le système pédagogique

G-ALG

pour bien démarrer en algorithmique

Gilles HUNAULT, université d'Angers

Sommaire

Remerciements aphabétiques

Remerciements alphabétiques

Jacques	BOYER
Laurence	BRUNET
Jean-Matthieu	CHANTREIN
Benoit	DA MOTA
Bertrand	NOUHAUD

+ toutes les étudiantes et tous les étudiants qui ont "subi" mes cours d'algorithmique, de programmation et/ou de développement
+ tous les collègues parce qu'elles/ils étaient pour ce projet ou contre ce projet, ce qui m'a obligé à le perfectionner

1. Objectifs de G-ALG

Le système pédagogique **G-ALG** a pour buts principaux :

- d'apprendre **à analyser un problème** en termes algorithmiques ;
- de fournir une **habitude aux *bonnes pratiques*** de la programmation et du développement ;
- d'aider à **écrire une solution algorithmique** une fois *une* solution "manuelle" trouvée ;
- de **vérifier l'exactitude** de cette solution algorithmique ;
- de **s'assurer de l'exhaustivité** de la solution fournie.

De façon annexe, **G-ALG** peut aussi aider à **découvrir** la programmation dans un langage donné, sans se substituer à l'apprentissage des langages.

2. Présentation rapide de G-ALG (1)

Ce n'est pas tant les langages que la démarche et la méthode qui posent problème, notamment pour l'apprentissage de l'algorithmique au lycée.

Il nous semble qu'on ne s'assure pas assez que les étudiant(e)s ont compris les notions fines d'instruction, de séquentialité et d'exhaustivité avant de les mettre à programmer des solutions explicites.

On n'insiste sans doute pas non plus assez sur l'écriture des commentaires, l'existence de plusieurs solutions, sur la disponibilité de bibliothèques de fonctions, sur le fait que l'on fait des choix en programmant telle ou telle solution.

G-ALG a pour ambition de forcer les étudiant(e)s à se poser des questions quant à la démarche et aux solutions avant de recourir à un langage.

2. Présentation rapide de G-ALG (2)

Voici des **bonnes pratiques** que les langages n'imposent pas :

- écrire des commentaires ;
- initialiser les variables (quand on doit les déclarer) ;
- indenter le code (sauf Python) ;
- mettre les fins de structure sur des lignes séparées ;
- décomposer les actions à réaliser ;
- factoriser du code similaire ;
- afficher la date, l'heure et calculer une durée entre deux dates ;
- résoudre "à la main" le problème, quitte à simplifier les étapes.

2. Présentation rapide de G-ALG (3)

Pour atteindre les objectifs proposés, **G-ALG** s'appuie sur :

- **des exercices** détaillés, progressifs et exhaustifs ;
- **un langage** simple, conceptuel et **contraignant en français** ;
- **une interface** Web simple à utiliser ;
- **un système** sous-jacent d'analyse, d'exécution et de validation des algorithmes qui s'exécute en ligne de commandes et qui peut donc s'automatiser.

Et, sans doute dans un avenir proche, une intégration dans ***Moodle*** avec **une base de données d'exercices progressifs** et une **certification**.

3. L'exemple puNaTTC (1)

Voici un exemple d'exercice que **G-ALG** peut aider à résoudre :

> *On voudrait connaitre le prix, toutes taxes comprises et au centième d'euro près, de n articles achetés au prix unitaire p, avec un taux de taxe donné t, sachant qu'au-delà de s articles, on a une réduction de r %.*

Il n'y a, sans doute, pour toute personne qui sait programmer, aucune difficulté à effectuer le calcul du prix TTC **hors réduction** défini par la formule $n \times p \times (100 + t)/100$ puis à arrondir au centième d'euro.

Mais pour bien répondre à l'exercice, si on veut appliquer la réduction, il faut être plus prudent car il faut **d'abord** appliquer la réduction au prix TTC "brut" **puis** arrondir au centième d'euro.

3. L'exemple puNaTTC (2)

Pour un(e) débutante(e), il est donc certainement prudent de décomposer le problème et d'écrire les étapes suivantes, à l'aide du symbole # que **G-ALG** utilise pour les commentaires :

 # 1. calcul du prix HT

 # 2. calcul du prix TTC brut

 # 3. application éventuelle de la réduction

 # 4. calcul de l'arrondi

 # 5. affichage du résultat

[pour la réduction, remarquer qu'il y a écrit **au-delà de** et non **à partir de** dans l'énoncé, soit pour $n > s$ et non $n \geqslant s$]

3. L'exemple puNaTTC (3)

Pour aider aussi l'enseignant, il serait bien que **toutes les variables** à utiliser en programmation **soient déjà nommées** dans l'énoncé de l'exercice, soit, au choix,

- le texte complémentaire :

  on nommera dans l'algorithme nbArt le nombre n d'articles, prixuHT le prix unitaire p...

