Paradigme de Programmation Impérative

Noureddine / Seddari

Paradigme de Programmation Impérative

Types de Données

Dr. Noureddine SEDDARI

Plan du Cours

- Le Concept d'un Type de Données
- Types de données primitifs et semi-primitifs
- Types de données Enumératifs et de Sous-Gamme
- Types de données Structurées: Tableaux, Enregistrements, Unions.
- Ensembles
- Pointeurs

Le Concept d'un Type de Données

- Dans les premiers langages (e.g., Fortran I), toutes les

  structures de données étaient définies par des structures de données de base supportées par le langage.
- Par la suite, d'autres langages tels que le Cobol et le PL/1 ont rajouté des types de données plus précis.
- En Algol 68, une autre approche a été proposée: quelques

  types de données de base sont présents ainsi que quelques opérateurs permettant au programmeur de définir ses propres types de données.
- Plus tard, ceci a donné lieu à l'idée de **type abstrait** de données ADT.

Le Concept d'un Type de Données (Suite)

- Un type de données n'est pas qu'un ensemble d'objets. On doit aussi considérer toutes les opérations qui peuvent être appliquées à ces objets.
- Une définition de type complète doit donc inclure une liste de toutes les opérations ainsi que leurs définitions
- Il faut aussi se rappeler qu'un type de données n'est qu'une abstraction: seul le bit et l'octet ont une existence tangible.
- Néanmoins, les types de données ***primitifs*** ne sont pas très éloignés du hardware.

Types de Données Primitifs: Les Entiers

- La plupart des ordinateurs permettent maintenant l'existence d'entiers de tailles différentes et ceci est reflété dans les langages de programmation qui parfois permettent: des entiers sans signes, des entiers avec un signe, des entiers courts, normaux ou long.
- Les entiers sont représentés directement dans l'ordinateur par une chaîne de bits.

Types de Données Primitifs: Les Points Flottants

- Les Points Flottants représentent les réels jusqu'à un certain degré de correction.
- Comme dans la notation scientifique, ils sont représentés par une fraction (mantisse) et une puissance (exponent).
- La plupart des langages incluent deux types de points flottants: float et double.
- Le float prend d'habitude 4 octets de mémoire alors que les doubles prennent deux fois la place des floats et donne au moins deux fois le nombre de bits de fraction.

Types de Données Primitifs: Les Booléens

- Le type booléen est le plus simple de tous: il n'a que deux valeurs: true ou false.
- Bien qu'une variable booléenne puisse-t-être sauvegarder dans un seul bit de mémoire, elle est souvent sauvegardée dans un octet car cela rend l'accès plus efficace.
- Les booléens peuvent être implémentés avec des entiers, mais ils rendent la lecture plus facile.
- Pas tous les langages supportent les booleens: il n'y en a pas en PL/1 ou en C.

Types de Données Primitifs: Les Caractères

- Les caracteères sont sauvegardés en mémoire avec des codes numériques, tels que l'ASCII.
- L'ASCII utilise les valeurs 0 ..127 pour encoder 128 caractères différents.
- Néanmoins, des codes plus complets, existent aussi. L'Unicode qui incluent les caractères de la plupart des langages du monde est un ensemble de caractères représenté avec 16 bits.
- Java est le premier langage populaire à utiliser l'Unicode.

Types de Données Semi-Primitifs: Les chaînes de caractères

- Une chaîne de caractères est une séquence de caractères. Elle peut être:
  - Un type de donnée spécial (dont les objets peuvent être décomposés en caractères): FORTRAN, BASIC
  - Un tableau de caractères: Pascal, ADA
  - Une liste de caractères: Prolog
  - Des caractères sauvegardés consécutivement: C
- **Forme Syntaxique:** Pascal n'a qu'un type de guillemets. Ada en a deux: 'a' est un caractère et "a" est une chaîne de caractères.

Types de Données Semi-Primitifs: Les chaînes de caractères (Suite)

- Opérations sur les chaînes de caractères lorsqu'elles sont définies comme un propre type de données:
- string x string→ string — concaténation
- string x entier x entier→ string — sous-chaîne
- string → caractères — décompose la chaîne en un tableau ou une liste de caractères
- caractères → string — convertit un tableau ou une Liste en une chaîne

Types de Données Semi-Primitifs: Les chaînes de caractères (Suite)

- string → entier — longueur
- string → booléen — chaîne vide?
- string x string → booléen — chaînes égales? dans quel ordre?
- Dans certains langages spécialises dans la manipulation de chaînes (Snobol, Icon), il y a des opérations de pattern-matching disponibles. Par exemple:
  - Trouver la première présence d'une petite chaine dans une chaîne plus large;
  - Trouver une chaîne avec des parenthèses balancées
  - Trouver des séquences de mots

