ソフトウェア演習Ⅴ 課題1(再々提出)

15122013 尾持涼介

提出日：2017年12月25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。

- token-list.c
  - main 関数
  - void error(char *mes)
- scan.c
  - int keyword_search(char *string)
  - int init_scan(char *filename)
  - int keyword_search(char *string)
  - int scan()
  - int get_linenum()
  - void end_scan()
- id-list.c
  - void init_idtab()
  - struct ID *search_idtab(char *np)
  - void id_countup(char *np)
  - void print_idtab()
  - 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。

- token-list.c
  - main 関数内
    - init_scan() 関数を呼び出し
    - scan() 関数を呼び出し
    - end_scan() 関数を呼び出し
    - print_idtab() 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
  - keyword_search() 関数内
    - id_countup() を呼び出し
  - scan() 関数内
    - keyword_search() 関数を呼び出し
- id-list.c
  - id_countup() 関数内
    - search_idtab() 関数を呼び出し
  - release_idtab() 関数内
    - init_idtab() 関数を呼び出し

1.2 各モジュールごとの構成

キーワードとそのトークンコードを格納するために以下の図 1 のような構造体をヘッダファイル token-list.h で定義した。

```
extern struct KEY {
char * keyword;
int keytoken;
} key[KEYWORDSIZE];
```

図 1: キーワードとそのトークンコードを格納する構造体

また、名前を識別するための構造体を以下の図 2 のような構造体を id-list.c 内で定義した。

```
struct ID {
char *name;
int count;
struct ID *nextp;
}  *idroot;
```

図 2: 名前を識別するための構造体

scan.c では、ctype.h をインクルードし、scan() 関数内で cbuf に読み込んだ文字がアルファベットかを判別する isalpha() 関数、数字、またはアルファベットかどうかを判別する isalnum() 関数、数字かどうかを判別する isdigit() 関数を使用している。それらを用いて if 文でキーワードあるいは名前、符号なし数字を判別し、その他の記号等は switch 文を用いて判別している。

次に使用した大域変数とその意味について述べる。

- token-list.c 内
  - int numtoken[NUMOFTOKEN+1] : 各トークンを数え上げるために使用。各トークンのトークンコードと同じ場所にその個数を格納する。すなわち、例えば、numtoken[1] には名前の個数が格納される。なお、NUMOFTOKEN とは定義したトークンの個数であり、49 と定義されている。
  - char *tokenstr[NUMOFTOKEN+1] ：各トークン名が格納されている。
- scan.c 内
  - int cbuf : ファイルから読み込んだ文字を格納する 1 文字分の文字バッファ
  - int line_cnt : 行数を数え上げる変数
  - int newline : 次の文字から新しい行が始まる場合は 0、そうでない場合は 1 を格納しておく。
  - FILE *fp : ファイルを操作するためのポインタ
- id-list.c 内
  - struct ID *idroot : 各「名前」とその個数を格納する線形リストの先頭要素を指すポインタ

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

- token-list.c
  - main 関数内
    - int token : scan() 関数から返却されたトークンコードを格納する。
    - int i : for 文によるループで用いる変数
- scan.c

* ikeyword_search() 関数内
  - int l：二分探索において左端を指す変数
  - int r：二分探索において右端を指す変数
  - int c：二分探索において真ん中を指す変数
  - int comp：文字列を比較する際に用いる strcmp() 関数からの返り値を格納する変数
* scan() 関数内
  - int num_attr：scan() の返り値が「符号なし整数」の時、その値を格納する変数。
  - char string_attr[MAXSTRSIZE]：scan() の返り値が「名前」または「文字列」の時、その実際の文字列を格納する配列。また、それが「符号なし整数」の時は、入力された数字列を格納する。また、MAXSTRSIZE とはこの配列に格納できる文字列の最大長であり、1024 と定義している。
  - int i：string_attr に文字、または数字を格納していくときにその配列の何番目の要素に格納するかを指定する変数。

