# Compiladores

- Práctica 07: Funciones y procedimientos -

Grupo 3CM7

Vargas Romero Erick Efraín
Prof. Tecla Parra Roberto

Instituto Politécnico Nacional
Escuela Superior de Cómputo
Juan de Dios Bátiz, nueva industrial Vallejo
07738 ciudad de México

# Chapter 1

# Práctica 07

## 1.1 Funciones y procedimientos

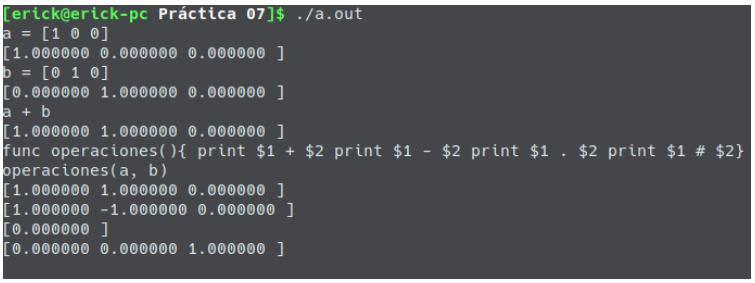
### 1.1.1 Descripción

En esta séptima práctica se han añadido funciones y procedimientos. Para realizar esto se han añadido más funciones a code.c, y se han añadido más símbolos gramaticales.

### 1.1.2 Ejemplos

A continuación muestro una captura de pantalla, la cual muestra la compilación del código en yacc, y también la compilación del código que es generado en c y finalmente la ejecución del programa.

**Figure 1.1:** Ejemplo de una función




### 1.1.3 Código

Nuevamente se ha modificado la gramática, se han añadido más símbolos gramaticales, se han añadido más elementos a la unión, esto se muestra a continuación

```
1 %union{
2   Symbol *sym; /*apuntador de la tabla de ísmbolos*/
3   Inst *inst; /* óinstruccin de ámquina*/
```

```
4    double val;
5    Vector *vec;
6    int narg;/*ùNmero de argumentos*/
7  }
8
9
10 %token <sym> VAR BLTIN INDEF VEC NUMERO WHILE IF ELSE PRINT STRING
11 %token <sym> FUNCTION PROCEDURE RETURN FUNC PROC READ
12 %token <narg> ARG
13 %token <val> NUMBER
14 //Agregue la siguiente linea
15 %type <inst> stmt asgn exp stmtlist cond while if end prlist begin
16 %type <sym> vector
17 %type <sym> procname
18 %type <narg> arglist
19 //%type <vec> exp asgn
20 %left '+' '-'
21 %left '*'
22 %left '#' '.'
23 //Agregue la siguiente linea
24 %left OR
25 %left AND
26 %left GT GE LT LE EQ NE
27 %left NOT
28 //óSeccin de reglas de yacc
29 %%
30   list:
31     | list '\n'
32     | list defn '\n'
33     | list asgn '\n'{code2(pop,STOP); return 1;}
34     | list stmt '\n' {code(STOP); return 1;}
35     | list exp '\n'{code2(print,STOP); return 1;}
36     | list error '\n' {yyerror;}
37     ;
38
39   asgn: VAR '=' exp {$$ = $3; code3(varpush,(Inst)$1,assign);}
40     | ARG '=' exp {defonly("$");code2(argassign,(Inst)$1); $$ = $3;}
41     ;
42
43   stmt: exp                    {code(pop);}
44     | RETURN                  {defonly("return");code(procret);}
45     | RETURN exp               {defonly("return");$$=$2;code(funcret);}
46     | PROCEDURE begin '(' arglist ')'    { $$ = $2; code3(call,(Inst)$1,(
         Inst)$4);}
47     | PRINT prlist              { $$  = $2;}
48     | while cond stmt end          {
49                         ($1)[1] = (Inst)$3;  /* cuerpo de la óiteracin*/
50                         ($1)[2] = (Inst)$4;  /* terminar si la ócondicin
                               no se cumple*/}
51
52     | if cond stmt end            { /* óproposicin if que no emplea else*/
53
54                         ($1)[1] = (Inst)$3; /* parte then */
```

```
55                      ($1)[3] = (Inst)$4; } /* terminar si la ócondicin
                            no se cumple */
56
57    | if cond stmt end ELSE stmt end  { /* óproposicin if ocn parte else
          */
58                      ($1)[1] = (Inst)$3; /*parte then*/
59                      ($1)[2] = (Inst)$6; /*paret else*/
60                      ($1)[3] = (Inst)$7; } /*terminar si la ócondicin
                            no se cumple*/
61    | '{' stmtlist '}'          { $$ = $2;}
62
63  ;
64
65  cond: '(' exp ')'             {code(STOP); $$ = $2;}
66  ;
67
68  while: WHILE              { $$ = code3(whilecode,STOP,STOP);}
69  ;
70
71  if: IF                  { $$ = code(ifcode); code3(STOP,STOP,STOP);}
72  ;
73
74  end: /* nada */              {code(STOP); $$ = progp; }
75  ;
76
77  stmtlist: /* nada */          {$$ = progp;}
78      | stmtlist '\n'
79      | stmtlist stmt
80  ;
81
82  exp: vector               {$$ = code2(constpush, (Inst)$1);}
83    |VAR                {$$ = code3(varpush,(Inst)$1,eval);}
84    |ARG                {defonly("$"); $$ = code2(arg,(Inst)$1);}
85    |asgn
86    |FUNCTION begin '(' arglist ')'   { $$ = $2; code3(call,(Inst)$1,(
          Inst)$4);}
87    |READ '(' VAR ')'           {$$= code2(varread,(Inst)$3);}
88    |BLTIN '(' exp ')'          {code2(bltin,(Inst)$1->u.ptr);}
89    |exp '+' exp            {code(add);}
90    |exp '-' exp            {code(sub);}
91    |exp '.' exp            {code(punto);}
92    |exp '*' NUMBER           {code(mul);}
93    |NUMBER '*' exp           {code(mul);}
94    |exp '#' exp            {code(cruz);}
95    |exp GT exp             {code(gt);}
96    |exp GE exp             {code(ge);}
97    |exp LT exp             {code(lt);}
98    |exp LE exp             {code(le);}
99    |exp EQ exp             {code(eq);}
100   |exp NE exp             {code(ne);}
101   |exp AND exp            {code(and);}
102   |exp OR exp             {code(or);}
103   |NOT exp                {$$ = $2; code(not);}
```

```
104   ;
105
106   begin:/*nada */              {$$ = progp;}
107   ;
108
109   prlist: exp                   {code(prexpr);}
110     | STRING                  {$$  = code2(prstr,(Inst)$1);}
111     | prlist ',' exp            {code(prexpr);}
112     | prlist ',' STRING           {code2(prstr,(Inst)$3);}
113   ;
114
115   defn:  FUNC procname            {$2->type=FUNCTION; indef =1;}
116                         '(' ')' stmt {code(procret);define($2);indef=0;}
117     |PROC procname              {$2->type = PROCEDURE; indef = 1;}
118                         '(' ')' stmt {code(procret);define($2); indef=0;}
119
120   ;
121
122   procname: VAR
123     | FUNCTION
124     | PROCEDURE
125   ;
126
127   arglist: /*nada*/             {$$=0;}
128     | exp                     {$$ = 1;}
129     | arglist ',' exp           {$$ = $1 + 1;}
130   ;
131
132   vector:'['NUMBER NUMBER NUMBER']'    {Vector *v = creaVector(3);
133                         v->vec[0] = $2;
134                         v->vec[1] = $3;
135                         v->vec[2] = $4;
136                         $$ = install("",VEC,v);}
137   ;
138
139 %%
```