Types de Données Semi-Primitifs: Les chaînes de caractères (Suite)

- La longueur permise des chaînes est un problème de conception du langage: en Pascal, Ada et Fortran, les chaînes sont de taille fixe. En C et Java, elles sont a longueur variable.
- Un problème avec les chaînes a longueur variable: l'allocation d'un espace possible cause du gâchis. Par contre, l'allocation caractère par caractère doit être faite trop fréquemment. Une stratégie efficace est d'allouer des blocs de 128 caractères (par exemple). Toutes les chaînes comprise entre 1 et 128 caractères occupent 1 bloc. Toutes celles entre 129 et 256, deux, etc.

Types de Données Énumératifs

- Il est possible de déclarer une liste de constantes symboliques à traiter littéralement (ceci veut dire qu'elles ne représentent pas d'autres valeurs tel que "const pi = 3.14" le fait).
- Implicitement, on donne également un ordre à ces constantes symboliques.

Exemple: type day=(mo, tu, we, th, fr, sa, su) est tel que mo<tu<we<…

- Pascal a trois opérations pour chaque type numératif:
  - succ: successeur eg, succ(tu)=we
  - pred: predecesseur, eg, pred(su)=sa
  - ord: position dans le type: ord(mo)=0; ord(we)= 2

Types de Données Énumératifs (Suite)

- En Ada, la liste est un peu plus complète: succ(mo)=tu, pred(tu)=mo, pos(mo)=0, val(3)=th, FIRST et LAST.
- Est il permis d'avoir une constante symbolique définie dans plus d'un type? En Pascal, non. En Ada, oui.

  type stoplight **is** (red, orange, green);
  type rainbow **is** (violet, indigo, blue, green, yellow, orange, red)
- Les descriptions qualifiées enlèvent la confusion: stoplight'(red) ou rainbow'(red)
- L'implémentation des types énumères: constantes c1, .., ck → entiers 0, …, k-1
- Néanmoins, ils améliorent la lisibilité.

Types de Sous-Gamme

- Il est possible de définir un type qui représente une sous-gamme de valeurs
- **Exemple** en Pascal: **type** capitales = 'A'..'Z'; toutes les opérations qui s'appliquent aux caractères s'appliquent également au type capitales qui est un sou type des caractères.
- En Ada il y a des types dérivés et des sous-gammes. Ce n'est pas la même chose:
- **type** petits_entiers_derives **is new** integer **range** 1..100;
- **subtype** petits_entiers_sous_gamme **is** integer **range** 1..100;
- Ces deux types héritent des opérations sur entiers. Néanmoins les variables de type petits_entiers_derives ne sont pas compatibles avec les entiers.

Types de Données Structures: Les Tableaux

- Un tableau représente l'application: type_d'index → type_de_composante
- Le type d'index doit être un type à gamme non continue: entier, caractère, énumération. Dans certains langages ce type est spécifié indirectement: A(n) en C signifie 0..N, alors qu'en Fortran c'est 1..N.
- Il y a en général peu de restrictions sur le type de composante (ils peuvent même être des tableaux, des procédures ou des fichiers).
- La forme des référence est A[I] en Algol, Pacal et C et A(I) en Fortran et Ada.

Types de Données Structurées: Les Tableaux (Suite)

- Les tableaux multi-dimensionnels peuvent être définis de deux façons différentes:
- type_d'indexe1 x type_d'indexe2 → type_de_composante, ou
- type_d'indexe1 → (type_d'indexe2 → type_de_composante)
- La première définition correspond à des références telles que A[I,J] alors que la deuxième correspond à des références telles que A[I][J]

Types de Données Structurées: Les Tableaux (Suite)

Opérations sur les tableaux:

- Sélection d'un élément A[I]
- Sélection d'une "tranche" d'éléments A(1:10) ou même A(2:10:2). Possible en Fortran 90 et de manière plus restreinte en Ada.
- Affectation d'un tableau complet a un autre: A := B; il y a une boucle implicite dans cette instruction.
- Calcul d'expressions avec des tableaux entiers: A:= A + B. Fortran 90 implémente ces opérations ainsi que APL qui en ajoute d'autres telles que les transposition, les inversions les "inner products"…

Types de Données Structurées: Les Tableaux (Suite)

Attachement d'indexes. 4 types de tableaux:

- Tableaux Statiques: l'indexe a un attachement statique et l'allocation de mémoire est statique.
- Tableaux "Fixed Stack-Dynamic": l'indexe a un attachement statique mais l'allocation de mémoire est faite au moment de l'élaboration des déclaration pendant l'exécution.
- Tableaux "Stack-Dynamic": l'indexe a un attachement dynamique et l'allocation de mémoire est dynamique mais cet attachement et allocation sont fixes une fois faits.
- Tableaux "Heap-Dynamic": l'indexe a un attachement dynamique et l'allocation de mémoire est dynamique et ces attachements et allocations peuvent changer pendant l'exécution.

