調査項目の拡張しやすさを考慮したソースコード解析システムの構築

香川大学創造工学部創造工学科 情報システム・セキュリティコース **卒　業　論　文**	
卒業年度	令和4年度 (2022年度)
指導教員	香川 考司

香川大学創造工学部創造工学科
情報システム・セキュリティコース

小方 亮人

令和5年2月9日

Implementation of a Source Code Analysis System Considering the Ease of Expansion of Survey Items.

Abstract A novice programmer who has just started learning programming often spends a lot of time fixing errors because she cannot understand the meaning of errors and warnings. By outputting easy-to-understand error messages for common mistakes made by novice users, it is possible for them to understand which line of the source code is wrong and how to correct errors efficiently. Also, it is required that the system can easily include new investigated items. Therefore, in this study, we have constructed a system to point out incorrect locations on source code input on a web-based system. The language-c-quote library of Haskell is used for source code analysis and it is expected to that its survey items can be easily expanded. By clearly displaying the line number and how to correct errors, it helps novice learner's programming learning. For the evaluation, we asked members in our laboratory to use the system and to answer a questionnaire. Based on the results of the questionnaire, it is considered useful as a learning support system. On the other hand, many issues were found, such as the lack of items that could be investigated for errors and the separation of the input screen for the source code and the page for outputting the analysis result.

あらまし プログラミングを学び始めたばかりの初学者は、コンパイル時のエラーや警告が何を意味しているのかが分からずエラーの修正に時間をかけてしまうことが多々ある。初学者によくある間違いに対して分かりやすいエラーメッセージを出力することによりソースコードの何行目がどう間違っているのかが理解でき、効率的に修正を行うことができる。また、新しく調査したいエラー項目ができた場合にシステムに組み込みやすいことが求められる。そこで本研究では、Web ベース上で入力されたソースコードに対して間違っている内容を指摘するシステムを構築した。ソースコードの解析には Haskell の language-c-quote を用い、調査項目の拡張性が期待できる。エラーのある箇所の行番号や修正方針を分かりやすく表示することで初学者のプログラミング学習を支援する。研究室の学生に実際に使用して評価をしてもらい、その結果からエラー箇所や内容の表示は学習支援システムとして有用であると考えられる。一方で調査できるエラー項目が少ない点、ソースコードの入力画面と解析結果の出力を行うページが分かれている点といった課題点が多く残っていることも分かった。

キーワード Haskell, 構文解析, プログラミング学習支援, ソースコード解析

目次

第 1 章

はじめに

1.1 研究背景

大学で情報系の専攻に進むとプログラミングを学習する機会が訪れるが、高校生までにプログラミングを行ったことがある経験者は少なく、大学に入ってから初めて学習するといった学生が多い。本学部では一年次後期にプログラミングの講義があるが、初めからソースコードに間違いもなくすらすらと課題をこなしていく学生は一握りである。プログラミングを学び始めたばかりの初学者は、幾度となく表示されるコンパイルエラーを見ながら自分が記述したコードの間違っている箇所を修正する作業を繰り返すことになる。しかし、初学者にはコンパイルエラーや警告が何を意味しているのかが分からず、エラーの修正に多くの時間をかけてしまうことが多々ある。また、エラーの中にはコンパイル時にエラーとして出力されないインデントのミスといったものも含まれる。そのような初学者にこそ数多く起こりがちなミスに対して、まだプログラミングを始めたばかりの学習者が自力で対応することは困難である。学習を続けるにつれて慣れていくものではあるが、初めはエラー表示の示している内容をいちいち調べて理解していく必要がある。その調子ではプログラミング学習の進む速度は遅く、かといって毎回教育者にエラーの内容を聞くことは学習者にとっても教育者にとっても負担となる。これは学習者に対してプログラミングへの苦手意識を抱かせることに繋がり、その苦手意識がその後のプログラミング学習へ悪影響を与えることは容易に想像できる。この問題を解決する方法として、初学者によくある間違いに対しより分かりやすいフィードバックを即座に出力することにより、エラーの修正方針が立てやすくなり、無駄な時間を削ることで、学習効率を向上させることができる。

この方法に取り組んだ研究として、C-Helper [1, 2, 3] やこれを用いた島川の研究 [4] が挙げられる。しかしこれらは実装に Java を用いており、構文解析結果である構文木を扱う際に visitor パターンというデザインパターンが用いられているため、データ構造に変化が生じる場合は新しいメソッドの追加と各クラスにそのメソッドに応じた実装を追加する必要がある。そのため調査できる項目の拡張性に優れていない。

C-Helper と島川の研究の拡張性に優れていないという問題点に対して Haskell による構文解析を用いることで解決を試みた研究が木村の研究 [5] である。木村の研究はソースコードのエラー箇所をハイライトすることによって教育者の添削を支援するものである。しかし学習者に対して即座にミスのフィードバックを示すことはできず、教育者に対しても実際に目視で確認しコメントを付け足す必要があるため完全な自動化は図られていない。

そこで本研究では、Haskell を用いることで調査項目の拡張性を保持しながら、学習者に即座にフィードバックを行うことができるシステムの構築を目標とする。調査するソースコードに用いられるプログラミング言語は、本コースのプログラミングの講義でも初めに学習する C 言語を対象とする。またシステム導入の負担をできる限り削減し、プログラムの組み込みや修正が行いやすいように Web ベースでの実装を目指す。

1.2 関連研究

1.2.1 C-Helper

C-Helper とは、東京工業大学の内田公太が開発した C 言語初学者向けの静的解析ツールである。ソースコードの問題点を分かりやすいエラーメッセージとして表示し、解決策も提示することでプログラミング学習を支援するシステムである。C-Helper が調査できるエラー項目は以下のとおりである。

- インデントの乱れ
- printf のパラメタ間違い
- scanf の値渡しのミス
- return の記述漏れ
- char 型変数への文字列の代入
- 識別子の重複
- 定義されていない関数の使用
- 構造体のセミコロン忘れ
- 関数定義の余分なセミコロン
- 警告を抑え込むキャスト
- メモリリーク

- 動的に確保した配列に対する sizeof
- ヘッダファイルでの実態定義

しかし、C-Helper は C 言語の開発環境として広く使われているわけではない Eclipse のプラグインであるためシステムの導入に手間がかかることに加え、特定の Eclipse のバージョンでしか使用することができないという問題点がある。

1.2.2 C-Helper を用いた Web ベースの C 言語開発環境の構築

島川の研究であるこのシステムでは、C-Helper の導入に関する問題点に対して C-Helper の機能を Web ベースで利用できるようにすることで解決を試みている。もともと Eclipse のプラグインである C-Helper の、Eclipse に依存しているライブラリやそのライブラリを使用しているメソッドを書き換えることでソースコードの解析や解析結果の出力部分を Web 上で行えるように変更している。しかし、C-Helper の機能を Web ベースに移植する形で実装しているため、C-Helper で調査できない項目に関しては実装することができない。調査項目を増やそうとすると C-Helper のシステムに変更を加えた上でこのシステムにも実装させるという無駄な手間がかかってしまう。

