Le C — Fichiers et structures simples

https://tinyurl.com/mw7wh8sm
https://codeshare.io/deojVK

Nathanael Demacon

Bibliothèque

Les fonctions de manipulation de fichiers se trouvent dans la bibliothèque “`stdio.h`”.

Ouverture

Pour ouvrir un fichier, nous utilisons la fonction “fopen”. La fonction renverra un pointeur de type “FILE”, une structure de données native prévue pour la manipulation de fichiers.

```c
#include <stdio.h>

int main()
{
   FILE *f = fopen("data.txt", "r");

   return 0;
}
```

Ouverture

```c
#include <stdio.h>

int main()
{
    FILE *f = fopen("data.txt", "r");

    return 0;
}
```

Le second paramètre est le mode d'ouverture. Liste des modes d'ouverture :

Mode		Description
r	rb	Ouverture en lecture seule. Le fichier doit exister.
r+	rb+	Ouverture en lecture et écriture. Le fichier doit exister.
w	wb	Ouverture en écriture. Le fichier sera créé s'il n'existe pas. S'il existe, les données seront effacées.
w+	wb+	Ouverture en lecture et écriture. Le fichier sera créé s'il n'existe pas. S'il existe, les données seront effacées.
a		Ouverture en écriture. Le fichier sera créé s'il n'existe pas. S'il existe, les données seront gardées et le curseur d'écriture sera mis à la fin.
a+		Ouverture en lecture et écriture. Le fichier sera créé s'il n'existe pas. S'il existe, les données seront gardées, le curseur d'écriture sera mis à la fin et le curseur de lecture sera mis au début.

Ajouter "b" au mode d'ouverture permet d'ouvrir le fichier en mode binaire.

Par défaut le système d'exploitation gèrera les sauts de lignes de façon spécifique ce qui risque de retirer certains caractères lors de la lecture. Le mode binaire permet de désactiver cette fonctionnalité.

Vérification

Il est important de vérifier que le fichier a bien été ouvert avant de faire des modifications dessus. Un fichier qui n'a pas réussi à être ouvert renverra un pointeur NULL.

```c
#include <stdio.h>

int main()
{
   FILE *f = fopen("data.txt", "r");
   if (f == NULL)
   {
       printf("Error opening file data.txt\n");
   }
   else
   {
       printf("File opened successfully\n");
   }

   return 0;
}
```

Fermeture

Quand un fichier a été ouvert, il est important de le fermer lorsqu'il ne sera plus utilisé. Pour ça nous utilisons la fonction "fclose".

```c
#include <stdio.h>

int main()
{
   FILE *f = fopen("data.txt", "r");
   if (f == NULL)
   {
       printf("Error opening file data.txt\n");
   }
   else
   {
       printf("File opened successfully\n");
       fclose(f);
   }

   return 0;
}
```

Attention à ne pas utiliser "fclose" sur un fichier non ouvert !

Écriture

Pour écrire dans un fichier, il faut l'ouvrir avec un mode qui supporte l'écriture.

```c
#include <stdio.h>

int main()
{
    FILE *f = fopen("data.txt", "w");

    return 0;
}
```

Écriture

Plusieurs fonctions sont à votre disposition pour écrire dans un fichier.

Fonction	Description
fputc	Ajoute un caractère à la position actuelle du curseur d'écriture.
fputs	Ajoute une chaîne de caractère à la position actuelle du curseur d'écriture.
fprintf	Ajoute une chaîne de caractère formatée à la position actuelle du curseur d'écriture.

Écriture
fputc

```c
#include <stdio.h>

int main()
{
   FILE *f = fopen("data.txt", "w");
   if (f != NULL)
   {
       fputc('H', f);
       fputc('e', f);
       fputc('l', f);
       fputc('l', f);
       fputc('o', f);
       fclose(f);
   }

   return 0;
}
```

Écriture
fputs

```c
#include <stdio.h>

int main()
{
    FILE *f = fopen("data.txt", "w");
    if (f != NULL)
    {
        fputs("Hello", f);
        fclose(f);
    }

    return 0;
}
```


```
Hello
```

Écriture
fprintf

```c
#include <stdio.h>

int main()
{
    FILE *f = fopen("data.txt", "w");
    if (f != NULL)
    {
        fprintf(f, "Hello, %d!", 42);
        fclose(f);
    }

    return 0;
}
```


```
Hello, 42!
```

Ajout d’une ligne à la fin du fichier

```c
#include <stdio.h>

int main()
{
   FILE *f = fopen("data.txt", "a");
   if (f != NULL)
   {
       fputs("Hello\n", f);
       fclose(f);
   }

   return 0;
}
```


```
Hello
Hello
Hello
```

Lecture

Pour lire un fichier, il faut l'ouvrir avec un mode qui supporte la lecture.

```c
#include <stdio.h>

int main()
{
    FILE *f = fopen("data.txt", "r");

    return 0;
}
```

Lecture

Plusieurs fonctions sont à votre disposition pour lire un fichier.

