ソフトウェア演習Ⅴ 課題4

15122013 尾持涼介

提出日：2018年2月9日

1　作成したプログラムの設計情報

1.1　全体構成

各ファイルで記述した関数は以下の通りである。

- main.c
  - main 関数
  - void error(char *mes)
- scan.c
  - int init_scan(char *filename)
  - int keyword_search(char *string)
  - int scan()
  - int get_linenum()
  - void end_scan()
- prettyprinter.c
  - int parse_program()
  - int block(int label)
  - int var_decl()
  - int var_names()
  - int type()
  - int ar_type()
  - int sub_decl()
  - int form_para()
  - int fukugou(int label)
  - int statement(int label)
  - int bunki(int label)
  - int kurikaeshi(int label)
  - int call_st(int label)
  - int exp_narabi()
  - int dainyu(int label)
  - int var()
  - int shiki()
  - int simple()
  - int kou()
  - int inshi()
  - int input_st(int label)
  - int output_st(int label)
  - int shitei()
  - int get_inlabel()
  - void lib()

- crossreferencer.c
  - void init_idtab()
  - struct ID *search_globalidtab(char *np)
  - struct ID *search_localidtab(char *np)
  - int globalid_def()
  - int procedure_def()
  - int globalid_ref(char *np)
  - int localid_def()
  - int localid_ref(char *np)
  - int type_mem(struct ID *p)
  - void joint_localtoglobal()
  - void release_idtab(struct ID *p)

次に、各関数の呼び出し関係、データ参照関係について述べる。

- main.c
  - main 関数内
    - init_scan() 関数を呼び出し
    - scan() 関数を呼び出し
    - parse_program() 関数を呼び出し
    - end_scan() 関数を呼び出し
  - error 関数内
    - get_linenum() 関数を参照
- scan.c
  - init_scan() 関数内
    - init_idtab() 関数を呼び出し
  - scan() 関数内
    - keyword_search() 関数を呼び出し
    - error() 関数を呼び出し
  - end_scan() 関数内
    - release_idtab() 関数を呼び出し
- prettyprinter.c
  - parse_program() 関数内
    - error() 関数を呼び出し
    - scan() 関数を呼び出し
    - get_inlabel() 関数を呼び出し
    - block() 関数を呼び出し
    - lib() 関数を呼び出し
  - block() 関数内
    - var_decl() 関数を呼び出し
    - sub_decl() 関数を呼び出し

    - fukugou() 関数を呼び出し
- var_decl() 関数内
    - scan() 関数を呼び出し
    - var_names() 関数を呼び出し
    - error() 関数を呼び出し
    - type() 関数を呼び出し
    - globalid_def() 関数を呼び出し
    - localid_def() 関数を呼び出し
- var_names() 関数内
    - error() 関数を呼び出し
    - scan() 関数を呼び出し
- type() 関数内
    - scan() 関数を呼び出し
    - ar_type() 関数を呼び出し
    - error() 関数を呼び出し
- ar_types() 関数内
    - scan() 関数を呼び出し
    - error() 関数を呼び出し
- sub_decl() 関数内
    - scan() 関数を呼び出し
    - error() 関数を呼び出し
    - procedure_def() 関数を呼び出し
    - form_para() 関数を呼び出し
    - var_decl() 関数を呼び出し
    - fukugou() 関数を呼び出し
    - joint_localtoglobal() 関数の呼び出し
- form_para() 関数内
    - scan() 関数を呼び出し
    - var_names() 関数を呼び出し
    - error() 関数を呼び出し
    - type() 関数を呼び出し
    - localid_def() 関数を呼び出し
- fukugou() 関数内
    - error() 関数を呼び出し
    - scan() 関数を呼び出し
    - statement() 関数を呼び出し
- statemen() 関数内
    - dainyu() 関数を呼び出し
    - bunki() 関数を呼び出し
    - kurikaeshi() 関数を呼び出し
    - scan() 関数を呼び出し
    - call_st() 関数を呼び出し
    - input_st() 関数を呼び出し