1.2.3 構文解析を用いた C 言語指導コメントシステムの構築

木村の研究であるこのシステムは、構文解析を用いて C 言語指導コメント支援を行うことによりプログラミング指導者を支援するシステムである。タブレット型端末上での使用を想定しているため、Web ベースでの開発を行い、構文解析によって特定の構文や条件式を選択することで、指摘したい箇所の特定を素早く行うことができる。しかし、長いソースコードでは指摘したい箇所への位置まで移るのに手間がかかる点、同一構文内に複数の指摘を行った場合指摘コメントが分かりにくい点といった、学習者に即座にフィードバックを表示するうえでの課題点が存在する。

1.3 Web ベースシステムの利点

本研究で構築したシステムは、Web ページの入力フォームにソースコードを入力することでサーバに送信され、同じく Web ページに解析した結果を出力する、Web ベースのシステムである。システムを Web ベースにすることによって利用者に以下のような利点があると考えられる。

- システムのインストールを行う必要がない
- Web ブラウザがあればシステムを利用することができる
- 常に最新のシステムを利用することができる

また、開発側にも以下のような利点が考えられる。

- システムの導入に関する手順を説明する必要がない
- システムの改良や不具合の修正が行いやすい

特にシステムの導入に関する負担が双方ともに軽減されることに繋がり、気軽にシステムを利用することができる。

1.4 静的解析の利点

本研究のシステムではソースコードの静的解析を行う。静的解析とは、コンピュータのソフトウェアの解析手法の一種であり、実行ファイルを実行せずに解析を行うものである。反対に、ソフトウェアを実行して行う解析を動的解析と呼ぶ。静的解析を行うことでソースコードの記述段階でエラーを見つけられ、動的解析よりも早い段階での改良が行えるようになる。また、ソースコードの静的解析を行うことは学習者がインデントのルールに従うといったコードの可読性を高める記述ができるようになることに繋がる。それによりコードの誤読による内容の誤解を防ぎ、予期しないミスやバグの修正が比較的容易になるなどの利点が考えられる。

1.5 本研究に求められること

これらの点を踏まえて、本システムに求められる要件は以下のとおりである。

- 初学者向けの静的解析を行うことができる
- 学習者に即座にフィードバックを行うことができる
- 調査項目の拡張性が期待できる
- Web ベースのシステムである

プログラミング初学者が、自分が記述したソースコードのどの箇所が間違っているのか、何が間違っているのか、修正するにはどのようにすれば良いのかを理解し、判断することができるために、初学者が陥りがちなエラーに対してソースコードの解析を行い即座にフィードバックを与えることで、学習効率の向上に寄与することが期待される。また、システムを運用していれば新しく調査項目を追加する必要が出てくる場合があるが、その項目を簡単に拡張することができれば、システムの対応できる幅が広がる。多くの問題に対処することができるようになることで学習者からの要望も応えやすくなり、利用者が増えることに繋がる。そして学習者が容易にシステムを利用することができるように Web ベースでシステムを実装することが求められる。

1.6　章構成

本論文の構成は以下のようになっている。第 2 章で本システムを実装するにあたって使用した技術について説明する。第 3 章では実装したシステムについて述べる。第 4 章では本システムの試用実験と評価について述べる。第 5 章で本論文のまとめと課題点について述べる。

第 2 章

使用技術

ここでは本システムに使用した技術について説明する。

2.1 Haskell

2.1.1 Haskell とは

Haskell は、計算や処理などを関数の定義の組み合わせとして数学的に扱い記述を行っていく関数型言語の 1 つである。言語の名称は記号論理学者の Haskell Brooks Curry 氏に由来している。現在主流として用いられている処理系は GHC (Glasgow Haskell Compiler) である。

2.1.2 Haskell の特徴

Haskell には参照透過性や遅延評価、静的型付けといった関数型言語に多く採用されている機能に加え、パターンマッチや型推論、モナドなどの特徴的な機能がある。これらの機能を組み合わせることで命令型言語に比べて Haskell はより簡潔かつ容易に関数を実装できることが多い。以下に Haskell の特徴的な機能の例を示す。

- 参照透過性

  C 言語や Java などの命令型の文法を持っている言語では、プログラミングで解決したい課題に対して、命令や命令をひとまとまりにした手続きを組み合わせて記述していく。命令の組み合わせによってシステムの状態を変更しているため、複数の処理で同じ変数を参照している場合予期していない結果が出力されてしまうことがある。これはプログラミングの副作用と呼ばれる。Haskell を含む関数型言語ではプログラミングの目的に着目し、その目的に沿った関数を定義していくことで記述を行う。関数型言語にはプログラミングの副作用

を基本的に排除する考えがあり、同じ引数で呼び出した関数が返却する値は常に同じ値になるという数学的な特徴がある。これを参照透過性といい、いつ関数を呼び出しても同じ結果を返すことが安全なプログラムの構築に寄与する。

- 遅延評価

  一般的なプログラミング言語の場合、関数を呼び出す前に引数が評価されその結果が関数に渡されるという正格評価が用いられている。しかし Haskell では引数の値が必要になる時まで評価が行われない特徴がある。これを遅延評価といい、関数の引数を参照するときだけでなく変数の値を参照する際にも行われる。無限リストの扱いに優れているため、主に無限リストから値を取り出す際に用いられることが多い。

- パターンマッチ

  Haskell の関数は実行時の引数の値に応じて関数の処理を場合分けして定義することができる。関数を呼び出した際は、関数定義の上から順番に引数とパターンを照合し、パターンにマッチした定義の右辺の処理が実行される。引数がどのパターンにもマッチしない場合エラーが表示される。以下の引数の階乗を返却する関数の例では、関数 fact の引数が 0 の場合返却値は 1 となり、0 でない場合は n に引数が束縛される。

```
fact 0 = 1
fact n = n * fact (n - 1)
```

- 型推論

  コンパイル時にデータ型を判定する機能を型推論と呼ぶ。静的型付けはコンパイルの段階で事前にエラーが発見できる代わりにあらかじめデータ型を指定する必要があり、プログラムの複雑さや冗長さの原因となってしまう。動的型付けはデータ型を指定する手間が省ける代わりに実行するまでエラーが分からないというデメリットがある。Haskell の型推論は、与えられたコードからコンパイラが自動的に型を推測して判断することができる。型推論の機能を用いることでデータ型を指定しなくてもコードを書くことができ、コンパイルの段階で間違いを発見することが可能なことで、コードの簡潔な記述とエラーの発生を抑えることの両方を実現している。

- モナド

  モナドとは、Haskell で変数などの値といった、状態を変更する破壊的代入や入出力といった様々な言語におけるプログラミングの副作用を扱うための方法である。そういった副作用という特徴の型は共通の構造を持ち、同じ構造の演算子で扱うことができる。その共通の構造を持つ型がモナドと呼ばれる。中でもよく用いられるのが IO モナドである。主に入出力に関する役割を担う IO モナドを用いた例を以下に示す。

```
main :: IO ()
main = do
          line <- getLine
          putStrLn line
```

上記のコード例では、getLine で標準入力から取得した文字列を line に束縛し、putStrLn の引数に入れている。putStrLn は String を引数として IO アクションを返す関数であり、標準出力への表示を行う。

2.2 Haskell 関連技術

2.2.1 GHCup

GHCup [6] は GHC とその周辺ツールのインストールやバージョン管理を行うツールである。周辺ツールには Cabal、HLS といったものが含まれる。GHCup では複数のバージョンを管理することができるため、最新版の GHC を使用したり、最新の GHC に対応していないツールを利用するためにあえて古いバージョンの GHC を用いたりすることができる。コマンド ghcup list によって何がインストールされているかを確認することができる。

