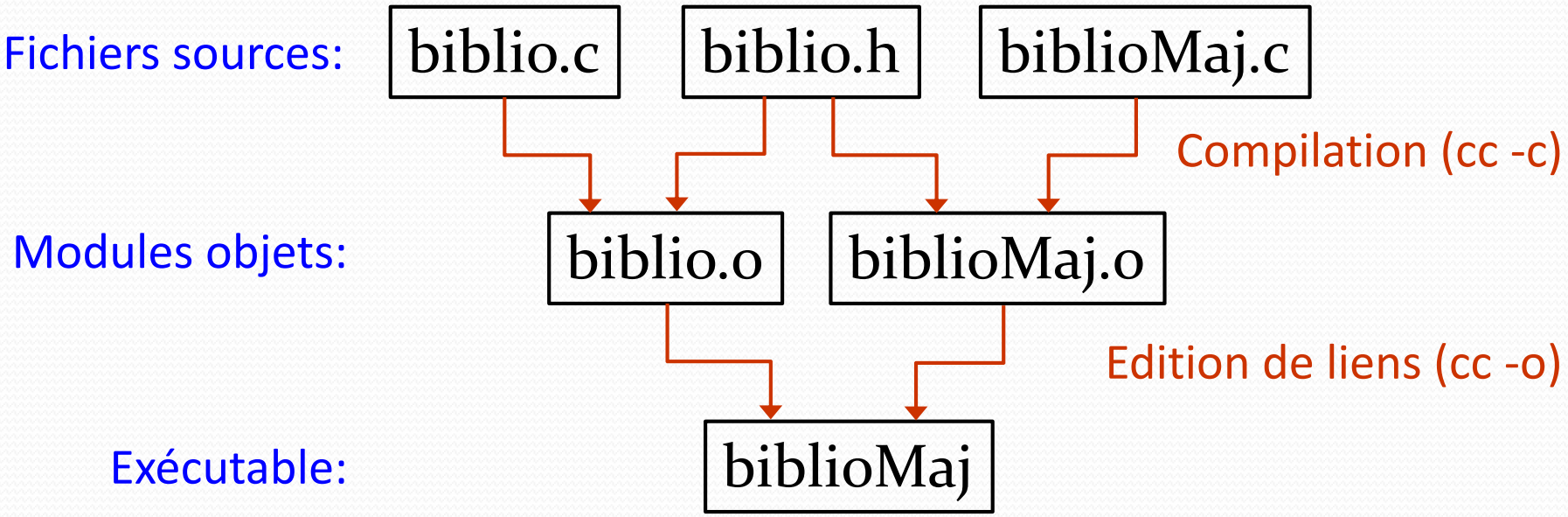
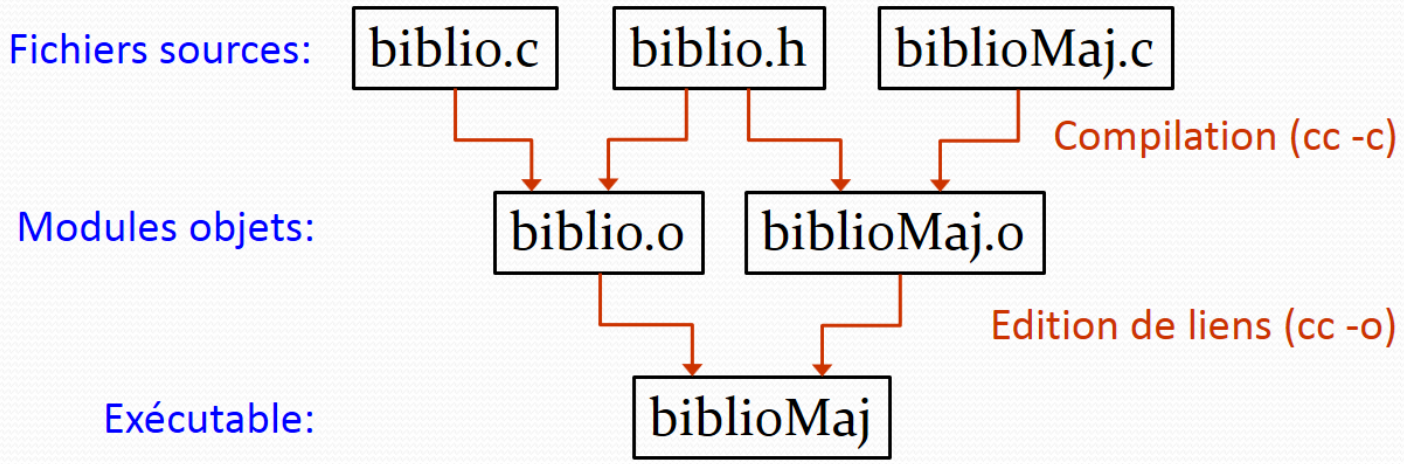
I2010 : langage C (19)