    - output_st() 関数を呼び出し
    - fukugou() 関数を呼び出し
- bunki() 関数内
    - scan() 関数を呼び出し
    - shiki() 関数を呼び出し
    - error() 関数を呼び出し
    - get_inbabel() 関数を呼び出し
    - statement() 関数を呼び出し
- kurikaeshi() 関数内
    - scan() 関数を呼び出し
    - shiki() 関数を呼び出し
    - error() 関数を呼び出し
    - get_inlabel() 関数を呼び出
    - statement() 関数を呼び出し
- call_st() 関数内
    - scan() 関数を呼び出し
    - error() 関数を呼び出し
    - globalid_ref() 関数を呼び出し
    - localid_ref() 関数を呼び出し
    - exp_narabi() 関数を呼び出し
- exp_narabi() 関数内
    - shiki() 関数を呼び出し
    - error() 関数を呼び出し
    - get_inlabel() 関数を呼び出し
- dainyu() 関数内
    - var() 関数を呼び出し
    - error() 関数を呼び出し
    - scan() 関数を呼び出し
    - shiki() 関数を呼び出し
- var() 関数内
    - error() 関数を呼び出し
    - globalid_ref() 関数を呼び出し
    - localid_ref() 関数を呼び出し
    - scan() 関数を呼び出し
    - shiki() 関数を呼び出し
- shiki() 関数内
    - simple() 関数を呼び出し
    - scan() 関数を呼び出し
    - get_inlabel() 関数を呼び出し
- simple() 関数内
    - kou() 関数を呼び出し
    - error() 関数を呼び出し
    - scan() 関数を呼び出し

        * error() 関数を呼び出し
    - kou() 関数内
        * inshi() 関数を呼び出し
        * error() 関数を呼び出し
        * scan() 関数を呼び出し
    - inshi() 関数内
        * var() 関数を呼び出し
        * scan() 関数を呼び出し
        * shiki() 関数を呼び出し
        * error() 関数を呼び出し
        * inshi() 関数を呼び出し
    - input_st() 関数内
        * scan() 関数を呼び出し
        * var() 関数を呼び出し
        * error() 関数を呼び出し
    - output_st() 関数内
        * scan() 関数を呼び出し
        * shitei() 関数を呼び出し
        * error() 関数を呼び出し
    - shitei() 関数内
        * scan() 関数を呼び出し
        * get_inlabel() 関数を呼び出し
        * shiki() 関数を呼び出し
        * error() 関数を呼び出し

- crossreferencer.c 内
    - globalid_def() 関数内
        * error() 関数を呼び出し
        * get_linenum() 関数を呼び出し
        * type_mem() 関数を呼び出し
    - procedure_def() 関数内
        * error() 関数を呼び出し
        * get_linenum() 関数を呼び出し
        * type_mem() 関数を呼び出し
    - globalid_ref() 関数内
        * search_globalidtab() 関数を呼び出し
        * get_linenum() 関数を呼び出し
        * error() 関数を呼び出し
    - lobalid_def() 関数内
        * error() 関数を呼び出し
        * type_mem() 関数を呼び出し
        * get_linenum() 関数を呼び出し
    - localid_ref() 関数内

        * search_localidtab() 関数を呼び出し
        * error() 関数を呼び出し
        * get_linenum() 関数を呼び出し
        * search_globalidtab() 関数を呼び出し
    - type_mem() 関数内
        * error() 関数を呼び出し
    - joint_localtoglobal 関数内
        * error() 関数を呼び出し
    - release_idtab() 関数内
        * release_idtab() 関数を呼び出し (再帰呼出し)
        * init_idtab() 関数を呼び出し

1.2 各モジュールごとの構成

出力文等で文字列が出てきたときに、その文字列を格納するための構造体を以下の図 1 のように定義した。

```
1 struct STRING{
2 char *string;
3 int label;
4 struct STRING *next;
5 };
```

