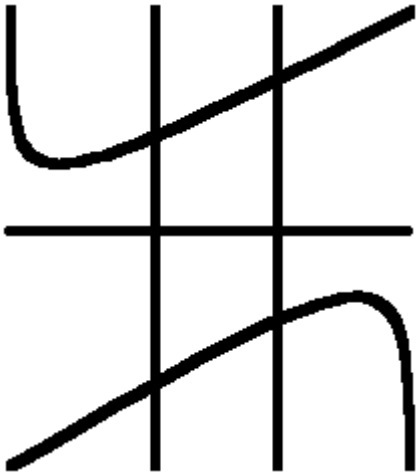
Info-SPE : A_1–A_2, Promo 2007

SIMTADYN

Quatrième soutenance



Quadrat Quentin Nguyen Minh-Long Marcot Benoit

SimTaDyn

Quadrat Quentin Nguyen Minh-Long Marcot Benoit

28 juin 2004

Table des matières

1 Introduction à SimTaDyn

Un tableur est un éditeur de feuilles de calcul, à savoir des tables de cellules formant un pavage rectangulaire. Chaque cellule contient des données de type divers y compris des fonctions susceptibles de faire du calcul sur son contenu et celui des autres cellules.

Un Système d'Information Géographiques (SIG) est un autre type d'éditeur manipulant des tables contenant des champs ayant des interprétations géographiques. Elles sont représentées par des cartes permettant d'accéder facilement aux champs n'ayant pas d'interprétations géographiques. De plus, les SIG ne contiennent pas de fonctions susceptibles d'agir sur la table donc sur la carte. L'accés aux champs non géographiques est obtenu en pointant sur un objet particulier. Seuls les champs de l'objet désigné sont affichés contrairement aux tableurs qui essaient toujours de montrer l'ensemble des données. Par contre, leurs outils d'édition de cartes sont très puissants.

Les fonctionnalités des tableurs et des SIG ont donc toujours été dissociées. SimTaDyn est un prototype de synthèse de ces deux types de fonctionnalités. Il manipule des tables dynamiques c.à.d. des tables pouvant contenir en plus des données standards des fonctions capables de modifier la table elle-même. Il représente les données géographiques en positionnant les objets correspondants dans le plan. Il essaie de viualiser simultanément tous les objets de même type.

SimTaDyn, à la différence des tableurs standards, ne possède plus forcément un pavage de cellules rectangulaires mais un pavage polygonal quelconque du plan. De plus, les frontières des pavés — qui sont des lignes brisées donc, finalement, une union de segments — sont considérées comme des cellules à part entière. De même, les bords des segments sont des sommets eux mêmes considérés comme des cellules à part entière. Donc finalement, SimTaDyn possède trois types de cellules : *les sommets, les segments et les pavés.*

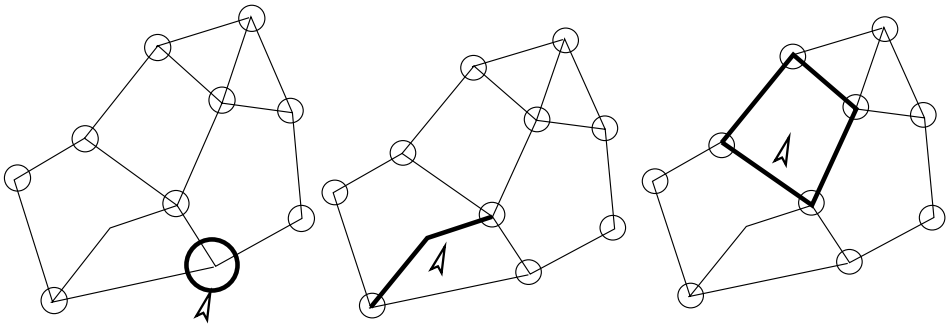


Figure a : un noeud Figure b : un arc Figure c : une zone

Fig. 1 – Sélection des différents types de cellules.

Les cellules contiennent aussi plusieurs champs, dont certains peuvent être des fonctions mathématiques capables de modifier les données des autres cellules.

SimTaDyn peut être utilisé à des fins trés variées, comme par exemple, représenter l'évolution de surfaces, résoudre des équations de dérivées partielles, créer des jeux de startégie, résoudre des problèmes de graphe, etc.

Définitions :

Table

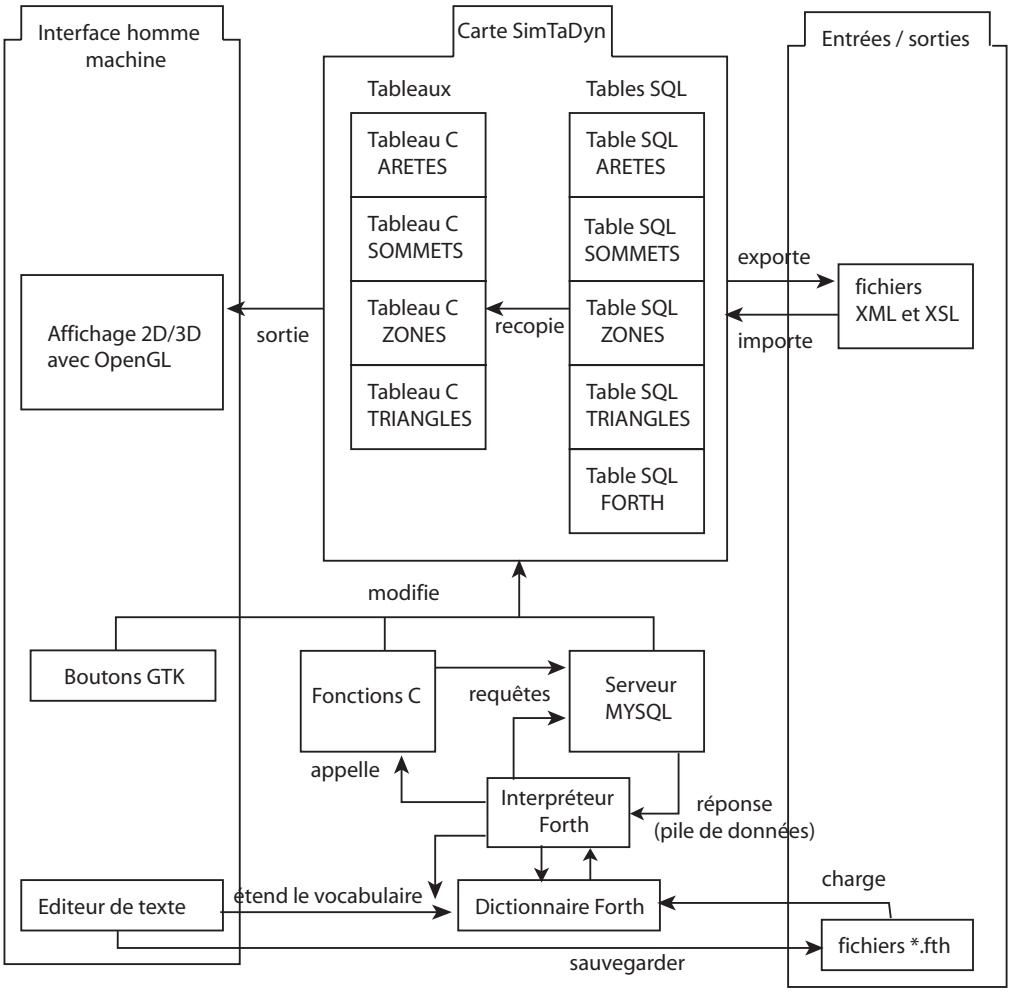


Fig. 2 – Squelette de SimTaDyn

Cartes
Sommets
Segments
Pavés
Champs
Calque

2 L'interface utilisateur

2.1 La fenêtre principale

SimTaDyn a été implémenté en C, MySQL, OpenGL et GTK. Le langage GTK a permis de réaliser une interface complexe. C et MySQL ont été utiles afin de définir et de gérer la structure de données des cellules de la carte — manipulation, accés, sauvegarde, ... —. OpenGL a permis le rendu des cellules à l'écran en 2D et en 3D.

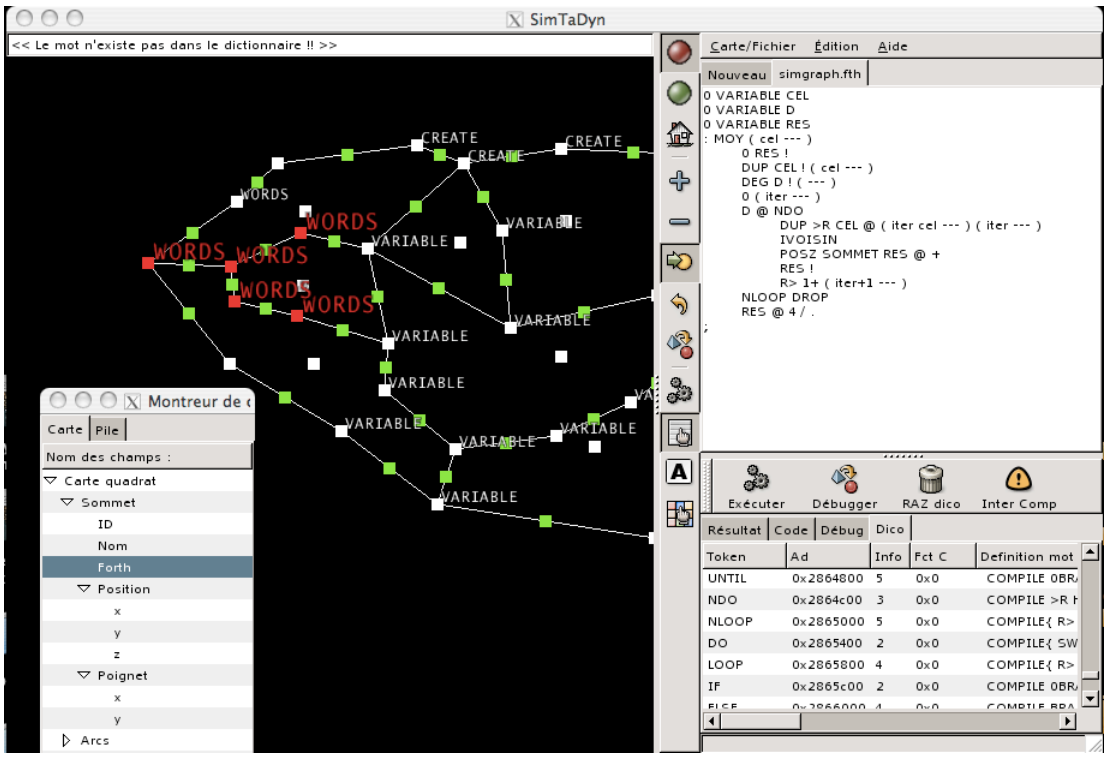


Fig. 3 – Apercu de la fenêtre principale de SimTaDyn

L'utilitaire GtkGlExt a permis de gérer OpenGL dans une fenêtre GTK. L'interface de SimTaDyn se décompose en deux parties. La partie sombre de la figure 3 et la barre verticale de boutons s'appelle SimGraph. La partie gauche s'appelle SimForth.

