Three Address Code generation using Lex, Yacc
This is a discussion on Three Address Code generation using Lex, Yacc within the **C Programming** forums, part of the General Programming Boards category; We are trying to write a C compiler using LEX and YACC Given the production rules, it uses syntax directed ...

11-04-2008 #1

Raman
Registered User
Join Date: Nov 2008
Posts: 5

Three Address Code generation using Lex, Yacc

We are trying to write a C compiler using LEX and YACC
Given the production rules, it uses syntax directed translation to generate three address code.

TAC.l

ode:

```
%{

#include <stdi
#include <stri
%}

LETTER [A-Za-z
DIGIT [0-9]
OPERATOR [+-*/
FOR "for"
WHILE "while"
IF "if"
DO "do"
ELSE "else"
THEN "then"

%%
{LETTER} {strc
return LETTER;
{DIGIT}+"."{DI
return FLOAT;}
{DIGIT}+ {strc
return INTEGER
{FOR} {return
{WHILE} {retur
{DO} {return D
{IF} {return I
{THEN} {returr
{ELSE} {returr
[ \n\t]* ;
"end" {return
%%
```

TAC.y

Code:

```
%{

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

#include<math.h>

#define START 100

typedef struct s

{

int* true;

int* false;

int* next;

int quad;

char place[5];

}ETYPE;

int nextquad=START;

char code[25][50];

%}


%union

{

char id[10];

ETYPE eval;

}


%left "|"

%left "&"

%left "!"

%left "<" ">"

%left "+" "-"

%left "*" "/"

%left "(" ")"


%right "="


%token <id> LETTER INTEGER FLOAT
```

```
%token FOR WHILE DO IF THEN ELSE

%type <eval> PROGRAM BLOCK STATEMENTS ASSIGN COND L\

%start PROGRAM

%%

PROGRAM: BLOCK {

int i;

Backpatch($1.next,nextquad);

for(i=START;i<nextquad;i++)

printf("\n%s",code[i-START]);

printf("\n%d\n",nextquad);

}

BLOCK: DO M BLOCK WHILE M COND {

Backpatch($6.true,$2.quad);

$$.next=$6.false;

}

| FOR ASSIGN M COND M ASSIGN Q BLOCK {

Backpatch($8.next,$5.quad);

Backpatch($6.next,$3.quad);

Backpatch($4.true,$7.quad);

$$.next=$4.false;

sprintf(code[nextquad-START],"%d\tgoto %d",nextquad

Gen();

}

| WHILE M COND M BLOCK {

Backpatch($5.next,$2.quad);

Backpatch($3.true,$4.quad);

$$.next=$3.false;

sprintf(code[nextquad-START],"%d\tgoto %d",nextquad

Gen();

}

| IF COND M BLOCK {

Backpatch($2.true,$3.quad);

$$.next=Merge($2.false,$4.next);

}

| IF COND M BLOCK N ELSE M BLOCK {
```

```
Backpatch($2.true,$3.quad);

Backpatch($2.false,$7.quad);

$$.next=Merge($4.next,$5.next);

$$.next=Merge($$.next,$8.next);

}

| '{' STATEMENTS '}' {

$$.next=$2.next;

}

| ASSIGN ';' {

$$.next=(int*)malloc(sizeof(int)*15);

$$.next[0]=0;

}

| E { }


STATEMENTS: STATEMENTS M BLOCK {

Backpatch($1.next,$2.quad);

$$.next=$3.next;

}

| BLOCK {

$$.next=$1.next;

}


ASSIGN: LVAL '=' E {

sprintf(code[nextquad-START],"%d\t%s = %s",nextquad

Gen();

}

| E { }


LVAL: LETTER {strcpy($$.place,$1);}


E: E '+' E {

strcpy($$.place,Newtemp());

sprintf(code[nextquad-START],"%d\t%s = %s + %s",next

Gen();

}

| E '-' E {

strcpy($$.place,Newtemp());

sprintf(code[nextquad-START],"%d\t%s = %s - %s",next

Gen();

}
```

```
230
231  | E '*' E {
232
233  strcpy($$.place,Newtemp());
234
235  sprintf(code[nextquad-START],"%d\t%s = %s * %s",next
236
237  Gen();
238
239  }
240
241  | E '/' E {
242
243  strcpy($$.place,Newtemp());
244
245  sprintf(code[nextquad-START],"%d\t%s = %s / %s",next
246
247  Gen();
248
249  }
250
