# LEWIS Arnaud 23 Juin 2012 MLUDEK Denis STEYER Valentin TERRIER Hélène

# PROJET ADA :

# RAPPORT

## U:\Documents\universite-de-lorraine.jpgSujet 1 : Réalisation d’un mini SGBD

Sommaire

[Introduction 3](#_Toc328186099)

[I. Découpage du travail adopté 4](#_Toc328186100)

[II. Analyse de chaque partie 5](#_Toc328186101)

[Groupe de travail : Denis-Valentin (Gestion des données) 5](#_Toc328186102)

[Groupe de travail : Arnaud-Hélène (Gestion des tables & attributs) 7](#_Toc328186103)

[Groupe de travail : Arnaud-Denis (Gestion des fichiers & de la mémoire) 9](#_Toc328186104)

[III. Problèmes majeurs rencontrés 10](#_Toc328186105)

[Conclusion 11](#_Toc328186106)

[Annexe : Notice d’utilisation du logiciel. 12](#_Toc328186107)

# Introduction

Dans le cadre du projet ADA à réaliser par les étudiants de première année de l'IUT Informatique de Metz, il nous a été demandé de créer un mini SGBD (Système de Gestion de Bases de Données) permettant la création et la gestion de petites bases de données. En effet, cet outil devra permettre à un utilisateur qui peut être un débutant en informatique, de créer, de saisir, ou encore de mettre à jour et d'afficher des tables ou des relations d'une base.

Nous allons vous présenter dans ce rapport la structure de notre projet, les différentes fonctions utilisées pour gérer les listes, c'est à dire les fonctions concernant les tables, les attributs et les données, ainsi que le main. Puis les fonctions permettant la gestion des fichiers.

Enfin, nous conclurons sur l'organisation de notre travail tout en donnant un avis collectif sur ce que nous a apporté ce projet. En annexe, un mode d'emploi simple et compréhensible par tous sera mis à disposition, afin d'assurer une utilisation optimale de notre programme

# Découpage du travail adopté

Le choix du sujet ainsi que la définition des structures et la répartition du travail ont fait l'objet d'un travail commun. Cependant, depuis la réalisation du rapport intermédiaire, nous avons dû changer toute notre structure car elle n'était pas optimale. Nous avons donc opté pour des structures avec des listes imbriquées. Les listes ayant l'avantage d'avoir la taille que l'on souhaite, sans jamais la contraindre. Ce que les tableaux ne nous permettaient pas.

Par la suite, notre groupe étant composé de 4 personnes de niveau différent, nous avons voulu faire des groupes assez homogènes pour ne pénaliser personne. C'est pourquoi nous avons choisi cette composition : Arnaud-Hélène & Denis-Valentin.

L'attribution des tâches a été la suivante : le premier groupe s'occupe de la gestion des tables et des attributs, tandis que l'autre s'occupe de la gestion des données (n-uplets).Si un problème survenait, nous nous aidions mutuellement.

Une fois la gestion des listes effectuée, nous nous sommes penchés sur la sauvegarde à l'aide de fichiers séquentiels. Face à ce problème, nous avons décidé de changer les groupes de travail afin de ne pas perdre de temps. L'attribution a donc été la suivante : Denis-Arnaud s'occupaient de la gestion des fichiers et de la mémoire pendant que Hélène-Valentin travaillaient sur le programme principal.

# Analyse de chaque partie

### Groupe de travail : Denis-Valentin Gestion des données

procedure Ajout\_Fin (F:in out access\_donnees; donnees : integer);  
procedure Ajout\_Fin (F:in out access\_donnees; donnees : character);  
procedure Ajout\_Fin (F:in out access\_donnees; donnees : T\_string; taille:integer);  
procedure Ajout\_Fin (Fi:in out access\_donnees; donnees : float);  
procedure Ajout\_Fin (F:in out access\_donnees; donnees : boolean);  
procedure Ajout\_fin (F:in out access\_donnees; donnees:T\_date);

Nous avons décidé de surcharger toutes nos fonctions d'ajout pour plus de facilité d'utilisation.  
Ainsi, lors de l'appel de la fonction ce sont les types des paramètres qui vont permettre à l'ordinateur de savoir quelle fonction utiliser.  
Pour chaque ajout, on alloue donc la place mémoire en fonction du type mutant (type de l'attribut) en remplissant la structure avec la donnée voulue. Puis on met cette structure à la fin de liste donnée en paramètre.

procedure Supprimer\_donnees (liste:access\_table; table\_cible:in out access\_table);

Pour la suppression de données, on affiche d'abord la table voulue parmi la liste des tables et on demande alors combien de n-uplets l'utilisateur veut supprimer. On choisit ensuite chaque numéro de n-uplets à supprimer. Lors de chaque suppression, une nouvelle liste est créée sans l'élément choisi et l'espace mémoire occupé par l'ancienne liste est libéré.  
Le paramètre table\_cible nous permet de ne pas avoir à sélectionner une table si cela a déjà été fait avant.

