Contrôle Exemple de C (Correction à la fin)

Ce sujet est donné à but d'entrainement, les questions et exercices du contrôle peuvent être plus nombreux.

Une correction est fournie en fin de ce document.

Pour toute question ou remarque relative aux cours/tps/sujets : pierre.bertin-johannet@proton.me

Exercice 0: QCM

Plusieurs réponses possibles, pas de points négatifs.

Pour accéder au deuxième élément du tableau T, j'ecris :

1. t[1]
2. (*t) + 2
3. *(t + 1)
4. (*t)[2]

Les variables en C

1. Utilisent de la mémoire réservée uniquement pour la fonction dans laquelle elles sont déclarées.
2. Sont enregistrées dans le **tas**.
3. Existent pour une durée définie.
4. Ont une valeur par défaut si elles ne sont pas initialisées.

Les arguments d'une fonction en C:

1. Sont copiés dans la mémoire de la fonction appelée.
2. Sont des pointeurs vers les variables de la fonction appelante.
3. Sont stockés dans la **pile**.
4. Sont déclarés en même temps que la fonction.

Une structure en C:

1. Peut servir à contenir plusieurs variables.
2. Peut changer de taille lors de l'execution du programme.
3. Peut etre créé avec le mot clé `struct` .
4. Peut etre enregistrée sur le **tas**

Laquelle de ces variables utilise le plus de mémoire ?

1. `int a[2]`
2. `int (\*b)(char\*)`
3. `char* c[4]`
4. `char** d[2]`

Quel est le résultat de l'opération suivante:

`(1 & 2) + (3 && !0)`

1. 3.
2. 2.
3. 1.
4. 0.

La chaine de charactères `"Bonjour"`

1. Occupe 7 octets en mémoire.
2. Occupe 8 octets en mémoire.
3. Occupe 14 octets en mémoire.
4. Occupe 15 octets en mémoire.

Une socket sous linux

1. Correspond à un port physique sur la carte réseau.
2. Est identifiée par un numéro.
3. Permet de communiquer directement avec le driver.
4. Peut être configurée pour utiliser un protocole particulier.

Pour allouer de la mémoire sur le tas je peux utiliser:

1. malloc.
2. dealloc.
3. alloca.
4. memalloc.

La taille mémoire d'un int est

1. 4 octets.
2. 2 octets.
3. n'est pas fixe.
4. 8 octets.

La taille mémoire d'un char est

1. 1 octets.
2. 2 octets.
3. n'est pas fixe.
4. 4 octets.

La taille d'une union

1. Dépend de l'attribut le plus petit.
2. Dépend de la somme de la taille des attributs.
3. Dépend de l'attribut le plus grand.
4. Dépend du nom de l'union.

Que dire d'un programme qui essaye d'accéder à l'élément 5 dans un tableau de taille 2:

1. Le programme crash immédiatement.
2. Le programme accède toujours à la mémoire, même si elle n'est pas réservée.
3. Le programme contient une faille de type buffer-overflow.

Exercice 1

Complétez les types manquants dans ces fonctions.

```
..... join(..... str1, ...... delim, ..... str2){
    ...... len1 = strlen(str1);
    ...... len2 = strlen(str2);
    ...... ret = malloc(sizeof(......) * (len1 + len2 + 2))
    ...... count = 0;
    for (count = 0; count < len1; count++){
      ret[count] = str2[count];
    }
    ret[count] = delim;
    count++;
    for (...... i = 0; i < len2; i++){
      ret[count + i] = str2[i];
    }
    ret[count + len2] = 0;
    return ret;
}
```

```
..... raccourcir(..... tableau, ..... nouvelle_taille){
  ..... nouveau_tab = malloc(sizeof(float) * nouvelle_taille);
  for (..... i = 0; i < nouvelle_taille; i++){
    nouveau_tab[i] = (*tableau)[i];
  }
  free(*tableau);
  *tableau = nouveau_tab;
}
```

Exercice 2

Que va afficher le code suivant dans la console ?

```
void a(int n, int *b){
    n = *b;
}
void b(int* n){
    *n = 2;
}
void c(int n, int k){
    n = n + k;
}
void d(int n, int* k){
  c(a(n, &k), b(&n));
}

int main(){
    int i = 1;
    int j = 2;
    int k = 3;
    a(k, &i);
    b(&i);
    c(j, k);
    printf("%d, %d, %d\n", i, j, k);
}
```

Exerice 3.

On utilisera la structure suivante dans l'exercice :

```
struct User{
  int age;
  char* name;
}
```

1. Ecrivez le code d'une fonction qui demande dans la console les informations d'un utilisateur, les enregistre dans une struct User et les renvoie (On considère que l'utilisateur ne saisira jamais un nom de plus de 10 charactères).
2. Ecrivez le code d'une fonction qui accepte en argument un entier n, demande les informations de n utilisateurs et les enregistre dans un tableau avant de le renvoyer.
3. Ecrivez une fonction qui accepte un entier n ainsi qu'un tableau d'utilisateurs de taille n en argument et affiche les noms de tous les utilisateurs.
4. Créez une fonction main qui utilise les fonctions précédentes pour d'abord demander une liste d'utilisateurs et ensuite les afficher dans la console. Toute la mémoire allouée devra être libérée avant la fin de l'execution du programme.