2.2.2 Cabal

Cabal [7] は Haskell のライブラリとプログラムを、ビルド及びパッケージ化するためのシステムである。パッケージの作成者と配布者がアプリケーションを移植可能な方法で容易に構築することができるように共通のインタフェースが定義されている。本システムではシステムプロジェクトの作成及び必要なライブラリやパッケージのインストールに Cabal を用いている。

2.2.3 HLS

HLS (Haskell Language Server) [8] は Haskell の IDE 環境であり、Language Server Protocol のためにサーバを実装したものである。警告やエラーのハイライト、コードの補完、マウスオーバーによる型やドキュメントの表示、定義へのジャンプなどの便利な機能がサポートされている。

2.2.4 language-c-quote

language-c-quote [9] は Haskell のライブラリであり、一般的な C 言語のパーサを提供している。language-c-quote は構文解析の結果が扱いやすいため、新しく調査項目を増やす際に容易にプロ

グラムを追加することができると考え本システムの構文解析に用いている。以下に実際に C 言語のソースコードを解析した結果を載せる (図 2.1)。その解析結果から調査したい構文等を抜き出し真偽を確かめている。

```
#include <stdio.h>

int main(void){
    int i = 5;
    printf("%d", i);
    return 0;
}

[FuncDef (Func (DeclSpec [] [] (Tint Nothing noLoc) noLoc) (Id "main" n
oLoc) (DeclRoot noLoc) (Params [Param Nothing (DeclSpec [] [] (Tvoid no
Loc) noLoc) (DeclRoot noLoc) noLoc] False noLoc) [BlockDecl (InitGroup
(DeclSpec [] [] (Tint Nothing noLoc) noLoc) [] [Init (Id "i" noLoc) (De
clRoot noLoc) Nothing (Just (ExpInitializer (Const (IntConst "5" Signed
 5 noLoc) noLoc) noLoc)) [] noLoc] noLoc),BlockStm (Exp (Just (FnCall (
Var (Id "printf" noLoc) noLoc) [Const (StringConst ["¥"%d¥""] "%d" noLo
c) noLoc,Var (Id "i" noLoc) noLoc] noLoc)) noLoc),BlockStm (Return (Jus
t (Const (IntConst "0" Signed 0 noLoc) noLoc)) noLoc)] noLoc) noLoc]
-----
```

図　2.1: language-c-quote によるソースコードの解析結果

2.2.5　Wai

Wai [10] は Web アプリケーションと Web サーバとの間の通信プロトコルを提供する Haskell のライブラリである。Wai を用いることで Web ブラウザ上での入出力とサーバ側の Haskell システムとを繋いでいる。

2.2.6　Warp

Warp [11] は Wai をハンドリングするための高速かつ軽量な HTTP サーバである。本システムでは Haskell プログラム内での Warp によりローカルサーバを立て、ブラウザ上でそこにアクセスすることでシステムの実行を行うことができる。以下に Wai と Warp を用いて簡単な Web アプリケーションを作成する方法を載せる (図 2.2)。

これを build して run し、localhost:8080 を開くと Hello が表示されている。

```haskell
{-# LANGUAGE OverloadedStrings #-}
module Main where

import qualified Network.Wai.Handler.Warp as Warp
import qualified Network.Wai as Wai
import qualified Network.HTTP.Types as Htypes

main :: IO ()
main = do
  putStrLn "http://localhost:8080/"
  Warp.run 8080 app

app :: Wai.Application
app req send = send $ Wai.responseBuilder Htypes.status200 [] "Hello"
```

図　2.2: Wai、Warp によるアプリケーション作成

第 3 章

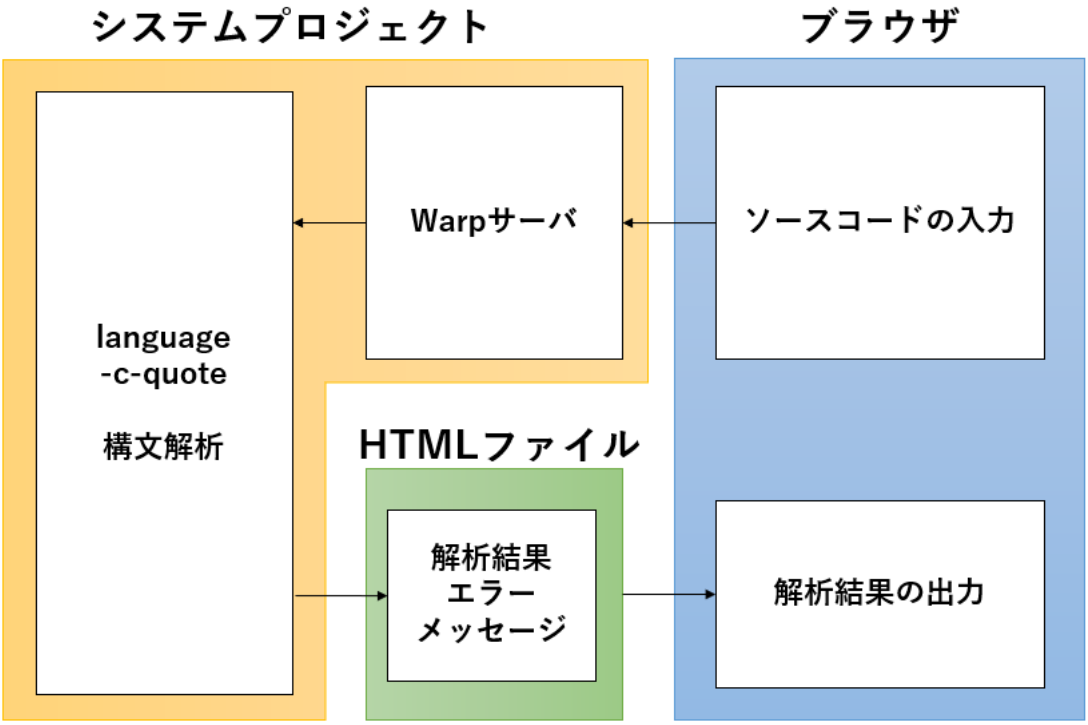
システムの実装

ここでは本システムについて実際の実行画面を踏まえて詳細に述べる。

3.1　システムの概要

C 言語のソースコードを受け取り、プログラミング初学者が犯しがちな間違いを見つけ、ソースコード中の間違っている箇所とそのエラーの修正案をそれぞれ提示する。プログラミングの学習を支援することで初学者がプログラミングに対して苦手意識を持つことを防ぎ、効率的に学習を進められるシステムを構築した。本研究のシステムは Haskell のプロジェクトであり、Wai と Warp を用いることで Web ベースでのシステム構築を行っている。

