Projet TB7 :

Techniques avancées de programmation

Présentation générale

Le projet consiste en la réalisation d'un *grapheur d'expressions fonctionnelles*, c'est à dire d'un logiciel destiné à représenter graphiquement des expressions du type:

```
sin(x*abs(x))+2 , exp(x+(6*log(x+1))) etc...
```

Ce projet est trop important pour être réalisé entièrement par un binôme en trois jours. Aussi un découpage de celle-ci en unité autonome vous sera proposé. Ce découpage repose sur les grandes étapes de traitements nécessaires pour passer d'une expression fonctionnelle à une représentation graphique:

- analyse lexicale
- analyse syntaxique
- interprétation du code généré
- interface graphique

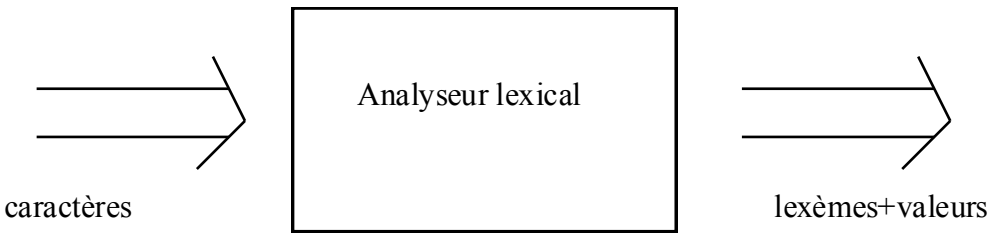
Chacune de ces étapes de traitement sera prise en charge par un binôme. Chaque projet sera donc réalisé par une équipe de quatre binômes. Dans chaque équipe un chef de projet sera responsable de l'intégration finale du produit. Ce dernier point est fondamental : chaque binôme devra travailler en suivant strictement le cahier des charges défini en commun par l'équipe pour l'unité dont il est responsable. Concrètement les hypothèses de travail au niveau du binôme sont les suivantes :

> *- il faut se contenter des services offerts par les modules avals*
>
> *- il faut offrir exactement les services attendus par les modules en amont.*

Si le développement fait apparaître la nécessité de modifier la spécification d'un de ces services cela est l'indication d'une insuffisance dans l'analyse de départ. La modification doit être validée par le chef de projet, qui doit rester le garant de l'*intégrabilité* finale du produit, et qui doit donc se soucier essentiellement de la cohérence d'ensemble du projet.

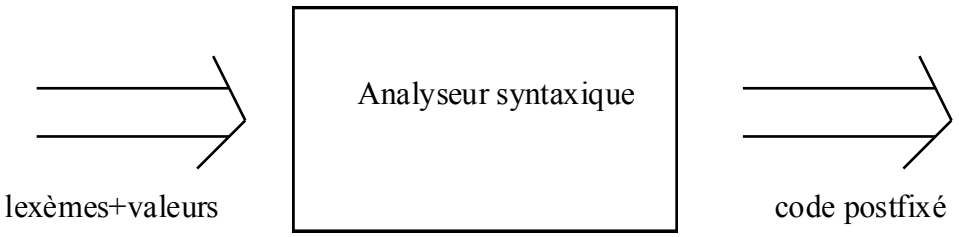
Description des traitements

Analyse lexicale
L'analyseur lexical traite un flux de caractère entrant. Son rôle est de produire en sortie un flux d'entités lexicales (appelées lexèmes ou *jetons*) accompagnées éventuellement d'une indication de *valeur*.



Les différents types de lexèmes peuvent être par exemple : REEL, OPERATEUR, FONCTION, ERREUR, FIN, PAR_OUVR, PAR_FERM, VARIABLE, NO_TOKEN
Les jetons REEL, OPERATEUR, FONCTION doivent être accompagnés d'une valeur lors de leur production.

Analyse syntaxique
L'analyseur syntaxique traite un flux de couples *lexème+valeur* entrant. Son rôle est de vérifier la conformité de l'expression avec la grammaire définie et de produire en sortie un flux de code *postfixé (en forme d'arbre binaire)*.