Exercice 4

1. Quels sont les deux problèmes de gestion mémoire dans la fonction ci-dessous ?

```
int somme_tab(int n, int* values){
  int somme = 0;
  for (int j = 0; j < n; j++){
    somme +=values[j];
  }
  return somme;
}

int calcul_somme(){
    for (int i = 0; i < 5; i++){
        printf("entrez un nombre");
        int n;
        scanf("%d\n", &n);
        int* values = malloc(n);
        for (int j = 0; j < n; j++){
          scanf("%d", values + j);
        }
        printf("somme %d \n", somme_tab(n, values));
    }
}
```

Exercice 5

On considère dans le code ci dessous que la socket est correctement configurée pour du TCP.

1. Complétez le code ci dessous afin qu'il :
    - Passe la socket en mode écoute.
    - Initie un handshake tcp.
    - recoive jusqu'a 100 octets de données.
    - renvoie ces données reçue à l'expéditeur.
    - retourne le nombre d'octets reçus.

```
int echo(int socket_ecoute) {
  listen(.....);
  .....
  .....
  ..... = read(.....);
  send(...);
  return .....
}
```

Exercice 6

Soit le code page suivante:

1. Qu'affiche le code si on entre les lignes suivantes ?

   > 5

   > "bonj"

   > 7, 10

2. Il est possible de faire crasher le programme.
    - Expliquez pourquoi.
3. Il est possible d'afficher une partie du mot de passe d'un utilisateur précédent.
    - Expliquez pourquoi.
4. Expliquez comment corriger le code.

```c
char* demande_text(char* what, int n){
  char name[n];
  printf("Entrez votre %s\n", what);
  for (int i = 0; i < n - 1; i++){
    scanf("%c", name + i);
  }
  name[n - 1] = 0;
  return name;
}

void calcul_somme(int start, int end) {
  char tab[end];
  int somme = 0;
  for (int i = start; i < end; i++){
    tab[i] = i;
  }
  for (int i = start; i < end; i++){
    somme += i;
  }
  return somme;
}

int main() {
  int n;
  while (1) {
    printf("quelle est la longueur de votre mot de passe ?\n");
    scanf("%d", &n);
    char* pwd = demande_text(n);
    int start, end;
    printf("entrez les entiers pour le calcul\n");
    scanf("%d, %d", &start, &end);
    int resultat = calcul_somme(start, end);
    printf("votre mdp est %s et le calcul a donné %d", pwd, resultat);
  }
}
```

Correction

Exercice 0: QCM

Pour accéder au deuxième élément du tableau T, j'ecris :

1. **t[1]**
2. (*t) + 2
3. ***(t + 1)**
4. (*t)[2]

Les variables en C

1. **Utilisent de la mémoire réservée uniquement pour la fonction dans laquelle elles sont déclarées**
2. Sont enregistrées dans le **tas**
3. **Existent pour une durée définie**
4. Ont une valeur par défaut si elles ne sont pas initialisées

Les arguments d'une fonction en C:

1. **Sont copiés dans la mémoire de la fonction appelée**
2. Sont des pointeurs vers les variables de la fonction appelante
3. **Sont stockés dans la pile**
4. **Sont déclarés en même temps que la fonction**

Laquelle de ces variables utilise le plus de mémoire ?

1. `int a[2]` -> 2 ints
2. `int (\*b)(char\*)` -> un pointeur
3. **`char* c[4]` -> 4 pointeurs**
4. `char** d[2]` -> 2 pointeurs

Quel est le résultat de l'opération suivante:

```
(1 & 2) + (3 && !0)
```

1. 3
2. 2
3. **1** -> (0) + (1)
4. 0

La chaine de charactères `"Bonjour"`

1. Occupe 7 octets en mémoire
2. **Occupe 8 octets en mémoire**
3. Occupe 14 octets en mémoire
4. Occupe 15 octets en mémoire

Une socket sous linux

1. Correspond à un port physique sur la carte réseau
2. **Est identifiée par un numéro**
3. Permet de communiquer directement avec le driver
4. **Peut être configurée pour utiliser un protocole particulier**

Pour allouer de la mémoire sur le tas je peux utiliser:

1. **malloc**
2. dealloc
3. alloca
4. memalloc

La taille mémoire d'un int est

1. 4 octets
2. 2 octets

3. **n'est pas fixe**
4. 8 octets

La taille mémoire d'un char est

1. **1 octet**
2. 2 octets
3. n'est pas fixe
4. 4 octets

La taille d'une union

1. Dépend de l'attribut le plus petit
2. Dépend de la somme de la taille des attributs
3. **Dépend de l'attribut le plus grand**
4. Dépend du nom de l'union

