Algorithmique et Programmation – Examen (durée 1h30)

Première session du 15 mai 2025

Notes : Aucun document autorisé. Sont interdits les calculatrices, les téléphones, ainsi que tout autre ustensile de calcul ou de communication.

Exercice 1 : Questions de connaissances générales (5 points)

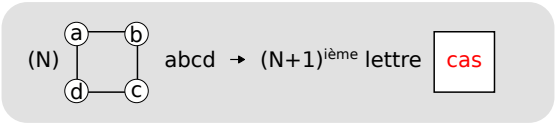
5 questions

Exercice 2 : Trois fonctions et un programme à compléter (6 points)

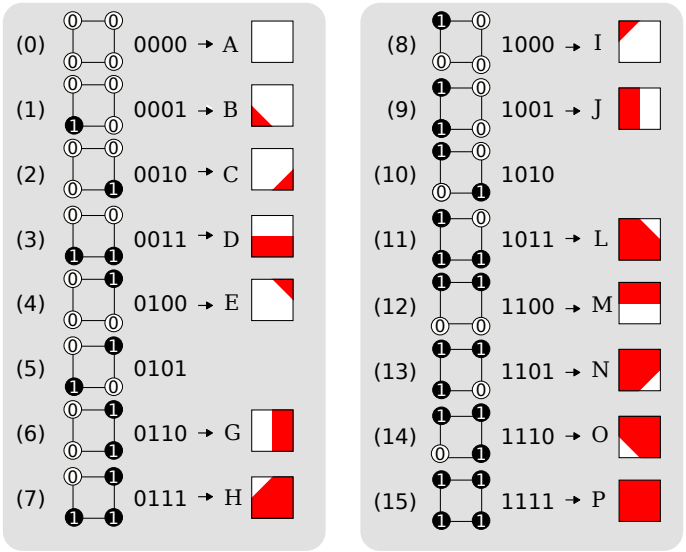
L'algorithme des Marching Cubes, publié en 1987 à la conférence ACM SIGGRAPH [1], fut créé pour reconstituer une forme 3D à partir de l'acquisition de quadrillages de points, par exemple dans le contexte médical (ex : IRM). Une façon de l'aborder plus facilement est d'étudier l'algorithme des Marching Squares, qui en est une version 2D. Cet exercice propose d'exécuter une version simplifiée des Marching Squares (càd une version sans pondération et ignorant deux cas prêtant à ambiguïté).

Le principe de l'algorithme des Marching Squares est d'examiner une grille de points 2D noirs et blancs obtenus par numérisation d'une forme dans le plan. Les points blancs sont situés à l'extérieur de la forme et les points noirs à l'intérieur. L'algorithme procède en interprétant la configuration de chaque carré constitué par quatre points par une forme 2D approximative, selon 16 cas possibles déterminés en fonction des couleurs des quatre coins (a, b, c, d). Chaque cas N (de 0 à 15) est associé à une forme 2D précalculée (en rouge), sauf pour deux cas ambigus (5 et 10) que nous ignorons volontairement dans cet exercice :

Légende :



