ArcSys1: Systèmes et Réseaux

Examen de C 2023 (1h 30)

Tous documents interdits à l'exception d'un A4 recto manuscrit de votre main et du pense-bête du C fourni en cours. Calculatrices, téléphones, ordinateurs et montres connectées interdites. La correction tiendra compte de la qualité de l'argumentaire et de la présentation. Un code illisible et/ou incompréhensible est un code faux.

★ Exercice 1: Chaînes de caractères et fichiers (6 pts).

▷ **Question 1** (2pt)**:** Écrivez une fonction `char plus_frequent(char* str)`, qui prend en paramètre la chaîne de caractères à analyser, et retourne la lettre la plus fréquente. Par exemple, `plus_frequent("␣Bon␣courage␣!␣␣")` doit retourner `'o'`. Vous pouvez supposer qu'il n'y a pas de lettre accentuée dans le texte.

▷ **Question 2** (2pt)**:** Écrivez un programme complet plus-frequent prenant une chaîne de caractères à analyser en premier argument de la ligne de commande, et affichant le résultat de la manière suivante :

```
$ ./plus-frequent " Bon courage !"
La lettre la plus fréquente est 'o'.
```

Réponse

J'ai choisi d'utiliser un tableau de fréquence trop grand afin que l'indice d'accès soit le code ascii de chaque lettre, car cela simplifie grandement le code. On pourrait argumenter que je gâche environ un kilo-octet de mémoire avec toutes les cases inutilisées, mais je pense que la mémoire n'est pas assez rare pour justifier l'effort. On a évidement le droit de penser autrement, mais les deux solutions rapportent autant de points.

J'utilise une forme avancée pour remplir initialement le tableau de fréquences avec des zéros : `= {0}`. Je ne sais plus si on l'a vu en cours, mais on peut faire une petite boucle pour l'initialisation, ou bien utiliser la fonction `memset`, mais il est important d'initialiser.

S'il y avait des lettres accentuées dans le texte, il faudrait se poser la question de l'encodage utilisé, et se préparer à avoir des problèmes (à écrire du code un peu plus compliqué) si c'est UTF-8.

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

char plus_frequent(char *str) {
  int freq[256] = {0}; // Tableau de fréquences
  char* p = str;
  while (*p != '\0') { // Tant qu'il reste des lettres
    if ((*p>'a' && *p<'z') || (*p>'A' && *p<'Z'))
      freq[*p]++; // On compte ce caractère
    p++; // On passe au suivant
  }
  int mval = 0; // Max dans le tableau de fréquences
  int mpos = 0; // Position du maximal
  for (int pos=0; pos<255; pos++)
    if (freq[pos] > mval) {
      mval = freq[pos];
      mpos = pos;
    }

  return mpos;
}

int main(int argc, char** argv) {
  if (argc<2) {
    fprintf(stderr, "Usage: plus-frequent arg\n");
    exit(1);
  }
  char c = plus_frequent(argv[1]);
  printf("La lettre la plus fréquente est '%c'.\n", c);

  return 0;
}
```

Fin réponse

▷ **Question 3** (2pt)**:** Écrivez un programme somme prenant un nombre arbitraires d'entiers sous la forme d'arguments de la ligne de commandes, et en affiche la somme. Vous pouvez supposer que les paramètres ne sont que des nombres comme attendu. Par exemple :

```
$ ./somme 5 2 11 24
La somme est : 42
```

Réponse

C'est relativement facile quand on se souvient de la fonction `atoi()` ou l'une de ses amies. Les prototypes ne sont pas exigés dans l'examen.

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

int main(int argc, char** argv) {
  int somme = 0;

  for (int i = 1; i<argc; i ++) {
    somme += atoi(argv[i]);
  }

  printf("La somme est %d\n", somme);
}
```

Fin réponse

★ **Exercice 2: Mémoire dynamique : liste simplement chaînée (8pts).**

On souhaite représenter une liste simplement chaînée d'entiers. Chaque cellule de la liste est représenté par une structure C comprenant deux champs: un entier représentant la valeur de cette cellule, et un champ `next` représentant la cellule suivante dans la liste, ou `NULL` pour la dernière cellule de la liste.

