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 MACRO pour permettre au compilateur de reconnaître le système d'exploitation dans lequel le logiciel compilé tournera. Voici une liste des fonctions clés utilisées :

- `fopen()` : Fonction qui sert à ouvrir un flux, plus précisement, un fichier.
  Exemple :

```c
/* Admettons que le fichier donnees.dat existe */
#include <stdio.h>
int main() {
  FILE *fichier_in = fopen( "donnees.dat", "r" );
  /* On ouvre le fichier donnees.dat en lecture */
  FILE *fichier_out = fopen( "resultats.txt", "w" );
  /* Si le fichier resultats.txt n'existe pas on le cree, s'il existe toute information
    presente est ecrasee, puis on y accede en ecrite */
  return 0;
}
```

- `fclose()` : Fonction qui sert à fermer tout flux ouvert.
  Exemple :

```c
/* Admettons que le fichier donnees.dat existe */
#include <stdio.h>
int main() {
  FILE *fichier_in = fopen( "donnees.dat", "r" );
  /* On ouvre le fichier donnees.dat en lecture */
  FILE *fichier_out = fopen( "resultats.txt", "w" );
  /* Si le fichier resultats.txt n'existe pas on le cree, s'il existe toute information
    presente est ecrasee, puis on y accede en ecrite */
  fclose( fichier_out );
  fclose( fichier_in );
  /* On ferme les fichiers lorsqu'on doit plus travailler avec eux */
  return 0;
}
```

- `printf()` : Fonction qui sert à afficher une châine de caractères dans la console.
  Exemple :

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

— `fprintf()` : Fonction qui sert à écrire une chaîne de caractères vers un flux spécifique.
Exemple :

```c
#include <stdio.h>
int main() {
  File *ficher_sortie = fopen( "fichier_sortie.txt", "w" );
  /* On cree et on 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 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 ? " );
  scanf( "%s", prenom ); /* Admettons que l'utilisateur insere "Jedrzej" */
  printf( "Quel age as-tu %s ? ", prenom );
  scanf( "%d", &age ); /* Admettons que l'utilisateur insere "21" */
  printf( "Salut %s, je vois que tu as %d ans!\n", prenom, age );
  /* Affiche "Salut Jedrzej, je vois que tu as 21 ans!" dans la console */
  return 0;
}
```

— `fscanf()` : Fonction qui sert à extrapoler des entrées à partir d'un flux spécifique en respectant une structure précise, et stocker les informations extrapolées dans des paramètres déclarés dans la fonction.
Exemple :

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

— `fgets()` : Fonction qui a le même principe que fscanf, mais garde les espaces.
Exemble :

```c
/* Contenu dans fichier_entree.txt */
/* Caculli Giorgio 23 */
#include <stdio.h>
int main() {
  char nom_prenom[15];
  int age;
  FILE *fichier_in = fopen( "fichier_entree.txt", "r" );
  fgets( nom_prenom, 16, fichier_in );
  /* Lit les 15 caracteres depuis le fichier de donnees fichier_entree.txt */
  fscanf( fichier_in, "%d", &age );
  /* Lit l'age depuis le fichier de donnees fichier_entree.txt */
  printf( "%s %d\n", nom_prenom, age );
  /* Affiche "Caculli Giorgio 23" dans la console */
  return 0;
}
```

— `strcpy()` : Fonction qui sert à copier une chaîne de caractères vers une autres chaîne de caractères.
Exemple :

```c
#include <stdio.h>
#include <string.h>
int main() {
  char prenom[10];
  char source[10] = "Giorgio";
  strcpy( prenom, source );
  printf( "%s\n", prenom ); /* Affiche "Giorgio" dans la console */
  return 0;
}
```

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

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

— `memcpy()` : Fonction qui copiés n octets depuis une zone mémoire vers une autre zone mémoire.
Exemple :

```c
#include <stdio.h>
#include <string.h>
int main() {
  int src[] = {1, 2, 3};
  size_t n = sizeof(src) / sizeof(src[0])
  /* En sachant que la taille d'un int = 4 bytes, et qu'il y a 3 int dans src, on
    obtient 4 * 3 = 12 bytes utilises. On divise la taille totale du vecteur, soit 12,
    par la taille du premier element du vecteur, soit 4, donc 12 / 4 = 3 elements dans
    le vecteur */
  int dest[n]; /* On initialiser le vecteur dest avec la meme taille de src, ici 3 */
  memcpy( dest, src, sizeof(dest) ); /* On copie les informations de src vers dest */
  int i;
  for( i = 0; i < n; i++ ) {
    /* On parcourt tous les elements dans dest, du premier (indice 0) jusqu'au troisieme
     (indide 2) */
    printf( "%d ", dest[i] ); /* Affiche 1 2 3 dans la console */
  }
  return 0;
}
```