Les fichiers binaires

Makefile


Fichiers sources:
biblio.c
biblio.h
biblioMaj.c
Compilation (cc -c)
Modules objets:
biblio.o
biblioMaj.o
Edition de liens (cc -o)
Exécutable:
biblioMaj

Makefile




```
CFLAGS=-std=c11 -pedantic -Wvla -Wall -Werror -g
biblioMaj : biblioMaj.o biblio.o
	cc $(CFLAGS) -o biblioMaj biblioMaj.o biblio.o
biblioMaj.o : biblioMaj.c biblio.h
	cc $(CFLAGS) -c biblioMaj.c
biblio.o : biblio.c biblio.h
	cc $(CFLAGS) -c biblio.c
clean :
	rm *.o; rm biblioMaj
```

biblio.h (1)

```
#include <stdio.h>
#include <stdbool.h>

#ifndef _BIBLIO_H_
#define _BIBLIO_H_

// Définition de constantes
#define MAX_TITRE          128
#define MAX_AUTEUR         80
#define MAX_EDITEUR        50
#define MAX_LIGNE_LIVRE    MAX_TITRE+MAX_AUTEUR+MAX_EDITEUR+50
```

biblio.h (2)

```
// Définition de types
typedef enum {BD, PO, TH, RO, RH, LF,
              LE, SC, IN, SF, SA, HI} Genre;
typedef enum {TITRE, AUTEUR, ISBN,
              EDITEUR, ANNEE, GENRE} InfoLivre;
typedef struct {
    char titre[MAX_TITRE+1];
    char auteur[MAX_AUTEUR+1];
    long isbn;
    char editeur[MAX_EDITEUR+1];
    int  an;
    Genre genre;
} Livre;

typedef int (*fctcmp)(const void *, const void*);
```

biblio.h (3)

```
// Déclaration de fonctions
bool lireLivre (Livre *l);
char* livre2str (char *s, Livre l);
void afficherBib (Livre *bib, int t);
bool ajouterLivre (Livre **bib, Livre l, int *nbreL, int *taille);
Genre str2genre (char *s);
char* genre2str (Genre g);
bool comparerLivre (Livre *a, Livre *b);
bool lireFichier (FILE *f, Livre **bib, int *nbreL, int *taille);
bool ecrireFichier (FILE *f, Livre *bib, int n);

#endif   // _BIBLIO_H_
```

+ spécifications (PRE / POST / RES) !

biblio.c (1)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "biblio.h"

