ÉCOLE POLYTECHNIQUE MONTRÉAL

CHAPITRE 7

ENVIRONNEMENT Ada

ÉCOLE POLYTECHNIQUE MONTRÉAL

OBJECTIFS

- Devenir familier avec les outils nécessaires pour développer des logiciels en Ada efficacement
- Comment l'environnement pourrait évoluer?

ENVIRONNEMENT DE SUPPORT D'Ada

- Un langage est utile s'il existe de bons outils de support à l'environnement de développement
- L'importance de tels outils a été prise en compte dès les premiers développements du langage Ada (1978) ([*APSEs*] Ada Programming Support Environnements)
- Le rapport STONEMAN (1980) définit les besoins d'*APSE* pour Ada

LE RÔLE D'UN ENVIRONNEMENT DE SUPPORT À LA PROGRAMMATION

- Un bon environnement de développement peut aider à toutes les phases du cycle de vie d'un logiciel:
  - Définition des besoins
    - Outils d'analyse des besoins et de gestion de projet
  - Spécifications
    - Outils de spécifications formelles
  - Conception (Désign)
    - Outils et langage de désign
  - Codage
    - Éditeurs structurés, gestionnaire de configuration de logiciel
  - Test
    - Outils de tests des composantes
  - Maintenance
    - Outils de gestion et de contrôle de code source

POURQUOI STANDARDISER UN PSE?

- Des PSEs normés augmentent la productivité et la portabilité
- UNIX démontre le besoin de PSEs normés et portables
- PSEs se basent sur des environnements *hôtes/cibles*
- Trois niveaux d'APSEs:
  - Noyau APSE
  - APSE minimal
  - APSEs

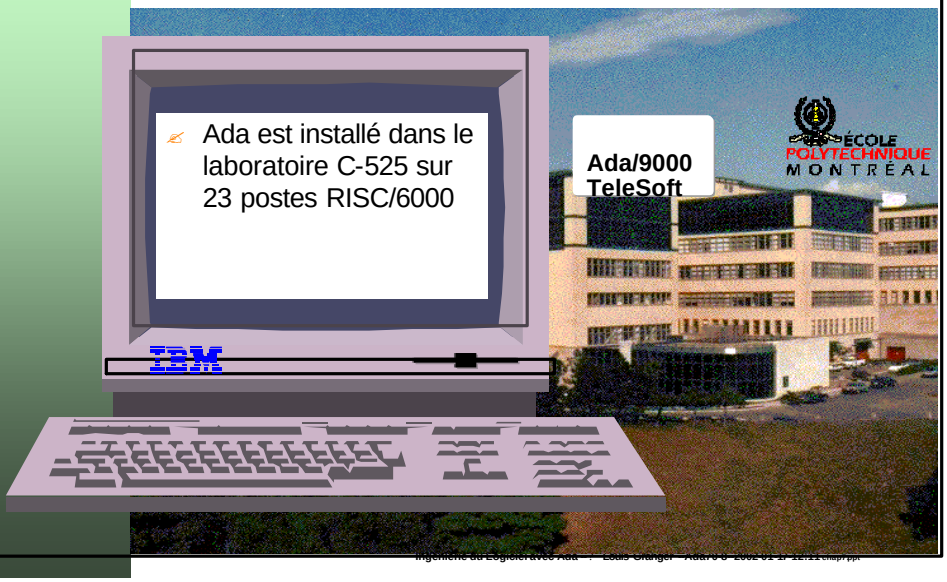
NOYAU APSE

- APSE inclut un support d'exécution pour Ada [*runtime support*]
- KAPSE [Kernel APSE] implémente un support d'exécution sur un matériel hôte en fonction du système d'exploitation

