SCIENCES SORBONNE UNIVERSITÉ

Votre numéro d'anonymat : | | | |

Programmation et structures de données en C– LU2IN018

Examen du 17 janvier 2020
2 heures
Aucun document n'est autorisé.

Les calculatrices, baladeurs et autres appareils électroniques sont interdits. Les téléphones mobiles doivent être éteints et rangés dans les sacs. Le barème sur 60 points (10 questions) n'a qu'une valeur indicative.

Les appels à `malloc` seront supposés réussir. Il ne sera pas nécessaire de vérifier leur valeur de retour. De la même manière, l'ouverture d'un fichier sera supposée réussir. Il ne sera pas nécessaire de vérifier que c'est bien le cas.

Le mémento qui vous a été distribué est reproduit à la fin de l'énoncé.

L'ensemble des structures et prototypes de fonctions est également rappelé à la fin de l'énoncé, sur une page détachable.

Bibliographie

Le travail scientifique est évalué essentiellement sur la base d'articles publiés dans des journaux scientifiques spécialisés ou dans des conférences. Ces articles sont le plus souvent signés par plusieurs auteurs. Dans le texte de ces articles sont cités d'autres articles écrits précédemment (par les mêmes auteurs ou par d'autres auteurs). Le travail scientifique d'un auteur est évalué en fonction du nombre d'articles qu'il a écrits mais aussi en fonction du nombre de fois que ses articles sont cités dans d'autres articles. Un article fondamental dans un domaine sera cité un très grand nombre de fois.

Il existe plusieurs mesures quantitatives qui peuvent donner une idée de l'impact d'un chercheur sur son domaine. Un de ces indicateurs est le H-Index. Vous allez par la suite écrire du code pour gérer un ensemble d'articles et leurs auteurs, et pour calculer la valeur de cet indicateur (dont le principe sera expliqué ultérieurement).

1 Gestion des auteurs

Les auteurs sont décrits par leur nom, prénom, affiliation (le laboratoire dont ils sont membres) et email. Ils sont représentés par la structure de données suivante :

```
typedef struct _auteur *PAuteur;
typedef struct _auteur {
  char *nom;
```

```
  char *prenom;
  char *affiliation;
  char *email;
} Auteur;
```

Chacune des chaînes de caractères est spécifique à un auteur et devra donc être allouée et libérée en même temps que cette variable.

Question 1 (3 points)

Écrivez une fonction permettant d'allouer et d'initialiser une structure `Auteur`. Aucune hypothèse ne sera faite sur la pérennité des chaînes de caractères transmises en argument. Prototype :

```
PAuteur creer_auteur(const char *nom, const char *prenom, const char
   *affiliation, const char *email);
```

Solution:

```
PAuteur creer_auteur(const char *nom, const char *prenom, const char
    *affiliation, const char *email) {
  PAuteur auteur=(PAuteur)malloc(sizeof(Auteur));
  auteur->nom=strdup(nom);
  auteur->prenom=strdup(prenom);
  auteur->affiliation=strdup(affiliation);
  auteur->email=strdup(email);
  return auteur;
}
```

Question 2 (3 points)

Écrivez une fonction permettant de libérer la mémoire allouée à un auteur. Prototype :

```
void liberer_auteur(PAuteur pauteur);
```

Solution:

```
void liberer_auteur(PAuteur pauteur) {
  free(pauteur->nom);
  free(pauteur->prenom);
  free(pauteur->affiliation);
  free(pauteur->email);
  free(pauteur);
}
```

Question 3 (6 points)

Les auteurs vont être stockés dans une liste chaînée donnée par la structure suivante :

```
typedef struct _liste_auteurs *PListeAuteurs;
typedef struct _liste_auteurs {
  PAuteur auteur;
  PListeAuteurs suivant;
} ListeAuteurs;
```

Écrivez une fonction permettant d'afficher une liste d'auteurs. Prototype :

```
void afficher_auteurs(PListeAuteurs lauteurs);
```

La fonction affiche, sur une ligne et pour chaque auteur, les différents champs séparés par des virgules. Exemple :

```
Dennis, Ritchie, dmr@bell-labs.com, Bell Labs
```

Solution:

```
void afficher_auteurs(PListeAuteurs lauteurs) {
  while(lauteurs) {
    printf("%s␣%s,␣%s,␣%s\n",lauteurs->auteur->prenom,
           lauteurs->auteur->nom,
           lauteurs->auteur->email,
           lauteurs->auteur->affiliation);
    lauteurs=lauteurs->suivant;
  }
}
```

2 Gestion des articles

Un article a une liste d'auteurs, un titre, une année, le nom du journal dans lequel il a été publié, ainsi que le numéro de volume correspondant et, pour finir, une liste d'articles qu'il cite.

