ソフトウェア演習5　レポート2

提出日　2015/11/24

13122029　高瀬正典

(1)作成したプログラムの設計情報

(1-1)全体構成

関数 p_error:	1度だけエラーメッセージを表示し、ファイルを閉じる。
関数 PrittyPrint:	引数に現在の token の種類、行末を改行するか否か、最初の字句か否かを入力し、現在の語を表示する。
関数 once_enter:	一度だけ改行する。
関数 prase:	はじめに1語読み込み、関数 p_program を起動する。
関数 p_program:	各字句を関数 PrittyPrint で表示し、関数 block を起動する。
関数 block:	関数 ValiableDeclaration、関数 SubProgram、関数 CompositeSentence を起動する。
関数 ValiableDeclaration:	各字句を関数 PrittyPrint で表示し、関数 ValiableSequence を起動する。
関数 ValiableSequence:	各字句を関数 PrittyPrint で表示する。
関数 Type:	関数 NomalType、関数 ArrayType を起動する。
関数 NomalType:	各字句を関数 PrittyPrint で表示する。
関数 ArrayType:	各字句を関数 PrittyPrint で表示し、関数 NomalType を起動する。
関数 FormalArgument:	各字句を関数 PrittyPrint で表示し、関数 ValiableSequence、関数 Type を起動する。
関数 SubProgram:	各字句を関数 PrittyPrint で表示し、関数 FomalArgument、関数 ValiableDeclaration、関数 CompositeSentence を起動する。
関数 ComositeSentence:	各字句を関数 PrittyPrint で表示し、関数 Sentence、関数 once_enter を起動する。
関数 Sentence:	関数 AssignmentSentence、関数 DivergencySentence、関数 LoopSentence、関数 CallSentence、関数 InputSentence、関数 OutputSentence、関数 CompositeSentence を起動する。
関数 AssignmentSentence:	各字句を関数 PrittyPrint で表示し、関数 Value、関数 Fomula を実行する。
関数 DivergencySentence:	各字句を関数 PrittyPrint で表示し、関数 Fomula、関数 Sentence、関数 once_enter を実行する。
関数 LoopSentence:	各字句を関数 PrittyPrint で表示し、関数 Fomula、関数 Sentence を実行する。
関数 CallSentence:	各字句を関数 PrittyPrint で表示し、関数 FomulaSequence を実行する。
関数 FomulaSequence:	各字句を関数 PrittyPrint で表示し、、関数 Fomula を実行する。
関数 InputSentence:	各字句を関数 PrittyPrint で表示し、関数 Value を実行する。
関数 OutputSentence:	各字句を関数 PrittyPrint で表示し、関数 OutputSpecification を実行する。
関数 OutputSpecification:	各字句を関数 PrittyPrint で表示し、関数 Fomula を実行する。
関数 Value:	各字句を関数 PrittyPrint で表示し、関数 Fomula を実行する。
関数 Fomula:	関数 SimplicityFomula、関数 RelationalOperator を実行する。
関数 SimplicityFomula:	各字句を関数 PrittyPrint で表示し、関数 Kou、関数 AddOperator を実行する。
関数 Kou:	関数 Factor、関数 MultiplOperator、関数 Kou を実行する。
関数 Factor:	各字句を関数 PrittyPrint で表示し、関数 Value、関数 Constant、関数 Fomula、関数 Factor、関数 NomalType を実行する。
関数 Constant:	各字句を関数 PrittyPrint で表示する。
関数 MultiplOperator:	各字句を関数 PrittyPrint で表示する。
関数 AddOperator:	各字句を関数 PrittyPrint で表示する。
関数 RelationalOperator:	各字句を関数 PrittyPrint で表示する。

