Ejemplos de función genCode para Código P

[Louden, 2004]

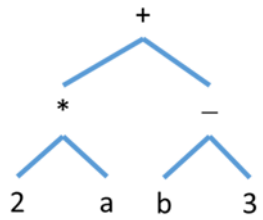
Código P (L 404)

El código P comenzó como un código ensamblador objetivo estándar producido por varios compiladores de Pascal en la década de 1970 y principio de la de 1980. Fue diseñado para ser el código real de una máquina de pila hipotética, denominada Máquina P, para que la fue escrito un intérprete en varias máquinas reales.

Como el código P fue diseñado para ser directamente ejecutable, contiene una descripción implícita de un ambiente de ejecución particular que incluye tamaños de datos, además de mucha información específica para la máquina P, que se debe conocer si se desea que un programa en código P sea comprensible.

Para nuestros propósitos la Máquina P está compuesta por una memoria de código, una memoria de datos (cuyo tamaño no se especifica) para variables nombradas y una pila para datos temporales, junto con cualquier registro que sea necesario para mantener la pila y apoyar la ejecución.

Como un ejemplo, considere el código P para la siguiente expresión: **`2*a+(b-3)`**. Su AST sería:




```
ldc 2       ; carga la constante 2 a la pila
lod a       ; carga el valor de la variable a en la pila
mpi         ; multiplicación entera
lod b       ; carga el valor de la variable b
ldc 3       ; carga la constante 3
sbi         ; resta entera
adi         ; adición de enteros
```

`mpi`, extrae los dos valores superiores de la pila, los multiplica en orden inverso y mete el resultado a la pila.

`sbi` y **`adi`**, hacen lo mismo que mpi pero resta y suma los valores, respectivamente, en lugar de multiplicarlos.

Para conocer más sobre código P busque algunas referencias en la red.

A continuación se da un ejemplo de cómo se haría la función **genCode()** para generar código P para:

- Expresión simple
- Arreglo
- Función if y while
- Declaración y llamada a una función

Gramática de expresiones simple (L 407)

exp → **id** **=** *exp* | *aexp*
aexp → *aexp* **+** *factor* | *factor*
factor → ***(*** *exp* ***)*** | **num** | **id**

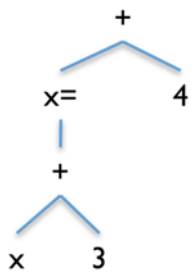
Código genCode()

```
// Es para código P pero da la idea de la estructura (L 411)
// emitCode es un procedimiento para generar una línea simple de código P
// strval es la propiedad que contiene el tokenstring

void genCode(SyntaxTree t) {
      string codestr; // contiene una línea de código
      if (t != NULL) {
            switch (t->kind) {
                  case OpKind:
                        switch (t->op) {
                              case Plus:
                                    genCode(t->leftChild);
                                    genCode(t->rightChild);
                                    emitCode("adi");
                                    break;
                              case Assign:
                                    sprintf(codestr, "%s %s", "lda",
                                          t->strval);
                                    emitCode(codestr);
                                    genCode(t->lchild);
                                    emitCode("stn");
                                    break;
                              default:
                                    emitCode("Error");
                                    break;
                        }
                        break;
                  case ConstKind:
                        sprintf(codestr, "%s %s", "ldc", t->strval);
```

```
                        emitCode(codestr);
                        break;
                case IdKind:
                        sprintf(codestr, "%s %s", "lod", t->strval);
                        emitCode(codestr);
                        break;
                default:
                        emitCode("Error");
                        break;
        }
    }
}
```

Expresión de ejemplo: (x = x + 3) + 4

Su AST:



Código P que debe generar (seguirlo en la función genCode():

```
lda x
lod x
ldc 3
adi
stn
ldc 4
adi
```

Su correspondiente código de tres direcciones también se podría generar con un procedimiento similar:

```
t1 = x + 3
x = t1
t2 = t1 + 4
```

Ahora una gramática con arreglos (L 422)

exp → *subs = exp | aexp*
aexp → *aexp* ***+*** *factor | factor*
factor → ***(*** *exp* ***)*** | ***num*** | ***id*** | *subs*
subs → ***id*** | ***id [*** *exp* ***]***

