3ETI, Examen [CSC2]
Developpement Logiciel en C
CPE Lyon

2012-2013 (1ere session)
durée 3h

Tous documents et calculatrices autorisés.

Le sujet comporte 11 pages

Le temps approximatif ainsi que le barème sont indiqués pour les grandes parties. Notez que le barème est donné à titre purement indicatif et pourra être adapté par la suite.

En cas de doute sur la compréhension de l'énoncé, explicitez ce que vous comprenez et poursuivez l'exercice dans cette logique.

Note préliminaire:

- Toutes les questions concernent le language de programmation C dans le cadre du developpement de logiciels sur architecture PC standard.
- Nous vous invitons à commenter le code et les réponses. En particulier, en cas d'ambiguité de compréhension d'une question, ajoutez toutes remarques et illustrations supplémentaires permettant d'expliquer votre démarche.
- Sauf mention contraire explicite, on supposera que l'on dispose d'un système Linux standard fonctionnant correctement sur un PC récent 32 ou 64 bits (identiques aux conditions de TP des PC de CPE: *int* étant encodé sur 4 octets, pointeurs étant encodés respectivement sur 4(/8) octets sur 32(/64) bits).
- On supposera que le code C est compilé avec une version récente de gcc sous la norme C99 (ou ultérieure) identique aux conditions de TP des PC de CPE.
- On supposera dans chaque cas que les `#include` des en-tête standards nécessaires à la bonne compilation et exécution des programmes décrits sont correctement installés et appelés (ex. stdio, stdlib, string, math, etc).

1 Syntaxe de base C

1.1 Types de bases

[15min max] 1.5 points

Question 1 *Pour chacune des définitions suivantes, donnez le type de variable C permettant de définir l'entité désignée dans la mémoire:*

1. *Un nombre entier positif ou négatif pouvant aller jusqu'à 10000 au minimum.*
2. *Un nombre entier strictement positif encodé sur un seul octet.*
3. *Un nombre à virgule flottante en simple précision.*
4. *Un tableau statique de 5 entiers strictement positifs. Chaque entier pouvant aller jusqu'à 10000 au minimum.*
5. *Un pointeur sur un caractère.*
6. *Un pointeur sur une variable dont le type n'est pas explicité.*
7. *Un pointeur sur un caractère, et dont le caractère pointé ne peut pas être modifié.*

Cas d'exemple: Pour un nombre à virgule flottante en double précision:
réponse attendue: *double*

1.2 Calculs flottants

[10min max] 1 point

Soit le programme suivant:

```
int main()
{
  float a=1.1;
  float b=0.3;
  float c=1.4;

  if(a+b==c)
    printf("Oui\n");
  return 0;
}
```

Ce programme n'affiche pas de manière certaine et portable le mot *Oui* à l'écran.

Question 2 *Expliquez pourquoi.*

Question 3 *Modifiez la ligne correspondant à* `if(a+b==c)` *de manière à obtenir le comportement souhaitée mais en étant plus robuste.*

2 Manipulation de données et méthodologie de développement

2.1 Généralités de programmation

[15min max] 1.5 points

2.1.1 Developpement logiciel

Repondez en quelques lignes de manière la plus précise possible.

Question 4 *Dans le cadre du developpement d'un logiciel, quels sont les critères principaux d'un code de bonne qualité ?*

Question 5 *Qu'est ce que* svn *ou* git *? Quels sont leurs utilités (une seule réponse attendue pour les deux)?*

2.1.2 Système de fichier Linux

Vous téléchargez un fichier d'internet correspondant à un programme executable.

Question 6 *Quelle commande devez vous taper en ligne de commande pour donner les droits d'execution à ce fichier. On supposera que le fichier se nomme* `pgm` *?*

2.2 Passage d'arguments et méthodologie

[20min max] 2 points

Soit la fonction suivante:

```
void fois_deux(float *valeur)
{
  *valeur=2 * (*valeur);
}
```

Soit les 2 programmes suivants utilisant la fonction `fois_deux`:

[Programme 1]

```
int main()
{
  float a=1;

  fois_deux(&a);
  printf("%f",a);

  return 0;
}
```

[Programme 2]

```
int main()
{
  float *a;
  *a=1;
  fois_deux(a);
  printf("%f",*a);

  return 0;
}
```

Question 7 *Parmi les deux programmes précédents, lequel permet d'afficher le nombre 2 à l'écran de manière certaine?*

Question 8 *Expliquez pourquoi l'autre programme est incorrect.*

Question 9 *Ecrivez le contrat correspondant à la fonction correcte.*

Question 10 *Dans le cadre d'une programmation défensive, quels assertions pourrait-on ajouter dans la fonction correcte?*

2.3 Manipulation de données

2.3.1 Enumeration

[10min max] 1 point

Soit le programme suivant:

```
enum type_alcool {whisky,vodka,rhum,tequila};

int main()
{
  enum type_alcool alcool_a=whisky;
  enum type_alcool alcool_b=rhum;

  printf("%d,%d\n",alcool_a,alcool_b);

  return 0;
}
```

Question 11 *Une fois executé, qu'affiche ce programme à l'écran?*

Soit la fonction `alcool_identique()` qui prend en paramètre d'entrée deux enums de type `type_alcool`. La fonction renvoie 1 si les alcools sont identiques et 0 sinon.