– id-list.c
* search_idtab() 関数内
  - struct ID *p：for 文により構造体を探索する際に構造体を操作するポインタ。
* id_countup() 関数内
  - struct ID *p：探索した結果既にその「名前」が構造体に格納されている場合は、その場所を指すポインタとなり、探索した結果初出の名前であった場合は構造体に格納するための情報を格納する新たな要素となる。
  - char *cp：search_idtab() 関数で探索した「名前」がその初出のものである場合、その「名前」を格納するポインタとなる。
* print_idtab() 関数内
  - struct ID *p：for 文において構造体を操作するためのポインタ
* release_idtab() 関数内
  - struct ID *p, *q：どちらも for 文において構造体を操作するために使用するポインタ

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 token-list.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。

  参照する大域変数 tokenstr

  変更する大域変数 numtoken

- void error(char *mes)

  機能 エラーが発生したときにエラーメッセージとエラー発生箇所を表示する。

  引数 char *mes：表示するエラーメッセージ

  返り値 なし

  参照・変更する大域変数 なし

1.3.2 scan.c 内で記述されている関数

- int init_scan(char *filename)

  機能 filename が表すファイルを入力ファイルとしてオープンする。また、変数 line_cnt と newline を 0 に初期化し、cbuf に 1 文字読み込んでおく。

  引数 char *filename : 開くファイル名

  返り値 正常にファイルが開けた場合は 0、ファイルが開けなかった場合など異常な場合は-1 を返す

  参照・変更する大域変数 line_cnt、newline、cbuf

- int keyword_search(char *string)

  機能 読み込んだ文字列がキーワードかどうかを調べる。

  引数 char *string : 探索する文字列

  返り値 探索した結果のトークンコードを返却する。

  参照・変更する大域変数 なし

- int scan()

  機能 トークンを 1 つスキャンし、そのトークンを識別する。

  引数 なし

  返り値 トークンのコードを返す。End-of-File が現れたときやエラーが発生したときは-1 を返す。

  参照する大域変数 cbuf、newline、fp

  変更する大域変数 cbuf、newline、line_cnt

- int get_linenum()

  機能 最も最近に scan() で返されたトークンが存在した行番号を返す関数。

  引数 なし

  返り値 最も最近に scan() で返されたトークンが存在した行番号を返す。まだ一度も scan() が呼ばれていないときには 0 を返す。

  参照・変更する大域変数 なし

- void end_scan()

  機能 init_scan() 関数でオープンしたファイルをクローズする。

  引数・返り値 なし

  参照・変更する大域変数 fp

1.3.3 id-list.c 内で記述した関数

- void init_idtab()

  機能 名前を識別するための構造体を初期化する。

  引数・返り値 なし

  変更する大域変数 idroot

- struct ID *search_idtab(char *np)

  機能 指定された「名前」が線形リスト内に存在するかを探索する。

  引数 char *np : 探索する「名前」

返り値 探索した結果その「名前」が線形リスト内に発見されればその要素を指すポインタ、もし見つからなければ NULL を返す。

参照する大域変数 idroot

- void id_countup(char *np)

  機能 np で指定された名前の個数を数え上げる。

  引数 char *np：数え上げる「名前」

  返り値 なし

  参照・変更する大域変数 idroot

- void print_idtab()

  機能 各「名前」とその個数を表示する。

  引数・返り値 なし

  参照する大域変数 idroot

- void release_idtab()

