중 하나는 단순한 통계 어플리케이션의 범주를 뛰어넘는 훌륭한 프로그램의 개발환경이며 프로그래밍 언어라는 점이다. 통계 및 공학관련 내장함수들과 그래픽 툴의 풍부함은 물론이고 사용자 중심의 인터프리터 개발환경으로 빠르게 프로그램을 완성해 낼 수 있게 한다.

S-PLUS는 또한 확장성과 유연성이 뛰어난 개발환경을 가지고 있으며 이는 매우 중요한 S-PLUS의 특징이라 할 수 있다. 컴파일러 환경의 고급언어인 C나 Fortran과의 인터페이스를 통하여 인터프리터 방식의 S-PLUS 단점인 처리속도 문제를 해결할 수 있으며 효율성의 증가를 이룰 수 있다. 또한, 시스템 독립적인 어플리케이션등도 제작 가능하게 되며 S-PLUS의 뛰어난 기능과 기타 고급언어들의 입출력처리 등을 통하여 수준높은 프로그램을 작성할 수 있게 된다. 그러나 상이한 두 언어간의 연결(linking)은 실제 구현단계에서 많은 난점을 가지고 있는 것이 사실이며 이러한 이유 때문에 실제 개발과정에서 혼합 프로그래밍(mixed mode programing)은 기피되어온 실정이다.

* 이 논문은 1998년 한양대학교 교내연구비에 의하여 연구 되었음.

1) (133-791) 서울특별시 성동구 행당동 17번지, 한양대학교 수학과, 교수

2) (133-791) 서울특별시 성동구 행당동 17번지, 한양대학교 수학과, 박사과정

S-PLUS와 관련된 manual에 이러한 인터페이스에 관한 언급은 되어 있으나 난해하게 설명이 되어 있고 주로 unix를 기반으로 언급이 되어 있으며 S-PLUS를 어느정도 완벽하게 이해하고 있는 사용자가 아니면 바로 사용할 수 없는 문제점이 있다. 따라서, 본 논문에서는 PC를 이용한 S-PLUS와 C 언어간의 인터페이스 구현에 대하여 보다 정립된 고찰을 통하여 정적로딩과 동적로딩의 구체적이고도 체계적인 과정과 예를 보여줌으로서 사용자가 보다 쉽게 인터페이스를 구현하는데 도움을 주고자 한다. 이러한 작업을 위하여 S-PLUS for Windows Ver3.3과 팬티엄급 PC를 사용하였다.

2. S-PLUS특징 및 연결의 필요성

S-PLUS는 인터프리터형 언어이기 때문에 한 행씩 입력할 때마다 S-PLUS안에서 해석과 실행이 병행되며 결과가 필요한 경우에는 즉시 출력도 가능케 해 준다. 이러한 인터프리터형 체제는 대화형 환경이므로 즉시 결과를 확인할 수 있고 단순한 문법오류 등도 즉시 체크할 수 있기 때문에 사용자가 원하는 어플리케이션을 빠른시간내에 완성시켜 줄 수 있는 장점이 있다.

그러나, 인터프리터형 언어들의 대부분은 컴파일러형 언어들보다 그 실행속도가 느리다. 따라서, 빠른 계산속도를 요구한다거나 데이터의 양이 많은 경우라면 다른 해결방법을 생각해 보아야 한다. S-PLUS는 자체 컴파일러를 제공하지는 않지만, 대신 다른 고급언어인 C나 Fortran과의 연결을 고려한 내장함수를 제공하고 있다. 따라서, 이 함수를 이용하여 효율적인 다른 컴파일러 언어의 도움을 받아 이러한 문제를 해결 할 수 있다.

S-PLUS는 원래 unix상에서 운영되던 S 언어를 발전시킨 것으로 다른 컴퓨터 프로그래밍 언어들의 특징을 많이 포함하고 있다. 기본적으로는 C언어와 비슷한 구조적 함수 체계와 문법을 지니고 있고 C++이나 LISP같은 언어들의 객체지향적 특성을 이어 받고 있으며 APL에서 볼 수 있는 강력한 벡터 및 배열 처리에 적합한 기능을 가진 복합적인 성격의 언어인 것이다. 또한 위에서 언급하였듯이 사용하기 간편한 BASIC 언어처럼 인터프리터환경을 가진 언어이다.

S-PLUS가 가지고 있는 객체지향적 프로그래밍(object oriented programing)에 관련된 특징을 살펴보면, 기존의 절차형 함수 중심의 프로그래밍과는 달리 각종 객체를 정의하고 그 객체가 취할 수 있는 동작들을 자유롭게 프로그래밍할 수 있게 되어 있으며 코드의 수정이나 재사용 측면에서도 유리하게 설계되어 있다. 각종 클래스(class)와 메소드(method) 또한 정의 및 사용이 가능하므로 C++과 같은 형태의 프로그램 작성도 가능하다.

