Documentation Mini Projet Langage C

Caculli Giorgio, Jedrzej Tyranowski
Haute École de Louvain en Hainaut (HELHa)

13 décembre 2020

Résumé

Documentation pour le projet de Langage procédural sur les listes chaînées. Projet basé sur le concept d'un centre de formations.

Mots-clés : liste, chaînée, c, noeud, centre, formation, tête

Table des matières

1 Introduction

1.1 Le langage C

La langage de programmation utilisé lors du développement et la mise en œuvre du programme est le `ANSI-C`. Les différentes versions du langage disponibles lors du développement de ce programme sont :

- **ANSI-C** : La première vérsion standardisée par le **American National Standard Institute**, abrégé en **ANSI** dans ce document, du langage `C` publiée en 1990.
- **C-99** : Révision de la version ANSI pour permettre aux développeurs d'utiliser les commentaires `//`, les booléans grâce à la librairie `<stdbool.h>`, la déclaration des int directement dans la boucle `for`, et d'autres modérnisations de la syntaxe.
- **C-11** : Mise à jour du langage `C` pour permettre le support des `thread` afin de pouvoir faire du multi-threading.
- **C-17** : Révision de la version **C-11** qui n'ajoute aucune nouvelle fonctionnalité, mais corriges beaucoup bugs présents dans la version 11.

Dans les différentes applications que l'on a fait dans le cours de Langage procédural, la plupart des interactions que l'on a eu avec le langage `C`, notammente le fait de devoir déclarer un `int` avant une boucle `for`, ressemblaient fortement à l'ANSI-`C`. C'est pourquoi nous avons choisi d'utiliser cette vérsion là.

1.2 Fonctions générales utilisées

Différentes fonctions ont été utilisée lors du développement de ce programme, ainsi que des macros pour permettre au compilateur de reconnaître le système d'exploitation dans lequel le logiciel compilé tournera. Voici une liste des fonctions clés utilisées :

- `printf()` : Fonction qui prend en paramètre un `const char *__format` et `...`, un nombre indéterminé de paramètres, qui sert à afficher la châine de caractères `format` dans la console.
  Exemple :

```c
#include <stdio.h>
int main() {
  printf( "Hello World!" ); /* Affiche "Hello World!" dans la console */
  return 0;
}
```

- `fprintf()` : Fonction qui prend en paramètre un `FILE *__stream`, un `const char *__format` et `...`, un nombre indéterminé de paramètres, qui sert à écrire la chaîne de caractères `format` vers le flux spécifique `stream` et extrapoler les informations des paramètres des arguments supplémentaires qui seront ajouté dans `format`.
  Exemple :

```c
#include <stdio.h>
int main() {
  File *ficher_sortie = fopen( "fichier_sortie.txt", "w" );
  /* On cree et ouvre un fichier nomme fichier_sortie.txt en ecriture */
  fprintf( ficher_sortie, "Hello World!\n" );
  /* Ecrit "Hello World!" dans le ficher "fichier_sortie.txt" mais pas dans la console
     */
  fprintf( stdout, "Hello World!\n" );
  /* Affiche "Hello World!" dans la console mais pas dans le fichier de sortie */
  return 0;
}
```

- `scanf()` : Fonction qui prend en paramètre un `const char *__format` et `...`, un nombre indéterminé de paramètres, qui sert à extrapoler une entrée du clavier et stocker les informations extrapolées dans les paramètres déclarés dans la fonction.
  Exemple :

```c
#include <stdio.h>
int main() {
  char prenom[50];
  int age;
  printf( "Comment t'appels-tu ? " );
```

```
6   scanf( "%s", prenom ); /* Admettons que l'utilisateur insere "Jedrzej" */
7   printf( "Quel age as-tu %s ? ", prenom );
8   scanf( "%d", &age ); /* Admettons que l'utilisateur insere "21" */
9   printf( "Salut %s, je vois que tu as %d ans!\n", prenom, age );
10  /* Affiche "Salut Jedrzej, je vois que tu as 21 ans!" dans la console */
11  return 0;
12 }
```