図 1: 文字列を格納するための構造体

なお、ここで string は格納する文字列、label はその文字列を定義するためのラベルの番号、next は次の要素へのポインタである。そしてプログラムの末尾を表す「end.」が出てきた後にまとめて出力している。
また、コード生成は適切な場所で fprintf を使って生成している。
次に使用した大域変数とその意味について述べる。

- main.c 内
    - 課題 3 のレポートで説明済み
- scan.c 内
    - int cbuf：課題 2 のレポートで説明済み
    - int line_cnt：課題 2 のレポートで説明済み
    - int newline：課題 2 のレポートで説明済み
    - char string_attr[MAXSTRSIZE]：課題 2 のレポートで説明済み
    - int num_attr：課題 2 のレポートで説明済み
    - FILE *fp：課題 2 のレポートで説明済み
    - FILE *fpw：出力するファイルへのポインタ
- prettyprinter.c 内
    - int arraynum：配列型の要素数を格納する変数
    - int typenum：宣言された変数の型を表す整数を格納する。
    - int paraflag：今調べている変数が仮引数かどうかを記憶する変数 (1 なら仮引数、0 ならその他)。
    - int gorl：調べている変数が大域変数か局所変数かを記憶する変数 (1 なら局所変数、0 なら大域変数)。

- int arraytype：配列の要素の型を表す整数を格納する。
- int shikitype :「式」の型を表す整数を格納する。
- int shikiarraysize :「式」が配列型であるときに要素数を記憶する変数。
- int shikiarraytype :「式」が配列型であるときにその要素の型を記憶する変数。
- int vartype :「変数」の型を表す整数を格納する。
- int vararraysize :「変数」が配列型であるときに要素数を記憶する変数。
- int vararraytype :「変数」が配列型であるときにその要素の型を記憶する変数。
- int koutype :「項」の型を表す整数を格納する。
- int kouarraysize :「項」が配列型であるときに要素数を記憶する変数。
- int kouarraytype :「項」が配列型であるときにその要素の型を記憶する変数。
- int inshitype :「因子」の型を表す整数を格納する。
- int inshiarraysize :「因子」が配列型であるときに要素数を記憶する変数。
- int inshiarraytype :「因子」が配列型であるときにその要素の型を記憶する変数。
- int arrayflag：その前に出てきた変数が配列型かどうかを表す変数 (1 なら配列型、0 なら標準型)
- int paranum：仮引数の個数を記憶する変数。
- int expnum：式の並びにおいて式の個数を記憶する変数。
- int inlab_num：ラベル番号を格納する変数。
- int adflag：変数からアドレスを取り出すべきか値を取り出すべきかを表す変数 (1 ならアドレスを、0 なら値を取り出す)。
- int callflag：手続き呼出し文の引数での処理であることを示す変数。
- int eorv：手続き呼出し文の引数が変数単体であるのか、複数の項からなる式であるのかを表す変数 (1 なら式、0 なら変数)
- struct NAME *names：変数宣言部、仮引数部における変数名の並びを記憶する線形リストの先頭要素を指すポインタ。
- struct TYPE *paratype：仮引数の型を記憶する線形リストの先頭要素を指すポインタ。
- struct ID *searchp：局所変数・大域変数のリストのうち、探索した変数名の要素を指すポインタ。
- char token_str[NUMOFTOKEN+1]：トークン名 (キーワード名) が格納された配列。
- struct STRING *stringhead：文字列を格納する線形リストの先頭要素を指すポインタ。
- struct STRING *stringtail：文字列を格納する線形リストの末尾要素を指すポインタ。
- struct NAME *para：仮引数を格納する線形リストの先頭要素を指すポインタ

- crossreferencer.c 内
  - struct ID *globalidroot：大域変数を格納する二分探索木の先頭要素を指すポインタ。
  - struct ID *localidroot：局所変数を格納する二分探索木の先頭要素を指すポインタ。
  - char *typename[NUMOFTYPE+1]：型名を格納した配列。

