FACULTÉ DES SCIENCES ET TECHNIQUES
DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Ingénierie Logicielle et Intégration
des Systèmes Informatiques

Compte Rendu

Création d'une base des données pour la gestion d'une promotion ILISI

Filière Ingénieur :

Ingénierie Logicielle et Intégration des Systèmes Informatiques

Réalisé par :

OUTGOUGA Jalal eddine

ZADDI Abdelmajid

Encadré par :

Prof. Abdelkrim BEKKHOUCHA

2022/2023

Table de matières

I- L ’analyse fonctionnel :

a) Structures adoptées :

Les structures définies dans ce programme sont les suivantes :

La structure Etudiant : Elle définit les informations sur un étudiant, telles que son nom, prénom, CNE, CNI, les notes qu'il a obtenues lors des deux semestres, s'il est en année de réserve ou non, l'année scolaire en question et sa moyenne annuelle.

La structure Module : Elle définit les informations sur un module scolaire, telles que son nom, le nombre de notes valides, le nombre de notes non valides, la note maximale et la note minimale.

La structure Ilisi : Elle définit un tableau d'étudiants divisé en trois années différentes, ainsi que le nombre total d'étudiants.

Le tableau Module Md : Il s'agit d'un tableau de 40 modules différents

Pour accéder à un module on doit savoir l’année courant selon cette formule : **indice du module +16*l’année.**

Exemples :

Le premier module de la première année : 0+16*(1-1) =0

Le premier module de la deuxième année : 0+16*(2-1) =16

Le cinquième module de la première année : 5+16*(1-1) =5

```
/* Structure pour stocker les informations d'un étudiant */
typedef struct
{
   char Nom[20]; // Le nom de l’étudiant
   char Prenom[20]; // Le prenom de l’étudiant
   char CNE[20]; // Numéro d'identification national pour les étudiants marocains
   char CNI[20]; // Carte nationale d'identité
   float notes[15]; // Les notes de deux semestres
   int Reserve; // Indique si l'étudiant est en année de réserve ou non
   int Annee ; // L’année en question
   float MoyAnnee; // La moyenne de l’année
} Etudiant;
```

```
/* Structure pour stocker les informations d'un module */
typedef struct
{
   char Nom[140]; // Le nom du module
   int Valide; // Nombre d'étudiants ayant validé le module
   int Nvalide; // Nombre d'étudiants n'ayant pas validé le module
   float NoteMax; // La note la plus élevée obtenue pour ce module
   float NoteMin; // La note la plus basse obtenue pour ce module
} Module;

/* Structure pour stocker les informations d'une année scolaire */
typedef struct
{
   Etudiant Etud[3][MAX]; // Tableau 2D pour stocker les informations de chaque étudiant selon l'année
   int NbEtud; // Nombre d'étudiants enregistrés pour cette année scolaire
} Ilisi;


Module Md[40]; // Tableau pour stocker les informations de chaque module
```

a) Fonctions adoptée :

La fonction **initialiserEtudiant()** retourne un pointeur sur une structure de type Etudiant, qui a été allouée dynamiquement dans la mémoire avec malloc(). Les champs de la structure Reserve, Annee et MoyAnnee sont initialisés à zéro.

La fonction **InitiaModule(Module Md[40])** prend en entrée un tableau de 40 structures de type Module. Elle initialise les champs de chaque structure avec NoteMax à zéro, NoteMin à 20, Nvalide à zéro et Valide à zéro.

La fonction **initialiserIlisi()** retourne un pointeur sur une structure de type **Ilisi**, qui a été allouée dynamiquement dans la mémoire avec malloc(). Le champ NbEtud de la structure est initialisé à zéro, et pour chaque étudiant dans le tableau Etud[3][MAX], la fonction **initialiserEtudiant()** est appelée pour initialiser les champs de la structure Etudiant.

La fonction **insererEtudiant** permet d'insérer un étudiant dans une structure de données de type Ilisi. Elle vérifie d'abord si le nombre d'étudiants ne dépasse pas la limite MAX,

puis enregistre l'étudiant dans la structure et incrémente le nombre d'étudiants. La fonction retourne un pointeur vers la structure modifiée.

