22.25/34

4.5

Nom et prénom : Saraiva Maia Leandro

Travail écrit no. 2

(Durée : 60 minutes)

Consignes :

- **ECRIVEZ VOS REPONSES DIRECTEMENT SUR CE DOCUMENT**
- Seules les réponses figurant sur ce document seront corrigées
- Une écriture lisible est exigée
- Vous pouvez écrire au crayon papier
- Ne pas dégrafer ce document
- Aucune documentation n'est autorisée, hormis le résumé C figurant à la fin de ce document

(3)

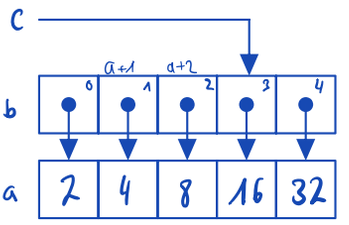
Question 1 (5 points)

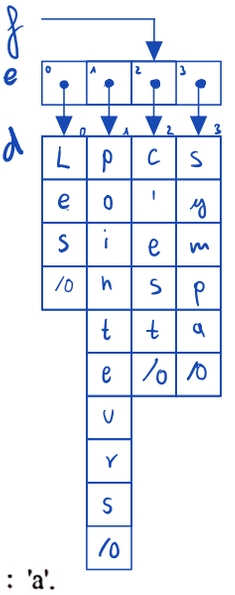
Soient les déclarations suivantes :

```
int a[] = {2, 4, 8, 16, 32};
int* b[] = {a, a+1, a+2, a+3, a+4};
int** c = b + 3;

const char* d[] = {"Les", "pointeurs", "c'est", "sympa"};
const char** e[] = {d, d+1, d+2, d+3};
const char*** f = &e[2];
```

Quelle valeur fournit chacune des expressions ci-dessous ?
(Conseil : Aidez-vous d'un petit dessin)





1) `**b+1`
2) `b[1][1]`
3) `--**c++`
4) `c[0][-1]`
5) `*--c-*(b+1)`
6) `**d`
7) `*++d[3]`
8) `*e[1]`
9) `*--*(e+1)+1`
10) `*(f[-1][3]+2)`

IMPORTANT

- Pour les caractères, donner le résultat entre apostrophes. Exemple : 'a'.
- Pour les chaînes de caractères, donner le résultat entre guillemets. Exemple : "ABC".
- Les 10 expressions sont supposées être évaluées les unes après les autres, en commençant par la no 1). Le résultat d'une certaine expression peut donc dépendre de ce qui s'est produit lors de l'évaluation des expressions précédentes.

Vos réponses :

1) 3 V	6) 'L' V
2) 8 V	7) 'y' V
3) 15 V	8) ~~'p'~~ "pointeurs"
4) ~~8~~ 15	9) ~~77~~ "es"
5) 2 V	10) ~~indéfini~~ 'p'

2.75

Question 2 (5 points)

La fonction *difference* proposée ci-dessous comporte **5 erreurs**.
Compléter le tableau ci-dessous en indiquant pour chaque erreur le numéro de la ligne problématique et en proposant un correctif complet écrit en C.

IMPORTANT

- Une réponse avec no de ligne problématique correct mais sans proposition de correctif sera considérée comme fausse. Idem si le correctif proposé est faux.
- Aucune ligne de code ne doit être ajoutée, supprimée ou déplacée.

```
// Retourne un tableau comprenant tous les éléments de tab1 qui ne figurent pas
// dans tab2.
// Retourne NULL si la mémoire est insuffisante pour créer le tableau solution.
// Remarques :
// 1) La fonction présuppose que ni tab1, ni tab2 ne sont NULL.
//    Le cas échéant, la fonction provoque une assert error à l'exécution
// 2) La fonction présuppose que taille1 <= INT_MAX
//    Le cas échéant, la fonction provoque une assert error à l'exécution
// 3) Le premier élément du tableau solution contient le nombre d'éléments
//    contenus dans ce dernier.

 1 int** difference(const int* tab1, size_t taille1,
 2                  const int* tab2, size_t taille2) {
 3    assert(tab1 != NULL && tab2 != NULL);
 4    assert(taille1 <= INT_MAX);
 5    int* tab = (int*) malloc(taille1 + 1, sizeof(int));
 6    if (tab) {
 7       *tab = (int) taille1;
 8       memcpy(tab + 1, tab1, taille1);
 9       for (size_t i = 0; i < taille2; ++i) {
10          size_t j = 0; // Cette ligne ne contient pas d'erreur
11          for (size_t k = 1; k <= (size_t) *tab; ++k) {
12             if (tab[k] != tab2[i]) {
13                tab[j++] = tab[k];
14             }
15          }
16          *tab = (int) j;
17       }
18       realloc(tab, (size_t) *tab * sizeof(int));
19    }
20    return tab;
21 }
```

