76,11

3 制御用ストラクチャドプログラミング言語SPLの処理系の特徴

中所武司*，野木兼六*，高藤政雄**，森清三***，加藤木和夫
(日立製作所 *ソフトウェア開発研究所，**日立研究所，***大みか工場) (日立エンジニアリング K.K.)

1．はじめに

制御用計算機システムのアプリケーションソフトウェアの品質向上および生産性向上を目的として，プログラミング言語SPLおよび処理系を開発した。

本言語は，本格的な階層化プログラミングを可能とするための機能を有しており，そのために，その処理系は，従来のFORTRANコンパイラなどとは異なる特徴を有している。

2．処理系から見たSPL言語の特徴

SPL言語の主な特徴として，3項目について述べる。

2.1 プログラムの階層構造

従来のFORTRAN言語などのプログラム構造は図1のような個々のプログラムが独立に作成され，プログラム間の静的な上下関係はない。プログラム間の共通データは個々のプログラム内で宣言する。一方SPLでは，図2のように，共通データ宣言部をプログラムから分離し，そのスコープを明確に表わすために階層構造にした。ここで，共通データのまとまりを環境モジュール，手続きの記述単位を関数，そのあつまりを処理モジュールと呼ぶ。

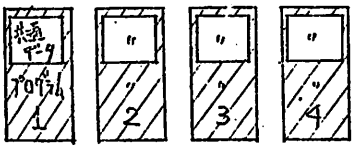

図1．FORTRANのプログラム構成

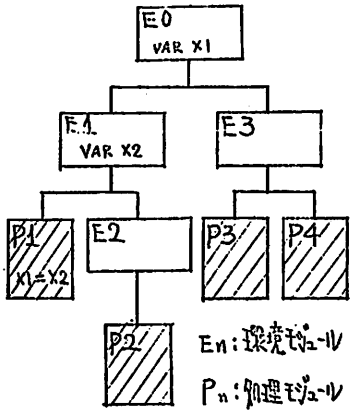

図2．SPLのプログラム構成

2.2 データ型定義機能

新しいデータ型をユーザが定義し，基本データ型と同様に用いることができる。

2.3 コンパイル時機能

コンパイル時機能として，コンパイル時参照関数（OPEN関数），コンパイル時実行文（%文），コンパイル時組込関数がある。

3．処理系の問題と技法

前章の言語の特徴的機能を処理するうえでの処理系の問題とその技法について，各々対応させて述べる。

3.1 階層構造における上位プログラムの情報処理

図2のような，プログラムが階層構造になっているため，下位プログラムのコンパイル時には，その上位に位置するプログラムの情報が必要である。例えば，E0，E1で宣言されたデータX1，X2をP1内で参照している場合，その構文解析，意味解析を行なうためには，X1，X2の型情報などが必要である。

そこで，本コンパイラでは，あらかじめコンパイルした上位プログラム（環境モジュール）の情報を環境モジュールライブラリに登録しておき，下位プログラムのコンパイル時に必要に応じて参照する。

3.2 新データ型変数参照部分の解析処理

データ型定義機能を導入した場合の1つの問題は，その型の詳細情報の参照方

法であるが，ＳＰＬでは，階層化プログラミングでの使用法およびinformation hidingの考えに基づき，新データ型の詳細情報は，それを定義したプログラムあるいはその下位プログラムにおいてのみ参照可能とした。

例えば，図３の例では，Ｐ１ではＳＴＡＣＫ型の詳細情報は参照できず，Ｐ２では可能である。そこで，コンパイラは，文(1)，文(3)は正常処理を行なうが，文(2)はエラーとする。

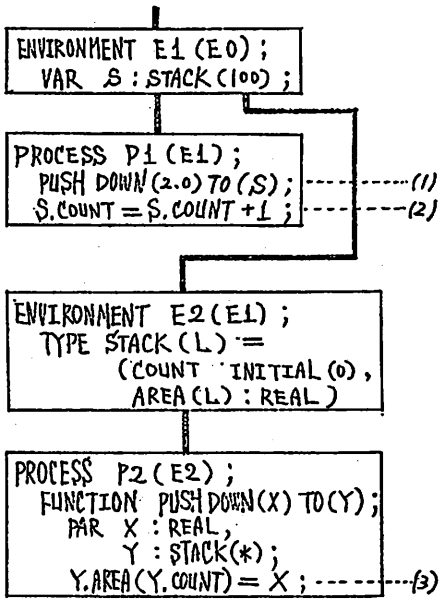

図3．データ型定義機能の使用例

3.3 コンパイル時機能の実行順序

コンパイル時機能の処理としては，ＯＰＥＮ関数を参照部分に展開することと，％文を実行することが主である。ここでの問題は，これらの処理と構文および意味解析との処理順序である。

従来，アセンブリ言語のマクロ機能，ＰＬ/Ｉのコンパイル時機能などは，解析処理の前に処理されていた。これは，プログラムの文法的正しさのチェックが後で行なわれることになり，信頼性の高いプログラム作成上好ましくない。そこで，われわれは，プログラムの信頼性重視の立場から，ＯＰＥＮ関数は，先ず解析処理によって文法的正しさをチェックした後，参照部分に展開するようにした。

4. 処理方式

全体の構成を図４に示すが，処理の流れは，ソースプログラムの種類により異なる。環境モジュールおよびＯＰＥＮ関数などライブラリに登録する処理モジュールは，環境モジュールライブラリ内の上位環境情報を用いて解析処理を行ない，各々のライブラリに登録する。

オブジェクトを出力する処理モジュールの場合は，解析処理後，コンパイル時実行処理フェイズにおいて，ＯＰＥＮ関数の参照があれば，処理グループライブラリからその本体を取り込み，％文およびコンパイル時組込関数があればそれを実行する。そして，最後に，生成フェイズでオブジェクトを出力する。

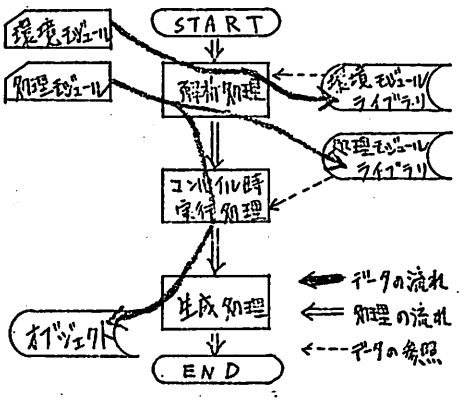

図４．コンパイラの処理方式

5. おわりに

階層構造のプログラムの処理方式上の問題とそれに伴なうＳＰＬの処理系の特徴について述べた。

従来の処理系の設計では，性能の良さが最優先されてきたが，今後は，プログラムの信頼性向上にできるだけ寄与しながら，なおかつ，処理性能の良いものを作る設計技法が重要であると思われる。