Les articles sont représentés avec la structure suivante :

```
typedef struct _liste_articles *PListeArticles;

typedef struct _article *PArticle;
typedef struct _article {
  PListeAuteurs lauteurs;
  char *titre;
  int annee;
  char *journal;
  int volume;
  PListeArticles lart_cites;
} Article;

typedef struct _liste_articles {
  PArticle article;
  PListeArticles suivant;
} ListeArticles;
```

Question 4 (6 points)

Écrivez une fonction pour vérifier si un auteur fait partie de la liste d'auteurs d'un article. Prototype :

```
int est_auteur(PAuteur pauteur, PArticle particle);
```

La fonction renvoie 1 si l'auteur est un des auteurs de l'article, 0 sinon.

Vous pouvez utiliser la fonction suivante (pas besoin de l'implémenter) :

```
int comparer_auteurs(PAuteur p1, PAuteur p2);
```

Cette fonction renvoie 0 si les deux auteurs passés en argument ont le même nom et le même prénom [1]. Elle renvoie une valeur négative si `p1` est avant `p2` dans l'ordre alphabétique et positive si `p1` est après `p2`. Il est à noter que la liste des auteurs d'un article n'est pas classée par ordre alphabétique [2].

Solution:

```
int est_auteur(PAuteur pauteur, PArticle particle) {
  PListeAuteurs lauteurs=particle->lauteurs;
  while(lauteurs) {
    if(comparer_auteurs(pauteur, lauteurs->auteur)==0) {
      return 1;
    }
    lauteurs=lauteurs->suivant;
  }
  return 0;
}
```

Nous allons lire les articles, les auteurs et les citations dans un fichier texte dont le format est le suivant :

```
titre article 1
année de publication
nom du journal
numéro de volume
<<<
nom de l'auteur 1
prénom de l'auteur 1
affiliation de l'auteur 1
email de l'auteur 1
===
nom de l'auteur 2
prénom de l'auteur 2
affiliation de l'auteur 2
email de l'auteur 2
===
nom de l'auteur 3
prénom de l'auteur 3
affiliation de l'auteur 3
email de l'auteur 3
[...]
>>>
titre de la citation 1
titre de la citation 2
titre de la citation 3
titre de la citation 4
[...]
+++
```

1. On fait l'hypothèse qu'il s'agit dans ce cas de la même personne.
2. L'ordre des auteurs a son importance et dépend des domaines. En informatique, par exemple, le premier auteur est le principal contributeur du travail et le dernier auteur est l'encadrant principal.

```
titre article 2
année de publication
[...]
```

Exemple :

```
UNIX time-sharing system: Portability of C programs and the UNIX system
1978
The Bell System Technical Journal
57
<<<
Johnson
Stephen C.
Bell Labs
scj@bell-labs.com
===
Ritchie
Dennis M.
Bell Labs
dmr@bell-labs.com
>>>
The  C  Programming  Language
UNIX Time-Sharing System: The C Programming Language
ALTRAN User's Manual
The PFORT Verifier
The  PORT  Mathematical  Subroutine  Library
+++
```

Les "<<<", "===" et ">>>" servent à délimiter les différentes parties.
Le titre d'un article est ici unique et pourra être utilisé comme identifiant. Ainsi, les articles cités dans un article (qui devront être listés dans le champs `lart_cites` de la structure `Article`) sont écrits dans le fichier entre les lignes ">>>" et les lignes "+++" et seul leur titre est donné. "+++" indique la fin des données liées à un article. Ce qui suit correspond à un nouvel article.