t1 { A, B, C }
t2 { A, D, E, C }
tab { 1, B }

Vos réponses :

Erreur	No Ligne	Correctif C
1	1	int* difference(const int* tab1, size_t taille1, ✓
2	5	plutôt utiliser calloc que malloc ✓
3	11	for (size_t k=1; k<=(size_t)*tab; ++k){
4	13	tab[j++] = tab[k]; ! 3/4
5	18	tab = realloc... = ++j.

6

Question 3 (6 points)

Pour chacune des suites d'instructions ci-dessous, indiquer dans la colonne de droite du tableau si celle-ci est ***juste*** ou ***fausse*** (c'est-à-dire compile ou pas).

IMPORTANT

- Il n'est PAS demandé de justifier vos réponses ou de proposer un quelconque correctif
- Barème appliqué : +0.5 point pour une réponse correcte; 0 point en cas d'absence de réponse; –0.5 point pour une réponse incorrecte

Instructions	Juste ou faux ?

`struct S {`
`  char c;`
`  float x = 1.5f;`
`} s;` — faux
	Faux ✓

`struct S {`
`  const char C;`
`  float x;`
`} s;`
`s = (struct S) {'A', 1.5f};` — faux
	Faux ✓

`typedef struct {`
`  char c;`
`  int n;`
`} S;`
`S s1 = {'A', 1}, s2 = {65, 1};`
`printf("s1 == s2 ? %s\n", (s1 == s2 ? "oui" : "non"));` — faux
	Faux ✓

`typedef struct {`
`  char c;`
`  static int n;`
`} S;`
`S s = {.c = 'A', .n = 1};` — faux
	Faux ✓

`struct S {`
`  const char C;`
`  float x;`
`} s = {'A', 1.5f};`
`printf("%c %f\n", s.C, s.x);` — juste
	juste ✓

`typedef struct {`
`  char c;`
`  float x;`
`} S;`
`S s1;`
`S s2 = s1 = (struct S) {.c = 'A', .x = 1.5f};` — faux
	Faux ✓

`typedef struct {`
`  struct T* a;`
`} S;`
 	
`struct T {`
`  S* b;`
`};`
	juste	juste ✓

`struct S {`
`  enum E {A, B = -1, C = 0} e;`
`} s;`
`s.e = E::B;`
`printf("%d\n", s.e);`
	faux	faux ✓

`typedef struct {`
`  T* a;`
`} S;`
 	
`typedef struct {`
`  S* b;`
`} T;`
	faux	faux ✓

`typedef union {`
`  int n;`
`  union V {`
`  char c;`
`  float x;`
`  } v;`
`} U;`
`U u = {{.x = 1.5f}};`
	faux	Faux ✓

`char s1[4];`
`const char* s2 = "ABC";`
`s1 = s2;`
	faux	Faux ✓

`typedef char Chaine[];`
`Chaine s1 = "ABC";`
`const char* s2 = s1;`
`printf("%s\n", s2);`
	juste	juste ✓

4.5

Question 4 (9 points)

a) **(4 pts)**

1.5

La fonction *adrDernierCaract* ci-dessous est censée, étant donnée une chaîne de caractères *s* passée en paramètre, renvoyer, via le paramètre *adr*, **l'adresse** du dernier caractère effectif de la chaîne *s* (*NULL* si *s* est *NULL* ou vide).

```
void adrDernierCaract(char* s, char* adr) {
   if (s == NULL && *s == '0') {
      adr = NULL;
   }
   adr = s[strlen(s)];
}
```

La fonction ci-dessus contient toutefois diverses erreurs ou maladresses.