Question 12 *Donnez la* **signature** *(=en-tête) de la fonction* `alcool_identique()`.

Question 13 *Ecrivez le* **corps** *complet de la fonction* `alcool_identique()`.

2.3.2 Manipulation d'adresse

[10min max] 1.5 points

Soit le programme suivant:

```
enum type_alcool {whisky,vodka,rhum,tequila};
struct bouteille
{
  enum type_alcool type;
  int volume;
};
#define MAX_BOUTEILLE 10
struct bar
{
  struct bouteille T[MAX_BOUTEILLE];
};

void ma_fonction(struct bar* p_bar_courant);

int main()
{
  struct bar bar_courant;
  ma_fonction(&bar_courant);

  return 0;
}
```

On s'interesse par la suite à compléter la fonction `ma_fonction` qui recoit en paramètre d'entrée un pointeur vers la variable `bar_courant`. On suppose dans la suite que l'on écrit du code **dans** la fonction `ma_fonction`.

```
void ma_fonction(struct bar* p_bar_courant)
{
  // On place le code ici ...
}
```

Question 14 *Creez une variable de type* `struct bar` *contenant une* **copie** *du* `bar_courant` *(notez que cette variable doit être du type* `struct bar` *et non* `struct bar*`*).*

Question 15 *Ecrivez le code C permettant d'afficher à l'écran le volume totale de vodka que possède le bar (somme de tous les volumes de bouteilles contenant de la vodka).*

Question 16 *Creez une variable de type* `struct bouteille*` *contenant l'***adresse** *pointant vers la 4ème bouteille de* `bar_courant`*.*

Question 17 *Creez une variable de type* `enum type_alcool*` *contenant l'***adresse** *pointant vers le type d'alcool de la 5ème bouteille de* `bar_courant`*.*

2.4 Chaine de caractères

[15min max] 1.5 points

```
int main()
{
  char nom[]="Antoine";
  char prenom[]="Dupont";

  printf("%d\n",strlen(nom));

  char nom_complet[X1];
  strcpy(nom_complet,nom);
  nom_complet[X2]='-';
  strcpy(nom_complet+X3,prenom);

  printf("%s\n",nom_complet);

  return 0;
}
```

Question 18 *Quel est la taille du tableau statique* nom *? C'est à dire: combien de cases mémoires sont utilisées?*

Question 19 *Que doit afficher la ligne* `printf("%d\n",strlen(nom))` *à l'écran ? (la page* `man` *de strlen est disponible en annexe à la dernière page de cet énoncé).*

Question 20 *Par quels nombres doit-on compléter X1, X2, et X3 pour que la ligne* `printf("%s\n",nom_complet)` *affiche à l'écran* `Antoine-Dupont` *?*

3 Compilation et système Linux

[20min max] 2 points

Dans cette section, on suppose les deux fichiers suivants contenant du code C. Ces fichiers sont situés dans le même répertoire. Vous disposez d'une ligne de commande situé dans ce répertoire.

[Fichier F1.h]

```
#ifndef F1_H
#define F1_H

#define MACHINE_A

#define MAX_NAME 20
struct chocolat
{
  int pourcentage_cacao;
  char marque[MAX_NAME];
};

#endif
```

[Fichier F1.c]

```
#include "F1.h"

#ifdef MACHINE_A
#define NBR 6
#else
#define NBR 4
#endif

#include "F1.h"

int main()
{
  struct chocolat boite[NBR];

  int k=0;
  for(k=0;k<NBR;k++)
    {
      boite[k].pourcentage_cacao
         =60;
      strncpy(boite[k].marque,
           "NESTLE",MAX_NAME);
    }

  return 0;
}
```

Question 21 *Quelle(s) commande(s) devez-vous taper pour compiler ce code en un executable du nom de* `pgm` *?*

Question 22 *Lors de l'étape de pré-processeur, par quoi est remplacé* `MAX_NAME` *?*

Question 23 *Lors de l'étape de pré-processeur, par quoi est remplacé* `NBR` *?*

Question 24 *Quelle est l'utilité des instructions* `#ifndef F1_H` *et* `#define F1_H` *dans le fichier d'en-tête ?*
Le programme donné compilerait-il sans ces deux instructions dans ce cas particulier ? Pourquoi?

