

Examen IN3R11-1 (Programmation C)

Janvier 2018 (2h)
PAS DE DOCUMENT
(ni de calculatrice / iPhone / etc.)

Résumé

Réfléchissez avant de rédiger, servez vous de vos brouillons, faites des copies propres, des phrases claires. Si vous ne savez pas n'inventez pas.

1 Cours – la base

1. Citez la zone mémoire impliquée dans chacune des déclarations suivantes :

```
1 char a[] = {1,2,3};
2 char * b = malloc (...);
3 char * c = "coucou";
```

2. Qu'affichent les programmes suivants?

(a) Programme 1

```
1 #include <stdio.h>
2 void f(int a)
3 {
4     a=42;
5 }
6 int main(void){
7     int a = 5;
8     f(a);
9     printf("%d\n",a);
10    return 0;
11 }
```

(b) Programme 2

```
1 #include <stdio.h>
2 void f(int* a)
3 {
4     int * b=a;
5     *b=42;
6 }
7 int main(void){
8     int a = 5;
9     f(&a);
10    printf("%d\n",a);
11    return 0;
12 }
```

(c) Programme 3

```

1 #include <stdio.h>
2 int f(int* a)
3 {
4     int b=*a;
5     return b+5;
6 }
7 int main(void){
8     int a = 5;
9     a=f(&a);
10    printf("%d\n",a);
11    return 0;
12 }

```

3. Ecrivez une fonction `min()` qui prend en paramètre deux entiers et renvoie le plus petit.
4. Ecrivez une fonction `cube()` qui prend en paramètre un réel et renvoie son cube.
5. Ecrivez une fonction `affiche_tab()` permettant d'afficher un tableau d'entiers de type `char`.
6. Ecrivez une fonction `affiche_chaine()` permettant d'afficher une chaîne de caractères.

2 Tableaux

1. Ecrivez une fonction `sum_tab` qui prend en paramètre un tableau d'entiers et renvoie la somme de ses éléments
2. Ecrivez une fonction `cumul()` qui prend en paramètre un tableau d'entiers et le modifie pour que chaque élément après l'appel de fonction soit la somme des éléments précédents avant l'appel de fonction. Par exemple [2 7 4 1] devient [2 9 13 14]
3. Voici un code écrit par un débutant en C. Expliquez son erreur, et le comportement probable de son programme :

```

1 #include <stdio.h>
2 int* init_tab()
3 {
4     int tab[5] = {1,2,3,4,5};
5     return tab;
6 }
7 int main()
8 {
9     int* tab = init_tab();
10    int max = max_tab(...) /* appel correct a max_tab */
11    cumul(...) /* appel correct a miroir() */
12    printf("%d\n",max);
13    affiche_tab(...) /* appel correct a affiche_tab */
14    return 0;
15 }

```

4. Donnez la bonne version de `init_tab()`, avec en plus la consigne qu'elle doit prendre en paramètre la taille voulue pour le tableau.
5. Que faut-il maintenant ajouter dans le `main()` lorsqu'on a fini d'utiliser le tableau (ici après la ligne 13) ? Donnez le code **complet** de la fonction `main()`.

3 Chaînes et main

Vous disposez d'une fonction `int cherche(char* aiguille, char* botte_de_foin)`; qui renvoie la première position de la chaîne `aiguille` dans la chaîne `botte_de_foin` si elle est présente au moins une fois, -1 sinon. Par exemple, si on cherche le mot *bio* dans le mot *antibiotique*, la fonction renvoie 4.

