プロジェクト演習3週目レポート

学籍番号：744177
氏名：上田晃

・担当したソースコード

exp_imp.h
expnode.h
expression.c
expression.h
exptree.c
functions.c

頑張った部分は認められますが、拡張機能の repeat - until で盛大に誤解をしているなど、基本的な理解という部分で不安が残ります。
グループのメンバーと相談しつつ、実装について検討を始めてください。

＊前回のレポートを変更した部分は色を変えている。

・exp_imp.h
struct _expnode、struct _oprExpnode、struct _argExpnodeでは、token.hで宣言されているtoken型のtok_id、tok_num、tok_str、などのkindとprefixはそれぞれ共通したものとなっている。 kindには式や数字などが、prefixには0や'+', '-','not'が入る。

構造体_expnodeにはunionという複数の型を同一のメモリ上で共用することができる構造が含まれている。kindが数字または文字列として代入されている時にlong型として、kindが変数名として代入されている時に構造体varinfoとして扱うことができる。この共用体をvとして扱うことができる。

構造体oprExpnodeにはkindとprefixの他に*operand[2]、長さ2の配列が用意されている。operand、被演算子が入るため2つ分の長さである。被演算子とは'1 + 2'とある場合に演算子ではない部分である'1'と'2'を指す。

構造体_argExpnodeには*args[1]、長さ1の配列が用意されているのは、exptree.cの関数newArgnode()で必要になるからである。これはexptree.cで説明する。

・expnode.h
expnode.hではexptree.cで扱う関数のプロトタイプ宣言を行なっている。

なんだか微妙に違う点があるので：
文字列リテラルの場合、構造体expnodeに、tok_strという値と、
文字列リテラルの通し番号が格納されたものが返り値となります。

・expression.c
用意された文書に乗
関数strExpression
数newExpnode()を
文字列ではなかった場合、普通の式があると考えて、変数sにいったん読み込んだトークンをungetItem()で「元に戻して」、改めて expression()を呼び出します。
に順番に割り当てられる通し番号）に当てはめた後に関数expression()に戻り値を返す。
文字列リテラルとは、0文字以上の連続した文字列を示す定数のことであり、例を挙げるとダブルクオーテーションで囲まれた”string”などのことである。
トークンが文字列でない場合には、getitem.cの関数ungetitem()を呼び出す。
ungetという意味は詳しくはわからなかったがC言語の関数にはungetcという1文字を読み込むというものがある。それをDuskul言語で使用したいためだと思われる。

関数expressionList()では、連続している演算子を関数expression()でまとめつつ配列に加えて式の優先順位を()で括ったようにした構文木を配列として納める。
これをすることでプログラムが四則計算の順序と（）を理解することができるようになる。
例を挙げると、
１＋４×３の式の場合に
（１＋（４×３））という風に計算順位を括弧で括った構文木を配列として納める。

・expression.h
expression.hではexpression.cで扱う関数のプロトタイプ宣言を行なっている。

・exptree.c
関数newExpnode()では、プログラム実行時に診断機能を付け加えることができるassertが宣言されている。assertには真となる評価式を渡すことが前提となる。評価式の結果が真の場合にはそのままプログラムが実行していく。評価式の結果が偽の場合に診断メッセージを出力してプログラムを終了する。exptree.cでは、kindの値がトークン型の数字、文字列、変数名である場合にはプログラムをそのまま実行されるが、それ以外の場合にabort.cの関数_assert()が呼び出され診断メッセージを出力して終了する。
assert機能をCで扱う場合にはassert.hをインクルードしないといけないのだが、今回のプログラムの中では、マクロ定義がなされている。
assertの評価式が真の場合にmalloc(size)によりsize分のメモリ領域を確保した後、構造体xpのkindとprefixに元のkindとprefixを代入して構造体xpに戻り値を返している。このsizeが構造体で宣言されている場合はその構造体が占有しているバイト数分を確保する。

関数newOprnode()では、malloc()でメモリ領域を確保した後に関数名の通りに被演算子をxp->operand[0]とxp->operand[1]にそれぞれ代入している。