La grammaire est la suivante:

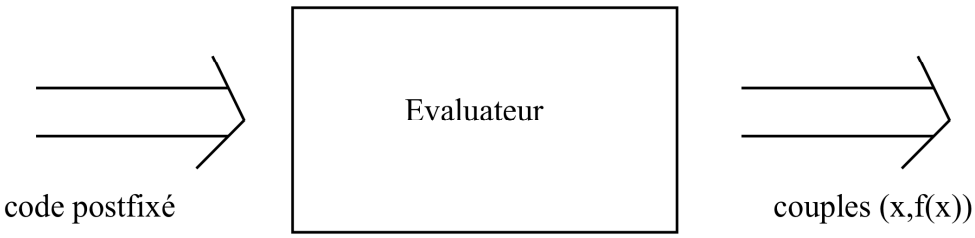
Règle 1	exp::=nombre réel
Règle 2	exp::=variable
Règle 3	exp::=fonction exp
Règle 4	exp::=(exp operateur exp)
Règle 5	exp::=(exp)
Règle 6	expression_complète::= exp fin

Les regles de production sont les suivantes:

Regle1	nombre réel
Règle 2	variable
Règle 3	prod(exp) fonction
Règle 4	prod(exp) prod(exp) operateur
Règle 5	prod(exp)
Règle 6	prod(exp)

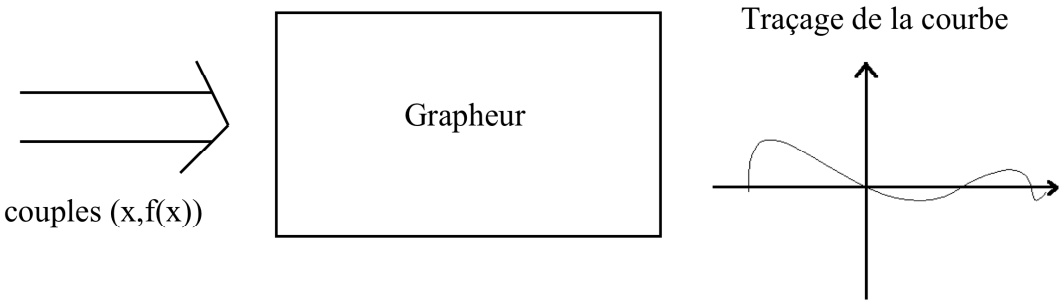
La conception de l'arbre binaire (ou la *pile*) pour stocker le code postfixé relève de la responsabilité du module suivant (*Evaluateur*)

L'évaluateur
Son rôle est de traiter le code postfixé produit en sortie par l'analyseur syntaxique et de permettre la production des couples *(x,f(x))* à destination du grapheur.



Nous conseillons le stockage du code de sortie dans un tableau, et la mise en oeuvre d'une évaluation utilisant une *pile*.

L'afficheur
Ce module va chercher une liste de couples *(x,f(x))* et prend les décisions nécessaires à son affichage (choix d'échelle, interface utilisateur, graduations, axes...). Ce module est chargé de gérer *toutes* les formes d'interaction entre l'utilisateur et l'ordinateur.



Utilisation des unités de compilation séparée

Chaque module sera implanté sous la forme d'une unité de compilation séparée C (couple fichier d'extension *c* et *h*) . C'est la définition de l'interface (le fichier header) et de la visibilité réciproque de chacune de ces unités qui sera l'objet d'une attention particulière sous la direction du chef de projet.
Certains modules communiquent entre eux par appels de procédures, d'autres en partageant des structures de données communes. Essayer de définir clairement le mode de coopération entre votre module et ceux avec lesquels il est en relation. La spécification de chacun des modules doit être complète et définie avant le développement de chacun d'entre eux.
Pour tester chaque module vous aurez à écrire un programme principal de test et à implanter sous une forme minimale les structures de données dont vous aurez besoin pour ces tests. L'intégration du module dans le projet final n'est envisageable que lorsque vous aurez testé la validité et la robustesse de votre module par rapport à son cahier des charges initial.

