Mise en Oeuvre

Objectifs

But : écriture d'un simple compilateur d'un langage de type Pascal

Pourquoi écrire un compilateur ?
la majorité des informaticiens n'auront jamais à écrire de compilateur.
Cependant, mener à bien l'écriture d'un « gros » programme, le tester, documenter... a déjà un intérêt en soi.
De plus, les algorithmes employés dans un compilateur sont repris dans la résolution d'autres problèmes : manipulation de listes de recherche (gestion de la table des symboles), vérification de la correction d'enregistrements (analyse syntaxique), traitement de caractères, de mots (grammaire BNF).
Enfin, il semble intéressant pour un programmeur de connaître les dessous des choses ; comprendre comment les programmes se compilent et s'exécutent.

Exemple

Prenons en exemple le texte source suivant :
while (1<P) and (P<3) do P:=P+Q

L'interface d'entrée va fournir une suite de caractères significatifs (l'espace blanc est noté par le caractère _) :
w h i l e _ (1 < P) _ a n d _ (P < 3) _ d o _ P : = P + Q _

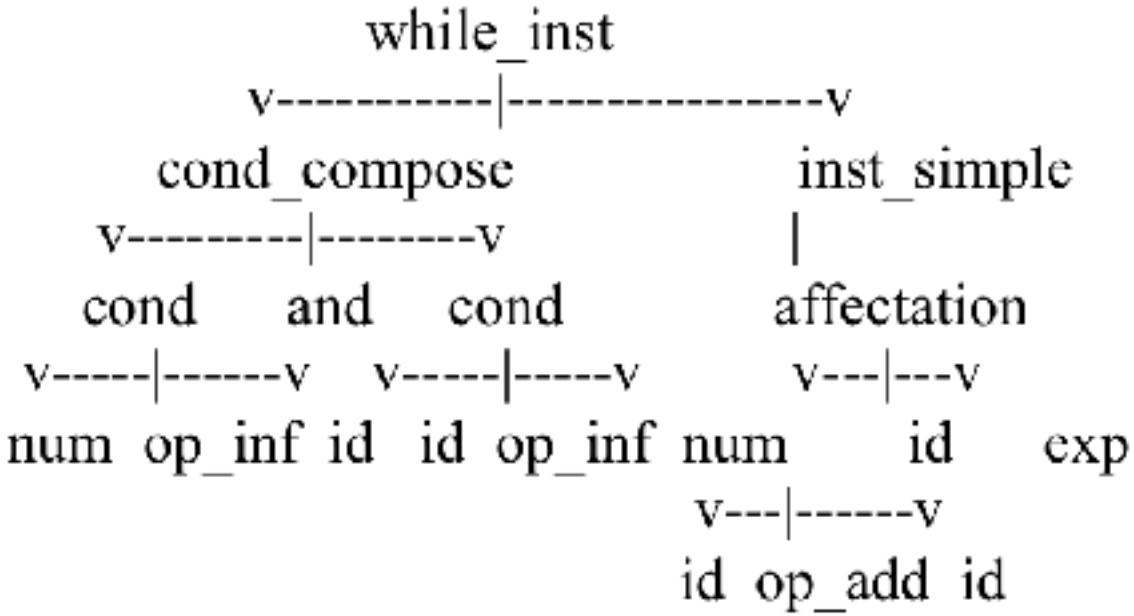
L'analyseur lexical va va reconnaître les mots (tokens) suivants

Exemple

mot clef	while
symbole	parenthèse ouvrante
constante numérique	1
opérateur relationnel	<
identificateur	P
symbole	parenthèse fermante
mot clef	and
symbole	parenthèse ouvrante
identificateur	P
opérateur relationnel	<
constante numérique	3
symbole	parenthèse fermante
mot clef	do
identificateur	P
symbole	affectation
identificateur	P
opérateur	+
identificateur	Q

Exemple

L'analyseur syntaxique va construire l'arbre suivant :



Exemple

Le code intermédiaire généré sera de la forme :

```
L0      if 1<P goto L1
        goto L3
L1      if P<3 goto L2
        goto L3
L2      P:=P+Q
        goto L0
L3      ...
```

Exemple

Il pourra être optimisé en :

```
L0      if 1<P goto L1
        goto L3
        if P>=3 goto L1
        P:=P+Q
        goto L0
L1      ...
```

Langage machine à générer : P-code

La machine n'a qu'une zone mémoire gérée en pile.
Elle n'a pas de registres sauf :

- le pointeur de sommet de pile (SP)
- le compteur d'instructions (PC)

La plupart des instructions du P-Code prennent donc leurs opérandes sur la pile et laissent leur résultat sur cette pile.

La structuration de la mémoire en pile facilite la compilation de langages de haut niveau autorisant la récursion et la compilation d'expressions arithmétiques complexes.

Langage machine à générer : P-code

Le bas de la pile est réservé pour l'allocation des variables globales

La première instruction d'un programme P-Code réserve une partie de la pile correspondant à la zone d'allocation des variables utilisées dans le programme ; cette réservation est faite par incrémentation du pointeur de pile de la taille mémoire nécessaire.

Les éléments de la pile sont de simples entiers ; les adresses sont des adresses absolues dans la pile.