次に各関数内で定義した変数とその意味について述べる

- scan.c
  - init_scan() 関数内
    - char *newfile：元のファイル名の拡張子を.csl に変えたものを格納するポインタ。
    - int lebgth：元のファイル名の長さを表す変数。
    - int i：ループ変数

- ikeyword_search() 関数内：課題 1 で記載済み
- scan() 関数内
  - int i：課題 1 で記載済み

- prettyprinter.c
  - parse_program() 内
    - int label；プログラムが最初に実行する部分を表すラベルを表す変数。
    - struct STRING *p,*q：文字列を格納している線形リストを辿るためのポインタ。
  - var_decl() 内
    - struct NAME *p：変数名が格納された線形リストを惰どるためのポインタ。
  - var_names() 内
    - struct NAME *p：names に追加する要素を指すポインタ。
    - struct NAME *q：作成した要素を names に挿入するために names をたどるポインタ。
    - char *cp：線形リストに格納する変数名を格納するポインタ。
  - type() 内
    - int a：ar_type() 関数からの返り値を記憶する変数。
  - sub_decl() 関数内
    - struct NAME *p, *q：仮引数を格納したリストを辿るためのポインタ
    - int flag：その副プログラムに引数があるかどうかを表す変数。
  - form_para() 関数内
    - struct NAME *n,*m：for 文で names を辿るためのポインタ。
    - struct NAME *l：仮引数を格納するリストに新たに加える要素を指すポインタ。
    - struct TYPE *p：新たな仮引数の型情報を格納するポインタ。
    - struct TYPE *q：p がさす要素を paratp につなげるためのポインタ。
    - char *cp：仮引数を格納するリストに新たに加える変数名を格納するポインタ。
  - bunki() 関数内
    - int flag：課題 2 で記載済み。
    - int label1,label2：ラベル番号を表す変数。
  - kurikaeshi() 関数内
    - int flag：課題 2 で記載済み
    - int label1,label2：ラベル番号を表す変数。
  - call_st() 関数内
    - struct TYPE *q：呼び出し文で呼び出された副プログラムの仮引数の型情報を記憶する構造体の先頭要素を指すポインタ。
    - char *callname：呼び出す副プログラムの名前を表すポインタ。
  - exp_narabi() 関数内
    - struct TYPE *p：式の型情報を記憶する構造体の先頭要素を指すポインタ。
    - int label；ラベル番号を表す変数。
    - struct STRING *p；文字列を格納するリストに新たに加える要素を指すポインタ。
    - char *cp：文字列を格納するリストに新たに加える文字列 (ここでは、領域を確保するために 0 を格納) を格納するポインタ。
  - dainyu() 関数内

    - int type1：左辺値の型を記憶する変数。
    - int type2：代入する式の型を記憶する変数。
  - var() 関数内
    - int arraytype：変数名が配列型の場合その要素の型を記憶する変数。
    - int arraysize：変数名が配列型の場合その要素数を記憶する変数。
  - shiki() 関数内
    - int type1, type2：それぞれの直前で出現した単純式の型を記憶する変数。
    - int arraysize1、arraysize2：それぞれの直前に出現した単純式が配列型だった場合にその要素数を記憶する変数。
    - int arraytype1、arraytype2：それぞれの直前に出現した単純式が配列型だった場合にその要素の型を記憶する変数。
    - int opr：関係演算子を記憶する変数。
    - int label1,label2：ラベル番号を記憶する変数。
  - simple() 関数内
    - int flag：最初の項の前に＋かーがあるかどうかを示す変数 (0 ならない、1 ならある)。
    - int minusflag：最初の項の前に-があるかどうかを示す変数 (0 ならない、1 ならある)
    - int type1, type2：それぞれの直前に出現した項の型を記憶する変数。
    - int kahou：加法演算子を記憶する変数。
    - int type1, type2：それぞれの直前に出現した因子の型を記憶する変数。
    - int jouhou：情報演算子を記憶する変数。
  - inshi() 関数内
    - int type1：因子の最初に標準型がある場合にその型を記憶する変数。
  - input_st() 関数内
    - int op：入力文を表す命令が「read」であるか「readln」であるかを記憶する変数。
  - output_st() 関数内
    - int op：出力文を表す命令が「write」であるか「writeln」であるかを記憶する変数。
  - shitei() 関数内
    - struct STRING *p：文字列を格納するリストに新たに加える要素を指すポインタ。
    - char *cp：文字列を格納するリストに新たに加える文字列を指すポインタ。
    - int label：ラベル番号を表す変数。