— `fscanf()` : Fonction qui prend en paramètre un `FILE *__stream`, un `const char *__format` et `...`, une quantité indéterminée de paramètres qui servent à satisafaire ceux demandés dans `format`. Cette fonction sert à extrapoler des entrées à partir du flux `stream`, remplir toutes les informations dans `format` et stocker les informations extrapolées dans les paramètres déclarés après le `format`.
Exemple :

```
1  /* Contenu dans ficher_in.txt */
2  /* Jedrzej 21 */
3  #include <stdio.h>
4  int main() {
5    char prenom[50];
6    int age;
7    FILE *ficher_in = fopen( "fichier_entree.txt", "r" );
8    /* On ouvre un fichier nomme fichier_entree.txt en lecture */
9    fscanf( fichier_in, "%s %d\n", prenom, &age );
10   /* On lit le contenu de fichier_entree.txt */
11   printf( "Salut %s, je vois que tu as %d ans\n", prenom, age );
12   /* Affiche "Salut Jedrzej, je vois que tu as 21 ans!" dans la console */
13   return 0;
14 }
```

— `fgets()`
— `fopen()`
— `fclose()`
— `strcpy()` : Fonction qui prend en paramètre un `char *__dest` et un `const char *__src`, soit la destination dans laquelle on souhaite stocker la chaîne de caractères, et la source par laquelle le compilateur va reprende la chaîne de caractères.
Exemple :

```
1  #include <stdio.h>
2  #include <string.h>
3  int main() {
4    char prenom[10];
5    char source[10] = "Giorgio";
6    strcpy( prenom, source );
7    printf( "%s\n", prenom ); /* Affichera "Giorgio" dans la console */
8    return 0;
9  }
```

— `sizeof()` : Fonction qui renvoie la quantité de mémoire (en bits) qu'une entité va occuper dans la Random Access Memory, ou mémoire vive en français, abrégé en RAM dans ce document.
Exemple

```
1  #include <stdio.h>
2  int main() {
3    printf( "Taille de char: %lu\n", sizeof( char ) );
4    /* Affiche 1 dans la console */
5    printf( "Taille de int: %lu\n", sizeof( int ) );
6    /* Affiche 4 dans la console */
7    printf( "Taille de log: %lu\n", sizeof( long ) );
8    /* Affiche 8 dans la console */
9    printf( "Taille de int[4]: %lu\n", sizeof( int[4] ) );
10   /* Affiche 16 dans la console, soit 4*4=16 */
11   return 0;
12 }
```

— `memcpy()` : Fonction qui prend en paramètre un `void *__dest`, un `const void *__src`, et un `size_t __n`. Le pointeur `void *__dest` est un paramètre qui accept tout type d'attribut, même cas pour le pointeur void `void *__src`. Les informations stockées dans `src` seront copiées vers `dest`, mais que la quantité maximale choisie par le `size_t __n`. `size_t` étant un entier non signé (qui ne peut pas avoir de valeur négative).
Exemple

```c
#include <stdio.h>
#include <string.h>
int main() {
  int src[] = {1, 2, 3};
  size_t n = sizeof(src) / sizeof(src[0])
  /* Calcul qui renvoi 3 car la taille d'un int = 4, il y a 3 int dans src donc 4*3 =
    12; on divise la taille totale du vecteur, soit 12, par la taille du premier element
     du vecteur, soit 4, donc 12 / 4 = 3 */
  int dest[n]; /* On initialiser le vecteur dest avec la meme taille src, ici 3 */
  memcpy( dest, src, sizeof(dest) ); /* On copie les informations de src vers dest */
  int i;
  for( i = 0; i < n; i++ ) {
    printf( "%d ", dest[i] ); /* Affiche 1 2 3 dans la console */
  }
  return 0;
}
```