Un article contient une liste de pointeurs vers les articles qu'il cite. La lecture d'un article ne peut donc pas se passer en une étape car on n'est pas sûr qu'un article cité ait été déjà lu. Elle se passe donc en deux étapes. Dans un premier temps, les articles (citations non comprises) et les auteurs sont lus. Dans un deuxième temps, les citations sont lues et la liste des pointeurs des articles cités est créée pour chaque article.
La fonction de lecture (à compléter) est donnée ci-dessous :

```
void lire_base_articles(const char *fichier, PListeAuteurs *plauteurs
   , PListeArticles *plarticles) {
  FILE *f=fopen(fichier,"r");

  /* Lecture des auteurs et des articles (citations non comprises) */
  *plauteurs=NULL;
  PArticle article=lire_article(f, plauteurs);
```

```
  while(article!=NULL) {
    *plarticles = ajouter_en_tete(article, *plarticles);
    article=lire_article(f, plauteurs);
  }

  /* On revient au debut du fichier */
  rewind(f);


  /* A COMPLETER POUR LIRE LES CITATIONS */

  fclose(f);
}
```

Vous n'avez pas à écrire la fonction `lire_article`.

Question 5 (6 points)

Donnez les instructions à ajouter à l'emplacement indiqué pour lire les citations. Vous devez lire des lignes, enlever le caractère '\n' final, rechercher l'article à partir de son titre et ignorer toutes les lignes jusqu'aux citations. À partir de là, il faut lire les différents articles cités, les rechercher dans la liste des articles lus et les ajouter à la liste des citations (cette liste n'est pas ordonnée). On suppose que tout article cité par un autre est forcément présent dans le même fichier - aucune vérification n'est donc nécessaire.

Vous pouvez utiliser la fonction suivante pour trouver un article à partir de son titre dans une liste d'articles (vous n'avez pas besoin de l'implémenter) :

```
PArticle trouver_article(const char *titre, PListeArticles larticles)
   ;
```

Vous pouvez également vous appuyer sur la fonction suivante pour ajouter un article en tête d'une liste d'articles (pas besoin de l'implémenter non plus) :

```
PListeArticles ajouter_en_tete(PArticle article, PListeArticles
   articles);
```

Solution:

```
  /* Lecture des citations */
  char buffer[256];

  while(fgets(buffer, 256, f)!=NULL) {
    buffer[strlen(buffer)-1]='\0';
    PArticle pa = trouver_article(buffer, *plarticles);
    printf("Searching␣for␣'%s'␣in␣the␣database\n",buffer);
    assert(pa);
    do {
      fgets(buffer, 256, f);
    }
    while(strncmp(buffer,">>>", 3)!=0);
    fgets(buffer, 256, f);
    while(strncmp(buffer,"+++", 3)!=0) {
      buffer[strlen(buffer)-1]='\0';
      printf("Looking␣for␣article:␣%s\n",buffer);
```

```
      PArticle pac = trouver_article(buffer, *plarticles);
      assert(pac);
      pa->lart_cites = ajouter_en_tete(pac, pa->lart_cites);
      fgets(buffer, 256, f);
    }

  }
```

Il est possible également de faire des strcmp avec '\n' à la fin de la chaîne. Si le '\n' est oublié, c'est considéré comme une erreur mineure qui, à elle seule, ne doit pas faire perdre de points.

3 Ensemble d'auteurs et d'articles

Nous avons besoin de manipuler une grande base d'auteurs, nous allons les stocker dans un arbre binaire de recherche où les auteurs sont ordonnés par ordre lexicographique de nom puis prénom pour les retrouver plus vite. Nous allons en profiter pour associer à un auteur la liste des articles qu'il a publiés.

La structure est la suivante :

```
typedef struct _abr_auteur *PABRAuteurs;
typedef struct _abr_auteur {
  PAuteur auteur;
  PListeArticles articles;
  PABRAuteurs gauche;
  PABRAuteurs droit;
} ABRAuteurs;
```

Question 6 (9 points)

La fonction de construction de l'arbre binaire de recherche a le prototype suivant :

```
PABRAuteurs construire_abr_auteurs(PListeArticles larticles);
```

Cette fonction parcourt la liste d'articles et ajoute chacun des auteurs des articles de la liste à un arbre binaire de recherche. La fonction s'appuie sur la fonction de prototype :

```
PABRAuteurs ajouter_abr_auteurs(PAuteur auteur, PArticle article,
   PABRAuteurs abr);
```

La fonction `ajouter_abr_auteurs` ajoute l'auteur `auteur` à l'arbre binaire de recherche `abr`. Elle ajoute également l'article `article` à la liste des articles associés à cet auteur.

Écrivez les deux fonctions.

Vous pouvez utiliser la fonction `comparer_auteurs` vue précédemment.

