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

大筋でまあいいんですが、前回のレポートで、コメントした部分は加筆修正しておいてくれたらよかったと思います。

- 第３週の課題
  - getItemLocal()と getItem()の動作の違い

定義や呼び出し方の話ではなく、「動作の違い」を説明してください。

getItem()と getItemLocal()の違いは getitem.h にて記述されている。元となっている関数は fgetItem(TIN *tip, bool current_only)というものであり、それを改めて定義する形となっている。この2つの違いとして、bool current_only という真理値を受け取る引数に、あらかじめ getItemLocal()なら true を、getItem()なら false を入れている。こうすることにより、変数を取得する際に、ローカル変数から参照するのか、グローバル変数から参照するのかをわかりやすく表すことができる。

  - getsymgol()内の undoch()について

関数 getsymbol では、受け取った文字が「==」や「>=」などといった比較演算子を表しているかどうかを判断し、それに相当するトークンを Item に代入して返す役割をしている。読み込み、解析するのは 1 文字ずつ行なっているが、比較演算子には「==」というように 2 文字見る必要がある。そこで 74 行目で nextch(tip)を行い次の文字は何かを nx に代入する。この nx と最初の文字 ch で比較演算子の判断を行なっている。

しかし、次に来る文字が比較演算子ではない場合、ここで処理するものではないため、1 文字戻しておく必要がある。この時に用いられているのが関数 undoch()である。

そうですね

  - identifierSearch()の返す値について

関数 identifierSearch()は受け取った文字列が何を表しているのか（変数なのか予約語なのか）を調べて、item として返す役割をしている。最初に予約語と一致するか参照し、うまく受け取れた場合は、予約語のトークンを代入して返している。その後、対象の文字列を

グローバルやローカルのスコープで探し、見つからなかった場合は item の kind に id_undefined を代入し、見つからなかったため、定義されていないということを示して値を返している。

だいたいOK

- currentLocalOffset という変数の役割を考察

  currentLocalOffset は extern で定義されており、ファイルをまたいでのグローバル変数であることがわかる。コードを見てもあまりわからなかったため、Xcode のデバッグツールを使って変数の動向を探ると、for 文を増やしたり、変数の定義を増やしたりすると値が増えることがわかった。このことから、この変数は現在のローカルなスコープの中で、どのぐらい変数が定義されているのかを記憶しておく変数であることが予想される。

レポート末尾にコメント

- 機能拡張について

  授業内では配列を無視するという風になっていたため、ブラケット([])の文字がきた時にシンボルではなく文字として扱うようにシンボルテーブルを変更したが、この場合 hoge[]という名前の1つの変数として捉えられることになる。それではただ変数に使える文字列が増えただけなので、変更する必要がある。

  現時点で getitem.c の文字を読み込む関数 get_identifier 内でブラケットを識別させることを考えている。get_identifier では while 文で文字がアルファベットか数字の場合は buf に格納して次の文字を観察していることから、文字列を見ていることがわかる。これを利用して、もし、ブラケットが来た場合に配列を作成する処理を追加してあげれば良いと考えられる。しかし、ここまでは考えることができるが配列を作るとなるとメモリ領域の確保の仕方や、どのようにテーブルに保存しておけば良いか、まだ考えないといけない点は多くあるため、この先はメンバーと話し合う予定である。

小手先の変更では実装できません。
構文定義から変更する必要があります。

――― 以下は前回レポート ―――

- 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。

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。

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。

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。

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。

- 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

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。

あるブロックの中でローカル変数を宣言すると、それまで宣言されていた個数に上乗せして総数をカウントする必要がありますが、その変数はブロックから出ると無効になります。そのため、現在有効なローカル変数の個数を currentLocalOffset に持ち、サブルーチンの中でローカル変数は最大いくつ必要かを覚えておく変数が maxLocalOffset になります。"offset" という呼び名を使うのは、この値がスタックの上でローカル変数にアクセスする時に使う「変位」だからです。
これらの変数の管理は、blockNestPop() と blockNestPush() の中で行なっています。