Cahier des charges de l'application

Il s'agit donc de concevoir et développer une application qui permette à un *utilisateur* (et non à un *programmeur*) d'afficher la représentation graphique d'une fonction quelconque. Celle-ci est exprimée en utilisant les opérateurs arithmétiques courants (+, -, *, /) et les fonctions de base usuelles (abs, sin, sqrt, log etc...).
Le parenthèse est possible sans limitation pourvu qu'il soit syntaxiquement correcte (*(((x+((5)))*2))*est correcte). Ces expressions comporteront une variable (notée *x* ou *X*). Lors de la saisie de l'expression il sera également proposé à l'utilisateur d'indiquer les bornes d'études de la fonction. La gestion des erreurs (division par zéro, racine de réel négatif etc...) sera contrôlée par l'application. Une interface conviviale sera proposée à l'utilisateur. Celle-ci devra lui permettre de modifier commodément l'expression en cas d'erreur, de modifier l'intervalle d'étude etc... D'autres facilités pourront facultativement être fournies : zoom etc...
Pour simplifier l'utilisation des opérateurs est restreinte à deux opérandes. Ainsi (5+(4+x)) est une expression légale, mais non (5+4+x).

Modélisation par objets

Objets pertinents de l'application

Les considérations précédentes conduisent à retenir les objets informatiques suivant :
Dialogueur, Analyseur lexical, Analyseur syntaxique, Générateur de code, Evaluateur, *Afficheur, Expression, Couple lexème-valeur, Pile*. Nous les représentons sous forme de rectangles.

Dialogueur	**Analyseur lexical**	**Analyseur syntaxique**

Générateur de code	Evaluateur	Afficheur

Couple lexème-valeur	Pile	Expression

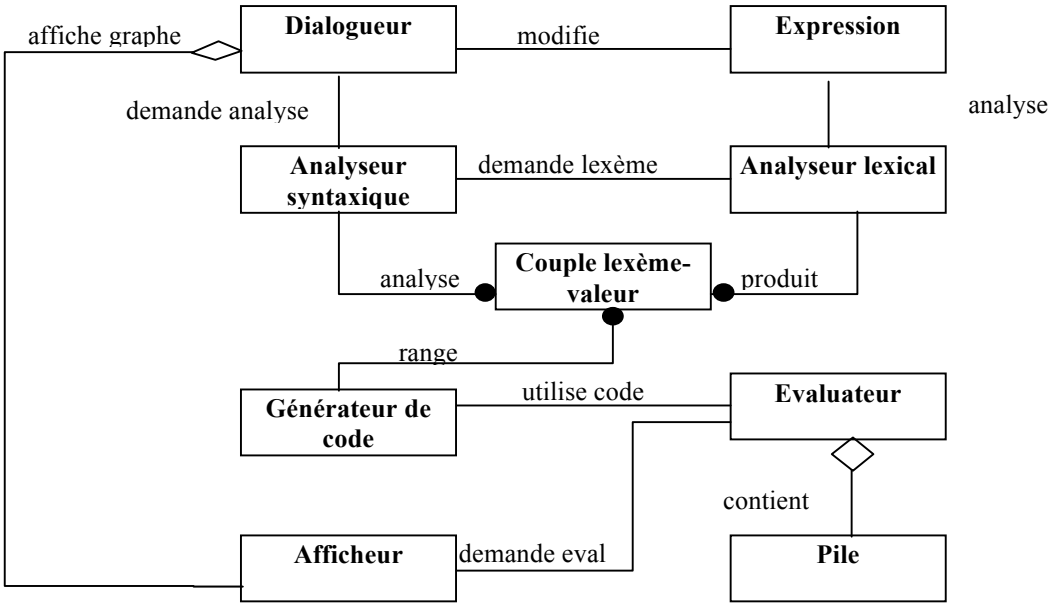
Dictionnaire des objets de l'application

