プロジェクト演習

学生番号：744221
氏名：　榎本 啓希

- 自分が担当したソースコード
  - getitem.c, getitem.h
  - getsymbol.c, getsymbol.h
  - identifires.c, identifires.h
  - idtable.c, idtable.h
  - main.c

処理内容の理解はまだパーフェクトではありませんが、
そもそも他人のコードに手を入れる際、必ずしも
完璧である必要はないと思います。
レポートとしては考察がしっかり書けており、
高く評価したいと思います。

前回、identifires と idtable があまり理解できなかったため、その2つを中心に見ていく。

- identifires.c, identifires.h

12~17 行目では、予約語が reserved として配列に格納されている。この定義のメモ書きに、symbol の順番と合わせる必要があることが記されている。これは、reserved[symbol]という風に値を取ると、対応する予約語の文字列を得ることができるようにするためである。この順番とは token.h の reserved words の場所であることがわかる。今回の課題に予約語を増やすものがあったため、実装に取り掛かる際には注意が必要である。実際には予約語の token は 0 から始まっているわけではないため、token では予約語の始まりと同時に reserved_word_0 を定義することによって、受け取ったシンボルから reserved_word_0 を引いた値が、配列番号となる。この処理は identifiers.c の reservedWord で行われている。

良い考察ですね

関数 idtablesInitialize は main ファイルから呼び出されており、初期化をする関数である。フィードバックで、idtable は不完全型宣言されていることがわかり、これはまだ詳細が決まっていない構造体であることを理解した。この関数では、まだ決まっていない idtable 型を持った resv_table と global_table の初期化も行なっている。初期化自体は idtable.c の関数 idtableCreate で行われており、その際に、確保するメモリの大きさと bool を渡す。

この時点で、予約語は決まっているため、予約語をテーブルに追加している。同時に、構造体 idRecord も作成している。

詳細が決まっていないと言うより、詳細を見せない、
が近いです。これによって、Javaの private のような
効果を作ることができます。

idtablesFree は free 関数を使って、resv_table と global_table の両方のメモリを解放している。この関数はテーブルのために確保しておいたメモリを解放するため、main にて1度のみ呼び出される。free 関数について調べたところ、malloc や calloc, realloc 関数によって動的に確保されたメモリは必ず free 関数でメモリを解放する必要があるようだ。

プログラムが終了するまで使い続けられるものはその限りではありません

makeStringInfo は文字列と bool を取り、string_info という構造体を作って返す。bool の役割は後で定義するかどうかを決めておくものであるが、受け取った文字列を符号なしでキャストした後の 0 番目と1番目をチェックする部分とハッシュ値の決め方がわからなかった。

おそらく 0 番目に何かあり、1番目が何もないなら、アルファベット 1 文字のため、アスキーで判断できるからそのままの値をハッシュ値にしている。それ以下は(^)が排他的論理和を表していることと、ハッシュ値が int の最大値を超えないように論理積を用いているぐらいしか理解できなかった。

それで十分です。と言うか、よく分かりましたね

identifierSearch は、StringInfoPtr という string_info のポインタ型であるものを引数とし、予約語のテーブルから情報を取ってきて、一致すれば token に symbol が格納されている order を代入して item を返している。それ以外であれば、グローバル、ローカル変数などを条件によって探して item を返す。

addIdentifier は str ポインタを受け取ってグローバル変数かローカル変数を新しく定義する命令を出すところである。display_index はおそらく、func や proc などのスコープの場所を示すものであり、0 の時はグローバルであることがわかる。

その通りです

- idtable.c, idtablel.h

構造体 string_pool は構造体 idtable に文字列を補完する箱として用いられる。make_string_pool 関数では受け取った整数値から構造体を作るためのメモリを確保し、構造体を作る。

関数*duplicate_str は string_pool と stringInfoPtr を受け取り、複製を行なっているものだと考えられるが、headp に対しての処理が何を行なっているかわからなかった。

```
    char *headp = pp->pool + pp->strindex;
    strcpy(headp, strp->name);
```

idtable.c 56, 57 行目より

かなり大きな char型配列に、文字列をずらずらと並べていくという処理をします。
その大きな領域の先頭が pp->pool で、これまでに使った文字数が pp->strindex です。
pool + strindex が、新たに文字列をコピーすべき先頭位置になります。

ハッシュテーブルで、いわゆる開番地法を使っています。
現在のキーで衝突が起きたとき、素数である17を加えることで
次の位置を決めています。ここで素数を使うことで、テーブル内を
もれなく探すことが保証されます（理由は考えてみてください）

関数 idtableFree は受け取った構造体 idtable において確保したメモリを解放するものである。渡された構造体のメモリが確保されていない場合は解放する必要がないため、処理を終了する。

関数 idtableSearch は受け取った stringInfoPtr から idtable のレコードから一致する識別子を探して見つけたレコードを返す関数である。無限ループで見つけるようだが、レコードのインデックスを不規則に変えている部分は理解できなかった。

関数 idtableAdd は予約語やグローバル変数のテーブルにレコードを追加するものである。追加することでテーブルの容量から超えてしまう場合はエラーメッセージを出す。また、すでにテーブルにある場合は NULL を返す。

関数 resetLocalList は構造体 local_pool を初期化するものである。この時点で、local_pool がない場合は新しく作る。

関数 blockNestPush と blockNestPop はそれぞれ local block に対して、追加と削除を行うものであるが、この local block というものが何を示しているのかがわからなかった。考えられることとして、ローカルブロックは proc か func のそれぞれの中でのスコープを分けているものだと思う。

seachLocalIdentifier は受け取った stringInfoPtr が以前登録されたものであるか確認をするために、箱である local_identifiers から一致するものを探している。

addLocalIdentifier は新たに local_identifiers に stringInfoPtr の内容を追加する。この関数の最後に local_id_index に1加算することで次回に
idrec = &local_identifiers[local_id_index] とすると新たな箱が作られる仕組みとなっている。

if文や while文などのブロック（文列）でローカルな変数が定義される場合を考えましょう。ネストが深くなる時は、それより浅いブロックの変数も参照できますが、深いネストから抜けて戻ってくると、深いネストの変数定義はもう使われません。この Pushはネストが１つ増えるとき、popはネストから抜ける時に実行されます。もう少し具体的には、配列 local_display の添字をコントロールして、ローカル変数の情報を追加したり、無効にしたりしています。