Types de Données Structurées: Les Tableaux (Suite)

Initialisation de Tableaux

- Beaucoup de langages permettent l'initialisation de tableaux dans la déclaration:
- C int vector [] ={10,20,30};
- Ada vector: array(0..2) of integer := (10, 20, 30);
- Autre type d'affectation en Ada:
  temp is array (mo..fr) of -40 .. 40;
  T: temp; T:= (5,12,8,30,25);
  T:= (mo => 5,we =>8,tu=>12,
  fr=>25,others=>30); T:=(5,12,8,fr=>25,th=>30);
- Attributs de tableaux en Ada:
  - TM: temp; TM'FIRST; TM'LAST; TM'LENGTH; TM'RANGE;

Types de Données Structurées: Les Tableaux (Suite)

Implémenation de Tableaux: Comment sauvegarder les tableaux et accéder à leurs éléments?

- Pendant l'exécution un tableau est représenté par un ***descripteur de tableau*** qui nous donne:
  - Le type de l'indexe
  - Le type des composantes
  - L'adresse de base du tableau (i.e., des données)
  - La taille d'une composante
  - Les limites inférieures et supérieures de l'indexe

Types de Données Structurées: Les Tableaux (Suite)

Implémentation de Tableaux Multi-dimensionnels

- Un tableau multi-dimensionnel est représenté par un descripteur incluant plusieurs paires de limites inférieures et supérieures.
- Les tableaux multi-dimensionnels doivent aussi être représentés en mémoire qui n'est que uni-dimensionnelle. Deux approches: **row-major** (le deuxième indexe change le plus vite) ou **column-major** (le premier indexe change le plus vite).

Types de Données Structurées: Les Tableaux (Suite)

Exemple d'accès à une composante:

- A: array [LOW1 .. HIGH1, LOW2..HIGH2] of ELT; la taille de ELT est SIZE.
- L'illustration est pour une organisation row-major (ce serait similaire pour column-major). On assume que l'adresse de base du tableau (i.e., A[LOW1, LOW2]) est LOC.
- On peut calculer l'adresse de A[I,J] comme ceci:

  (I-LOW1)* ROWLENGTH * SIZE +
  (J-LOW2)*SIZE+LOC
  Avec ROWLENGTH=HIGH2-LOW2+1

- Quoique les tableaux n'existent pas en Scheme, ML et Prolog, ils peuvent être simulés par des listes

Types de Données Structurées: Les Enregistrements

- Un enregistrement est une collection hétérogène de champs (ou composantes). Les tableaux, par contre, sont homogènes.
- Les enregistrements sont supportés par la plupart des langages importants: Cobol (qui les a introduits), Pascal, PL/1, Ada, C (appelés structures), Prolog, C++.

- Les champs ont des noms plutôt que des indices et on ne peut pas itérer sur ces indices.
- Un champ est indiqué par un nom de variable qualifié, mais le nom de la variable peut être omis si la variable de référence est déjà connue (**"with"** en Pascal).

Types de Données Structurées: Les Enregistrements (Suite)

Opérations sur les enregistrements:

- Sélection d'une composante par nom de champ.
- Construction d'un enregistrement à partir de ses composantes. Ou bien champ par champ ou bien d'un seul coup (à partir d'une constante structurée).
- Affectation d'enregistrements complets
- Comparaison entre deux enregistrements (seulement une vérification d'égalité. Il n'y a pas d'ordre standard pour les enregistrements).

Types de Données Structurées: Les Enregistrements (Suite)

- Ada permet des valeurs de défaut pour les champs d'enregistrements.
- Il n'y a pas beaucoup de restrictions sur les types des champs. Toute combinaison d'enregistrement et de tableau
  (quelque soit sa profondeur) est habituellement permise
  Un champ peut même être un fichier ou une procédure.
- Les champs d'un enregistrement sont sauvegardés dans des locations adjacentes de mémoire. Lorsque les cham sont accédés, il faut se souvenir que comme la taille de chaque champ peut-être différente, la méthode d'accès est un peu différente de celle décrite pour les tableaux: chaque champ a son propre "offset" relative au début d l'enregistrement