— `getchar()` : Fonction qui sert à lire un caractère depuis un flux.
Exemple :

```c
#include <stdio.h>
int main() {
  char lettre;
  printf( "Inserez une lettre: " ); /* Admettons que l'utilisateur insere "a" */
  char c = getchar();
  printf( "Lettre inseree : %c\n", c );
  /* Affiche "a" dans la console */
}
```

— `system()` : Fonction qui permet de lancer l'execution d'une commande sur le système d'exploitation hôte.
Exemple :

```c
#include <stdlib.h>
int main() {
  char *commande = "dir";
  system( commande );
  /* Execute la commande "dir", fonction qui sert a afficher ce qui est present dans le
    repertoire */
  return 0;
}
```

— `calloc()` : Fonction qui permet de faire de l'allocation dynamique de mémoire.
Exemple :

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
  int *pointer;
  int i, n;
  printf( "Nombres d'elements a ajouter: " );
  scanf( "%d", &n ); /* Admettons que l'utilisateur insere 3 */
  pointer = ( int * ) calloc( n, sizeof( int ) );
  for( i = 0; i < n; i++ )
  {
    printf( "Entrez le numero N.%d ", i + 1 );
    scanf( "%d", &pointer[i] ); /* Admettons que l'utilisateur insere 1 2 et 3 */
  }
  for( i = 0; i < n; i++ )
  {
    printf( "%d ", pointer[i] );
    /* Affiche 1 2 3 dans la console */
  }
  return 0;
}
```

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

Une allocation dynamique de mémoire est le processus d'alluer de la mémoire lors de l'exécution d'un programme. Il existe quatre fonctions en `C` qui peuvent être utilisée pour allouer et libérer de la mémoire : `calloc()`, `free()`, `realloc()` et `malloc()`. Ces fonctions sont accéssible lors de l'introduction du header `<stdlib.h>` dans le code.

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

`calloc()` est une fonction qui renvoie un pointeur vers un espace mémoire suffisamment libre pour stocker un tableau au nombre d'objets indéterminé et à la taille spécifiée. Si c'est pas le cas, la fonction renverra `NULL`. Le stockage est initialisé à zéro.

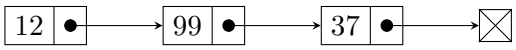
1.2.3 Qu'est-ce `malloc()` ?

`malloc()` est une fonction qui renvoie un pointeur vers un espace mémoire non initialisé. Si l'allocation n'est pas possible, la fonction renverra `NULL`. Si l'espace affecté par l'allocation est saturé, les résultats ne seront pas définis.

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

`malloc()`	`calloc()`
`void * malloc( size_t n );`	`void * calloc( size_t n, size_t size );`
`malloc()` prend un argument (le nombre d'octets)	`calloc()` prend deux arguments (le nombre de blocs et la taille de chaque bloc)
`malloc()` est plus rapide que `calloc()`	`calloc()` prends plus de temps que `malloc()` car la mémoire doit être initialisée à zéro

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

Voici les différentes structures qui ont été utilisée dans la conception du programme :

```
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;
```

Cette structure sert à stocker les informations qui composent une personne quelconque.
Voici ce que représente chaque partie de la structure :

- `int id` : L'identifiant unique de la personne
- `char nom[25]` : Le nom de la personne (25 caractères maximum)
- `char prenom[25]` : Le prénom de la personne (25 caractères maximum)
- `int formateur` : 1 si la personne est un formateur, 0 si la personne est un étudiant
- `int nb_formations` : Le nombre de formations auquel la personne participera
- `int formations[30]` : Vecteur qui stockera les idéntifiants des différentes formations auquel la personne participera (On suppose dans une année, une personne ne peut participer qu'à 30 formations maximum)
- `int nb_jours_indisponible` : Si la personne est un formateur, il se peut qu'il/elle ait des jours d'indisponibilé, cette variable va stocker le nombre de jours où cette personne est indisponile (maximum 7)
- `int jours_indisponibles[7]` : Le vecteur qui stockera les jours auquel le formateur ne sera pas disponible(1 - lundi, 2 - mardi, etc...)
- `int reduction` : Si la personne est un étudiant, il se peut qu'il ait une réduction sur son minérval, 1 s'il a droit à une réduction, 0 si pas
- `int val_reduction` : Le pourcentage de réduction auquel un étudiant à droit

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

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

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

```
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