S-PLUS가 가지고 있는 프로그래밍 환경으로서 큰 장점 중 하나는 뛰어난 확장성과 유연성을 들 수 있다. 다른 컴파일러가 생성한 프로그램도 S-PLUS의 일부로서 함께 작동 시킬 수 있으며 여러가지 운영체제에서도 널리 쓰이는 유연성을 지니고 있다.

그러므로 자체적으로 가지고 있는 내장함수와 사용자 정의 함수들만 가지고도 많은 작업들을 해결 할 수 있지만 계산속도의 증대를 고려한다거나 강력한 입출력 인터페이스를 가진 어플리케이션을 만들고 싶다면 다른 언어와의 연결을 고려하여 보는 것이 타당하리라 생각된다. 따라서, C나 Fortran과 같은 고급 컴파일 언어와의 연결을 위하여 약간의 제

약만 감수한다면 내장함수와 헤더파일등을 갖추고 있는 S-PLUS와의 연결을 위하여 고유의 루틴을 첨가하거나 내장함으로서 더욱 효율성이 뛰어난 작업을 수행 할 수 있을 것이다. 따라서, 3절에서는 S-PLUS와 C의 인터페이스에 관하여 알아 보았다.

3. S-PLUS와 C의 인터페이스

똑 같은 기능을 가진 함수라 하더라도 인터프리터 언어환경인 S-PLUS 고유의 언어로 작성된 프로그램은 C 언어로 작성되고 컴파일 된 프로그램보다 수행속도가 느리기 마련이다. 또한, 수학 및 공학분야에 사용되는 함수의 관점에서 본다면 S-PLUS가 많은 내장함수를 지원하고 있지만 운영체제나 하드웨어접근 또는 강력한 입출력 설계등의 부분에서는 일반적으로 C 언어가 강력한 도구로서의 역할을 한다. 우리가 C 언어와의 연결을 고려하는 이유는 한마디로 실행속도와 효율성의 증가라 할 수 있다.

재귀호출이 많아 속도가 느린 계산 루틴이나 데이터의 크기가 방대한 경우가 연결의 좋은 예가 될 수 있다. C 언어와 인터페이스 함으로서 얻어지는 잇점을 확인 할 수 있는 가장 좋은 경우는 S-PLUS만으로 작성되었으며 계산 부하량이 많은 함수가 이미 존재하고 있는 경우 C로 변환하여 수행시켜보면 눈으로도 그 차이를 실감 할 수 있기 때문이다.

S-PLUS Programmer's Manual UNIX & Windows (1995)에 언급되었듯이 워크스테이션에서 1,000개의 배밀도 고정소수점 연산의 경우 S-PLUS 함수만으로 작성된 프로그램이 약 8초 걸려 작업을 수행하는 것을 컴파일 언어로 재구성하여 수행시킨 경우에는 0.08초만에 수행하는 경우가 좋은 예일 것이다.

그렇다고 모든 연산루틴을 다른 언어로 작성하고 다시 인터페이스해야만 하는 것은 아니다. 우선 데이터의 형태, 메모리 할당등의 문제들에 있어서 실질적인 제한을 받으므로 모든 루틴을 용이하게 C 언어로 작성한다는 것은 어렵고 개발작업이나 디버깅(debugging) 작업시 소요되는 시간은 S-PLUS에 비하여 훨씬 길어 질 것이다. 따라서, 데이터 모드(C 언어에서는 data type에 해당됨)를 자주 바꿔야 한다거나 데이터 모드가 가변적인 프로그램들은 인터페이스 고려시 신중해야 할 것이다. 그리고, 시스템에 따라서 완벽한 호환성을 기대할 수 없는 경우도 있으므로 역시 주의해야 한다. 결론적으로 데이터의 배치 및 재배치, 에러처리, 기본 입출력 처리와 메모리 할당등의 부분은 S-PLUS를 이용하여 수행하고 속도가 현저히 떨어지는 부분을 중심으로 연결등의 개발을 해 나가는 것이 적절한 방법이 될 것이다.

S-PLUS와 다른 종류의 프로그래밍 언어를 인터페이스 하고자 할 때, 각각의 조건에 따라 그 방법과 내용이 달라진다. 먼저 인터페이스 할 언어가 C또는 Fotran인지에 따른 해당 컴파일러의 선택이 선행되어야 하고, 이 때 WATCOM 컴파일러를 사용하는 경우와 그렇지 않은 경우로 나뉘게 된다. S-PLUS는 WATCOM C 언어를 사용하여 제작되었기 때문에 여러모로 WATCOM 컴파일러를 사용하는 것이 좋으나 다른 종류의 컴파일러를 사용하는 경우도 있기 때문이다.

