2G－3

OS／omicronにおける言語Cインクリメンタルパーザの基本設計

玉木裕二，森田雅夫，並木美太郎，高橋延匡
東京農工大学　工学部　電子情報工学科　情報工学講座

1．はじめに

当研究室では，システムやアプリケーションの開発に，主に言語Cを使っている．また，OS／omicron上で，当研究室で設計・開発した言語CコンパイラCAT [2] が存在する．一方，プログラムの入力手段として，プログラムの編集だけでなく，論文や仕様書などの文書やデータの作成などを行う汎用のテキストエディタのALTHEA [3] を開発し，使用している．しかし，汎用であるがゆえ，括弧の対応付けもプログラマが自分で確認しなくてはならない．編集の対象を特定のプログラミング言語に限定すれば，専用のエディタ（構造化エディタ）を開発するアプローチがある．すなわち，プログラムを構文規則に対応した構造を持つものとして扱う．

本研究の目標は，言語Cを対象とした構文要素を意識したエディタを開発し，プログラミング支援環境を実現することである．本稿では，そのための手段としての言語Cインクリメンタルパーザと，それによって実現されるプログラミング支援環境について述べる．

2．インクリメンタルパーザの開発目的

関数・データ仕様書ができていて，その上でコーディングを行ったとしても，最初にコンパイラにかけたとき，一つの文法エラーもなくオブジェクトコードが生成されるのはまれである．多くの場合，プログラム中には文法エラーが存在する．その原因として，タイプミスや，プログラムを書いた人間の勘違いなど多くの原因があげられる．経験から，文法エラーをすべて取り除くためには，エディタでプログラムを書き，それをコンパイルしてエラーがあれば，再度エディタを起動してプログラムを修正するという作業を繰返し行わなければならない．

また，文法エラーをすべて取り除き，ロードモジュールが生成できたとしても，それを実行させたとき期待したとおりの結果が得られないときがある．その場合は，再びエディタを起動してプログラムの修正を行うことになる．

例えば，int i；と宣言したところをlong i；と変更すると，このiを参照している箇所もそれに対応した変更を加えなければならない．このように，プログラム中の一箇所を修正すると，その影響が複数の他の箇所に及ぶことが多い．プログラムのサイズが大きくなればなるほど，それを把握するのは困難になってくる．

インクリメンタルパーザをエディタの中に埋め込み，それをエディタ上からコマンドとして呼び出せるようにすることにより，上述の問題点を解決する，次のようなプログラミング支援環境を実現できると考える．

(1) エラーの指摘

エディタ上でプログラムを入力した後，バインドされたキーを打鍵することによりインクリメンタルパーザが呼び出され，構文解析を行う．このとき，文法エラーを発見すると，適切なエラーメッセージを出力するとともに，エラーのあった箇所にカーソルを移動させ，編集モードに戻る．ユーザは，指摘されたエラーをその場で修正し，再度バインドされたキーを打鍵することにより構文解析を再開させる．

(2) 参照関係の指摘

プログラム中のある箇所に変更を加えたとき，その波及状態をシステムで管理することにより，影響の及ぶ箇所を，カーソルの移動とメッセージを伴って，ユーザに指摘する．ユーザは必要に応じてその箇所を修正する．

(3) 関数・データ仕様書の自動生成

設計の段階で作成した仕様書を見れば，その詳細を知ることができるが，その仕様書が最新のものでない場合，例えば，仕様を変更したが仕様書の更新をしなかった場合は，ソースプログラムから抽出した情報の方が信頼性が高い．ソースプログラムを解析して得られた情報から仕様書を自動生成すれば，それが最新の仕様書となる．

3．構文主導的なアプローチ

上述のようなプログラミング支援環境を実現するために，属性文法を用いた構文エディタを開発しようというアプローチもある．しかし，本研究では，次に述べる理由により，そのようなアプローチは採用しない．

(1) 構文解析することによって得られた情報を，他のシステムの入力として使用することができる．例えば，仕様書自動生成システムなど，構文解析の機能が必要なシステムに対して，インタフェースを提供できる．

(2) 将来的に，言語Cインタプリタへの発展を意図している．

4．インクリメンタルパーザの設計方針

(1) インタラクティブな構文解析

インクリメンタルに構文解析を行うためには，ソースプログラム中の任意の構文要素を，構文解析の対象として指定できなければならない．これを第一の目標とする．

(2) プリプロセス

字句解析の段階でプリプロセッサが行うべき処理も行う．このため，解析結果には，言語Cの構文要素だけでなく，プリプロセッサで処理する＃で始まるコンパイラ制御行の情報も含める．これにより，＃ifdefのようなコンパイラ制御指令による条件分岐も解析結果に反映できる．

A Design of Incremental Parser of Language C on OS／omicron
Yuji TAMAKI , Masao MORITA , Mitarou NAMIKI , Nobumasa TAKAHASHI
Tokyo University of Agriculture and Technology