- `calloc()`
- `getchar()`
- `system()`

1.2.1 Qu'est-ce l'allocation de mémoire dynamique ?

1.2.2 Qu'est-ce `calloc()` ?

1.2.3 Qu'est-ce `malloc()` ?

1.2.4 Pourquoi utiliser `calloc()` ?

1.2.5 Pourquoi utiliser `malloc()` ?

1.2.6 Syntaxe de `calloc()`

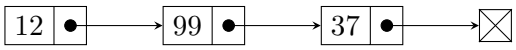
1.2.7 Exemples de `calloc()`

1.2.8 Syntaxe de `malloc()`

1.2.9 Exemples de `malloc()`

1.2.10 Différences entre `calloc()` et `malloc()`

2 Listes chaînées



2.1 Création d'un nouveau nœud

2.2 Insertion d'un nœud dans une liste chaînée

2.3 Suppression d'un nœud d'une liste chaînée

2.4 Affichage d'une liste chaînée

3 Énoncé

4 Programme

4.1 Mode d'emploi

5 Code

5.1 Structures

```c
typedef struct personne
{
  int id;
  char nom[25];
  char prenom[25];
  int formateur;
  int nb_formations;
  int formations[30];
  int nb_jours_indisponible;
  int jours_indisponible[7];
  int reduction;
  int val_reduction;
} personne;
```

```c
typedef struct noeud_db_personne
{
  personne *p;
  struct noeud_db_personne *next;
} noeud_db_personne;
```

```c
typedef struct db_personne
{
  noeud_db_personne *head;
} db_personne;
```

```c
typedef struct noeud_formation
{
  personne *p;
  struct noeud_formation *next;
} noeud_formation;
```

```c
typedef struct formation
{
  int id;
  char nom[40];
  float prix;
  int nb_jours;
  int jours[7];
  float heures[24];
  float durees[10];
  int nb_prerequis;
  int prerequis[10];
  noeud_formation *head;
} formation;
```

```c
typedef struct noeud_db_formation
{
  formation *f;
  struct noeud_db_formation *next;
} noeud_db_formation;
```

```c
typedef struct db_formation
{
  noeud_db_formation *head;
} db_formation;
```

5.2 Fonctions

```
1 personne *creer_personne( char nom[], char prenom[], int formateur );
```

```
1 void afficher_personne( personne *p );
```

```
1 db_personne *creer_db_personne();
```

```
1 void ajouter_db_personne( db_personne *db, personne *p );
```

```
1 int supprimer_db_personne( db_personne *dbp, int id );
```

```
1 void afficher_db_personne( db_personne *db );
```

```
1 personne *get_personne( db_personne *db, char nom[], char prenom[], int formateur );
```

```
1 formation *creer_formation( char nom[], float prix );
```

```
1 int ajouter_formation( formation *f, personne *p );
```

```
1 int supprimer_personne_de_formation( formation *f, int id );
```

```
1 void afficher_formation( formation *f );
```

```
1 db_formation *creer_db_formation();
```

```
1 void ajouter_db_formation( db_formation *db, formation *f );
```

```
1 int supprimer_db_formation( db_formation *dbf, int id );
```

```
1 formation *get_formation( db_formation *dbf, char nom_formation[] );
```

```
1 void afficher_db_formation( db_formation *dbf );
```

```
1 void menu_creer_formation( db_formation *f );
```

```
1 void menu_creer_personne( db_personne *p );
```

```
1 int menu_creer( db_formation *f, db_personne *p );
```

```
1 void menu_ajouter_formation( db_formation *f, db_personne *p );
```

```
1 void menu_supprimer_personne( db_formation *dbf, db_personne *dbp );
```

```
1 void menu_supprimer_formation( db_formation *dbf, db_personne *dbp );
```

```
1 int menu_supprimer_personne_de_formation( db_formation *dbf );
```

```
1 int menu_supprimer( db_formation *dbf, db_personne *dbp );
```

```
1 int menu_affichage( db_formation *f, db_personne *p );
```

```
1 int menu( db_formation *f, db_personne *p );
```

```
1 int main( void );
```

Glossaire

ANSI American National Standard Institute. 3

RAM Random Access Memory. 3