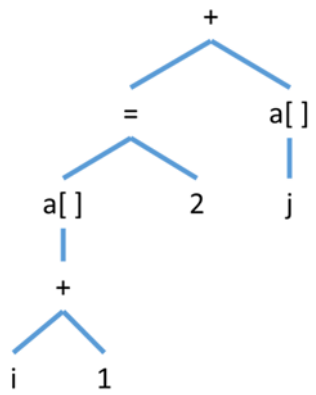
Código genCode() (L 423)

```
// Es para código P pero da la idea de la estructura (L 423)
// emitCode es un procedimiento para generar una línea simple de código P
// strval es la propiedad que contiene el tokenstring
void genCode(SyntaxTree t, int isAddr) {
      string codestr; // contiene una línea de código
      if (t != NULL) {
            switch (t->kind) {
                  case OpKind:
                        switch (t->op) {
                              case Plus:
                                    if (isAddr) emitCOde("Error");
                                    else {
                                          genCode(t->leftChild, FALSE);
                                          genCode(t->rightChild, FALSE);
                                          emitCode("adi");
                                    }
                                    break;
                              case Assign:
                                    genCode(t->leftChild, TRUE);
                                    genCode(t->rightChild, FALSE);
                                    emitCode("stn");
                                    break;
                              case Subs:
                                    sprintf(codestr, "%s %s", "lda",
                                          t->strval);
                                    emitCode(codestr);
                                    genCode(t->lchild, FALSE);
                                    sprintf(codestr, "%s%s%s",
                                       "ixa elem_size(", t->strval, ")");
                                    emitCode(codestr);
                                    if (!isAddr) emitCode("ind 0");
                                    break;
                              default:
                                    emitCode("Error");
                                    break;
                        }
                        break;
```

```
                case ConstKind:
                        if (isAddr) emitCode("Error");
                        else {
                                sprintf(codestr, "%s %s", "ldc",
                                        t->strval);
                                emitCode(codestr);
                        }
                        break;
                case IdKind:
                        if (isAddr)
                                sprintf(codestr, "%s %s", "lda",
                                        t->strval);
                        else
                                sprintf(codestr, "%s %s", "lod",
                                        t->strval);
                        emitCode(codestr);
                        break;
                default:
                        emitCode("Error");
                        break;
        }
    }
}
```

Expresión de ejemplo: `(a[i+1]=2)+a[j]`

Su AST:



Código P que debe generar (seguirlo en la función genCode():

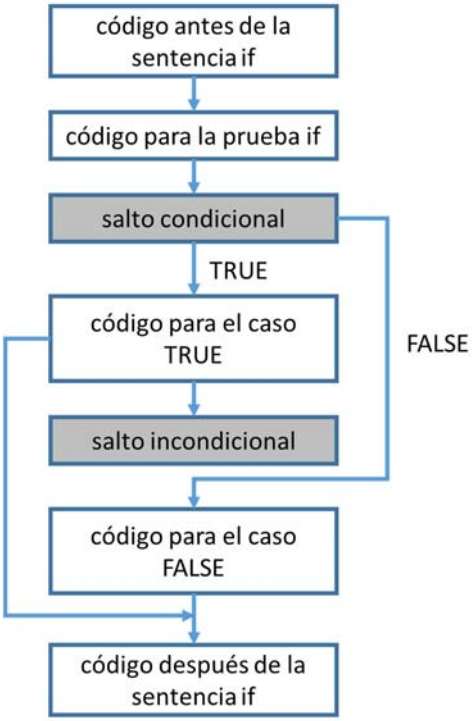
```
lda a
lod i
ldc 1
adi
ixa elem_size(a)
```

```
ldc 2
stn
lda a
lod j
ixa elem_size(a)
ind 0
adi
```