Fonction	Description
fgetc	Lis un caractère à la position actuelle du curseur de lecture.
fgets	Lis une chaîne de caractère à partir de la position actuelle du curseur de lecture jusqu'à la limite donnée ou jusqu'à un saut de ligne.
fscanf	Lis une chaîne de caractères formatée à partir de la position actuelle du curseur de lecture.

Lecture

fgetc

```c
#include <stdio.h>

int main()
{
   FILE *f = fopen("data.txt", "r");
   if (f != NULL)
   {
       char c;
       c = fgetc(f);
       printf("%c", c); // H
       c = fgetc(f);
       printf("%c", c); // e
       c = fgetc(f);
       printf("%c", c); // l
       c = fgetc(f);
       printf("%c", c); // l
       c = fgetc(f);
       printf("%c", c); // o
       fclose(f);
   }

   return 0;
}
```

Lecture
EOF

EOF est l'acronyme de End Of File. C'est un marqueur pour indiquer la fin d'un fichier en cours de lecture.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
  FILE *file = fopen("data.txt", "r");
  if (file == NULL)
  {
    return 1;
  }

  char content[1000] = {0};
  int index = 0;

  char ch = fgetc(file);
  while (ch != EOF)
  {
    content[index++] = ch;
    ch = fgetc(file);
  }

  printf("Content of the file:\n%s\n", content);
  fclose(file);
  return 0;
}
```

Lecture
fgets

```c
#include <stdio.h>

int main()
{
   FILE *f = fopen("data.txt", "r");
   if (f != NULL)
   {
       char buffer[100] = {0};

       fgets(buffer, 100, f);
       printf("%s", buffer); // Hello, 1!

       fgets(buffer, 100, f);
       printf("%s", buffer); // Hello, 2!

       fgets(buffer, 100, f);
       printf("%s", buffer); // Hello, 3!

       fclose(f);
   }

   return 0;
}
```

Quand fgets arrive à la fin d'un fichier la fonction retourne NULL;

data.txt

```
Hello, 1!
Hello, 2!
Hello, 3!
```

Lecture
fscanf

```c
#include <stdio.h>

int main()
{
    FILE *f = fopen("data.txt", "r");
    if (f != NULL)
    {
        int n;
        char data[100] = {0};

        fscanf(f, "%d %s", &n, data);
        printf("%d: %s\n", n, data); // 1: First

        fscanf(f, "%d %s", &n, data);
        printf("%d: %s\n", n, data); // 2: Second

        fscanf(f, "%d %s", &n, data);
        printf("%d: %s\n", n, data); // 3: Third

        fclose(f);
    }

    return 0;
}
```


```
1 First
2 Second
3 Third
```

Déplacements du curseur

Plusieurs fonctions sont à votre disposition pour déplacer la position du curseur dans un fichier ouvert.

Fonction	Description
`ftell`	Renvoie la position actuelle du curseur.
`fseek`	Déplace la position du curseur à l'index donné.
`rewind`	Déplace la position du curseur au début du fichier (index 0).

Attention : on ne doit pas utiliser ces fonctions sur un fichier ouvert en mode “a”.

Déplacements du curseur

ftell

```c
#include <stdio.h>

int main()
{
   FILE *f = fopen("data.txt", "w");
   if (f != NULL)
   {
       printf("%ld\n", ftell(f)); // 0

       fputc('a', f);
       printf("%ld\n", ftell(f)); // 1

       fputc('b', f);
       printf("%ld\n", ftell(f)); // 2

       fclose(f);
   }

   return 0;
}
```

Les fonctions de lecture et d'écriture déplacent automatiquement la position du curseur.

Déplacements du curseur
fseek

La fonction “fseek” prend 2 arguments :

- Un offset en fonction de l’origine (second argument)
- L’origine du déplacement

Origine	Description
SEEK_SET	Début du fichier.
SEEK_CUR	Position actuelle du curseur.
SEEK_END	Fin du fichier.

L’offset (premier argument) peut être positif ou négatif.

```c
#include <stdio.h>

int main()
{
   FILE *f = fopen("data.txt", "r");
   if (f != NULL)
   {
       fseek(f, 5, SEEK_SET);
       printf("%ld\n", ftell(f)); // 5

       fseek(f, 1, SEEK_CUR);
       printf("%ld\n", ftell(f)); // 6

       fseek(f, -3, SEEK_END);
       printf("%ld\n", ftell(f)); // 27

       fclose(f);
   }

   return 0;
}
```

Les structures

Une structure est un type contenant un ensemble de propriétés.

Elles permettent au code d'être plus organisé et maintenable.

Structures
Déclaration

Pour déclarer une structure, nous utilisons le mot clé “struct” suivi de son nom puis de ses propriétés. Par exemple pour une structure “élève” :

```
struct Student
{
    char *firstname;
    char *lastname;
    int age;
};
```

Structures
Déclaration

Pour déclarer une variable de type structure, par défaut il faut rajouter “struct” devant le nom du type.

