Structure des classes C++ du programme

Le parcours de l'arbre

Le programme EZ-language est décomposé en plusieurs parties distinctes (déclaration de variables et constantes, début du programme, instructions… etc). Chacune de ces parties va être découpée en différents blocs qui seront matérialisés sous la forme de classes C++ que l'on va implémenter dans le compilateur.
Au fur et à mesure de son parcours du fichier .ez, le compilateur crée les objets C++ que l'on va utiliser pour traduire ce fichier en fichier .cpp.

La structure d'un arbre **binaire** sera utilisée pour retracer le parcours du compilateur. Ainsi, chaque classe C++ créée hérite d'une classe « Node » qui lui fixera une structure : les éléments relatifs à la classe créée dans son fils gauche et un accès à la suite du programme dans son fils droit.

Une fois que le compilateur a parcouru le fichier .ez, on appelle la méthode ***translate*** du ***noeud racine*** et la création du/des fichiers C++ nécessaires sont générés (la traduction de chacun des noeuds entraînant la traduction des autres).

La classe Node

Cette classe abstraite sert de base à l'architecture de nos classes. Elle permet de représenter l'arbre et retranscrire simplement tout notre programme EZ langage. Cette classe contient un nom, un fils gauche ainsi qu'un fils droit. L'arbre est conçu de telle sorte que pour un nœud donné, son fils gauche contient les éléments nécessaires au bon fonctionnement de ce dernier et son fils droit donne accès à la suite des instructions du programme.
L'intérêt principal de cette classe ***Node*** réside dans la présence d'une méthode ***translate*** qui retournera une chaîne de caractères correspondant à la traduction C++ d'un élément ***EZ-language***.

Exemple : si dans notre programme on trouve deux boucles *While* qui se suivent dans le programme, alors le fils gauche du premier *While* contiendra sa condition d'exécution et les instructions qu'il contient et le fils droit de ce *While* pointera sur le 2^{e} *While*.
Un appel à la fonction ***translate*** du ***noeud racine*** d'un programme ne contenant que ces deux boucles While entraînera ainsi la traduction du premier *While* (par l'intermédiaire de la traduction des éléments contenus dans son fils gauche), qui engendrera la traduction du fils droit ce premier *While* (le 2e *While*) et ainsi de suite jusqu'à arriver à la fin du programme.

Les classes génériques

La classe Operator

La classe dédiée aux opérateurs de tout type est appelée *Operator*.
Cette classe contient les champs suivants :

- ope_nb : valeur entière qui indique s'il s'agit d'un opérateur unaire ou binaire. On pourra ajouter par la suite des opérateurs ternaires au besoin.
- ope_type : valeur entière qui donne le type de l'opérateur (opérateur logique, arithmétique, allocation, comparaison ou incrémentation).
- ope : chaîne de caractères correspondant à la valeur de l'opérateur (+, -, ++, etc).
- operande_1 et operande_2 : chaînes de caractères correspondant aux éléments qui sont liés à cette opérateur. Dans le cas d'un opérateur unaire, seul *operande_1* sera pris en compte.

Exemple d'utilisation :

```
Operator new_ope= new Operator(2, 2, "+", "x", "y");
new_ope.translate();
```

Lors de la traduction de l'arbre, pour ce noeud new_ope, la chaîne de caractères "x+y" sera alors retournée dans le fichier C++ traduit.

La classe Range

Cette classe est composée de deux champs : les deux entiers qui correspondent au début et à la fin de ce *Range*.

Exemple d'utilisation :

```
Range my_range= new Range(1,10);
my_range.translate();
```

Lors de la traduction de l'arbre, pour ce noeud my_range, la chaîne de caractères "1..10" sera alors retournée dans le fichier C++ traduit.

La classe Repeat

Cette classe se compose d'une condition et d'une suite d'instructions. Elle correspond au "Do...while" en C++.
Elle contient un champ *Condition* (une classe que l'on a implémenté également) et héritant de *Node*, contient également un fils gauche et un fils droit.
Dans ce fils gauche, on trouvera l'ensemble des instructions contenues dans la boucle.
Le fils droit contient l'instruction qui suit cette boucle *Repeat*.