Types de Données Structurées: Les Enregistrements IV

- En Prolog, les enregistrements indiquent tous leurs types et composantes:
- date(day(15), month(10), year(1994))
- person(name(fname("Jim"), lname("Berry")), born(date(day(15),month(10),year(1994)), gender(m))
- On peut simplifier cette notation si on peut s'assurer d'une utilisation correcte:
- date(15,10,1994)
- Person(name("Jim","Berry"), born(date(15,10,1994),m)

Types de Données Structurées: Union Discriminées de Types

- Une union de type signifie un ensemble d'objets provenant de ces types ainsi que des opérations sur ces objets.
- Bien que les types peuvent être incompatible, cela pose des problèmes.
- Néanmoins, bien que des types incompatibles ne peuvent pas être attachés à un objet au même moment, ceci est possible à différents moments de la vie de l'objet
- Une union discriminée est une union dans lequel le type actuel de l'objet peut toujours être connu.
- Ceci peut être implémenté en ajoutant une composante spéciale à une union: un "tag" ou "discriminateur" qui indique le type de l'objet.

Types de Données Structurées: Union Discriminées de Types (Suite)

- Les opérations sur les enregistrements a variante seront correctes si le champ discriminateur est vérifié.
- S'il n'est pas bien vérifié, il y aura des problèmes car l'utilisation de variantes peut conduire a des situations inconsistantes lorsque les champs reçoivent de affectations individuelles.
- Un langage comme l'Ada rend ces inconsistances impossible car il ne permet les affectations qu'en agrégation.

Types de Données Structurées: Union Discriminées de Types (Suite)

Implémentation d'Unions Discriminées:

- Les unions discriminées sont implémentées en donnant la même adresse a chaque variante possible.
- Une taille de mémoire suffisante à accommoder la variante la plus large est choisie pour toute les variantes.

Ensembles

- Les ensembles ne sont permis qu'en Pascal et en Modula-2.
- Ils ne sont possibles qu'avec des types basés sur les entiers et avec très peu d'objets (moins de 100 dans beaucoup d'implémentations de Pascal).

- **Exemple**
- **type** charset= set of char;
  var set1=charset;
  vowels=['a','e','I','o','u'];
- On peut écrire **if** ch **in** vowels…
- Plutôt que **if** (ch='a') **or** (ch='e') **or**…
- Il y a des opérations sur les ensembles: affectation, union, intersection, différence, appartenance, inégalité inclusion
- Implémentation: avec un "bit map"

Pointeurs

- Une variable de type pointeur a comme valeur une adresse (et une adresse spéciale (nil ou null) lorsqu'elle n'a pas de valeur).
- Ils sont utilises principalement pour construire des structures de taille et de formes imprédictibles (listes, arbres, graphes) à partir de petits fragments alloués dynamiquement pendant l'exécution.
- Un pointeur à une procédure est possible mais normalement on a des pointeurs à des données simples ou composées.
- Une variable est composée d'une adresse, d'un type de données et d'une valeur. Mais c'est une ***variable anonyme*** car elle n'a pas de nom.

Pointeurs (Suite)

- L'accès a la valeur d'une telle variable se fait en déréférenciant le pointeur:

- Valeur(p)= p^ = α et Valeur(p^)= α^ =17
- Comme avec des variables normales, dans "p^ :=23;", on veut dire l'adresse de p^ (ou bien, la valeur de p). Dans "m=p^;", on veut dire la valeur de p^.

Pointeurs (Suite)

- Une variable de type pointeur est déclarée explicitement et a une portée et une durée de vie comme d'habitude.
- Une variable anonyme n'a pas de portée (car elle n'a pas de nom) et sa durée de vie est déterminée par le programmeur.
- Une telle variable est créée (dans la partie de la mémoire spéciale qui s'appelle le "heap") par le programmeur. E.g.: new(p) en Pascal; p=malloc(4) en C.
- Elle est aussi détruite par le programmeur: dispose(p) en Pascal; free(p) in C.

Pointeurs (Suite)

- Si une variable anonyme existe à l'extérieur de la portée d'une variable explicite de type pointeur, on a su "garbage" (un objet perdu).
- Si, par contre, une variable anonyme a été détruite à l'intérieur de la portée d'une variable explicite de type pointeur, on a une "dangling" référence.
- Garbage Collection est le processus par lequel la mémoire inaccessible est récupérée. Ce processus est complexe et cher. Il est essentiel dans les langages dont l'implémentation dépend elle-même de pointeurs comme le LISP et le Prolog.
- En PL/1 les pointeurs n'ont pas de type. En Pascal, Ada et C ils sont déclarés par rapport a un type, ce qui veut dire qu'un pointeur déférencié (^p ou *p) a un type fixée.
- Les opérations sur les pointeurs sont très riches en C.