IUT Paris Descartes

2020-2021

Initiation à l'Algorithmique et à la Programmation

Mémoire de projet

Enes Inac
MAXIME WANG

Groupe 108

Tables des matières

Enes Inac
MAXIME WANG

Groupe 108

I / Présentation de l'application

Introduction :

Les entreprises de nos jours doivent traiter et trier des milliers voire des millions d'information à la fois. Prenons l'exemple de Amazon, ils doivent s'occuper à la fois des clients qui passent des commandes sur Internet ainsi que de vérifier la disponibilité des stocks chez le vendeur qu'ils ont commandé et tout ça constitue la base de donnée. Ainsi la solution adéquate face à cette montagne d'information est la bonne gestion de sa base de donnée.

Problématique :

Notre but est de programmer pour notre entreprise un interpréteur de commandes pour gérer une base de donnée.

Rôle fonctionnel du projet :

Fondre une entreprise qui au fur et à mesure du temps des nouveaux clients vont s'ajouter ainsi que de nouvelles commandes par les clients, des employés s'occupant de leurs commandes, des nouveaux employés etc...

L'objectif de ce projet est de programmer un interpréteur de commande qui permet de créer ces données. Il doit ensuite être capable d'afficher ces données. Il faut aussi pouvoir quitter le programme grâce à une commande.

Les entrées et sorties :

Ce projet projet contient treize entrées, qui correspondent aux treize commandes à coder : enregistreSpecialites, afficheSpecialites, enregistreEmbauche, afficheTravailleurs, enregistreClients, afficheClient, enregistreNewCommande, enregistreTache, enregistreProgressionTravail, avancementCommande, afficheTacheRestante, Quitter.

La reconnaissance de ces treize commandes s'opère dans le main, par l'utilisation de la fonction strcmp() issue de la bibliothèque , qui permet de comparer deux chaînes de caractère. Elle retourne 0 si les chaînes sont identiques et autre chose si elles sont différentes.

Son prototype est le suivant : int strcmp(const char* str1, const char* str2); On utilise donc son retour associé à une condition if) pour reconnaître les commandes, pour ensuite appeler la fonction qui correspond à chaque commande.

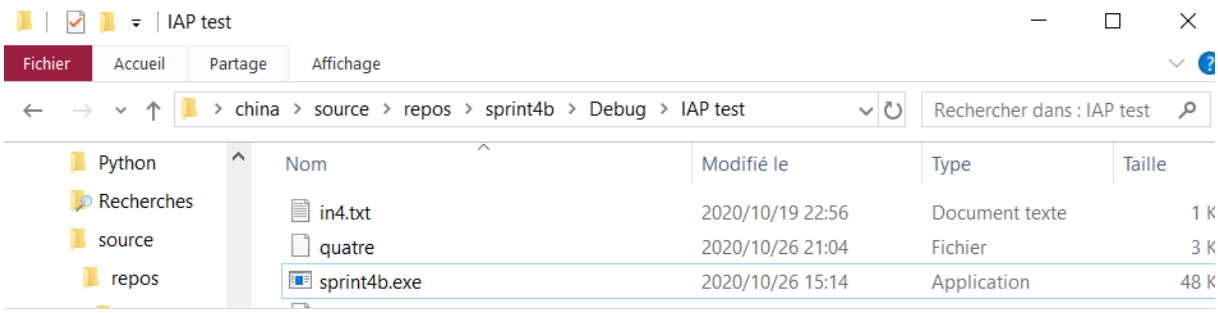
Pour ce qui est des sorties, seules les commandes d'affichage sont associées à des sorties. En effet, ces fonctions contiennent des commandes d'affichage printf . Comme les fonctions afficheSpecialites, afficheTravailleurs, afficheClient, afficheTacheRestante.

II/ Organisation des tests

Le projet a été mené sous forme de sprints définis par le génie logiciel. Un sprint est défini par une spécification, un jeu de données de test (inSp#) ainsi que des résultats attendus (OutSp#).

Un test permet de valider un objectif. Il utilise donc le jeu de données de test (JDT) et le résultat précis. Nous avons donc utilisé la redirection de flux d'entrée pour tester rapidement et efficacement nos programmes afin de valider nos sprints.

Nous avons donc créé un dossier contenant les fichiers requis :



La barre écrit « china>source>repos>sprint4b>Debug>IAP test » effacé la et écrivez « cmd ».