Object 코드형태로 존재하는 컴파일된 코드를 S-PLUS와 연결시키는 방법에는 정적로딩(static loading)과 동적로딩(dynamic loading)의 두 가지 방법이 있다. 또한, 운영체제,

예를들면 unix, windows, dos에 따라 다소간의 실행옵션에 차이를 가져 올 수 있고 아예 동적로딩이 지원되지 않는 운영체제와 하드웨어가 있다. 예를들면, The New S Language (1988)나 Becker 외 2인 (1995)이 언급하였듯이 IBM RS/6000 이나 Apollo와 같은 컴퓨터에서는 CPU와 하드웨어의 문제 때문에 일부 지원하지 않는 함수가 있으므로 동적로딩을 수행할 수 없는 것으로 알려져 있다. 또한 C 프로그램 안에서 S-PLUS 루틴을 부르는 방법과 S-PLUS 프로그램 안에서 C 루틴을 부르는 방법은 서로 많은 차이점을 가지고 있다. 본 논문에서는 C 언어로 컴파일된 루틴을 정적로딩과 동적로딩의 각 방법으로 S-PLUS for Windows Ver 3.3에 적재한 후 적절하게 작성된 S-PLUS 프로그램이 호출하여 연결하는 예들을 보여 줄 것이다. 위에서 설명된 방법들에 대하여 좀더 자세히 알아보면 다음과 같다.

1) 정적로딩(Static Loading)

사용자가 새롭게 작성한 C 프로그램의 컴파일 코드를 포함시켜 새로운 S-PLUS 실행파일을 생성해 낸다. 이 경우에 추가시킬 루틴은 S-PLUS Programmer's Manual Supplement (1995) 와 S-PLUS User's Manual (1995)에 언급되었듯이 반드시 WATCOM Ver 9.5 32bit 컴파일러를 이용하여 컴파일 해야만 새로운 S-PLUS 실행파일의 한 부분으로 미리 자리잡게 된다. 자주 수정을 해야하는 루틴인 경우에는 바람직한 방법이 아니다. S-PLUS 전체를 다시 컴파일하는 경우이므로 새로운 실행파일을 만들어 내는 시간과 저장공간이 필요하며 따라서 루틴을 수행하는 속도도 동적로딩보다 다소 느린면이 있다. 로딩시 LOAD 유틸리티를 사용하며 일반적으로 동적로딩이 불가능한 시스템인 경우나 항상 사용하는 루틴을 새로운 S-PLUS내에 첨가시키고자 할 때 사용한다.

2) 동적로딩(Dynamic Loading)

동적로딩을 설명하기 위하여 우선 MS-WINDOWS와 동적 연결 라이브러리 (Dynamic Link Library: DLL) 의 관계를 알아 보는 것이 도움이 된다. 사실, MS-WINDOWS는 사용자의 어플리케이션을 로드 할 때 사용자의 어플리케이션에서의 함수 호출에 필요한 도입함수들을 연결한다. 그래서 동적연결(일하며서 동시에 연결한다는 의미)이라는 말을 사용한다.

MS-WINDOWS내에는 많은 동적 연결 라이브러리가 있으며 이러한 DLL의 목적 중 한가지는 많은 서로 다른 프로그램들이 사용할 수 있는 메모리 사용의 효율화일 것이다.

예전에 도스상에서 프로그램을 작성한 경우는 라이브러리 파일이 .LIB파일의 형태로 되어 있었다. 따라서, 한글 라이브러리를 만들 때에 .LIB의 형태로 만들어져 실행파일에 포함되었다. 그리고 프로그램이 실행될 때에는 이 라이브러리가 실행파일 안에서 호출되어 사용되었다. 즉, 하나의 프로그램만이 실행되는 도스용 프로그램에서는 큰 문제가 없었다고 할 수 있다. 하지만, 윈도우즈환경에서는 사정이 다르다고 할 수 있다. 극단적인 예로, 한글 라이브러리를 사용한다고 가정할 때 이 라이브러리와 프로그램의 크기를 합하여 0.5Mbyte라고 한다면 이 프로그램을 네 번 실행하였을 때 물론 2Mbyte의 메모리가 필요할 것이다. 하지만 이는 매우 비효율적인 방법이다. 왜냐하면 공통적으로 사용되는 한글 라이브러리가 매 실행마다 메모리에 올라오기 때문이다. 따라서, 이러한 경우에 한글 라이브러

리를 한번만 메모리에 올려 놓고 네 번의 프로그램 실행시 이를 공통적으로 이용할 수 있다면 매우 효율적일 것이다. 이것이 바로 DLL의 개념이라고 할 수 있다. 즉, 고정되어 읽혀지는 것이 아니라 프로그램에서 필요할 때 마다 읽혀질 수 있고 또한 여러 프로그램에서 동시에 사용할 수 있다는 것이다. 이 방법은 매우 효율적이며 윈도우즈 디렉토리에 있는 여러개의 DLL파일들도 프로그램에서 요구할 때만 읽혀지므로 메모리를 효율적으로 사용할 수 있다. (볼랜드 C++ 개발자 바이블, 1994와 내손으로 짜는 윈도우즈, 1995 참조)

S-PLUS에서 사용되는 동적로딩은 사용자의 컴파일된 코드나 동적 연결 라이브러리(Dynamic Link Library: DLL)를 S-PLUS 프로그램이 작동될 때 직접 S-PLUS 안에 로딩하는 방법으로 속도가 빠르며 필요한 루틴만 수정 또는 컴파일 하면 되므로 편리하다는 장점이 있다. WATCOM Ver 9.5 32bit 컴파일러를 사용하여 컴파일된 코드를 로딩하는 경우 dyn.load 함수를 사용하며 DLL 로딩이거나 WATCOM 이외의 컴파일러를 사용하여 컴파일 코드를 얻었으면 dll.load 함수를 사용한다.