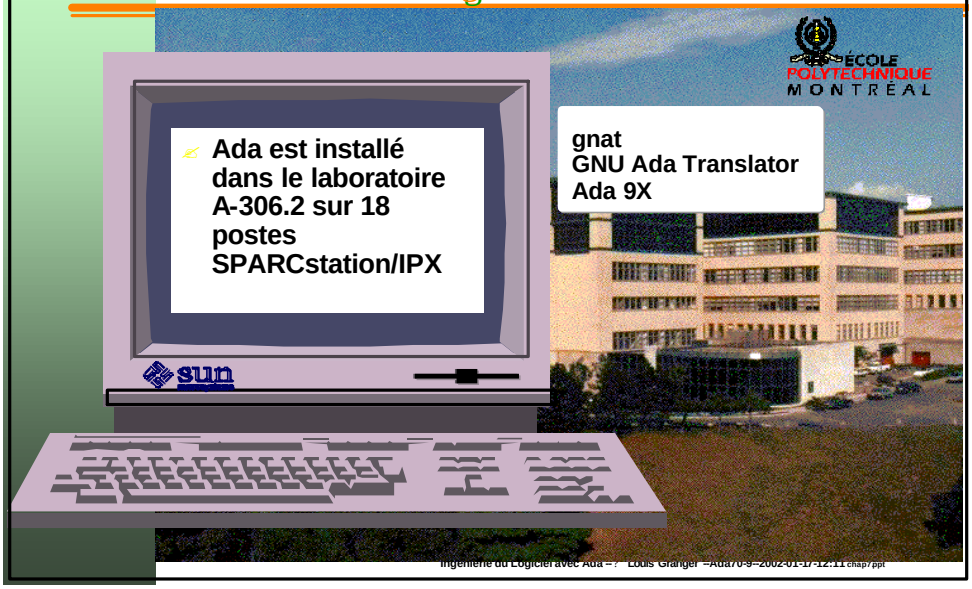
APSE Minimal

- Tel qu'énoncé dans STONEMAN, un MAPSE inclut des outils tels que:
  - un Éditeur de texte et un PP [Pretty Printer]
  - un Compilateur et Éditeur des Liens
  - outils statiques et dynamiques d'analyse
  - gestionnaire de fichiers
  - interpréteur de commande
  - gestionnaire de configuration

COMPILATEUR Ada/9000



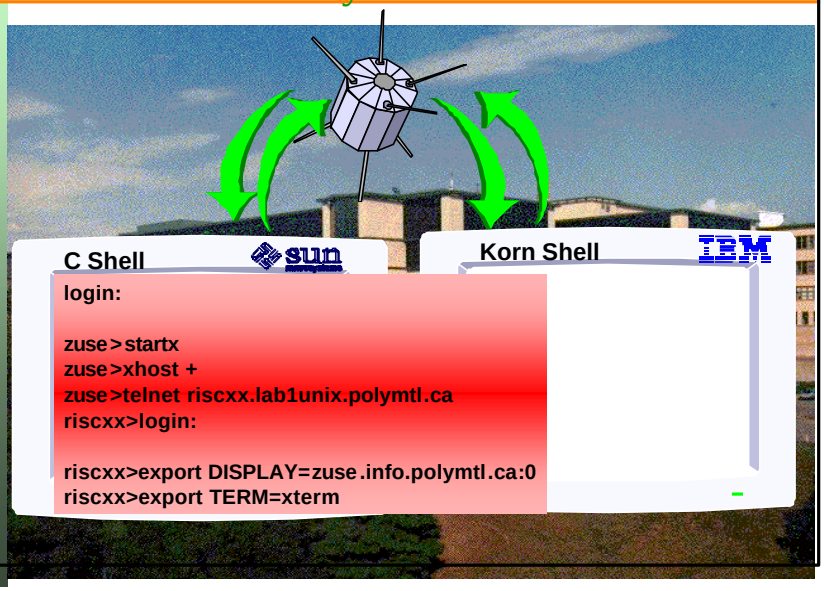
COMPILATEUR gnat



- Ada est installé dans le laboratoire A-306.2 sur 18 postes SPARCstation/IPX

gnat
GNU Ada Translator
Ada 9X

Laboratoires Poly



C Shell

```
login:

zuse>startx
zuse>xhost +
zuse>telnet riscxx.lab1unix.polymtl.ca
riscxx>login:

riscxx>export DISPLAY=zuse.info.polymtl.ca:0
riscxx>export TERM=xterm
```