2.2 La fenêtre SimGraph

Les objets contenus dans la fenêtre.

2.3 La fenêtre de SimForth

L'interface SimForth est une application GTK+2 conposée de trois parties :
- la première partie est un notebook (terme GTK) qui contient par défaut un tampon d'entrée qui est un éditeur de texte multiligne permettant d'entrer et d'éditer les programmes Forth à executer.
- La deuxième partie est une rangée de boutons : — Exécuter ou compiler des commandes Forth — Activer ou désactiver le débogueur — Remise à zéro du dictionnaire — Remise à zéro de la pile.
- Une troisième partie est un autre notebook avec quatres onglets chacun contenant une zone de texte pour afficher des informations de l'interpréteur SimForth : — résultat d'un programme SimForth — historique des anciens commandes SimForth exécutées ou compilées — informations du débugger — contenu du dictionnaire SimForth.

Fig. 4 – Apercu de l'éditeur Forth de SimTaDyn

SimForth attend que l'utilisateur entre des commandes pour pouvoir les exécuter. Lorsque l'utilisateur veut que SimForth fasse l'action souhaitée, il doit entrer au clavier la suite de commande dans le tampon d'entrée, puis cliquer sur le bouton *exécuter* ou en appuyant en même temps les touches contrôle et entrée. SimForth interprète alors la totalité du tampon. Le code disparait et est stocké dans l'historique. L'utilisateur a la possibilté d'exécuter une sous-partie du code en le selectionnant à la souris. L'utilisateur a la possiblité de récupérer son ancien code à tout moment par simple glisser–déposer ou copier–coller entre les diff zones.

Chaque nouveau mot est stocké en mémoire dans le dictionnaire. Un tableau GTK permet de visualiser tous les mots du dictionnaire, ainsi que leur définition. Cette fenêtre permet de trier les noms des mots en ordre croissant — décroissant.

Un débuggeur peut être activé. Pour cela il suffit d'enfoncer le bouton *débuggeur*. Tant que le débuggeur est actif, il affiche l'exécution pas à pas des mots Forth stockés dans le tampon d'entrée.

L'utilasteur peut ouvrir des fichiers texte ASCII, via un widget spécialisé GTK, appelé grâce au menu situé en haut de la fenêtre. Le premier notebook crée alors un nouvel onglet contenant une autre zone de texte éditable.

2.4 L'éditeur ligne des champs

2.5 Le visualisateur de la pile de données

2.6 Les outils de maniplulation standard

2.7 Le boutons avancés

3 SimForth

Les boutons de l'interface homme-machine ont un rôle important pour la création des cartes. Ils permettent l'exécution des fonctions, sans paramètres, prédéfinies dansSimTaDyn mais ils ne sont qu'en nombre fini et donc n'ont pas toute la puissance d'expression d'un véritable langage bien qu'ils soient très faciles à utiliser. C'et pourquoi on a décidé de doter Simtadyn d'un véritable interpréteur. On a choisi le langage Forth pour sa puissance d'expression et sa vitesse d'interprétation bien que sa syntaxe soit plus difficile à utiliser que celle des langages comme Basic ou C. L'interpréteur est écit en C. Il est trés facile d'interfacer dans ce langage les bibliothèque C standard. Appeler un opérateur de ce langage revient souvent à appeler une fonction C. Un programme peut être vu alors comme un script d'appels de fonctions C. Enfin le caractère interpreté du langage permet d'eviter les phases de compilation et donc améliore l'interactivité par rapport à un langage compilé.

Pour le distinguer des Forths standards (c'est à dire : GForth, PForth, MacForth [7], [5], etc) on l'appellera SimForth. SimForth est donc un langage inspiré des Forth standards mais plus adapté à la structure de données des cartes, définies dans la section **??**. Par conséquent, certains mots du Forth standards auront disparus au profit d'autres ne fonctionnant qu'avec SimForth. Enfin, SimForth gérera et fera évoluer la base de données qui en plus de stocker des valeurs, contiendra des mots Forth donc des programmes. Le programmeur pourra créer son propre vocabulaire qui rendra la base encore plus dynamique.

L'avantage de relier des mots à des fonctions, est que, si un programme SimForth se révéle trop lent, il peut être recompiler en C avec de nouvelles librairies ou bibliothéques. Le nouveau programme peut voir sa vitesse augmenter d'un facteur 30, où 30 est le rapport de vitesse constaté lors de l'exécution d'un même algorithme l'un écrit en SimForth, l'autre en C compilé puis exécuté. Un programme interprété est toujours plus lent que ce même programme compilé.

3.1 Le Forth de base

Pour pouvoir programmer avec SimTaDyn, l'utilisateur doit avoir au moins lu cette section. La section **??** est utile pour apprendre les derniers engrenages du fonctionnement d'un Forth standard — notamment la création de mots de structurations et de structures —. La section **??** résume de quelles façons a été implémenté SimForth en C.

3.1.1 Fonctionnement général d'un Forth

SimForth — comme les Forth standards — est un langage à pile dans lequel on définit des opérateurs, appelés mots, agissant sur la pile. Un programme Forth est donc une suite de mots gérant des données stockées dans une pile. Les mots ayant un trait commun sont regroupés en vocabulaire. La pile stocke trois données différentes : entiers, réels et adresses. L'utilisateur agit en permanence sur cette pile en permanence.

SimForth a deux modes de fonctionnement, un mode compilation un mode exécution.

- Le mode exécution modifie la pile de données en empilant des données au fur et à mesure de leur rencontre et en exécutant les mots qui consomment les opérandes au sommet de la pile et qui y restockent les résultats. Par exemple, on empile 1 et 2 et on exécute le mot `+` qui va modifier le contenu de la pile.

– Le mode compilation permet de définir de nouveaux mots à partir de ceux déjà connus de la machine. On donne donc un nom à une définition (une suite de mots). Un programme Forth consiste donc à construire et à étendre un vocabulaire. Par exemple, on peut vouloir définir un repas comme la succession d'une entrée, d'un plat et d'un dessert.

Forth utilise la notation polonaise inversée qui simplifie l'analyse syntaxique, puisque elle élimine le paranthésage. expression $((2+5)\ (7-2))\ /\ ((5+2)\ 3)$ se traduit en Forth `2 5 + 7 2 - * 5 2 + 3 * /`

3.1.2 Le dictionnaire Forth

Le dictionnaire est le coeur du Forth. Il contient toutes les mots que comprend SimForth. A chaque fois qu'un nouveau mot est ajouté, SimForth le stocke dans le dictionnaire. L'utilisateur étend son langage jusqu'à accomplissement de tous les aspects de son application.

Les mots ajoutés au dictionnaire restent présents aussi longtemps que l'ordinateur ou SimTaDyn fonctionnent. A chaque redémarage de SimTaDyn un dictionnaire de base (minimal) est généré et toutes les commandes créées lors des sessions précédentes doivent être redéfinies. C'est pour cette raison que SimForth est capable de lire et d'interpréter des fichiers ASCII définissant des anciennes commandes.

3.1.3 Syntaxe et conventions

L'utilisateur entre ses programmes soit en chargeant un fichier ASCII, soit en les tapant dans l' éditeur de texte (cf. section **??**).

Un programme Forth est défini par une suite de mots Forth. Un mot Forth est une chaîne de caractère chiffre ou caractère spéciaux à l'exception des parenthèses ouvrantes et fermantes. Les mots Forth sont séparés par un espace, une tabulation ou un retour chariot. Les minuscules ne sont pas distingués des majuscules. L'interpréteur reconnait deux types de nombres. Les entiers (125) et les décimaux (12.678). Toute chaîne de cractère entre parenthèses dans un programme sera considéré comme un commentaire.

Pour expliquer l'action d'un mot Forth sur la pile de donńes on utilise les conventions suivantes. 0n figure entre parenthèses et séparés par un tiret les paramètres de la pile de donnée, utilisés par le mot, et ceux éventuellement retournés aprés exécution.

Par exemple

```
+ ( 1  2  ---  3 )
```

signifie qu'avant d'exécuter la division, le nombre `1` et `2` ont été empilés dans la pile de données et qu'après execution du mot "`+`" 1 et 2 ont été dépilés et que le résultat de l'addition `3` a été retourné dans la pile.

```
. ( n ---  )
```

signifie et que le mot `.` dépile un entier. En fait, il l'affiche à l'écran puis le supprime.

On exécute ces programmes, simplement en tapant dans le tampon d'entrée `1 2 + .` puis en cliquant sur le bouton exécuter vue section **??**.

3.1.4 Les mots manipulant la pile de données et le dictionnaire

Il existe six mots importants gérant la pile de données.

– `DUP` (n — $n\ n$) duplique le sommet de la pile.

– `DROP` (n —) detruit le sommet de la pile.
– `.` (n —) detruit et affiche le sommet de la pile.
– `SWAP` (n_1 n_2 — n_2 n_1) commute les deux dernières entrées.
– `OVER` (n_1 n_2 — n_1 n_2 n_1) duplique l'avant dernière entrée sur le sommet de la pile.
– `ROT` (n_1 n_2 n_3 — n_2 n_3 n_1) permutte les trois dernières données.