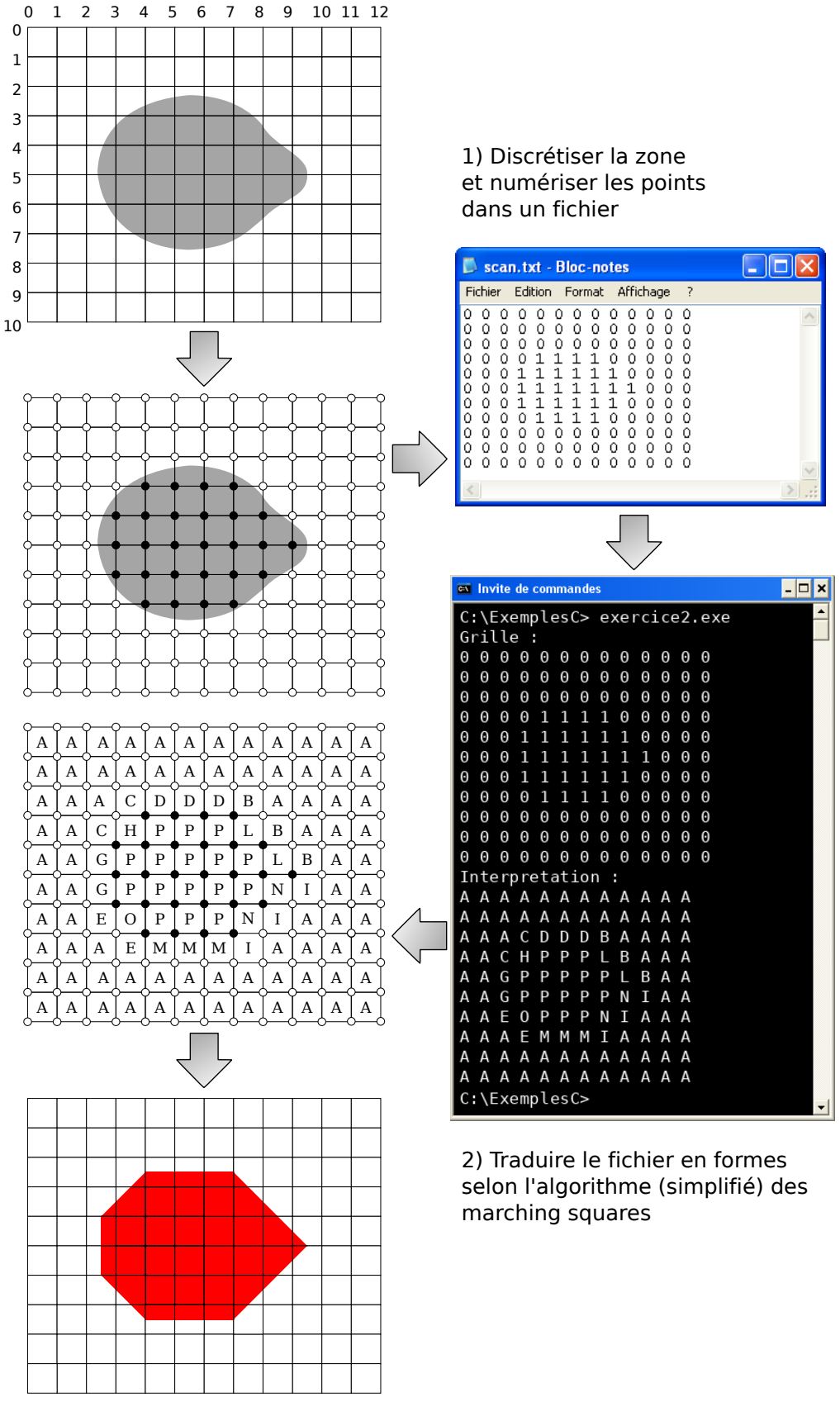
Les 16 cas selon la configuration de carré :



Pour pouvoir traiter ce sujet depuis un programme simple (càd sans graphismes), nous avons fait correspondre chaque cas à une lettre majuscule (càd de `'A'` à `'P'`, en ignorant `'F'` et `'K'`).

1. William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. SIGGRAPH Comput. Graph. 21, 4 (July 1987), 163–169. https://doi.org/10.1145/37402.37422

Cette correspondance sera utilisée dans notre programme pour afficher le résultat de l'algorithme, calculé à partir des points 2D lus dans un fichier texte. Le déroulement de l'algorithme, avec notre programme d'interprétation des configurations par des lettres, est illustré ci-dessous par un exemple sur une forme ovoïdale (en gris) :



L'exécution de notre programme commence donc par lire la grille depuis un fichier texte. Ce fichier texte contient des chiffres 0 ou 1, séparés par des espaces, sur 11 lignes de 13 colonnes. Ensuite, l'exécution du programme affiche d'abord la grille qui a été lue en mémoire, puis affiche le résultat de son interprétation.

Nous considérons que le nombre de lignes (H) et le nombre de colonnes (W) du dispositif d'acquisition sont définis par deux macro-constantes au début du fichier :

```
#include <stdio.h>
#include <stdlib.h>
#define W 13
#define H 11
```

1) Complétez ci-dessous le code de la fonction en langage C qui permet la lecture ASCII d'une grille depuis un fichier texte dont le nom est donné en paramètre :

```
void read (char grille[W][H], char *filename) _
  FILE* desc _
  int c, i=0 _

  ____ = fopen(filename, "r") _

  __ (____ == NULL) _
    perror("Error, cannot open file in read mode") _
    exit(-1) _
  _

  _____ (!feof(____)) _
    fscanf(____, "%d", &_) _
    grille[ _ % _ ][ _ / _ ] = c _
    _++ _
  _

  fclose(____) _
_
```

2) Complétez ci-dessous le code de la fonction en langage C qui permet d'afficher le contenu d'une grille :

```
void show (char grille[W][H]) _
  int i, j _

  ___ (_=0 _ _<_ _ _++) _
    ___ (_ = 0 _ _ < _ _ _ ++) _
      ______("%d ", grille[i][j]) _
    _
    ______("__") _
  _
_
```

