3L-3

C 言語プログラミングにおけるフォーマット特性の抽出

中島 正登　伊藤 一成　Martin J. DÜRST
青山学院大学理工学部

1　はじめに

現在のプログラミング教育の授業において，最も重要視されているのは意図通りに可動するかという点であり，可読性という点については教わる機会が少ない．そこで昨年度より我々は，本学の授業で主に学習する C 言語に対して，記法の不規則性を指摘するシステムの構築を開始した．これまでにプログラムパターンを選択し，パターンの適合性を調べ，違反する箇所を検出する研究 [1] が行われている．

しかし，我々はより低いレベルのスタイルに注目し，規則性を指摘する．C 言語の一般的な記述スタイルには K&R スタイル [2] や GNU スタイル [3] など様々な方式があるが，完全に定まった記述スタイルはないため，学習者にはある程度自由なスタイルでコーディングさせるのが本研究の目的である．

このシステムを構築するに当たって，bison[4] を利用する．そこで，プログラムを診断するために，スペースもトークンとして扱えるよう拡張する必要がある．LALR(1) 構文解析を使うと，解析する際に 1 つだけ先のトークンを読み込むことで文法規則の判定を行う．空白類を解析の対象として扱うと，文法の多くのところで先読みトークンが空白類を捕らえるため，次にどの状態に遷移すべきか判断できなくなってしまい，解析が困難である．そこで，昨年度池谷 [5] が，空白を読み込んでも文法規則に基づいた解析ができるよう拡張した．解決方法として，bison[4] の ver2.1 以降で使用可能である GLR 構文解析 [6] を利用した．

本稿は，この解析機能を利用し，C 言語におけるスペース，改行の使い方など学習者によって差が出やすい要素において，その統計量からフォーマットの規則性を抽出する方法について提案する．

2　規則性の抽出

診断する箇所を決定するために，Web ページ，本などから C プログラムの書かれている箇所を列挙した．その中から，同じ構文が書かれていてもインデント，改行などの使われ方が異なる箇所を抽出し診断するプログラムを構築する．本章では現段階で診断を行っている 13 項目のうち，特に学生の間違いが起こりやすいと思われる箇所をいくつか例示しながら規則性の抽出方法を紹介する．

2.1　演算子の前後のスペース

学生は計算式の優先度をあまり意識しないため，乱雑に計算式を書くことがある．よって，演算子が見られた際に前後のスペースが計算の優先度に応じて適切に使用されているか診断を行う．前提条件として演算子前後のスペースの数が等しくない場合には可読性が低下するので，1 つの演算子前後のスペースが等間隔で使用されていなかった場合に警告を表示する．

2.1.1　優先度

演算子の優先度によって前後のスペースの数が異なる書式がある．その際，次のように優先度の低い演算子前後のスペースが，優先度の高い演算子前後のスペースより小さくなっていると，可読性の低い計算式になる．

```
a = b+c * d;
```

次のように優先度が適切に順守されていると，可読性は向上するので，優先度の順守されていない計算式には警告を表示する．

```
a = b + c*d;
```

2.1.2　ノードの深さによる優先度

優先度について検討するために構文木におけるノードの深さに注目して診断を行っていく．例として，加算より乗算のほうが優先度が高く，ノードも深くなっていることが分かる．よってノードの深い演算子前後のスペースの方がノードの浅い演算子前後のスペースより小さければ問題ないが，反対の場合には警告を表示する．

2.1.3　式中の改行

式が長くなった場合，変数名が長い場合などに式を複数行にわたって書くことがある．その際，次のように 1 行目と 2 行目の演算子がそろっていない場合，可読性が低下する．

```
answer = variable1 + variable2
    + variable3 + variable4;
```

次のように 1 行目と 2 行目の演算子が揃っていれば，可読性は向上するので，演算子の上下が揃ってない箇所に警告を表示する．

```
answer = variable1 + variable2
       + variable3 + variable4;
```

2.2　規則性の抽出方法

本稿で規則性を抽出するために主に用いている手法を，コンマ前後のスペースの診断方法を例として挙げて説明する．

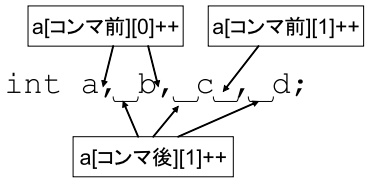
2.2.1　診断部分

診断を行う箇所について，スペースの数などの統計を取り，その結果を保存する配列を何行何列確保すればよいか思慮する．行数は，コンマの前後のスペースの扱いを診断する場合，コンマの前と後で 2 行分確保する．列は使用されることが予想されるスペースの数より余裕を持ち，若干大きめに確保する．ここでは 0 個から 19 個のスペースを想定したため，20 列分確保し，`a[2][20]` という配列で思考する．