La fonction **afficherNomsEtudiants** permet d'afficher les informations de tous les étudiants dans une structure de données de type Ilisi. Elle affiche le nom, prénom et CNE de chaque étudiant, ainsi que leurs notes et leur moyenne annuelle.

La fonction **VerifierNote** permet de vérifier une note et de mettre à jour les informations associées à un module. Elle vérifie si la note est supérieure ou égale à 12 (validée), incrémente le compteur de notes validées si nécessaire, met à jour la note maximale et minimale pour le module, et retourne la structure modifiée.

La fonction **insererInfoEtudiant** permet de saisir les informations d'un étudiant Elle demande le prénom, nom, CNE, CNI, année, statut de réserve, et les notes de l'étudiant pour les deux semestres. Elle utilise la fonction **VerifierNote** pour vérifier chaque note et mettre à jour les informations associées aux modules. La fonction retourne un pointeur vers la structure modifiée.

Les fonctions **storeData et readFile** gèrent la sauvegarde et la lecture des données d'étudiants à partir d'un fichier texte. La fonction storeData prend en entrée un tableau d'objets Etudiant et le nombre d'étudiants, puis écrit ces informations dans un fichier selon l'annéé en utilisant la fonction fprintf. Si le fichier ne peut pas être ouvert, une erreur est affichée et le programme s'arrête.

La fonction **readFile** lit les données du fichier selon l'année et les stocke dans un objet Ilisi et un tableau d'objets Module. Si le fichier ne peut pas être ouvert, une erreur est affichée et le programme s'arrête. Les données sont lues à l'aide de la fonction fscanf et stockées dans les champs correspondants des objets. La fonction MoyenneValid est ensuite appelée pour calculer la moyenne de chaque étudiant.

La fonction **LireModule** lit les informations de module à partir d'un fichier texte nommé "Module.txt". Si le fichier ne peut pas être ouvert, une erreur est affichée et le programme s'arrête. Les informations de module sont lues et stockées dans un tableau d'objets Module.

PlusValide retourne le module avec le taux de validation le plus élevé parmi les 16 modules d'une année donnée.

MoinsValide retourne le module avec le taux de validation le plus bas parmi les 16 modules d'une année donnée.

NotePlusEleve retourne le module avec la note maximale la plus élevée parmi les 16 modules d'une année donnée.

NoteMoinsEleve retourne le module avec la note minimale la plus basse parmi les 16 modules d'une année donnée.

MoyenneValid Cette fonction calcule la moyenne annuelle de l'étudiant en divisant la somme de ses notes par 16, puis stocke ce résultat dans le champ "MoyAnnee" de la structure Etudiant. Ensuite, la fonction appelle Valid et enregistre le résultat dans le champ "Réserve" de l'étudiant.

Valid Cette fonction détermine si l'étudiant a réussi ou non. Si la moyenne annuelle de l'étudiant est supérieure ou égale à 12 et que toutes ses notes sont supérieures ou égales à 10, alors l'étudiant est considéré comme ayant réussi. Sinon, l'étudiant est considéré comme n'ayant pas réussi.

Chercher Cette fonction recherche un étudiant en comparant le CNE de l'étudiant recherché avec les CNE des étudiants stockés dans le tableau "Etud" de la structure Ilisi. Si l'étudiant est trouvé, la fonction affiche ses informations en appelant la fonction **afficherEtudiant** Si l'étudiant n'est pas trouvé, un message indiquant que l'étudiant n'existe pas est affiché.

AfficherEtudiant Cette fonction affiche les informations d'un étudiant, telles que le nom, le prénom, le CNE et la moyenne annuelle.

AfficherEtudiantsOrdreAlphabetique Cette fonction trie les étudiants par ordre alphabétique et les affiche en utilisant le tri à bulles et la fonction **afficherEtudiant**.