- ou le tableau :

énoncé	*algorithme*
n	*nbArt*
p	*prixuHT*

...

sachant qu'au brouillon on utilise des noms courts pour les variables.

3. L'exemple puNaTTC (4)

Enfin, il serait utile de disposer de **valeurs d'entrées et de sorties** prévues pour les deux cas au minimum à tester : **n** $\leqslant$ **s** et **n** $>$ **s** afin de vérifier qu'on prend bien en compte la réduction.

Pour s'assurer que la séquentialité **calcul puis arrondi** est bien prise en compte, il faudrait aussi avoir un exemple où appliquer l'arrondi avant le calcul de la réduction donne un résultat différent du bon résultat.

Comme cela peut demander un peu de réflexion et de préparation, **G-ALG** fournit un catalogue d'exercices "déjà pensés" avec des valeurs judicieuses en entrée et en sortie, ce qui permet leur **validation**.

Du coup, l'étudiant(e) peut rapidement s'entrainer et **s'auto-évaluer** sur de **nombreux exemples courts**. De plus la progression dans les variantes proposées de résolution doit l'amener à réfléchir sur la façon de coder.

3. L'exemple puNaTTC (5)

Ici, par exemple, en mode **validation**, **G-ALG** teste 6 cas dont le numéro 6 suivant :

```
<execution numero="6">
     <entrees>
        <variable nom="nbArt"    valeur ="52"    />
        <variable nom="prixuHT"  valeur= "10.95" />
        <variable nom="tauxTAXE" valeur="12.87"  />
        <variable nom="seuilArt" valeur="10"     />
        <variable nom="tauxRed"  valeur="12.87"  />
     </entrees>
     <sorties>
        <!-- 559.9686 arrondi, 12.87 deux fois (volontaire) -->
        <variable nom="prixFinal" valeur="559.97" />
     </sorties>
  </execution>
```

4. galg, le langage de G-ALG (1)

galg, le langage utilisé, est **contraignant** sur certains aspects :

- il s'écrit avec des mots-clés **en français** ;
- les points-virgules sont interdits et les parenthèses aussi ;
- il faut un commentaire avec **auteur** en début d'algorithme ;
- chaque fin de stucture doit être commentée ;
- le mot **affecter** est obligatoire ;
- l'imbrication de calculs et fonctions est limitée ;
- les chaînes de caractéres sont uniquement définies via le guillemet ;
- les indices de tableau sont limités à une variable simple ;

...

4. galg, le langage de G-ALG (2)

Mais ce langage, **totalement syntaxique**, est aussi "de haut niveau" donc parfois assez **"permissif"** sur d'autres aspects :

- tout ce qui s'écrit algébriquement est acceptable (mais peut-être pas traduisible ou exécutable) : ainsi, une suite d'instructions peut aboutir à

  ```
  AFFECTER nbP <-- "A" + 2
  ```

  ce qui ne pose pas de problème algorithmique *a priori* ;
- l'instruction **APPELER** permet de simuler/provoquer des initialisations... ;
- **galg** autorise toutes les constructions, tant qu'on peut remplacer une expression par un appel de fonction avec des arités cohérentes.

4. galg, le langage de G-ALG (3)

Voici donc ce qui produit des **erreurs** en **galg** :

- Exemple 1 d'erreur

```
    } ;
  ) } ; } ; }
```

- Exemple 2 d'erreur

```
SI a>b ALORS x <-- 1 FIN_SI
```

- Exemple 3 d'erreur

```
idLig <-- paste("LIGNE",sprintf("%03d",1:nrow(df)),sep="")
```

- Exemple 4 d'erreur

`date()` et `date("i")` dans le même algorithme

4. galg, le langage de G-ALG (4)

Par contre, **galg** accepte sans erreur les instructions suivantes :

- ```
  ECRIRE "Le",date(),"à",heure("s"),"au revoir", maju(id)
  ```

- ```
  AFFECTER nombre1 <-- entierAuHasard()
  AFFECTER nombre2 <-- entierSemiQuadratique(alpha)
  AFFECTER nombre2 <-- n*(n+1)*(2*n+1)/6
  ```

- ```
  affecter V <-- tableauAuHasard(n,0,20)
  appeler afficheTableau(V)
  ```

- ```
  ouvrir nomFic1 en_ecriture comme fic1
  ecrire concatene(" (fic1) " , ligne1) sur fic3
  ecrire format(nbl2,4) , " lignes lues sur " , nomFic2
  appeler afficheFichier(nomFic3)
  ```

5. galgi, l'interface de démonstration de G-ALG

http://forge.info.univ-angers.fr/~gh/Galg/Interface/

6. À l'aide !

Des **volontaires** seraient bienvenu(e)s pour :

- **rajouter des exercices** (énoncé, modèle de validation) ;
- **construire la base de données** des progressions pour la validation ;
- intégrer **G-ALG** dans *Moodle* ;
- **tester et améliorer** le code, l'interface, le langage...

Par avance, **merci** pour votre aide.

Merci de votre attention !