uCsharp

Compilateur pour un sous ensemble du Csharp

Auteurs

- Antoine Beyet
- Matthieu Daumas
- Volodia Laniel
- Benoit Lemarchand

uC

Fonctionnalités

- Opérations sur les entiers
- Structures conditionnelles et opérations booléennes
- Définitions et appels de fonctions, mêmes imbriqués
- Définitions et utilisations de structures
- Affectations en cascades et récupération de la valeur d'une affectation
- Définition et déréférencement de pointeurs
- Définitions de types personnalisés
- Typage fort et transtypage (cast) statique
- Gestion des chaines de caractères
- Journal (log) des actions de la TDS lors de la compilation
- Début du programme sur la fonction `main`

Exemple de pile

Tests

Le compilateur est testé grâce à la commande `make test`. Celle-ci effectue deux choses:

- Elle vérifie que tous les fichiers *utests/*-wrong.mcs* renvoient des erreurs de compilation.
- Elle vérifie que tous les fichiers *utests/*.mcs* compilent et engendrent le même code que les *utests/*.expected* (fichiers écrits à la main). Si une différence est constatée celle-ci est affichée.

Nécessite python3

Améliorations possibles & limitations

Optimisations possibles

- **constexpr :** Notre compilateur effectue tous ses calculs à l'exécution (ex: dans `int a = 3 + 4;` `3 + 4` est calculé à l'exécution). Une optimisation possible serait de poser un booléen `constexpr` indiquant si la valeur d'un facteur est connue à la compilation. Ainsi, une opération entre deux `constexpr` peut être effectuée à la compilation.

- **assignation :** Lors d'une assignation la valeur affectée est recopiée en sommet de pile afin de permettre l'assignation en cascade. Si l'assignation est unique la valeur est immédiatement dépilée. Ce comportement pourrait être facilement évité.

- **nop :** Notre code effectue parfois des `nop` (dépilement de 0, cast de I2B, ...). Ex: dans `a || b || c` chaque expression est castée en booléen mais aussi les résultats.

uCSharp

Fonctionnalités

- Définition de classe sans héritage
- Définition d'attributs et de méthodes, publiques ou privés
- Résolution privée uniquement depuis l'intérieur de la classe. Lors de sa déclaration tous les attributs de la classe sont accessibles car les méthodes en cours de définition sont internes. Une fois la classe finalisée, on restreint `searchVar()` aux méthodes et attributs publics grâce à la méthode `CLASS::close()`.
- Création d'un paramètre factice « this » dans les méthodes.
- Définition de namespaces, imbriqués ou non.
- Utilisation des namespaces via « using », erreurs associées (appels ambigüs)

Tests

Ces fonctionnalités ont été testées de manière marginale et incomplète. La méthode de tests exhaustifs appliquée à la première partie est transposable pour l'écriture de tests de la seconde partie.

Pistes et idées

Certaines fonctionnalités de la partie 2 n'ont pas été finalisées :

- **L'instanciation de classe :** Similaire à l'instanciation d'une structure, mais sur le tas via un `SUBR MAlloc`, et le code des méthodes n'est généré qu'une fois. La génération de code associée à l'appel de méthodes, à la construction de classe, au pointeur « this ».
- **L'héritage :** Une classe fille connaît la TDS de sa classe parente. Lors de l'ajout d'une méthode, on ne lève pas d'erreur en cas de duplicata (surcharge). La taille du type (et donc des offsets) de la classe fille est construite à partir de la taille de la classe parente (au lieu de partir de 0).
- **Le pointeur « base » :** Le pointeur « base » est un pointeur « this » dont le type a été casté entre la classe courante (fille) vers sa classe parente.
- **La compatibilité de type et sous-type :** La comparaison dans la en uC entre deux types est faite par une comparaison des références entre les deux instances de la classe TYPE. Pour le uCSharp nous aurions dû implémenter une comparaison de compatiblité.
- **Le mode de passage des paramètres :** Le mode de passage

des paramètres serait un attribut supplémentaire dans le TYPE. Une vérification serait ajoutée lors de l'accès à ce paramètre. Une référence (variable IN/OUT) est un type héritant de PTR dont le déréférencement est implicite.

- **Liaison tardive dynamique :** La liaison tardive dynamique, ou polymorphisme, ajoute un attribut caché pour chaque méthode susceptible d'être surchargée. Cet attribut est initialisé à l'instanciation d'une classe à la valeur du « vtag » de la méthode associée. Ce « vtag » est un entier qui numérote de manière unique, les méthodes de chaque classe. Une classe fille utilisant le compteur de sa classe parente. Lors de l'appel d'une méthode, la valeur du « vtag » courant est ajoutée à l'adresse de début de la vtable, et l'adresse résultante est appelée via `CALLI`. L'instruction à cette adresse dans la vtable est un saut sur le début de la méthode. Lors du polymorphisme, le vtag est propagé à la classe parente, ainsi les fonctions surchargées restent dotés du vtag de la classe fille.