Korn Shell

Aide -- Éditeurs

- Aide, documentation

  info -- fait appel à InfoExplorer

- Éditeur

  vi NomFichier

  emacs &

  xemacs -u emacs &

  workbench.sh

STYLES NORMALISÉS D'Ada

- Le MRL adopte un standard consistent pour l'indentation et l'utilisation de majuscules/minuscules
- Autres styles:
  - usage de commentaires
  - choix judicieux de nom d'objets
  - nom de sous-programmes : verbe
  - restrictions sur des constructions non-sécuritaires
- Un bon style améliore la lisibilité et la maintenance
- Un bon style peut assister la portabilité et l'efficacité

FORMATEUR DE PROGRAMMES

- *afmt* est un utilitaire pour formater les fichiers sources d'Ada selon certaines règles d'indentation et d'utilisation des majuscules/minuscules
- Les conventions sont des abréviations des mots suivants:

firstupper	Première lettre des identificateurs ou mots réservés en majuscule
mixed	Première lettre des identificateurs ou partie des identificateurs précédé d'un _ ou mots réservés en majuscule
lowercase	Tout en minuscules
uppercase	Tout en majuscules
same	Mots réservés ou identificateurs non-altérés

FORMATEUR DE PROGRAMMES

- **Options**
  - i *Convention* — Convention utilisée pour les identificateurs
  - r *Convention* — Convention utilisée pour les mots réservés
  - h — Aide
  - k — Conserver les chaînes et littéraux intacts
  - n *Indentation* — Défini l'indentation (Défaut 2)
  - w *LargeurLigne* — Défini la largeur de la ligne (Défaut 80)
  - N — Ajouter les numéros de lignes à gauche

  NomFichier — Nom du fichier à formater
- La sortie est dirigée sur la sortie standard *[stdout]* (peut être redirigée dans un fichier avec >)

FORMATEUR DE PROGRAMMES

```
afmt -i upp -r low -n 3 alert.c
```

```
with NEW_ALERT_SYSTEM;
package EMERGENCY_ALERT_SYSTEM is
   type EMERGENCY_ALERT is new
            NEW_ALERT_SYSTEM.ALERT with private;
   procedure HANDLE ( EA : in out EMERGENCY_ALERT);
   procedure DISPLAY (EA : in EMERGENCY_ALERT;
                ON : in NEW_ALERT_SYSTEM.DEVICE);
   procedure LOG ( EA : in EMERGENCY_ALERT);
end EMERGENCY_ALERT_SYSTEM;
```

```
with New_Alert_System;
package EMERGENCY_Alert_System is
  type Emergency_Alert is new
      NEW_ALERT_SYSTEM.ALERT with privat
  procedure Handlle ( EA : in out EMERGENCY_
  procedure DISPLAY (EA : in EMERGENCY_ALERT;
            ON : in NEW_ALERT_SYSTEM.DEVICE);
  procedure Log ( EA : in EMERGENCY_ALERT);
end EMERGENCY_ALERT_SYSTEM;
```


COMPILATION & ÉDITION DES LIENS

```
ada/tp2>$alibinit
ada/tp2>$ada -mvl ration.ads ration.adb main.ada
ada/tp2>$ada -b evaluate
```

Initialiser une librairie pour Ada 1 ière fois seulement

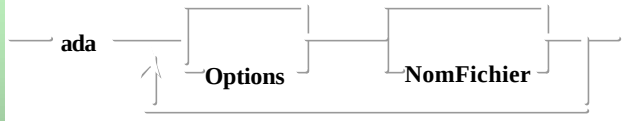
Compiler les fichiers: ration.ads ration.adb main.ada

