Examen final de programmation impérative

ÉNSIIE, semestre 1

mercredi 8 janvier 2025

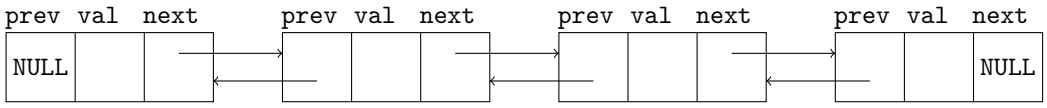
Durée : 1h45.
Tout document papier autorisé. Aucun appareil électronique autorisé (sauf dérogation par la scolarité).
Ce sujet comporte 3 exercices indépendants, qui peuvent être traités dans l'ordre voulu. Il contient 3 pages. Le barème est donné à titre indicatif, il est susceptible d'être modifié. Le total est sur 20 points. Il va de soi que toute réponse devra être justifiée, et que toute fonction devra être commentée (au minimum `require`, `assigns` et `ensures`).

Exercice 1 : Fonctions simples (6 points)

1. Proposer une procédure dont l'effet est d'échanger les valeurs de deux variables de type `char` définies auparavant. (Exemple de cas d'utilisation : la fonction est définie *avant* `main`. À l'intérieur de `main`, deux variables sont définies et la fonction appelée avec ces variables. On n'écrira pas la fonction `main`.)
2. Proposer une procédure qui prend en paramètre un tableau non-vide de `int` et sa taille et qui tourne d'un cran le contenu du tableau, c'est-à-dire que la valeur contenue dans une case se retrouvera à la case suivante, sauf pour la dernière case dont le contenu se retrouvera dans la première. Exemple : un tableau contenant | 3 | 7 | 2 | 8 | 1 | 4 | deviendra | 4 | 3 | 7 | 2 | 8 | 1 |.
3. Proposer une fonction qui prend en paramètre un entier n et qui retourne une chaîne de caractères contenant n fois le caractère `*`.
4. Proposer une fonction qui prend en paramètre une liste chaînée sur des `int`, et qui retourne l'adresse du dernier maillon de la liste, ou `NULL` s'il n'y en a pas.

Exercice 2 : Listes doublement chaînées (10pts)

On considère des listes dans lesquelles, en plus du champ `next` qui indique le prochain maillon, on a également un champ `prev` qui indique le maillon précédent. Ceci permet donc, à la différence des listes chaînées simples, de pouvoir parcourir la liste dans les deux sens. Une telle liste sera donc de la forme :

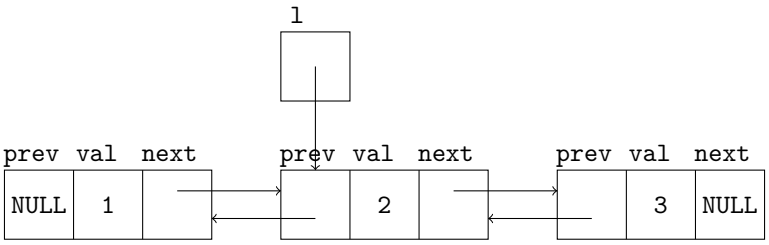


On considérera ici des listes doublement chaînées contenant des entiers.

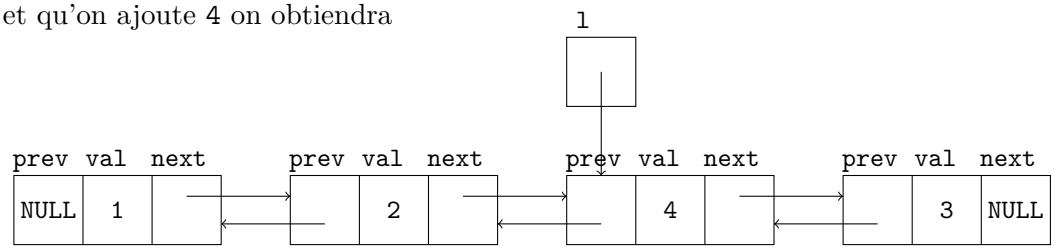
1. Définir le type `dl_list` des listes doublement chaînées comme un pointeur vers un type `cell`, ce type `cell` étant défini comme un enregistrement contenant les trois champs `prev`, `val` et `next`. (Vous écrirez aussi la définition de `cell`.)

2. Écrire une fonction `singleton` qui prend en paramètre un entier et qui retourne un liste doublement chaînée constituée d'un seul maillon qui contient cet entier.
3. Écrire une procédure `go_to_first` qui prend une liste doublement chaînée par référence et qui la modifie par effet pour quelle pointe maintenant sur son premier maillon. Sans effet si la liste est vide.
4. Écrire une procédure `add_next` qui prend en paramètre un liste doublement chaînée par référence et un entier, et qui ajoute un maillon contenant l'entier juste après le maillon courant, sans perdre le reste de la liste, puis qui fait pointer la liste sur le nouveau maillon. Si la liste est initialement vide, elle pointera ensuite sur un maillon sans prédécesseur ni successeur et contenant l'entier.

   Par exemple, si on a la liste `l`



   et qu'on ajoute `4` on obtiendra



5. Écrire une fonction `length` qui prend en paramètre une liste doublement chaînée et qui retourne le nombre de maillons qu'elle contient. On prendra en compte les maillons situés avant et après le maillon pointé.
6. Écrire une procédure `free_dl_list` qui prend en paramètre une liste doublement chaînée et qui libère toute la mémoire allouée dans cette liste.
7. Soit le programme suivant :

```
1  dl_list l = singleton(1);
2  add_next(&l, 2);
3  dl_list q = l;
4  go_to_first(&l);
5  add_next(&l, 3);
6  add_next(&q, 4);
7  q->next = l;
```

   Représenter l'état de la mémoire au cours de son exécution.

   Quelle première remarque peut-on faire ?
8. On rajoute la ligne

```
8  l->prev = q;
```

   Quelles deux autres remarques peut-on faire ?

Exercice 3 : Jeu des 4 erreurs (4 points)

Les procédures et fonctions suivantes ne sont pas correctes par rapport à la spécification donnée. Les corriger en expliquant pourquoi il y a une erreur. Les erreurs ne proviennent pas de **#include** manquants, parce qu'ils sont supposés avoir été écrits s'il y en a besoin.

1.
```
/*@requires s >= 0 and t has size s
    assigns t[0] ... t[s-1]
    ensures initialize t with the value of the index */
int initialize(int t[], int s) {
  for (int i = 0; i < s; i += 1)
    t[i] = i;
}
```

2.
```
/*@requires s nul-terminated string
  @assigns nothing
  @ensures return a copy of s */
char *string_copy(char *s) {
  int i = 0;
  while (s[i] != '\0') i += 1;
  char *c = malloc(i + 1);
  i = 0;
  while (s[i] != '\0') c[i] = s[i]; i += 1;
  c[i] = '\0';
  return c;
}
```

3. On considère la définition des listes chaînées vue en cours.

```
/*@requires n is a well-formed list
    assigns nothing
    ensures return a list starting by a new bucket containing v,
      followed by n */
list cons(int v, list n) {
  list r = malloc(sizeof (list));
  r->val = v;
  r->next = n;
  return r;
}
```

4.
```
/*@requires l is a well-formed list
    assigns *l
    ensures remove the first bucket of the list
      and go to the next */
void remove_first(list l) {
  list tmp = l;
  if (l == NULL) return;
  l = l->next;
  free(tmp);
}
```