procedure affiche\_donnees (liste\_table :access\_table; table\_cible: in out access\_table);

Pour l'affichage des données, nous avons essayé de réaliser un affichage par n-uplets qui se rapproche des logiciels existants. Nous avons donc opté pour un affichage à l'aide d'un tableau à deux dimensions de taille dynamique contenant des chaînes de caractères.  
C'est pourquoi il nous a fallu passer par deux étapes : le remplissage d'un seul tableau puis son affichage en plusieurs tableaux afin de pouvoir afficher correctement des tables contenant beaucoup d'attributs. Nous avons limité l'affichage à 3 attributs par tableau afin de rendre visible le programme sur Windows (la taille de la console étant limitée) comme sur Linux.  
Lors du remplissage, nous avons donc dû convertir chaque donnée en type String.

procedure edit\_donnees (liste\_table: in out access\_table);

Cette procedure permet d'ajouter des n-uplets dans une table. On choisit le nombre de n-uplets à saisir puis on les remplit un par un, attribut par attribut. Le type de l'attribut est chaque fois rappelé entre crochets.  
La table est affichée une fois la saisie finie.

procedure detruire\_liste\_do (liste:in out access\_donnees);

Cette fonction parcourt la liste de données jusqu'à la fin, sauvegarde l'adresse de l'élément précédent, passe à l'élément suivant puis libère la mémoire de l'adresse sauvegardée.

### Groupe de travail : Arnaud-Hélène Gestion des tables & attributs

function Liste\_Vide (F:access\_table) return boolean;  
 function Liste\_Vide (F:access\_attribut) return boolean;  
 function Liste\_Vide (F:access\_donnees) return boolean;

La fonction liste\_vide est une fonction surchargée. Elle est utilisée pour les tables, les attributs et les données. La fonction retournera un booléen, c'est à dire vrai ou faux. Ici, nous avons choisi de retourner vrai si le pointeur F pointe sur NULL, c'est à dire si la liste des attributs est vide.

function Queue (F:access\_table) return access\_table;  
function Queue (F:access\_attribut) return access\_attribut;  
function Queue (F:access\_donnees) return access\_donnees;

La procédure Queue est une procédure avec surcharge qui permet d’accéder au dernier élément de la liste. On l’utilise pour les tables, les attributs et les données.

function nb\_elem (liste\_tab:in access\_table) return integer;  
function nb\_elem (liste\_att:in access\_attribut) return integer;  
function nb\_elem (liste\_do:in access\_donnees) return integer;

Cette fonction est une fonction surchargée. Elle permet de retourner le nombre d’éléments que contient une liste.

procedure Ajout\_Fin (F:in out access\_table; nom:T\_string; taille : integer ;contenu:Access\_Attribut);  
procedure Ajout\_Fin(F:in out access\_attribut; nom:T\_string; taille : integer ; type\_attribut:T\_enum; cle:T\_cle;contenu:Access\_Donnees);

C’est une fonction surchargée. Elle permet de créer un nouvel élément et de l’ajouter en fin de liste grâce à une allocation dynamique.

procedure Ajout\_en\_tete (F:in out access\_attribut; nom:T\_string; taille : integer ; type\_attribut: T\_enum ; cle : T\_Cle; contenu : Access\_Donnees);

Elle permet de créer un nouvel élément et de l’ajouter en début de liste grâce à une allocation dynamique. Elle est utilisée que dans le cas d’une liste d’attributs si celui-ci est clé primaire.

procedure affiche\_table (liste\_table : access\_table) ;

Cette procédure permet d'afficher la liste des tables que contient une base. Elle permet aussi d'attribuer un nombre devant chaque table afin de pouvoir par la suite les sélectionner en entrant leur numéro plutôt que leur nom.

procedure edit\_table (liste:in out access\_table);

Cette procédure permet d'ajouter une ou plusieurs tables dans la base courante et également de contraindre l'utilisateur à entrer un entier compris entre 0 et 20 lorsqu'on lui demande le nombre de tables qu'il souhaite créer.

procedure affiche\_attribut(liste\_table :access\_table; table\_cible:in out access\_table);

Cette procédure permet d’afficher la liste des attributs de la table sélectionnée. Dans le cas où un attribut est une clé primaire ou une clé étrangère alors on le précise à l’affichage à l’aide d’une concaténation.

procedure edit\_attribut(base: access\_table; liste: in out access\_table);

Cette procédure permet d’ajouter un ou plusieurs attributs à la table sélectionnée à l‘ouverture du sous-menu de gestion des attributs. Pour cela, on demande à l’utilisateur le nombre d’attributs qu’il veut ajouter ainsi que leur type, il faudra aussi qu’il précise s’ils sont clé primaire ou clé étrangère ou neutre. Il faut allouer de la mémoire pour chaque attribut à créer. Les nouveaux attributs sont ajoutés en fin de liste afin qu’ils soient dans l’ordre. Dans le cas où l’attribut serait clé primaire alors on l’ajoute en tête de liste.

procedure MAJ(nom:in out T\_string; taille:in integer);  
procedure MAJ\_char(char:in out character);

Ces procédures permettent de convertir une chaîne ou un caractère en majuscule grâce au contrôle de leur position dans la table ASCII.