本システムの開発環境としては、GHCup 0.1.18.0、Cabal 3.6.2.0、HLS 1.8.0.0、GHC 8.10.7 を用いている。Wai の機能を用い、URL によるルーティング機能が付いたサーバを立ち上げる。システムを実行するとまずソースコードを入力する画面が表示される。ブラウザ上のテキストエリアの入力フォームに入力されたソースコードを文字列として受け取り、受け取った文字列を Haskell の language-c-quote を用いて構文解析を行う。解析した結果から初学者が間違いを起こすことの多い if 文などの情報を抜き出し、間違った記述をしていないか調査を行う。そこで間違っている行番号と内容を HTML ファイルに出力し、そのファイルをルーティングされた解析結果を表示するページで出力する。この出力表示ではソースコードの間違っている行番号とその内容が一目で分かるように、間違っている行番号の表示を他の行と比べて目立つように色を変えている。ソースコードの入力画面と出力画面はそれぞれクリックひとつで即座にページの遷移を行うことができる。以下にシステムの概要図を示す。



3.2　ソースコードの解析

本システムではソースコードの解析に Haskell のライブラリである language-c-quote を用いている。

```
import Language.C
import Data.ByteString.UTF8 as UTF8 (fromString)

parseProg :: String -> Either SomeException [Definition]
parseProg str = parse [C11] [] parseUnit (UTF8.fromString str)
    Nothing
```

上記の parseProg のような関数を作成し、引数 str としてソースコードの文字列を渡すことで構文解析が行われる。実際に以下のソースコードを解析した結果を載せる。(図 3.1)

```
#include <stdio.h>

int main(void){
    int i = 5;
    if (i == 5) {
        printf("%d", i);
    }
    return 0;
}
```

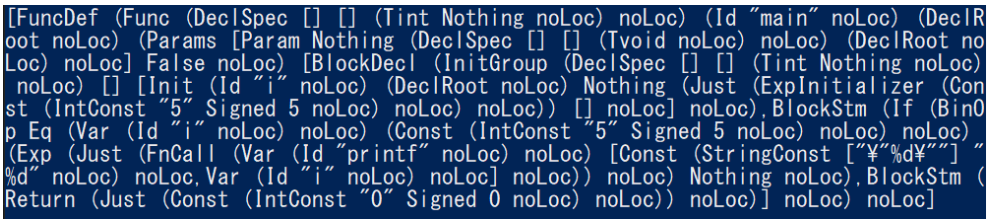


図　3.1: language-c-quote で解析した結果

このような解析結果から調査したい構文や変数等を抜き出すことで真偽の判定を行っている。図 3.1 の解析結果から実際に if 文や printf の情報を抜き出す方法を以下に述べる。

language-c-quote で解析された結果を読み取る際に重要なことは、いくつも含まれるデータ型に対してそれぞれのコンストラクタを理解することである。データ型の種類とコンストラクタは、Language.C.Syntax [12] に詳しく記されている。例えば図 3.1 の初めにある FuncDef の構成は、FuncDef Func !SrcLoc となっている。このようにして目的の構文を探す。if 文が始まっている箇所のデータ型は、5 行目の BlockStm から始まる部分に当てはまっている。Language.C.Syntax と見比べながら詳しく見ると、BlockStm はデータ型 Stm から始まり、Stm が If の形を取る場合 If Exp Stm (Maybe Stm) !SrcLoc という構成となっていることが分かる。ここでは Exp には (BinOp Eq (Var (Id “i” noLoc) noLoc) (Const (IntConst “5” Signed 5 noLoc) noLoc) noLoc) が含まれている。この BinOp Eq は Var と Const がイコールであることを示しており、元のソースコードの if 文の条件式である (i == 5) を解析した部分と対応している。if 文中の printf は構成の Stm に対応しており、解析した結果の (Exp (Just (FnCall (Var (Id “printf” noLoc) noLoc) [Const (StringConst [“¥“%d¥””] “%d” noLoc) noLoc, Var (Id “i” noLoc) noLoc] noLoc)) noLoc) が printf の内容を示している部分である。

本システムではこれらの情報を Haskell のコードによって抜き出している。if 文の条件式や printf のパラメタミスといった間違いをこのようにして調査している。以下に printf の情報を抜き出す関数の例を載せる。

```
strsearch :: Int -> String -> Int
strsearch num (x1:x2:xs)
  | (((x1 == '\%') && (x2 == 'd')) == True) = strsearch (num+1)
      xs
  | otherwise = strsearch num (x2:xs)
strsearch num _ = num

myStm :: Stm -> Maybe Int -> Bool
myStm (Exp (Just (FnCall (Var (Id "printf" _) _) ((Const (
    StringConst _ str _) _):args) _)) loc) _
  | (strsearch 0 str) == (length args) = True
  | otherwise = False
```

関数 myStm によって解析結果の中から printf の部分を抜き出している。抜き出す際に設定した値の str と args にはそれぞれ、“%d” と Var (Id “i” noLoc) noLoc が含まれている。複数の変数が指定されている場合、“%d %d”や (Var (Id “i” noLoc) noLoc, Var (Id “i” noLoc) noLoc) が値に入る。関数 strsearch で文字列中の%d の数を数えることによって、(strsearch 0 str) は printf の書式文字列内で指定されている数を特定している。また、args には printf で指定されている変数が複数の場合も含まれているため、length を用いて変数の数を求めることで、関数 myStm は printf で記述されている書式文字列内と変数それぞれの数が一致していれば True を返し、一致していなければ False を返す。このようにコードを記述することで printf の情報を抜き出し、間違っているかどうかの調査を行うことができる。他の調査項目に関しても同様に、解析結果から調査したい構文の部分を抜き出して調査を行っている。そのため本システムの調査項目は Language.C.Syntax を理解することで、Haskell の知識さえあれば拡張することが可能である。

3.3　調査可能な項目

本システムで調査することができる、初学者が犯しがちな間違いを以下に述べる。それぞれミスの例を載せ、エラー表示の文を示している。

- インデントのミス

  インデントについては別項目で詳しく説明する。

- printf の変数の数が間違っているミス

```
1  #include <stdio.h>
2
3  int main(void){
4      int i = 5;
5      printf("%d %d", i);
6      return 0;
7  }
```

  エラー表示：5 行目 printf で指定されている変数の数が違います。

  printf で変数を表示する際、第 1 引数の書式文字列内で「%」記号と変数の型を表す変換指定子を用い、第 2 引数以降に変数を指定することで表示させている。書式文字列内と変数それぞれで指定した数が違う場合エラーメッセージが表示される。現状対応している変換指定子は「d」のみである。

- scanf の変数の数が間違っているミス

```
1   #include <stdio.h>
2
3   int main(void){
4       int m;
5       int n;
6       scanf("%d", &m, &n);
7       return 0;
8   }
```

エラー表示：6 行目 scanf で指定されている変数の数が違います。

printf と同様変数を指定する際に、第 1 引数の書式文字列内で「%」記号と変数の型を表す変換指定子を用い、第 2 引数以降に変数を指定する。書式文字列内と変数それぞれで指定した数が違う場合エラーメッセージが表示される。現状対応している変換指定子は「d」のみである。

- if 文の条件式が関係演算子ではなく代入式になっているミス

```
1   #include <stdio.h>
2
3   int main(void){
4       int i = 5;
5       if (i = 6){
6           printf("%d", i);
7       }
8       return 0;
9   }
```