La table de hachage — le dictionnaire, donc — est manipulée par quatre mots Forth :
– `SVGDE` sauve le nouveau mot Forth créé par `BUILDS` dans la table,
– `SAVE` agit comme `SVGDE` et demande en plus à GTK d'afficher la définition du dernier mot créé dans la fenêtre qui montre le contenu du dictionnaire (cf. section **??**),
– `FORGET` detruit le mot qui le suit,
– `BUILDS` .
– `COMP` .
– `EXEC` .

Pour rappel, on peut visualiser le contenu de la pile de données et du dictionnaire via les fenêtres spécialisée de GTK (cf. section **??**).

3.1.5 Les mots créant de nouvelles définitions Forth

Toute définition d'un nouveau mot commence par le mot `:` suivi du nom du mot que l'on définit et se termine par le mot `;` . Pour qu'une définition soit valide il faut que tous les mots soient connus de l'interpréteur (donc être présents dans le dictionnaire) sauf celui qui suit l'on définit .

Supposons que les mots `AILE BLANC CUISSE FRITE ENTREE DESSERT` soient des mots déjà connus, voici un programme créant un repas (exemple tiré de [5]) :

```
: POULET AILE BLANC CUISSE ;
: PLAT POULET FRITE ;
: REPAS ENTREE PLAT DESSERT ;
```

On peut donner une définition vide à un mot. Par exemple, si on veut créer un programme permettant de françiser la syntaxe Forth, par exemple les mots suivants (déjà connus du dictionnaire minimal) `+` pour l'addition et `.` pour afficher à l'écran un entier.

```
: SOIENT_LES _DEUX_NOMBRES ;
: ET ;
: LEUR_SOMME_VAUT +  .  ;
```

Pour voir le résultat de la somme de deux nombres il suffit d'écrire :

```
SOIENT_LES_DEUX_NOMBRES 3 ET 5 LEUR_SOMME_VAUT
```

Le mot `:` joue un double rôle. Il fait passer l'interpréteur en mode compilation et crée un nouveau mot avec le nom du mot qui suit mais ne le stocke pas immédiatement dans le dictionnaire. Le mot `;` bascule en même temps l'interpréteur en mode exécution et entre le nouveau mot dans le dictionnaire.

Cette façon de faire, non classique, permet d'éviter d'entrer des mots mal formés dans la table. Car il faudrait soit – utiliser un algorithme de suppression d'élements dans le dictionnaire (qui serait plus coûteux en temps que de supprimer un pointeur temporaire) – soit de mettre un jeton `SMUDGE` sur le mot pour indiquer à l'interpréteur que sa définition n'est pas valide (le Forth de [5] applique cette méthode).

L'autre raison est d'éviter de faire des algorithmes récursifs par erreur. En effet dans l'exemple suivant :

`: TUTU 1 2 + TUTU ;` (pas bon !)

l'interpréteur refuse de compiler le programme parce qu'il recherche la presence du deuxième `TUTU` dans le dictionnaire. Or, ici, la définition du premier `TUTU` n'a pas été encore stockée dans la mémoire du dictionnaire.

Une méthode pour résoudre ce problème et d'utiliser respectivement les mots `:REC` et `;REC` au lieu de `:` et `;` .

`:REC TUTU 1 2 + TUTU ;REC` (correct !)

En effet, `;REC` force l'interpréteur à stocker le premier `TUTU` dans le dictionnaire. Finalement, quand il recherche le deuxième `TUTU` il trouve l'adresse du premier.

La définition de la factorielle est la suivante ;

```
:REC FACT
    DUP
    IF
        DUP 1- FACT *
    ELSE
        DROP 1
    ENDIF
;REC
```

On peut supprimer un mot connu du dictionnaire grâce au mot `FORGET` suivit du mot à oublier. Ce dernier devient inutilisable et ne peut plus être appeler dans une définition, sauf si, avant, on lui redonne une nouvelle définition.

3.1.6 Les mots créant de nouvelles données

Pour lire le contenu d'une cellule mémoire, il suffit de placer son adresse sur la pile et d'exécuter le mot `@` (*adr* — *n*). Inverssement, pour modifier la valeur d'une cellule mémoire, il suffit de placer la nouvelle valeur puis l'adresse de la cellule dans la pile, et d'exécuter le mot `!` (*n adr* —).

Il existe deux mots Forth de bases servant à manipuler les données : les variables et les constantes. La valeur d'une constante ne peut pas être modifiée, contrairement à celle d'une variable. Pour déclarer et initialiser une variable, on écrit la valeur puis le mot `VARIABLE` suivit du nom de la variable. Idem pour les constantes avec le mot `CONSTANT`.

```
212 VARIABLE AGENT
13  CONSTANT XIII
```

Pour stocker la valeur d'une constante sur la pile, il suffit simplement d'exécuter le nom de la constante. Pour le cas d'une variable, il faut exécuter le nom de la variable et le mot `@`. En effet, l'appel du nom de la variable, place l'adresse de la valeur sur le sommet de la pile de donnnées.

```
: ? @ . ;
212 VARIABLE AGENT ( modifie la valeur de la variable )
AGENT ? ( affiche la valeur de la variable )
```

Pour modifier la valeur d'une variable, on utilise le mot `!` .

```
22 AGENT !
AGENT ?
```

Une autre alternative pour créer des constantes est de compiler un un nombre à un mot :

```
: XIII 13 ;
```

Mais il y a une nuance importante entre ces deux définitions. Pour comprendre la subtilité, il faut connaître le fonctionnent des mots `BUILDS` et `DOES` — dont le mot les mots `CONSTANT` et `VARIABLE` utilisent (confère section 4.6) — ainsi que le fonctionnement du mot `LITERAL` (expliquer dans la section 4.3).

3.1.7 Les mots créant des chaines de caractères

La création d'une chaine de carctère se fait part l'appel du mot Forth `."` . La fin du texte est signalée par le mot `"` . Il doit toujours y avoir un espace entre le premier caractère de la chaine et le premier guillemet. Par contre il peut ne pas y avoir d'espace entre le dernier caractère et le dernier guillemet. Si l'interpréteur est en mode exécution l'adresse de la chaine de caractère est directement mise sur la pile. Pour afficher la chaine il suffit d'exécuter le mot `.` mais la chaine ne pourra plus être utilisée.

```
." Nouvelle Chaine !"
. ( affiche la chaine )
```

Les chaines peuvent être utiliser dans une définition.

```
: TUTU
    ." Nouvelle Chaine !"
    .
;
```

On peut concaténer deux chaines par le mot `CONCAT` (adr_1 adr_2 — adr de chaine$_2$+chaine$_1$) l'adresse de la nouvelle chaine est stockée sur la pile. Les deux anciennes chaines sont libérées.

Il est possible de transformer un entier ou un réel en une chaine de caractère avec le mot `>STRING` (n — adr), l'adresse de la nouvelle chaine est placée sur le sommet de la pile.

3.1.8 La structure conditionelle `IF ... THEN ... ELSE`

La syntaxe classique `IF` *condition* `THEN` *traitement 1* `ELSE` *traitement 2* des langages traditionnels, tels que le C, aura pour équivalent en Forth *condition* `IF` *traitement 1* `ELSE` *traitement 2* `THEN` où le *traitement 1* sera executé si la *condition* est vraie, sinon se sera le *traitement 2*.

Par exemple :

```
: INTERVAL
    OVER >
    IF
        <
    ELSE
        2DROP
        0
    THEN ;
```

On peut remplacer `ENDIF` par `FI` ou `THEN`. Question de goût ! Il suffit de taper :
`: ENDIF THEN ;`

3.1.9 La structure répétitive `DO ... LOOP`

La structure répétitive `DO` ... `LOOP` a aussi été implémentée. L'action des mots sur la pile de données s'écrit :

```
DO ( IndFin+1 IndDebut --- )
LOOP ( --- )
```

`DO` empile dans la pile de retour l'indice de fin + 1, ansi que l'indice de début, puis enchaîne l'exécution des mots du traitement à répéter.

`LOOP` compare les deux valeurs du sommet de la pile de retour. Si le dernier est inférieur à l'avant dernier, `LOOP` incrémente le sommet de 1 et transfère l'exécution du mot qui suit immédiatement `DO`. Dans le cas contraire, les deux valeurs du sommet de la pile de retour sont supprimées et l'exécution est passée au premier mot suivant `LOOP` [5].

Exemple :

```
: GRILLE
    3 0 DO
        3 0 DO
            ( ... )
        LOOP
        CR        ( retour chariot )
    LOOP ;
```

Les mots `NDO` et `NLOOP` jouent le même rôle que `DO` et `LOOP` sauf que l'on n'a plus besoin d'écrire 0 entre le nombre d'itération et le `DO` . Donc `3 0 DO` s'écrit `3 NDO`. Par contre il ne faut pas se tromper avec `LOOP` et `NLOOP`, car se sont deux mots différents. Enfin, `NDO` s'execute plus rapidement que `DO`.

Une remarque importante est que Forth exécute au moins une fois la boucle, c.à.d que le code suivant s'execute qu'une seule fois :

```
: GRILLE
    0 0 DO     ( ou bien 0 NDO )
        ( ... )
    LOOP ;     ( ou bien NLOOP )
```

3.1.10 La boucle indéfinie `BEGIN ... UNTIL`

Le débranchement se fait au niveau de `UNTIL`, uniquement si le l'entier au sommet de la pile a une valeur 1. Dans le cas contraire, l'exécution est transférée au mot suivant `BEGIN`. Le traitement est effectué au moins une fois. Le débranchement de la boucle indéfinie `BEGIN` ... `UNTIL`

3.1.11 Mots géographiques

Il existe un certain nombre de mots SimForth permettant de manipuler la carte SimTaDyn

Chaque structure des différents types de cellules d'une carte SimTaDyn est à la fois stockée dans un tableau C et une table MySQL. Pour connaître le numéro d'identité MySQL à partir de la ième case du tableau C, il suffit d'empiler le numéro de la case du tableau et d'appeler le mot `>IDS` ou `>IDA` ou `>IDZ` selon qu'on est respectivement en mode *sommet*, *arc* ou *zone*.

