ソースファイルの内容をいちいちレポートに掲載する必要はありません。説明の都合上、必要な場合はその限りではありませんが、そうでなければレポート作成の労力が増えるだけです。このレポートは比較的要点が絞られていて良いと思いますが、それでも、レポートの体裁よりも内容を重視して下さい。

プロジェクト演習課題 1-2 ソースコードの理解

学籍番号 745086
氏名 松浦　康大

丁寧にまとめられていて、大変良いと思います。
文法的にも不明な点がまだ多いと思いますが、仕事に関係なく、C言語をがっつり勉強できる最後の機会かもしれません。頑張りましょう。

○自分が担当したファイル

①abort.c　②abort.h　③commontype.h　④errmessages.c

⑤evaluate.c　⑥evaluate.h　⑦execute.c　⑧execute.h

①abort.c

ファイル名の abort は直訳すると”中断する”という意味がある。また、fprintf・fputc 関数の出力ストリーム先に stderr(標準エラー出力ストリーム)が指定されていることから、このコードはエラーメッセージを画面に出力するものであると考えた。

```
12          fprintf(stderr, "ERROR: %s", getErrorMessage(msg));
```

上記のコードから getErrorMessage 関数の返り値をエラーメッセージとして出力していることがわかる。ここで getErrorMessage 関数を他のプログラムの中から探すと、errmessages.c の中にあった。このプログラムの中には実際に出力するエラーメッセージが配列で複数格納されている。よって、この一覧

の中からエラーの種類に応じてメッセージを返し出力する仕組みであることがわかる。次に、

```
13          if (str)
14              fprintf(stderr, " %s", str);
15          fputc('¥n', stderr);
16          if (stdtxin) {
17              printTextBuffer(stderr, stdtxin);
18              freeTextBuffer(stdtxin);
19              stdtxin = NULL;
20          }
21          exit(1); // === EXIT THE PROGRAM ===
22      }
24      void abortMessage(const char *msg) {
25          abortMessage_string(msg, NULL);
26      }
27
28      void abortMessageWithToken(const char *msg, const item *s)
29      {
30          char buffer[80];
31          itemToString(buffer, s);
32          abortMessage_string(msg, buffer);
33      }
34
35      void abortMessageWithString(const char *msg, const char *str)
36      {
37          char buffer[40];
38          sprintf(buffer, "'%s'", shorten(str, 32));
39          abortMessage_string(msg, buffer);
40      }
```

13 行目の if(str)は、str が NULL ではない場合という意味だと解釈し、エラーメッセージの他に何かを表示したいときの処理であると考えた。32 行目と 39 行目から str=buffer であるとわかるが、buffer が何を指しているのかがわからなかった。また、if(stdtxin)がどのような条件であるかや、shorten()がどういう関数であるかが理解できず、他のプログラムのコードの理解を深める必要があると感じた。

すぐ上で定義していますけど？

これは担当した人に聞きましょう

②abort.h

main.c、abort.c ほか複数プログラムののヘッダファイルで、

```
7       void abortMessage(const char *msg);
8       void abortMessageWithToken(const char *msg, const struct _item *s);
9       void abortMessageWithString(const char *msg, const char *str);
```

上記などで、abort.h で使う関数のプロトタイプ宣言をしている。プロトタイプ宣言をすることで、関数の引数などの記述に誤りがあった場合にコンパイラがコンパイル時にエラーを表示するようになる。

コンパイラに、関数の情報を正しく伝えるのが目的です。

#id !defined(マクロ名)～#endif は、マクロ名が定義されていなければ、それ以降の文をコンパイル

するという意味がある。3 行目ですぐに__ABORT_H__は定義されている為、コンパイルは問題なく行われると考えられる。また、14 行目の defined(DEBUG)に関しては、その名の通りデバッグ用に作られているものと考えられ、動作を検証するためにプログラムを追加したりする場合は#define で DEBUG を定義して検証するのだと考えられる。

そういうことですね

③commontype.h

newnode.c、nextch.h、statements.h、token.h のヘッダファイルで、

```
6     /* Note that it may conflict with the implementation of C (in
      the future) */
8     # define true     1
9     # define false    0
10    # define bool     int
13     /* "True" in the conditional expression of C is not necessarily
       1.So, don't describe as "expression == TRUE" for comparation. */
15    #define BOOL(c)   ((c) ? true : false)
```

C言語には _bool という、ブール型が存在します。
将来的に Bool とか bool とかに改名される可能性もあるよ、
という程度のこと。

どのコードでも同様に定義される普遍な変数を定義していることがわかる。また、6 行目・15 行目のコメントを訳してみると、”それは C 言語の実装と競合する恐れがある”、”C 言語の条件としての True は必ずしも 1 とは限らない。よって、比較のために”式”==真と見なしてはいけない”となる。この文章から、define による定義としての true と C 言語の boolean における定義としての true が競合するのではないかと考えた。boolean における true は 0 以外の意であり、それを 1 のみと定義してしまうことで、true の定義に対して競合が起こるのだと予測した。

記述

例えばポインタ変数 p があった時、NULLは 0、どこかをポインタで指している時は 0 以外なので、 if (p) と記述すると、「p が NULLでない時」という意味になります。この記法はCでは当たり前に使われますが、これを if (p == true) と記述するのは間違いです。そんな感じの話です。