AfficherEtudiantsParOrdreDeMerite Cette fonction trie les étudiants par ordre de mérite et les affiche en utilisant le tri à bulles et la fonction **afficherEtudiant.**

La fonction **afficherModule** prend en entrée une structure de type **"Module"** et affiche les informations suivantes sur ce module : le nom, le nombre de validations, le nombre de non-validations, la note maximale et la note minimale.

La fonction **afficherTousModules** prend en entrée un tableau de structure **"Module"** et un entier "annee". Elle affiche les informations sur les 16 modules correspondants à l'année scolaire spécifiée. Les informations affichées sont les mêmes que pour la fonction **"afficherModule"**

III-Dossier de programmation :

```
// Fonction pour initialiser un objet Etudiant
Etudiant *initialiserEtudiant()
{
        // Allouer de la mémoire pour un nouvel objet Etudiant
        Etudiant *e = (Etudiant *) malloc(sizeof(Etudiant));
        // Définir les valeurs initiales pour les membres de l'objet Etudiant
        e->Reserve = 0;
        e->Annee = 0;
        e->MoyAnnee = 0;
         // Renvoyer un pointeur vers l'objet Etudiant
        Return ((Etudiant *) e) ;
}
// Fonction pour initialiser un tableau de modules
void initialiserModules(Module Md[40])
{
        // Boucle pour initialiser les 40 modules
        for (int i = 0; i < 40; i++)
        {
                // Définir les valeurs initiales pour les membres de l'objet Module
                Md[i].NoteMax = 0; ;//Initialiser la note maximal à 0
                Md[i].NoteMin = 20;//Initialiser la note minimal à 20 pour avoir la note minimal
                Md[i].Nvalide = 0; //Initialiser la non validation à 0
                Md[i].Valide = 0; //Initialiser la validation à 0
        }
}
```

```
// Fonction pour ajouter un étudiant à la structure Ilisi
Ilisi *insererEtudiant(Ilisi *I, Etudiant e,int annee)
{
        // Décrémenter l'année pour la rendre compatible avec le tableau Etud
        annee=annee-1;
        // Vérifier si le nombre d'étudiants est inférieur à la limite maximale
        if (I->NbEtud < MAX)
        {
                // Ajouter l'étudiant à la liste des étudiants pour cette année
                I->Etud[annee][I->NbEtud] = e;
                // Incrémenter le nombre d'étudiants
                I->NbEtud++;
                // Stoker les informations de l'étudiant dans un fichier texte
                storeData(e,1);
        }
        else
        {
                // Afficher une erreur si le nombre maximal d'étudiants est atteint
                printf("Erreur n");
        }
        // Renvoyer la structure Ilisi
        return ((Ilisi *)I);
}

// Fonction pour afficher le nom et les informations de chaque étudiant
void afficherNomsEtudiants(Ilisi *i,Module Md[40],int annee)
{
        // Décrémenter l'année pour la rendre compatible avec le tableau Etud
        annee=annee-1;
       // Afficher le titre "Liste des étudiants"
        printf("Liste des étudiants:\n");
        // Boucle pour parcourir la liste des étudiants
        for (int j = 0; j < i->NbEtud; j++)
        {
           // Afficher les informations de l'étudiant
          printf("\n%s\t%s\t%s",i->Etud[annee][j].Nom,i->Etud[annee][j].Prenom,i>Etud[annee][j].CNE);
           for(int ind=0; ind<16; ind++)
              printf("\n %s : %f",&Md[ind],i->Etud[annee][j].notes[ind]);
           printf("\nMoyenne : %f",i->Etud[annee][j].MoyAnnee);
        }
}
```

```
// Fonction pour vérifier une note et la comparer avec les autres
Module VerifierNote(float Note,Module Md)
{
        // Incrémenter le compteur de notes validées si la note est supérieure ou égale à 12
        if(Note>=12)Md.Valide++;
         // Incrémenter le compteur de notes non-validées si la note est inférieure à 12
         if(Note<12)Md.Nvalide++;
        // Si la note est supérieure à la note maximum enregistrée, la remplacer
        if(Note>Md.NoteMax)
            Md.NoteMax=Note;
       // Si la note est inférieure à la note minimale enregistrée, la remplacer
          if(Note<Md.NoteMin)
              Md.NoteMin=Note;
      // Renvoyer le module mis à jour
      return ((Module )Md);
}
Etudiant *insererInfoEtudiant(Etudiant e, Module Md[40])
{
      / Demander et enregistrer le prénom de l'étudiant */
      printf("Entrez le prénom de l'étudiant : ");
      scanf("%s", e->Prenom);
      /* Demander et enregistrer le nom de l'étudiant */
      printf("Entrez le nom de l'étudiant : ");
      scanf("%s", e->Nom);
      /* Demander et enregistrer le CNE de l'étudiant */
      printf("Entrez le CNE de l'étudiant : ");
      scanf("%s", e->CNE);
      /* Demander et enregistrer le CNI de l'étudiant */
      printf("Entrez le CNI de l'étudiant : ");
      scanf("%s", e->CNI);
      /* Demander et enregistrer l'année de l'étudiant */
      printf("Entrez l'année de l'étudiant : ");
      scanf("%d", &e->Annee);
      /* Demander et enregistrer le statut de réserve de l'étudiant */
      printf("Entrez le statut de réserve de l'étudiant (0 pour non, 1 pour oui) : ");
      scanf("%d", &e->Reserve);
      /* Demander et enregistrer les notes de l'étudiant pour les deux semestres */
      printf("Entrez les notes de l'étudiant pour les deux semestres (séparées par un espace) : ");
      for (int i = 0; i < 16; i++) {
        float note;
        printf("%s \n", Md[i]);
        scanf("%f", &note);
        Md[i] = VerifierNote(note, Md[i]);
      }
      scanf("%f %f", &e->notes[0], &e->notes[1]);
              return ((Etudiant *)e); /* Retourner l'étudiant */
}
```