関数newArgnode()では、まず、変数szに構造体argExpnodeが占有しているバイト数を代入したあとargnumが1より大きい、つまり引数の量が1より大きい場合にその量分のメモリ領域を確保するためにszを増やしている。その後関数の名前やその情報を構造体agpに代入している。

exp_imp.hの構造体_argmExpnodeで*args[1]だけ用意されているのは、引数の個数に応じて配列に入れるためである。34行目の計算では、メモリ領域を引数の個数分だけ変数szに追加することで解決している。

また、この関数の説明で用意された文書には項に相当する構文木（expnodeによる木構造）を返す関数 term()とあった。構文木を返す作業を繰り返すことで、木全体の情報を与えることができるようになり結果的にプログラムの構文の情報を全て渡すことができるようになる。　**分かったような分からんような文ですが、term() は expression() から呼ばれて式を表す木構造を作ります。二項演算子が使われていれば term() から返された木は式全体の部分木になります。**

・functions.c
functionとある通り、関数に対して情報を格納するプログラムである。

functions.cはstatements.hでプロトタイプ宣言されている。まず関数parseProgram()から呼び出され、fgetItem()で得たs.tokenがサブルーチン定義（関数の予約語'func'または手続きの予約語'proc'）または変数宣言（局所(ローカル)変数の予約語'var'）またはサブルーチンの前方参照（予約語'declare'）かによって関数funcDefine()またはstatements.cの関数varsDeclareList()または関数funcDeclare()が呼び出される。

変数宣言以外の時に関数func_header()が呼び出され、処理をした後に関数parameter_list()が呼び出される。

関数parameter_list()では、func '関数名'(引数1、引数2)などの引数を読み込む。引数がない場合（）は0を返す。引数がある場合には（左括弧から）右括弧まで読み込む。
変数宣言の場合はstatements.cの関数varsDeclareList()が呼び出され、varに続く変数名を読み込む。変数宣言が複数存在している場合は、コンマがなくなるまで読み込まれる。

func_header()の中の51行目のint func_proc = isfunc ? id_func : id_proc;
の部分で、なぜint型のfunc_procにbool型のisfuncが代入できるのかと疑問に思ったが、commontype.hにtrueを1、falseを0として定義しているからである。isfunc?とクエスチョンマークがある場合にはnullを許容することができるため、プログラムにエラーが起きない。
kindがid_undefined、つまりまだ読み込まれていない関数である限りその関数名を構造体idpに読み込み、構造体finfに登録する。その後添字（funcindex）を返す。既に読み込まれている関数の場合には登録されている分の添字を返す。

3週目のレポート課題

（1）担当するレポート課題

・expression.c
関数expression()、strExpression() の中では関数ungetItem() が使われているが、関数expressionList() では使われていない。この違いを考察しなさい。

・exptree.c
関数term()内部で、74行目からの if文の処理内容を詳しく説明しなさい。

・functions.c
maxLocalOffsetという変数の役割を考察しなさい。

・expression.c　　**プログラムを見て考えてください**
レポート上部でも書いたが、関数ungetItem()は1文字を読み込む関数であると考えた。関数expressionList()は連続する演算子をまとめて式の計算順位を示した構文木を配列として納める関数である。関数expression()、strExpression() で行なったことをまとめる作業であると考えると、1文字を読み込むという作業はまとめの部分には必要ないものだろう。その違いだと思われる。　**違います。次ページにコメント(★1)**

・exptree.c
s.tokenが変数であり、s.kindがfuncの場合に、接頭語、関数の情報、引数の個数を構造体agpに代入し、戻り値にagpを返す。　**エラーを表示したらもう終了です**
s.kindがprocの時、または未定義の時にはエラーを表示する。
未定義の場合には、エラーを表示した後、情報をstnode_imp.hの構造体varinfoに格納し、構造体termpに登録、戻り値にtermpを返す。
s.tokenが数字である場合は、termpに情報を登録し戻り値にtermpを返す。
この最後の場合は何ですか？　情報はどんな内容？