Que dire d'un programme qui essaye d'accéder à l'élément 5 dans un tableau de taille 2:

1. Le programme crash immédiatement
2. Le programme accède toujours à la mémoire, même si elle n'est pas réservée
3. **Le programme contient une faille de type buffer-overflow**

Exercice 1

```
char* join2 (char* str1, char delim, char* str2){
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    char* ret = malloc(sizeof(char) * (len1 + len2 + 2));
    int count = 0;
    for (count = 0; count < len1; count++){
      ret[count] = str2[count];
    }
    ret[count] = delim;
    count++;
    for (int i = 0; i < len2; i++){
      ret[count + i] = str2[i];
    }
    ret[count + len2] = 0;
    return ret;
}

void raccourcir(float** tableau, int nouvelle_taille){
  int nouveau_tab = malloc(sizeof(float) * nouvelle_taille);
  for (int i = 0; i < nouvelle_taille; i++){
    nouveau_tab[i] = (*tableau)[i];
  }
  free(*tableau);
  *tableau = nouveau_tab;
}
```

Exercice 2

Les fonctions c et a ne modifient rien car les arguments sont copiés, la fonction d ne modifie que des variables locales, on affiche donc 2, 2, 3

Exerice 3.

```
struct User ask_user(){
  struct User user;
  user.name = malloc(11);
  scanf("%d", &(user.age));
  scanf("%s", user.name);
  scanf("%f", &(user.height));
  return user;
}

struct User* fill_users(int* n){
  printf("how many \n");
  scanf("%d", n);
  struct User* users = malloc(sizeof(struct User) * *n);
  for (int i = 0; i < *n; i++){
      users[i] = ask_user();
  }
  return users;
}

void print_users(struct User* users, int nb_users){
  for (int i = 0; i < nb_users; i++){
      printf("name : %s, height : %f, age: %d\n", users[i].name, users[i].height,
users[i].age);
  }
}

int main(){
  int n;
  struct User* users = fill_users(&n);
  print_users(users, n);
  free(users);
}
```

Exercice 4

1. La fonction calcul_somme ne libère pas la mémoire après l'avoir alloué, cela cause une fuite mémoire.
2. L'appel à la fonction malloc alloue seulement n octets pour n `int`, la taille d'un int peut être superieure à un octet, il faudrait prendre en compte la taille d'un int ainsi: `malloc(n*sizeof(int))`.

Exercice 5

```
int echo(int socket_ecoute) {
  listen(socket_ecoute);
  // on récupère la socket créée par le handshake
  int socket_connectee = accept(socket_ecoute);
  // on réserve 100 octets pour les données
  char data[100];
  // on lit les données
  int compte_recues = read(socket_connectee, data, 100);
  // on renvoie les même données
  send(socket_connectee, data, 100);
  // on retourne le compte
  return compte_recues;
}
```

Exercice 6

1. Votre mdp est bonj et le calcul a donné 8
2. Le programme alloue dynamiquement des tableaux d'une taille définie par l'utilisateur sur la pile, la pile ayant une taille limitée, un grand nombre causerait une erreur de type **stackoverflow**.

3. Les fonctions `demande_text` et `calcul_somme` renvoient un pointeur vers leur variable locale (name et tab respectivement).

   Une modification de la variable tab permettrait alors de remplacer le zéro qui termine la chaine `pwd` et d'afficher ce qui est ecrit après dans la mémoire.

   On pourrait ainsi écrire un petit mot de passe, remplacer le charactère de fin par un nombre et printf afficherait les charactères de la fin du mot de passe d'un précédent utilisateur.
4. Pour corriger cela, on remplace l'allocation sur la pile par une allocation sur le tas. Il faut aussi penser à libérer la mémoire des deux tableaux en utilisant la fonction free.

```c
char* demande_text(char* what, int n){
  char* name = malloc(n);
  printf("Entrez votre %s\n", what);
  for (int i = 0; i < n - 1; i++){
    scanf("%c", name + i);
  }
  name[n - 1] = 0;
  return name;
}

void calcul_somme(int start, int end) {
  char* tab = malloc(end);
  int somme = 0;
  for (int i = start; i < end; i++){
    tab[i] = i;
  }
  for (int i = start; i < end; i++){
    somme += i;
  }
  free(tab);
  return somme;
}

int main() {
  int n;
  while (1) {
    printf("quelle est la longueur de votre mot de passe ?\n");
    scanf("%d", &n);
    char* pwd = demande_text(n);
    int start, end;
    printf("entrez les entiers pour le calcul\n");
    scanf("%d, %d", &start, &end);
    int resultat = calcul_somme(start, end);
    printf("votre mdp est %s et le calcul a donné %d", pwd, resultat);
    free(pwd);
  }
}
```