- crossreferencer.c 内
  - search_globalidtab() 関数内
    - struct ID *p：for 文による探索のために globalroot を辿るポインタ。
  - search_localidtab() 関数内
    - struct ID *p：for 文による探索のために localroot を辿るポインタ。
  - globalid_def() 関数内
    - struct ID *new；新しく globalroot につなげる要素を指すポインタ。
    - struct ID *p；globalidroot を辿るポインタ。
    - struct NAME *np：names のうち globalidroot に登録する要素を指すポインタ。
    - struct NAME *nq：for 文で names を辿るためのポインタ。
  - procedure_def() 関数内

    - struct ID *new；新しく globalroot につなげる要素を指すポインタ。
    - struct ID *p；globalidroot を辿るポインタ。
    - char *cp：新しく加える手続き名を指すポインタ。
  - globalid_ref() 関数内
    - struct ID *p：探索した要素を指すポインタ。
    - struct LINE *next：新たに出現した行を格納すべき要素を指すポインタ。
    - struct LINE *prev：新たに出現した行を格納すべき要素の 1 つ前を指すポインタ。
    - struct LINE *m：新たに付け加える行番号を格納した要素を指すポインタ。
  - localid_def() 関数内
    - struct ID *new；新しく localroot につなげる要素を指すポインタ。
    - struct ID *p；locaidroot を辿るポインタ。
    - struct NAME *np：names のうち globalidroot に登録する要素を指すポインタ。
    - struct NAME *nq：for 文で names を辿るためのポインタ。
    - char *cp：その局所変数が定義されている手続き名を格納するポインタ。
  - localid_ref() 関数内
    - struct ID *p：探索した要素を指すポインタ。
    - struct LINE *next：新たに出現した行を格納すべき要素を指すポインタ。
    - struct LINE *prev：新たに出現した行を格納すべき要素の 1 つ前を指すポインタ。
    - struct LINE *m：新たに付け加える行番号を格納した要素を指すポインタ。
  - type_mem() 関数内
    - struct TYPE *q：新しく追加する型情報を格納した要素を指すポインタ。
    - struct TYPE *r：新しく追加する型が配列型の時の要素の型情報を格納するポインタ。
    - struct TYPE *pt：仮引数の型情報を記憶するときに paratype を辿るポインタ。
  - joint_localtoglobal() 関数内
    - struct ID *x：globalroot につなげた要素を指すポインタ
    - struct ID **p：globalidroot を辿るためのポインタ。
    - struct ID **q：localidroot を辿るためのポインタ

1.3 各関数の外部 (入出力) 仕様

ここでは、各関数の機能、引数と返り値等について説明する。

1.3.1 main.c 内で記述されている関数

- main 関数：課題 3 で記載済み。
- void error(char *mes)：課題 2 で記載済み

1.3.2 scan.c 内で記述されている関数

- int init_scan(char *filename)

  機能 初期化関数。指定されたファイルを読込形式で open し、指定されたファイルの拡張子を変更して書き込み形式で open する。読み込むファイルの最初のトークンを読み込んでおく。

  引数 char *filename：読み込みファイル名

  参照・変更する大域変数 line_cnt,newline,fp,fpw,cbuf

- int keyword_search(char *string)：課題 1 で記載済み
- int scan():課題 2 で記載済み
- int get_linenum()：課題 1 で記載済み
- void end_scan()：課題 3 で記載済み