251  | '-' E %prec '*' {
252
253  strcpy($$.place,Newtemp());
254
255  sprintf(code[nextquad-START],"%d\t%s = - %s",nextqua
256
257  Gen();
258
259  }
260
261  | LETTER {
262
263  strcpy($$.place,$1);
264
265  }
266
267  | INTEGER {
268
269  strcpy($$.place,$1);
270
271  }
272
273  | FLOAT {
274
275  strcpy($$.place,$1);
276
277  }
278
279
280
281  COND: COND '&' M COND {
282
283  Backpatch($1.true,$3.quad);
284
285  $$.true=$4.true;
286
287  $$.false=Merge($1.false,$4.false);
288
289  }
290
291  | COND '|' M COND {
292
293  Backpatch($1.false,$3.quad);
294
295  $$.true=Merge($1.true,$4.true);
296
297  $$.false=$4.false;
298
299  }
300
301  | '!' COND {
302
303  $$.true=$2.false;
304
305  $$.false=$2.true;
306
307  }
308
```

```
| '(' COND ')' {

$$.true=$2.true;

$$.false=$2.false;

}

| E '<' E {

$$.true=Makelist(nextquad);

$$.false=Makelist(nextquad+1);

sprintf(code[nextquad-START],"%d\tif %s < %s goto "

Gen();

sprintf(code[nextquad-START],"%d\tgoto ",nextquad);

Gen();

}

| E '>' E {

$$.true=Makelist(nextquad);

$$.false=Makelist(nextquad+1);

sprintf(code[nextquad-START],"%d\tif %s > %s goto "

Gen();

sprintf(code[nextquad-START],"%d\tgoto ",nextquad);

Gen();

}

| E {

$$.true=Makelist(nextquad);

$$.false=Makelist(nextquad+1);

sprintf(code[nextquad-START],"%d\tif %s goto ",next

Gen();

sprintf(code[nextquad-START],"%d\tgoto ",nextquad);

Gen();

}


M: {

$$.quad=nextquad;

}

N: {

$$.next=Makelist(nextquad);

sprintf(code[nextquad-START],"%d\tgoto ",nextquad);

Gen();

}

Q: {

$$.next=Makelist(nextquad);

sprintf(code[nextquad-START],"%d\tgoto ",nextquad);
```

```
388
389  Gen();
390
391  $$.quad=nextquad;
392
393  }
394
395
396
397  %%
398
399
400
401  /*
402  #include "lex.yy.c"
403
404
405
406  main()
407
408  {
409
410  yyparse();
411
412  }
413
414  */
415
416  yyerror(char *errmesg)
417
418  {
419
420  printf("\n%s\n",errmesg);
421
422  }
423
424
425
426  char* Newtemp()
427
428  {
429
430  static int count=1;
431
432  char* ch=(char*)malloc(sizeof(char)*5);
433
434  sprintf(ch,"T%d",count++);
435
436  return ch;
437
438  }
439
440
441
442  int* Makelist(int nquad)
443
444  {
445
446  int* list=(int*)malloc(sizeof(int)*15);
447
448  list[0]=nquad;
449
450  list[1]=0;
451
452  return list;
453
454  }
455
456
457
458  int* Merge(int* list1,int* list2)
459
460  {
461
462  int* temp=list1,count1=0,count2=0;
463
464  while(list1[count1]!=0) count1++;
465
466  while(list1[count2]!=0)
```

```
{

list1[count1]=list2[count2];

count1++;

count2++;

}

return temp;

}


void Backpatch(int* list,int nquad)

{

char addr[10];

sprintf(addr,"%d",nquad);

while(*list!=0)

{

int index=*list++-START;

strcat(code[index],addr);

}

}


void Gen()

{

nextquad++;

}
```

main.c

Code:

```
#include "y.tab.c"
#include "lex.yy.c"

int main() {
yyparse();
return 0;
}
```

lex TAC.l
yacc TAC.y
gcc main.c -ll

./a.out
while (a<b) { a=a+b; c=c+b; }
Segmentation Fault

Can someone please help out?

11-04-2008 #2

C_ntua

Registered User

Join Date: Jun 2008
Posts: 1,853

...joking of course?

(the code is too big. Indent at least. Use debugging in order to find a segmentation fault. At least find out where it crashes or try different input and see what happens. Or wait for a yacc expert)

« Previous Thread | Next Thread »