プログラミングII

http://bit.ly/Prog3i

言語処理系（2）

前期 第11週

2019/7/2

本日は・・・

構文規則から、自分で作った関数を呼び出して、より複雑な処理をする演算子を作ってみます。

【例】2つの整数から最大値を求める演算子

字句解析プログラムの作成

「calc2.lex」の中身

```
%%
"m"  return MAX;
"+"  return ADD;
"-"  return SUB;
"*"  return MUL;
"/"  return DIV;
"%"  return MOD;
"("  return LP;
")"  return RP;
"\n" return NL;

[0-9]+ {
  yylval = atoi(yytext);
  return NUMBER;
}
%%
```

最大値を求める演算子「m」を追加する

構文解析プログラムの作成

「calc2.yacc」の中身

```
%token NL LP RP NUMBER
%token ADD SUB MUL DIV MOD MAX
%%
list :
     | list expr NL { printf("%d\n", $2);}
     ;
expr : expr ADD expr { $$ = $1 + $3;}
     | expr SUB expr { $$ = $1 - $3;}
     | expr MUL expr { $$ = $1 * $3;}
     | expr DIV expr { $$ = $1 / $3;}
     | expr MOD expr { $$ = $1 % $3;}
     | MAX expr expr { $$ = max($2, $3); }
     | LP expr RP    { $$ = $2;}
     | NUMBER        { $$ = $1;}
     ;
%%
#include "lex.yy.c"
#include "myproc.h"
```

最大値を求める演算子のトークンを追加

最大値を求める構文を追加

最大値を求める関数maxを呼び出す

ヘッダファイルの読み込みを追加する

自分で作る関数の定義

ヘッダファイル「myproc.h」の中身

```
int max(int x, int y);
```

最大値を求める関数のプロトタイプ宣言を追加

Cプログラムファイル「myproc.c」の中身

```
int max(int x, int y)
{
    if(x > y) return x;
    else return y;
}
```

計算結果を戻り値とする

最大値を求める関数の定義を追加

プログラム生成から実行までの流れ

1. yaccの実行（y.tab.cが生成される）

```
$ yacc calc2.yacc
```

2. lexの実行（lex.yy.cが生成される）

```
$ flex calc2.lex
```

3. Cコンパイラ実行（警告が出るが今回は無視）

```
$ cc y.tab.c myproc.c -ly -lfl
```

4. 実行して動作を確かめる

```
$ ./a.out
```

自分で作った関数が定義されているCプログラムファイルも一緒にコンパイルする

【課題の準備】

演習室で作業する前に、以下のコマンドを
入れるだけで準備が完了する

```
$   mygitclone   「自分のGitHubユーザ名」
$   cd prog3i-(ユーザ名)
$   ./myconf
```

※本体をシャットダウンするまでは、
上記「mygitclone」と「myconf」の設定は有効です

【課題の準備】

以下の流れで、課題のプログラムを作るためのフォルダを準備しましょう。

1. 端末を起動して、以下のコマンドを実行して前期第11週のフォルダを作る

```
$ cd prog3i-(ユーザ名)        (←既に移動しているなら不要)
$ mkdir week111
$ cd week111
```

【練習11-1】

サンプルプログラムから「最大値の演算」が追加された電卓の動作を確認して下さい。

[実行結果]

```
$ ./a.out
m 5 3          (←演算式を入力すると)
  5            (←最大値が表示される)
m 2 8
  8
```

【課題11-1】

3つの整数の中から最大値を求める演算子「t」を追加して下さい。

- calc.lexに、演算子「t」と識別子TRIの字句定義を追記する
- calc.yaccに、この演算に関する構文を追加する
  演算子の後にexprが3個続く【例】「t ５ ７ 2」は最大値7が求まる
- myproc.hにこの演算で呼び出す関数のプロトタイプ宣言を追加する

```
int tri(int x, int y, int z);
```

- myproc.cに関数triの定義を追加する
  - 仮引数x, y, zを何回か比較して、最大値を戻す処理を作る

【課題11-1】

[実行結果]

```
$ ./a.out
t 7 4 2                (←演算式を入力すると)
   7                   (←3つ整数の最大値が表示される)
t 3 9 5
   9
t 1 4 8
   8
```

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
$ git commit -m “課題11-1提出”
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名)

【課題11-2】

1つの整数xを指定して、1～xの合計値を求める演算子「s」を追加して下さい。

- calc.lexに、演算子「s」と識別子SUMの字句定義を追記する
- calc.yaccに、この演算に関する構文を追加する
  演算子の後に、exprが1個続く　【例】「s 4」は10が求まる
- myproc.hにこの演算で呼び出す関数のプロトタイプ宣言を追加する

```
int sum(int x);
```

- myproc.cに関数sumの定義を追加する
  - 1から仮引数xまでの合計を求めるfor文を作り、合計結果を戻す

【課題11-2】

[実行結果]

```
$ ./a.out
s 4              (←演算式を入力すると)
 10              (←1～4の合計結果が表示される)
s 10
 55
```

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
$ git commit -m “課題11-2提出”
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名)

【課題11-3】

2つの整数s, eを指定して、s～eの合計値を求める演算子「r」を追加して下さい。

- calc.lexに、演算子「r」と識別子RANの字句定義を追記する
- calc.yaccに、この演算に関する構文を追加する
  演算子の後に、exprが2個続く　【例】「r ４ 6」は15が求まる
- myproc.hにこの演算で呼び出す関数のプロトタイプ宣言を追加する

```
int range(int s, int e);
```

- myproc.cに関数rangeの定義を追加する
  - 仮引数sからeまでの合計を求めるfor文を作り、合計結果を戻す

【課題11-3】

[実行結果]

```
$ ./a.out
r 4 6          (←演算式を入力すると)
  15           (←4～6の合計結果が表示される)
r 2 10
  54
```

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
$ git commit -m “課題11-3提出”
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名)

できた人は…

これまで作った演算子を組合わせて確認してみましょう

例えば…

- ３ + m ８ ２
- t (2+4) 5 1
- (s 3) + (m 7 5)
- (r 6 9) + (r 2 (2 + 2))
- などなど