ソフトウェア演習Ⅴ 課題3(再提出)

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 var_decl()
  - int var_names()
  - int type()
  - int ar_type()
  - int sub_decl()
  - int form_para()
  - int fukugou()
  - int statement()
  - int bunki()
  - int kurikaeshi()
  - int call_st()
  - int exp_narabi()
  - int dainyu()
  - int var()
  - int shiki()
  - int simple()
  - int kou()
  - int inshi()
  - int input_st()
  - int output_st()
  - int shitei()
- 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 printtab()
  - void inorder(struct ID *p)
  - void joint_localtoglobal()
  - void release_idtab()

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
    - printtab() 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
  - scan() 関数内
    - keyword_search() 関数を呼び出し
    - error() 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
    - block() 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
    - ar_types() 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
    - 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
    - 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
- 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
- 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
        * 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
    - printtab() 関数内
        * error 関数を呼び出し
        * inorder 関数を呼び出し

- inorder 関数内
  - inorder 関数を呼び出し (再帰呼出し)
- 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

大域変数 (手続き名も含む)、と局所変数 (仮引数として定義されたものも含む) を二分探索木で記憶するために、以下の図 1 のような構造体を定義した。

```
struct ID {
char *name;
char *procname;
struct TYPE *itp;
int ispara;
int deflinenum;
struct LINE *irefp;
struct ID *left, *right;
} ;
```

図 1: 大域変数、局所変数を記憶するための二分探索木の構造体

なお、ここで、name はその変数の名前、procname はその変数が局所変数か仮引数であるときにその変数が定義されている関数名、itp はその変数の型、ispara はその変数が仮引数かそれ以外か (仮引数なら 1、それ以外なら 0)、deflinenum はその変数が定義された行、irefp はその変数が使用された行、left と right はそれぞれ、左部分木、右部分木を指すポインタである。このように定義することにより、inorder 関数で辞書順に出力することができる。「struct TYPE」と「struct LINE」はそれぞれ変数の型、出現した行を記憶するために以下の図 2、3 のように定義された構造体である。

```
struct TYPE{
int ttype;
int arraysize;
struct TYPE *etp;
struct TYPE *paratp;
};
```

図 2: 型を記憶するための構造体

```
struct LINE{
int reflinenum;
struct LINE *nextlinep;
};
```

図 3: 出現した行を記憶するための構造体

ここで図 2 において ttype はその変数の型を表す整数、arraysize はその変数が配列型であった場合の要素数、etp はその変数が配列型であった場合の要素の型、paratp はその変数が procedure つまり手続き名であったときに仮引数の型を指すポインタである。すなわち、副プログラムの仮引数を格納する際には線形リストとして用いている。図 3 において reflinenum はその行数、nextlinep は線形リストの次の要素を指すポインタである。

```
struct NAME{
char *name;
struct NAME *next;
};
```

図 4: 変数名を記憶する構造体

また、変数宣言部において宣言される変数名のうち、コンマによって連続して宣言された同じ型の変数名を記憶するために以下の図 4 のような構造体を定義して、線形リストとして用いた。

なお、型を表す整数は次の図 5 のよう定義し、型名を格納する配列を crossreferencer.c 内で図 6 のように定義している。

```
#define TPINT 1 /* integer */
#define TPCHAR 2 /* char */
#define TPBOOL 3 /* boolean */
#define TPARRAY 4 /* array */
#define TPPROC 5 /* procedure */
```

図 5: 型を表す整数

```
char *typename[NUMOFTYPE + 1] = {
"",
"integer", "char", "boolean", "array", "procedure"
};
```

図 6: 型名を格納する配列

NUMMOFTYPE はこの配列に格納した型名の数であり、5 とヘッダファイル内で定義されている。

そして、局所変数用二分探索木はその副プログラム宣言が終了するときに大域変数用二分探索木に追加して、最後にまとめて出力している。そのことにより、辞書順に出力される。