Édition des liens: nom de la procedure principale dans main.ada

Module exécutable

a.out

COMMANDE ada
Compilation & Édition des Liens



COMMANDE ada
Compilation

Options	Description	Défaut
-v	Affiche tous les messages	Sans copyright
-o Nom	Nom du module exécutable ou du module objet	a.out
-l	Produit un listage avec les erreurs aux lignes appropriées	
-a	Produit un listage ASM	
-G	Produit code optimisé (débogueur)	
-O	Produit code optimise (sans débogueur)	
-p	Produit du code fonctionnant avec le profileur	

COMMANDE ada
Édition des Liens

Options	Description	Défaut
-b *NomUnité*	Produit un module exécutable en utilisant NomUnité comme nom d'unité principale	
-m	Compile les fichiers sources et produit un module exécutable dont le nom est le nom de la dernière unité	
-e	Produit un module objet si l'option -b ou -m est spécifiée.	
-i *NomFIchier*	Inclure un module objet	

GESTIONNAIRE DE FICHIERS (BIBLIOTHÈQUES)

- Une bibliothèque d'Ada est une liste de bibliothèques de programmes appelées sous-bibliothèque (sous-librairie)
- Le compilateur d'Ada utilise ces sous-bibliothèques pour tenir à jour son information sur une application
- Le système Ada/9000 contient donc des commandes pour la gestion des bibliothèques de programmes Ada
- Il est fortement recommander de ne pas utiliser les commandes de UNIX telles : *rm, mv cp* pour la gestion de fichiers sources

GESTIONNAIRE DE FICHIERS (BIBLIOTHÈQUES)

- Les principales commandes pour la gestion des bibliothèques sont:

afilemv	Renomme le fichier source compilé dans la bibliothèque
alibinit	Initialise une sous-bibliothèque
alibchk	Vérifie une sous-bibliothèque
alibmv	Renomme une sous-bibliothèque
alibcp	Copie une sous-bibliothèque dans une autre
alibrm	Élimine une sous-bibliothèque
aunitmv	Renomme ou déplace des unités de compilation d'une sous-bibliothèque à une autre
aunitcp	Copie une ou plusieurs unités de compilation d'une sous-bibliothèque à une autre
aunitrm	Élimine une ou plusieurs unités de compilation d'une sous-bibliothèque
areport	Affiche l'information au sujet des unités compilées

FAPSE *[Full]*

- STONEMAM énumère des outils tels que:
  - un gestionnaire de base de données
  - système d'analyse, transformation, d'affichage, et de maintenance du logiciel
- D'autres outils pourraient générer automatiquement du code

GESTIONNAIRE DE BIBLIOTHÈQUE DE PROGRAMMES ET CONFIGURATIONS

- La plupart des gros systèmes logiciels évoluent dans le temps en une "famille arborescente" de versions
- Les outils nécessaires:
  - outils d'archivage de version (e.g. UNIX rcs)
  - compilation automatisée (e.g. UNIX make)
- Ada spécifie quand une recompilation est nécessaire, mais ne spécifie pas comment les implantations des autres bibliothèques sont gérés

Outils de Développement - MAKE

- *make* est un utilitaire de UNIX, qui à partir d'un fichier de commandes *makefile* fait appel au compilateur approprié pour compiler des fichiers sources et ensuite, s'il y a lieu effectuer l'édition des liens.
- La commande *amakedep* produit un fichier *makefile* qui peut être utilisé par la commande *make* en tenant compte des dépendances des différents fichiers sources contenant des programmes Ada.