Pour savoir si un ID est associé à une cellule on execute le mot `EXISTS` . SimForth empile un flag d'existence de la cellule.

Tous les boutons manipulant la carte vus section **??** peuvent être appellés avec un mot Forth. L'ajout d'un sommet se fait en empilant la position y puis x du sommet et d'exécuter `S+` . L'ajout d'un arc s'effectue en stockant les numéros ID des deux sommets suivit de `A+` . La suppression d'un sommet ou d'un arc se fait en appelant le numéro ID suivit de `S-` ou de `A-` .

L'accés ou la modification des valeurs des champs des cellules, s'effectue de la même manière qu'une variable. Les champs SimForth des cellules se comporte à la fois comme un tableau et une variable,

Pour accéder aux addresses des valeurs des champs il suffit de donner l'ID de la cellule suivit du nom du champ et du nom du type de cellule. Le nom du type de cellule est soit `SOMMET` soit `ARC` soit `ZONE` . Actuellement, les noms des champs Forth d'un sommets sont : `ID NOM FORTH POSITION, POSX POSY POSZ`

Par exemple l'exécution de `42 ID SOMMET @` empile 42.

Pour modifer les valeurs des champs il suffit d'empiler la nouvelle donnée et donner l'ID de la cellule suivit du nom du champ et du nom du type de cellule et du mot `!`.

Par exemple l'exécution de `100.0 42 POSZ SOMMET !` modifie l'altitude du sommet 42.

Toutes modifications du tableau C ne sont pas enregistrées dans la table MYSQL. Pour cela il faut appeler le mot `>TABLE` .

3.1.12 Lancer des requêtes MySQL

L'utilisateur peut intérroger la base de données où sont sauvegardées les cartes SimTaDyn via des requêtes MySQL. Une requête MySQL est une chaine de caractères envoyées au serveur grâce au mot `MYSQL`.

Elle peut être soit être exécutée immédiatement.

```
." UPDATE som set coord_z = 50 WHERE id = 47"
MYSQL
```

Ou bien, directement compilée dans une définition Forth :

```
: MYREQUETE
        ." UPDATE som set coord_z = 50 WHERE id = 47"
        MYSQL
;
```

On peut pas passer des paramètres aux requêtes MySQL via la pile de données. Voici un exemple.

```
: %VAL ( recupere un entier ou reel, le transforme en chaine et le
                concatene avec la chaine du sommet )
    >R >STRING R> SWAP CONCAT
;
: MYREQUETE
    ." UPDATE som set coord_z = "
    %VAL
    ." WHERE id = " CONCAT
    %VAL
    MYSQL
;
```

Si l'utilsateur exécute une requête MySQL à base de SELECT, les cellules consernées seront sélectionnées et leurs numéros d'ID seront sempilés dans la pile de données. Le mode *sélection* sera activé et un des modes *sommet*, *arcs*, *zones* aussi.

4 Programmation SimForth avancée

4.1 Mots immédiats

Certains mots peuvent avoir un comportement différent selon que l'interpréteur est en mode compilation ou en mode exécution.

Si un mot est immediat et est présent dans une définition, il sera exécuté à la compilation. Donc, c'est le résultat de son éxécution qui est utilisé lors de la définition d'un autre mot. Par contre, il aura disparu en mode exécution car il ne fait pas partie de la définition.

C'est le mot `IMMEDIATE` qui rend immediat le dernier mot entré dans le dictionnaire.

Par exemple, définissons le mot `COUCOU` :

```
: COUCOU ." COUCOU"  . ; IMMEDIATE
```

Si on utilise ce mot dans la définition suivante :

```
: TEST COUCOU ." COCORICO" ;
```

Lors de la compilation de cette définition, l'interpréteur affiche `COUCOU` et l'exécution de `TEST` affiche `COCORICO` et l'exécution de `COUCOU` affiche `COUCOU`.

4.2 Structure interne des mots Forth

Un mot Forth est formé de un seul bloc mémoire. On peut distinguer deux parties différentes : la première s'appelle l'entête, la seconde le corps. L'entête sert à stocker des informations. Par la suite on l'appelera NF pour Name Field en anglais. Le corps permet de stocker la définition du mot c.à.d une suite de mot Forth. Par la suite on l'appelera CF pour Code Field. Si un mot est déjà présent en mémoire, on peut accéder à tout son contenu en connaissant l'adresse du début du CF. On parlera, par la suite de CFA, pour Code Field Address. De même pour le NF, l'adresse du bloc s'appelera NFA pour Name Field Address.

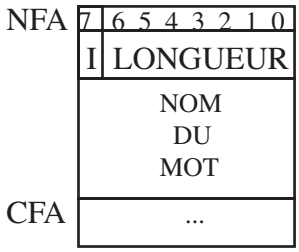


Fig. 5 – Structure interne d'un mot

Le NF du Forth de [5] stocke le nom du mot de moins de 32 caractères, la longueur du nom sur 5 bits ($2^5 = 32$). Le sixième bit (smudge bit en anglais) permet de savoir si la définition mot comporte une erreur. Le septième bit détermine si le mot est un mot immédiat (precedence bit en anglais). La notion de mot immédiat est définie dans la section 4.1. Le huitième est toujours vaut toujours un. Si le smudge d'un mot est activé, l'interpréteur considère ce mot

comme ne faisant plus partie du dictionnaire (même s'il reste toujours présent dans la mémoire). L'interpréteur refuse, alors d'exécuter le mot, même s'il est il appeller (cf. 3.1.5). Le NFA n'est pas immédiatement calculable, puisque les noms de définition sont de longueur variables. Le huitième bit permettait de le retrouver facilement : il suffit de rechercher vers les adresses décroissantes le premier octet dont le MSB (Most Signifiant Bit) est à un.

Le NF d'un mot SimForth ne posséde pas de smudge bit , en effet, quand le mot n'est pas valide, il est directement supprimé de la mémoire (cf. section **??**). Le huitième bit mis à un n'est plus utile car avec les langages récents, on n'a plus besoin de rechercher le MSB. Il servira, dans notre cas de precedence bit.

Le CF est une liste pouvant contenir trois sortes d'informations :
- soit une adresse vers du code exécutable (un pointeur sur une fonction C),
- soit des paramètres (entiers, réels, ou adresses), comme dans le cas des constantes et variables,
- soit efin dans le cas le plus courant, un CFA d'un mot qui le constitue.

Par exemple si on définit :

```
212 VARIABLE AGENT
```

Le mot `AGENT` aura dans le dictionnaire la structure suivante (Fig. 6) :

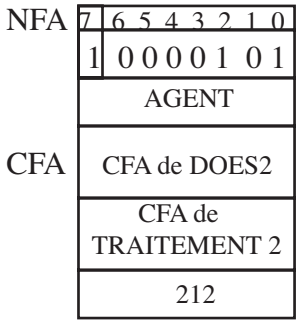


Fig. 6 – Structure interne d'une variable

Par exemple si on définit :

```
: REPAS ENTREE PLAT DESSERT ;
```

Le mot `REAPS` aura dans le dictionnaire la structure suivante (Fig. 7) :

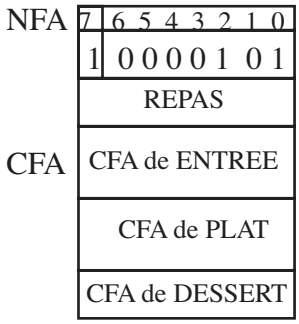


Fig. 7 – Structure interne d'un nouveau mot

4.3 Le fonctionnement interne de l'interpréteur

La compilation

Toute définition commence par le mot `:` . L'interpréteur entre en mode compilation et crée un nouveau mot avec l'expression (token) suivante du tampon d'entrée. Tous les mots du tampon sont compilés jusqu'au mot `;` qui termine la définition du mot courant et remet l'interpréteur en mode exécution. Le mot `;` n'est pas compilé parce qu'il directement exécuté car il fait partie des mots immédiats (cf section 4.1).

Une fois, le corps du mot courant créé, le travail de l'interpréteur consiste à reconnaitre un mot :

- si le mot existe dans le dictionnaire, son CFA est ajouté en fin de liste des CFA du mot courant.
- si le mot n'existe pas, l'interpréteur cherche si son expression correspond à un nombre entier, réel ou adresse. Si c'est le cas il intercalera le mot `LITERAL` en fin de liste de CFA puis le nombre.
- Sinon le mot n'est pas correct.

Peut être, qu'à se stade, vous vous posez la question comment définir le mot `:` ? Peut-on utiliser le mot `:` pour définir le mot `:` ? Comme par exemple :

```
: :
    ( je mets quoi ici ? )
;
```

dire comment j'ai fait.

L'exécution

L'interpréteur va exécuter tous les CFA du mot courant qui à leur tour vont exécuter leur CFA. L'exécution revient à un parcours d'arbre n-aire de CFA.

On veut par exemple exécuter le mot `REPAS` de l'exemple donné à la section **??** construit selon la figure suivante (Fig8).

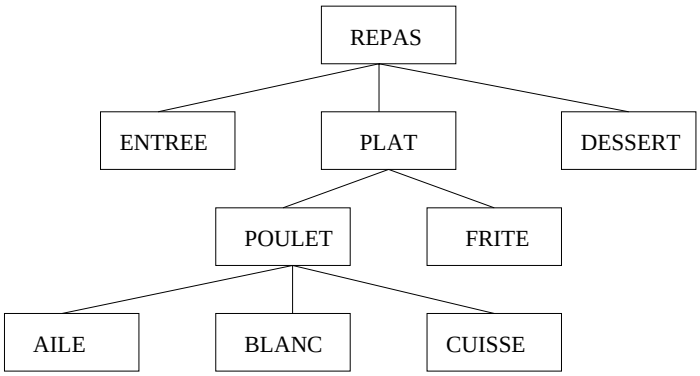


Fig. 8 – Représentation d'un repas sous la forme d'un arbre n-aire

Plus adapté que la représentation n-aire est la représentation premier fils — frère droit, car il existe une relation d'ordre entre les noeuds d'un même niveau (Fig 9).