Solution:

```
PABRAuteurs ajouter_abr_auteurs(PAuteur auteur, PArticle article,
   PABRAuteurs abr) {
  if (abr) {
    int cmp=comparer_auteurs(auteur, abr->auteur);
    if (cmp==0) {
      abr->articles=ajouter_en_tete(article, abr->articles);
```

```
    }
    if (cmp<0) {
      abr->gauche=ajouter_abr_auteurs(auteur, article, abr->gauche);
    }
    if (cmp>0) {
      abr->droit=ajouter_abr_auteurs(auteur, article, abr->droit);
    }
  }
  else {
    abr=(PABRAuteurs)malloc(sizeof(ABRAuteurs));
    abr->auteur=auteur;
    abr->articles=NULL;
    abr->gauche=NULL;
    abr->droit=NULL;
    abr->articles=ajouter_en_tete(article, abr->articles);
  }
  return abr;
}

PABRAuteurs construire_abr_auteurs(PListeArticles larticles) {
  PABRAuteurs abr_aut=NULL;
  while (larticles) {
    PListeAuteurs lauteurs=larticles->article->lauteurs;
    while (lauteurs) {
      abr_aut = ajouter_abr_auteurs(lauteurs->auteur, larticles->
         article, abr_aut);
      lauteurs=lauteurs->suivant;
    }
    larticles=larticles->suivant;
  }
  return abr_aut;
}
```

Question 7 (6 points)

Écrivez la fonction de recherche d'un auteur dans l'arbre binaire de recherche. Prototype :

```
PABRAuteurs chercher_abr(char *nom, char *prenom, PABRAuteurs pabra);
```

La fonction cherche dans l'arbre binaire de recherche des auteurs l'auteur ayant le nom et le prénom transmis en argument. La valeur renvoyée est le pointeur sur le nœud contenant cet auteur s'il a été trouvé, `NULL` sinon.

Solution:

```
PABRAuteurs chercher_abr(char *nom, char *prenom, PABRAuteurs pabra)
    {
  if (pabra==NULL) return NULL;
  Auteur a;
  a.nom=nom;
  a.prenom=prenom;
```

```
  a.affiliation=NULL;
  a.email=NULL;
  if (comparer_auteurs(&a, pabra->auteur)<0) {
    return chercher_abr(nom, prenom, pabra->gauche);
  }
  if (comparer_auteurs(&a, pabra->auteur)>0) {
    return chercher_abr(nom, prenom, pabra->droit);
  }
  return pabra;
}
```

Un appel à `creer_auteur` est tout à fait acceptable à partir du moment où il y a un free ensuite.

Question 8 (9 points)

Le H-Index est un entier qui caractérise l'impact des publications d'un auteur. Le principe de son calcul est le suivant :

— on considère la liste des articles écrits par cet auteur
— on regarde combien de fois chacun de ces articles est cité
— on trie cette liste par ordre décroissant du nombre de citations

Le H-Index correspond à l'entier `i` tel que le i-ième article le plus cité est cité au moins i fois et que le i+1-ième est cité moins de i+1 fois.

Exemple : Soit Zennis Ritchie qui a écrit 6 articles, un cité 11 fois, un 9 fois, un 7 fois, un 5 fois, un 3 fois et un 1 fois.

Voici la liste des articles triés par nombre de citations décroissantes :

nombre de citations :	11	9	7	5	3	1
indice de l'article :	1	2	3	4	5	6

Son H-index est de 4 : il a 4 articles cités au moins 4 fois (et il n'a pas 5 articles cités au moins 5 fois). C'est donc l'indice de la case avant la case où l'indice est supérieur aux nombre de citations.

Écrivez la fonction de calcul du H-Index. Prototype :

```
int calcul_HIndex(PAuteur auteur, PABRAuteurs abr, PListeArticles
   larticles);
```

Vous pouvez utiliser les fonctions suivantes (vous n'avez pas à les écrire) :

```
int compter_articles(PListeArticles larticles);

int nb_citations(PArticle article, PListeArticles larticles);
```

La fonction `compter_articles` renvoie le nombre d'articles de la liste passée en argument.

La fonction `nb_citations` parcourt la liste d'articles passée en deuxième argument, compte le nombre d'articles qui citent l'article passé en premier argument et renvoie cette valeur.