名前は全文字列が一致している場合にのみ同一の名前とした。

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
  - struct KEY key[KEYWORDSIZE] :予約語 (キーワード) が格納されている。
  - char *tokenstr[NUMOFTOKEN+1] ：各トークン名が格納されている。
  - int token：読み込んだトークンのトークン番号が格納されている。
- scan.c 内
  - 課題 2 のレポートで説明済み。
- prettyprinter.c 内
  - 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 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 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。
- 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・大域変数の二分探索木のうち、探索した変数名の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
  - 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
  - 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。
  - form_para() 関数内
    - struct NAME *n：for 文で names を辿るためのポインタ。
    - 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。
  - bunki() 関数内

    - int flag：課題 2 で記載済み。
  - kurikaeshi() 関数内
    - int flag：課題 2 で記載済み
  - 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。
  - 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。
  - 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。
  - 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。
  - simple() 関数内
    - int flag：最初の項の前に＋か－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。
- 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。
- printtab() 関数内
  - int i：ループ変数。
  - space1：名前を出力してから型名を出力するまでどれだけ空白を開けるかを示す変数。
  - space2 : 型名を出力したから定義された行番号を出力するまでどれだけ空白を開けるかを示す変数。
  - char *cp：大域変数は変数名 (手続き名) を、局所変数のときは変数名にコロンと手続き名を足したものを格納するポインタ。
  - char ar[100]：配列型の要素数を文字列に変換したものを格納する配列。
- inorder() 関数内
  - struct LINE *q：出現した行番号を出力するときに行情報のリストを辿るポインタ。
  - int i：ループ変数。
  - space1：名前を出力してから型名を出力するまでどれだけ空白を開けるかを示す変数。
  - space2 : 型名を出力したから定義された行番号を出力するまでどれだけ空白を開けるかを示す変数。
  - char *cp：大域変数は変数名 (手続き名) を、局所変数のときは変数名にコロンと手続き名を足したものを格納するポインタ。
  - char ar[100]：配列型の要素数を文字列に変換したものを格納する配列。
- 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

  引数 コマンドライン引数として int nc と char *np[] を指定する。nc は指定された引数の個数を表し、np はプログラムを起動するときに指定する引数であり、本プログラムでは読み込むファイル名を指定する。

  返り値 プログラムが終了した際に 0 を返す。

  参照・変更する大域変数 token

- 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)：課題 1 で記載済み
- 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 1 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

- int block()

  機能 ブロックを解析する関数

  引数 なし

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

  参照する大域変数 token

- 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

- 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

- 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

  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

  参照する大域変数 token、string_attr

- 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

  参照する大域変数 names、token、typenum

- int fukugou()

  機能 複合文を解析する関数

  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

- int statement()

  機能 文を解析する関数

  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

- int bunki()

  機能 分岐文を解析する関数

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、indentnum

- int kurikaeshi()

  機能 繰り返し文を解析する関数

  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 shikitype、token

- int call_st()

  機能 手続き呼び出し文を解析する関数

  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、paranum、expnum

  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

  参照する大域変数 token、searchp、shikitype、

- int dainyu()

  機能 代入文を解析する関数

  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 vartype、shikitype、token

- 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、arrayflag

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

- 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、arrayflag、token、simplearraysize、simplezrraytype

- 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

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

- 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

  参照する大域変数 inshitype、token、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、arrayflag、inshitype、inshiarraysize、inshiarraytype

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

- int input_st()

  機能 入力文を解析する関数

  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 token、vartype

- int output_st()

  機能 出力文を解析する関数

  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

- 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

  参照する大域変数 shikitype、token

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・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 printtab()

  機能 二分探索木に格納した情報を出力する。

  引数・返り値 なし

  参照する大域変数 globalidroot、typename

  変更する大域変数 なし

- void inorder(struct ID *p)

  機能 二分探索木を通りがけ順で辿り、出力する。

  引数 struct ID *p：二分探索木の根を指すポインタ

  返り値 なし

  参照・変更する大域変数 なし

- 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、sample31p.mpl、sample032p.mpl、sample33p.mpl、sample34.mpl、sample35.mpl、sample29p.mpl、sample014.mpl についてテストを行った。さらに、ホワイトボックステストとして mysample031.mpl、mysample032.mpl、mysample033.mpl、mysample034.mpl、mysample035.mpl、mysample036.mpl、mysample037.mpl、mysample038.mpl、mysample039.mpl、mysample0310.mpl、mysample0311.mpl、mysample0312.mpl、mysample0313.mpl、mysample0314.mpl、mysample0315.mpl、mysample0316.mpl、mysample0317.mpl、mysample0318.mpl、mysample0319.mpl、mysample0320.mpl、mysample31.mpl、mysample32.mpl、mysample33.mpl というテストデータを用意してテストを行った。