Nous avons donc les défintions suivantes :

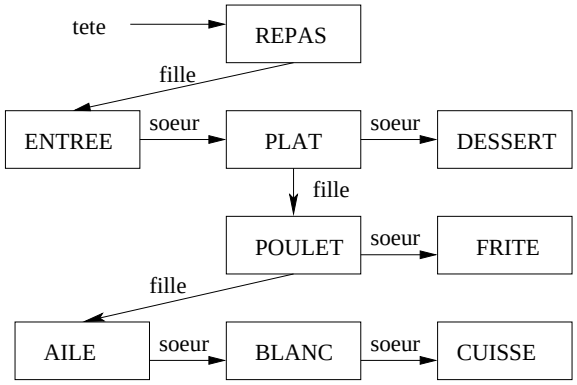


Fig. 9 – Représentation d'un repas sous la forme premier fils — frère droit

```
: POULET AILE BLANC CUISSE ;
: PLAT POULET FRITE ;
: REPAS ENTREE PLAT DESSERT ;
```

Lors dde l'exécution de `REPAS` (lors du parcours de l'arbre), l'interpréteur exécutera les mots suivants :

```
ENTREE AILE BLANC CUISSE FRITE DESSERT
```

Car seules les feuilles sont des mots Forth à exécuter (ce sont les primitives Forth pointant sur une fonction C). L'exploration de l'arbre nécéssite une deuxième pile pour stocker les adresses : c'est le rôle de la pile de retour. On note RP le pointeur de pile de retour (l'adresse stockée au sommet de la pile de retour) et IP le pointeur d'interprétation. Au cours de l'exécution il pointe en permanence sur la cellule mémoire contenant le CFA du prochain mot à exécuter. AD est le CFA courant (AD comme adresse).

L'algortihme du parcours d'arbre est le suivant :

```
EMPILER tete
REPETER
    IP = DEPILER
    TANT QUE IP n'est pas nul FAIRE
        SI le fils de IP est nul ALORS
            AD recoit IP
            SI le pointeur fct C de IP n'est pas nulle
                EXECUTER la fct C de IP
            FINSI
            IP recoit l'adresse soeur de IP
        SINON
            AD recoit IP
            EMPILER l'adresse soeur de IP
            IP recoit l'adresse fille de IP
        FIN SI
    FIN TANT QUE
JUSQU'A ce que la pile soit vide
```

La pile de retour

4.4 Les mots retardant la compilation

Le mot `HERE` empile dans la pile de donnée la sentinelle des CFA du mot en cours. Le mot `COMPILE` place le CFA du mot qui suit à l'adresse contenue dans `HERE`.

Par exemple :

```
: AJOUT COMPILE MOT1 COMPILE MOT2 ; IMMEDIAT
: NEW TUTU AJOUT TOTO ; EXECUTER
```

`NEW` sera équivalent à la suite de CFA suivant :

CFA de `TUTU` CFA de `MOT1` CFA de `MOT2` CFA de `TOTO`

4.5 Mots de branchement

SimForth dispose au départ de deux mots servant au branchement `BRANCH` qui est un `GOTO` absolu et `ZBRANCH` qui exécute un branchement absolu conditionnel à la présence d'un 0 du sommet de la pile.

Le fonctionnement de `BRANCH` est simple : l'adresse qui précde la sentinelle des CFA (qui est nulle à ce moment) boucle sur le contenu du sommet de la pile de donnée.

`ZBRANCH` fera un branchement relatif sur le contenu du paramètre de la cellule du IP en cours (qui codera une adresse vers un CFA) si la valeur présente au sommet de la pile vaut 0. Sinon il continuera comme s'il n'y avait pas eu de test.

Avec ces mots on peut écrire des mots de structuration et de répétition. Par exemple, la boucle infinie s'écrit simplement comme :

```
: BEGIN HERE ; IMMEDIATE
: AGAIN BRANCH ; IMMEDIATE
```

La definition d'un mot utilisant cette structure de contrôle :

```
: BANG
    BEGIN
        TUTU
        TOTO
    AGAIN
;
```

IF THEN ELSE DO LOOP

sera : CFA de `TUTU` CFA de `TOTO` CFA `BRANCH`

4.6 Les mots de définition

On appelle mot de définition tout mot capable de créer une tête de chaine. Les plus courants sont `VARIABLE CONSTANT : STRING ARRAY` Tous les quatres créent un mot Forth vide tel que vu dans la section XXXX. Par exemple :

```
16 CONSTANT TUTU
```

va créer le mot `TUTU`. Pour relire sa valeur on tapera la commande `TUTU .` qui va afficher la valeur 16 car l'entier est stocké dans la liste des PFA du mot `TUTU`. C'est l'exécution du code pointé par le CFA de la constante `TUTU` qui va placer 16 sur la pile. Ce code pointé n'est autre

que la deuxième partie de la définition du mot `CONSTANT`, la première étant celle qui génère la tête de chaine.

Pour cela nous avons besoins de deux mots . Il s'agit de `BUILDS` et `DOES` toujours employés par paire. La structure générale de la définition classique d'un mot de définition est sur le modèle suivant :

```
: MOTDEDEFINITION
    BUILDS
        traitement 1
    DOES
        traitement 2
;
```

C'est `BUILDS` qui créera effectivement la tête de chaîne avec deux adresses vides dans la liste des CFA. A l'exécution du mot de définition, la tête de chaîne créée portera le nom du mot qui le suit.

L'exécution du `traitement 1` a lieu lors de l'exécution du mot de définition, aprés la création de la tête de chaine. Par contre `traitement 2` n'est pas à exécuter lors de l'exécution du mot de définition.

La fonction de `DOES` sert à dérouter l'interpréteur sur le `traitement 2` lors de l'exécution du mot aprés avoir placé sur la pile le contenu de son PFA.

`DOES` est composé de deux autres mots `DOES1` et `DOES2`. `DOES1` prépare le déroutement, en modifiant les deux pointeurs de la liste des CFA créée par le mot `BUILDS`. Le contenu du premier CFA pointe désormais sur `DOES2` et son suivant pointe sur `traitement 2`. Ensuite il stoppe l'exécution, donc `traitement 2` n'est pas exécuté.

`DOES2` quant à lui mets sur la pile de donnée le contenu du PFA, c'est à dire la valeur de la constante. Enfin il déroute IP sur son `traitement 2` grâce á la pile de retour [5].

Par exemple, le fonctionnement d'une constante suit le schéma suivant :

- traitement 1 : place la valeur sur la pile dans le champ du premier PFA (c'est le mot `,`).
- traitement 2 : comme le contenu du premier PFA est sur la pile (action de `DOES2`), on ne fait rien.

```
: CONSTANT
    BUILDS
        ,
    DOES
    @
;
```

Nous avons donc le schéma suivant (**??**)

4.7 Implémentation de SimForth

Cette section décrit le fonctionnement interne de notre interpréteur SimForth. Il sinspire de celui de [5]. La structure interne de la pile et des mots Forth a été totalement refondue car trop lourde a utilisée. En effet, lors de la première soutenance, un prototype dinterpréteur était lancé. Il utilisait abondamment des listes chainées qui impliquent des appels à des allocations et des libérations de blocs de mémoire (ce qui augmente de 100 le nombre le nombre de cycles machines pour chacune delles). De plus la structure de données dun mot était trés mal faite.

Beaucoup de champs étaient nécessaires pour pour pouvoir accéder à toutes les données du mot. Par exemple, lorsquon se déplace sur un mot, on na plus accés aux sentinelles et à son nom, doù un pointeur supplémentaire dans chaque cellule. Beaucoup de champs pour rien ! Un bon ménage était nécéssaire. 5.1 Le corps du dictionaire Ce nest pas la partie la plus intéressante. Cest un simple tableau à deux champs. Un pour stocker une chaine de caractères, un autre pour stocker un pointeur sur un mot Forth. 5.2 Le corps de la pile La vitesse de la pile et de lexécution dun mot sont les deux facteurs les plus importants de linterpréteur Forth. Lors de la deuxième soutenance, le temps pour empilerdépiler une donnée était catastrophique, parce que la pile était basée une liste chainée. Maintenant cest un tableau de structures appellées Code Field (cf. sous section suivante). 5.3

Les données dans un tableau en C sont consécutives en mémoire, cest-à-dire que ladresse du deuxième élément doit etre égale à ladresse du premier plus la taille dun élement (ici un CF). Pour simplifier la suite du rapport on donne les définitions suivantes : Un mot Forth est ladresse de la première case du tableau, il est donc du type CFA. Le contenu de la première case du tableau est reservée au NFA. Les cases suivantes stockent soit un CFA soit un entier base 10 ou 16 soit un réel. Pour indiquer la fin du mot on stocke une adresse nulle (0x0), parce quavec un tableau on peut pas savoir si on est sortit. On appelera définition dun mot la suite des CF dun mot sans le NF. Par abus de langage, on peut lappeler liste des CFA.

Un mot Forth est un tableau de même structure que celle de la pile (Code Field). On appelle Code Field (CF) une structure qui contient deux champs : une union entre trois sortes dinformations : un entier, un réel, une adresse. et un entier qui indique quel est le type de la valeur stockée dans lunion. On appelle Name Field (NF) une structure qui contient les champs suivants : une chaine pouvant aller jusquà 128 caractères qui contient le nom du mot, le codage sur sept bits de la longueur de la chaine, un huitième bit appelé precedence bit (PB) qui est lié au concept de mot immédiat (cf. section 4.3). un pointeur sur une fonction C (éventuellement nul), un entier qui contient le nombre de CF utilisés. On appelle CFA et NFA (Code Field Addresse, Name Field Addresse) respectivement les pointeurs sur le début des structures CF et NF.

4.7.1 Les fonctionnalités et le fonctionnement du Debuggeur

Un mode debug permet de suivre pas à pas sur le terminal le nom du IP de AD et le nom du WFA en cours (sur quelle branche de l'arbre on se situe) et le contenu des deux piles. Il doit être utilisé avec le mot `HASH` pour un meilleur débogage.