Pour le tri des nombres de citations, vous devez construire un tableau des nombres de citations que vous devez trier avec la fonction de la libC `qsort` dont le prototype est le suivant :

```
void qsort(void *base, int nel, int width, int (*compar)(const void *
   a, const void *b));
```

`base` pointe sur le premier octet des données à trier, `nel` est le nombre d'éléments à trier, `width` est la taille d'un élément en octets et `compar` le pointeur sur la fonction de comparaison. Pour que les données soient triées par ordre décroissant, vous devez définir la fonction de comparaison appropriée.

Celle-ci doit renvoyer une valeur positive si `a` est plus petit que `b`, négative si `a` est plus grand que `b` et nulle sinon.

Solution:

```
int comparer_int(const void *i1, const void *i2) {
  const int *ii1=(const int *)i1;
  const int *ii2=(const int *)i2;
  return  (*ii1<*ii2) - (*ii1>*ii2);
}

int calcul_HIndex(PAuteur auteur, PABRAuteurs abr, PListeArticles
   articles) {
  PABRAuteurs abra=chercher_abr(auteur->nom, auteur->prenom, abr);
  if(abra!=NULL) {
    PListeArticles aut_articles=abra->articles;
    assert(aut_articles!=NULL);
    int nb_art=compter_articles(aut_articles);
    int *art_citations=(int *)malloc(nb_art*sizeof(int));
    int i;
    for (i=0;i<nb_art; i++) {
      art_citations[i] = nb_citations(aut_articles->article,
         articles);
      aut_articles = aut_articles->suivant;
    }
    qsort(art_citations, nb_art, sizeof(int), comparer_int);
    for (i=0;i<nb_art;i++) {
      printf("nb␣citations:␣%d\n",art_citations[i]);
    }
    int hindex=0;
    for (i=0;i<nb_art; i++) {
      if (art_citations[i]<i) {
        hindex=i;
        break;
      }
    }
    free(art_citations);
    return hindex;
  }

  return 0;

}
```

4 Documents de type variés

Les publications ne sont pas uniquement dans des journaux. Il peut s'agir également d'un livre. Dans ce cas, c'est l'éditeur qu'il faut donner. Elles peuvent être également dans des conférences, dans ce cas, il

faut fournir le nom de la conférence et le lieu à la place du nom du journal et du numéro de volume (ce type de document ne sera pas géré dans la suite).

Pour pouvoir gérer des types de documents variés, une autre structure va être utilisée. Cette structure associe les données du document aux fonctions permettant de manipuler ces données :

```
typedef void (*Afficher)(void *data);
typedef void (*Detruire)(void *data);
typedef void * (*Lire)(FILE *f);

typedef struct _document *PDocument;
typedef struct _document {
  void *data;
  Afficher afficher;
  Detruire detruire;
  Lire lire;
} Document;

typedef struct _liste_documents *PListeDocuments;
typedef struct _liste_documents {
  PDocument doc;
  PListeDocuments suivant;
} ListeDocuments;
```

Question 9 (3 points)

Écrivez la fonction d'affichage de documents qui parcourt la liste des documents transmise en argument et fait appel à la fonction d'affichage associée. Prototype :

```
void afficher_documents(PListeDocuments ldocs);
```

Solution:

```
void afficher_documents(PListeDocuments ldocs) {
  while(ldocs) {
    ldocs->doc->afficher(ldocs->doc->data);
    ldocs=ldocs->suivant;
  }
}
```

Question 10 (9 points)

Les documents sont écrits dans un fichier les uns après les autres. Avant chaque document, une ligne indique le type du document. La ligne "#article\n" indique que ce qui suit correspond à un article. "#ouvrage\n" indique que ce qui suit correspond à un livre. Les fonctions afficher, détruire et lire d'un article sont, respectivement, `afficher_article`, `detruire_article` et `lire_article`. Les fonctions pour un ouvrage sont : `afficher_ouvrage`, `detruire_ouvrage` et `lire_ouvrage`. Vous n'avez pas à écrire ces différentes fonctions.

Écrivez la fonction de lecture d'un ensemble de documents. Prototype :

```
PListeDocuments lire_docs(const char *nom_fichier);
```

La fonction ouvre le fichier dont le nom est transmis en argument et, tant que la fin du fichier n'est pas atteinte, va lire des documents et les mettre dans une liste chaînée qui sera renvoyée par la fonction. La lecture d'un document s'appuye sur les fonctions de lecture appropriée (lecture d'ouvrage ou d'article).