// Définition de constante
#define NB_LIVRES  10

// Définition de variable globale
char* lesGenres[] = {"Bande dessinée", "Poésie", "Théâtre",
       "Roman", "Roman historique", "Littérature française",
       "Littérature étrangère", "Sciences", "Informatique",
       "Science-fiction", "Santé", "Histoire", NULL};
```

Où définir la variable lesGenres ?

- dans le fichier livre.c : `char* lesGenres[] = ...`
- dans le fichier livre.c : `static char* lesGenres[] = ...`
- dans le fichier livre.h : `char* lesGenres[] = ...`
- dans le fichier livre.h : `static char* lesGenres[] = ...`
- dans le fichier biblioMaj.c : `char* lesGenres[] = ...`
- dans le fichier biblioMaj.c : `static char* lesGenres[] = ...`

Où définir la variable lesGenres ?

- dans le fichier livre.c : `char* lesGenres[] = ...` ✓

  → OK utilisable ailleurs en déclarant dans livre.h: `extern char* lesGenres[];`

- dans le fichier livre.c : `static char* lesGenres[] = ...` ✓

  → OK mais impossible à utiliser en dehors du fichier livre.c

- dans le fichier livre.h : `char* lesGenres[] = ...` ✗

  → définitions multiples lors de l'include de livre.h

- dans le fichier livre.h : `static char* lesGenres[] = ...` ✗

  → pas de conflit mais plusieurs variables *lesGenres* en mémoire (dangereux!)

- dans le fichier biblioMaj.c : `char* lesGenres[] = ...` ✗

  → erreur de compilation du module livre (*lesGenres* inconnu) + modularisation KO

- dans le fichier biblioMaj.c : `static char* lesGenres[] = ...` ✗

  → erreur de compilation du module livre (*lesGenres* inconnu) + modularisation KO

biblio.c (2)

```
/// LECTURE DANS FICHIER BINAIRE

/* Charge les livres d'un fichier
 * PRE: f: fichier binaire contenant des Livre, ouvert en lecture
 *      bib: tableau de livres
 *      nbreL: nombre de livres dans bib
 *      taille: capacité du tableau bib
 * POST: tous les livres de f ont été chargés dans bib et nbreL
 *       et taille (capacité physique) mises à jour
 * RES: renvoie vrai si l'opération s'est réalisée avec succès,
 *      faux sinon (erreur de lecture dans f ou d'ajout dans bib)
 */
bool lireFichier (FILE *f, Livre **bib, int *nbreL, int *taille);
```

biblio.c (2)

```
/// LECTURE DANS FICHIER BINAIRE

// VERSION 1: Lecture des livres de f, livre par livre
bool lireFichier (FILE *f, Livre **bib, int *nbreL, int *taille) {
     Livre l;
     while (fread(&l, sizeof(Livre), 1, f)) {
         if (!ajouterLivre(bib, l, nbreL, taille)) {
             return false;
         }
     }
     if (ferror(f)) {  // ou  if (!feof(f))
         return false;
     }
     return true;
}
```

biblio.c (2)

```
// VERSION 2: Lecture par bloc de NB_LIVRES livres
bool lireFichier (FILE *f, Livre **bib, int *nbreL, int *taille) {
    Livre tmp[NB_LIVRES];
    size_t nread;
    while ((nread = fread(tmp, sizeof(Livre), NB_LIVRES, f) != 0) {
       for (int i=0; i<nread; i++) {
          if (!ajouterLivre(bib, tmp[i], nbreL, taille)) {
               return false;
          }
       }
    }
    if (ferror(f)) {  // ou  if (!feof(f))
       return false;
    }
    return true;
}
```

biblio.c (2)

```
// VERSION 3: Lecture par bloc de NB_LIVRES livres, sans appel à ajouterLivre
// (copie directement dans bib, avec gestion de la mémoire dynamique)
bool lireFichier (FILE *f, Livre **bib, int *nbreL, int *taille) {
    *nbreL = *taille = 0;
    size_t nread = NB_LIVRES;
    while (nread == NB_LIVRES) {
        // allocation de mémoire dynamique à bib
        if (*taille == 0) {
            *taille = NB_LIVRES;
            if ((*bib = (Livre*)malloc((*taille)*sizeof(Livre)))==NULL){
                perror("Malloc");
                return false;
            }
        } else {
            *taille += NB_LIVRES;
            if ((*bib = (Livre*)realloc(*bib, (*taille)*sizeof(Livre)))==NULL){
                perror("Realloc");
                return false;
            }
        }
```

biblio.c (2)