(1-2)各モジュールごとの構成

グローバル変数	int token:	関数 scan の返り値を保存する。
静的変数	int tabnum:	出力するタブ数の変数。
	int once_enter_t:	一度しか改行しないための変数。
関数 p_error	静的変数 int once_end:	一度しかエラー文を出力しないための変数。
	仮引数　char *mes:	エラーメッセージ。
関数 PrittyPrint	int i:	キートークンとの照合用の for 変数。
	int tabnum_t:	タブ数の for 変数。
	仮引数　int PPtoken:	関数 scan の返り値。
	仮引数　int NextLine:	行末に改行するか否かの判定用変数。
	仮引数　int StartWord:	字句が行頭か否かの判定用変数。
	静的変数 int first:	最初の字句は改行も空白も挟まないための変数。
関数 ValiableSequence	仮引数 int top:	字句が先頭か否かの判別用変数。
関数 Value	仮引数 int top:	字句が先頭か否かの判別用変数。
関数 DivergencySentence	静的変数 int D_top:	字句が「else if」の場合のタブ数変化のための変数。

(1-3)各関数の外部仕様

関数 p_error　エラーメッセージを表示し、関数 end_scan によってファイルを閉じる。
2度以上ファイルを閉じるとエラーが発生するため静的変数 once_end によって一度しか実行されないようにした。
エラーの発生した行番号と仮引数の mes を表示する。

関数 PrittyPrint　仮引数として受け取った PPtoken に対応した数字や名前やキーワードを表示する。
関数 scan によって現在の語の情報が num_attr,string_attr,key[].keyword, signs[].keyword に保存されているので、PPtoken によって分岐した後に各字句を表示している。
仮引数 NextLine が 1 の場合は行末に改行が挿入され、そうでない場合は空白が挿入される。改行するとグローバル変数 once_enter が 1 となる。静的変数 first によって初めの字句は改行も空白も挿入されない。
仮引数 StartWord が 1 の場合はグローバル変数 tabnum の数だけ行頭にタブが挿入される。

関数 once_enter　グローバル変数 once_enter が 0 の場合のみ改行する。

関数 prase　最初にグローバル変数 token に関数 scan の返り値を代入する。その後関数 p_program を実行する。成功すれば P_SUCCESS を返す。

関数 p_program　最初の字句が「program」であれば行頭に表示し、行末に改行を挿入する。
そうでなければ関数 p_error にて「Keyword 'program' is not found」を表示する。
そして次の字句を token に代入する。
以下同様に「name」、「;」と行い、「;」の後は改行する。
関数 block を実行し、次の字句が「.」であれば P_SUCCESS を返す。

関数 block　関数 ValiableDeclaration または関数 SubProgram が成功している間はそれらを繰り返す。その後関数 CompositeSentence を行頭指定で実行し、成功すれば P_SUCCESS を返す。

関数 ValiableDeclaration　最初の字句が「var」かどうかを確認する。「var」であればグローバル変数 tabnum をインクリメントし、行末に改行を挿入して行頭に表示する。
その後また tabnum をインクリメントし、関数 ValiableSequence によって行頭に変数を表示する。その後「:」が続く限り変数を表示し続け、「;」であれば行末に改行を挿入し、ループから抜ける。tabnum を 2 デクリメントし P_SUCCESS を返す。

関数 ValiableSequence　最初の字句を確認し、「begin」か「procedure」なら P_FAIL を返す。字句が「name」でない場合も同様。以上の場合でなければ仮引数 top に対応して行頭か前の字句の後に字句を表示する。次の字句が「,」である限り次の字句を確認し、「begin」、「procedure」、「name」でなければ前の字句の後に表示する。「,」が途切れれば P_SUCCESS を返す。

関数 Type　関数 NomalType または関数 ArrayType を実行する。

関数 NomalType　字句が「integer」、「boolean」、「char」であれば前の字句の後に表示し、P_SUCCESS を返す。

関数 ArrayType　字句が順に「array」、「[」、「符号なし整数」、「]」、「of」であれば各々表示し、最後に関数 NomalType を実行する。成功すれば P_SUCCESS を返す。

関数 FomalArgument　最初の字句を確認し、「(」でなければ P_FAIL を返す。次に関数 ValiableSequence を実行し、前の字句の後に変数を表示する。次の字句が「:」であれば表示し、関数 Type を実行する。次の字句が「;」であれば関数 ValiableSequence、「:」、関数 type を繰り返し実行する。次の字句が「;」でなく、「)」の場合は表示して P_SUCCESS を返す。