Solution:

```
PListeDocuments lire_docs(const char *nom_fichier) {
  PListeDocuments res=NULL, pld;
  FILE *f=fopen(nom_fichier,"r");
  char buffer[256];
  while(fgets(buffer, 256, f)){
    PDocument pdoc=(PDocument)malloc(sizeof(Document));
    if(strcmp(buffer, "#article\n")==0) {
      pdoc->afficher=afficher_article;
      pdoc->detruire=detruire_article;
      pdoc->lire=lire_article;
      pdoc->data=lire_article(f);
    }
    if(strcmp(buffer, "#ouvrage\n")==0) {
      pdoc->afficher=afficher_ouvrage;
      pdoc->detruire=detruire_ouvrage;
      pdoc->lire=lire_ouvrage;
      pdoc->data=lire_ouvrage(f);
    }
    pld=(PListeDocuments)malloc(sizeof(ListeDocuments));
    pld->doc=pdoc;
    pld->suivant=res;
    res=pld;
  }
  fclose(f);
  return res;
}
```

Mémento de l'UE LU2IN018

Ce document a pour vocation de présenter un bref descriptif des fonctions offertes par les bibliothèques standards et qui sont susceptibles d'être utilisées dans l'UE.

Entrées - sorties

Prototypes disponibles dans `stdio.h`.

Entrées, sorties formatées

```
int printf(const char *format, ...);
```

La fonction `printf` écrit sur la sortie standard (par défaut le terminal), un message correspondant au texte spécifié dans la chaîne `format`. Le texte affiché peut contenir des portions *variables* spécifiées par des codes de conversion précédés par le caractère `%`. Les arguments suivant `format` doivent correspondre (ordre et type) aux codes de conversion présents. Voici quelques codes de conversion courants :

- `%d` : entier
- `%c` : caractère
- `%s` : chaîne de caractères
- `%f` : nombre réel

`printf` renvoie le nombre de caractères imprimés et la chaîne de format peut contenir des codes de contrôle permettant le formatage comme `\n` pour forcer le passage à la ligne et `\t` pour insérer une tabulation.

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

Les fonctions `scanf` et `sscanf` permettent de saisir et analyser un texte saisi sur l'entrée standard, par défaut le clavier (`scanf`) ou depuis une chaîne de caractères passée en argument (`sscanf`). Le texte devra respecter le `format` spécifié et les arguments suivants doivent correspondre à des pointeurs sur des variables de type appropriés. Les codes de conversion son identiques à ceux de `printf`.

Entrées, sorties caractères

```
int getchar(void);
```

Lit un caractère dans le flux d'entrée standard (par défaut le clavier). La fonction retourne le code **EOF** en cas d'erreur, le caractère lu dans le cas contraire.

```
int putchar(int c);
```

Affiche un caractère dans le flux de sortie standard (par défaut le terminal). La fonction retourne le code **EOF** en cas d'erreur, le caractère lu dans le cas contraire.

```
int puts(const char *s);
```

Affiche une chaîne de caractères dans le terminal et passe à la ligne, renvoie **EOF** en cas d'erreur.

Fichiers

Prototypes disponibles dans `stdio.h`.

```
FILE *fopen(const char *path, const char *mode);
```

Ouvre un fichier dont le chemin est spécifié par la chaîne `path` et retourne un pointeur de type `FILE *` (`NULL` en cas d'échec). L'argument `mode` permet de spécifier le type d'accès à réaliser sur le fichier :

- `[r]` pour un accès en lecture,
- `[w]` pour un accès en écriture et le contenu précédent du fichier est écrasé,
- `[a]` pour un accès en écriture, le contenu du fichier est préservé et les écritures sont effectuées à la suite des contenus déjà présents.

En cas d'erreur la fonction retourne le code **NULL** sinon un pointeur vers le fichier ouvert.

```
int fclose(FILE *fp);
```

Cette fonction provoque la fermeture du fichier pointé par `fp`. En cas d'erreur la fonction retourne le code **EOF** sinon `0`.

```
int fprintf(FILE *stream, const char *format, ...);
```

Identique à `printf` mais l'argument `stream` permet de spécifier le flux de sortie.

```
int fscanf(FILE *stream, const char *format, ...);
```

Identique à `scanf` mais l'argument `stream` permet de spécifier le flux d'entrée.

```
int fputc(int c, FILE *stream);
```

Identique à `putchar` mais l'argument `stream` permet de spécifier le flux de sortie.

```
int fputs(const char *s, FILE *stream);
```

Identique à `puts` mais l'argument `stream` permet de spécifier le flux de sortie et il n'y a pas d'ajout de passage à la ligne.

```
int getc(FILE *stream);
```

Identique à `getchar` mais l'argument `stream` permet de spécifier le flux d'entrée.