4.7.2 Primitive C

Si le mot étudié est une primitive de base, à savoir un mot qui ne peut etre créer par aucun autre mot Forth, linterpréteur va appeler une fonction C (par exemple les mots manipulant les piles, les adresses ou les opérations arithmétiques comme SWAP R¿ +). Comment distinguer une primitive dun mot Forth non primitif ? Tout simplement un mot Forth non primitif a un pointeur nul sur la fonction. De plus lorsque linterpréteur est sur une feuille terminal il sait que le mot en cours est primitif. La gestion des primitives par linterpréteur est transparente pour lutilisateur. Ce dernier ne peut pas modifier une fonction C : il ne fait que les appeler.

4.7.3 Table HA des mots

4.7.4 Tests de rapidités

SimForth posséde un timer mesurant jusqu'aux microsecondes ! Pour mesurer la rapdité de notre Forth par rapport aux langages C et Caml et au programme Scilab. On a effecuté n fois l'opération $x * 0.9 + 1.02$, où $n \in [10^4, 10^5, .., 10^7]$. le temps d'exécution est linéaire, même pour celui de notre nouveau interpréteur — ce qui n'était pas le cas de l'ancien du à une fuite de mémoire — . Par contre nous n'avons pas pu tester la vitesse de GForth et de PForth car nous n'avons pas pu trouver de mot lançant un timer.

Nb iters	C	Caml	Nouveau SimForth	Scilab	Ancien SimForth
10^5	0.0XX	0.35	1.2	12	XXX
10^6	0.0XX	0.35	1.2	12	XXX
10^7	0.0XX	0.35	1.2	12	XXX

En conclusion SimForth est XX fois moins rapide que le C, XX fois moins rapide que Caml, mais est 10 fois plus rapide que Scilab.

4.8 Le manuel de référence

5 La base de données de données de Simtadyn

Introduction MySQL (pourquoi MySQL)

5.1 Rappels sur le fonctionnement et l'utilisation de MySQL

5.2 Lancement du serveur

5.3 Le client SymtaDyn

Fonctionnement client serveur
Les tableaux C des cartes
Les tables MySQL
Les transactions
Les fonctions Forth appelant MySQL

5.4 Conversion des cartes et tables de SymtaDyn en html

XML
XLS
HTML
Exemple.

6 Premiers exemples avec SimTaDyn

6.1 Approximation du Laplacien

Description formelle

Soit Ω un ensemble connexe ouvert du plan et son bord $\Gamma = \partial\Omega$ et une fonction g définie sur Γ. On veut résoudre le problème :

$$\begin{aligned} \Delta u(x,y) &= 0,\ \forall (x,y) \in \Omega, \\ u(x,y) &= g(x,y),\ \forall (x,y) \in \Gamma. \end{aligned}$$

Pour résoudre ce problème on réalise un maillage de Ω que l'on appellera Ω_h où h désigne un pas de discrétisation. On appelle Γ_h les points de la grille appartenant au bord. Pour approximer le problème initial on résout :

$$\begin{aligned} u_h(x,y) &= \frac{1}{4}\Big(u_h(x-h,y) + u_h(x+h,y) + u_h(x,y-h) + u_h(x,y+h)\Big),\ \forall (x,y) \in \Omega_h, \\ u_h(x,y) &= g(x,y),\ \forall (x,y) \in \Gamma_h. \end{aligned}$$

Pour implémenter la résolution de ce problème dans Simtadyn : Construire la grille qui ne sera

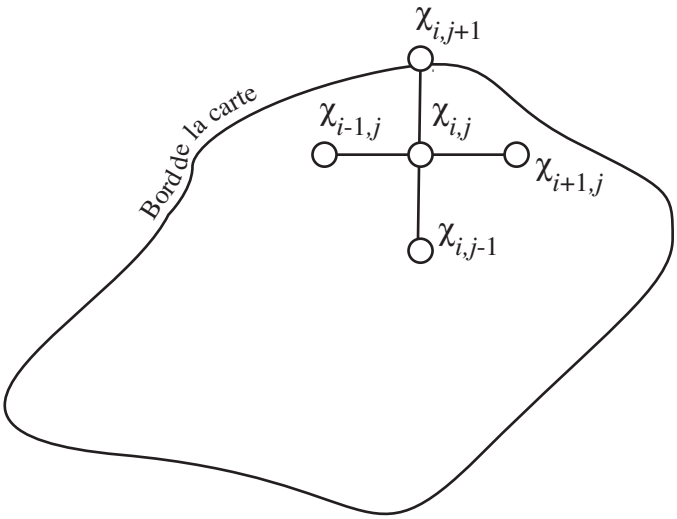


Fig. 10 – $\mathcal{X}_{i,j}$ recoit la moyenne de ses voisins.

pas nécessairement régulière en pavant une zone ; Dessiner un pavage de zones rectangulaires, puis trianguler toutes les zones en exécutant le mot `TRIANGULER`. Passer en mode *sommet*, et *sélection au lassot*, et en mode *position z* (ce dernier mode est possible). Grâce à l'arbre de visualisation des noms des champs des cellules et grâce au prompt, modifier, éventuellement, l'altitude des sommets qui sont au bord de la carte. Les sommets qui n'appartiennent pas au bord de la carte doivent posséder le mot Forth `MOY`. Les sommets du bord de la carte ne doivent pas avoir de mot Forth (éventuellement le mot `DROP`, mais aucun mot est plus recommandé). Dans la zone tampon d'entrée de texte de l'éditeur Forth, mettre sur la pile un nombre entier n assez

grand (un million, par exemple), puis cliquer sur le bouton *éxecuter toute la carte*. L'interpréteur itére n fois l'action qui consiste à exécuter le mot Forth de chaque cellule de la carte (dans notre cas, cela revient à exécuter le `MOY` des sommets). Les altitudes des sommets, qui n'appartiennent pas au bord de la carte, ont du changer. Au bout d'un certain nombre d'itérations les altitudes ne changent plus : l'approximation du Laplacien est terminée. On peut changer l'altitude d'un sommet et reexécuter toute la carte.

7 S'il y avait une cinquième soutenance ...

SimTaDyn est loin d'être fini pour pouvoir être considéré comme un logiciel potentiellement commercialisable. Voici une listre des améliorations futures possibles :

Interface

- Gérer entre les différents éditeur de SimForth : le copier — coller ; la distinction entre sauver-sous et sauver ; déplacer et fermer les onglets qui servent à stocker les fichiers textes.
- On a utilisé Glade pour construire l'interface. Glade construit les fenêtre GTK avec un fichier XML, mais l'interface de SimTaDyn a été générée en C. Il faudrait passer du C au XML, car ainsi l'interface peut être plus facilement modifiable et personnalisable (par programmation).
- Faire le bouton annuler une action.

SimForth

- La vitesse de l'Interpréteur peut encore être améliorer.
- Il faut ajouter la notion de tableaux et de structures.
- Liés SimForth avec XML, OpenGL, des librairies et des bibliothèques libres comme (inversion de matrices) et (graphes).
- Etre capable de modifier l'interface utilisateur avec des mots.
- Gérer le multithreading entre SimForth et SimGraph. Car, tant que SimForth n'a pas entièrement exécuté une commande Forth, SimGraph ne se rafraîchit pas.

SimGraphe

- Ajoût de nouveaux outils (bezier, lignes brisées, etc).
- Trianguler des polygones non convexes.
- Un outil permettant de relier les sommets et les arcs isolés afin d'obtenir des cartes uniquement constituées de zones. En effet, les arcs et les sommets ne forment pas un pavage du plan (cf. section **??**).
- Utiliser le cliking avec OpenGL pour la sélection des cellules, plutôt que d'utiliser notre méthode.
- Sélectionner plusieurs champs et types de cellules en même temps.
- Gérer le billboarding du texte lorsqu'on passe à la 3D. Le billboarding permet d'orienter une texture vers la caméra, quelque soit la position de la caméra dans l'espace.

8 Annexes

Guide de référence du Forth de SimTaDyn				
Nom	*Gestion pile*	*P*	*I*	*Commentaires*
!	($n\ adr$ —)	✓		Ecrit n à l'adresse adr.
(	(—)	★		Indique que le texte qui suit est du commentaire, délimité par) . Ce n'est pas un mot Forth mais doit être encadré par deux espaces. Il est utilisé par la préprocésseur : l'interpréteur ne le prend pas en compte.
×	($n_1\ n_2$ — $n_2 \times n_1$)	✓		Effectue la multiplication de n_1 par n_2 et stocke le résultat dans la pile.
+	($n_1\ n_2$ — $n_2 + n_1$)	✓		Effectue l'addition de n_1 et n_2 et stocke le résultat dans la pile.
,	(x —)	✓		Ajoute à la fin du dernier mot créé x.
−	($n_1\ n_2$ — $n_2 - n_1$)	✓		soustrait n_2 de n_1 et stocke le résultat sur la pile.
.	(x —)	✓		Affiche à l'écran x.
/	($n_1\ n_2$ — n_2/n_1)	✓		Effectue la division de n_1 par n_2 et stocke le résultat dans la pile.
0<	(n — $flag$)	✓		Compare n à zéro et laisse un flag sur la pile : $flag = \begin{cases} 1, \text{ si } n < 0 \\ 0, \text{ sinon.} \end{cases}$
0=	(n — $flag$)	✓		Compare n à zéro et laisse un flag sur la pile : $flag = \begin{cases} 1, \text{ si } n = 0 \\ 0, \text{ sinon.} \end{cases}$
0>	(n — $flag$)	✓		Compare n à zéro et laisse un flag sur la pile : $flag = \begin{cases} 1, \text{ si } n > 0 \\ 0, \text{ sinon.} \end{cases}$
1+	(n — $n+1$)	✓		Incrémentage rapide de un le sommet de la pile.
1−	(n — $n-1$)	✓		Décrémentage rapide de un le sommet de la pile.
Suite du tableau page suivante...				