1.3.3 prettyprinter.c 内で記述した関数

なお、以下で NORMAL と ERROR はそれぞれ prettyprinter.c 内で 0、1 と定義されている。

- int parse_program()

  機能 プログラムを解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  参照・変更する大域変数 token,fpw, stringhead

- int block(int label)

  機能 ブロックを解析する関数

  引数 int label：プログラムが最初に実行する場所を表すラベル番号

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  参照する大域変数 token,fpw

- int var_decl

  機能 変数宣言部を解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token

  参照する大域変数 gorl、token,fpw,names,typenum,procname

- int var_names()

  機能 変数名の並びを解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token, names

  参照する大域変数 string_attr、token

- int type()

  機能 型を解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  参照・変更する大域変数 token,typenum

- int ar_type()

  機能 配列型を解析する関数

  引数 なし

返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

変更する大域変数 token、typenum、arraynum,arraytype

参照する大域変数 token、string_attr、num_attr

- int sub_decl()

  機能 副プログラム宣言を解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、gorl、localidroot、procname、para

  参照する大域変数 token、string_attr、fpw、para

- int form_para()

  機能 仮引数部を解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、paratype、paraflag、para

  参照する大域変数 names、token、typenum

- int fukugou(int label)

  機能 複合文を解析する関数

  引数 int label：繰り返し文の終わりを表すラベル番号

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  参照・変更する大域変数 token

- int statement(int label)

  機能 文を解析する関数

  引数 int label：繰り返し文の終わりを表すラベル番号

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  参照・変更する大域変数 token、fpw

- int bunki(int label)

  機能 分岐文を解析する関数

  引数 int label：繰り返し文の終わりを表すラベル番号なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  参照・変更する大域変数 token、indentnum

  参照する大域変数 fpw

- int kurikaeshi(int label)

  機能 繰り返し文を解析する関数

  引数 int label：繰り返し文の終わりを表すラベル番号

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、

  参照する大域変数 shikitype、token、fpw

- int call_st(int label)

  機能 手続き呼び出し文を解析する関数

  引数 int label：繰り返し文の終わりを表すラベル番号

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、paranum、expnum、adflag、callflag

  参照する大域変数 gorl、token、string_attr、procname、searchp、paranum、expnum

- int exp_narabi()

  機能 式の並びを解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、expnum、eorv、stringhead、stringtail

  参照する大域変数 token、searchp、shikitype、eorv、fpw、stringhead

- int dainyu(int label)

  機能 代入文を解析する関数

  引数 int label：繰り返し文の終わりを表すラベル番号

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、adflag

  参照する大域変数 vartype、shikitype、token、fpw

- int var()

  機能 変数を解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、vartype

  参照する大域変数 gorl、token、string_attr、searchp、vartype、shikitype、fpw、adflag、inputglag

- int shiki()

  機能 式を解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、shikitype

  参照する大域変数 simpletype、token、simplearraysize、simplezrraytype、fpw、arrayflag、eorv

- int simple()

  機能 単純式を解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、simpletype、simplearraytype、simplearraysize、eorv

  参照する大域変数 token、koutype、kouarraytype、kouarraysize、fpw、arrayflag

- int kou()

機能 項を解析する関数

引数 なし

返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

変更する大域変数 token、koutype、kouarraysize、kouarraytype、eorv

参照する大域変数 inshitype、token 、fpw、callflag、arrayflag

- int inshi()

  機能 因子を解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、inshitype、inshiarraysize、inshiarraytype、arrayflag

  参照する大域変数 token、、fpw、vartype、vararraysize、vararraytype、shikitype、shikiarraysize、shikiarraytype、inshitype

- int input_st(int label)

  機能 入力文を解析する関数

  引数 int label：繰り返し文の終わりを表すラベル番号

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、adflag、inputflag

  参照する大域変数 token、vartype、fpw

- int output_st(int label)

  機能 出力文を解析する関数

  引数 int label：繰り返し文の終わりを表すラベル番号

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  参照・変更する大域変数 token、fpw

- int shitei()