Utilisation de l'invite de commande pour la redirection :

```
Microsoft Windows [version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\china\source\repos\sprint4b\Debug\IAP test>sprint4b.exe <in4.txt> quatre
```

On crée ainsi un fichier quatre.txt qui contient les sorties de notre programme.

Il s'agit ensuite de comparer ce ficher de sorties quatre.txt avec le ficher des sorties attendues outsprint#.txt. Pour cela on utilise un outil diffchecker.com :

Ce programme nous montre ainsi les différences entre le fichier de sortie produit par notre programme et le fichier de sorties attendues. Si des différences apparaissent, il convient de revoir le programme, sinon, il est déclaré les deux fichiers sont identiques , et on passe à la suite.

III/ Bilan de validation des tests

Pour le bilan de la validation de tests, notre projet valide les sprints 0 à 4 Release.

Pour le sprint zéro, il faut juste afficher ce qu'on a entré dans la console. Un simple exercice pour bien commencer le projet rien de méchant. Elle affiche également « !!! instruction inconnue » si l'entrée n'est pas reconnu par le programme car il vérifie à chaque fois les mots-clés qui lui sont données.

Pour le premier sprint, il faut coder les fonctions correspondant aux commandes enregistreSpecialites, afficheSpecialites, etc.., Quitter. Pour ce premier sprint, il faut juste faire des affichages de ce qu'on a entré dans la console. Par exemple pour enregistreSpecialites, il faut juste afficher le nom de la spécialité et leur coût horaire. La même chose pour toutes les fonctions qui seront à coder. Elle affiche également « !!! instruction inconnue » si l'entrée n'est pas reconnu par le programme car il vérifie à chaque fois les mots-clés qui lui sont données. De plus la comparaison des fichiers obtenus par redirection nous a montré des espaces en trop à certains endroits, ce que nous avons facilement corrigé, nous sommes ensuite passés au sprint de niveau supérieur.

Pour le deuxième sprint, il faut s'aider des types structurés Specialite, Specialites, Travailleur, Travailleurs et Clients pour coder les fonctions correspondant aux commandes enregistreSpecialites, afficheSpecialites, enregistreEmbauche, afficheTravailleurs, enregistreClients, afficheClient. Pour enregistreSpecialites il faut mémoriser le nom de la spécialité et le coût horaire dans le tableau des spécialité. Pour afficheSpecialites, il faut afficher le nom de la spécialité et le coût horaire qui se trouve dans le tableau des spécialités. Pour enregistreEmbauche, il faut mémoriser le nom et la spécialité du travailleur dans le tableau des travailleurs. Pour afficheTravailleurs, il faut afficher le nom et la spécialité du travailleur qui se trouve dans le tableau des travailleurs. Pour enregistreClients il faut mémoriser le nom du client dans le tableau des clients. Pour afficheClient, il faut afficher le nom du client qui se trouve dans le tableau des clients. Elle affiche également « !!! instruction inconnue » si l'entrée n'est pas reconnu par le programme car il vérifie à chaque fois les mots-clés qui lui sont données. La comparaison des fichiers obtenus par redirection et par diffcheker nous a permis de déclarer ce sprint comme 0-défauts et de passer au sprint de niveau supérieur.

Pour le troisième sprint, il faut s'aider des types structurés Commande,

Commandes, Tache pour coder les fonctions correspondant aux commandes

enregistreNewCommande, enregistreTache, enregistreProgressionTravail, afficheClient, avancementCommande. Pour enregistreNewCommande, il faut mémoriser la liste de la commande du client concernée dans le tableau des commandes. Pour enregistreTache, il faut mémoriser la tâche d'une commande qui précise la spécialité et le nombre d'heure nécessaire. Pour enregistreProgressionTravail, il faut mémoriser la progression d'une commande (actualise l'état de la tâche). Pour afficheClient, il faut afficher la liste des produits que le client a commandé. Pour avancementCommande, il faut afficher l'avancement d'une commande. Elle affiche également « !!! instruction inconnue » si l'entrée n'est pas reconnu par le programme car il vérifie à chaque fois les mots-clés qui lui sont données. La comparaison des fichiers obtenus par redirection et par diffcheker nous a permis de déclarer ce sprint comme 0-défauts et de passer au sprint de niveau supérieur.