Nom	*Gestion pile*	*P*	*I*	*Commentaires*
:	: nom			Crée un nouveau mot Forth (tableau) dont le nom est le token qui suit le : dans le tampon d'entrée (MOT). L'interpréteur passe en mode compilation et ajoute l'adresse des mots non immédiats qui suivent dans le tableau. Les mots immédiats sont directement exécutés. Le nouveau mot n'est pas encore stocker dans le dictionnaire. Il doit être utilisé avec le mot ; .
;			✓	Indique à l'interpréteur la fin d'un mot. L'interpréteur repasse en mode exécution. L'adresse du nouveau mot créé est stockée dans le dictionnaire.
:REC	:REC nom			Identique que le mot : mais stocke immédiatement l'adresse du nouveau mot dans le dictionnaire. Il est utile pour la définition de mots récursifs. Met l'interpréteur en mode compilation. Il doit être employé avec ;REC .
;REC			✓	Indique la fin de la définition d'un mot récursif. Remet l'interpréteur en mode exécution.
$<$	(n_1 n_2 — $flag$)	✓		Compare n_1 à n_2 et laisse un flag sur la pile : $flag = \begin{cases} 1, \text{ si } n_1 < n_2 \\ 0, \text{ sinon.} \end{cases}$
$=$	(n_1 n_2 — $flag$)	✓		Compare n_1 à n_2 et laisse un flag sur la pile : $flag = \begin{cases} 1, \text{ si } n_1 = n_2 \\ 0, \text{ sinon.} \end{cases}$
$>$	(n_1 n_2 — $flag$)	✓		Compare n_1 à n_2 et laisse un flag sur la pile : $flag = \begin{cases} 1, \text{ si } n_1 > n_2 \\ 0, \text{ sinon.} \end{cases}$
@	(adr — n)	✓		Empile le contenu n de la pile à l'adresse adr.
ABORT		✓	✓	Nettoie les deux piles, met l'interpréteur en mode exécution.
ALLOT	(n —)	✓		Ajoute n mots vides à la fin du dernier mot crée.

Suite du tableau page suivante...

Nom	*Gestion pile*	*P*	*I*	*Commentaires*
BEGIN	BEGIN ... AGAIN			Marque le début d'une séquence répétitive de mots. BEGIN est un HERE immédiat.
COMP		✓		Met l'interpréteur en mode compilation.
COMPILE	COMPILE MOT	✓		Lorsqu'un mot contenant COMPILE est exécuté, le mot qui le suit — dans la définition du mot en cours d'exécution — est stockée en dernière position dans la définition du dernier mot créé.
CONSTANT	n CONSTANT ¡nom¿	✓		Créé une définition de ¡nom¿ et place n en dernière position dans la d'efinition du dernier mot créé. Lorsque ¡nom¿ sera exécuté, il empilera n.
CREATE		✓		
DO	DO ... LOOP			Début d'une boucle qui se terminera suivant la valeur de paramètres de contrtôle.
DOES	CREATE ... DOES	✓		jjjjj
DROP	(n —)	✓		Supprime le sommet de la pile de données.
DUP	(n — n n)	✓		Duplique le sommet de la pile.
ELSE	MOT définition	✓		jjjjj
EXEC	MOT définition	✓		jjjjj
HERE	MOT définition	✓		jjjjj
IF	MOT définition	✓		jjjjj
IMMEDIATE	MOT définition	✓		jjjjj
LEAVE	MOT définition	✓		jjjjj
LITERAL	MOT définition	✓		jjjjj
LOOP	MOT définition	✓		jjjjj
OVER	(n1 n2 — n1 n2 n1)	✓		Duplique l'avant dernière de la pile de données.
R>	(n —)	✓		Transfère la cellule du sommet de la pile de donnée vers le sommet de la pile de retour.
SWAP	(n1 n2 n3 — n2 n3 n1)	✓		Permutte les trois dernières entrées de la pile de données.
SWAP	(n1 n2 — n2 n1)	✓		Commute les deux dernières entrées de la pile de données.
VARIABLE	n VARIABLE 'nom'	✓		Créé une définition de 'nom', et place n à la fin de la définition du dernier mot créé.
WORDS		✓		Affiche tous les mots du dictionnaire ainsi que leur définition.
Symboles :				

Conclusion sur le travail effectué

8.0.1 Forth

Par rapport à la première soutenance, le travail à progressé. Lors de la première soutenance, le mode exécution avait un bogue car il n'executait que des mots ayant trois mots dans leur définition. Maintenant il marche parfaitement.

Tous les mots vus dans ce rapport, sauf le mot `.''` et `''`, ont été crés aucours des deux soutenances et marchent. Il manque quelques mots importants comme ceux qui gérent les structures et les switch. SimForth à desormais assez de primitives pour développer les mots restants en Forth (sans plus passer par des fonctions C). Ils seront implémentés pour la troisième soutenance.

SimForth n'est pas encore ratché avec la base de donnée SQL et la carte, mais le sera pour la troisième soutenace. Il pourra gérer la syntaxe SQL et pourra manipuler les cellules de la carte.

8.0.2 Base de données et cartes

Pour cette deuxième soutenance nous avons respecter le planning du cahier des charges. Ainsi pour la troisième soutenance, afin d'optimiser les modifications du graphes, nous n'allons plus utilisé les tableaux créés en global afin d'utiliser que les bases de donnée mysql. De plus, nous allons commencer à redessinner le graphe en trois dimensions.

8.0.3 Moralité

'O Fortunatos nimium sua si bona norint agricolas !', ceci voulant dire : 'Trop heureux les hommes des champs s'ils connaissaient leur bonheur !' (latin).

Vers de Virgile dont on ne cite souvent que la première partie, laquelle s'applique à ceux ceux qui jouissent d'un bonheur qu'ils ne savent pas apprécier. Cette citation se trouve en page 1, dans *Asterix chez les bretons*.

Références

[1] http ://www.mapinfo.com

[2] http ://www.inria.fr/scilab

[3] http ://www.simcity.com

[4] http ://www.soton.ac.uk/∼trawww/stardust

[5] Walid P. Salman, Olivier Tisserand, Bruno Toulout *Forth*, Eyrolles, 1983.

[6] http ://idll.tuxfamilly.org/forth/forth.shtml

[7] MacForth Edt XXXX 1978.

[8] Xavier Michelon, xavier@linuxgraphic.org, Programmer avec OpenGL, Linux Magazine N° 25—34.

[9] Xavier Michelon, xavier@linuxgraphic.org, *OpenGL et GTK : programmer avec GTKGLArea*, Linux Magazine N° 35.

[10] Xavier Michelon, xavier@linuxgraphic.org, *OpenGL et GTK : programmer avec GTKGLArea*, Linux Magazine N° 35.

[11] David Odin, david@dindinx.org, david.odin@cpe.fr, *Programmer en GTK+*, Linux Magazine N° 6 à N° 44.

[12] *Professional Linux Programing*

qqqqqqqqq

9 MySQL

9.1 Présentation du tutorial

Ce tutorial a pour but d'expliquer en détail le fonctionnement du gestionnaire de base de données MySQL. Il accompagne l'utilisateur du projet SimTaDyn qui repose en grande partie pour le stockage en mémoire des graphes sur le système de base de données MySQL. Il présente dans une première partie le fonctionnement des base de données en gènèral et de MySQL, puis dans une seconde partie comment utiliser l'API MySQL pour créer des applications en C interfacées avec une base de donnée type MySQL.

Ce tutorial a évolué en même temps que le projet SimTaDyn, à chaque utilisation d'une nouvelle fonction ou d'une nouvelle requête au cours de la conception du programme, je l'ai ajoutée a ce tutorial.

9.2 A qui est destiné ce tutorial

Ce tutorial demande un connaissance minimum de l'utilisation d'un ordinateur, néanmoins les concepts fondamentaux sur les bases de données en gènèral sont accessible à tous.

9.3 Présentation de MySQL

MySQL est un Système de Gestion de Bases de Données (SGBD) fonctionnant sous Linux et Windows. MySQL permet d'accéder, de séléctionner, de modifier, d'ajouter, de supprimer des données dans une collection de données. SQL (Structured Query Language) est le langage de gestion de base de donnée le plus utilisé. MySQL est Open Source et sous license GPL ce qui permet de l'utiliser librement en respectant les conditions de la liscence GPL consultable à l'adresse suivante : http ://www.fsf.org/licenses. MySQL comprend un client et un serveur de base de données. Le serveur MySQL est réputé pour être le plus rapide serveur de base données mais aussi le plus facile à utiliser. De plus, MySQL est très bien documenté. L'intéret d'utiliser MySQL est donc évident : communauté active, usage libre et facilité d'utilisation !

9.4 Installation

Toutes les informations concernant l'installation de MySQL sont consultables sur le site http ://www.mysql.com Les fichiers qui sont proposés d'installer ici permettent de lancer un serveur MySQL, d'y accéder avec un client, et de développer des applications ayant accés à une base de données MySQL. Nous allons détailler l'installation d'un client et d'un serveur sur le systme Linux.

L'installation la plus simple est d'utiliser les rpms proposé sur le site, il faut alors récupéré :

- Le package serveur :
  `MySQL-<version>.<archi>.rpm`
- Le package client :
  `MySQL-client-<version>.<archi>.rpm`

Comme nous souhaitons développer des applications interfacées avec MySQL, il faut ajouter les packages de bibliothèque :

- Le package shared component :
  `MySQL-shared-<version>.<archi>.rpm`

– Le package developpement :
`MySQL-devel-<version>.<archi>.rpm`

Une fois ces packages installés, assurez vous de lancer le serveur MySQL avec :

```
$ /etc/rc.d/init.d/mysql start
```

Vous pouvez aussi regardé qu' il existe bien un processus mysqld avec :

```
$ ps -aux
```

9.5 Reprèsentation des données dans une base de données

Les bases de données MySQL sont composées de tables (elle même contenue dans la base de donées) qui contiennent en ligne des données et en colonnes des champs. A chaque ligne correspond une donnée composée de plusieurs champs (on peut faire le rapprochement avec une structure en C, les champs sont le prototype de la structure).

Exemple 1 :

Champ 1	Champ 2	Champ 3
Valeur 1.1	Valeur 1.2	Valeur 1.3
Valeur 2.1	Valeur 2.2	Valeur 2.3

Exemple 2 (table "voiture") :

Nom	Marque	Couleur	Id
Voiture 1	Audi	Bleu	1
Voiture 2	Porsche	Gris	2

Ainsi, chaque ligne correspond un nom de voiture, sa marque, sa couleur et un identifiant.