4. C 루틴의 정적로딩과 동적로딩의 구현

지금까지 살펴본 것처럼 S-PLUS와 C의 인터페이스를 통하여 처리속도의 향상, 프로그램 효율성의 증대, 부가적 기능의 구현등과 같은 잇점을 기대할 수 있다.

이러한 인터페이스를 위하여 C 언어로 작성된 서브루틴을 S-PLUS내로 불러들여 프로그램을 완성하는 작업순서를 작성해 보면 다음 표 4.1과 같으며, 따라서 본 절에서는 표 4.1에 주어진 각 작업순서를 실질적인 작업환경의 설정, S-PLUS 프로그램 및 C 프로그램의 작성등을 통하여 사용자의 입장에서 보다 자세히 체계적으로 살펴 보았다.

1) 환경설정 및 준비과정

본 연구에서는 S-PLUS와 컴파일러를 운용하는 컴파일 및 실행환경으로 Microsoft Windows 95가 설치된 팬티엄 기종을 이용하였으며 C 컴파일러는 WATCOM Ver 9.5 C를 사용하였다. 따라서, 작업을 위한 환경설정 및 준비과정은 다음과 같다.

(1) S-PLUS for Windows내에 있는 확장 라이브러리를 설치한다.

(2) WATCOM Ver 9.5 C 컴파일러를 설치한다.

(3) 컴파일러가 위치하고 있는 디렉토리들에 대한 정보를 정확하게 알려주어야 하기 때문에 경로(path)에 관한 내용을 적절하게 설정한다. 원래 있던 내용에 아래의 경로를 추가하면 되고 아래의 경우 wat란 디렉토리에 WATCOM 컴파일러가 설치 되어 있다는 의미이다.

SET PATH=% path %; c: \ splus \ cmd;c: \ wat \ bin;c: \ wat \ binb;c: \ wat \ binw

(4) 다음과 같은 환경변수들이 설정되었는지 확인한다. 즉, SPLUS.INI 파일 내에 SHOME과 같은 디렉토리로 환경변수 SHOME을 설정한다. 디폴트 값은 S-PLUS가 설치된 바로 밑의 HOME이라는 디렉토리이다. 컴파일이나 로딩시 경로설정이 적절치 못하면 원하는 결과를 얻을수 없으므로 주의해야 하며 경로설정은 다음과 같다.

SET SHOME=C:\ SPLUS;

SET WATCOM=C: \ WAT;

SET INCLUDE=C: \ WAT \ H;C: \ WAT \ H \ WIN;

표 4.1: 인터페이스 구현 순서

환경설정 및 준비과정	컴파일러와 운영체제에게 적절한 정보(경로나 컴파일 옵션등)를 설정
⇓	
작성하고자 하는 루틴의 설계, 전달인수, 로딩방법 결정	필요한 서브루틴을 설계하고 양쪽 함수에 필요한 인수와 데이터형 결정
⇓	
C 프로그램 작성	만약 프로그램이 C 언어로부터 출발한다면 S-PLUS 프로그램 작성순서와 바꾸어도 됨
⇓	
C 프로그램 컴파일	컴파일 하여 목적파일(object file)을 생성
⇓	
C 루틴의 로딩	정적로딩의 경우 LOAD유틸리티 동적로딩의 경우 함수 dyn.load를 사용하여 해당루틴을 S-PLUS내로 적재
⇓	
C 루틴을 호출할 S-PLUS 프로그램을 작성/실행	C 함수를 이용 적재된 루틴을 호출

2) 작성하고자 하는 루틴의 설계, 전달인수 및 로딩방법 결정

코딩에 들어가기 전에 우선 작성하고자 하는 루틴의 설계는 필수적이다. 따라서, 어떤 기능을 가진 함수로 제작할 것인가를 먼저 결정해야 한다. 특히, S-PLUS내에서 C 함수를 호출 할 때에는 참조에 의한 호출지원, 벡터성분의 길이 전달, 리턴값 처리에 대한 관례등 일정한 제한을 가지고 있으므로 이 점에 주의 해야만 한다.

