TP n°6, I41 – Projet *Emmanuel Bruno*
L2 Sciences et Technologies
Université du Sud Toulon-Var
Année universitaire 2005 – 2006

Construction de la table des Symboles

Objectif

L'objectif de ce TP est de produire les fonctions qui, à partir de l'analyse syntaxique de la partie déclarative d'un programme Algo, produiront la table des symboles, c'est-à-dire une table contenant les identificateurs (par exemple les variables) déclarés ainsi que des informations les concernant (type, taille et adresse relative).

1. Création des fonctions de gestion de la table des symboles

Nous allons écrire les 4 fonctions permettant de gérer la table des symboles. Celle-ci sera gérée comme une liste chaînée d'éléments de type `objet`.

ATTENTION : vous devrez créer vous-même les fichiers contenant ces fonctions et leurs prototypes : `algo_declaration.h` et `algo_declaration.c`. Vous pouvez ecrire un programme `test_algo_declaration.c` qui contiendra une méthode main() qui permettra de tester ces fonctions.

La table des symboles sera représentée comme une liste chaînée. Nous utiliserons une variable globale `debut_liste_objets` pour pointer sur la première variable de la liste.

1.1. Réalisation

Dans le fichier `algo_declaration.h`, déclarer le type `s_objet` qui est une structure qui représente les informations associées à une variable. Déclarer les type `p_objet` qui est un pointeur sur un `s_objet`. Déclarer une variable globale `debut_liste_objets` qui est un pointeur sur un `s_objet`. Dans le fichier `algo_declaration.c`, créer les fonctions :

- `p_objet creer_objet(char *nom)` : cette fonction alloue la mémoire pour un nouvel `objet` et retourne un pointeur sur celui-ci. Elle initialise le champs `nom`.
- `p_objet nouvel_objet(char *nom)` : cette fonction créée (en utilisant `creer_objet`) et ajoute un nouvel objet dans la liste chaînée `debut_liste_objets` (de type `p_objet`, que vous avez déclarée en variable globale). Attention, on doit vérifier que cet identificateur n'est pas déjà déclaré. Le champs addresse devra être mise à jour pour cela utiliser une variable globale initialisée avec la taille de la mémoire puis décrémentée à chaque ajout de 4 octets. **Cette fonction retourne un pointeur sur le nouvel élément**.
- `void affiche_liste_objets()` : cette procédure affiche chaque élément de la liste chaînée `debut_liste_objets`.
- `void detruire_liste_objets()` : cette procédure libère la mémoire allouée à chaque entrée de la table des symboles, pensez à libérer la mémoire allouée à la chaine de caractères qui représente le nom de la variable.

1.2. Test

Créer et exécuter une procédure `void test_table_symbole()` qui ajoute des éléments dans la liste `debut_liste_objets` et les affiche.

2. Construction effective de la table des symboles

2.1. Réalisation

Dans le programme d'analyse syntaxique, créer les fonctions `declaration` et `suite_declaration` pour pouvoir traiter les déclarations de la forme :

- `(OBLIGATOIRE) A un entier` mais aussi
- `(FACULTATIF) A,B,C,D des entiers`.

Pour cela vous procèderez comme suit :

- La fonction `declaration` lit le premier identificateur d'une ligne de déclaration puis appelle `suite_declaration` en lui passant en paramètre la valeur du premier identificateur.
- Dans `suite_declaration` il y a deux cas :
  1. On lit **« un »** suivit d'un *type*, on peut donc construire l'arbre et ajouter un symbole à la table `debut_liste_objets`.
  2. On va lire une suite de couples (**« , »** ***identificateur***). Dans ce cas on ajoute le premier identificateur (celui passé en paramètre) à la table et **on mémorise le pointeur sur cet élément dans une variable globale**, puis on ajoute chaque identificateur de la liste à la table des symboles **sans mettre à jour leur type et toujours en vérifiant qu'il n'appartiennent pas déjà à la table**. Finalement, on lit « **des** » suivi d'un *type*.
     On va donc parcourir la partie de la table des symboles que l'on vient de créer (à partir de la valeur sauvegardée) pour mettre à jour le type et construire l'arbre correspondant. Pour cela vous créerez une procédure `nœud typage_liste_objets(p_objet premier,int tp)` qui, à partir d'un pointeur sur l'`objet` *premier*, affecte à tous les éléments suivants le type **`tp`** et ajoute chaque identificateur comme fils de `arbre_suite_declaration`
- La procédure `declaration` ajoute l'arbre que `suite_declaration` vient

de construire à l'arbre des déclarations et peut ensuite analyser la ligne suivante jusqu'à ce qu'elle trouve autre chose qu'un identificateur.

-

IMPORTANT: si vous êtes pressé vous pour écrire une fonction déclaration qui reconnait uniquement une suite de la forme "*identificateur* **un** *type"* tant qu'elle trouve des identificateurs. Les identificateurs trouvés sont ajoutés comme des variables du type correspondants.
La boucle s'arretera quand on trouvera autre chose qu'un identificateur en générale le mot clé **Début** qui marque le début de l'algorithme.

2.2. Test

Créer une procédure `test_declaration` qui met à jour la table des symboles et construit l'arbre pour des déclarations de la forme :

- `A un entier`
- `B,C des réels`
- `E,F,F des booléens`

Ces déclarations seront contenues dans un fichier `declaration.txt`.