  機能 出力指定を解析する関数

  引数 なし

  返り値 エラーがあれば ERROR を、なければ NORMAL を返す。

  変更する大域変数 token、stringhead、stringtail

  参照する大域変数 shikitype、token、fpw

- get_inlabal()

  機能 ラベルを確保する関数

  引数 なし

  返り値 確保したラベル番号

  参照する大域変数 inlab_num

- voidlib()

  機能 ライブラリ部のコード生成を行う関数

  引数・返り値 なし

  参照する大域変数 fpw

1.3.4 crossreferecer.c 内で記述した関数

- void init_idtab()

  機能 大域変数、局所変数を格納するそれぞれの二分探索木を初期化する。

  引数・返り値 なし

  参照・変更する大域変数 globalidroot、localidroot

- struct ID *search_globalidtab(char *np)

  機能 指定された変数名・手続き名が大域変数を格納する二分探査木の中に存在するか探索する。

  引数 char *np：探索する変数名・手続き名

  返り値 指定された変数名・手続き名が二分探索木の中に存在すればその要素を指すポインタが、存在しなければ NULL を返す。

  参照する大域変数 globalidroot

  変更する大域変数 なし

- struct ID *search_localidtab(char *np)

  機能 指定された変数名が局所変数を格納する二分探索木の中に存在するか探索する。

  引数 char *np：探索する変数名

  返り値 指定された変数名が二分探索木の中に存在すればその要素を指すポインタが、存在しなければ NULL を返す。

  参照する大域変数 localidroot

  変更する大域変数 なし

- int globalid_def()

  機能 新たに宣言された大域変数とその型情報、宣言された行番号を globalidroot に格納する。

  引数 なし

  返り値 エラーがあれば ERROR を、無ければ NORMAL を返す。

  参照・変更する大域変数 globalidroot、names

- int procedure_def()

  機能 新たに宣言された手続き名とその仮引数の型情報、宣言された行番号を globalidroot に格納する。

  引数 なし

  返り値 エラーがあれば ERROR を、無ければ NORMAL を返す。

  参照する大域変数 procname、globalidroot

  変更する大域変数 globalidroot

- int globalid_ref(char *np)

  機能 宣言済みの大域変数が使用された場合に、その行番号を globalidroot の該当する要素に追加する。

  引数 char *np：使用された変数名・手続き名

  返り値 エラーがあれば ERROR を、無ければ NORMAL を返す。

  参照・変更する大域変数 searchp、globalidroot

- int localid_def()

  機能 新たに宣言された局所変数とその型情報、その変数が宣言された副プログラムの手続き名、宣言された行番号を globalidroot に格納する。

引数 なし

返り値 エラーがあれば ERROR を、無ければ NORMAL を返す。

変更する大域変数 localidroot、names

参照する大域変数 localidroot、names、procname、paraflag

- int localid_ref(char *np)

  機能 宣言済みの大域変数が使用された場合に、その行番号を globalidroot の該当する要素に追加する。

  引数 char *np：使用された変数名・手続き名

  返り値 エラーがあれば ERROR を、無ければ NORMAL を返す。

  参照・変更する大域変数 searchp、localidroot

- int type_mem(struct ID *p)

  機能 型情報をリストに格納する。

  引数 struct ID *p：リストの型情報を格納すべき要素を指すポインタ。

  返り値 エラーがあれば ERROR を、無ければ NORMAL を返す。

  参照する大域変数 typenum、arraytype、arraynum、paratype

  変更する大域変数 paratype

- void joint_localtoglobal()

  機能 局所変数用の二分探索木を大域変数用の二分探索木につなげる。

  引数・返り値 なし

  参照する大域変数 globalidroot、localidroot

  変更する大域変数 globalidroot

- void release_idtab()

