Maîtrise d'informatique
Module de Projet 1

Examen – Compilateur Pascal

Philippe MARQUET

Janvier 2004

Documents autorisés — Durée : 2 heures

Soit PPx une extension de PP8 (chapitre 12, Procédures paramétrées) introduisant la nouvelle structure de contrôle suivante :

```
INST     ::=  INSTS | AFFEC | SI | TANTQUE | ECRIRE | LIRE | TRY
TRY      ::=  try DO_UNDO ; { DO_UNDO ; } INST
DO_UNDO  ::=  do INST on COND undo INST
```

L'instruction `try` est une séquence d'*instructions contrôlées* (DO_UNDO).

Chaque instruction contrôlée est composée

- d'une première instruction INST
- associée à une condition d'erreur COND
- et à une *instruction de retour* (le second INST).

L'exécution d'une construction `try` est réalisée ainsi :

- chacune des instructions contrôlées est exécutée, dans l'ordre ;
- l'exécution d'une instruction contrôlée est suivie de l'évaluation de la condition d'erreur ;
- si cette condition est évaluée à vrai, la construction `try` se termine prématurément ; sinon, son exécution se poursuit avec l'instruction contrôlée suivante ;
- si toutes les instructions contrôlées sont des succès, l'instruction finale est exécutée ;
- enfin, chacune des instructions de retour associées à une instruction contrôlée qui a été exécutée est exécutée, dans l'ordre inverse ;
- en particulier si l'instruction finale est exécutée, toutes les instructions de retour seront exécutées.

Cette instruction est illustrée par l'exemple de la figure 1.

Exercice 1 (Analyse syntaxique)
Donner une procédure d'analyse syntaxique pour l'instruction `try`. ■

```
try
    do new(A); on A = nil undo dispose(A);
    do new(B); on B = nil undo dispose(B);
    do new(C); on C = nil undo dispose(C);

    begin
        do_something(A,B,C)
    end
```

FIG. 1 – Exemple d'utilisation de la structure `try`
Les procédures `new()` et `dispose()` sont les allocateurs et désallocateurs de mémoire dynamique. Le mot réservé `nil` est une valeur particulière de pointeur marquant la non validité.

Id: xam04.tex,v 1.3 2004/01/22 08:05:33 marquet Exp

Tests imbriqués	Sorties chaînées

```
if (malloc(first)) {
    if (malloc(second)) {
        if (malloc(third)) {
            do_something();
            free(third);
        }
        free(second);
    }
    free(first);
}
```

```
    if (!malloc(first))
        goto out_first;
    if (!malloc(second))
        goto out_second;
    if (!malloc(third))
        goto out_third;

    do_something();

    free(third);
out_third :
    free(second)
out_second :
    free(first)
out_first :
```

FIG. 2 – Deux exemples de codes alternatifs à une construction `try` : tests imbriqués et sorties chaînées. Il ne s'agit pas de codes PP ou Pascal, mais de structures de codes possibles dans un pseudo-langage impératif classique.

Exercice 2 (Alternatives)
Discuter brièvement de l'intérêt de la construction `try` sur des codes tels ceux de la figure 2. ■

Exercice 3 (Schéma de génération de P-Code)
Donner le schéma de P-Code à générer pour une instruction `try`.

Exercice 4 (Traduction en P-Code)
Traduire en P-Code l'extrait de programme de la figure 1. On suppose que les paramètres de la fonction `do_something()` et des « fonctions » `new` et `dispose` sont passés par adresse, que la nouvelle instruction P-Code `NIL` laisse sur la pile la valeur `nil`, et que les opérateurs P-Code de comparaison sont valides sur les valeurs des trois variables `A`, `B`, et `C`. ■

Exercice 5 (Génération de code)
Étendre la procédure d'analyse syntaxique de l'exercice 1 pour y inclure les instructions de génération de P-Code. ■