```
// VERSION 3: Lecture par bloc de NB_LIVRES livres, sans appel à ajouterLivre
// (copie directement dans bib, avec gestion de la mémoire dynamique)
bool lireFichier (FILE *f, Livre **bib, int *nbreL, int *taille) {
    *nbreL = *taille = 0;
    size_t nread = NB_LIVRES;
    while (nread == NB_LIVRES) {
        // allocation de mémoire dynamique à bib
        ...
        // lecture de NB_LIVRES livres dans f
        nread = fread(*bib+*nbreL, sizeof(Livre), NB_LIVRES, f);
        *nbreL += nread;
    }
    if (ferror(f)) {  // ou  if (!feof(f))
        return false;
    }
    return true;
}
```

biblio.c (3)

```
/// ECRITURE DANS FICHIER BINAIRE

/* Sauve des livres dans un fichier
 * PRE: f: fichier binaire ouvert en écriture
 *      bib: tableau de livres
 *      n: nombre de livres dans bib
 * POST: tous les livres de bib ont été sauvés dans f
 * RES: renvoie vrai si l'opération s'est réalisée avec succès,
 *      faux sinon (erreur d'écriture) */
bool ecrireFichier (FILE *f, Livre *bib, int n);
```

biblio.c (3)

```
/// ECRITURE DANS FICHIER BINAIRE

// VERSION 1: Ecriture des livres de bib, livre par livre
bool ecrireFichier (FILE *f, Livre *bib, int nbreL) {
     for (int i=0; i<nbreL; i++) {
         if (fwrite(bib+i, sizeof(Livre), 1, f) != 1) {
             return false;
         }
     }
     return true;
}
```

biblio.c (3)

```
// VERSION 2: Ecriture en un bloc de tous les livres de bib
bool ecrireFichier (FILE *f, Livre *bib, int nbreL) {
    return (fwrite(bib, sizeof(Livre), nbreL, f) == nbreL);
}
```

biblioMaj.c (1)

```
#include <stdlib.h>
#include <stdio.h>
#include "livre.h"

int main (int argc, char **argv) {

    Livre *maBib= NULL;
    int nbreLivre = 0;
    int tailleP = 0;
    Livre unLivre;
    char **files = argv + 1;

    FILE *fin = NULL;
    FILE *fout = NULL;
```

biblioMaj.c (2)

```
switch (argc) {
    default :
        fprintf(stderr, "Usage : %s [fin] fout \n", *argv);
        exit(1);
    case 3 :
        printf("Ouverture du fichier '%s' en lecture\n",*files);
        if ((fin = fopen(*files, "rb")) == NULL) {
            perror("Erreur d'ouverture de fichier en lecture");
            exit(10);
        }
        printf("Chargement des livres du fichier '%s'\n",*files);
        if (!lireFichier(fin, &maBib, &nbreLivre, &tailleP)) {
            if (ferror(fin)) {
                perror("Erreur de lecture");
                exit(12);
            }
            else {
                perror("Erreur d'ajout de livre (alloc. dyn.)");
                exit(20);
            }
        }
        files++;
        fclose(fin);
    case 2 :
        printf("Ouverture du fichier '%s' en écriture\n",*files);
        if ((fout = fopen(*files, "wb")) == NULL) {
            perror("Erreur d'ouverture de fichier en écriture");
            exit(11);
        }
}
```

biblioMaj.c (3)

```
    printf("Lecture de livres au clavier\n");
    while (lireLivre(&unLivre)) {
        if (!ajouterLivre(&maBib, unLivre, &nbreLivre, &tailleP)) {
            perror("Erreur d'ajout de livre (allocation de mémoire)");
            exit(20);
        }
    }
    afficherBib(maBib, nbreLivre);

    printf("Tri de la bibliothèque\n");
    qsort(maBib, nbreLivre, sizeof(Livre), (fctcmp)comparerLivre);
    afficherBib(maBib, nbreLivre);

    printf("Ecriture des livres dans le fichier '%s'\n",*(files));
    if (!ecrireFichier(fout, maBib, nbreLivre)) {
        perror("Erreur d'écriture dans le fichier");
        exit(13);
    }

    fclose(fout);
    free(maBib);
}
```