Bases de la programmation (été 2018/19)
Repères pour se préparer à l'interrogation

Il y aura 5 problèmes dont la plupart seront composés de sous-problèmes. Parmi ces sous-problèmes, 2 ou 3 auront un caractère pratique et vaudront environ 1/3 du nombre de points total ; le reste aura un caractère théorique.

A. Gamme des problèmes théoriques

Les problèmes marqués avec ✶ concernent seulement les interrogations de rattrapage.

Les nombres entre crochets correspondent aux numéros des diapositifs (selon la numérotation en bas) où il faut chercher les informations (correspondant aux problèmes énumérés ci-dessous et non pas toute l'information trouvée sur un diapositif donné). N.B. Ces numéros sont donnés à titre indicatif pour faciliter l'apprentissage ; si vous ne trouvez pas l'information sur le diapositif donné, signalez-le à l'enseignant au lieu d'omettre le problème en cause.

I. Notions de base

1. ✶ Définitions : problème, énoncé, algorithme, programme, codification. [8, 11, 12]
2. ✶ Schéma du processus de la programmation. [12]
3. ✶ Langages bas-niveau et haut-niveau : caractéristiques (surtout les différences). [14]
4. ✶ Langages compilés et interprétés : schéma d'exécution d'un programme source, bons et mauvais côtés des langages interprétés. [17]
5. ✶ Cycle de vie d'un programme. [18]

II. Données

1. Les trois rôles possibles des données. [21]
2. Définition « vulgaire » de la donnée ; constantes et variables. [24]
3. Types fondamentaux des données ; représentation numérique de valeurs pour les types non-numériques (exemples). [26]
4. Types composés des données : définitions, exemples ; indices d'un tableau ; saisie d'une chaîne de caractères et son interprétation comme un tableau. [28, 33]

III. Opérateurs

Pour tous les opérateurs mentionnés ci-dessous il faut : (a) connaître leurs symboles en langage Matlab ; (b) comprendre l'opération réalisée ; (c) savoir déterminer le résultat d'une séquence d'opérations donnée.

1. Affectation. [13, 35]
2. Opérateurs arithmétiques (sans le calcul matriciel en Matlab et sans les opérateurs composés avec affectation). [37]
3. Opérateurs de relation (sans le calcul matriciel en Matlab). [67, 69]
4. Opérateurs logiques (traitement des opérandes standard seulement). [68, 69]

IV. Fonctions

1. Déclaration des fonctions (y compris arguments et résultats) en Matlab : avec arguments mais sans résultat (renvoyé) ; avec arguments et un résultat. [45, 54–55]
2. Portée des variables : variables locales et globales. Espaces de variables : définition ; espace de base et espace d'une fonction en Matlab ; moments de création et de suppression ; placement de variables. Exemples à réanalyser et comprendre. [46–53]
3. Appel des fonctions. Mécanismes et formes du passage des arguments (explicite et implicite) et du renvoi du résultat. Valeurs par défaut. Exemples à réanalyser et comprendre. [48–52, 56–60]

V. Structures de contrôle

1. Structures de contrôle : but, classification. [63]
2. Structures de répétition – boucles (pour, tant que, jusqu'à ce que) : code Matlab et effet ; comment sont effectuées les répétitions (passages) ; quelle est la condition de terminaison et comment elle est codifiée ; critères du choix entre les trois structures. Exemples à réanalyser et comprendre (sans les versions d'un vrai programmeur). [64–66, 76, 79–80, 82]
3. Structures de condition – alternatives : code Matlab et fonctionnement ; codification des conditions. Structure si/sinon : simple, avec alternative, condition multiple. Structure selon/cas (négliger break) : les raisons et les conditions de son application. Structures avec alternatives : conditions considérées et ignorées lors d'exécution d'une structure donnée. Exemples à réanalyser et comprendre. [70–72, 74–75]
4. ✶ Instructions de terminaison. Exemples à réanalyser et comprendre. [77–79]
5. ✶ Structures imbriquées : niveau plus haut et plus bas, ordre d'ouverture et de fermeture ; exemples. [73]