▷ **Question 1** (2pt)**:** Écrivez la structure nécessaire, ainsi que lafonction `N` permettant de créer une nouvelle cellule en mémoire dont la valeur et la suite sont passées en paramètre. Donnez ensuite le code permettant de créer la liste dont les valeurs successives sont 27, 14, 34, 66 et 58 (dans cet ordre).

```
struct list * N(int val, struct list * next);
```

Réponse

```c
#include <assert.h>
#include <limits.h> // INT_MIN
#include <stdio.h>
#include <stdlib.h>

struct list {
  int val;
  struct list* next;
};

struct list * N(int val, struct list * next) {
  struct list *res = malloc(sizeof(struct list));
  assert(res != NULL); // Arrête le programme bruyamment en cas de pb

  res->val = val;
  res->next = next;

  return res;
}

void list_affiche(struct list* l) {
  struct list * p = l;
  printf("Liste: ");
  while(p != NULL) {
    printf("%d, ", p->val);
    p = p->next;
  }
  printf("\n");
}

struct list * make_A() { // Fonction retournant A
  return N(27, N(14, N(34, N(66, N(58, NULL)))));
}
```

Fin réponse

▷ **Question 2** (2pt)**:** Écrivez les fonctions `list_len` (retournant la longueur d'une liste), `list_max` (retournant la valeur maximale d'une liste donnée, ou `INT_MIN` si la liste est vide), `list_concat` (modifiant la première liste passée en paramètre pour ajouter la seconde à la fin, et retournant la liste résultante de la concaténation) et `list_free` (libérant la mémoire prise par une liste).

```
int list_len(struct list * l);
int list_max(struct list * l);
struct list * list_concat(struct list * l1, struct list * l2);
void list_free(struct list * l);
```

Réponse

```
int list_len(struct list * l) {
  int res = 0;
  struct list *p = l;
  while (p != NULL) {
    res++;
    p = p->next;
  }
  return res;
}
int list_max(struct list *l) {
  int res = INT_MIN;
  struct list *p = l;
  while (p != NULL) {
    if (res < p->val)
      res = p->val;
    p = p->next;
  }
  return res;
}
struct list * list_concat(struct list *l1, struct list *l2 ) {
  if (l1 == NULL)
    return l2;

  // Cherche la fin (on arrête un cran avant la fin)
  struct list *p = l1;
  while (p->next != NULL)
    p = p->next;

  // concatène
  p->next = l2;

  return l1;
}
void list_free(struct list * l) {
  struct list *p = l;
  while (p != NULL) {
    struct list * temp = p->next;
    free(p);
    p = temp;
  }
}
```

Fin réponse

▷ **Question 3** (2pt)**:** Écrivez la fonction `list_insert` permettant d'ajouter un élément dans une liste triée. On supposera que la liste est triée avant l'insertion, et on veillera à ce que le nouvel élément soit bien placé dans la liste. La fonction retournera le début de la liste modifiée, contenant toutes les cellules pré-existantes plus la nouvelle créée pour l'élément inseré.

```
struct list * list_insert(struct list * l, int val);
```

▷ **Question 4** (2pt)**:** Écrivez la fonction `list_sort` prenant une liste en paramètre et retournant une nouvelle liste triée contenant toutes les valeurs de la liste d'entrée. Vous implémenterez un tri par insertion en parcourant toutes les valeurs de la liste d'entrée et en les insérant les unes après les autres dans la liste résultat en construction.

Réponse

```
struct list * list_insert(struct list *l, int val) {

  if (l == NULL) // Création du premier élément
    return N(val, NULL);

  if (l->val > val) // Ajout en tête
    return N(val, l);

  // Cherche la position
  struct list *p = l;
  while (1) { // boucle infinie dont on sort si l est une liste correcte
    // Le résultat est trié si l'entrée était triée

    if (p->next == NULL) { // En fin de la liste, on ajoute un nouvel élément
      p->next = N(val, NULL);
      return l;
    }
    if (p->next->val > val) { // Point d'insertion trouvé, let's go
      p->next = N(val, p->next);
      return l;
    }
    // On continue
    p = p->next;
  }
  return NULL; // Cela n'arrivera jamais, mais le compilateur exige cette ligne
}
struct list * list_sort(struct list * l) {
  struct list * res = NULL;
  struct list * p = l;
  while (p != NULL) {
    res = list_insert(res, p->val);
    p = p->next;
  }
  return res;
}

int main() {
  struct list * A = make_A();
  list_affiche(A);
  printf("Len: %d ; max: %d\n",list_len(A), list_max(A));

  struct list *B = list_concat(A, N(12, N(22, N(33, NULL))));
  list_affiche( B );

  struct list * triee = list_sort(B);
  list_affiche(triee);

  list_free(B);
  list_free(triee);

  return 0;
}
```

Fin réponse

★ **Exercice 3: Mémoire statique en C (6 pts).**

Tous les programmes de cet exercice compilent et ne provoquent pas d'erreur à l'exécution (même si on utilise `valgrind` ou `gdb`).

fichier1.c

```
1  #include <stdio.h>
2
3  static int i = 6;
4
5  int f() {
6    static int i = 1;
7    return ++i;
8  }
9
10 int g() { return i++; }
11
12 int main(void) {
13   printf("g() 1ere fois %d\n", g());
14   printf("f() 1ere fois %d\n", f());
15   printf("g() 2ieme fois %d\n", g());
16   printf("f() 2ieme fois %d\n", f());
17 }
```

fichier2.c

```
1 #include <stdio.h>
2 int main() {
3   int a[] = {1, 2, 4, 8, 16};
4   int *p[2] = {&a[2], &a[1]};
5   p[0] += *(p[1]);
6   printf("%d\n", **p);
7 }
```

fichier3.c

```
1 #include <stdio.h>
2 int main() {
3   char s[] = "chaine";
4   char *p = s + 1;
5   p++;
6   *p = *s + 1;
7   printf("%s", p);
8 }
```

▷ **Question 1** (2pt)**:** Indiquez la sortie affichée lorsqu'on exécute le programme `fichier1.c`, en justifiant par un schéma mémoire global du processus.