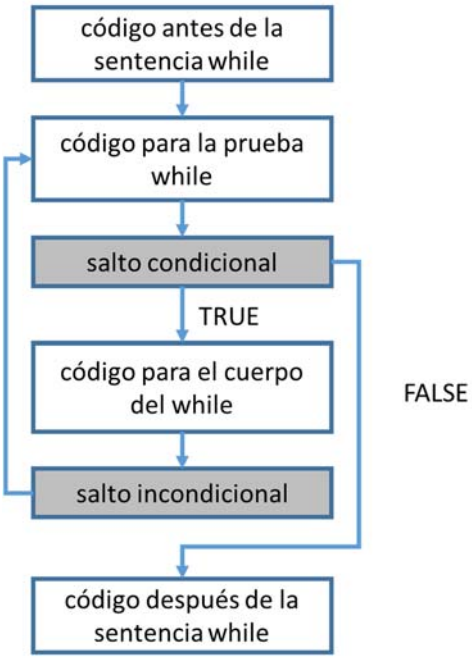
Ahora una gramática con sentencias de control if y while (L 428)

sent-if → **`if (`** *exp* **`)`** *sent* | **`if (`** *exp* **`)`** *sent* **`else`** *sent*
sent-while → **`while (`** *exp* **`)`** *sent*

Arreglo de código típico para una sentencia if (L figura 8.10 par 429)



Arreglo de código típico para una sentencia while (L figura 8.11 par 430)



Expresión if de ejemplo: `if ( E ) S1 else S2`

Su patrón código P:

```
<code to evaluate E>
fjp L1
<code for S1>
ujp L2
lab L1
<code for S2>
lab L2
```

Expresión while de ejemplo: `while ( E ) S`

Su patrón código P:

```
lab L1
<code to evaluate E>
fjp L2
<code for S>
ujp L1
lab L2
```

Gramática completa (L 433)

sent → *sent-if* | *sent-while* | **break** | **other**
sent-if → **if (** *exp* **)** *sent* | **if (** *exp* **)** *sent* **else** *sent*
sent-while → **while (** *exp* **)** *sent*
exp → **true** | **false**

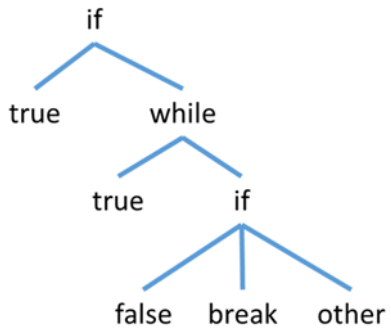
Código genCode() (L 435)

```
// Es para código P pero da la idea de la estructura (L 435)
// emitCode es un procedimiento para generar una línea simple de código P
// strval es la propiedad que contiene el tokenstring

void genCode(SyntaxTree t, string label) {
      string codestr; // contiene una línea de código
      string lab1, lab2;
      if (t != NULL)
            switch (t->kind) {
                  case ExpKind:
                        if (t->val == 0) emitCOde("ldc false");
                        else emitCode("ldc true");
                        break;
                  case IfKind:
                        genCode(t->child[0], label);
```

```
                        lab1 = genLabel();
                        sprintf(codestr. "%s %s", "fjp", lab1);
                        emitCode(codestr);
                        genCode(t->child[1], label);
                        if (t->child[2] != NULL) {
                              lab2 = genLabel();
                              sprintf(codestr, "%s %s", "ujp", lab2);
                              emitCode(codestr);
                        }
                        sprintf(codestr, "%s%s%s", "lab", lab1);
                        emitCode(codestr);
                        if (t->child[2] != NULL){
                              genCode(t->child[2], label);
                              sprintf(codestr, "%s %s", "lab", lab2);
                              emitCode(codestr);
                        }
                        break;
                  case WhileKind:
                        lab1 = genLabel();
                        sprintf(codestr, "%s %s", "lab", lab1);
                        emitCode(codestr);
                        genCode(t->child[0], label);
                        lab2 = genLabel();
                        sprintf(codestr, "%s %s", "fjp", lab2);
                        emitCode(codestr);
                        genCode(t->child[1], lab2);
                        sprintf(codestr, "%s %s", "ujp", lab1);
                        emitCode(codestr);
                        sprintf(codestr, "%s %s", "lab", lab2);
                        emitCode(codestr);
                        break;
                  case BreakKind:
                        sprintf(codestr, "%s %s", "ujp", label);
                        emitCode(codestr);
                        break;
                  case OtherKind:
                        emitCode("Other");
                        break;
                  default:
                        emitCode("Error");
                        break;
      }
}
```

Expresión de ejemplo: if (true) while (true) if (false) break else other

Su AST (L 434):



Código P que debe generar (seguirlo en la función genCode(): (L 436)

```
ldc true
fjp L1
lab L2
ldc true
fjp L3
ldc false
fjp L4
ujp L3
ujp L5
lab L4
other
lab L5
ujp L2
lab L3
lab L1
```

