5E−5

プログラミング環境における、物理的モジュール分割の制約の緩和に関する研究

広瀬雄二　大駒誠一†
慶應義塾大学理工学部管理工学科‡

1　はじめに

今日プログラミング言語には多種多様のものが存在し、それぞれが種々の分野で利用されている。多くのプログラミング言語は、ある特定の分野・用途・目的を想定して設計されるため、言語仕様には扱えるデータの型とその宣言の仕方、それらデータに対して施すことのできる演算とその演算子などが規定されている。

しかし、近年オブジェクト指向プログラミング(OOP)が叫ばれるようになってから、新しく設計される言語仕様の含むところも大きく変化してきた。つまり、それまで、言語のもつ機能や、扱えるデータタイプなどで差別化するものであった言語仕様は、さらに情報隠蔽の方式や、そのための記法へと管轄範囲を広げて行く過程をたどってきた。

また、情報隠蔽のための仕組みを言語仕様に組み込むことの必要性はコンピュータによって扱う事象の大型化にともない、年々大きくなってきた。それまで一人でプログラムを書いていた時には、問題となることのなかった名前の衝突という事態を常に意識して実装を進めなければならなくなった。そこで開発者集団はこれを回避するために、より情報隠蔽の強力な言語処理系へと環境を変えてきたが、これが可能でない場合は、命名規則を設けたりするなどして対処してきた。

2　モジュール分割の制約

今日の(コンパイル)言語処理系では分割コンパイルの手法を用いて、論理的に分割したモジュールを別々のファイルに記述しコンパイルすることで、それぞれを独立した部品として扱うことが容易になっている。しかし、これは同時にファイル分割が、論理分割と独立でないという制約でもある。

たとえばC言語などの場合、論理構造が図1のような親子関係のものを作成する場合でも、一般的には実際のコンパイルファイルの分割は図2のAあるいは、Bのようなものとするのであるが、実は論理的な参照関係もファイル分割と同じもの、すなわち、独立した同一レベルのモジュールという関係になってしまう。

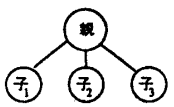

図1: モジュール間の論理的関係

図2: 実際のコンパイルファイルの論理関係

さらに、一つのコンパイルファイルの中に書かれていた、一つの閉じたモジュールを、物理的な分割手続き[1]だけで、それまで保たれていた情報隠蔽を崩すことなく二つのコンパイルファイルに分けることはできない。これはたとえば、資源の乏しい計算機環境でコンパイルする場合などに、能率低下を招く。

さらに前述のように、自由に言語処理系を選択することのできない環境では、プログラマ自ら名前の衝突を回避するため、命名規則に縛られ続けることとなる。

3　スコーププロセッサ

前記の問題点を回避するため、プログラミング言語に依存しない部分でプログラム(群)の論理単位を記述するキーワードと、物理分割時の制約条件を示す記法を導入する(図3)。

このモジュール段落記法をソースプログラムに埋め込み、スコーププロセッサでもとの論理設計の可視性を失うことなく、実際のコンパイルファイルを作成する。こうすることで、情報隠蔽の弱い言語処理系を使う時も、特別な命名規則の束縛を受けることなくプログラムの作成が行える。また、一つのことに関する記述を複数箇所(ファイル)に散在させ、いずれかの部分の更新し忘れによる矛盾の発生なども未然に防ぐことができる。

4　対象言語の前提

プログラム中での識別子の宣言と参照を抽出する必要があるので、対象とする言語は次の条件を満たすものとする。

- スタティックスコープ
- 十分な長さの識別子を扱うことができる

[1]適当な関数の切れ目でファイルを二分する程度の手続き

A Study on Reducing the Restriction of Physical Module Dividing in Programming Environment
†Yuuji HIROSE, Seiichi OKOMA
‡Keio University