Exemple d'utilisation :

Repeat my_repeat= new Repeat(Condition cond, Node fils_g, Node fils_d);
my_repeat.translate();

La classe While

Cette classe se compose d'une condition et d'une suite d'instructions. Elle correspond au *while* en C++.
Elle contient un champ *Condition* (une classe que l'on a implémenté également) et héritant de *Node*, contient également un fils gauche et un fils droit.
Dans ce fils gauche, on trouvera l'ensemble des instructions contenues dans la boucle.
Le fils droit contient l'instruction qui suit cette boucle *While*.

Exemple d'utilisation :

While my_while= new While(Condition cond, Node fils_g, Node fils_d);
my_while.translate();

La classe Condition

Cette classe contient une expression qui, *in fine*, aura pour valeur *true* ou *false*.
Cette classe hérite de *Node* et contient donc un fils gauche et un fils droit.
Cette classe est probablement "finale", i.e. ses fils gauches et droits sont nuls.

Exemple d'utilisation :

```
Condition cond= new Condition();
cond.translate();
```

La classe Instruction

Cette classe est une classe générique pour les différentes instructions du programme. Elle hérite de *Node* et certaines classes en héritent également (les classes de boucles par exemple).
Son fils gauche contient les éléments nécessaires au fonctionnement de cette instruction (tout dépend du type d'instruction) et le fils droit contient l'instruction qui suit.

Exemple générique d'utilisation :

Soit le cas d'une boucle *While* avec trois instructions inst_1, inst_2 et inst_3.
Le fils gauche de l'instruction inst_1 contiendra les éléments nécessaires à l'instruction en question. Son fils droit contient l'instruction inst_2.
De la même manière, le fils droit de l'instruction inst_2 contient l'instruction inst_3 et le fils droit de l'instruction inst_3 est vide.

La classe DeclarationFunction

Cette classe est utilisée pour représenter une fonction qui a été reconnue en EZ language. Elle est appelée dans les fichiers .yacc et crée un objet *DeclarationFunction* qui traduira ensuite la fonction EZ en fonction C++. Elle hérite de la classe Node et possède donc également un fils gauche et un fils droit. Le fils gauche contient les instructions qui sont présentes dans la fonction et le fils droit pointe sur la suite du programme (après la déclaration de la fonction). Dans cette classe on trouve également un vecteur de "*Variable"* pour la liste des éventuels arguments de cette fonction. Des chaînes de caractères pour son nom, son type de retour et le nom de la variable utilisée comme variable de retour.

Exemple d'utilisation :

DeclarationFunction *f= new DeclarationFunction(*nom fonction*, *liste d'arguments, type de retour de la fonction, liste d'instructions de la fonction, nom de la variable de retour de la fonction*);

La classe DeclarationProcedure

Cette classe est utilisée pour représenter une procédure qui a été reconnue en EZ language. Elle est appelée dans les fichiers .yacc et crée un objet *DeclarationProcedure* qui traduira ensuite la procédure EZ en fonction C++. Elle hérite de la classe Node et possède donc également un fils gauche et un fils droit. Le fils gauche contient les instructions qui sont présentes dans la procédure et le fils droit pointe sur la suite du programme (après la déclaration de la procédure). Dans cette classe on trouve également un vecteur de "*Variable"* pour la liste des éventuels arguments de cette procédure et une chaîne de caractères pour son nom.

Exemple d'utilisation :

DeclarationProcedure *f= new DeclarationProcedure(*nom procédure*, *liste d'arguments, liste d'instructions de la procédure*);

- La classe Conditionnal_instruction
    - → If / Switch

- La table des symboles
    - → La classe hashElement
    - → La classe hashTable

- La classe Iterative_instruction
    - → forall
    - → foreach

- Les classes variable et function