Réponse

Le décorateur `static` sur la variable de la ligne 3 ne change pas le comportement (ça fait que cette variable n'est visible que depuis ce fichier, i.e. pas grand chose dans ce cas).

En revanche, le même mot-clé à la ligne 6 fait que cette variable là est persistante bien qu'elle soit locale. Durée de vie d'une globale, mais visibilité dans le bloc seulement.

Au final, les deux variables ont la même durée de vie qu'une globale. Reste à se souvenir que `i++` retourne la valeur avant l'affectation (le nom de la variable est écrite avant l'incrément) tandis que `++i` retourne la valeur après l'affectation (le nom est après l'incrément). L'affichage est:

```
1 g() 1ere fois 6
2 f() 1ere fois 2
3 g() 2ieme fois 7
4 f() 2ieme fois 3
```

Il faut dessiner les segments data (où se trouve la globale i de la ligne 3 et la locale de la ligne 4 puisqu'elle est marquée `static`), le tas (qui est vide) et la pile. Les cadres de pile sont vides (pas de variable locale).

Fin réponse

▷ **Question 2** (2pt)**:** Discutez de la même façon la sortie du programme `fichier2.c` avec un schéma des variables locales avant et après la ligne 5.

Réponse

Cela affiche 16. L'explication suivante n'est pas exigée, mais dans la correction, il est plus simple de faire un peu de texte qu'un schéma expliquant la même chose.

- `p[1] = &a[1]` donc `*(p[1]) = a[1] = 2`
- `p[0] += 2` décale le pointeur à gauche de deux cases dans le type du tableau.
- Avant, `p[0]` pointait sur la case `a[2]` (i.e. `p[0] = &a[2]`) donc après le décalage, `p[0]` pointe sur `a[4]` (i.e. `p[0] = &a[4]`) puisqu'on déc ale de 2 cases.
- `**p = *(p[0])` car on a toujours que `tab[i] = *(tab+i)`. Je remplace l'une des étoiles de déréférencement par sa définition.
- donc `**p = *(&a[2])` donc `**p = a[2] = 16`

Tout ceci est un peu complexe, mais en allant pas à pas, on s'en sort. Il ne s'agit pas d'un code artificiellement complexe : j'aurais pu retirer les parenthèses dans le membre droit de la ligne 5. Il aurait fallu se souvenir de l'ordre d'évaluation des opérateurs (i.e. se souvenir que [] est plus prioritaire que *) pour réaliser que c'est pareil qu'avec les parenthèses. De même, on aurait pu écrire `a + 2` à la place de `&a[2]` à la ligne 4 si on avait voulu compliquer les choses. Mais s'il me semble important que vous compreniez du code un peu compliqué, je ne vois pas l'intérêt d'utiliser des casse-têtes artificiellement complexe en exam.

Fin réponse

▷ **Question 3** (2pt)**:** Discutez de la même façon la sortie du programme `fichier3.c` avec les schémas mémoire nécessaires, et expliquez pourquoi ce programme produit une erreur mémoire lorsqu'on remplace la ligne 3 par la suivante: `char* s = "chaine"` (le type de `s` change).

Réponse

Cherchons d'abord à comprendre pourquoi ça marche, alors que l'exemple du TD où l'on modifiait une chaîne de caractères semblable à `char* s = "chaine";` menait à un segfault. C'est que dans le cas du TD, `"chaine"` est placé dans le segment RO data du processus, c'est à dire avec les globales en lecture seule. Tenter de les modifier mène à un segfault, puisque c'est en lecture seule.

Ici, c'est très différent car `s` est un tableau dont on a laissé le compilateur compter la taille. Que ce soit ici ou dans le TD, `s` vit sur la pile. Dans le TD, c'est un pointeur qui vit sur la pile et pointe vers le segment RO data. Dans le cas de l'exam, les lettres de `"chaine"` sont recopiées dans le tableau `s` qui se trouve sur la pile. On a parfaitement le droit de modifier ce qui se trouve sur la pile, tout va bien.

Une fois qu'on a compris ça, le reste est très simple.
Après la ligne 5, `p` pointe sur la 3ieme case du tableau, celle contenant 'a'.
Après la ligne 6, cette 3ieme case prend la valeur `*s + 1`, c'est à dire `'c' + 1`, c'est à dire `'d'`.
Donc la ligne 7 affiche `dine`.

Fin réponse