LI5 : Rapport Utilisateur du compilateur pascal

Nicolas LASSALLE Maxime GRYL

15/05/2004

Table des matières

1 Présentation

Ce Programme est le compilateur d'un programe "à la Pascal". Il permet donc d'exécuter un programme écrit dans un sous-essemble du langage Pascal. Pour cela 2 opérations sont nécessaires :

- La Compilation du fichier Pascal dans un langage cible propre à notre logiciel (le Pcode).
- L'interprétation du code cible.

Première partie

Compilateur

2 Fonctionnement Global

2.1 Traducteur (Compilateur)

Le traducteur va analyser le programme Pascal pour créer un fichier Pcode. Il va aussi détecter les erreurs lexicales et de syntaxes éventuelles.

2.2 Utilisation

Le compilateur va prendre comme entrée un fichier Pascal. La sortie sera soit un fichier Pcode si la compilation s'est déroulée sans erreur, soit l'affichage des erreurs rencontrées.

Commande d'exécution Usage : ../pp5 [options] <nom_fichier_pp5>
Les options possibles sont :
-tablesym : affiche la table des symboles

3 Erreurs de sortie

Codes de retour valeur 0 : Compilation terminée avec succès.

1. *Valeur non atribuée.*
2. *Valeur non atribuée.*
3. *Valeur non atribuée.*
4. *Valeur non atribuée.*
5. Erreur dans les paramètres.
6. Erreur à la lecture du fichier.
7. Erreur à l'écriture du fichier.
8. Erreur à l'ouverture du fichier en entrée.
9. Erreur à l'ouverture du fichier en sortie.
10. Erreur à la fermeture du fichier.
11. Erreur à la suppression du fichier.
12. Stack Overflow pile sauts.
13. Erreur lors de la mise à jour d'un saut.
14. identificateur attendu.
15. PROGRAM attendu.
16.) parenthèse fermante attendue.
17. END attendu.
18. ; attendu.
19. = attendu.
20. begin attendu.
21. erreur dans la partie déclaration.
22. , attendue.
23. erreur dans une constante.
24. := attendu.
25. then attendu.
26. do attendu.
27. erreur dans un facteur (expression erronée).
28. identificateur déclaré deux fois.
29. identificateur non déclaré.
30. nombre attendu.
31. affectation non permise.
32. constante entière dépassant les limites.
33. division par zéro.
34. . attendu.
35. (parenthèse ouvrante attendue.
36. write attendu.
37. writeln attendu.

38. read attendu.
39. for attendu.
40. to attendu.
41. downto attendu.
42. repeat attendu.
43. until attendu.
44. erreur inconu.
45. while attendu.
46. if attendu.
47. : attendu.
48. case attendu.
49. of attendu.
50. erreur dans la déclaration d'un record.
51. erreur, intervalle non croissant.
52. variable de type record attendue.
53. variable de type champ attendue.
54. seuls les opérateurs '=' ou '<>' autorisés pour les booléens.
55. il faut comparer un boolean avec un boolean.

4 Langage pascal reconnu

Pour toute la description du langage, le mot BLOCK désignera ceci une instruction simple ou bien une suite d'instruction encadrées par les mots clés begin et end.

4.1 Programme pascal de base

```
program identificateur_de_programme;
{Déclaration de constantes}
{Déclaration de types}
{Déclaration de variables}

{Déclaration de procédures}
{Déclaration de fonctions}

BLOCK.
```

4.2 Déclaration de constantes

Voici différentes façon de déclarer des constantes :

```
const A = 15;
      B = -5;
      C = 1 , D = 4;
const E = -(((4+5)*8)/(4*(8+5))-8)/2;
```

On notera bien que toutes les constantes doivent être initialisées, par une valeur simple, négative, ou une expression.

4.3 Déclaration de types

Le compilateur possède les types prédéfinis suivant :
- integer
- char
- boolean

Il permet également de définir des types complexes (nommés ou anonymes).

