NSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE COMPUTO

PRACTICA 7 funciones y procedimientos

Opción calculadora de vectores

COMPILADORES

Grupo: 3CM17

ALUMNO: MORA GUZMÁN JOSE ANTONIO

FECHA ENTREGA: Miercoles 1 Diciembre 2021

Descripción

En esta practica se han añadido funciones y procedimientos. Se añadieron mas funciones a code.c y también mas símbolos gramaticales.

Código

Se modifico la gramática y se añadieron mas símbolos gramaticales, y mas elementos a la unión

```
%union{
        Symbol *sym; /*apuntador de la tabla de símbolos*/
        Inst *inst; /* instrucción de máquina*/
        double val;
        Vector *vec;
        int narg;/*Nùmero de argumentos*/
}


%token <sym> VAR BLTIN INDEF VEC NUMERO WHILE IF ELSE PRINT STRING
%token <sym> FUNCTION PROCEDURE RETURN FUNC PROC READ
%token <narg> ARG
%token <val> NUMBER
//Agregue la siguiente linea
%type <inst> stmt asgn exp stmtlist cond while if end prlist begin
%type <sym> vector
%type <sym> procname
%type <narg> arglist
//%type <vec> exp asgn
%left '+' '-'
%left '*'
%left '#' '.'
//Agregue la siguiente linea
%left OR
%left AND
%left GT GE LT LE EQ NE
%left NOT
//Sección de reglas de yacc
%%
        list:
                | list '\n'
                | list defn  '\n'
                | list asgn '\n'{code2(pop,STOP); return 1;}
                | list stmt '\n' {code(STOP); return 1;}
                | list exp '\n'{code2(print,STOP); return 1;}
                | list error '\n' {yyerror;}
                ;
```

```
64      asgn: VAR '=' exp {$$ = $3; code3(varpush,(Inst)$1,assign);}
65              | ARG '=' exp {defonly("$");code2(argassign,(Inst)$1); $$ = $3;}
66              ;
67
68      stmt: exp                                                       {code(pop);}
69              | RETURN                                                {defonly("return");code(procret);}
70              | RETURN exp                                    {defonly("return");$$=$2;code(funcret);}
71
72              | PRINT prlist                                  { $$  = $2;}
73              | while cond stmt end                   {
74                                                                      ($1)[1] = (Inst)$3;  /* cuerpo de la iteración*/
75                                                                      ($1)[2] = (Inst)$4;  /* terminar si la condición no
    se cumple*/}
76
77              | if cond stmt end                              { /* proposición if que no emplea else*/
78
79                                                                      ($1)[1] = (Inst)$3; /* parte then */
80                                                                      ($1)[3] = (Inst)$4; } /* terminar si la condición no
    se cumple */
81
82              | if cond stmt end ELSE stmt end       { /* proposición if ocn parte else*/
83                                                                      ($1)[1] = (Inst)$3; /*parte then*/
84                                                                      ($1)[2] = (Inst)$6; /*paret else*/
85                                                                      ($1)[3] = (Inst)$7; } /*terminar si la condición no
    se cumple*/
86              | '{' stmtlist '}'                      { $$ = $2;}
87
88      ;
89
90      cond: '(' exp ')'                                       {code(STOP); $$ = $2;}
91      ;
92
93      while: WHILE                                            { $$ = code3(whilecode,STOP,STOP);}
94      ;
95
96      if: IF                                                          { $$ = code(ifcode); code3(STOP,STOP,STOP);}
97      ;
```

```
99      end: /* nada */                                         {code(STOP); $$ = progp; }
100     ;
101
102     stmtlist: /* nada */                                    {$$ = progp;}
103                     | stmtlist '\n'
104                     | stmtlist stmt
105     ;
106
107     exp: vector                                                     {$$ = code2(constpush, (Inst)$1);}
108             |VAR                                                    {$$ = code3(varpush,(Inst)$1,eval);}
109             |ARG                                                    {defonly("$"); $$ = code2(arg,(Inst)$1);}
110             |asgn
111             |FUNCTION begin '(' arglist ')'         { $$ = $2; code3(call,(Inst)$1,(Inst)$4);}
112             |READ '(' VAR ')'                               {$$= code2(varread,(Inst)$3);}
113             |BLTIN '(' exp ')'                              {code2(bltin,(Inst)$1->u.ptr);}
114             |exp '+' exp                                    {code(add);}
115             |exp '-' exp                                    {code(sub);}
116             |exp '.' exp                                    {code(punto);}
117             |exp '*' NUMBER                                 {code(mul);}
118             |NUMBER '*' exp                                 {code(mul);}
119             |exp '#' exp                                    {code(cruz);}
120             |exp GT exp                                     {code(gt);}
121             |exp GE exp                                     {code(ge);}
122             |exp LT exp                                     {code(lt);}
123             |exp LE exp                                     {code(le);}
124             |exp EQ exp                                     {code(eq);}
125             |exp NE exp                                     {code(ne);}
126             |exp AND exp                                    {code(and);}
127             |exp OR exp                                     {code(or);}
128             |NOT exp                                                {$$ = $2; code(not);}
129             |PROCEDURE begin '(' arglist ')'        { $$ = $2; code3(call,(Inst)$1,(Inst)$4);}
130     ;
131
132     begin:/*nada */                                         {$$ = progp;}
133     ;
134
135     prlist: exp                                                     {code(prexpr);}
```

```
135         prlist: exp                                                 {code(prexpr);}
136                 | STRING                                            {$$  = code2(prstr,(Inst)$1);}
137                 | prlist ',' exp                                  {code(prexpr);}
138                 | prlist ',' STRING                             {code2(prstr,(Inst)$3);}
139         ;
140
141         defn:  FUNC procname                                        {$2->type=FUNCTION; indef =1;}
142                                                                             '(' ')' stmt {code(procret);define($2);indef=0;}
143                 |PROC procname                                              {$2->type = PROCEDURE; indef = 1;}
144                                                                             '(' ')' stmt {code(procret);define($2); indef=0;}
145
146         ;
147
148         procname: VAR
149                 | FUNCTION
150                 | PROCEDURE
151         ;
152
153         arglist: /*nada*/                                           {$$=0;}
154                 | exp                                                       {$$ = 1;}
155                 | arglist ',' exp                                   {$$ = $1 + 1;}
156         ;
157
158         vector:'['NUMBER NUMBER NUMBER']'           {Vector *v = creaVector(3);
159                                                                             v->vec[0] = $2;
160                                                                             v->vec[1] = $3;
161                                                                             v->vec[2] = $4;
162                                                                             $$ = install("",VEC,v);}
163         ;
164
165 %%
```