Ahora una gramática con funciones (definición y llamada): (L 440)

program → *decl-list exp*
decl-list → *decl-list decl* | ε
decl → **fn id (** *param-list* **) =** *exp*
param-list → *param-list* **,** **id** | **id**
exp → *exp* **+** *exp* | *llamada* | **num** | **id**
llamada → **id (** *arg-list* **)**
arg-list → *arg-list* **,** *exp* | *exp*

Código genCode(): (L 441)

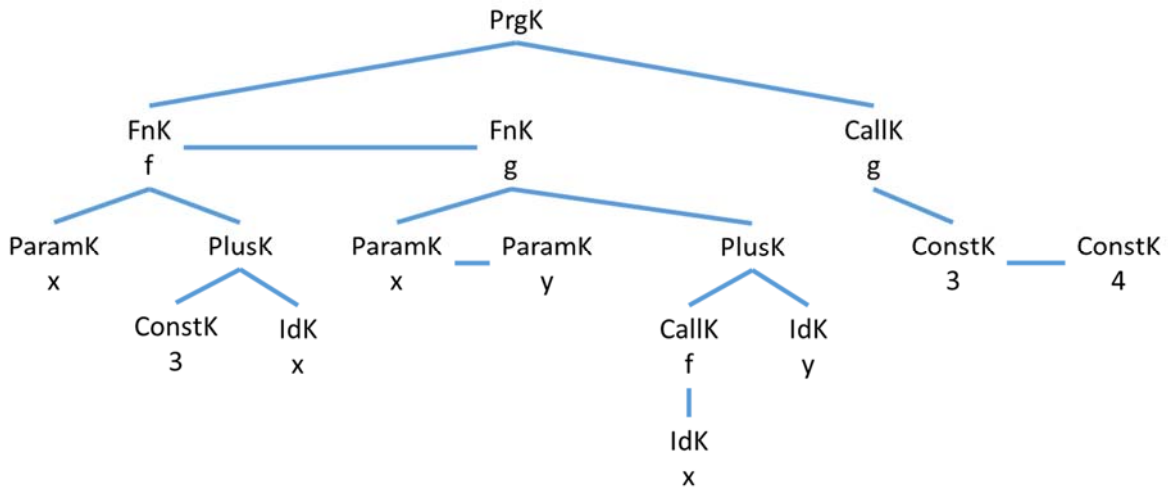
```
// Es para código P pero da la idea de la estructura (L 435)
// emitCode es un procedimiento para generar una línea simple de código P
// strval es la propiedad que contiene el tokenstring

void genCode(SyntaxTree t) {
      string codestr; // contiene una línea de código
      SyntaxTree p;
      if (t != NULL)
            switch (t->kind) {
                  case PrgK:
                        p = t->lchild;
                        while (p != NULL){
                              genCode(p);
                              p = p->sibling;
                        }
                        break;
                  case FnK:
                        sprintf(codestr. "%s %s", "ent", t->name);
                        emitCode(codestr);
                        genCode(t->rchild);
                        emitCode("ret");
                        break;
                  case ParamK:
                        break;
                  case ConstK:
                        sprintf(codestr, "%s %d", "ldc", t->val);
                        emitCode(codestr);
                        break;
                  case PlusK:
                        genCode(t->lchild);
                        genCode(t->rchild);
                        emitCode("adi");
                        break;
                  case IdK:
```

```
                        sprintf(codestr, "%s %d", "lod", t->name);
                        emitCode(codestr);
                        break;
                  case CallK:
                        emitCode("mst");
                        p = t->rchild;
                        while (p != NULL){
                              genCode(p);
                              p = p->sibling;
                        }
                        sprintf(codestr, "%s %d", "cup", t->name);
                        emitCode(codestr);
                        break;
                  default:
                        emitCode("Error");
                        break;
      }
}
```

Expresión de ejemplo: (L 440)

```
fn f(x)=2+x
fn g(x, y)=f(x)+y
g(3,4)
```

Su AST: (L 441)



Código P que debe generar (seguirlo en la función genCode(): (L 442)

```
ent f
ldc 2
lod x
```

```
adi
ret
ent g
mst
lod x
cup f
los y
adi
ret
mst
ldc 3
ldc 4
cup g
```