(3) 解析結果の出力形式

本パーザが生成した解析結果を，仕様書作成システムなどのシステムの入力として用いることを想定して，ファイルへの出力フォーマットを設定する．

(4) 中間コードの生成は行わない

本パーザは上述の目的のために開発するものであり，中間コードの生成を目的とはしない．したがって，初版では中間コードの生成は行わないが，その拡張性は残しておく．

5．インクリメンタルパーザの実現方針

5．1 解析結果の管理

基本的に，言語Cの構文規則によって定義される構造にしたがって，木を生成していく方針であるが，内部の処理が簡単にできるように，これより簡単な構造を持つ木を生成する．これを具体的に説明するために，図1，図2を示す．図1はサンプルプログラム，図2は図1を解析した結果を表す．

図2において，それぞれのセルは対応するソースの始点と終点の情報を持たせる．これにより，ソースと解析結果の対応がとれ，ソースの解析済みの箇所と未解析の箇所が管理できる．

5．2 表の管理

本パーザが内部で生成する表は，単に構文解析に必要な情報のみを登録せず，スコープごとの個々の識別子や型の参照関係も要素として登録する．これにより，前述の参照関係の指摘を行えるようになる．

なお，パーザは [2] のコンパイラを流用する．

```
extern int printf(), print() ;

main(argc, argv)
int argc ;
char *argv[] ;
{
        int i ;

        i = argc ;
        while(i > 0) {
                print(argv[argc - i]) ;
                i-- ;
        }
}

int print(s)
char *s ;
{
        printf("%s¥n", s) ;
}
```

図1　サンプルプログラム

5．3 インクリメンタルな構文解析

変更が加えられた箇所に対して，再度構文解析する場合，前回の解析によって生成した木をできるだけ保持しようとしながら，解析を行う．これにより，既に解析済みな箇所を，再度解析し直すという冗長な手続きを省くことができる．

6．おわりに

本稿では，言語Cインクリメンタルパーザの目的と，その設計，開発方針を述べた．今後これらの設計をもとに開発し，最終目標であるプログラミング支援環境の実現を目指す予定である．

参考文献

[1] 鈴木茂夫，小林伸行，田中泰夫，中川正樹，高橋延匡：“OS／omicronにおける日本語プログラミング環境”，情報処理学会論文誌，vol．30，No．1，1989．1

[2] 並木美太郎，屋代寛，田中泰夫，篠田佳博，中川正樹，高橋延匡：“OS／omircon用言語C処理系Catのソフトウェア工学的見地からの方式設計”，電子情報通信学会論文誌D，vol．J71-D，No．4，1988．4

[3] 小林伸行，並木美太郎，高橋延匡：“OS／omicron上の日本語エディタ”，情報処理学会第38回全国大会，1988．3

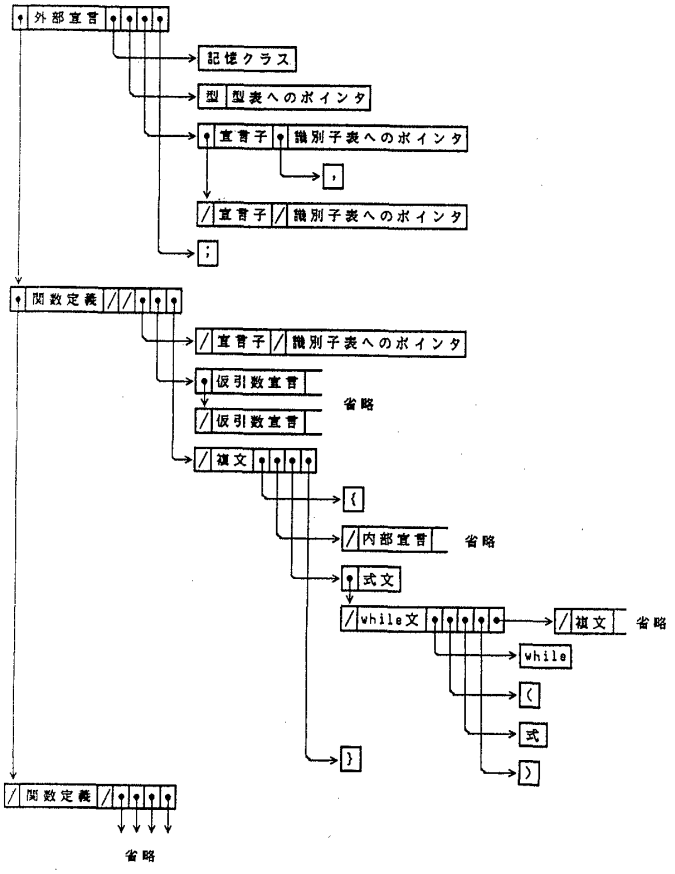

図2　解析結果の例