  機能 idroot が確保していた領域を解放する。

  引数・返り値 なし

  参照・変更する大域変数 idroot

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、sample011.mpl、sample014.mpl、sample11.mpl、sample11p.mpl、sample11pp.mpl、sample12.mpl、sample13.mpl、sample14.mpl、sample15.mpl、sample15a.mpl、sample16.mpl、sample17.mpl、sample18.mpl、sample19p.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 mysample01.mpl、mysample02.mpl、mysample03.mpl、mysample04.mpl、mysample05.mpl、mysample06.mpl、mysample07.mpl、mysample08.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、mysample05.mpl 以外の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「kadai1-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 11 月 27 日 10 時 08 分である。

mysample05.mpl については以下で説明する。このファイルは何も書き込まれていない空ファイルである。すなわちファイルから 1 文字読み込むといきなり EOF が現れるファイルである。このファイルを用いてテストを行ったところ、標準出力には何も表示されないままプログラムは終了した。なお、これは想定通りの動作である。

2.2 テストデータの十分性

改行コードについては環境によって 4 通りのうちどれが現れるかは決まるので、以下では 4 通りのうちどれかが出現したら「改行コードに関する命令は網羅された」として議論する。

sample014.mpl 以外の配布されたテストデータと mysample05.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。

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 1 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
10月2日	10月2日	1	スケジュールを立てる
10月3日	10月5日	3	配布資料・サンプルプログラムを熟読する
10月6日	10月6日	2	コンパイラのテキストを熟読する
10月7日	10月9日	5	字句解析系(スキャナ)の概略設計
10月10日	10月15日	7	スキャナの作成
10月16日	10月16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)
10月17日	10月17日	0.2	配布されたメインプログラムのトークンカウント用の配列の初期化部分の作成
10月17日	10月17日	0.2	配布されたメインプログラムのトークンのカウント部分の作成
10月17日	10月17日	0.4	配布されたメインプログラムのカウントした結果の出力部分の作成
10月18日	10月18日	2	ホワイトボックステスト用プログラムの作成
10月19日	10月19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)
10月20日	10月24日	8	テストとデバッグを行う
10月25日	10月25日	1	作成したプログラムの設計情報を書く
10月26日	10月26日	1	テスト情報を書く
10月26日	10月26日	1	事前計画と実際の進捗状況を書く
10月27日	10月27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。

開始予定日	終了予定日	見積もり時間	作業内容
10月2日	10月2日	1	スケジュールを立てる
10月3日	10月5日	3	配布資料・サンプルプログラムを熟読する
10月6日	10月6日	2	コンパイラのテキストを熟読する
10月7日	10月9日	5	字句解析系(スキャナ)の概略設計
10月10日	10月18日	10	スキャナの作成
10月19日	10月19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)
10月20日	10月20日	0.2	配布されたメインプログラムのトークンカウント用の配列の初期化部分の作成
10月20日	10月20日	0.2	配布されたメインプログラムのトークンのカウント部分の作成
10月29日	10月20日	0.4	配布されたメインプログラムのカウントした結果の出力部分の作成
10月21日	10月21日	2	ホワイトボックステスト用プログラムの作成
10月22日	10月22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)
10月23日	10月27日	8	テストとデバッグを行う
10月28日	10月28日	1	作成したプログラムの設計情報を書く
10月29日	10月29日	1	テスト情報を書く
10月29日	10月29日	1	事前計画と実際の進捗状況を書く
10月30日	10月30日		プログラムとレポートの提出
11月27日	11月27日		再提出
12月25日	12月25日		再々提出

3.2　実際の進捗状況

表 2 のように変更したように、スキャナのコーディン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。

しかし、課題 2 に取り組んでいる際に scan.c 内に誤りを見つけたため再提出を行った。また、keyword_search() 関数において、不必要な関数を呼び出していたので、再々提出を行った。

3.3　当初の事前計画と実際の進捗との差の原因

配布資料・プログラムやテキストの熟読が十分でなく、理解不足であったことが原因であると考えられる。そのため、コーディングの際にも何度もテキストや配布資料を読むこととなり、余計に時間がかかってしまったのであ

る。このようなことをなくすために、次回以降はテキストや配布資料を熟読する時間を長めにとるようにすればよいと考えられる。また、再提出となったのは、見直しが不十分であったからであると考えられる。