3) C 프로그램의 작성

프로그램의 기능과 전달인수가 결정되었으면 본격적으로 C 프로그램을 작성하게 된다. 특히, 전달되는 모든 인수들은 포인터 형태의 변수임에 유의해야 한다. S-PLUS는 C 루틴 호출시 주소참조방식(call by reference)에 의한 전달만을 허락하므로 이 점에 유의하면서 C 루틴을 작성해야 하며 전달받는 측의 변수 선언시 모두 포인터 형태로 선언하여야 한다.

즉, S-PLUS내에서 C루틴을 부르는 .C함수의 값은 list이며, 이 때 각 객체들의 끝을 C안에서 적절하게 판단할 수 없기 때문에 객체들의 길이를 넘겨주는 것 또한 필수적이다. 그리고 .C함수를 경유하여 불려지는 C루틴의 값은 포인터 형태이어야만 하고 S-PLUS 모드인 list는 C언어의 void형태로만 통과가 된다. 따라서, S-PLUS에서 호출하는 함수의 형태는 void형태이어야 하며 리턴값을 받아야 하는 루틴에서는 다른 방법을 고려해야 한다. 즉, 포인터를 이용한 간접적인 변수조작을 통하여 리턴값을 설정하고, 다시 되받는 S-PLUS측

에서 참조하는 방법을 이용하면 된다. 전달되는 인수들의 모드는 list 이므로 list 첨자연산자인 [[ ]] 혹은 $를 사용하여 지정하게 된다. 잘못되는 경우 메모리 관련 에러가 발생하여 프로그램이 중단될 가능성을 배제할 수 없다.

다음은 세가지 다른 로딩방법을 보여주기 위하여 작성된 C 프로그램의 예제들이다. [리스트 1]은 1부터 n까지의 합을 구하는 프로그램이고 [리스트 2]는 n번째 피보나치 수를 구하는 프로그램이다. [리스트 3]은 DLL 로딩의 예제에 사용될 프로그램으로서 반지름이 주어졌을 때 원의 면적을 구하는 프로그램이며 마지막으로 [리스트 4]는 [리스트 3]을 위한 module definition 파일로서 16bit DLL을 생성하기 위한 프로그램이다.

[리스트 1 : my _ sum.c]

```
#include <s.h> / * S-PLUS가 제공하는 헤더파일로 INCLUDE 디렉토리내
                        에위치 * /
void my _ sum(x, n, sum) / * S-PLUS가 호출하는 함수는 void형 * /
double *x ; long * n; double *sum ; / * 모두 포인터 형태의 변수선언 * /
{
    long i ;
    *sum = 0 ;
    for (i=0 ; i< *n ; i++)
        if (is _ na(&x[i], DOUBLE)) {
            na _ set(sum, DOUBLE) ; break ;
        } else {
            *sum +=x[i] ;
        }
}
```

[리스트 2 : fib.c]

```
/* 되부름(recursion)을 이용하여 n번째 피보나치 수를 구하는 C 루틴 */
long
fibiter(a, b, c)
long a, b, c ;
{
    if (c <= 0L) return b ;
    else return fibiter(a+b, a, c-1) ;
            /* 종료될 때까지 자기자신을 재귀호출 * /
}
void fib(n)
long *n ;
{
```

```
    *n=fibiter(1, 0, *n) ;
}
```

[리스트 3 : area.c]

```
/* DLL 로딩의 예: 볼랜드 C++ 4.5 검파일러로 16bit DLL생성 * /
#include <windows.h>
#define PI 3.1415926
/* 표준적인 DLL 시작함수 */
int FAR PASCAL LibMain (HNADLE hInstance, Word wDataSeg,
                        WORD wHeapSize, LPSTR lpszCmdLine)
{
        /* if local (moveable) heap was allocated, unlocked it * /
        if (wHeapSize > 0)
            UnlockData (0) ;
        return (1) ;
}
/* DLL exit 함수 */
int FAR PASCAL _ export
WEP (int nParam)
{
        return (1) ;
}
/* 원의 면적을 구하는 루틴 */
void FAR PASCAL _ export
area (double FAR *lpArea, double FAR *lpRadius)
{
        /* calculate area */
        *lpArea = PI * (*lpRadius) * (*lpRadius) ;
}
```

[리스트 4 : area.def]

```
/* area.def module definition file */
/* 볼랜드 C++에서 16bit DLL을 생성하기 위하여 필요한 파일 */
LIBRARY          AREA
DESCRIPTION      'Sample DLL for S-PLUS'
EXETYPEWINDOWS
CODE             PRELOAD MOVEABLE DISCARDABLE
```

```
DATA        PRELOAD MOVEABLE SINGLE
HEAPSIZE    1024
EXPORTS
            WEP     @1      RESIDENTNAME
            area    @2
```