9.6 L'acrhitecture client/serveur de MySQL

MySQL fonctionne avec une architecture composée en client/serveur. Les données sont stockées dans des tables (une sorte de tableau). On ne peut pas accéder directement à ces tables depuis un fichier texte pour en modifier le contenu pour des raisons évidentes :

– Si on travaille sur une table lourde (avec des dizaines de milliers d'éléments), la recherche de l' élément à modifier devient fastidieuse.
– On peut accéder à la base de données que depuis l'ordinateur ou elle est stockée.

Donc, avec MySQL, c'est un serveur qui est chargé de vous transmettre les données que vous lui demandez. Il peut tout autant renvoyer toute la table, comme la valeur du champs j pour la ligne i. Pour demander au serveur de faire des opérations sur la base de donnée (on dit "faire exécuter au serveur une reqète"), il faut pour cela utiliser un client. C'est le même principe que consulter une page web, la page se trouve sur un serveur, par exemple Apache, quelque part sur Internet, et on demande à ce serveur de nous renvoyer la page qu'on lui demande pour l'afficher dans notre client (on parle de browser pour un client HHTP). A l'instar d'un serveur web, le serveur MySQL est accéssible depuis n'importe où sur Internet.

Vous devez la première fois vous connecter au serveur MySQL à l'aide du client MySQL en root :

```
$ mysql -u root -p mysql
```

Pour quitter, taper simplement `quit` dans le shell du client.

9.7 Ajouter un utilisateur sur un server MySQL

Vous pouvez consulter la documentation sur le site de MySQL concernant la création et la gestion d'utilisateurs du serveur MySQL :

`http://www.mysql.com/documentation/mysql/bychapter/manual_Tutorial.html#Connecting-disconne`

Connecter vous en root sur le serveur, puis rentrer la requéte de crèation d'utilisateur :

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'juzam'@'%'
    ->     IDENTIFIED BY 'biscuit' WITH GRANT OPTION;
```

Ici on crée un utilisteur nommè juzam, qui pourra se connecter au serveur depuis n'importe où (en local, depuis internet, ...) qui s'identifiera avec le mot de passe biscuit, et avec un accès total au serveur. Tous ces paramtres sont modifiables pour dèfinir certains accès au serveur.

9.8 Les requêtes

Les requêtes sont les expressions interprétées par le serveur MySQL, c'est le language SQL (Structured Query Language) qui permet de dialoguer avec le serveur (en quelque sorte). Ainsi grâce au requêtes on peut demander au serveur de nous renvoyer que certaines données (lignes) avec certains champs (colonnes). On peut aussi lui demander de créer une nouvelle base de donées, une nouvelle table dans une base, de supprimer la valeur du champ j pour la ligne i, ou bien encore de la modifier, ...

Vous pouvez tester toutes ces requêtes en vous connectant sur le serveur MySQL. Il suffit de taper ces requêtes directement dans le shell du client MySQL.

Il existe en SQL des mots qui sont le squelettes de toutes requêtes :

CREATE DATABASE

Permet de crèer un nouvelle base sur le serveur MySQL. Il faudra ensuite remplir cette base de données avec des tables.

Prototype :

```
CREATE DATABASE nom_de_la_base;
```

USE

Permet de séléctionner et d'utiliser sur le serveur MySQL une base de donnée. En effet un serveur peut hèberger plusieurs base de données et il faut faut lui indiquer sur quelle base on souhaite travailler.

Prototype :

```
USE nom_de_la_base;
```

DROP DATABASE

Permet de supprimer une base de donnée sur le serveur MySQL.
Prototype :

```
DROP DATABASE nom_de_la_base;
```

CREATE

Permet de créer une base de données ou une table dans une base de données.
Prototype :

```
CREATE TABLE nom_de_la_table (
    Champs_1 TYPE,
    Champs_2 TYPE,
    ...
    Champs_x TYPE
);
```

Exemple :

```
CREATE TABLE voiture (
    Nom TEXT,
    Marque TEXT,
    Couleur TEXT,
    Id INTEGER
);
```

Cette requête créée la table voiture dans la base de donnée courante avec les champs Nom, Marque, Couleur, Id. (Se rapporter à l'exemple 2 section 9.6)

INSERT

Permet d'insérer des données dans une table.
Prototype :

```
INSERT INTO nom_de_la_table(Champs_i, Champs_j ... Champs_k)
VALUES(Val_i, Val_j ... Val_k);
```

Exemple :

```
INSERT INTO voiture(Nom, Marque, Couleur, Id) VALUES
("Voiture 3", "BMW", "Noir", 3);
```

Cette requête insert à la 3ème ligne la Voiture 3 avec ces attributs. Note : Pour utiliser `INSERT` il faut que l'emplacement où l'on souhaite insérer des données soit libre, sinon il faut utiliser une requête basée sur `UPDATE`.

UPDATE

Permet de modifier la valeur d'une donnée
Prototype :

```
UPDATE nom_de_la_table SET Champ_i = Val_i;
```

Exemple :

```
UPDATE voiture SET Couleur = "Rouge" WHERE Nom = "Voiture 2";
```

Cette requête va changer la valeur du champ Couleur pour la Voiture 2.

DELETE

Permet de supprimer la valeur d'une donnée. Cette requête fonctionne de la même facon que `SELECT`.
Prototype :

```
DELETE Champ_i = Val_i FROM nom_de_la_table;
```

Exemple :

```
DELETE * FROM voiture;
```

Cette requête vide entièrement la table voiture (mais ne la supprime pas !).

SELECT

Permet de récupérer des données dans une table.
Prototype :

```
SELECT Champ_i, Champ_j ... Champ_k FROM nom_de_la_table;
SELECT id, Name FROM voiture WHERE Couleur = "Bleu";
```

Cette requête renvoie un tableau composée des champs Id et Name (dans cette ordre) et en lignes de toutes les voitures de couleur bleue. On a pu pu sèlèctionner toutes les voitures de couleur bleue grâce à l'expression conditionelle WHERE qui permet de trier des données, et peut s' utiliser avec `INSERT`, `UPDATE`, `DELETE`, `SELECT`.
Exemple :

```
SELECT * FROM VOITURE;
```

Cette requête renvoie le tableau de l'exemple 3.2

10 L'API MySQL

10.1 Présentation de l'API

L'API MySQL permet d'éxécuter des requêtes sur une base de données MySQL depuis une application codée en C (donc un client). Pour compiler les sources utilisant la bibliothèque mysqlclient, il faut que la zli soit installée. La commande gcc pour compiler du code ayant recourt à mysqlclient dans le cadre de SimTaDyn est :

```
$ gcc -Wall -W file.c -o prog -I/usr/include/mysql
      -L/usr/lib/mysql -lz
```

Note : La commande permet de connaitre les options de compilation néccéssaires sur une machine.

10.2 La bibliothèque mysqlclient

Fonctions de l'API utilisées pour la premières soutenance du projet SimTADYN :

```
MYSQL *mysql_init(MYSQL *conection);
```

Initialise une connection vers un serveur MySQL.

```
MYSQL *mysql_real_connect(
    MYSQL *connection
    const char *server_host,
    const char *sql_user_name,
    const char *sql_password,
    const char *sql_db_name,
    unsigned int port_number,
    const char *unix_socket_name,
    unsigned int flags
);
```

Se connecte et se log au serveur, sélectionne une base de données.

```
mysql_query(MYSQL *connection, const char *request_select);
```

Exécute une requête SQL.

```
mysql_store_result(MYSQL *connection);
```

Stocke en mémoire le résultat d'une requête SQL.

```
mysql_num_rows(MYSQL_RES *result);
```

Renvoie le nombre de ligne du résultat d'une requête SQL (type `SELECT`).

```
mysql_fetch_row(MYSQL_RES *result);
```

Récupère une ligne du résultat d'une requête SQL (type `SELECT`).

```
mysql_free_result(MYSQL_RES *result);
```

Libère la mémoire utilisée pour stocker le résultat d'une requête SQL (type `SELECT`).

```
mysql_close(MYSQL *connection);
```

Termine la connection à un serveur MySQL.

11 Représentation des graphes en mémoire

11.1 Problèmatique

La représentation des graphes en mémoire était le problème que j'ai étudiée (Benoit M.) pour la première soutenance. Il fallait trouver un représentation en mémoire statique pour stocker et sauvegarder un graphe, mais aussi une représentation en mémoire vive qui soit utilisable par Minh pour représenter graphiquement le graphe, et optimiser la vitesse d'éxécution des différentes opérations et algorithmes qui seront implémentés au fur et a mesure de l'évolution du projet.

11.2 Utilisation de MySQL pour la représentation en mémoire statique

Pour sauvegarder un graphe, le choix le plus évident était de travailler avec un fichier texte. Avec une telle représentation, plusieurs problèmes se posaient :

- L' implémentation des fonctions de parsing du fichier.
- Des fichiers textes lourds dans le cas de graphes composés de nombreux sommet.
- Des fichiers textes inutilisables s'ils venaient à être endomagés.
- Le travail sur un graphe est mono-utilisateur, et on est obligé d'avoir le graphe sur la machine pour travailler dessus.

J'ai donc pensé à utiliser une base de données plutot qu'un fichier texte. Les besoins de rapidité, fiablilité, facilité d'utilisation et utilisation libre m'ont amené à opter pour MySQL (http ://www.mysql.com). MySQL est en effet fourni avec un API détaillé qui permet de construire des applications ayant accés à une base de données MySQL.

Les avantages sont nombreux :

- Plus de parsing de fichiers, les fonctions de recherche, insertion, suppression, modification ... sont directement implémentées dans MySQL sous forme de requête SQL.
- L'architecture client/serveur de MySQL permettra de charger par internet depuis n'importe quel client Simtadyn un graphe stocké sur un serveur MySQL situé à l'autre bout du monde, de faire des traitements dessus, puis de le sauvegarder sur le serveur distant, comme en local.