### Groupe de travail : Arnaud-Denis Gestion des fichiers & de la mémoire

procedure create\_open (Base:in out T\_base; nom\_fichier:in out string; taille:in out integer; sortir:in out boolean);

Fonction qui est exécutée au lancement du programme, elle propose via un menu la création d'une nouvelle base ou l'ouverture d'une base existante. Si, lors de l'ouverture d'une base le fichier n'existe pas, on propose alors à l'utilisateur s'il veut la créer.

procedure open\_base (B:in out T\_Base;nom\_fic:in out string; taille:in out integer);

Cette procédure ouvre un fichier existant (correspondant au nom de la base), puis, "transfère" le tableau (qui était stocké dans le fichier) dans les listes de travail.

procedure Create\_Base (B:out T\_Base; nom\_fichier:in out string; taille:in out integer);

Procédure qui demande à l'utilisateur le nom de la base qu'il veut créer.  
Pour plus de lisibilité ce nom de fichier est ensuite formaté de la manière suivante : "f\_NOM.dat".  
Le fichier est ensuite créé, puis fermé.

procedure save\_base (Ba:in T\_Base; nom\_fic:in string; taille:integer);

Cette procédure permet de sauvegarder la base de données créée dans un fichier séquentiel. Pour cela, elle "transfère" les listes imbriquées dans des tableaux imbriqués. Permettant ainsi la sauvegarde de la base dans le fichier.

procedure del\_base(nom\_fic:string; taille:integer; close:in out boolean);

Procedure qui permet la suppression de la base, c'est à dire du fichier contenant la base.  
Demande la confirmation avant la suppression.

procedure Supprimer (table:in out access\_table);  
procedure Supprimer (liste\_tab:access\_table; table\_cible:in out access\_table);

Cette procédure est une procédure surchargée, elle est utilisée pour les tables et pour les attributs. Elle permet de supprimer une ou plusieurs tables, pour cela on demande à l’utilisateur le nombre de tables qu’il veut supprimer puis les numéros successifs des tables à supprimer. La liste est alors modifiée et on libère la mémoire de l’élément supprimé.

Pour libérer la mémoire c’est la fonction free\_tab qui est employée. Elle va permettre de supprimer toutes les données et tous les attributs relatifs à la table sectionnée.

# Problèmes majeurs rencontrés

Voici les différents obstacles auxquels nous avons fait face pendant la réalisation du projet.

* Le problème le plus récurrent concerne l'utilisation des types String. Notamment lors de la saisie du nom de la base (f\_NOM.dat) puis lors de l'affichage des données. Procedure au cours de laquelle il a fallu convertir chaque type en String afin de les afficher correctement.  
  Le cas de l'affichage des réels fut aussi intéressant : passage de l'écriture scientifique à l'écriture normale.
* La sauvegarde et l'ouverture de la base nous a également donné du fil à retordre. En effet, nous nous sommes rendu compte qu'il n'était pas possible de stocker nos listes imbriquées dans le fichier séquentiel : stocker les adresses mémoires des éléments suivant de chaque liste posait problème.  
  Pour pallier à ce problème, nous avons recréé les mêmes structures en substituant les listes par les tableaux.

# Conclusion

Ce projet ADA nous a permis d'avoir, une fois de plus, un aperçu de ce qu'était le travail d'équipe. Il nous a permis d'approfondir le fonctionnement des listes et la gestion des fichiers.

Nous avons constaté que tout le programme vu en ADA durant l'année a eu son utilité pour réussir à mener ce projet à bien, même s'il a fallu faire des recherches personnelles.  
Ce projet fut enrichissant en tout point car il nous a permis de faire un lien avec les autres matières vues au courant de l'année : Base de Données, Modélisation des Systèmes d'Informations ou encore Delphi. Ainsi, nous avons compris que lors de la réalisation d'un projet, nous devons avoir des connaissances supplémentaires hormis la programmation en elle-même afin d'être apte à développer dans n'importe quel domaine.

Avant de commencer à programmer, nous nous sommes vus plusieurs fois pour réfléchir ensemble à la structure principale de notre projet afin que tout le monde puisse apporter une idée.

Notre groupe n'ayant pas un niveau homogène, nous avons dû nous entraider afin que personne ne se retrouve lésé. C'est pourquoi nous avons décidé de faire deux groupes équilibrés : Arnaud-Hélène & Denis-Valentin. Puis nous avons changé la composition des groupes lorsque le problème des fichiers s'est présenté de sorte à trouver une solution rapidement.

En conclusion, nous pensons que ce projet est celui qui nous a le plus apporté durant l'année en terme de connaissances, de recherches personnelles à effectuer et d'organisation.

# Annexe : Notice d’utilisation du logiciel.