4) C 프로그램 컴파일

프로그램을 컴파일하는 과정은 비교적 쉽다. S-PLUS가 제공하는 COMPILE 유틸리티를 사용하면 적절한 옵션을 자동적으로 설정하면서 컴파일을 실행한다. [리스트 1]과 [리스트 2]에 작성된 두 예제를 아래와 같이 컴파일 하였으며 dll로딩을 위한 area.c 파일은 Borland C++을 이용하여 컴파일 하였다.

```
C:\SPLUS\CMD> complie my_sum.c
C:\SPLUS\CMD> compile fib.c
```

5) 로딩과정

(1) 정적로딩

정적로딩의 경우 컴파일 코드는 반드시 WATCOM으로 컴파일 되어야 한다. 작성된 C 루틴을 컴파일 한 후 정적로딩을 하려면 도스 프롬프트상에서 LOAD 유틸리티를 이용하면 된다. LOAD 유틸리티는 자동적으로 새로운 버전의 사용자 루틴이 첨가된 S-PLUS를 생성해 낸다. 아래의 내용은 피보나치 수를 구하는 루틴인 fib.obj를 정적로딩하는 과정이다.

```
C:\SPLUS\CMD> LOAD fib.obj
```

위와 같이 정적로딩을 하는 경우는 새로운 S-PLUS 프로그램을 만들어내는 것과 마찬가지이므로 컴파일 시간이 다소 걸리며 약 2MB 정도의 공간도 필요로 하게된다. 따라서, 잦은 수정을 요하는 루틴인 경우에는 정적로딩보다는 동적로딩을 하는 것이 훨씬 유리할 것이다. LOAD 유틸리티로 정적로딩을 할 때 특별한 이름을 주지 않은 경우에는 NSPLUS.EXE란 이름의 실행파일이 생성되므로 이 파일을 사용하면 된다. 동적로딩과 비교하여 정적로딩이 편리한 점은 이렇게 실행시 매번 해당 루틴을 로딩하지 않아도 된다는 점이다.

(2) 동적로딩

dyn.load함수를 이용하여 동적로딩을 하고자 하는 컴파일 코드의 이름을 적어주면 바로 S-PLUS는 해당 코드를 적재한다. 이 때 참조하는 디렉토리가 잘못될 경우 에러가 발생되는데 디렉토리를 확인하거나 경로까지 함께 적어주면 된다. 프로그램 수행시나 호출을 받았을때만 사용하는 동적로딩 방법은 정적로딩 방법에 비하여 컴퓨터의 지원을 덜 받게 되며 속도 또한 빠르다는 장점이 있다.

```
dyn.load("c:\\splus\\cmd \\my_sum.obj")
dyn.load("fib.obj")
```

위와 같이 입력하면 동적로딩의 과정은 모두 끝나고 언제든지 S-PLUS 프로그램내에서 호출하기만 하면 된다.

(3) DLL로딩

일반적인 동적로딩과정과 흡사하며 아래와 같이 dll.load함수를 사용하여 실행한다. 다른점은 괄호안에 목적파일(object file) 이름을 쓰는 것이 아니고 아래와 같이 dll 파일 이름과 S-PLUS 함수이름을 적는다는 것이다.

dll.load("area.dll", "area")

6) C 루틴을 호출할 S-PLUS 프로그램의 작성 및 실행

S-PLUS내로 로딩된 함수는 이제 S-PLUS내에서 자유롭게 사용이 가능하며 S-PLUS내에서 로딩된 C 루틴을 호출하고자 할 때에는 .C 라는 내장함수를 사용하게 된다. C 호출함수는 다음과 같은 형식을 지닌다.

.C(NAME, ···, NAOK=F, specialsok=F, pointers=NULL)

함수의 전달인수(parameter)를 살펴보면 다음과 같다. NAME에는 호출하고자 하는 C 루틴의 이름을 스트링 형식으로 기술하면 된다. 작성한 C 루틴의 이름을 인용부호(" ")를 포함하여 입력한다. ···으로 표시된 부분은 서브루틴으로 전달될 인수들의 리스트이다. 반드시 기술된 인수들의 개수와 실제 루틴속의 인수들의 수가 같아야 하며 인수들의 저장모드 또한 적절하게 대응되어야 한다. 사실 S-PLUS가 가지는 데이터 모드와 C 언어가 가지는 데이터 형태는 서로 상이하기 때문에 이것을 적절하게 대응시켜 주지 못하면 예기치 못한 결과를 갖게 된다. 표 4.2를 참조하여 적절한 대응을 시켜 줘야만 원하는 결과를 얻을 수 있다.

표 4.2: S-PLUS와 C의 인터페이스시 데이터의 형태

S-PLUS	C
"logical"	long ⋆
"integer"	long ⋆
"single"	float ⋆
"double"	double ⋆
"charcter"	char ⋆⋆
"complex"	struct{ double re, im}
"list"	void ⋆⋆

앞에서 언급하였듯이, S-PLUS는 C 루틴 호출시 주소 참조방식에 의한 전달만을 허락하기 때문에 전달되는 모든 인수들은 C 측면에서 본다면 모두 포인터 형태의 변수가 된다는 것에 특히 유의해야 한다. 또한 각 객체들의 끝을 C안에서 적절하게 판단할 수가 없기 때문에 객체들의 길이를 넘겨주는 것 또한 필수적이다. 잘못되는 경우 메모리 관련 에러가 발생하고 프로그램이 중단될 가능성이 높다. NAOK옵션이 TRUE로 설정되면 S-PLUS는 NA(Not Available)값들을 그대로 C로 넘겨주며 FALSE로 설정되면 NA값들이 발견될 때 에러를 발생시킨다.

