I Implémentation des structures de contrôle

I.1 Implémentation des if et if-else

Principe général

Dans un premier temps on évalue la condition dans l'instruction `if` en utilisant la même fonction qui génère/évalue des expressions.

- si la condition est vraie, elle renvoie 1 et le programme exécute les instructions à l'intérieur du bloc if
- sinon, la condition renvoie un entier different de 1 et la condition sera consideré comme étant fausse, dans ce cas là il passe au bloc else s'il existe, sinon on continue l'exécution du programme

Ce résultat est stocké temporairement dans un registre, qui est ensuite comparé à 1 afin de déterminer quelle branche exécuter.

Concrétement :

1. le registre `rax` (qui contient l'évaluation de l'expression) est comparé à 1. Si la condition est fausse, on saute directement au label `else_...` ou `end_if_...` en utilisant l'instruction `jne`
2. si la condition est vraie, le code du bloc `if` s'exécute noramelement. A la fin de ce bloc, un saut `jmp` permet d'éviter le bloc else
3. si la condition est fausse et si on a un bloc else **après le if**, l'éxecution continue au label `else_...`, où le bloc else est exécuté
4. tous les chemins rejoignenet le label `end_if_...` après l'éxecution du bloc concerné

Remarque (Gestion des labels)
POur chaque structure de contrôle, les labels en assembleur sont nommés de façon systématique en combinant le type du bloc, le numéro du bloc dans le code et lé numéro de ligne dans le code. Par exemple `else_0_5` marqué le début d'un bloc else, qui est le premier bloc else dans le code et qui est situé à la ligne 5. Cette convention permet de se répérer plus facilement et permet d'éviter les doublons eventuels.

I.2 Implémentation des blocs for

Les boucles `for` sont traduites sous forme de **fonctions** en assembleur, avec protocole d'entrée/sortie usuel. Chaque appel à une boucle correspond

donc à un appel de fonction. Pour bien gérér la gestion des boucles imbriquées, on a fait le choix de considérer que les paramètres implicides de la fonction for sont :

- le compteur d'itération `i`
- l'élement courant de la liste `list[i]`

Ces deux valeurs son stockés dans la pile avant l'appel du for comme les paramètres classique, le compteur `i` est initalisé à 0 et mis à jour à chaque iteration et l'élement courant `list[i]` est mis à jour à chaque tour de boucle. Ainsi, les blocs for peuvent être réutilisés, copiés ou déplacés dans le programme sans dépendre de variables globales ou des registres qui sont susceptibles d'être reécrits.

Remarque : Les labels sont gérés de la même manière que pour les blocs `if else` etc

Structure générale

1. avant d'appeler la boucle, on empile les deux paramètres implicites nécessaires à son exécution : le compteur d'itération `i` et l'élément courant de la liste.
2. l'exécution du bloc for se fait via un appel standard à la fonction correspondante, utilisant l'instruction call `for_...`.
3. à l'entrée de la fonction, on réalise le protocole habituel : sauvegarde de la base de pile (rbp) et allocation d'un espace mémoire local sur la pile (sub rsp, ...).
4. en début de boucle, on compare la valeur du compteur `i` à la taille de la liste (stockée dans la section .data). Si `i` est supérieur ou égal à cette taille, un saut conditionnel (`jge`) permet de sortir de la boucle en se dirigeant vers le label `for_end`. Sinon, l'exécution du corps de la boucle continue normalement.
5. à la fin de chaque itération, le compteur est incrémenté, puis un saut inconditionnel (`jmp`) renvoie au début du test de boucle pour lancer la suivante.

Remarque sur ces deux parties : POur assurer une bonne gestion des variables dans les différentes structures de contrôles, on a fait le choix de créer des **tables de symbols séparées** pour chaque bloc. POur les boubles for, en plus de la table de symbols propre au bloc, on applique une allocation dynamique des paramètres implicites (le compteur `i` et l'élément courant

`el`) sur la pile. Ces paramètres sont placés à des offsets négatifs pour ne pas intérférer avec les variables locales et leur déplacements.