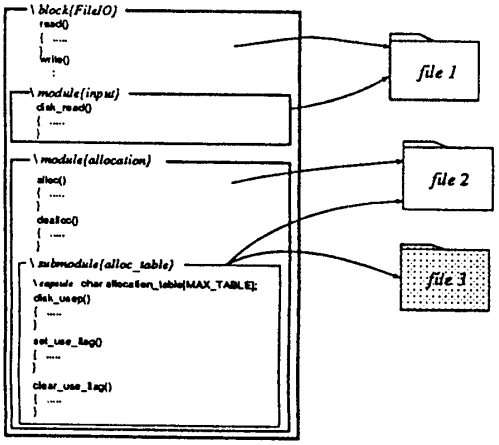

図 3: モジュール段落記法とその分割

- 明示的な変数宣言が必要

本プロセッサは、コンパイラに渡す前の段階で可視範囲を判定するので、ダイナミックスコープの言語は対象から外した。また、可視範囲外からの識別子の参照を、識別子の名前を変える事により予防するので識別子の有効文字数が数文字程度の処理系では実装が困難であるのでこれも除外し、さらに変数の宣言場所を確実に把握する必要があるので、暗黙の変数宣言を禁止した。

5 設計

Cのように比較的情報隠蔽度の弱いものから、Modula-3のように厳格に記述できる言語までを対象とすることを考え、モジュール段落記法を用いて記述するスコープルールの仕様は基本的に次のようなものとした。

- 親・兄弟・子供の論理階層に属するシンボルの参照は許す
- 二世代以上子供の階層に属するシンボルの参照は不可

また、本スコープルールで「可視」と判定する参照関係が実際の言語処理系で本当に参照可能なものであるかどうかは関知せず、その処理系のスコープルールに任せることとする。またこれは、必要以上に対象言語のスコープルールに介入すると、本システムの透明感が減少し、かえってプログラム作成効率を落しかねないと言う配慮でもある。

以上のルールにしたがって、ソースプログラム中の識別子の可視性を判定し、次のような書換えを行う。

1. 埋め込まれたモジュール管理記法から現在の論理的段落位置を把握する。
2. 識別子の宣言箇所と参照箇所を抽出し、それらがどの論理的段落位置に属しているかを調べる。
3. 参照された識別子と同名の宣言が可視範囲にあれば、それらを一意に定まる名前に変更し対応づける。
4. もし参照している識別子が、モジュールのどこにも見つからない場合は、ライブラリ関数、あるいは担当モジュール外のグローバルな識別子の参照とみなし何ら変更を加えない。

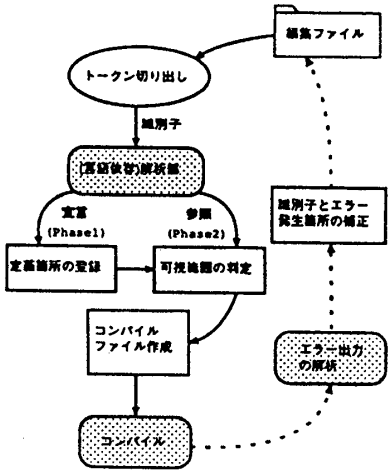

図 4: スコーププロセッサの処理の流れ

これを利用したコンパイル手順は図 4のようなものとなる。図中網かけでしるした部分は、対象言語、およびその処理系に依存する部分である。

また、破線で示した流れの部分では、コンパイラが出力したエラーメッセージを解析し、本来の編集ファイルの誤りのあった箇所との対応を取る。この部分は、デバッグ箇所の整合性を保つために不可欠な部分であり、プログラミング (デバッギング) の効率上非常に重要な位置を占めるが、現段階では触れていない。

6 おわりに

現在までのところ、言語非依存のスコープ処理部と、C言語に依存した解析部分からなる試作システムでの実験を試み、これまでC言語でコンパイルファイルを作成する時に生じていた上記の諸問題は、本プロセッサが自動的に解決し、プログラマの負担を軽減することが分かった。さらに対象言語を拡大していくことで、既存の言語での共同プログラミングの効率が改善できよう。

また、今後設計する言語の仕様は、言語の用途を左右する機能と、言語の使用規模を左右するスコープルールを独立性の高いものにすることで、言語選択がプログラムの作成規模などに依存するということが避けられる。

参考文献

[1] Ellis Horowitz; FUNDAMENTALS OF PROGRAMMING LANGUAGES; Computer Science Press; 1983

[2] Samuel P. Harbison; MODULA-3; PRENTICE HALL; 1991