4.3.1 Intervalles

```
type inter = 0..5;
     inter2 = B..A;
```

4.3.2 Tableaux

A une dimension :

```
pile = array [0..10] of type;
pile2 = array [inter] of type;
pile2 = array [5] of type;
```

A plusieurs dimensions :

```
pile3 = array [0..10,5,inter,A] of type;
pile4 = array [inter] of type;
```

4.3.3 Records

```
complex     = record
                 re,im : integer;
              end;
complex2     = record
                 re : integer;
                 im : integer;
              end;
```

4.3.4 Types anonymes

Notre langage pascal permet la déclaration de type anonymes lors de la déclaration de n'importe quelle variable, ou bien même à l'intérieur d'une déclaration de type.

```
complexpile = record
                 p : array [0..2] of record
                        re,im : integer ;
                     end;
                 sommet  : integer;
              end;
```

4.4 Déclaration de variables

Les variables peuvent être de n'importe quel type. On peut également leurs déclarer un type anonyme.

```
cp      : complex;
i,i2    : integer;
b,b2    : boolean;
c,c2    : char;
mapile  : complexpile;
test    : array [10] of integer;
test2    : array [10,45,12] of integer;
```

4.5 Déclaration de procédures

Le nombre de paramètre peut être zéro.
ATTENTION : les procédures sont syntaxiquement / sémantiquement reconnues mais la génération de code n'a pas encore été implémentée.

```
procedure test (param1 :integer, param2 : complexe);
{Déclaration de constantes}
{Déclaration de variables}
BLOCK
```

4.6 Déclaration de fonctions

Le nombre de paramètre peut être zéro. Le type de retour d'une fonction doit toujours être un type prédéfini.
ATTENTION : les fonctions sont syntaxiquement / sémantiquement reconnues mais la génération de code n'a pas encore été implémentée.

```
function test (param1 :integer) : type_simple_de_retour;
{Déclaration de constantes}
{Déclaration de variables}
BLOCK
```

4.7 Accès aux variables

Les accès aux tableaux et records peuvent être mixés.

4.7.1 Tableaux

```
test[4]
test2[4,5,6]
```

4.7.2 Records

```
cp.re
cp.im
```

4.8 Affectations

```
i := 4;
i := i2;
i := EXPRESSION;
b := true;
b := false;
b := b2;
c := 't';
c := c2;
```

4.9 Structures de controle

4.9.1 IF / ELSE

Malheureusement le else n'est pas implémenté. Le if n'accepte que des conditions simples (pas de AND / OR / NOT).

```
if (expression OP expression)
if (var_booléene OP2 false)
if (var_booléene OP2 true)
if (false OP2 var_booléene)
if (true OP2 var_booléene)
if (var_booléene OP2 var_booléene)
if ('c' OP2 var_char)
if (var_char OP2 'c')
```

OP est égal à '=' ou '<>' '<=' ou '<' ou '>=' ou '>' .
OP2 est égal à '=' '<>'

4.9.2 Switch

```
switch (var_entière)
```

4.10 Structures itératives

4.10.1 for

```
for i:=4 to 10 do
BLOCK

for i:=EXPRESSION to 10 step 2 do
BLOCK

for i:=10 downto 0 do
BLOCK

for i:=10 downto 0 step 2 do
BLOCK
```

4.10.2 while

La syntaxe de la condition utilisée dans le while est la même que celle utilisée dans le if.

```
while ( COND ) do
BLOCK
```

4.10.3 repeat until

```
repeat
BLOCK
until (cond);
```

4.11 Entrées / Sorties

Les entrées / sorties ne peut utiliser que des variables qui sont de types primitifs (char, intger, boolean).

4.11.1 write

```
write (var_integer);
write (var_char);
write (var_boolean);
write ((4+45)-78); { une expression donc }
write ('une chaine de caractère');
write ('c',i,c,b,c2);
```

4.11.2 writeln

La syntaxe de writeln est la même que celle de write, sauf qu'elle affiche un retour à la ligne.

4.11.3 read

Le read ne peut lire que des variables qui sont de types primitifs (char, intger, boolean).

```
read (var_integer);
read (var_char);
read (var_boolean);
read (i,b,c,c2);
```

Deuxième partie

Interpréteur

5 Fonctionnement Global

5.1 Utilisation

L'interpréteur va prendre comme entrée un fichier Pcode. Sa sortie sera l'exécution de ce fichier (donc du programme Pascal) et bien sur l'affichage des erreurs si il y en a.

