Compilateur de langage de type Pascal vers langage Assembleur

Ce projet est un compilateur qui traduit un langage structuré simplifié de type Pascal en langage Assembleur 64 bits 80x86. Il est basé sur le travail réalisé par Pierre Jourlin en 2019 et complété par BOUDOUNT Youssef.

Sommaire

    - CaseLabelList
    - CaseListElement
    - CaseStatement
    - RepeatStatement
- Test des nouvelles fonctionnalités ajoutées
    - ForStatement
    - CaseLabelList
    - CaseListElement
    - CaseStatement
    - RepeatStatement
- Compilation et test
- Auteur

Licence

Ce programme est un logiciel libre : vous pouvez le redistribuer ou le modifier selon les termes de la licence publique générale GNU telle que publiée par la Free Software Foundation, version 3 ou ultérieure.

Fonctionnalités ajoutées

ForStatement

La fonction ForStatement gère les boucles "for" dans le langage de type Pascal. Elle prend en charge les syntaxes suivantes :

```
FOR <ID> := <Expression1> TO <Expression2> DO
    <Statement>
```

Ou :

```
FOR <ID> := <Expression1> DOWNTO <Expression2> DO
    <Statement>
```

La fonction ForStatement génère le code assembleur correspondant à la boucle "for" en utilisant les registres appropriés et les instructions de comparaison et de saut conditionnel.

CaseLabelList

La fonction CaseLabelList gère la liste des étiquettes d'une instruction "CASE" dans le langage de type Pascal. Elle prend en charge la syntaxe suivante :

```
<Constant1>, <Constant2>, ..., <ConstantN>
```

La fonction CaseLabelList empile les constantes correspondantes et compare leur valeur avec la valeur de l'expression. Elle génère les instructions de comparaison et de saut conditionnel nécessaires pour chaque étiquette.

CaseListElement

La fonction CaseListElement gère un élément de la liste d'instructions "CASE" dans le langage de type Pascal. Elle prend en charge la syntaxe suivante :

```
<CaseLabelList1> : <Statement1>;
<CaseLabelList2> : <Statement2>;
...
<CaseLabelListN> : <StatementN>;
```

CaseStatement

La fonction CaseStatement gère l'instruction "case" complète dans le langage de type Pascal. Elle prend en charge la syntaxe suivante :

```
case <Expression> of
    <CaseListElement1>
    <CaseListElement2>
    ...
    <CaseListElementN>
end;
```

RepeatStatement

La fonction RepeatStatement gère les instructions "repeat" dans le langage de type Pascal. Elle prend en charge la syntaxe suivante :

```
REPEAT
    <statements>
UNTIL <expression>
```

La fonction RepeatStatement effectue une boucle répétitive des instructions dans `<statements>` jusqu'à ce que `<expression>` soit évaluée comme vraie (différente de zéro).

Test des nouvelles fonctionnalités ajoutées

Test du ForStatement

Voici des exemples de code que vous pouvez ajouter dans le fichier test.p pour tester le ForStatement :

- **Scénario Simple : Incrémentation par pas de 1**

Ce scénario affiche les nombres de 1 à 5 en utilisant une boucle FOR avec un pas de 1.

```
VAR
    i: INTEGER;

FOR i := 1 TO 5 DO
BEGIN
    DISPLAY i;
    DISPLAY '\n';
END.
```

- **Scénario Intermédiaire : Décrémentation par pas de 2**

Ce scénario affiche les nombres de 10 à 2 en utilisant une boucle FOR avec un pas de 2.

```
VAR
    i: INTEGER;

FOR i := 10 DOWNTO 2 STEP 2 DO
BEGIN
    DISPLAY i;
    DISPLAY '\n';
END.
```

- **Scénario Avancé : Utilisation d'une variable comme limite supérieure**

Ce scénario utilise une variable limit comme limite supérieure de la boucle FOR. La variable limit est préalablement initialisée à la valeur 8.

```
VAR
    limit: INTEGER;
    i: INTEGER;

limit := 8;

FOR i := 1 TO limit DO
BEGIN
    DISPLAY i;
    DISPLAY '\n';
END.
```

Test du CaseStatement

Voici des exemples de code que vous pouvez ajouter dans le fichier test.p pour tester le CaseStatement :

- **Scénario Basique : Utilisation d'une seule constante**

Ce scénario teste la fonction CaseStatement en utilisant une seule constante dans la liste des cas.

```
VAR
  value: INTEGER;
```

```
BEGIN
  value := 1;
  CASE value OF
    1: DISPLAY 'VALEUR 1';
  END;
END.
```

Explication : Le code initialise une variable value avec la valeur 1. La fonction CaseStatement compare la valeur de value avec la constante 1. Comme il y a une correspondance, le bloc de code correspondant est exécuté et affiche "Valeur 1".

- **Scénario Intermédiaire : Utilisation de plusieurs cas**

Ce scénario teste la fonction CaseStatement en utilisant plusieurs cas dans la liste des cas.

```
VAR
  value: CHAR;

BEGIN
  value := 'b';
  CASE value OF
    'a': DISPLAY 'Valeur A';
    'b': DISPLAY 'Valeur B';
    'c': DISPLAY 'Valeur C';
  END;
END.
```

Explication : Le code initialise une variable value avec la valeur 'b'. La fonction CaseStatement compare la valeur de value avec les différentes constantes. Dans ce cas, la constante 'b' correspond, donc le bloc correspondant est exécuté et affiche "Valeur B".

- **Scénario Avancé : Utilisation d'une constante par défaut**

Ce scénario teste la fonction CaseStatement en utilisant une constante par défaut lorsque aucun cas ne correspond.

```
VAR
  value: INTEGER;

BEGIN
  value := 5;
  CASE value OF
```

```
    1: DISPLAY 'Valeur 1';
    2: DISPLAY 'Valeur 2';
    ELSE DISPLAY 'Autre valeur';
  END;
END.
```

Explication : Le code initialise une variable value avec la valeur 5. La fonction CaseStatement compare la valeur de value avec les différentes constantes. Comme aucun des cas ne correspond, le bloc else est exécuté et affiche "Autre valeur".

Test du RepeatStatement

Voici un exemple de code que vous pouvez ajouter dans le fichier test.p pour tester le RepeatStatement :

```
VAR
    i: INTEGER;

BEGIN
    i := 1;
    REPEAT
        DISPLAY i;
        i := i + 1;
    UNTIL i > 5;
END.
```

Explication : Le code initialise une variable i à 1. Ensuite, il exécute les instructions à l'intérieur de la boucle REPEAT qui affichent la valeur de i et incrémentent i de 1 à chaque itération. La boucle continue jusqu'à ce que i dépasse 5.

Compilation et test

Pour compiler le projet, utilisez la commande :

```
make test
```

Pour vérifier la sortie :

```
gedit test.s
```

Pour déboguer l'exécutable avec DDD (Data Display Debugger), utilisez la commande :

```
ddd ./test
```

Pour installer DDD sur Ubuntu :

```
sudo apt install ddd
```

Auteur

Youssef BOUDOUNT Licence informatique, L2 Groupe 4

- youssef.boudount@alumni.univ-avignon.fr
- Github: @youssefb14
- LinkedIn: @youssefb14