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

Lit au plus `size-1` octets dans le flux `stream`. La lecture s'arrête dès qu'un passage à la ligne est rencontré. Les octets lus sont stockés dans `s`. La fonction retourne `s` en cas de succès et `NULL` en cas d'erreur.

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Lecture binaire de `nmemb` éléments de `size` octets dans le fichier `stream`. Les données lues sont stockées en mémoire à partir de l'adresse `ptr`. La fonction retourne le nombre d'éléments effectivement lus.

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb,
              FILE *stream);
```

Écriture de `nmemb` éléments de `size` octets dans le fichier `stream`. Les données à écrire sont lues en mémoire à partir de l'adresse `ptr`. La fonction retourne le nombre d'éléments effectivement écrits.

Chaînes de caractères

Prototypes disponibles dans `string.h`.

Une chaîne de caractères correspond à un tableau de caractère et doit contenir un marqueur de fin `\0`.

```
size_t strlen (const char *s);
```

Renvoie la longueur d'une chaîne de caractères (marqueur de fin `\0` non compris).

```
int strcmp (const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
```

Comparaison entre chaînes de caractères éventuellement limité aux `n` premiers caractères. La valeur retournée est :

- 0 si les deux chaînes sont identiques,
- négative si `s1` précède `s2` dans l'ordre lexicographique (généralisation de l'ordre alphabétique),
- positive sinon.

```
char *strcpy (char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t n);
```

Copie le contenu de la chaîne `src` dans la chaîne `dest` (marqueur de fin `\0` compris). La chaîne `dest` doit avoir précédemment été allouée. La copie peut être limitée à `n` caractères et la valeur retournée correspond au pointeur de destination `dest`.

```
void *memcpy(void *dest, const void *src, size_t n);
```

Copie `n` octets à partir de l'adresse contenue dans le pointeur `src` vers l'adresse stockée dans `dest`. `dest` doit pointer vers une zone mémoire préalablement allouée et de taille suffisante. `memcpy` renvoie la valeur de `dest`.

```
size_t strlen(const char *s);
```

Retourne le nombre de caractères de la chaîne `s` (marqueur de fin `\0` non compris).

```
char *strdup(const char *s);
```

Cette fonction permet de dupliquer une chaîne de caractères, elle retourne un pointeur vers la chaîne nouvellement allouée. La nouvelle chaîne pourra être libérée avec la fonction `free`.

```
char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src, size_t n);
```

Ajoute la chaîne `src` à la suite de la chaîne `dst`. La chaîne `dest` devra avoir été allouée et être de taille suffisante. La fonction retourne `dest`.

```
char *strstr(const char *haystack, const char *needle);
```

La fonction renvoie un pointeur sur la première occurrence de la sous-chaîne `needle` rencontrée dans la chaîne `haystack`. Si la chaîne recherchée n'est pas présente, la fonction retourne `NULL`.

Conversion de chaînes de caractères

Prototypes disponibles dans `stdlib.h`.

```
int atoi(const char *nptr);
```

La fonction convertit le début de la chaîne pointée par `nptr` en un entier de type `int`.

```
double atof(const char *nptr);
```

Cette fonction convertit le début de la chaîne pointée par `nptr` en un `double`.

```
long int strtol(const char *nptr, char **endptr, int base);
```

Convertit le début de la chaîne `nptr` en un entier long. l'interprétation tient compte de la `base` et la variable pointée par `endptr` est affectée avec l'adresse du premier caractère invalide (au sens de la conversion).

Allocation dynamique de mémoire

Prototypes disponibles dans `stdlib.h`.

```
void *malloc(size_t size);
```

Alloue `size` octets de mémoire et retourne un pointeur générique correspondant à l'adresse du premier octet de la zone, renvoie `NULL` en cas d'échec.

```
void *realloc(void *ptr, size_t size);
```

Permet de modifier la taille d'une zone de mémoire allouée dynamiquement. `ptr` doit correspondre à l'adresse du premier octet de la zone précédemment allouée par `malloc` ou `realloc`. `size` correspond à la taille en octet de la nouvelle zone allouée. `realloc` garantie que la nouvelle zone contiendra les données présentes dans la zone initiale.