Outils de Développement - MAKE

– h	Aide
Fichiers	Balaye tous ces fichiers pour créer les dépendances
– F	Indique d'écraser un fichier *makefile* existant
– a	Indique d'ajouter les Fichiers au *makefile*
– i	Ignorer certaines dépendances provenant du pragma INLINE
– g	Ignorer certaines dépendances provenant d'unités génériques
– b *Nom*	Spécifie le nom de l'unité principale pour l'édition des liens
– o *Nom*	Spécifie le nom du module exécutable contenant l'unité principale

Outils de Développement - MAKE

- La compilation est lancée par la commande *make*
  - $ amakedep ration.ads ration.adb eval.ada
  - $ make
  - $ make -f MonMakefile
  - $ make -f MonMakefile
- Symboles spéciaux (Variables d'environnement)

ADAFLAGS	Chaîne de caractères contenant les options de compilation
ADABINDFLAGS	Chaîne de caractères contenant les options d'édition des liens
ADAINITFLAGS	Options de *alibinit*
ADARESOLVE	*=no* ne regarde pas les dépendances dans les sous-librairies

Ada COMME un langage PDL

- [*PDL : Program Design Language*] est un outil pour développer, raffiner et concevoir les documents d'un logiciel. De tels outils sont:
  - diagrammes structurés
  - pseudo-code
  - organigrammes
- Idéalement, la conception d'un programme devrait être suffisamment complète de sorte que la codage dans un langage approprié ne devrait requérir aucun changement.

UTILISATION D'Ada comme PDL

- Le support de la modularité et l'abstraction d'Ada lui permette d'agir comme PDL en:
  - choisissant un sous-ensemble d'Ada
  - relaxant les contraintes syntaxiques et sémantiques
  - ajoutant des extensions à Ada
- L'utilisation d'Ada comme PDL réduit l'effort de codage, mais peut limiter les alternatives de conception

OUTILS DE TESTS ET DÉBOGUAGE

- Ces outils incluent
  - compilateurs incrémentaux / interpréteur
  - profileurs
  - débogueurs statique et dynamique

PROFILEUR

- Le profileur fournit:
  - information sur le nombre d'appels à des sous-programmes
  - statistiques temporelles sur les sous-programmes
- Unilisation:
  - Compiler les unités avec l'option **-p**
  - Exécuter le programme normalement
    Un fichier **gmon.out** est produit
  - Exécuter la commande ***aprof [ModuleExécutable]*** pour produire un rapport de performance

Débogueur statique

- Les débogueurs statiques analysent le code source et détectent les bogues potentiels tels que:
  - les variables inutilisées ou non-initialisées
  - les violations de contraintes
  - les verrous mortels dans les tâches
- L'efficacité des débogueurs statiques peut grandement augmenter l'efficacité en incorporant dans le programme des assertions que le débogueur peut vérifier

DÉBOGUEUR dynamique

- Permet de suivre pas à pas l'exécution d'un programme Ada
- Utilisation:
  - compiler les unités avec l'option **-p**
  - démarrer le débogueur avec la commande

    adbg exemple

DÉBOGUEUR dynamique
Exemple

```
adbg exemple
Debug>run
Program Ready
Debug>source 52-56, exemple
   52:
   53:  begin
   54:
   55=    a := 10;
   56=    c :=  56 + a;
Debug>break 56
Debug>continue
56*  c := 56 + a;
Breakpoint encountered at 56,sec/exemple
```


DÉBOGUEUR dynamique
Exemple

```
Debug>?a
a => 10
Debug>continue
Program Completed Normally
```

RÉFÉRENCES CROISÉES

- Produire une liste des symboles et leurs références dans les différentes unités d'Ada
- Utilisation:

  axref *Unité*

DÉBOGUEUR SYMBOLIQUE

- Les débogueurs symboliques permettent de monitorer et de contrôler l'exécution du programme au niveau du langage source
- Le déboguage d'Ada peut être compliqué par la programmation concurrente, parce que les bogues pourraient ne pas être reproductibles

ÉCOLE POLYTECHNIQUE MONTRÉAL

COMPILATEURS CROISÉS ET SYSTÈMES EMBARQUÉS

- Les compilateurs Ada et les outils de développement peuvent être trop complexes pour être installés et exécutés dans les systèmes cibles embarqués.
- Il existe des compilateurs croisés (*cross-compiler*), et des simulateurs de systèmes cibles pour développer et tester sans le système cible