エラー表示：if 文の条件式は = ではなく == を使用してください。

条件分岐を行う if 文の条件の部分には関係演算子という 2 つの値の大小関係や等値関係を判定する演算子が用いられる。関係演算子には「>」、「<=」などや、2 つの値が等しいことを示す「==」というものがある。これはイコールを 2 つつなげて等しいことを表しているが、間違ってイコールを 1 つで表そうとしているとエラーメッセージが表示される。通常では if 文の条件を記述する部分に代入式を書いても文法的には間違っていないためコンパイルが成功してしまうが、今回は初学者が間違いに気付けるようにイコール 1 つの場合にエラー表示を行うようにした。

- 関数名が他の関数と重複しているミス

```
1   #include <stdio.h>
2
3   int func(int i){
4       printf("%d", i);
5       return 0;
6   }
7
8   int func(int i){
9       printf("%d", i*5);
10      return 0;
11  }
12
```

エラー表示：３行目の関数名が他の関数と重複しています。

定義されている関数の名前が他の関数と重複している場合エラーメッセージが表示される。関数名の調査はソースコードの上から順番に行っているため、重複している二つの関数のうち行番号が少ない方の関数のみエラーメッセージの表示を行う。また、行番号の強調表示に関しては、関数定義の関数名が書かれている箇所に強調が行われている。

- 返却値が int 型の関数に return が記述されていないミス

```
1   #include <stdio.h>
2
3   int func1(int i){
4       printf("%d", i);
5       return 0;
6   }
7
8   int func2(int i){
9       printf("%d", i*5);
10  }
11
```

エラー表示：８行目の関数に return がありません。

C 言語の関数では戻り値の型を指定して関数の定義が行われるが、その返却値が int 型の場合関数内で返却する int 型の値を return する必要がある。その return が行われていない場合、エラーメッセージの出力が行われる。また、エラー箇所の強調表示はその関数の定義が始まっている箇所の行番号を強調している。

3.4　インデントのミス

ソースコードを記述する際のインデントには様々な方法が存在する。本システムで調査するインデントの正しい方法に関しては、筆者の指導教員であり、本学で最初にプログラミングを学習する講義の担当教員でもある香川が講義で指導しているインデント方法 [13] を参考として作成している。以下に本システムでのインデントルールの特徴的な点について示す。

- 一行に文は一つしか記述しない。
- 開きブレース「{」は if や for などのキーワードと同じ行に改行せずに記述する。開きブレースのあとは何も書かずに改行を行う。
- 閉じブレース「}」は if や for などのキーワードの初めの文字と列をそろえて記述する。その行には閉じブレース以外何も書かない。
- ブレース「{ 〜 }」の中は外より空白文字 4 つ分字下げを行う。
- 字下げにはタブ文字を使わずに空白文字だけで字下げを行う。
- if 文や for 文などでは、選択されたり繰り返したりされる文が一つだけの場合でもブレースで囲んで記述を行う。
- 関数の定義は行頭から記述を行う。
- 余分なブレースは記述しない。

```
1    #include <stdio.h>
2
3    int main(void){
4      int i = 5;
5           printf("%d %d", i);
6        return 0;
7    }
```

エラー表示：４行目 インデントをスペース２つ分追加してください。

5行目 インデントをスペース４つ分減らしてください。

インデントの基準は空白文字４つ分であり、それより少ないか多い場合は正しいインデントになるまで変更するべき空白文字の数をエラーメッセージとして表示を行う。

3.5　追加できなかった調査項目

実装を目指したが実装できなかった調査項目について以下に示す。

- for 文の書式の間違い

  `for（ 変数の初期化; 繰り返しの条件式; 変数の変化式）`

  上記のように、C 言語の for 文ではいくつかの式と 1 つの変数を用いて記述を行う。まず代入式等を用いて変数に初期値を設定する。次に、条件式で繰り返しを続けるための条件を記述する。条件式が真の値をとり続けている間は for 文の繰り返しが行われる。そのためここでは、「変数の値が 0 より大きいか」のような変数を含む 2 つの値の大小関係といった式が記述される。変化式では繰り返しが行われるたびに、変数に 1 を足したりといった変数の値をどう変化させるかを記述する。このように、for 文を記述するためには三種類の異なる様式の式を用いる必要がある。そのため初学者が記述する際に間違いを起こしやすいため調査の実装を行いたかったが、同じく解析する際も解析結果が複雑になってしまい、時間の都合で実装できなかった。

- if 文の中にあるミスの特定

  解析結果

```
1   #include <stdio.h>
2
3   int main(void){
4       int i = 5;
5       if (i == 5){
6           printf("%d %d", i);
7       }
8       if (i = 5){
9           printf("%d %d", i);
10      }
11      return 0;
12  }
```

```
ERROR
6行目 printfで指定されている変数の数が違います
8行目 if文の条件式は = ではなく == を使用してください
```

上図のように、if 文の条件式が成立する場合の処理の中に調査可能なミスがあった場合、その if 文の条件式が間違っていなければ正確に判定されるが、if 文の条件式が間違っているとそのミスのエラー判定をしたところでその構文の解析が終了してしまい、中の処理部分の

エラー判定が行われない。そのため修正を挟んで二度システムを実行すればエラーの特定は正常に行えるが、最初の実行で if 文の条件式と処理との両方のエラーを特定できれば修正の手間も一度で済む。Haskell の Generic Programming に関するライブラリ [14] を用いて、間違っている項目が見つかった場合もそこで調査を止めることなく、ソースコードの隅々まで調査が行えるようにする必要がある。

- else 文への対応

```
1   #include <stdio.h>                    ERROR
2
3   int main(void){
4       int i = 5;
5       if (i == 5) {
6           printf("%d", i);
7       } else {
8           printf("%d %d", i*5);
9       }
10      return 0;
11  }
```

上図の 8 行目のように if 文の条件式が偽となった場合の処理に用いられる else 文で実行される調査可能なはずのエラーの判定が行われない。else 文にも対応させようとする場合、language-c-quote で解析された結果を Haskell で抜き取る際に if 文の情報だけでなく、そこに付随する else 文に関する情報も値を設定して格納することで参照することができるようになると考えている。

- ソースコード中に日本語がある場合

ソースコード内に日本語がある場合、解析が上手く行われず、解析結果の表示がされなくなってしまっている。そのため printf での日本語表示やコメントが含まれるソースコードに対応ができない。これは language-c-quote で解析が行われる際に、字句解析によってコードが構文上意味のある最小単位のトークンに区切られることが原因と考えられる。これを解消するためには、文字列リテラルやコメントに対して pack を用いて bytestring に変換し、エンコードによって UTF-8 が含まれているテキストをデコードする必要があると考えられる。

- 宣言した変数の型と違う型の値を代入しているミス

初学者が起こしがちなミスの一つてして、宣言している変数の型と違う型の値を代入してしまうミスが挙げられる。例として、以下のソースコードを挙げる。

```
int a = 5;
double b = 3.6;
int c;
char d;
```

```
c = a + b;
d = "string"
```

この例では、int 型の変数 c に代入される (a + b) の値は double で行われる。そのため int 型の変数に代入しようとするとエラーとなる。また char 型の変数 d には文字列を格納できないためこちらもエラーとなる。このように変数宣言時の型によって起こる間違いを特定したいと考えていた。そのためには宣言される変数とその型、またそれに代入される値の型を全て把握し、照らしあわせる必要がある。