④errmessages.c

```
51    const char *getErrorMessage(const char *key)
52    {
53        for (int i = 0; table[i].tag; i++) {
54            if (strcmp(key, table[i].tag) == 0)
55                return table[i].localized;
56        }
57        return key;
58    }
```

table[]配列には各エラーメッセージが格納されている。54 行目の strcmp()は引数どうしが等しければ 0 を返すので、key と table[i].tag が等しければ table[i].localized を返すことになる。ここで、このエラーメッセージはコンパイルの際に表示するエラーメッセージであると考え、key にはメッセージに記され

たような行為が入るのではないかと考えた。その行為と table[]内のエラーメッ
??
セージが一致した時にそれを返すようになっているのだと予想する。

⑤evaluate.c

```
15     switch (opr) {
16             case sym_or: val = BOOL((o1 != 0) || (o2 != 0)); break;
17             case sym_and: val = BOOL((o1 != 0) && (o2 != 0)); break;
18             case sym_plus: val = o1 + o2; break;
19             case sym_minus: val = o1 - o2; break;
20             case sym_ast: val = o1 * o2; break;
21             case sym_sls:
22                 if (o2 == 0) abortMessageWithString("arith exception", "/");
23                 val = o1 / o2; break;
24             case sym_pcnt:
25                 if (o2 == 0) abortMessageWithString("arith exception", "%");
26                 val = o1 % o2; break;
27             case sym_equal: val = BOOL(o1 == o2); break;
28             case sym_neq:   val = BOOL(o1 != o2); break;
29             case sym_gt:    val = BOOL(o1 > o2); break;
30             case sym_lt:    val = BOOL(o1 < o2); break;
31             case sym_geq:   val = BOOL(o1 >= o2); break;
32             case sym_leq:   val = BOOL(o1 <= o2); break;
33             default:
34                 assert(false); break;
35         }
```

15 行目から 35 行目までの switch-case 文では opr の値に応じて val が変わるようになっている。ここで opr には、48 行目から expptr->kind が渡されているが、expptr がプログラム中の何を指しているのかわからなかった。ただ、48 行目の applyOperator 関数が含まれる if 文のコメントに binary operator とあることから、受け取った式が二項演算子を含んでいた場合の処理であることが推測できる。他のコードは変数や文法がややこしく内容を把握できなかったため、わ
そうですね
からなかった文法について調べた。

・a -> b

→アロー演算子。ポインタ a が指す構造体のフィールド b を読み書きできる。

=(*a).b

・stack

→データを積み上げていき、最後に入力したデータが先に出力される。

・push で挿入

・pop で取り出す

・putchar

→文字を 1 文字標準出力に出力

⑥evaluate.h

```
12    extern int sp;         // stack pointer
13   extern int localbase;   // index of 1st local var
```

extern 宣言とは宣言だけを行い定義は行わない宣言方法である。これがあるこ

とで異なるソースファイル間で変数を共有することができる。他には、他のヘッ

ダファイル同様関数のプロトタイプ宣言を行なっている。

sp、localbase 二つの変数については、コメント文から以下のように読み取れる。

スタックの使い方については、配布したソースファイルおよび説明書一式の中に説明の文書が入っています。

sp: stack の先頭のポインタをグローバル領域で宣言

localbase:　1 番目のローカル変数のインデックス(配列の要素)

⑦execute.c

```
3     #include <stdio.h>
4     #include <stdlib.h>
5     #include "evaluate.h"
6     #include "exp_imp.h"
```

まず、#include が<>で指定している場合と””で指定している場合がありこの違いについて調べると。<>は探査パスにカレントディレクトリを含めないのに対し、””は探査パスにおいてカレントディレクトリから探し、なかったらパスが通っているところを探すという意味であった。この使い分けは、stdio.h など自分で作ったものではないヘッダファイルは””で囲んだところでカレントディレクトリを探す意味がないため<>で囲むようにすればよいと思う。

内容を読み取るのが困難だったため、初めて見た文法をまとめることにする。

```
20    typedef enum { ex_normal, ex_break, ex_return } ex_condition;
55        ex_condition r = ex_normal;
66        ex_condition r = ex_normal;
94        ex_condition r;
```

・20 行目　typedef enum {メンバ 1, メンバ 2, …} 型名:

→ここでは enum 型 ex_condition に、それぞれメンバを定義している。これによって、55、66、94 行目などにある r に ex_normal などのメンバを代入するこ

とができる。

```
28      long *target = asp->global ? &globals[idx] : &stack[localbase - idx];
```

・&~はポインタにアドレスを代入していることを意味する。

このプログラムはプログラム名や include しているヘッダファイルの多さから、

様々な情報をすべてまとめて実際に動作を実行するファイルだと考える。

⑧execute.h

execute.h をヘッダファイルにしているファイルを検索しても見つからなかっ

たため、このファイルの存在している意味がわからず疑問に感じた。

すいません、execute.h というファイルは不必要なので削除して下さい。
execute.h はどこからも include されていません。バージョン 1.0.1 から 1.0.2 に
する時点で消去すべきでしたが、残ってしまっています。
ただし、このファイルが存在していても、いなくても、コンパイルには全く影響ありません。