配布されたテストデータによるテスト結果と、自作のテストデータとその結果についてはメールにより提出する。なお、テスト結果を格納するファイル名は「(テストプログラム名)_test.txt」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「kadai3-test.zip」というファイルにまとめて圧縮して提出する。テスト結果を格納しているファイルには想定される出力結果と、実際に行ったテスト結果が書き込まれており、「想定」以下が想定される出力結果で、「結果」以下が実際の出力結果である。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 31 分である。

2.2　テストデータの十分性

sample31p.mpl、sample33p、sample34.mpl、sample35.mpl、mysample31.mpl、mysample32.mpl、mysample33.mpl で、エラーメッセージが表示されない場合に通るすべての命令が網羅されている。

残りのテストデータにおいて通常実行されない、コマンドラインが与えられていないときのエラーメッセージ、ファイルが開けないときのエラーメッセージ、malloc が失敗したときのエラーメッセージが表示される場合を除く、エラーメッセージを表示するすべての場合に通る命令を網羅している。なお、ここでは prettyprinter.c 内において scan() 関数からの返却値が-1 であった場合に ERROR を返すという処理については、sampe014.mpl によって一か所正常に実行されることが確認できたので、全ての処理において正常に実行されると判断し、命令が網羅されているとした。

すなわち、C0 カバレッジで 100%であると言える。

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 3 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
11月27日	11月27日	1	スケジュールを立てる
11月28日	11月30日	3	配布資料・サンプルプログラムを熟読する
12月1日	12月1日	2	コンパイラのテキストを熟読する
12月2日	12月4日	5	クロスリファレンサの概略設計
12月5日	12月13日	9	クロスリファレンサの作成
12月14日	12月14日	1	バグがない場合の想定テスト結果の準備(配布されたテストプログラムについて)
12月15日	12月15日	2	ホワイトボックステスト用プログラムの作成
12月16日	12月16日	1	バグがない場合の想定テスト結果の準備(自分で作成したテストプログラムについて)
12月17日	12月22日	10	テストとデバッグを行う
12月23日	12月23日	1	作成したプログラムの設計情報を書く
12月23日	12月23日	1	テスト情報を書く
12月23日	12月23日	1	事前計画と実際の進捗状況を書く
12月23日	12月23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 3 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
11月27日	11月27日	1	スケジュールを立てる
11月28日	11月30日	3	配布資料・サンプルプログラムを熟読する
12月1日	12月1日	2	コンパイラのテキストを熟読する
12月2日	12月4日	5	クロスリファレンサの概略設計
12月5日	12月13日	9	クロスリファレンサの作成
12月14日	12月14日	1	バグがない場合の想定テスト結果の準備(配布されたテストプログラムについて)
12月15日	12月15日	2	ホワイトボックステスト用プログラムの作成
12月16日	12月16日	1	バグがない場合の想定テスト結果の準備(自分で作成したテストプログラムについて)
12月17日	12月23日	10	テストとデバッグを行う
12月24日	12月24日	1	作成したプログラムの設計情報を書く
12月24日	12月24日	1	テスト情報を書く
12月24日	12月24日	1	事前計画と実際の進捗状況を書く
12月25日	12月25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

クロスリファレンサのコーディングとテスト・デバッグの時間を長めに確保した。

3.3 実際の進捗状況

表 2 のように変更したように、コーディング、テストとデバッグが予想以上に時間がかかってしまった。その他の作業についてはだいたい計画通りに進んだ。

3.4 当初の事前計画と実際の進捗との差の原因

私自身の理解不足、命令を網羅するために必要なテストプログラムの数の見積もりが少なかったことが原因であると考えられる。