Réécrire INTEGRALEMENT le code proposé de manière à corriger toutes les erreurs / maladresses qu'il contient.

IMPORTANT

- Le type de retour de *adrDernierCaract* (*void*) ne doit en aucun cas être modifié
- Des points seront décomptés si une modification, au lieu de corriger une erreur, en ajoute une supplémentaire

Votre réponse :

```
Void adrDernierCaract(char* s, char* adr) {
   if (s == NULL || *s == '\0') {
      adr = NULL;
      return;
   }
   adr = s + strlen(s) - 1;
}
```

b) **(5 pts)**

3

Sachant que la gravité g à la surface d'une planète est donnée par la formule :

$$g = \frac{Gm}{r^2} \; [m\,s^{-2}]$$

où :
- G = constante de gravitation universelle = $6.6743 \cdot 10^{-11}$ [$m^3\,kg^{-1}\,s^{-2}$]
- m = masse de la planète [kg]
- r = rayon de la planète [m]

compléter la partie notée *<à compléter>* dans le code ci-dessous (et rien d'autre !) de telle sorte que celui-ci compile sans warnings et affiche à l'exécution :

```
gravite sur Mercure = 3.7
gravite sur Terre = 9.8
gravite sur Uranus = 8.87
```

IMPORTANT

- A noter que le code proposé ci-dessous n'importe ni <math.h>, ni <string.h> !
- Le code à compléter ne doit comporter aucune boucle

```
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>

double gravite(const char* s) {
   <à compléter>
}

int main(void) {
   typedef enum {MERCURE, TERRE, URANUS} Planetes;
   const char* const PLANETES[] = {"Mercure", "Terre", "Uranus"};
   // masse [kg] et rayon [km] des diverses planètes
   const char* const MASSE_ET_RAYON[] = {"3.303E23 2440.5",
                                         "5.976E24 6378",
                                         "8.686E25 25560"};
   for (Planetes p = MERCURE; p <= URANUS; ++p) {
      printf("gravite sur %s = %.3g\n",
             PLANETES[p], gravite(MASSE_ET_RAYON[p]));
   }
   return EXIT_SUCCESS;
}
```

double gravite = 6,6743 ;
double masse, rayon ;
sscanf ("%lf %lf", &masse, &rayon) ;
return (gravite * masse) / (rayon * rayon)

Votre réponse :

```
double masse, rayon;
sscanf(s, "%lE%lf", &masse, &rayon);
return((6.6743e-11)* masse)/(rayon * rayon);
```

à définir comme cte

6

Question 5 (9 points)

3 1) **(2 pts)**
On suppose disposer d'un programme C, appelé *myprog.c*, dont l'exécutable se nomme *myprog.exe*, et ayant pour entête du *main* :

```
int main(int argc, char* argv[]) { ... }
```

argc = 7
argv = 8

On suppose maintenant exécuter la ligne de commande suivante :

```
myprog test.dat -opt1=10 src 10 -O2 20
1      2        3        4   5   6  7
```

a) Que vaut *argc* ?

Votre réponse : 7 ✓

b) Combien *argv* compte-t-il d'éléments ?

Votre réponse : 8 ✓

3 2) **(2 pts)**
Soit la déclaration suivante :

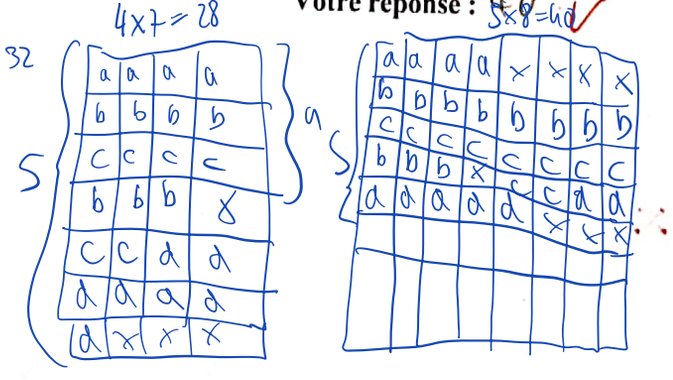
```
typedef struct {
   struct {
      uint32_t a;
      size_t b;
      char* c;
   } a;
   unsigned char b[3];
   uint16_t c;
   uint8_t d[7];
} S;
```