Dialogueur : Entité permettant la saisie interactive des informations demandées à l'utilisateur concernant l'expression, l'intervalle d'étude et autres informations facultatives.
Analyseur lexical : Entité construisant à partir de l'expression une séquence de couple lexème+valeur.
Analyseur syntaxique : Entité chargée de contrôler la conformité de la séquence des lexèmes générés avec la grammaire choisie.
Générateur de code : Entité chargée de produire une séquence de codes postfixés (éventuellement représentés sous une forme d'arbre) à partir d'une séquence de couples lexème-valeur.
Afficheur : Entité chargée d'afficher la représentation graphique.
Couple lexème valeur : Codage résultant de l'interprétation de l'expression par l'analyseur lexical et destiné à identifier dans une expression les fonctions, opérateurs, nombres, parenthèses etc..
Expression : Chaîne de caractères saisie par l'utilisateur et destinée à représenter une expression arithmétique.
Pile : Structure de données nécessaire pour l'évaluation d'une séquence de codes postfixés. Cette structure supporte principalement deux opérations: empiler et dépiler.

Identifications des associations

Il s'agit ici de définir les relations entre les différents objets constitutifs de l'application.

Le *Dialogueur* modifie *l'Expression* et l'intervalle d'étude.
Le *Dialogueur* demande à l'*Analyseur syntaxique* d'analyser l'expression.
Le *Dialogueur* demande à l'*Afficheur* de mettre en œuvre la représentation graphique.
L'*Analyseur Lexical* analyse l'*Expression*.
L'*Analyseur Lexical* produit des *Couples Lexème-valeur*, sur demande de l'*Analyseur Syntaxique*.
L'*Analyseur syntaxique* demande à L'*Analyseur Lexical* un nouveau *Couples lexème valeur*
L'*Analyseur syntaxique* analyse des *Couples lexèmes valeurs*.
Le *Générateur de code* range certains couples dans un ordre postfixé.
L'*Evaluateur* utilise la séquence de codes post fixés construite par le générateur.
L'Evaluateur calcule la valeur de *f(x)* pour une valeur de *x* donnée, grâce à sa *Pile.*
L'*Afficheur* demande à l'*Evaluateur* la valeur de *f(x)* pour une valeur de *x* donnée.



Modélisation dynamique

Exemples de scénarios utilisateur

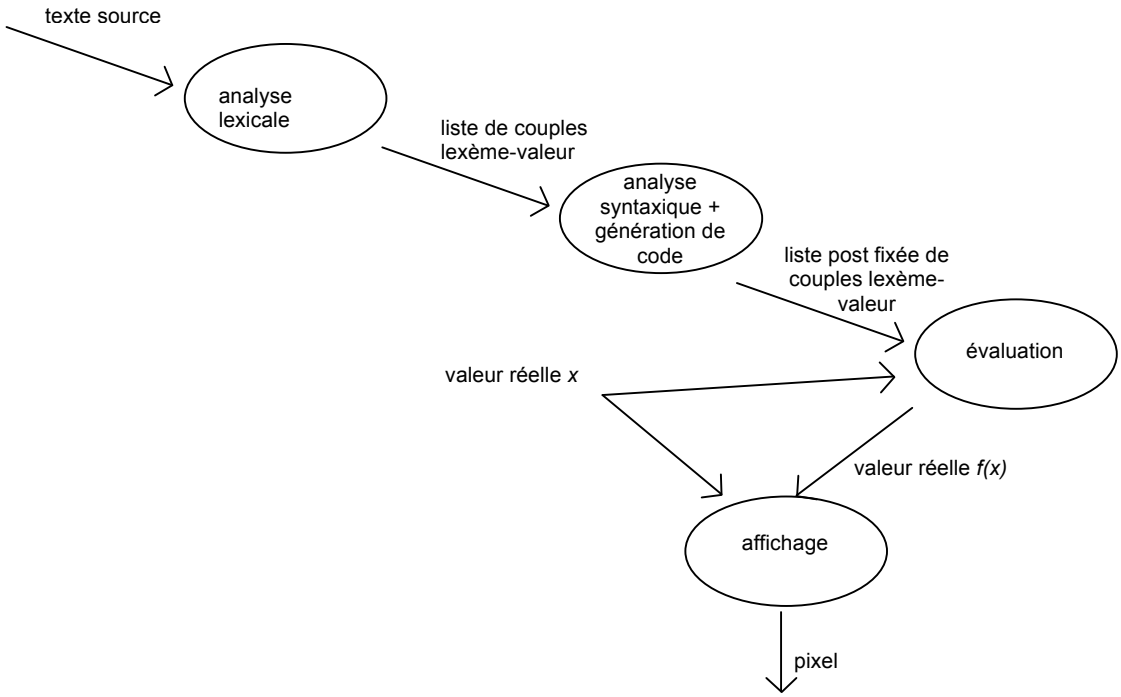
Scénario normal :
Le *Dialogueur* invite *l*'utilisateur à entrer une expression
L'utilisateur entre une expression et en déclenche l'analyse.
Le *Dialogueur* invite *l*'utilisateur à entrer la borne inférieure de l'intervalle d'étude
L'utilisateur entre une borne inférieure
Le *Dialogueur* invite *l*'utilisateur à entrer la borne supérieure de l'intervalle d'étude
L'utilisateur entre une borne supérieure.
l'*Afficheur* affiche la représentation graphique.
Le *Dialogueur* invite *l*'utilisateur à entrer une nouvelle expression ou un nouvel intervalle d'étude.