関数 SubProgram　最初の字句を確認し、「procedure」でなければ P_FAIL を返す。tabnum をインクリメントし、行頭に表示する。その後名前であるかどうかの判別、関数 FomalArgument、「;」、関数 CompositeSentence を実行し、tabnum をデクリメントする。

関数 CompositeSentence　最初の字句を確認し、「begin」でなければ P_FAIL を返す。仮引数 begin_tab が 1 であれば tabnum をデクリメントする。これは前の字句が「else」、「do」、「then」の場合の処理である。その後「begin」を表示し、tabnum をインクリメントする。その後「;」の判別、関数 Sentence を実行し、tabnum をデクリメントする。begin_tab が 1 であればその後 tabnum をインクリメントし、P_SUCCESS を返す。

関数 Sentence　関数 AssignmentSentence、関数 DivergencySentence、関数 LoopSentence、関数 CallSentence、関数 InputSentence、関数 OutputSentence、関数 CompositeSentence の実行または字句が「return」もしくは「;」、「end」の判別を行う。

関数 AssignmentSentence　関数 Value によって行頭に字句を表示する。その後「:=」の判別、関数 Fomula の実行をする。

関数 DivergencySentence　最初の字句を確認し、「if」でなければ P_FAIL を返す。前の字句が「else」であればその後ろに表示し、そうでなければ行頭に表示する。静的変数 D_top はそのための変数である。その後各字句の判別、tabnum の調整を行う。

関数 LoopSentence　最初の字句を確認し、「while」でなければ P_FAIL を返す。その後各字句の判別、tabnum の調整を行う。

関数 CallSentence　最初の字句を確認し、「call」でなければ P_FAIL を返す。その後各字句の判別を行う。

関数 FomulaSequence　まず関数 Fomula の判別を行う。その後次の字句が「,」である限り繰り返す。

関数 InputSentence　最初の字句を確認し、「read」もしくは「readln」でなければ P_FAIL を返す。その後各字句の判別を行う。

関数 OutputSentence　最初の字句を確認し、「write」もしくは「writeln」でなければ P_FAIL を返す。その後各字句の判別を行う。

関数 OutputSpecification　最初の字句が文字列か式かを判別する。文字列であれば表示し、式であれば関数 Fomula、字句の判別を行う。

関数 Value　最初の字句が「name」でなければ P_FAIL を返す。仮引数 top で行頭に表示するか否かを決定する。その後各字句の判別を行う。

関数 Fomula　まず関数 SimplicityFomula を実行する。その後関数 SimplicityFomula と関数 RelationalOperator を共に失敗するまで繰り返す。

関数 SimplicityFomula　最初の字句が「+」もしくは「-」であれば表示し、関数 Kou を実行する。その後関数 AddOperator と関数 Kou を共に失敗するまで繰り返す。

関数 Kou　まず関数 Factor を実行する。その後関数 MultipleOperator と関数 Factor を共に失敗するまで繰り返す。

関数 Factor　各字句の判別を行う。

関数 Constant　字句が「number」、「false」、「true」、「string」であれば表示する。

関数 MultipleOperator　字句が「*」、「div」、「and」であれば表示する。

関数 AddOperator　字句が「+」、「-」、「or」であれば表示する。

関数 RelationalOperator　字句が「=」、「<>」、「<」、「<=」、「>」、「>=」であれば表示する。

・以前の課題で作成した関数

関数 scan　「"」の際に「'」「'」と認識されないようにした。空白や改行が起こった時に 0 を返さずに次の字句を読み込むようにした。他の機能は課題 1 レポートで記載済み