Vous souhaitez générer le fichier objet `F1.o` avec les options de warnings recommandés et le mode debug activé.

Question 25 *Quelle commande devez vous tapez pour générer ce fichier?*

Question 26 *Le fichier* `F1.o` *est-il un fichier executable? Dans le cas général, quel est l'utilité d'un fichier objet? (2-3 lignes d'explications sont attendues).*

4 Problèmes: Conversion d'un fichier de données

[1h max] 8 points

4.1 Problématique

Supposons que nous possédons deux capteurs (température et pression) enregistant les valeurs relevées à différents instants. Les capteurs sont interfacés à un PC et écrivent les résultats relevés dans un fichier commun `data.txt`.

Un exemple de fichier de données enregistrées est le suivant:

[Fichier data.txt]

```
17:32 temperature 14.5
18:25 temperature 13.3
20:00 pression 1018
20:15 temperature 11.6
20:38 temperature 9.4
21:00 pression 1012
21:19 temperature 8.8
```

On suppose que le fichier possède **toujours** ce type de syntaxe avec l'heure de la mesure, le type de capteur (temperature ou pression), et la valeur relevée. On ne s'interessera pas à traiter de manière robuste le cas où les fichiers d'entrées sont corrompus.

Nous souhaitons écrire un programme qui viens convertir ce fichier texte en un autre où l'on écrira les valeurs compatible avec une syntaxe de type Matlab (/ou Scilab, Octave) de manière à pouvoir analyser ces résultats ou tracer des courbes.
Par exemple, le fichier d'exemple devra être convertit en un fichier `valeurs.txt` contenant:

[Fichier valeur.txt]

```
Temperature=[14.500000,13.300000,11.600000,9.400000,8.800000];
dateTemperature=[1052,1105,1215,1238,1279];
Pression=[1018.000000,1012.000000];
datePression=[1200,1260];
```

On notera que

- Les valeurs de pressions et de températures sont séparées et stockées dans des tableaux.
- Les dates correspondantes sont stockées à la suite dans des tableaux différents et correspondent aux **nombres de minutes** écoulées depuis minuit.

Vous êtes chargé d'écrire le programme de conversion en C. L'approche suivante est retenue:

1. Lecture des données à partir du fichier d'entrée et stockage temporaire des valeurs dans la RAM.
2. Ecriture des données dans le fichier de sortie suivant une syntaxe de type Matlab.

4.2 Structs mises en pratique

Dans une logique de developpement logiciel, vous préparez un ensemble de structures qui vont acceuillir les données lues à partir du fichier d'entrée. Voici les `struct` définies:

```
#define MAX_EVENEMENT 10 //nbr d'evenement max par jour

//une struct contenant une heure precise (heure+minute)
struct conteneur_temps
{
  int heure;
  int minute;
};

//un evenement de type temperature ou pression
struct evenement
{
  struct conteneur_temps temps;  //heure de l'evenement
  float valeur; //donnee de temperature ou pression
};

//un conteneur pour un ensemble de prises de temperatures ou de pressions
struct tableau_evenement
{
  struct evenement donnees[MAX_EVENEMENT];
  int nombre_evenement_enregistres;
};
```

Le programme principal d'appels aux fonctions de lecture et d'ecriture est le suivant:

```
int main()
{
  //noms des fichiers
  const char* nom_fichier_entree="data.txt";
  const char* nom_fichier_sortie="valeurs.txt";

  //gestion d'erreur
  if(fichier_est_accessible_lecture(nom_fichier_entree)==0 ||
     fichier_est_accessible_ecriture(nom_fichier_sortie)==0)
    {
      printf("Fichiers %s ou %s non accessibles\n",
             nom_fichier_entree,
             nom_fichier_sortie );
      exit(1);
    }


  //lecture
  struct tableau_evenement temperature;
  struct tableau_evenement pression;
  lecture_fichier(nom_fichier_entree,&temperature,&pression);

  //ecriture
  ecriture_fichier(nom_fichier_sortie,&temperature,&pression);

  return 0;
}
```

4.2.1 Verification d'accessibilité d'un fichier

La fonction `fichier_est_accessible_lecture()` permet de vérifier si un fichier est accessible en lecture. Le principe de vérification est le suivant:

1. On vérifie que l'ouverture du fichier désigné (commande `fopen()`) en mode `lecture` ne renvoie pas d'erreur.
2. On vérifie que la fermeture du fichier désigné (commande `fclose()`) ne renvoie pas d'erreur.