void chercher(Ilisi I, char CNE[20], int annee)

```
{
        int x = 0;
        // Décrémentation de l'année pour l'utiliser dans le tableau
        annee = annee - 1;
        // Boucle pour parcourir tous les étudiants
        for (int i = 0; i < I.NbEtud; i++)
        {
                // Comparaison du CNE de l'étudiant actuel avec le CNE recherché
                if (strcmp(CNE, I.Etud[annee][i].CNE) == 0)
                {
                   x = 1;
                   afficherEtudiant(I.Etud[annee][i]);
                }
        }
        // Si aucun étudiant n'a été trouvé
        if (x == 0)
           printf("l'étudiant n'existe pas\n");
}
s
```

void afficherEtudiantsOrdreAlphabetique(Ilisi I, int anne)

```
{
   int i, j;
   Etudiant temp;
   // Tri des étudiants par ordre alphabétique en utilisant le tri à bulles
   for (i = 0; i < I.NbEtud - 1; i++)
   {
      for (j = 0; j < I.NbEtud - i - 1; j++)
      {
         // Comparaison des noms des étudiants actuel et suivant
         if (strcmp(I.Etud[anne][j].Nom, I.Etud[anne][j + 1].Nom) > 0)
         {
            // Échange des positions des étudiants
            temp = I.Etud[0][j];
            I.Etud[0][j] = I.Etud[0][j + 1];
            I.Etud[0][j + 1] = temp;
         }
      }
   }

   // Affichage des étudiants triés par ordre alphabétique
   for (i = 0; i < I.NbEtud; i++)
   {
      afficherEtudiant(I.Etud[anne][i]);
   }
}
```

```
void afficherEtudiant(Etudiant e)
{
   // Affichage des informations de l'étudiant
   printf("\n%s\t%s\t%s\t%f", e.CNE, e.Nom, e.Prenom, e.MoyAnnee);
}

Fonction pour afficher les étudiants triés par ordre de mérite
Prend en entrée une structure Ilisi et l'année pour laquelle afficher les étudiants
*/
void afficherEtudiantsParOrdreDeMerite(Ilisi I, int annee)
{
        int i, j;
        Etudiant temp;

        // Tri des étudiants par ordre de mérite en utilisant le tri à bulles
        for (i = 0; i < I.NbEtud ; i++)
        {
                for (j = 0; j < I.NbEtud - i - 1; j++)
                {
                if (I.Etud[annee][j].MoyAnnee < I.Etud[annee][j+1].MoyAnnee)
                {
                        temp = I.Etud[annee][j];
                        I.Etud[annee][j] = I.Etud[annee][j+1];
                        I.Etud[annee][j+1] = temp;
                }
                }
        }
        // Affichage des étudiants triés
        for (i = 0; i < I.NbEtud; i++)
        {
                afficherEtudiant(I.Etud[annee][i]);
        }
}
```

```
/*Fonction pour trouver le module le plus validé dans une année donnée Prend en entrée un tableau de modules et l'année Retourne le module le plus validé
*/
Module PlusValide(Module Md[40],int annee)
{
        int i;
        annee=annee-1;

        Module maxi=Md[0];

        for(i = 1+16*annee; i < 16+16*annee; i++)
        {
        if(Md[i].Valide>maxi.Valide)maxi=Md[i];
        }

        return ((Module)maxi);
}

/*Fonction pour trouver le module le moins validé dans une année donnée Prend en entrée un tableau de modules et l'année Retourne le module le moins validé
*/
Module MoinValide(Module Md[40],int annee)
{