SPECIAL옵션은 숫자가 아닌 특별값들에 대한 처리를 어떻게 할 것인가를 결정하는 옵션이고 TRUE/FALSE로 설정한다. 리턴값의 처리는 전달되는 변수 전체가 list 모드로 전

달되기 때문에 [[n]]의 형태로 .C함수 끝에 기입하여 전달받으면 된다.

5. S-PLUS 프로그램과 실행

본 절에서는 앞의 4절에서 작성된 C 프로그램을 S-PLUS내에서 실행하는 예를 보였다.

다음 예는 벡터량의 합을 C 루틴인 [리스트 1]을 통하여 계산해 내는 내용과 C 루틴을 부르는 S-PLUS 프로그램의 리스트이다. 아래에서 [리스트 5]에 작성된 my.sum함수는 .C함수내에서 인용부호(" ")안에 C 루틴과 동일한 이름인 my.sum을 명시하므로서 적재된 C 루틴을 호출하여 작업을 수행하는 S-PLUS 프로그램이다.

실제로 [리스트 1]에 작성된 C 루틴에서는 입력값, 입력값의 길이 및 출력값과 같은 세부적인 것을 정의해야 한다. 그러나, my.sum함수는 C 루틴에 사용된 void문에 의하여 데이터 모드가 list가 되며 또한 .C함수내에서는 C 루틴으로 전달될 인수들과 실제 C 루틴속의 인수들의 개수와 모드가 같게 대응시켜주게 된다. 따라서, my.sum함수에서는 x값만 전달받으면 입력값에 대한 정보와 출력값을 가질수 있고 작업수행 후 3번째 요소인 sum값을 출력하게 된다.

```
> dyn.load("my_sum.obj")
> x <-(1:10000)
> my.sum(x)
   [1] 50005000
```

[리스트 5 : 합계루틴을 부르는 S-PLUS 코드; my.sum]

```
function(x)
{
        .C ("my_sum",
                as.double(x),
                length(x),
                double(1),
                NAOK = T)[[3]]
        /* LIST의 3번째 요소인 SUM 변수의 값을 리턴 받음 */
}
```

아래의 내용은 [리스트 2]를 이용하여 n번째 피보나치 수를 구하는 S-PLUS 프로그램이다. S-PLUS를 실행시킨 뒤 바로 fib.obj를 동적로딩하고 9번째 피보나치 수를 구해본 것이다.

```
> dyn.load("fib.obj")
> .C ("fib", as.integer(9))
   [[1]] :
   [1] 34
```

마지막으로 아래의 내용은 DLL 로딩을 통하여 원의 면적을 구하는 내용이다.

```
> dll.load("area.dll", "area")
    [1] 1
```

[리스트 6 : S-PLUS내에서 DLL루틴을 호출하는 함수; area]

```
function(Radius)
{
        .C ("area",
                as.double(0),
                as.double(Radius))
}
> area(1)
  [[1]] :
  [1] 3.1415926
  [[2]] :
  [1] 1
```

6. 결론

앞에서 살펴본 것처럼 S-PLUS는 매우 유용한 프로그래밍 환경을 가지고 있으며, 특히 확장성과 유연성면에서 볼 때 그 가능성은 크다고 할 수 있다. 인터프리터 환경이 지니는 속도상의 단점이나 C와 같은 범용언어와의 인터페이스가 자유롭다면 좀더 많은 분야에서 활용될 수 있는 강력한 개발 및 분석 도구가 될 수 있을 것이다.

또한, S-PLUS의 장점으로는 WATCOM 컴파일러가 아닌 컴파일러로 컴파일 했을 경우에도 로딩이 가능하다는 것과 C 프로그램내에서 call_S 함수를 이용하면 S-PLUS의 함수에 접근 할 수 있다는 것이다. S-PLUS를 호출할 C 코드는 컴파일된 형태로 이미 S-PLUS내에 로딩되어 있어야 하며 이렇게 쌍방향 호출이 더욱 짜임새 있는 어플리케이션의 작성을 가능케 할 수 있을 것이다.

로딩에 관하여 S-PLUS version 3.2와 version 3.3을 비교하기 위하여 version 3.2에 관한 manual(1993a, 1993b, 1993c)과 version 3.3에 언급된 내용을 비교하였으나 전혀 다른점을 발견하지 못하였고 정적로딩과 동적로딩에 관하여 표 6.1에 간략히 요약하였다.