Exemple de scénario en cas d'erreur :
Le *Dialogueur* invite *l*'utilisateur à entrer une expression
L'utilisateur entre une expression et en déclenche l'analyse.
L'*Analyseur syntaxique (*ou *lexical*) détecte le cas d'une expression erronée
Le *Dialogueur* affiche un message d'erreur approprié.
Le *Dialogueur* invite *l*'utilisateur à entrer une expression
Le *Dialogueur* invite *l*'utilisateur à entrer la borne inférieure de l'intervalle d'étude
L'utilisateur entre une borne inférieure
Le *Dialogueur* invite *l*'utilisateur à entrer la borne supérieure de l'intervalle d'étude
L'utilisateur entre une borne supérieure.
Le *Dialogueur* détecte une erreur borne inférieure > borne supérieure
Le *Dialogueur* invite *l*'utilisateur à entrer la borne inférieure de l'intervalle d'étude
L'utilisateur entre une borne inférieure
Le *Dialogueur* invite *l*'utilisateur à entrer la borne supérieure de l'intervalle d'étude
L'utilisateur entre une borne supérieure.

L'Afficheur affiche la représentation graphique.
L'*Evaluateur* détecte une division par zéro.
L'Afficheur affiche un message d'erreur "Division par zéro"
Le *Dialogueur* invite l'utilisateur à entrer une nouvelle expression ou un nouvel intervalle d'étude.

Modélisation fonctionnelle

On ne s'intéresse ici qu'aux flux de données.



Remarques sur la réalisation du projet

Dépendance des modules

Les 4 modules principaux de ce projet (analyseur lexical, analyseur syntaxique, évaluateur et grapheur) ne présentent pas le même niveau de dépendance par rapport au contexte système de la réalisation du logiciel.

- Les modules *analyseur lexical*, *analyseur syntaxique* et *évaluateur* doivent être conçus de façon à les rendre indépendant du dispositif d'affichage. Ces modules mettent en effet des algorithmes présentant une logique intrinséque et ne nécessitant pas de fonctionnalités spécifiques au système. En d'autres termes il doit être possible de replonger ces modules dans d'autres contextes d'utilisation (contexte système, contexte d'environnement graphique ou contexte applicatif) sans être contraint de les remettre en cause profondément dans leur conception.
- Le module *Grapheur* présente quant à lui un grand niveau de dépendance vis à vis du système. Il représente l'IHM de l'application et, par nature, sera dans sa conception très dépendant du contexte système, graphique ou applicatif de sa mise en oeuvre. Dans ce projet nous proposons d'utiliser le contexte fourni par openGL : il est donc clair qu'un changement de contexte serait une opération a priori couteuse pour ce module, mais il néanmoins intéressant qu'un tel changement de contexte ne nécessite que des modifications circonscrite à ce module.