        int i;
        annee=annee-1;

        Module mini=Md[0];

        for(i = 1+16annee; i < 16+16annee; i++)
        {
        if(Md[i].Valide<mini.Valide)mini=Md[i];
        }

        return ((Module)mini);
}
```

```
// Fonction qui retourne le module avec la note la plus élevée pour une année donnée
Module NotePlusEleve(Module Md[40], int annee)
{
        // Déclaration et initialisation de la variable de boucle 'i'
        int i;
        // Calcul de l'indice de début de la plage des modules de l'année donnée
        annee = annee - 1;
        // Déclaration et initialisation du module maxi à partir du premier module de l'année donnée
        Module maxi = Md[0 + 16 * annee];

        // Boucle pour parcourir les modules de l'année donnée
        for(i = 1 + 16 * annee; i < 16 + 16 * annee; i++)
        {
           // Si la note max de ce module est supérieure à celle de maxi, maxi est mis à jour
           if(Md[i].NoteMax > maxi.NoteMax)
              maxi = Md[i];
        }

        // Retourne le module avec la note la plus élevée
        return ((Module)maxi);
}

// Fonction qui retourne le module avec la note la moins élevée pour une année donnée
Module NoteMoinsEleve(Module Md[40], int annee)
{
        // Déclaration et initialisation de la variable de boucle 'i'
        int i;
        // Calcul de l'indice de début de la plage des modules de l'année donnée
        annee = annee - 1;
        // Déclaration et initialisation du module mini à partir du premier module de l'année donnée
        Module mini = Md[0 + 16 * annee];

        // Boucle pour parcourir les modules de l'année donnée
        for(i = 1 + 16 * annee; i < 16 + 16 * annee; i++)
        {
           // Si la note min de ce module est inférieure à celle de mini, mini est mis à jour
           if(Md[i].NoteMin < mini.NoteMin)
              mini = Md[i];
        }

        // Retourne le module avec la note la moins élevée
        return ((Module)mini);
}
```

```
// Fonction pour afficher les informations d'un module
void afficherModule(Module Md)
{
        printf("\n%s\t%d\t%d\t%f\t%f\t",Md.Nom,Md.Valide,Md.Nvalide,Md.NoteMax,Md.NoteMin);

}

// Fonction pour afficher les informations de tous les modules d'une année donnée
void afficherTousModules(Module Md[40], int annee)
{
        annee = annee - 1; // décrémentation pour accéder à l'index correct dans le tableau de modules
        for (int i = 0 + annee * 16; i < 16 + annee * 16; i++)
        {
        printf("\n%s\t%d\t%d\t%f\t%f\t", Md[i].Nom, Md[i].Valide, Md[i].Nvalide, Md[i].NoteMax,
        Md[i].NoteMin);
        }
}


void stockerDesDonnees(Etudiant *etud, int nbEtud)
{
            FILE *fp;
            fp = fopen("data.txt", "w");

            // Si le fichier ne peut pas être ouvert, affichez une erreur et quittez le programme
            if (fp == NULL)
            {
               printf("Erreur lors de l'ouverture du fichier!\n");
               exit(1);
            }

            // Boucle pour écrire les données de chaque étudiant dans le fichier
            for (int i = 0; i < nbEtud; i++)
            {