Le contrat de la fonction est le suivant:

```
/**
   Fonction de verification qu'un fichier passe en parametre est accessible
       (en lecture).

   Prerequis:
   - un pointeur constant non NULL contenant le nom du fichier a verifier

   Garanties:
   - Renvoie 0 si le fichier designe est non accesible en lecture.
   - Renvoie 1 si le fichier designe est accessible en lecture.
 */
```

Question 27 *Ecrivez la signature (=en tête) de la fonction* `fichier_est_accessible_lecture`*.*

Question 28 *Donnez le code correspondant à cette fonction qui vérifie le contrat donné.*

Question 29 *Donnez un exemple de tests unitaires à réaliser (au moins 2 tests) pour vérifier que votre fonction possède le comportement attendu.*

Question 30 *Quelle ligne de votre implémentation (=corps de la fonction) serait à modifier dans le cas de la fonction* `fichier_est_accessible_ecriture` *?*

4.2.2 Lecture d'un fichier

Le contrat et la signature de la fonction de lecture du fichier d'entrée sont les suivants:

```
/**
   Fonction de lecture du fichier designe

   Prerequis:
   - un pointeur constant non NULL contenant le nom d'un fichier accessible
       a lire
   - un pointeur non constant et non NULL vers un tableau statique d'
      evenements pour les prises de temperatures (taille du tableau >= 10)
   - un pointeur non constant et non NULL vers un tableau statique d'
      evenements pour les prises de pressions (taille du tableau >= 10)

   Garantie:
   - Remplit les tableaux de temperature et de pression avec les valeurs
      correspondantes lues dans le fichier tant que celui-ci contient la
      syntaxe attendue. (le cas de syntaxe non attendue n'est pas traitee
      specifiquement)

 */
void lecture_fichier(const char* filename,
                     struct tableau_evenement* temperature,
                     struct tableau_evenement* pression);
```

Question 31 *Pourquoi les pointeurs* `temperature` *et* `pression` *sont non constants?*

Question 32 *Quelles assertions pouvez vous réaliser en début de cette fonction afin de satisfaire à une* programmation défensive*?*

On rappelle que la lecture d'un fichier ASCII formaté se réalise avec la commande `fscanf`. On utilisera la syntaxe `fscanf(fid,chaine,arg...)`, avec

- `fid`: le descripteur de fichier.
- `chaine`: la chaine de caractere du format de fscanf (par exemple ``` ``%d %f'' ``` indique un entier, un espace suivit d'un flottant).
- `arg...`: les arguments de fscanf à compléter.

Un rapel d'utilisation de quelques arguments de `fscanf()` est également disponible en annexe de cet énoncé (dernière page).

Question 33 *Quel format donnez vous à* `chaine` *dans le cas du fichier courant ? De combien de variables allez vous avoir besoin dans* `arg...` *?*

Question 34 *Que retourne* `fscanf` *? En vous servant de cette valeur de retour, définissez un critère permettant de savoir que l'on a atteint la fin du fichier ?*

Question 35 *Ecrivez* **l'algorithme** *général de lecture du fichier et du stockage des variables.*

Question 36 *Ecrivez le corps de la fonction de lecture du fichier correspondant à la fois au contrat donné et à l'algorithme que vous avez écrit.*

4.2.3 Ecriture d'un fichier

Question 37 *Ecrivez le contrat de la fonction d'ecriture du fichier.*

Question 38 *Ecrivez* **l'algorithme** *général de l'écriture du fichier.*

Question 39 *Ecrivez le corps de la fonction d'ecriture du fichier correspondant à votre contrat et votre algorithme.*
(Note: Vous détaillerez la syntaxe d'écriture de la température et des dates associées, mais pourrez sauter la partie d'écriture correspondant à la pression et à ses dates au besoin).

4.2.4 Tests d'intégrations

Question 40 *Proposez une méthodologie de tests permettant de vérifier l'intégrité de votre programme complet de manière la plus automatique possible.*

A Page man de strlen

```
STRLEN(3)                  Linux Programmer's Manual                  STRLEN(3)
NAME
       strlen - calculate the length of a string
SYNOPSIS
       #include <string.h>

       size_t strlen(const char *s);
DESCRIPTION
       The  strlen() function calculates the length of the string s,
       excluding the terminating null byte ('\0').
RETURN VALUE
       The strlen() function returns the number of bytes in the string s.
```

B Rappel d'utilisation de fscanf

Rappel de quelques arguments passés à `fscanf()` (similaire à `printf()`)

```
//ouverture d'un fichier valide
FILE *fid=NULL;fid=fopen(filename,"r");

int a=0; fscanf(fid,"%d",&a); //lecture d'un entier
float b=0; fscanf(fid,"%f",&b); //lecture d'un flottant
//lecture d'une chaine de caractere d'au plus 10 caracteres
char buffer[10]; fscanf(fid,"%10s",buffer);

//lecture multiple dans un fichier contenant la ligne suivante:
//il y a 6 pommes et 2 cerises
int nombre_pommes=0;
char buffer_cerise[8];
fscanf(fid,"il y a %d pommes et 2 %8s\n",
                    &nombre_pommes,
                    buffer_cerise);
```