Extraction of Format Characteristics in C Programs
Masato NAKAZIMA, Kazunari ITO and Martin J. DÜRST
College of Science and Engineering, Aoyama Gakuin University
5–10–1 Fuchinobe, Sagamihara, Kanagawa 229-8558, Japan
masato@sw.it.aoyama.ac.jp, {kaz, duerst}@it.aoyama.ac.jp

2.2.2 統計

スペース数とその個数を要素とする二次元配列を用意し，プログラムのスペース数を走査しながら，用意した配列に格納していく．



2.2.3 規則性の抽出

配列の中で一番大きな値が格納されている列をそれぞれの行から探索する．次に探索する配列 a[2][20] を例示する．

a[x][y]

x \ y	0	1	2	3	4	…	19
コンマ前	2	1	0	0	0	…	0
コンマ後	0	3	0	0	0	…	0

コンマの前にはスペース 0 個の場合が 2 回で一番多く，コンマの後にはスペース 1 個の場合が 3 回で一番多いことが導き出せる．これを作成者の規則性として考え，これに準拠しない箇所に警告を表示する．つまり，この場合はコンマの前にスペースが 1 個の場合が規則性に矛盾するので，ここに警告を表示する．

2.3 改行の有無の診断

構文によっては，診断箇所の改行の有無を考慮しなくてはならない．例として中括弧が診断箇所の場合，図 1 のように中括弧前の改行の有無 2 通り考えられるので，配列を 2 行確保する．列は 2.2 節と同様に 20 列分確保し，b[2][20] という配列で思考する．そして，改行がある場合には改行後のインデントと改行前のインデントの差を，改行が無い場合は中括弧前のスペースの数をそれぞれ配列に格納する．ここで，次のように列の合計値を求めることで改行の有無どちらが多いかが分かる．

b[x][y]

x \ y	0	1	2	3	4	…	19	合計
改行有	1	3	0	0	0	…	0	4
改行無	1	0	0	0	0	…	0	1

今回は中括弧の前に改行があり，かつインデントが空白 1 つ分あがっていない中括弧の場所に警告を表示する．

2.4 その他の要素

2.2 節で用いた手法を主に使用し，診断を行う．診断を行う箇所の例としては，列 switch 文中の case のインデントと switch とのインデント，for 文条件式のセミコロン前後のスペースなどが挙げられる．

改行あり	改行なし

void main() ↓
△△{
int a;
}
	void main()△{
	int a;
	}

（△…スペース 1 個分）

図 1: 中括弧の改行

3 プリプロセッサへの対応

C 言語では，プリプロセッサを通すと，ヘッダーファイルからの読み込みやシンボルの置換，マクロの展開などを行う．また，本稿で最も注目しているスペース数個を一つにまとめる処理を行うものもある．そのため，学習者個人が作成したプログラム以外の記述も診断の対象になるので，学習者個人のプログラムのみの診断は不可能である．

一般学生のプログラムにおいては構文マクロを使用する機会が少ないので，プリプロセッサ前でも C 言語の文法に合致している．しかし，C 言語の構文解析において typedef で定義された型を把握しないと構文解析が不可能である．そのため#include で定義された型情報を別処理で取得する．そして診断するファイルを読み込む際に#include で定義されたファイルを見つけたら別処理を行ったファイルを読み込み，情報を取得することで解決した．

4 まとめと今後の展望

本稿では，授業で使用されるような基本的な文法についての記述スタイルの診断について提案した．大きな課題として，for，if のような基本的なものだけでなく，学生のようなプログラム初心者はあまり使用しないライブラリ関数などへの対応も検討の必要がある．また，実際に学生が利用可能になるように，Web 上へ展開，またそれに応じた診断結果の出力方法の検討などが列挙される．

参考文献

[1] 関本理佳, 海尻賢二. プログラミングスタイルの診断システムの構築. 教育システム情報学会誌, Vol. 17, No. 1, 2000.

[2] B. W. Kernighan and D. M. Ritchie. プログラミング言語 C. 共立出版株式会社, 1989. 石田 晴久 訳.

[3] Gnu coding standards, December 25 2005. http://www.gnu.org/prep/standards/ 日本語: http://www.sra.co.jp/wingnut/standards/standards-ja.html.

[4] Bison Manual, December 25 2004. http://www.gnu.org/software/bison/manual/.

[5] 池谷武, 伊藤一成, Martin J. Dürst. プログラミング教育におけるフォーマット診断の提案. 第 68 回情報処理学会全国大会, 2006.

[6] Masaru Tomita. An Efficient Augmented-Context-Free Parsing Algorithm. *Computational Linguistics*, Vol. 13, No. 1-2, pp. 31–46, 1987.