```
void free(void *ptr);
```

Libère une zone mémoire allouée dynamiquement. `ptr` doit correspondre à l'adresse du premier octet de la zone précédemment allouée par `malloc` ou `realloc`.

La liste des fonctions du programme considéré est indiquée ci-après. Certaines fonctions ne sont pas à écrire. Ces fonctions peuvent tout de même être utilisées et considérées comme disponibles.

```
#ifndef _ARTICLE_H_
#define _ARTICLE_H_

#include <stdio.h>

typedef struct _auteur *PAuteur;
typedef struct _auteur {
  char *nom;
  char *prenom;
  char *affiliation;
  char *email;
} Auteur;

typedef struct _liste_auteurs *PListeAuteurs;
typedef struct _liste_auteurs {
  PAuteur auteur;
  PListeAuteurs suivant;
} ListeAuteurs;

typedef struct _liste_articles *PListeArticles;

typedef struct _article *PArticle;
typedef struct _article {
  PListeAuteurs lauteurs;
  char *titre;
  int annee;
  char *journal;
  int volume;
  PListeArticles lart_cites;
} Article;

typedef struct _liste_articles {
  PArticle article;
  PListeArticles suivant;
} ListeArticles;

typedef struct _abr_auteur *PABRAuteurs;
typedef struct _abr_auteur {
  PAuteur auteur;
  PListeArticles articles;
  PABRAuteurs gauche;
  PABRAuteurs droit;
} ABRAuteurs;


PAuteur creer_auteur(const char *nom, const char *
   prenom, const char *affiliation, const char *
   email);

void afficher_auteurs(PListeAuteurs lauteurs);

void afficher_articles(PListeArticles larticles);

int est_auteur(PAuteur pauteur, PArticle particle);

PListeAuteurs trouver_auteur(const char *nom, const
   char *prenom, PListeAuteurs lauteurs);

PListeAuteurs ajouter_auteur(PAuteur auteur,
   PListeAuteurs *plauteurs, int *ajoute);

PListeArticles ajouter_en_tete(PArticle article,
   PListeArticles articles);

PArticle trouver_article(const char *titre,
   PListeArticles larticles);

PABRAuteurs construire_abr_auteurs(PListeArticles
   larticles);

PABRAuteurs ajouter_abr_auteurs(PAuteur auteur,
   PArticle article, PABRAuteurs abr);
```

```
int comparer_auteurs(PAuteur p1, PAuteur p2);

PABRAuteurs chercher_abr(char *nom, char *prenom,
   PABRAuteurs pabra);

int calcul_HIndex(PAuteur auteur, PABRAuteurs abr,
   PListeArticles larticles);

int nb_citations(PArticle article, PListeArticles
   larticles);

int compter_articles(PListeArticles larticles);

PAuteur lire_auteur(FILE *f);

PArticle lire_article(FILE *f, PListeAuteurs *
   pplauteurs);

void lire_base_articles(const char *fichier,
   PListeAuteurs *plauteurs, PListeArticles *
   plarticles);

void liberer_auteur(PAuteur pauteur);

void liberer_article(PArticle particle);

void liberer_liste_articles(PListeArticles larticles
   , int lib_article);

void liberer_liste_auteurs(PListeAuteurs lauteurs,
   int lib_auteur);

void liberer_abr_auteurs(PABRAuteurs pabra);



#endif
```

```
#ifndef _DOCUMENT_H_
#define _DOCUMENT_H_

#include <stdio.h>

typedef void (*Afficher)(void *data);
typedef void (*Detruire)(void *data);
typedef void * (*Lire)(FILE *f);

typedef struct _document *PDocument;
typedef struct _document {
  void *data;
  Afficher afficher;
  Detruire detruire;
  Lire lire;
} Document;

typedef struct _liste_documents *PListeDocuments;
typedef struct _liste_documents {
  PDocument doc;
  PListeDocuments suivant;
} ListeDocuments;

void afficher_documents(PListeDocuments ldocs);
void liberer_documents(PListeDocuments ldocs);

PListeDocuments lire_docs(const char *nom_fichier);

void afficher_article(void *data);
void detruire_article(void *data);
void *lire_article(FILE *f);

void afficher_ouvrage(void *data);
void detruire_ouvrage(void *data);
void *lire_ouvrage(FILE *f);

#endif
```