Devoir surveillé

27 octobre 2017 - Durée 1h30

Document autorisé : **Mémento C** vierge de toute annotation manuscrite

Les deux exercices sont indépendants et peuvent être traités dans un ordre quelconque.

Exercice 1. Questions de cours (7 pt)

Modularité et compilation séparée (5 pt)

La modularité est le fait, pour un logiciel, d'être écrit en plusieurs modules/paquetages relativement indépendants les uns des autres.

Q1. (1 pt) Citez quelques avantages d'un tel découpage.

Q2. (1 pt) En quoi consiste la *compilation séparée* ?

L'étape finale du processus de compilation est le regroupement de toutes les données et de tout le code des fichiers objets du programme et des bibliothèques ainsi que la résolution des références inter-fichiers.

Q3. (1 pt) Comment s'appelle cette étape ?

Q4. (1 pt) Que produit-elle ?

Q5. (1 pt) Décrire la structure d'une règle d'un fichier Makefile.

Tests (2 pt)

Q6. (1 pt) Citez deux types de tests.

Q7. (1 pt) Quelle méthode de tests peut-on utiliser lorsqu'on ne dispose que de la spécification et de l'exécutable d'un programme ?

Exercice 2. Programmation (13 pt)

On considère le programme dont la spécification est la suivante :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  /*
4  Cette fonction permet d'ouvrir un fichier dont le nom est passé
5  en argument : nom_fichier
6  Le mode d'ouverture est spécifié par l'argument mode:
7         "r" correspond à une ouverture en lecture
8         "w" correspond à une ouverture en écriture
9  */
10 FILE * ouvrir_fichier (char* nom_fichier, char* mode) ;
11
12 /* Cette fonction permet de fermer un fichier ouvert */
13 void fermer_fichier (FILE * f);
14
15 /*
16 Cette fonction initialise un tableau d'entiers depuis un fichier f.
17 Résultat :
18  - Les valeurs lues sont placées dans le tableau tab
19  - nb contient la taille du tableau lu
20 */
21 void lire (FILE * f, int * tab, int * nb );
22
23 /* Cette fonction sauvegarde un tableau d'entiers dans un fichier */
24 void ecrire (FILE * f, int * tab, int nb);
25
26 /* Cette fonction trie un tableau d'entiers de nb elements */
27 void operation_tri (int * tab, int nb);
```

L'implémentation de ce programme est fournie en annexe p. 3.

Tests et correction du programme (8 pt)

Q8. (1 pt) Que fait ce programme ?

Q9. (2 pt) Quel est le format des entrées du programme ?

Q10. (2 pt) Décrire un jeu de tests permettant de tester ce programme.

Q11. (1,5 pt) L'implémentation de la fonction **ouvrir_fichier** correspond t-elle à sa spécification ?

1. Argumenter votre réponse.
2. Proposer une correction de l'implémentation si vous estimez qu'elle n'est pas conforme.

Q12. (1,5 pt) L'implémentation de la fonction **fermer_fichier** correspond t-elle à sa spécification ?

1. Argumenter votre réponse.
2. Proposer une correction de l'implémentation si vous estimez qu'elle n'est pas conforme.

Modularité et Compilation séparée (5 pt)

Le programme fourni pourrait être structuré en plusieurs modules.

Q13. (2 pt) Donnez une structuration possible du programme fourni en plusieurs modules, en donnant pour chaque module :

— la liste des fonctions qu'il contient ;
— ses dépendances.

NB : il n'est pas demandé de réécrire entièrement chaque module. Seule la liste des fonctions doit être donnée.

Q14. (3 pt) Écrivez un Makefile pour compiler le programme ainsi structuré.

Annexe : implémentation du programme

```c
#include "header.h"
#define NMAX 500

FILE * ouvrir_fichier (char* nom_fichier, char* mode) {
  FILE * f;
  f = fopen(nom_fichier, "rw");
  return f;
}

void fermer_fichier (FILE * f) {
  f = NULL;
}

void lire (FILE * f, int * tab, int * nb_elem) {
  int i;
  fscanf(f, "%d", nb_elem);
  for( i = 0 ; i < *nb_elem ; i++){
    fscanf(f, "%d", &(tab[i]));
  }
}

void ecrire (FILE * f, int * tab, int nb_elem) {
  int i;
  fprintf(f, "%d\n", nb_elem);
  for( i = 0 ; i < nb_elem; i++){
    fprintf(f, "%d\n", tab[i]);
  }
}

void operation_tri (int* tab, int nb_elem) {
  int i,j, tmp;
  for(i = 0 ; i < nb_elem; i++) {
    for (j = i ; j > 0 ; j--)
      if (tab[j] < tab[j-1]){
        tmp = tab[j]; tab[j] = tab [j-1];
        tab[j-1] = tmp;
      }
  }
}

int main (int argc, char ** argv ) {
  FILE* f1;
  FILE* f2;
  int tab[NMAX];
  int nb_elem;

  if (argc != 3){
    printf ("Erreur, nombre d'arguments incorrect\n");
    return -1;
  }

  f1 = ouvrir_fichier (argv[1], "r");
  lire(f1, tab , &nb_elem);
  fermer_fichier (f1);

  printf("Tri du tableau\n");
  operation_tri ( tab, nb_elem);

  f2 = ouvrir_fichier (argv[2], "w");
  ecrire (f2, tab, nb_elem);
  fermer_fichier (f2);
  return 0;
}
```