- scanf の値渡しのミス

  C 言語における scanf では第 2 引数以降に値を格納する変数の記述を行うが、C 言語では変数に格納された値のコピーのようなものを渡すことしかできないため、int 型の変数等に入れ込む場合変数のアドレスを渡すことによって scanf 関数で値の格納を行うことができる。これは変数に「&」をつける形で記述される。しかし文字列を渡す際は配列名に「&」をつける必要がない。これは配列名が配列の最初の要素のアドレスを表していることによるものである。この違いは初学者が困惑する項目の一つであり、優先的に実装を行う必要がある。

3.6　システムの実行

本システムは Haskell プログラム内での Warp によりローカルサーバを立て、その Web ページにアクセスすることでシステムを実行する。システムページにアクセスすると、以下の画面が表示される (図 3.2)。

表示されているテキストエリアに解析したいソースコードを入力し、CHECK ボタンを押すことで Haskell プログラムに文字列としてソースコードが送信され、構文解析が行われる。そして解析した結果とエラーの箇所や内容を HTML ファイルに書き込むことで記録する。CHECK ボタンを押した際にページが結果表示画面に遷移し、その書き込まれた HTML ファイルが表示される。結果表示の際にはエラーがある行番号の色を変更することで強調している。また、左右に分割してコードとエラー内容を表示している。実際にソースコードを入力した際の結果表示画面を以下に示す (図 3.3)。

本システムで調査できる項目の中には、通常のコンパイルではエラーとして認識されないミスもある。インデントのミスと if 文の条件式のミスがそれにあたる。そのため本システムを用いることで通常のコンパイルでは気づけないミスに対してもエラー表示がされ、エラー内容の出力も日本語で行われるためそのミスの修正も容易に行うことができる。実際に下記のソースコードをコマンドプロンプトでコンパイルした場合 (図 3.4) と本システムで調査した場合 (図 3.5) の結果を示す。

```
#include <stdio.h>

int main(void){
  int i = 5;
    printf("%d␣%d", i);

    if (i = 5) {
        printf("%d", i);
    }

return 0;
}
```

このように、コマンドプロンプトではインデントのミスや if 文の条件式に対する警告が表示されていない。本システムではそれぞれのミスの箇所と内容を表示させており、特にインデントのミスは正しい状態にするためにスペースをいくつ追加したり減らしたりすればいいかが表示されているため、利用者が自身のソースコードのインデントを容易に整えることができる。

解析結果の表示画面では特に初学者がソースコードのエラーを修正できるように分かりやすくエラー内容を表示することを意識している。表示画面の左側では入力されたソースコードを表示させ、エラーがある行の行番号は赤色に変更することで一目で間違っている行を理解することができる。図 3.4 のコマンドプロンプトの表示ではエラー内容の表示しか行われていないため、視覚的にエラー箇所を理解することができない。画面の右側には生じているエラーの内容について行番号と共に記している。エラーによっては修正するために何を行えば良いかの表示もしており、表示を見てすぐにエラーの修正を行うことができる。

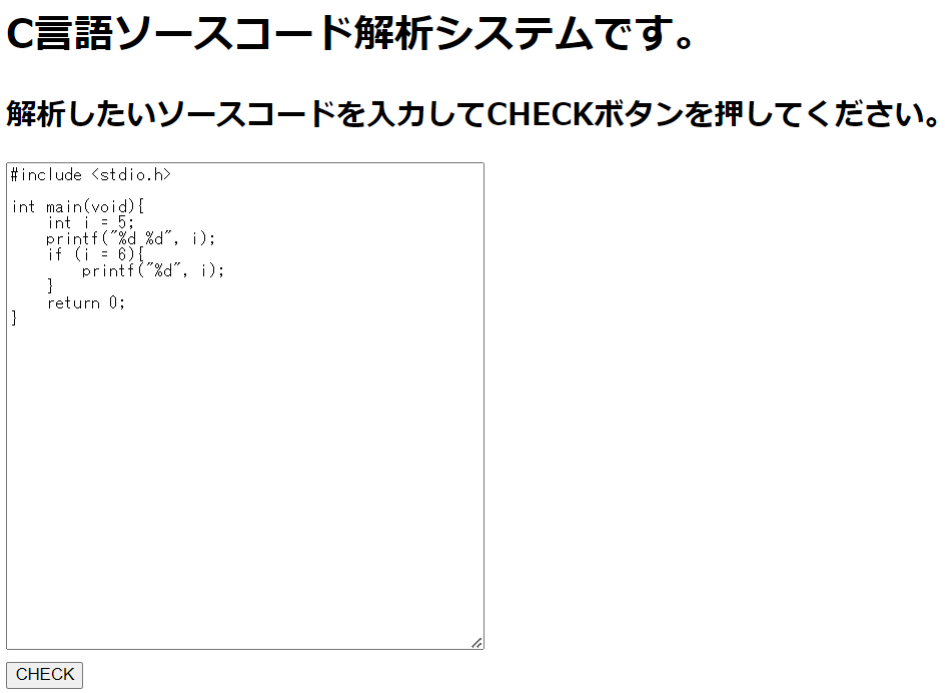


図　3.2: ソースコード入力画面

図　3.3: 解析結果表示画面

```
Microsoft(R) C/C++ Optimizing Compiler Version 19.21.27702.2 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

test.c
test.c(5): warning C4473: 'printf': 書式文字列として渡された引数が不足しています
test.c(5): note: プレースホルダーとそのパラメーターには 2 の可変個引数が必要ですが、1 が指定されています。
test.c(5): note: 不足している可変個引数 2 が書式文字列 '%d' に必要です
Microsoft (R) Incremental Linker Version 14.21.27702.2
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:test.exe
test.obj
```

図　3.4: コマンドプロンプトでのコンパイル結果

解析結果

```
1    #include <stdio.h>
2
3    int main(void){
4      int i = 5;
5        printf("%d %d", i);
6
7        if (i = 5){
8            printf("%d", i);
9        }
10
11   return 0;
12   }
```

ERROR
4行目 インデントをスペース2個分追加してください
5行目 printfで指定されている変数の数が違います
7行目 if文の条件式は = ではなく == を使用してください
11行目 インデントをスペース4個分追加してください

図　3.5: 本システムでの解析結果

第 4 章

システムの試用実験と評価

ここでは、システムの試用実験と評価について述べる。

4.1　システムの試用実験

本研究室の学生を対象に、本システムの試用実験を行った。実験方法としては、実際に本システムのソースコード解析機能を使用してもらい、その後アンケートに回答してもらう形式で行った。この試用実験では調査項目の拡張しやすさに関する実験を行うことは困難であったため、もう一つのシステムの目的である間違っている箇所の指摘に関連した、エラーが起きている箇所が分かりやすいか、エラーメッセージによってしっかりと修正を行えそうかという点に特に注目して行ってもらった。

4.2　評価項目

評価項目に関して、全部で 6 つの項目を設けた。1 つ目の項目は、「表示画面は分かりやすいか」という質問である。2 つ目の項目は、「エラー箇所は分かりやすいか」という質問である。3 つ目の項目は、「エラー内容は分かりやすいか」という質問である。これら 3 つの質問に対する回答は、1 の「わかりやすい」から、4 の「わかりにくい」までの目盛りを選択するようにした。4 つ目の項目は、「良いと思う点はどこか」という質問である。5 つ目の項目は、「悪いと思う点はどこか」という質問である。6 つ目の項目は、「気になった点、欲しい機能はあるか」という質問である。これら 3 つの質問に対する回答は、自由記述とした。