1. Ecrivez un **programme** qui affiche la **première et la dernière** position de *vo* dans *avez vous voulu le voir avant de l'avoir vu ?* (5 et 36 donc). Votre programme doit bien évidemment appeler la fonction `cherche()`. Attention, on ne vous demande pas le code de la fonction `cherche()`.
2. Modifier le programme pour qu'il utilise des chaînes de caractères passées en paramètre du programme.
3. Nous souhaitons modifier le programme pour qu'il affiche une erreur si il n'y a pas exactement un et un seul argument sur la ligne de commande, qu'il utilise cet argument comme "aiguille" et qu'il demande à l'utilisateur de saisir la "botte de foin" sur l'entrée standard
 - (a) il existe (au moins) deux fonctions pour récupérer des chaînes saisies par l'utilisateur sur l'entrée standard :

```
1  int scanf(const char *format , ... );
```

que l'on peut appeler avec le format `%s`.

```
1  char *fgets(char *s, int size , FILE *stream );
```

dont la page de manuel nous apprend :

fgets() reads in at most one less than size characters from stream and stores them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte ('\0') is stored after the last character in the buffer.

Pour rappel, `FILE` est la structure définie dans la librairie standard pour représenter un fichier ouvert en lecture ou en écriture. Les valeurs `stdin`, `stdout` et `stderr` sont des valeurs particulières de pointeurs pour les adresses mémoires respectives de l'entrée standard, la sortie standard et la sortie d'erreur standard. Donner `stdin` comme valeur au troisième paramètre de `fgets` revient donc à lire sur l'entrée standard (comme `scanf`).

Laquelle de ces fonctions (`scanf` ou `fgets`) vous semble la plus pertinente si on suppose que les chaînes ne peuvent pas dépasser `MAX` caractères, `MAX` étant une constante définie au préalable ? **Justifiez.**

- (b) donnez la ligne à ajouter en début de fichier pour définir la constante `MAX` à la valeur 100 ainsi que le nouveau code de la fonction `main()` (juste ce qui change).
4. Nous souhaitons adapter notre code pour pouvoir rechercher une chaîne dans un texte de taille relativement modeste, mais conséquente quand même. Il n'est donc plus envisageable d'utiliser un buffer de taille fixe pour la "botte de foin" comme précédemment. Après quelques recherches, nous constatons que la man-page de `scanf` se termine sur cet exemple :
To use the dynamic allocation conversion specifier, specify m as a length modifier (thus `%ms` or `%m[range]`). The caller must free(3) the returned string, as in the following example :

```
1      char *p;
2      int n;
3
4      errno = 0;
5      n = scanf("%m[a-z]", &p);
6      if (n == 1) {
7          printf("read: %s\n", p);
8          free(p);
9      } else if (errno != 0) {
10         perror("scanf");
11     } else {
12         fprintf(stderr, "No matching characters\n");
13     }
```

As shown in the above example, it is necessary to call `free(3)` only if the `scanf()` call successfully read a string.

Pour rappel ou info :

— La variable globale `errno` est un entier.
— La fonction `perror` est simplement une fonction d’affichage normalisée d’erreur.
Donner le nouveau code de votre fonction `main()`, qui utilisera `scanf` et son ”dynamic conversion specifier”. N’oubliez pas de libérer la mémoire qui serait éventuellement allouée dynamiquement sur le tas.

4 Client

1. Dans le cadre du développement d’une application de gestion pour un magasin, donnez le code C nécessaire à la définition d’un nouveau type permettant de manipuler facilement un client, c’est à dire un nom, un prénom et un nombre d’achat. Dans une fonction, il devra être possible d’écrire :

```
1 Client c;
```

2. Ecrivez une fonction `nouveau_client()` qui prend en paramètre un nom et un prénom (pourquoi pas le nombre d’achat ?), et renvoie un client.
3. Ecrivez une fonction `affiche_client()` permettant d’afficher un client sous la forme ”nom, prénom : nombre d’achat”
4. Ecrivez une fonction `swap()` qui prend en paramètre deux clients et les échange. Vous êtes libre sur la signature de la fonction mais devez donner un code `main()` cohérent avec votre fonction permettant de la tester. Aide : écrivez au brouillon la fonction `swap()` qui échange deux entiers, et adaptez ensuite...