3) Complétez ci-dessous le code de la fonction en langage C qui permet d'interpréter une grille et d'afficher la forme 2D correspondante selon le codage alphabétique :

```
void interpretation(char grille[W][H]) _
  int cas, a, b, c, d _
  int i, j _

  ___ (j=1 _ j<H _ j++) _
    ___ (i=1 _ i<W _ i++) _
      a = ______[___][___] _
      b = ______[_][___] _
      c = ______[_][_] _
      d = ______[___][_] _
      cas = _ * a + _ * b + _ * c + d _
      ______ ("%c ", 'A' + ___) _
    _
    ______("__") _
  _
_
```

4) Complétez ci-dessous le code de la fonction principale de notre programme en langage C qui charge une grille depuis le fichier "scan.txt", qui affiche la grille, puis affiche son interprétation selon l'algorithme des Marching Squares, en utilisant les trois fonctions précédentes.

```
___ ____ () _
  ____ ______[_][_] _

  ____(______, __________) _

  ______("Grille :\n") _
  ____(grille) _

  ______("Interpretation :\n") _
  ______________(______) _

  ______ 0 _
_
```

Exercice 3 : Une fonction et deux programmes à écrire (9 points)

Nous proposons d'amorcer la réalisation d'un outil d'assistance aux calculs de planification de chantiers. Un artisan doit préparer ses chantiers de ravalements de façades. Ces chantiers comprennent quatre phases : installation, nettoyage, revêtement et désinstallation. Sur des roulements de 10 semaines, les phases de nettoyage des chantiers en cours doivent cumuler au plus 20 jours entre les différents chantiers (c'est-à-dire 4 semaines, pour des semaines de 5 jours). Une journée de travail dure de 8 H 00 à 16 H 00 et permet **8 heures** de travail.

Le diagnostic d'un chantier permet de calculer la **surface** de façade (en m^2) et de classer le travail en trois **catégories**, selon le type de revêtement, son âge, son état et le nombre d'étapes qui seront nécessaires :

- `'A'` : nettoyage rapide = 20 minutes par m^2
- `'B'` : nettoyage lent = 30 minutes par m^2
- `'C'` : nettoyage multiple = 40 minutes par m^2

Aussi, la durée du nettoyage varie selon le **taux d'acide** qui sera ajouté par volume de décapant. Cependant, l'artisan souhaite calculer au plus juste le taux qui sera nécessaire pour rester en-dessous de 20 jours de nettoyage cumulés sur ses chantiers. Le coefficient multiplicateur suivant est alors appliqué sur la durée du chantier estimée en minutes : 10 / taux.

Ainsi, l'estimation de la durée d'un chantier se calcule comme suit :

— *Traitement (en minutes) = 20 si catégorie `'A'`, 30 si catégorie `'B'`, 40 si catégorie `'C'`*

— *Durée (en minutes) = surface * traitement * 10 / taux*

Enfin, pour estimer le nombre de jours occupés par un chantier, l'artisan applique les contraintes d'organisation suivantes :

— Une heure entâmée est ignorée tant que la demi-heure n'est pas dépassée (càd inférieur à 30 minutes)

— Le dépassement d'une demi-heure (càd 30 minutes strictement dépassées) compte comme une heure pleine

— Une demi-journée commencée (càd inférieur à 4 heures) compte comme une demi-journée (càd 4 heures)

— Une demi-journée dépassée (càd 4 heures strictement dépassées) compte comme une journée pleine (càd 8 heures)

Notation : *Il vous est possible de traiter les trois questions suivantes de ce sujet de deux manières, soit en utilisant le compilateur ci-dessous (via les boutons "Vérifier"), soit sur feuille (copies doubles fournies). En revanche, le compilateur attribue les points uniquement si l'exécution est correcte. Vous pouvez exécuter autant de fois que vous le souhaitez sans perte de points.*

a) Pour commencer, nous écrivons une fonction en langage C qui permet d'estimer la durée d'un chantier (en minutes), en fonction de la surface de façade (en m^2), de la catégorie du chantier (`'A'`, `'B'` ou `'C'`) et du taux d'acide ajouté au décapant.