4.3　結果

アンケートの結果を以下に述べる。まずは選択式の評価項目について述べる。1 つ目の項目に対する回答は図 4.1 に示すとおりである。1 番の「分かりやすい」を選んだ回答者と 3 番を選んだ回答者で大きく 2 つに分かれている。他の 2 番と 4 番にはそれぞれ 1 名ずつが回答している。

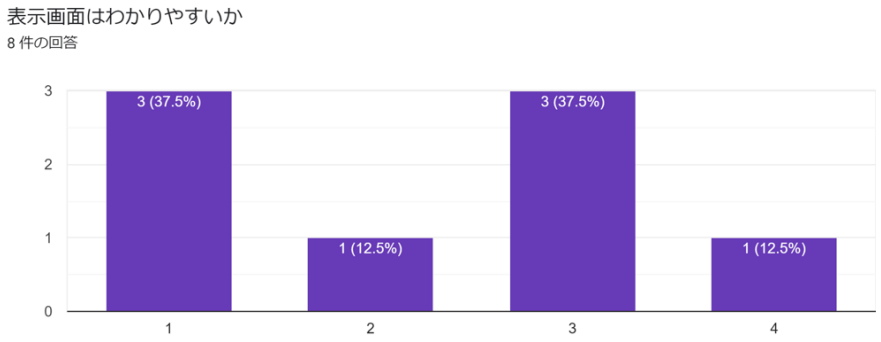


図　4.1: 1 つ目の項目に対するアンケート結果

2 つ目の項目に対する回答は図 4.2 に示す。1 番の「分かりやすい」を選んだ回答者が過半数を占めている。他の 2 番、3 番、4 番にはそれぞれ 1 名ずつ回答している。

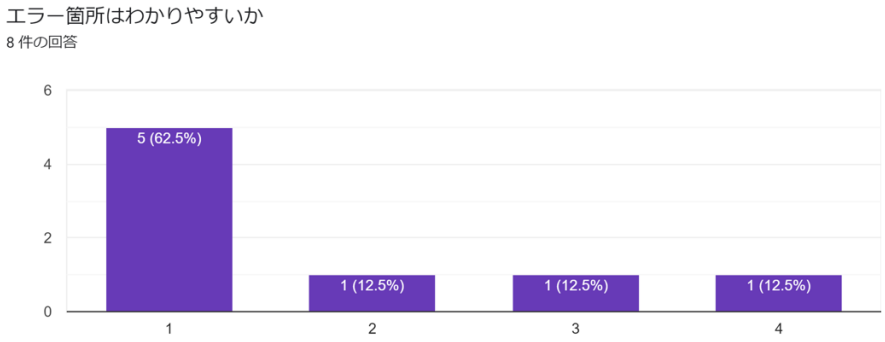


図　4.2: 2 つ目の項目に対するアンケート結果

3 つ目の項目に対する回答は図 4.3 に示すとおりである。1 番の「分かりやすい」と 2 番を選んだ回答者が多いことが見て取れる。他の 3 番、4 番にそれぞれ 1 名ずつ回答している。

次に、自由記述での評価項目について記す。「良いと思う点はどこか」についての回答には以下のような回答があった。

- 修正内容を日本語で出力しているため、分かりやすい
- エラーのある行番号の色が変わっているため、修正を行うべき箇所が分かりやすい
- システムがシンプルで、操作に迷うことがなかった

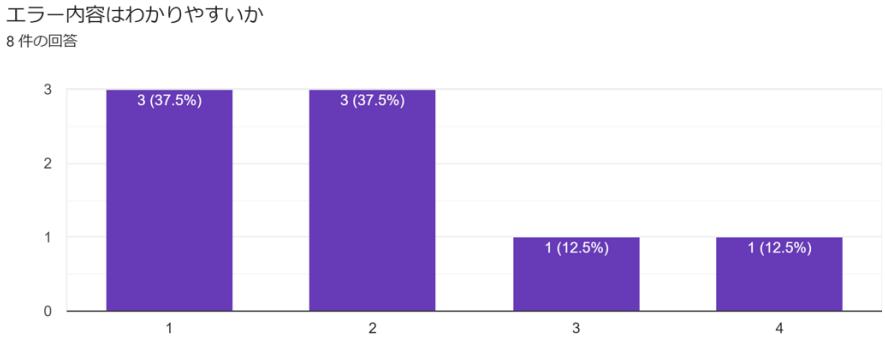


図　4.3: 3 つ目の項目に対するアンケート結果

- 長いコードでも実行速度が速かった
- エラーがある箇所の表現が分かりやすかった
- 何がどう間違っているのかが具体的だった

「悪いと思う点はどこか」についてには以下のような回答があった。

- 対応していない要素が多い点
- インデントの数は固定せず、変えることができるようにして欲しい
- スペルミスがあった場合エラーの検知ができない
- 解析結果が別ページで表示される点

「気になった点、欲しい機能はあるか」という質問には以下のような回答があった。

- エラー箇所の行番号だけではなく、行全体を強調して欲しい
- 対応している、していない要素についてシステム内で説明が欲しい
- テキストエリアへの入力だけでなく、ファイルで送信したい
- インデントのミスをエラーととらえるべきかどうか
- 解析結果を見てその場でコードを修正したい
- 自動でインデントのミスを修正して欲しい

4.4　考察

アンケートの結果から、エラーの箇所や内容に対する評価は「分かりやすい」と答えた回答者が多く、本システムがエラーの起きている箇所や修正方針を示すことで、初学者がエラーを修正する作業を効率的に行うことに寄与できると考えられる。しかし、まだ対応していないエラーも多く、調査できるエラーとできないエラーについてシステム内で説明を付け足す必要がある。また、結果を表示する画面に関しても課題点が多く、特にページが遷移してしまうことで元のコードへの修正が行いにくいという意見があった。本システムの開発段階では既にエディタ上で記述を済ませているソースコードをペーストすることでの利用を想定していたため、そのような点への配慮が欠けてしまっていた。そのためソースコードの入力と解析結果の表示は同じページ上で行えるようにし、スムーズにコードの修正と解析を交互に行えるようにできればより良いシステムになると考えられる。また、インデントに関しての意見として、インデントのミスを他のエラーと同様の扱いをすることは学生にとって印象が良くないのではないかといった意見があった。確かにインデントのミスは他のエラーと比べて、大きくプログラムの動作に変化が起こるわけではない。しかしインデントを整えることでどの行が同じブロックに含まれるのかが明示的になり、コードの可読性が上がる。自分にとっても他者にとってもコードが読みやすく、ミスをしている箇所の探しやすさやプログラムの処理や流れの理解しやすさに繋がるため、コーディングにおける重要な項目の一つであると考えている。そのコーディング時に重視するべきであるインデントは、一度自分が行っている記述の仕方に慣れてしまうと後から修正を行うことが困難になってしまう。そのためプログラミングを始めたばかりの時からインデントを意識的に行うようにすることが重要であると考え、本システムでも特に重要視している。そのインデントのルールには様々な考えがあるが、本システムでは既に示したインデントルールにのみ対応しているため、字下げの際のスペースの数や改行を行う位置など違う方法でのインデントにも対応させることができればと考えている。