Tambien se añade mas codigo a code.c, ahi se realizo todo lo necesario para ejecutar las funciones

```
315 // CÓDIGO AÑADIDO EN LA PRÁCTICA 7
316 void define(Symbol *sp)
317 {
318         sp->u.defn = (Inst)progbase; /* principio de c◆digo */
319         progbase = progp;                       /* el siguiente c◆digo comienza aqu◆ */
320 }
321
322 void call()
323 {
324         Symbol *sp = (Symbol *)pc[0]; /*entrada en la tabla de s◆mbolos*/
325         if (fp++ >= &frame[NFRAME - 1])
326                 execerror(sp->name, "call nested too deeply");
327         fp->sp = sp;
328         fp->nargs = (int)pc[1];
329         fp->retpc = pc + 2;
330         fp->argn = stackp - 1; /*◆ltimo argumento*/
331         execute(sp->u.defn);
332         returning = 0;
333 }
334
335 void ret()
336 {
337         int i;
338         for (i = 0; i < fp->nargs; i++)
339                 pop(); /*saca argumentos*/
340         pc = (Inst *)fp->retpc;
341         --fp;
342         returning = 1;
343 }
344
345 void funcret()
346 {
347         Datum d;
348         if (fp->sp->type == PROCEDURE)
349                 execerror(fp->sp->name, "(proc) returns value");
```

```
350         d = pop(); /* preservar el valor de regreso a ♦a funcion*/
351         ret();
352         push(d);
353 }
354
355 void procret()
356 {
357         if (fp->sp->type == FUNCTION)
358                 execerror(fp->sp->name, "(func) return no value");
359         ret();
360 }
361
362 Vector **getarg()
363 {
364         int nargs = (int)*pc++;
365         if (nargs > fp->nargs)
366                 execerror(fp->sp->name, "not enough arguments");
367         return &fp->argn[nargs - fp->nargs].val;
368 }
369
370 void arg()
371 { /*meter el aergumento en la pila*/
372         Datum d;
373         d.val = *getarg();
374         push(d);
375 }
376
377 void argassign()
378 {
379         Datum d;
380         d = pop();
381         push(d);
382         *getarg() = d.val;
383 }
384
```

```
385 void prstr()
386 {
387         printf("%s", (char *)*pc++);
388 }
389
390 void varread()
391 {
392         Datum d;
393         extern FILE *fin;
394         Symbol *var = (Symbol *)*pc++;
395 Again:
396         switch (fscanf(fin, "%lf", &var->u.val))
397         {
398         case EOF:
399                 if (moreinput())
400                         goto Again;
401                 d.val = var->u.val = NULL;
402                 break;
403         case 0:
404                 execerror("non-number read into", var->name);
405                 break;
406         default:
407                 d.val = NULL;
408                 break;
409         }
410         var->type = VAR;
411         push(d);
412 }
```