결론적으로 계산속도나 효율성 증대 측면에서 C 언어등과의 인터페이스를 고려하여 어플리케이션을 제작한다면 분명 유용한 프로그램을 작성할 수 있을 것이며 또한 개발된 라이브러리들을 공유하여 프로그램의 개발기간 단축 및 프로그램 개발기술의 축적도 도모 할 수 있을 것이다.

표 6.1: 정적로딩과 동적로딩의 비교

구 분	정적로딩 (Static Loading)	동적로딩 (Dynamic Loading)	
		일반적인 동적로딩	DLL로딩
컴파일러	WATCOM 계열만 가능 (WATCOM ver 9.5등)	WATCOM 계열만 가능 (WATCOM ver 9.5등)	기타 컴파일러 (볼랜드 C/MS C) 또는 기타언어들로 제작된 DLL들도 가능
컴 파 일 방 법	COMPILE 유틸리티를 사용(편리)	COMPILE 유틸리티를 사용(편리)	해당 컴파일러를 사용
로딩방법	LOAD 유틸리티로 새로운 S-PLUS실행파일 생성	S-PLUS에서 dyn.load()함수 사용	S-PLUS에서 dll.load() 함수 사용
장 점	ㅇ 가장 간단한 로딩방법 ㅇ컴파일/로딩이 편리 ㅇ 한번 실행하면 해당코드는 다시 적재하지 않아도 됨 ㅇ 같은 종류의 부수적인 컴파일러와 복합사용이 가능	ㅇ 컴파일/로딩이 편리 ㅇ수행속도가 빠르다 ㅇ수정작업이 정적로딩에 비해 간편함 ㅇ S-PLUS 내에서 새로운 코드 사용이 가능	ㅇ 일반적인 동적로딩의 장점포함 ㅇ 컴파일러의 제한이 없음 ㅇ여러가지의 DLL들을 사용할 수 있어 어플리케이션의 완성도를 높일 수 있슴
단 점	ㅇ 사용가능한 컴파일러가 제한됨(only Watcom) ㅇ로딩된 루틴의 수정이 매우 불편함 ㅇ동적로딩에 비해 수행시간이 길다 ㅇ 새로운 코드를 사용하려면 s-plus 내에서 나간 후 다시 시작	ㅇ사용가능한 컴파일러가 제한됨(only Watcom) ㅇ항상 사용하는 루틴 일 경우 매번 로딩하는 불편함이 따름 ㅇ library 함수의 사용이 복잡	ㅇDLL제작시 제한규칙이 많아 작성이 어려움 ㅇ해당 C루틴에서 S-PLUS내부 루틴 호출 불가 ㅇ 컴파일러에 따라 DLL작성에 차이가 있슴

참고문헌

[1] 이 도희(1994). 〈볼랜드 C++ 개발자 바이블〉, 성안당, 서울.

[2] 박 현수(1995). 〈내손으로 짜는 윈도우즈〉, 에스컴, 서울.

[3] *S-Plus Programmer's Manual Supplement Ver3.3 for Windows* (1995), StatSci, A Division of MathSoft Inc., Seattle Washington.

[4] *S-Plus Programmer's Manual Unix & Windows* (1995), StatSci, A Division of MathSoft Inc., Seattle, Washington.

[5] *S-Plus User's Manual Ver3.3 for Windows* (1995), StatSci, A Division of MathSoft Inc., Seattle, Washington.

[6] *S-Plus for Windows Programmer's Manual* (1993a), StatSci, A Division of MathSoft Inc., Seattle, Washington.

[7] *S-Plus for Windows Ver3.2 Supplement* (1994), StatSci, A Division of MathSoft Inc., Seattle, Washington.

[8] *S-Plus for Windows Ver3.2 User's Manual Vol.1* (1993b), StatSci, A Division of MathSoft Inc., Seattle, Washington.

[9] *S-Plus for Windows Ver3.2 User's Manual Vol.2* (1993c), StatSci, A Division of MathSoft Inc., Seattle, Washington.

[1998년 5월 접수, 1998년 12월 최종수정]

Interface between S-PLUS and C using Static Loading and Dynamic Loading *

Kyung-Joon Cha[1] Young-Sun Park [2]

ABSTRACT

S-PLUS is one of the most widely used statistical packages for statistical data analysis and simulation which has its own built-in and graphic functions. However, with these functions, we can not perform all the jobs we need. We, sometimes, need to combine a few built-in functions or need to link S-PLUS and C(or Fortran). The interface between S-PLUS and C has not been widely used because of its difficulties, even though it has some merits. In this paper, we show the interface between S-PLUS and C using both static and dynamic loadings, and show some examples of these.

* The author wishes to acknowledge the financial support of Hanyang University, Korea, made in the program year of 1998.

1) Department of Mathematics, Hanyang University, Seoul, Korea
2) Department of Mathematics, Hanyang University, Seoul, Korea