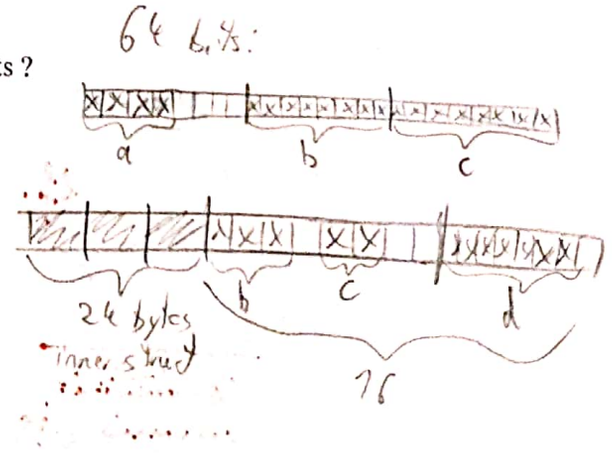
a) Que vaut sizeof(S) en architecture 32 bits ?

Votre réponse : 28 ✓

b) Que vaut sizeof(S) en architecture 64 bits ?

Votre réponse : ~~48~~ 40 ✓





3) (5 pts)

Soit le code suivant, censé modéliser des singes.

```
 1 #include <inttypes.h>
 2 #include <stdbool.h>
 3 #include <stdio.h>
 4 #include <stdlib.h>
 5
 6 #define NB_MAX_FRUITS 3
 7
 8 typedef enum {BANANE, ORANGE, POMME, PAMPLEMOUSSE} Fruit;
 9 typedef enum {CHIMPANZE, GORILLE} Espece;
10
11 const char* const ESPECES[] = {"Chimpanze", "Gorille"};
12 const char* const FRUITS[] = {"Banane", "Orange", "Pomme", "Pamplemousse"};
13
14 typedef struct {
15    const char* nom;
16    struct {
17       struct {
18          uint16_t poids; // [kg]
19          struct {
20             Fruit fruits[NB_MAX_FRUITS];
21          } fruitsPreferes;
22       } gorille;
23       struct {
24          uint8_t age;
25          bool estAgressif;
26       } chimpanze;
27    } specificites;
28 } Singe;
29
30 void afficher(const Singe* s,
31               const char* const especes[],
32               const char* const fruits[]);
33
34 int main(void) {
35
36    Singe chita = <à compléter 1>;
37    Singe bobo  = <à compléter 2>;
38
39    afficher(&chita, ESPECES, FRUITS);
40    afficher(&bobo, ESPECES, FRUITS);
41
42    return EXIT_SUCCESS;
43 }
44
45 void afficher(const Singe* s,
46               const char* const especes[],
47               const char* const fruits[]) {
48 // code volontairement omis ici mais supposé correct et opérationnel
49 }
50
```

a) Le code proposé ci-dessus (hormis bien sûr les deux parties <à compléter>) comporte **3 erreurs.**
Indiquer pour chaque erreur où celle-ci se situe et proposer un correctif écrit en C.

2

Vos réponses :

1) à la ligne 76, utiliser une union en lieu et place d'un struct
Sinon on utilise de la mémoire inutilement

2) On doit ajouter l'attribut Espece espece dans le struct Singe
Sinon on ne peut pas distinguer différents types de singe.

3) struct inutile, on pourrait juste utiliser un Tableau

Quelle ligne ? Soyez plus précis

b) En supposant le point a) résolu, compléter les deux parties notées ***<à compléter>***, de telle sorte à déclarer les deux singes suivants :

0

- "Chita", un chimpanzé de 12 ans, non agressif
- "Bobo", un gorille de 250 [kg] dont les fruits préférés sont les bananes et les pommes

IMPORTANT
Les déclarations des deux singes doivent être implémentées chacune **de la manière la plus courte possible**[1].
[1] Attention! La présence de toutes les paires d'accolades est toutefois requise.

Vos réponses :

<à compléter 1 >

```
{ "Chita", { .chimpanze = {12, False}}}
```

manque tsp

<à compléter 2 >

```
{ "Bobo", { .gorille = { 250, { POMME, BANANE }}}}
```

faux

manque tsp