Le prototype de la fonction sera le suivant :

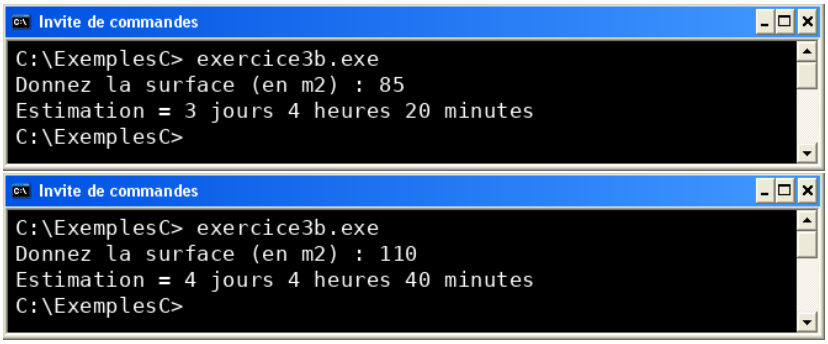
```
int calculer_duree (int surface, char categorie, int taux) ;
```

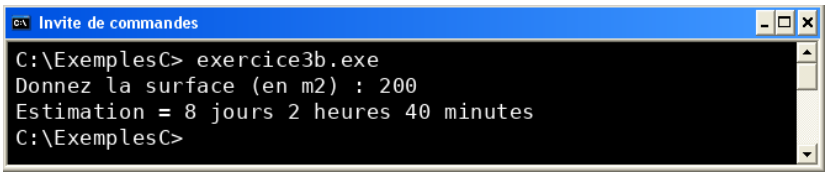
Écrivez ci-dessous la déclaration de votre fonction, puis vérifiez son exécution par un programme de tests (déjà écrit).

(Vous pouvez exécuter autant de fois que vous le souhaitez sans perte de points.)

b) Pour tester la fonction du (a), nous écrivons maintenant un programme en langage C qui affiche la durée estimée en minutes d'un chantier de catégorie `'A'` et un taux d'acide valant 10 par volume, et dont la surface est demandée à l'utilisateur.

Ci-après trois exemples d'exécution de ce programme :





Écrivez ci-dessous le code du programme, puis vérifiez son exécution (une version déjà prête de la fonction sera ajoutée automatiquement lors du test, donc vous écrivez ici uniquement le programme demandé).

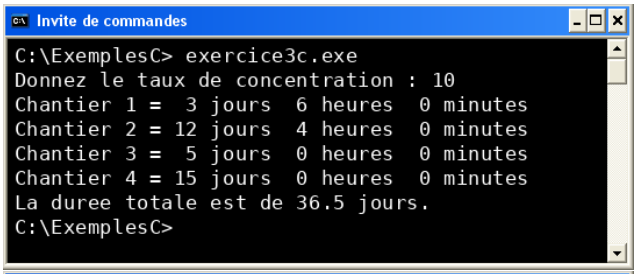
(Vous pouvez exécuter autant de fois que vous le souhaitez sans perte de points.)

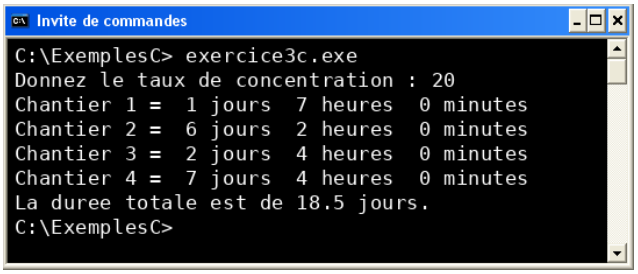
c) Pour amorcer le développement du logiciel de planification, nous écrivons maintenant un programme en langage C qui compte le nombre de jours cumulés par quatre chantiers, dont les surfaces et catégories diagnostiquées sont enregistrées dans les deux tableaux suivants :

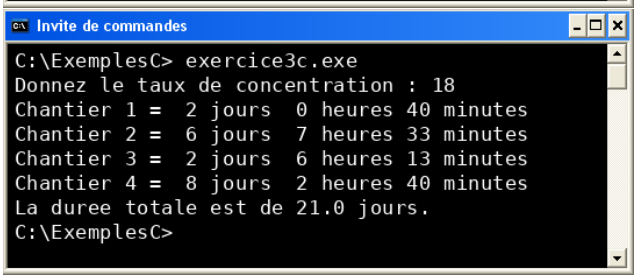
```
int surfaces[4] = { 90, 200, 120, 180 } ;
int categories[4] = { 'A', 'B', 'A', 'C' } ;
```

Le programme commencera par demander à l'utilisateur de saisir le taux d'acide, avant d'afficher ensuite la durée de chaque chantier, puis le nombre de jours cumulés. Ce programme doit lui aussi utiliser la fonction du (a).

Ci-après trois exemples d'exécution de ce programme :







Écrivez ci-dessous le code du programme, puis vérifiez son exécution (une version déjà prête de la fonction sera ajoutée automatiquement lors du test, donc vous écrivez ici uniquement le programme demandé).

(Vous pouvez exécuter autant de fois que vous le souhaitez sans perte de points.)