Y como debemos generar marcos de funcion se creo una estructura que contiene informacion de cada funcion, y de igual manera se creo la pila de llamadas donde se van a ir apilando los marcos de funcion

```
typedef struct Frame
{                               /*nivel en la pila si hay llamada a proc/fun*/
        Symbol *sp;  /*entrada en la tabla de s◆mbolos*/
        Inst *retpc; /*donde continuar despu◆s de regresar*/
        Datum *argn; /*n-◆simo argumento en la pila*/
        int nargs;   /*n◆mero de argumentos*/
} Frame;

#define NFRAME 100
Frame frame[NFRAME];
Frame *fp;
```

Pruebas del programa

```
tony@tony-Aspire-E5-523: ~/Escritorio/compiladores/Practica 7
tony@tony-Aspire-E5-523:~/Escritorio/compiladores/Practica 7$ yacc -d hoc4.y
hoc4.y:32 parser name defined to default :"parse"
conflicts:  38 shift/reduce
tony@tony-Aspire-E5-523:~/Escritorio/compiladores/Practica 7$ gcc y.tab.c code.c
 vector_cal.c math.c symbol.c init.c -lm -w
tony@tony-Aspire-E5-523:~/Escritorio/compiladores/Practica 7$ ./a.out
a=[1 0 0]
[1.000000 0.000000 0.000000 ]
b=[2 1 3]
[2.000000 1.000000 3.000000 ]
a+b
[3.000000 1.000000 3.000000 ]
func operaciones(){print $1 + $2 print $1 - $2 print $1 . $2 print $1 # $2}
operaciones(a,b)
[3.000000 1.000000 3.000000 ]
[-1.000000 -1.000000 -3.000000 ]
[2.000000 ]
[0.000000 -3.000000 1.000000 ]
```

Conclusiones

Esta practica ya deja practicamente terminado nuestro compilador puesto que al realizar todas las practicas se le fue dando forma poco a poco, finalmente agregamos las funciones y procedimientos, esta practica al ya tener la experiencia de las anteriores se me hizo un tanto facil, solo fue agregar unas cosas que mencioné en el documento

Link video

https://youtu.be/z7GY6AZai7A