               fprintf(fp,"%s\t%s\t%s\t%s\t%s\t%s\t",etud[i].Nom, etud[i].Prenom, etud[i].CNE,
        etud[i].CNI,etud[i].MoyAnnee, etud[i].Reserve);
               for(i=0; i<=15; i++)
               {
                  etud[i].notes[i];
               }
               fprintf(fp, "\n");
            }

            // Fermeture du fichier
            fclose(fp);
}
```

```
/* Cette fonction lit les données des étudiants à partir d'un fichier */
void readFile(Ilisi *ilisi,Module Md[40],int annee)
{
        // Ouvrir un fichier en mode lecture
        FILE *fp;
        int i, j;
        fp = fopen("data.txt", "r");
        // Vérifier si le fichier est ouvert correctement
        if (fp == NULL)
        {
           printf("Error opening file!\n");
           exit(1);
        }
        // Réduire l'année de 1
        annee=annee-1;
        // Lire le nombre d'étudiants dans le fichier
        fscanf(fp, "%d", &ilisi->NbEtud);
        // Lire les données de chaque étudiant
        for (i = 0; i < ilisi->NbEtud; i++)
        {
           // Lire les informations de l'étudiant
           fscanf(fp, "%s %s %s %s ",
                ilisi->Etud[annee][i].Nom,
                ilisi->Etud[annee][i].Prenom,
                ilisi->Etud[annee][i].CNE,
                ilisi->Etud[annee][i].CNI);

           // Lire les notes de l'étudiant pour les 16 modules
           for(int j=0; j<16; j++)
           {
              fscanf(fp,"%f",&ilisi->Etud[annee][i].notes[j]);
              // Vérifier si la note est valide
              Md[j]=VerifierNote(ilisi->Etud[annee][i].notes[j],Md[j]);
           }
           // Calculer la moyenne de l'étudiant et vérifier sa validité
           MoyenneValid(&ilisi->Etud[annee][i]);

        }
        // Fermer le fichier
        fclose(fp);
}
```

```
/* Cette fonction lit les noms des modules à partir d'un fichier */
Module LireModule(Module Md[40])
{
        // Ouvrir un fichier en mode lecture
        FILE *fp;
        fp = fopen("Module.txt", "r");

        // Vérifier si le fichier est ouvert correctement
        if (fp == NULL)
        {
           printf("Error opening file!\n");
           exit(1);
        }

        // Lire les noms des modules dans un tableau
        int i=0;
        while (fgets(Md[i++].Nom, 140, fp) != NULL);

        // Retourner le tableau de modules
        return ((Module)Md[40]);


        // Fermer le fichier
        fclose(fp);
}
```

/* Cette fonction calcule la moyenne d'un étudiant et vérifie sa validité */

*void MoyenneValid(Etudiant *e)*

```
{
        // Initialise une variable pour stocker la somme des notes
        float somme=0;

        // Boucle pour additionner toutes les notes
        for(int i=0; i<16; i++)
        {
           somme += e->notes[i];
        }

        // Calcule la moyenne en divisant la somme des notes par 16
        somme = somme / 16;
        // Stocke la moyenne annuelle de l'étudiant
        e->MoyAnnee = somme;

        // Vérifie si l'étudiant est admis ou réservé
        if(Valid(e))
        {
           e->Reserve = 0;
        }
        else
        {
           e->Reserve = 1;
        }
}
```