También se ha añadido más código a code.c en donde realizamos todas las acciones que sean necesarias para ejecutar las funciones.

```
1  void define(Symbol *sp)
2  {
3    sp->u.defn = (Inst)progbase; /* principio de ócdigo */
4    progbase = progp;       /* el siguiente ócdigo comienza íaqu */
5  }
6
7  void call()
8  {
9    Symbol *sp = (Symbol *)pc[0]; /*entrada en la tabla de ísmbolos*/
10   if (fp++ >= &frame[NFRAME - 1])
11     execerror(sp->name, "call nested too deeply");
12   fp->sp = sp;
13   fp->nargs = (int)pc[1];
14   fp->retpc = pc + 2;
```

```
15    fp->argn = stackp - 1; /*último argumento*/
16    execute(sp->u.defn);
17    returning = 0;
18  }
19
20  void ret()
21  {
22    int i;
23    for (i = 0; i < fp->nargs; i++)
24      pop(); /*saca argumentos*/
25    pc = (Inst *)fp->retpc;
26    --fp;
27    returning = 1;
28  }
29
30  void funcret()
31  {
32    Datum d;
33    if (fp->sp->type == PROCEDURE)
34      execerror(fp->sp->name, "(proc) returns value");
35    d = pop(); /* preservar el valor de regreso a la funcion*/
36    ret();
37    push(d);
38  }
39
40  void procret()
41  {
42    if (fp->sp->type == FUNCTION)
43      execerror(fp->sp->name, "(func) return no value");
44    ret();
45  }
46
47  Vector **getarg()
48  {
49    int nargs = (int)*pc++;
50    if (nargs > fp->nargs)
51      execerror(fp->sp->name, "not enough arguments");
52    return &fp->argn[nargs - fp->nargs].val;
53  }
54
55  void arg()
56  { /*meter el aergumento en la pila*/
57    Datum d;
58    d.val = *getarg();
59    push(d);
60  }
61
62  void argassign()
63  {
64    Datum d;
65    d = pop();
66    push(d);
67    *getarg() = d.val;
```

```
68 }
69
70 void prstr()
71 {
72   printf("%s", (char *)*pc++);
73 }
74
75 void varread()
76 {
77   Datum d;
78   extern FILE *fin;
79   Symbol *var = (Symbol *)*pc++;
80 Again:
81   switch (fscanf(fin, "%lf", &var->u.val))
82   {
83   case EOF:
84     if (moreinput())
85       goto Again;
86     d.val = var->u.val = NULL;
87     break;
88   case 0:
89     execerror("non-number read into", var->name);
90     break;
91   default:
92     d.val = NULL;
93     break;
94   }
95   var->type = VAR;
96   push(d);
97 }
```

Como se deben generar marcos de función se ha creado una estructura la cual contendrá la información de cada función. También se ha creado la pila de llamadas en la cual iremos apilando los marcos de función.

```
1  Inst *progbase = prog; /* empieza el subprograma actual*/
2  int returning;        /* 1 si ve óproposicin  return */
3
4  typedef struct Frame
5  {         /*nivel en la pila si hay llamada a proc/fun*/
6    Symbol *sp;  /*entrada en la tabla de ísmbolos*/
7    Inst *retpc; /*donde continuar édespus de regresar*/
8    Datum *argn; /*né-simo argumento en la pila*/
9    int nargs;   /*únmero de argumentos*/
10 } Frame;
11
12 #define NFRAME 100
13 Frame frame[NFRAME];
14 Frame *fp;
```