Commande d'exécution Usage : ./ipc [-d] <nom_fichier_pcode>

6 Erreurs de sortie

Les erreurs sont toutes acconpagnées d'un message.

Codes de retour valeur 0 : Compilation terminée avec succès.

1. Fichier de pcode non correct. Arret du programme.
2. *Valeur non atribuée.*
3. Erreur survenue dans le module de la pile des variables
4. Stack Overflow. La pile instruction n'a pu etre agrandie.
5. Stack Overflow. Dépassement de la pile d'instruction.
6. Stack Overflow. La pile d'exécution n'a pu etre agrandie.
7. Impossible d'incrémenter la zone mémoire de la pile.
8. Stack Underflow. Dépassement de pile.
9. Stack Overflow. Dépassement de pile.
10. Stack Overflow. Adresse invalide.
11. Stack Underflow. Pile vide.
12. Stack Underflow. Accés hors pile.
13. Interpréteur : division par zéro.
14. Missing parameter for %s. *
15. Option '%s' non reconnue. *
16. Erreur à la lecture du fichier.
17. *Valeur non atribuée.*
18. Erreur à l'ouverture du fichier en entrée.
19. *Valeur non atribuée.*
20. Erreur à la fermeture du fichier.
21. *Valeur non atribuée.*
22. *Valeur non atribuée.*
23. End of instruction waited.
24. Caratère non reconnu en fin d'instruction.
25. Pas d'espace avant le paramètre.
26. Missing parameter.
27. missing open ".
28. missing close ".
29. missing open '.
30. missing close '.
31. Mnémonique non reconnu.

* %s correspond à un parametre passé en ligne de commande.

7 Langage pcode reconnu

7.1 Opérateurs

- **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).

7.2 Ecriture

- **PRN** : Imprime la valeur entière au sommet, dépile.
- **PRD** : Imprime le caractère qui est au sommet de pile, dépile.
- **PRB** : Imprime 'vrai' si le sommet et != 0, 'faux' sinon, dépile.
- **PRI** : Imprime le sommet, ne dépile pas (utile pour débuguage manuel).
- **PRC 'c'** : Imprime le caractère passé en paramètre de l'instruction.
- **PRS "chaine"** : Imprime la chaîne de caractères passée en paramètre de l'instruction.
- **PRL** : Imprime le caractère "retour à la ligne".

Remarque : L'instruction *PRS* n'est pas implémentée en tant que fonction à part entière : elle est traduite à la lecture du fichier Pcode par une suite d'instruction **PRC 'c'** pour chaque caractère de la chaîne.

7.3 Lecture

- **INN** : Lit un entier, le stocke à l'adresse trouvée au sommet de pile, dépile.
- **INC** : Lit un caractère, le stocke à l'adresse trouvée au sommet de pile, dépile.
- **INB** : Lit un booléen, le stocke à l'adresse trouvée au sommet de pile, dépile. Booléen : **'1'** pour VRAI, **'0'** pour FAUX.

Remarque : Les instructions de lectures sont munies de tests ne permettant pas l'enregistrement de valeurs saisies ne correspondant pas au type souhaité.
Le dépassement de capacité n'est pas géré. (exemple : l'entier 1111111111111 saisi ne correspondra pas à la valeur enregistrée).

7.4 Insts de base

- **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.

7.5 Sauts

- **BRN i** : Branchement inconditionnel à l'instruction i. Adressage absolu.
- **BRR** : Branchement inconditionnel à l'instruction i. Adressage relatif.

– **BZE i** : Branchement à l'instruction i si le sommet = 0, dépile. Adressage absolu.
– **BZR i** : Branchement à l'instruction i si le sommet = 0, dépile. Adressage relatif.

Attention : La gestion des sauts imcomplets dans le pcode .

7.6 Procédures / fonctions

– **CAL i** : Empile le compteur d'instructions et réalise le branchement à l'instruction i.
– **RET i** : Néttoie la pile des emplacements pour les paramètres des procédures.

7.7 Autres

– **CPA / CPI** : Copie la valeur de l'adresse/entier en sommet de pile sur le sur-sommet.
– **CPJ** : Copie la valeur de l'entier en sommet de pile sur le sur-sommet.
– **DEL** : Supprime l'entier du dessus de la pile (sert rarement (cf cas())).
– **HLT** : Termine le programme