・functions.c
maxLocalOffsetという変数はidentifiers.hとidtable.cで宣言されている。
dentifiers.hのextern intという宣言の方法はグローバル変数という複数のファイルでも共有して使うことができるものである。
オフセットを調べるとC言語では、データの基準との位置の差分の距離のことであった。
つまりmaxLocalOffsetとはデータの基準とのいちの差分の距離の最大値を示している変数であると思われる。

これだけでは詳しくわからないので、コードを調べていく。

function.cの関数funcDefine()の103行目のコメントにmaxLocalOffset is updateとある。そこでは、blockNestPop()が呼び出されている。

blockNestを調べるとforループを何重にも行う時に階層が分かれていくことの{}の部分という意味であった。

```
for(; ;){
    for(; ;){
        for(; ;){
            :
            :
            :
        }
    }
}
```

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。
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。

popを調べるとスタックからデータを取り出すという意味であった。逆にpushはスタックにデータを追加するという意味であった。
blockNestPop()ではcurrentLocalOffsetがmaxLocalOffsetより大きい時にmaxLocalOffsetにcurrentLocalOffsetの値を代入するとある。ここでネストを除去しているということだろう。
つまり、maxLocalOffsetの変数は、ネストから抜けるという役割を持っていると考える。　　**違います。**

(★1) ある構文や式が終わったかどうかを調べるために、次のトークンを取ってきて、終わりかどうかを調べることがあります。例えば var の後の変数とカンマの並びはそうなっています。この処理では、取ってきたトークンがカンマでなければ宣言の終わりですが、その「カンマでない何かのトークン」は次の何かの宣言や文の先頭として使われると思われます。そこで、ungetItem() を使って、一度取ってきたトークンを「元へ戻し」ます。次にどこかで getItem() を呼び出すと、今元へ戻したトークンが改めて取り出されます。一方、expressionList() で調べているのは (と) で囲まれた式の列なので、')' が来たら終わりであることが分かっています。取り出した ')' はそれ以上使いませんので、ungetItem() を使って戻す必要がないのです。

（2）担当するソースコードの機能拡張について
このグループで行う機能拡張の内容は、
・repeat文
・組込関数
・配列の導入
である。

・repeat文
・expression.c
oppPutOperator()、*expression()、expressionList()でループが存在している。
oppPutOperator()では、
　top > 0 && opp->prec[top - 1] >= prec
である限りループが実行されるので、

repeat
　文…
until (top < 0 || opp->prec[top - 1] < prec)

と変更すればいいだろう。　**C言語に repeat - until はありません。**
*expression()とexpressionList()では、無限ループのため、repeat文は実装する必要はないと思った。

Duskul言語で repeat - until が使えるようにするのであって、
C言語のソースに repeat - until を書き込んでも意味がありません。

・exptree.c
exptree.cでは、ループが存在しなかった。

・function.c
function.cでは条件付きループは、parseProgram()のみあった。
127行目～139行目のループは

repeat
　文…
until(!getEOF())

143行目～149行目のループは

int i = 0;
repeat
　文…
　i++;
until(i < numberOfFunctions)

で良いだろう。

・組込関数
用意するべき関数は、
abs(n)　値nの絶対値を返す。
max(a,b)　値a,bの大きい方の値を返す。
min(a,b)　値a,bの小さい方の値を返す。
random(n)　0 ≦ x < n となる乱数値xを返す。n ≦ 1の場合は常に0を返す。
である。
担当するプログラムには組込関数は入らないと思われる。

式の評価のところに直接関係してきます。

・配列の導入
配列を導入するには、Duskul言語上で配列を読み込めるようにする必要がある。
Duskulでは変数宣言は予約語のvarに続いて変数名を記述するので、私の担当する範囲ではないものと思われる。変更するとすれば、varを読み込むのはfunction.cの133行目で行われるものだと思うため、関数varsDeclareList()があるstatements.cを変更するのだろう。もし変数名の中に'['が存在するならば、mallocで[ ]の中の数字分メモリをとる、とすれば良いのではないか。

配列を使った式があれば、当然、式の構文解析や、
実行時に添字の評価をする必要があります。