Pour le quatrième sprint, il faut coder les commandes enregistreTache, afficheTacheRestante. Pour enregistreTache, il faut affecter la tâche est affectée au premier travailleur compétent embauché ayant le moins d'heures à faire pour compléter toutes ses tâches. Pour afficheTacheRestante, il faut afficher pour le travailleur choisi le nombre d'heures restantes pour l'accomplissement de chacune des tâches qui lui sont affectées. Elle affiche également « !!! instruction inconnue » si l'entrée n'est pas reconnu par le programme car il vérifie à chaque fois les mots-clés qui lui sont données. La comparaison des fichiers obtenus par redirection et par diffcheker nous a permis de déclarer ce sprint comme 0-défauts et de passer au sprint de niveau supérieur.

IV/ Bilan du projet

Au cours de ce projet nous avons rencontré un certain nombre de problèmes, mais nous avons pu y apporter les solutions adéquates. Tout d'abord, nous nous sommes heurtés à un problème de compréhension au tout début du projet. Nous avons pensé à faire les sprints sans les aides qui nous ai fourni par le sujet mais en discutant avec un de nos camarades de classe nous avons compris que le sujet nous obligé d'utiliser ces données car ça aller nous faciliter par la suite. Ainsi nous avons recommencé tout à partir du sprint 1 Release. Au-delà de cela l'une de nos difficultés a été l'utilisation des propriétés des types structurés avec des objets de type pointeurs. En effet, nous étions repris par Visual Studio qui produisait des erreurs que nous ne comprenions pas. Cependant, en assistant aux cours et TD d'IAP sur ce sujet, notre incompréhension a disparu, nous permettant de manipuler nos données avec bien plus de clarté.

Sinon pour le sprint 4 Release, il nous a fallu un peu de temps à inventer et à tester les nouvelles combinaisons de IN(entrée) possible car il fallait que la tâche soit affectée au premier travailleur compétent embauché ayant le moins d'heures à faire pour compléter toutes ses tâches.

Enfin nous pensons qu'écrire son pseudo-code sur un tableau dans le projet nous a énormément aidé comprendre et à trouver une solution plus optimale.

Conclusion :

Ce projet collaboratif en programmation nous a permis de découvrir le langage C, ses possibilités, mais aussi ses aspects difficiles à appréhender.
Ce projet, nous a également introduit au travail de codage et de développement en groupe, il nous a appris à suivre les directive de génie logiciel, à l'appliquer correctement et à tirer le maximum de notre environnement de développement pour maximiser l'optimisation de notre travail.
Cela nous a aidé non seulement à maîtriser les outils de la programmation mais aussi à intégrer la rigueur nécessaire à la création de bons programmes.
Enfin, l'aspect collaboratif de ce projet fait partie de ses points forts.
En effet, la motivation mutuelle est un moteur si puissant qu'il permet de surmonter tout type de désespoir.
Chacun comblant les faiblesses de l'autre, nous avons pu nous tirer mutuellement vers le haut.
Le travail collaboratif est enrichissant car il permet d'apprendre autant de l'autre que de soi.