第 5 章

結論

5.1 まとめ

本研究では、Haskel による構文解析を用いてプログラミングを始めたばかりの初学者が犯しがちな間違いを特定し、その間違っている箇所と間違いの内容を分かりやすく表示させることで、初学者のプログラミング学習を支援するシステムの開発を行った。Web ブラウザ上の入力フォームに解析したいソースコードを入力することで即座にフィードバックが行われる。調査可能な項目は Haskell の知識があれば今後もあまり手間をかけることなく実装できることが想定される。システムを Web ベース上で実装しているため、利用者はシステムの導入作業を行う必要がなく、気軽に利用することができる。しかし、構文解析によって調査が行われる項目がまだ少なく、解析結果を表示する際にも多くの課題が見つかった。

5.2 今後の課題

以下に、本システムの課題点及び解決策を述べる。

- 調査が可能な項目が少ないこと

  現状調査可能な項目として、インデントのミス、printf や scanf の受け取る変数の数が間違っているミス、if 文の条件式が関係演算子ではなく代入式になっているミス、関数名が他の関数と重複しているミス、返却値が int 型の関数に return が記述されていないミスがあるが、これだけでは到底プログラミング学習を完全に支援できるとはいいがたい。そのため特に初学者が犯しがちなミスについて優先的に実装をしていく必要がある。実際にプログラミングの講義中に提出された課題を分析し、実装するべき項目を精査することも重要である。

- ソースコードの入力画面と解析結果の表示画面が別ページであること

  本システムでは解析したいソースコードを入力して解析を始めると、自動的に別のページに遷移して解析結果の表示が行われている。それは学習者にとってエラーの修正と解析を繰り返すことのスムーズさに繋がるものではない。初学者にとってこのエラーの修正作業は何度も行うことになるものであり、その点が煩わしいとシステムを利用する機会が減ってしまう恐れがある。そのため入力画面と解析結果の表示は同じページ内で行えるようにすることで、容易に自分のソースコードを改善することができ、システムの積極的な利用に繋がると考えられる。

- 実際に大人数が同時にシステムを利用する実験ができていないこと

  本システムが想定しているシステムの利用として、学習者がプログラミングの講義を受け、講義中や講義後の課題に取り組む際に自身の記述したソースコードが間違っていないか確認を行う際に利用されることを想定している。特に講義中に出された課題に対して取り組む場合、一度に多くの学習者が利用することが考えられる。本研究の試用実験は 10 人程度での実験となっているため、実際に何十人もが同時にシステムを利用することになった場合にどのようなアクシデントが起こるかが想定できない。現状長いソースコードに対しても解析結果の表示が素早く行われ、ページの遷移等の動作が重いといった挙動は確認されていないが、実際に想定している環境で試用実験が行えていないことは課題点である。

- Web ベースのシステムである利点を最大限に活かせていないこと

  本システムは利用者がシステムの導入にかかる負担をできる限り削減するということを目的として Web ベースでシステムの実装を行った。利用者は Web ページにアクセスするだけでシステムを利用することができるため、その目的は達成できているといえる。しかし、Web ベースシステムの利点を最大限に活かすには他の Web ベースのシステムとの連携や掲示板のような学習者間でのコミュニケーションが行える機能といった、Web ベースならではの機能を実装することが必要である。例えば他のシステムから本システムのページに対して、そのシステムで利用しているソースコードを送信するといったシステム間のテキスト送受信の仕組みが考えられる。

- インデントのエラーを自動で調整する機能を実装すること

  初学者にとってインデントのミスは誰しも犯しがちなミスである。インデントには様々なルールがあり、正解が一意に定まっているわけではないため、初めから正しいインデントを行うことやインデントの修正を行うことはプログラミングに慣れるまで難しい。そこでインデントのミスを自動的に修正してくれる機能が欲しいという意見があった。正しいインデントにするために追加したり減らしたりするスペースの数を特定することはできているため、

実際にそのスペースの増減を反映したソースコードを出力することは可能であると考えられる。これは本システムにおける構文解析によってエラーを出力する機能とは別の機能として実装することになり、システム自体に複数の機能を持たせることができるようになる。

- 調査項目の拡張性に関する評価が主観的であること

  本システムの目的として、新しく調査したいエラー項目ができた際の調査項目の拡張しやすさというものがあった。しかしソースコードの解析結果から情報を抜き出すには Haskell や language-c-quote の知識が必要であり、調査項目の拡張性に関しての試用実験を行うことが困難であった。そのため拡張性に関しての評価は筆者の主観的なものでしかない。そこで、本学には Haskell を学ぶ講義があるためその講義内で調査を行うコードを記述する課題を出したり、知識がなくても調査のイメージがつかめるように分かりやすさを重視したビジュアルプログラミングを利用したりといった方法が考えられる。

謝辞

本研究においてご指導を承りました香川考司先生に心から感謝の意を表します。そして、本研究のシステム開発や試用評価にご協力いただいた本研究室の藤井陸さん、平西宏彰さん、大山哲平さん、濱井柚任さん、鎗分汰地さん、MUHAMMAD SULAIMI BIN SABUDIN さん、木野誠真さん、田中舜さん、軒原峻維さん、SHEN WEI さんに深く感謝いたします。

参考文献

[1] サイボウズ・ラボユース Ucida Kota, “C-Helper GitHub”,
https://github.com/uchan-nos/c-helper (閲覧日:2023 年 2 月)

[2] 内田 公太, 権藤 克彦, “C 言語初学者向けツール C-Helper の現状と展望”,
第 54 回プログラミングシンポジウム予稿集, pp.153-160, 2013

[3] 内田 公太, 権藤 克彦, “C 言語初学者向けツール C-Helper の予備評価”,
電子情報通信学会技術研究報告=IEICE technical report: 信学技報, 巻 113, 号 159, pp.67-72, 2013

[4] 島川 大輝, 香川 考司, “C-Helper を用いた Web ベースの C 言語開発環境の構築”,
教育システム情報学会第 40 回全国大会 (JSiSE2015) 講演論文集, A3-1, 2015.

[5] 木村 光星, 香川 考司, “構文解析を用いた C 言語指導コメント支援システムの構築”,
教育システム情報学会 (JSiSE) 2018 年度 第 4 回研究会, 2018

[6] “GHCup”,
https://www.haskell.org/ghcup/ (閲覧日:2023 年 2 月)

[7] “The Haskell Cabal”,
https://www.haskell.org/cabal/ (閲覧日:2023 年 2 月)

[8] “Haskell Language Server”,
https://github.com/haskell/haskell-language-server (閲覧日:2023 年 2 月)

[9] “language-c-quote”,
https://hackage.haskell.org/package/language-c-quote (閲覧日:2023 年 2 月)

[10] “wai”,
https://hackage.haskell.org/package/wai-3.2.3 (閲覧日:2023 年 2 月)

[11] “warp”,

https://hackage.haskell.org/package/warp-3.3.23 (閲覧日:2023 年 2 月)

[12] “Language.C.Syntax”,

https://hackage.haskell.org/package/language-c-quote-0.13/docs/Language-C-Syntax.html (閲覧日:2023 年 2 月)

[13] Koji Kagawa, “インデンテーションについての約束事”

https://guppy.eng.kagawa-u.ac.jp/2022/Programming/indentation.html

[14] “Generics”,

https://wiki.haskell.org/Generics (閲覧日:2023 年 2 月)