La bibliothèque OpenGL

Sur l'un (des nombreux sites) consacré à OpenGL on trouve la définition suivante de cette librairie :
"*OpenGL is a low-level graphics library specification. OpenGL makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered into the frame buffer*."
Nous utilerons OpenGL à travers une interface simplifiée fournie pour les besoins de ce projet. Ce module intermédiaire apparaît sous la forme (classique) de 2 fichiers :

- *graph.h* présente les quelques fonctionnalités nécessaires dans le cadre de ce projet
- *graph.c* présente leur implémentation. Cette dernière s'appuie sur la librairie native OpenGL.

L'accès à ces fonctionnalités simplifiées nécessite donc l'intégration de ces deux fichiers au projet .
L' exemple simple de mise en oeuvre ci-dessous peut être également intégré à titre de test :

```
/ *     test.c     */
#include "graph.h"

int bascule=0;

void myKey(int c) {
  switch(c) {
    case 'a':
      bascule^=1; /* La bascule passe alternativement de 0 a 1 */
      break;
  }
}

void myDraw(void) {
  /* trace une ligne blanche diagonale */
  setcolor(1.0F,1.0F,1.0F);
  line(-1.0,-1.0,1.0,1.0);
  if (bascule) {
    /* Trace un rectangle rouge a l'ecran si active
     * par appui de la touche 'a' */
    setcolor(1.0F,0.0F,0.0F);
    bar(-0.5F,-0.5F,0.5F,0.5F);
  }
  /* ecrit le message "bonjour" en jaune */
```

```
  setcolor(1.0F,1.0F,0.0F);
  outtextxy(0.0,0.0,"Bonjour");
}

int main(int ac,char *av[]) {
  InitGraph(ac,av,"Essai Glut",640,480,myDraw,myKey);
  return 0;
}
```

Il faut noter que dans ce type de programmation la logique de déroulement de l'application est conditionnée par la survenues de différents *événements* : dans un contexte OpenGL une application réagit à des évènements extérieurs tels que, par exemple, un clic de souris, un ordre de réaffichage, l'appui sur une touche du clavier etc... Cette caractéristique est d'ailleurs commune à toutes les applications graphiques évoluant dans les contextes systèmes multi-fenêtrés (Windows, X-Window etc...)
Dans cet exemple les fonctions (dites *callback*) *myDraw* et *myKey* sont installées ici par l'appel *InitGraph* en tant que fonctions *réagissantes* aux évènements de "re-dessinage" (pour *myDraw*) et aux évènements d'appui sur une touche du clavier (*myKey*).

Planning de réalisation du projet

Le projet est composé de 2 créneaux de TD, 14 créneaux de TP et de 3 créneaux de TOP, d'1h30 chacun. Les créneaux incluent :

- 2 créneau TD de conception (lundi après-midi)

- 10 créneaux de réalisation des modules du projet (mardi et mercredi)

- 2 créneaux d'assemblage des modules (jeudi après-midi)

- 3 créneaux d'évaluation du projet (vendredi après midi)

- 2 créneaux de réalisation des rapports (vendredi matin) : rapport de maintenance, rapport d'utilisation du logiciel, et les commentaires des codes.

La fourniture du projet est un répertoire en format zip contenant :

- L'application (l'objet exécutable issu de la compilation et l'édition des liens)

- Le rapport d'utilisation

- Le rapport de maintenance

- Un document ***ReadMe*** qui contient le nom du chef de projet, les noms des développeurs et leurs modules.

- Un sous-répertoire contenant les codes sources des modules, incluant le projet. Il faut clairement préciser, en en-tête de chaque fichier de code source, les noms des développeurs impliqués. Les noms de fichiers doivent être significatifs.

La fourniture (délivrable) du projet doit être déposée sur la plateforme pédagogique Whippet (**Dépôt de projet**) **plus tard le19 Mars (Gr. 3 & 4) et le 26 Mars (Gr. 1 & 2).** Au delà de cette date, aucune fourniture ne sera acceptée.