  機能 大域変数用二分探索木の領域を解放する。

  引数・返り値 なし

  参照・変更する大域変数 globalidroot

2 テスト情報

2.1 テストデータ・テスト結果

私はまず、ブラックボックステストとして配布されたテストデータである、sample11.mpl、sample11p.mpl、sample11pp.mpl、sample12.mpl、sample13.mpl、sample14.mpl、sampl15.mpl、sample15a.mpl、sample16.mpl、sample17.mpl、sample18.mpl、sample19p.mpl、sample21.mpl、sample2a.mpl、sample22.mpl、sample23.mpl、sample24.mpl、sampl25.mpl、sample25t.mpl、sample26.mpl、sample27.mpl、sample28p.mpl、sample29p.mpl、sample31p.mpl、sample33p.mpl、sample34.mpl、sample35.mpl についてテストを行った。

配布されたテストデータによるテスト結果 (CSL ファイルとシミュレータによる実行結果) ついてはメールにより提出する。なお、CSL ファイルについては「(テストプログラム名).csl」、シミュレータによる実行結果については「(テストプログラム名).png」としている (テストプログラム名の.mpl は省略)。それらをまとめて「kadai4-test.zip」というファイルにまとめて圧縮して提出する。なお、sample21 から 23 については、何も出力しないファイルであるため、シミュレータによる実行結果はこの圧縮ファイルには含めていない (実際に何も出力されなかった。また、sample31p と sample2a については、実行結果からわかる通り、正しい動作をしなかった。テスト情報を添付したメールの送信日時は 2 月 9 日 16 時 21 分である。

2.2 テストデータの十分性

上記のサンプルプログラムを実行することにより、エラーメッセージが出る場合を除く、テキストに書かれている全ての命令が実行されている。よって、このテストデータは十分であると言えると判断した。

3 本課題を行うための事前計画 (スケジュール) と実際の進捗状況

3.1 事前計画 (スケジュール)

事前計画は以下の表 1 のように立てた。

表 1: 課題 4 における事前計画

開始予定日	終了予定日	見積もり時間	作業内容
12月26日	12月26日	1	スケジュールを立てる
12月27日	12月30日	5	配布資料・サンプルプログラムを熟読する
12月31日	1月2日	3	コンパイラのテキストを熟読する
1月3日	1月7日	5	コード生成部の概略設計
1月8日	1月16日	10	コード生成部の作成
1月17日	1月17日	2	ホワイトボックステスト用プログラムの作成
1月18日	1月24日	8	テストとデバッグを行う
1月15日	1月25日	1	作成したプログラムの設計情報を書く
1月26日	1月26日	1	テスト情報を書く
1月27日	1月27日	1	事前計画と実際の進捗状況を書く
1月28日	1月28日		プログラムとレポートの提出

しかし、演習中に計画を以下の表 2 のように修正した。

表 2: 課題 4 における修正後のスケジュール

開始予定日	終了予定日	見積もり時間	作業内容
12月26日	12月26日	1	スケジュールを立てる
12月27日	12月30日	5	配布資料・サンプルプログラムを熟読する
12月31日	1月2日	3	コンパイラのテキストを熟読する
1月3日	1月7日	5	コード生成部の概略設計
1月8日	1月31日	10	コード生成部の作成
2月1日	2月1日	2	ホワイトボックステスト用プログラムの作成
2月2日	2月6日	8	テストとデバッグを行う
2月7日	2月7日	1	作成したプログラムの設計情報を書く
2月7日	2月7日	1	テスト情報を書く
2月8日	2月8日	1	事前計画と実際の進捗状況を書く
2月9日	2月9日		プログラムとレポートの提出

3.2 事前計画の立て方についての前課題からの改善点

今回はアセンブリ言語の仕様など、理解すべき事項が多かったので、テキストの熟読の時間を長めに確保した。

3.3 実際の進捗状況

約 3 週間に及ぶ入院のため、大きく遅れを取ることになった。

3.4 当初の事前計画と実際の進捗との差の原因

上記の通り、約 3 週間怪我のため入院していたため、締め切りは伸ばしていただいたものの、コーディングが大幅に遅れを取ることとなった。