```c
struct Student
{
    char *firstname;
    char *lastname;
    int age;
};

int main()
{
    struct Student dennis;
    dennis.firstname = "Dennis";
    dennis.lastname = "Ritchie";
    dennis.age = 30;

    return 0;
}
```

Structures
Déclaration

Astuce : on peut simplifier l'affectation des propriétés par défaut via une syntaxe spécifique au C :

```c
struct Student
{
    char *firstname;
    char *lastname;
    int age;
};

int main()
{
    struct Student dennis = {
        .firstname = "Dennis",
        .lastname = "Ritchie",
        .age = 30,
    };

    return 0;
}
```

Structures
Déclaration

Il est aussi possible d'utiliser les listes d'initialisation pour affecter des valeurs par défaut à une structure mais ce n'est pas conseillé dans la plupart des cas.

```c
struct Student
{
    char *firstname;
    char *lastname;
    int age;
};

int main()
{
    struct Student dennis = {"Dennis", "Ritchie", 30};

    return 0;
}
```

Structures
Typedef

Comme c'est pénible de spécifier "struct" à chaque fois, on peut utiliser "typedef" pour créer un alias de cette structure.

```c
// typedef type name;
typedef struct
{
   char *firstname;
   char *lastname;
   int age;
} Student;

int main()
{
   Student dennis;

   return 0;
}
```

Structures
Utilisation

Pour accéder aux propriétés d'une structure, nous utilisons le point :

```c
#include <stdio.h>

typedef struct
{
    char *firstname;
    char *lastname;
    int age;
} Student;

int main()
{
    Student dennis = {
        .firstname = "Dennis",
        .lastname = "Ritchie",
        .age = 30,
    };

    printf("%s %s is %d years old\n", dennis.firstname, dennis.lastname, dennis.age); // Dennis Ritchie is 30 years
old
    return 0;
}
```

Structures

Tableaux

Une structure fonctionne comme tous les autres types, il est donc possible de les utiliser dans des tableaux

```c
#include <stdio.h>

typedef struct
{
    char *firstname;
    char *lastname;
    int age;
} Student;

int main()
{
    Student students[2];

    Student dennis = {
        .firstname = "Dennis",
        .lastname = "Ritchie",
        .age = 30,
    };
    students[0] = dennis;

    Student kamal = {
        .firstname = "Kamal",
        .lastname = "Hennou",
        .age = 53,
    };
    students[1] = kamal;

    for (int i = 0; i < 2; i++)
    {
        Student s = students[i];
        printf(
            "%s %s is %d years old\n",
            s.firstname, s.lastname, s.age);
    }

    return 0;
}
```

Structures
Pointeurs

Pour accéder aux propriétés d'un pointeur de structures, vous pouvez utiliser l'opérateur "->".

```c
#include <stdio.h>

typedef struct
{
    char *firstname;
    char *lastname;
    int age;
} Student;

int main()
{
    Student dennis = {
        .firstname = "Dennis",
        .lastname = "Ritchie",
        .age = 30,
    };

    Student *p = &dennis;
    printf("%s\n", p->firstname); // Dennis

    return 0;
}
```

Structures

Usage avec les fonctions

Tout comme les types classiques, une structure passée en argument d'une fonction sera copiée, il est donc impossible de la modifier.

```c
#include <stdio.h>

typedef struct
{
   char *firstname;
   char *lastname;
   int age;
} Student;

void increment_age(Student s)
{
   s.age++;
}

int main()
{
   Student dennis = {
        .firstname = "Dennis",
        .lastname = "Ritchie",
        .age = 30,
   };

   increment_age(dennis);
   printf("%d\n", dennis.age); // 30

   return 0;
}
```

Structures

Usage avec les fonctions

Toujours comme les types classiques, on peut passer un pointeur de structure en paramètre d'une fonction pour pouvoir la modifier.

```c
#include <stdio.h>

typedef struct
{
   char *firstname;
   char *lastname;
   int age;
} Student;

void increment_age(Student *s)
{
   s->age++;
}

int main()
{
   Student dennis = {
        .firstname = "Dennis",
        .lastname = "Ritchie",
        .age = 30,
   };

   increment_age(&dennis);
   printf("%d\n", dennis.age); // 31

   return 0;
}
```

Structures
Allocation de mémoire

On peut allouer de la mémoire pour stocker des structures de la même manière que pour les autres types. Il ne faut pas oublier de libérer la mémoire quand elle n'est plus utilisée.

```c
#include <stdlib.h>

typedef struct
{
   char *firstname;
   char *lastname;
   int age;
} Student;

int main()
{
   Student *dennis = malloc(sizeof(Student));
   // ...
   free(dennis);

   return 0;
}
```

C’est fini !