Lexique Programmation – *LISP*

Département informatique ENSEIRB
24 janvier 2007

Adresse : la mémoire associée à un programme en cours d'exécution est une séquence ordonnée d'emplacements. Une adresse est le numéro d'un emplacement.

Affectation : une affectation est l'action qui consiste à écrire une valeur dans un emplacement mémoire.

Allocation de mémoire : pour exister, une entité a besoin de mémoire. Toute entité définie bénéficie donc d'une allocation mémoire implicite. De plus, certains objets en mémoire sont pointés par des entités et nécessitent eux aussi de la mémoire. *En C, on utilise pour cela une allocation mémoire explicite.* En lisp, l'allocation mémoire est toujours implicite.

Application : on dit d'une expression qu'elle applique une fonction si elle contient un appel à cette fonction.

Argument : un argument est une valeur passée en paramètre à une fonction lors d'un appel de cette fonction.

Automatique : une variable est automatique si sa durée de vie n'excède pas le bloc dans lequel elle est définie.

Bloc : un bloc est une séquence d'instructions pouvant contenir aussi des déclarations et définitions. Aucune instruction ne peut exister en dehors d'un bloc.

Constante : une constante est une entité associée à un identificateur qui ne peut prendre qu'une seule valeur. Une expression constante est une expression qui ne peut prendre qu'une seule valeur.

Déclaration : *la déclaration d'une entité fournit le type de cette entité et des informations sur l'endroit où peut se trouver la définition de cette entité. Une même entité peut être déclarée à plusieurs endroits ; toutes les déclarations doivent s'accorder avec la définition de cette entité.*

Définition : *la définition d'une entité déclare le type de cette entité et lui réserve un espace en mémoire. Dans un programme donné, une entité est /définie/ à un seul endroit. Le corps d'une fonction doit être fourni lors de sa définition.*

Durée de vie : la durée de vie d'un objet correspond à la période de l'exécution d'un programme comprise entre la création de cet objet et sa destruction.

Effet de bord : une expression a un effet de bord si elle modifie un emplacement mémoire. Une fonction a un effet de bord si elle contient une expression dont l'effet n'est pas limité aux variables automatiques de cette fonction. Par exemple, écrire ou lire dans un fichier est un effet de bord.

Entité : dans ce lexique, parmi tous les objets qui peuvent exister dans la mémoire d'un programme en cours d'exécution, nous avons distingué les objets nommés déclarés dans le programme et les appelons entités. Par exemple, dans la définition 'const char *s = "bla";', 's' est une entité qui pointe sur un objet représentant la chaine de caractères "bla".

Entité dynamique : une entité dynamique utilisée dans le corps d'une fonction est recherchée dans l'environnement de l'appel à la fonction.

Entité lexicale : une entité lexicale utilisée dans le corps d'une fonction est recherchée dans l'environnement lexical de la fonction.

Environnement : un environnement est un ensemble de liaisons (dictionnaire) associant un identificateur à une entité. L'environnement de définition d'une fonction est appelé **environnement lexical**.

Évaluation : afin qu'une expression produise un effet, elle doit être évaluée.

Exécution : une instruction, une fonction, ou un programme doivent être exécutés afin qu'ils produisent les résultats dont ils décrivent la marche opératoire.

Expression : une expression est une construction du langage dont l'évaluation fournit une valeur, à l'exception des expressions de type 'void' en C.

Fermeture : une fermeture est la représentation d'une fonction sous forme d'un couple associant l'environnement lexical et le code de la fonction.

Fonction : une fonction est un bloc précédé d'un prototype donnant un nom à la fonction. Une fonction peut être appelée à partir d'autres endroits du programme et peut recevoir des arguments au travers de paramètres.

Global : une entité est globale si sa portée dépasse le cadre d'une fonction. Sinon, on parle d'entité locale.

Identificateur : un identificateur est le nom d'une entité.

Initialisation : l'initialisation d'une entité consiste à donner à son espace mémoire associé un contenu suffisamment déterminé pour être cohérent.

Instruction : en C, la notion d'instruction est syntaxique : elle dénote les morceaux du programme source qui ne sont pas des déclarations de variables mais qui sont terminés par des point-virgules, ou des blocs délimités par des accolades. Un bloc peut commencer par des déclarations de variables dont la durée de vie est limitée à ce bloc. En lisp, on parle uniquement d'expressions.

Local : une entité est locale à un bloc si elle n'est visible qu'à l'intérieur de ce bloc.

Mémoire : la mémoire associée à un programme en cours d'exécution contient plusieurs entités et objets : le code du programme, issu des fonctions, une pile d'exécution où se trouvent les variables internes aux fonctions, une zone de données où se trouvent les autres entités, et enfin un tas, dans lequel peuvent avoir lieu des allocations de mémoire.

Opérateur : un opérateur est une des opérations de base proposées par le langage, et qui est utilisée par l'intermédiaire d'une syntaxe spécifique qui n'est pas l'appel de fonction, pour former des expressions. Par exemple l'addition et l'affectation sont deux opérateurs du langage C.

Paramètre : un paramètre est une variable locale à une fonction qui est initialisée lors de l'appel d'une fonction par une copie de l'argument correspondant.

Pointeur : un pointeur est une entité contenant l'adresse d'un objet.

Portée : la portée d'une entité est la partie du code source dans laquelle il est possible d'utiliser cette entité.

Prototype : le prototype est le nom donné à la déclaration d'une fonction.

Récursivité terminale : un appel récursif est dit terminal si il n'est pas argument d'une application. Une fonction est dite récursive terminale si tous ses appels récursifs sont terminaux.

Référence : une référence est un objet correspondant à une adresse mémoire et dont l'indirection est faite automatiquement dans toute situation où une valeur est requise. L'adresse associée à une référence n'est pas directement manipulable en tant que telle (il n'existe pas d'opérations pour le programmeur sur les références).

Type : chaque objet possède un type qui définit les opérateurs ou fonctions qui peuvent le manipuler, et quelle est leur action. Tout type possède une taille, qui représente la quantité de mémoire requise pour le stockage des valeurs du type.

Typage statique : association d'un type à chaque entité.

Typage dynamique : seuls les objets sont typés et non les entités.

Valeur : une valeur est un objet sur lequel on ne peut pas faire d'affectation.

Variable : une variable est une entité typée, associée à un identificateur, qui peut contenir une valeur parmi toutes celles de son type.

Visibilité : la visibilité d'une entité associée à un identificateur est la partie du code source où l'utilisation de l'identificateur est possible et où il correspond à cette variable. Il est possible que la réutilisation d'un même identificateur pour une deuxième variable masque temporairement la première variable.