Annexe

```c
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

/*                                                  main.c
                                                    PROJETS IAP Sprint 4 Release
                                                    Auteurs : Enes Inac ET MAXIME WANG
                                                    Date de création : 21/10/2020

*/


                                                    // Toutes les définitions constantes
typedef enum
{
    FAUX = 0, VRAI = 1
}Booleen;
Booleen EchoActif = FAUX;

// Messages emis par les instructions
#define MSG_INTERRUPTION "## fin de programme\n"

// Lexemes
#define LGMOT 35
#define NBCHIFFREMAX 5

// Nombres maximum de ...
#define MAX_SPECIALITES 10
#define MAX_TRAVAILLEURS 50
#define MAX_CLIENTS 100
#define MAX_COMMANDES 500

typedef char Mot[LGMOT + 1];

// Données

// Enregistre le nom et cout horaire d'une specialite
typedef struct {
    Mot nom;
    unsigned int cout_horaire;
} Specialite;

// Stocke toutes les spécialités proposés et le nombre de spécialité proposés en
entier
typedef struct {
    Specialite tab_specialites[MAX_SPECIALITES];
    unsigned int nb_specialites;
} Specialites;

// Enregistre les compétences d'un travailleur
typedef struct Travailleur {
    Mot nom;
```

```
    Booleen tags_competences[MAX_SPECIALITES];
    unsigned int charge;
} Travailleur;

// Stocke tous les travailleurs et le nombre de travailleur en entier
typedef struct {
    Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
    unsigned int nb_travailleurs;
} Travailleurs;

// Enregistre un tableau de clients avec leurs noms
typedef struct {
    Mot tab_clients[MAX_CLIENTS];
    unsigned int nb_clients;
} Clients;

// Initialisation des variables pour Taches des commandes
typedef struct {
    int nb_heures_effectuees;
    unsigned int nb_heures_requises;
    unsigned int idx_travailleur;
} Tache;

// Enregistre un tableau de spécialité, le nom et l'index du client
typedef struct {
    Mot nom;
    unsigned int idx_client;
    Tache taches_par_specialite[MAX_SPECIALITES]; // nb_heures_requises == 0 <=> pas
de tache pour cette specialite
} Commande;

// Enregistre une liste de commandes et le nombre de commandes
typedef struct {
    Commande tab_commandes[MAX_COMMANDES];
    unsigned int nb_commandes;
} Commandes;

// Fonction de type void ne retourne aucune valeur mais excute les instructions -->
// Nom de la fonction get_id / Paramètre de la fonction Mot id,
// 'Mot' defini ci-dessus et id est un string de capacité de mémoire maximum de 35
caractères
void get_id(Mot id)
{
    scanf("%s", id);
    if (EchoActif) printf(">>echo %s\n", id);
}

// fonction de type int
int get_int()
{
    char buffer[NBCHIFFREMAX + 1];
    scanf("%s", buffer);
    if (EchoActif) printf(">>echo %s\n", buffer);
    return atoi(buffer);
}

// Prototypes

// Lorsque l'on écrit "developpe" dans la console
/** Brief : Enregistre le nom de la spécialité et le coût horaire
            dans le tableau de spécialités proposés par l'entreprise.

    - pt_liste_specialites [in] [out] le nom et le nombre de spécialité
```

```
    ansi que la liste des specialites initialisés
    [pre] nom de la spécialité != nom de spécialité dans la liste des spécialités
    [pre] nombres de spécialité <= MAX_SPECIALITES
*/
void enregistreSpecialites(Specialites* pt_liste_specialites);

// Lorsque l'on écrit "specialites" dans la console
/** Brief : Afficher le prix de facturation d'une heure de travail
            pour toutes les spécialités proposés par l'entreprise

    - pt_specialites [in] le nom/cout horaire et le nombre de spécialités
*/
void afficheSpecialites(Specialites* pt_specialites);

// Lorsque l'on écrit "embauche" dans la console
/** Brief : Enregistrer les noms et les compétences des travailleurs dans la liste des
travailleurs

    - pt_list_travailleurs [in] [out] la liste des travailleurs initialisés
    - pt_specialites [in] le nom et le nombre de spécialités
    [pre] nombres de travailleurs <= MAX_TRAVAILLEURS
*/
void enregistreEmbauche(Travailleurs* pt_list_travailleurs, Specialites*
pt_specialites);

// Lorsque l'on écrit "travailleurs" dans la console
/** Brief : Affiche les noms des travailleurs compétents pour la spécialité d'entrée

    - pt_travailleurs [in] le nom et le nombre de travailleur ainsi que la compétence
    - pt_specialites [in] le nom et le nombre de spécialités
*/
void afficheTravailleurs(Travailleurs* pt_travailleurs, Specialites* pt_specialites);

// Lorsque l'on écrit "demarche" dans la console
/** Brief : Enregistrer le client dans la liste de client "tab_clients"

    - pt_clients [in] [out] le nom et le nombre de clients
    [pre] nombres de clients <= MAX_CLIENTS
*/
void enregistreClients(Clients* pt_clients);

// Lorsque l'on écrit "client" dans la console