関数 get_linenum　課題 1 レポートで記載済み

関数 error_scan　課題 1 レポートで記載済み

関数 end_scan　課題 1 レポートで記載済み

(2)テスト情報
(2-1)テストデータ　送信日時　2015/11/24/03:16
sample021.mpl　sample13.mpl　sample22.mpl
sample022.mpl　sample14.mpl　sample23.mpl
sample023.mpl　sample14p.mpl　sample24.mpl
sample024.mpl　sample15.mpl　sample25.mpl
sample025.mpl　sample16.mpl　sample25t.mpl
sample026.mpl　sample17.mpl　sample26.mpl
sample11p.mpl　sample18.mpl　sample27.mpl
sample11pp.mpl　sample19p.mpl　sample28p.mpl
sample12.mpl　sample21.mpl　sample29p.mpl

111.mpl　183.mpl　274.mpl　335.mpl　438.mpl
115.mpl　189.mpl　278.mpl　348.mpl　78.mpl
132.mpl　199.mpl　287.mpl　351.mpl　82.mpl
154.mpl　205.mpl　293.mpl　382.mpl　86.mpl
157.mpl　255.mpl　303.mpl　397.mpl　90.mpl
160.mpl　260.mpl　304.mpl　398.mpl　91.mpl
163.mpl　264.mpl　316.mpl　429.mpl
174.mpl　273.mpl　324.mpl　430.mpl

(2-2)テスト結果　送信日時　2015/11/24/03:16
sample021_result.txt　sample13_result.txt　sample22_result.txt
sample022_result.txt　sample14_result.txt　sample23_result.txt
sample023_result.txt　sample14p_result.txt　sample24_result.txt
sample024_result.txt　sample15_result.txt　sample25_result.txt
sample025_result.txt　sample16_result.txt　sample25t_result.txt
sample026_result.txt　sample17_result.txt　sample26_result.txt
sample11p_result.txt　sample18_result.txt　sample27_result.txt
sample11pp_result.txt　sample19p_result.txt　sample28p_result.txt
sample12_result.txt　sample21_result.txt　sample29p_result.txt

111_result.txt　183_result.txt　274_result.txt　335_result.txt　438_result.txt
115_result.txt　189_result.txt　278_result.txt　348_result.txt　78_result.txt
132_result.txt　199_result.txt　287_result.txt　351_result.txt　82_result.txt
154_result.txt　205_result.txt　293_result.txt　382_result.txt　86_result.txt
157_result.txt　255_result.txt　303_result.txt　397_result.txt　90_result.txt
160_result.txt　260_result.txt　304_result.txt　398result.txt　91_result.txt
163_result.txt　264_result.txt　316_result.txt　429_result.txt
174_result.txt　273_result.txt　324_result.txt　430result.txt

(2-3)テストデータの十分性
　頭文字に「sample」のつくテストデータ 27 種類で正常な動作、エラーが発生する動作の確認を行った。これらでブラックボックステストは達成された。
　数字から始まるテストデータ 38 種類で全てのエラーメッセージを表示した。これでホワイトボックステストは達成された。なお、ファイル番号は pripri.c の実行された p_error のある行番号である。

(3)進捗状況

(3-1)事前計画

開始予定日	終了予定日	見積もり時間	作業内容
11/10	11/17	20	適当な改行、タブなしでの表示
11/18	11/22	20	改行、タブの実装及びバグの対処
11/23	11/23	5	レポートの作成

(3-2)事前計画の立て方についての前課題からの改善点

予定を建てる段階では何が必要なのかわからないので大雑把な予定とした。また、期限が遠いとあまりやる気が起こらない性格なので提出の 2 週間前から開始とした。

(3-3)実際の進捗状況

開始予定日	終了予定日	見積もり時間	作業内容
11/14	11/15	12	適当な改行、タブなしでの表示
11/18	11/18	4	バグの対処
11/20	11/22	10	改行、タブの実装
11/23	11/23	5	テストケースの作成
11/23	11/24	5	レポートの作成

(3-4)当初の事前計画と実際の進捗との差の原因

時間のかかる実験レポートと被ってしまったため、本課題は後回しになってしまった。結果として大きな遅れが発生してしまった。また、終盤でプログラムが完成して油断してしまった。テストケース作成に予想外に時間がかかってしまった。