Le jeu d'instruction du P-Code simplifié que nous considérerons est donné dans la suite

Langage machine à générer : P-code

Le jeu d'instruction du P-Code :

ADD	additionne le sous-sommet de pile et le sommet, laisse le résultat au sommet (idem pour SUB, MUL, DIV)
EQL	laisse 1 au sommet de pile si sous-sommet = sommet, 0 sinon (idem pour NEQ, GTR, LSS, GEQ, LEQ)
PRN	imprime le sommet, dépile
INN	lit un entier, le stocke à l'adresse trouvée au sommet de pile, dépile
INT c	incrémente de la constante c le pointeur de pile (la constante c peut être négative)
LDI v	empile la valeur v
LDA a	empile l'adresse a
LDV	remplace le sommet par la valeur trouvée à l'adresse indiquée par le sommet (déréférence)
STO	stocke la valeur au sommet à l'adresse indiquée par le sous-sommet, dépile 2 fois
BRN i	branchement inconditionnel à l'instruction i
BZE i	branchement à l'instruction i si le sommet = 0, dépile
HLT	halte

Langage machine à générer : P-code

L'exemple suivant :

```
begin
   repeat
      read (A) ;
      B := A + B ;
   until A = 0 ;
   write (B) ;
end
```

produira le code suivant

Langage machine à générer : P-code

0	INT 2	réserve 2 emplacements pour A (0) et B (1)
1	LDA 0	empile l'adresse de A
2	INN	lit une valeur la range dans A
3	LDA 1	empile l'adresse de B
4	LDA 0	empile l'adresse de A
5	LDV	déréférence. Valeur de A au sommet
6	LDA 1	empile l'adresse de B
7	LDV	déréférence. Valeur de B au sommet
8	ADD	addition de A et B
9	STO	stocke le résultat dans B
10	LDA 0	empile l'adresse de A
11	LDV	déréférence. Valeur de A au sommet
12	LDI 0	empile la valeur 0
13	EQL	test d'égalité
14	BZE 1	branchement en 1 si A <> 0
15	LDA 1	empile l'adresse de B
16	LDV	déréférence. Valeur de B au sommet
17	PRN	écrit le résultat
18	HLT	fin

Interpréteur du P-code

L'interpréteur consiste en un logiciel permettant d'exécuter les instructions générées par le compilateur que vous devez écrire.
Ces instructions sont écrites à l'aide du jeu d'instructions étudié en cours.

Ces instructions sont stockées dans un tableau que nous appellerons P-CODE.

L'exécution est réalisée à l'aide d'une pile appelé MEM ; le pointeur de pile s'appellera SP.

Interpréteur du P-code

La structure de la mémoire en pile facilite la compilation de langages de haut niveau autorisant la récursivité. Les éléments de la pile sont des entiers : les adresses sont des adresses absolues dans la pile.

Le travail se décompose en 3 parties :
1- Réfléchir aux structures de données permettant de représenter la pile de la machine et le pointeur de pile.
2-Réflechir à la structure de données permettant de stocker instructions
3-Ecrire le programme qui permet d'exécuter les instructions contenues dans la structure de données(tableau)représentant le P-CODE à l'aide de la pile mémoire. On utilisera une variable PC qui représente le pointeur d'instruction (adresse de l'instruction du tableau P-CODE devant être traitée).
On utilisera aussi la variable INST pour désigner l'instruction en cours de traitement.

Interpréteur du P-code

Les structures de données nécessaires lors de l'écriture d'un interprète simplifié pour le P-Code sont
1- un tableau MEM représentant la pile de la machine et un pointeur de pile associé (SP, stack pointer) :

```
var   MEM : array [0 .. TAILLEMEM] of integer ;
      SP : integer ;
```

2- un tableau PCODE stockant les pseudo-instructions, auquel sont associés l'instruction courante INST et un pointeur d'instruction PC ; une instruction est un couple (mnémonique, opérande éventuelle)

Interpréteur du P-code

```
type   MNEMONIQUES = (ADD,SUB,MUL,DIV,EQL,NEQ,GTR,LSS,GEQ,LEQ,
         PRN,INN,INT,LDI,LDA,LDV,STO,BRN,BZE,HLT) ;
       INSTRUCTION = record
         MNE : MNEMONIQUES ;
         SUITE : integer
       end
var    PCODE : array [0 .. TAILLECODE] of INSTRUCTION ;
       PC : integer ;
       INST : INSTRUCTION ;
```

L'algorithme se divise en deux parties : le chargeur >> et l'interpréteur
Le chargeur remplit la table PCODE à partir d'un fichier de code P-Code. L'interprète est basé sur le modèle suivant

Interpréteur du P-code

```
begin
  (* initialise pointeur d'instruction PC et pointeur de pile SP *)
  (* initialise program status PS a EXECUTION *)
  repeat
    (* chargement *)
    INST = PCODE [PC]
    incremente PC
    (* execution *)
    with INST do
      case MNE of
        ADD : begin SP:=SP-1; MEM[SP]:=MEM[SP]+MEM[SP+1] end ;
        ...
        HLT : PS:=FINI ;
      end
  until PS <> EXECUTION
end.
```

Syntaxte du langage à implémenter

PROGRAM ::= program ID ; BLOCK .
BLOCK ::= CONSTS VARS INSTS
CONSTS ::= const ID = NUM ; { ID = NUM ; } | e
VARS ::= var ID { , ID } ; | e
INSTS ::= begin INST { ; INST } end
INST ::= INSTS | AFFEC | SI | TANTQUE | ECRIRE | LIRE | e
AFFEC ::= ID := EXPR
SI ::= if COND then INST
TANTQUE ::= while COND do INST
ECRIRE ::= write (EXPR { , EXPR })
LIRE ::= read (ID { , ID })
COND ::= EXPR RELOP EXPR
RELOP ::= = | <> | < | > | <= | >=
EXPR ::= TERM { ADDOP TERM }
ADDOP ::= + | -
TERM ::= FACT { MULOP FACT }
MULOP ::= * | /
FACT ::= ID | NUM | (EXPR)