VI. Problèmes avancés

1. Stockage des données. Représentation des données numérique (décimale, binaire, hexadécimale) et textuelle ; unités de quantité de données ; taille de la mémoire nécessaire. Fichiers : définition, rôle, gestion, modes d'accès. Fichiers binaires et texte : différences, bons et mauvais côtés. Exemples à réanalyser et comprendre. [84–85, 91–94, 102]
2. Algorithmes. Tri à bulles : description (idée), code en pseudo-code ou en Matlab. Exemple à réanalyser, comprendre et reproduire (pour n'importe quelle autre liste). Complexité algorithmique : temporelle et spatiale, en moyenne et dans le pire cas ; notation de Landau ; cas pratiques pour de différentes fonctions de borne (temps d'exécution seulement) ; complexités favorables et défavorables ; analyse des algorithmes : sommation, recherche d'une valeur et tri à bulles (meilleur et pire cas seulement). Idées de meilleurs algorithmes du tri. [104–115]
3. Critères de qualité des programmes : en quoi consistent et avec quels moyens on peut les achever ; exemple à réanalyser et comprendre. [118–137]
4. ✶ Paradigmes de programmation : définition et caractéristiques ; en quoi consistent les principaux paradigmes (sans énumérer les langages associés) ; exemples à reconnaître et classifier. [138–144]
5. ✶ Erreurs d'exécution : définition, gestion d'erreurs, structure essaye/attrape, erreur et avertissement, exemples à réanalyser et comprendre. [152–154]

B. Exemples des problèmes pratiques

Les trois genres des problèmes pratiques présentés ci-dessous sont les seuls à attendre mais avec de différents opérateurs, structures/commandes de contrôle et algorithmes simples (valeur minimale/maximale, position d'une valeur, somme/moyenne d'éléments etc.)

- Étant donné que : a = 5, b = −2, c = 0, t = 0, quel sera le résultat de l'expression suivante ?

  ```
  (t<a) & ((t<b) | (t==c))
  ```

  Réponse : vrai *ou* 1

  L'essentiel, c'est comprendre les opérateurs (ici : logiques et de relation).

- Définissez une fonction qui calculerait et renverrait la valeur de l'expression algébrique « sin ωt », où *sin* est la fonction sinus, ω et t sont des paramètres. Pour calculer le sinus, la fonction s'appelle « sin » en Matlab.

  Solution :

  ```
  function res=monsin(omega,t)
    res=sin(omega*t);
  endfunction
  ```

  Ce qui compterait le plus dans l'évaluation, serait :

  - l'en-tête de la fonction où les arguments et le résultat sont correctement définis ;
  - dans le code de la fonction (ici une ligne seulement), usage correct des noms des variables (arguments, résultat) en accord avec l'en-tête ;
  - l'usage correct de la fonction *sin*, avec son argument entre les parenthèses et une variable affectée avec son résultat en utilisant l'opérateur correct (=) ;
  - l'application de l'opérateur arithmétique correct pour la multiplication (*) ;
  - réalisation d'une fonction qui *prend* ses *arguments* et *renvoie* son *résultat*, comme demandé dans le problème posée (à ne pas confondre avec une demande à l'utilisateur en ligne ni avec un affichage sur l'écran).

- La fonction *bas* est définie comme suit (*numel* renvoyant le nombre d'éléments d'un vecteur) :

  ```
  function bb=bas(vect)
    bb = vect(1);
    for i = [2:numel(vect)]
      if (vect(i) < bb)
        bb = vect(i);
      endif
    endfor
  endfunction
  ```

  Quel sera la valeur de la variable *b* après l'exécution des instructions suivantes ? Quelle tâche est-ce que cette fonction accomplit ?

  ```
  A = [1 8 5]
  b = 10*bas(A)
  ```

  Réponse : b = 10. Cette fonction détermine et renvoie la valeur la plus petite dans un vecteur.

  L'essentiel, c'est comprendre les structures de contrôle (en analysant les exemples provenant de la conférence et des TP), y compris les conditions, les répétitions introduites par les boucles, les méthodes d'appeler les fonctions et de se servir de leurs résultats.