/** Brief : Afficher la liste des produits que le client a commandé

    - pt_clients [in] le nom et le nombre de clients
    - pt_commande [in] le nom de la commande, l'index client,
    le nombre de commandes ainsi que le tableau des commandes
*/
void afficheClient(Clients* pt_clients, Commandes* pt_commande);

// Lorsque l'on écrit "commande" dans la console
/** Brief : Enregistrer la liste de la commande du client

    - pt_liste_commandes [out] le nom de la commande, l'index client,
    ainsi que le nombre de commandes
    - pt_client [in] le nom et le nombre de clients
    [pre] le nombres de commandes <= MAX_COMMANDES
*/
void enregistreNewCommande(Commandes* pt_liste_commandes, Clients* pt_client);

// Lorsque l'on écrit "progression" dans la console
/** Brief : Enregistre la progression d'une commande (actualise l'état de la tâche)
```

```
    - pt_commandes [in] le nom et le nombre de commandes ainsi que le tableau des
commandes
    - pt_specialites [in] le nom et le nombre de spécialités
    - pt_travailleur [in] le tableau des travailleurs
*/
void enregistreProgressionTravail(Commandes* pt_commandes, Specialites*
pt_specialites, Travailleurs* pt_travailleur);

// Lorsque l'on écrit "supervision" dans la console
/** Brief : Affiche l'avancement d'une commande

    - pt_commande [in] le nom et le nombre de commandes ainsi que le tableau des
commandes
    - pt_specialite [in] le nom et le nombre de spécialités
*/
void avancementCommande(Commandes* pt_commande, Specialites* pt_specialite);

// Lorsque l'on écrit "tache" dans la console
/** Brief : Enregistrer la tâche d'une commande qui précise la spécialité et le nb
heure nécessaire

    - pt_commande [in] [out] le nom et le nombre de commandes ainsi que la tâche de
cette commande initialisée
    - pt_tab_specialites [in] le nom et le nombre de spécialités
    - pt_travailleur [in] [out] la compétence et le nombre de travailleurs ainsi que
la charge initialisée
*/
void enregistreTache(Commandes* pt_commande, Specialites* pt_tab_specialites,
Travailleurs* pt_travailleur);

// Lorsque l'on écrit "charge" dans la console
/** Brief : Pour le travailleur choisi, affiche le nombre d'heures restantes pour
            l'accomplissement de chacune des tâches qui lui sont affectées.

    - pt_travail [in] le nom et le nombre de travailleurs
    - pt_spe [in] le nom et le nombre de spécialités
    - pt_commande [in] le nom et le nombre de commandes, le nombre d'heures
effectuées/requises
    ainsi que le l'index du travailleur s'occupant de cette commande
*/
void afficheTacheRestante(Travailleurs* pt_travail, Specialites* pt_spe, Commandes*
pt_commande);

// Lorsque l'on écrit "interruption" dans la console
/* Brief : Affiche un message d'interruption qui fait quitter du programme */
void Quitter();



// Boucle principale
int main(int argc, char* argv[])
{

    // Initialisation des variables et stocke des valeurs
    Specialites Couts_Specialite = { " " , 0 };

    Travailleurs Embauche_travailleurs = { " " , 0 };

    Clients Mes_Clients = { " " , 0 };

    Commandes Les_Commandes = { " " , 0 };
```

```
    if (argc >= 2 && strcmp("echo", argv[1]) == 0)
    {
        EchoActif = VRAI;
    }

    // Buffer de type Mot et ne peut contenir au max 35 caratères
    Mot buffer;

    // Une boucle à l'infini
    while (VRAI)
    {
        // Attente d'une entrée dans la console ------------
        get_id(buffer);

        // Appel de la fonction enregistreSpecialites lorsque l'on écrit
'developpe ...' dans la console
        if (strcmp(buffer, "developpe") == 0)
        {
            enregistreSpecialites(&Couts_Specialite);
            continue;
        }

        // Appel de la fonction afficheSpecialites lorsque l'on écrit 'specialite ...'
dans la console
        if (strcmp(buffer, "specialites") == 0)
        {
            afficheSpecialites(&Couts_Specialite);
            continue;
        }

        // Appel de la fonction enregistreEmbauche lorsque l'on écrit 'embauche ...'
dans la console
        if (strcmp(buffer, "embauche") == 0)
        {
            enregistreEmbauche(&Embauche_travailleurs, &Couts_Specialite);
            continue;
        }

        // Appel de la fonction afficheTravailleurs lorsque l'on écrit
'travailleur ...' dans la console
        if (strcmp(buffer, "travailleurs") == 0)
        {
            afficheTravailleurs(&Embauche_travailleurs, &Couts_Specialite);
            continue;
        }

        // Appel de la fonction enregistreClients lorsque l'on écrit 'demarche ...'
dans la console
        if (strcmp(buffer, "demarche") == 0)
        {
            enregistreClients(&Mes_Clients);
            continue;
        }

        // Appel de la fonction afficheClient lorsque l'on écrit 'client ...' dans la
console
        if (strcmp(buffer, "client") == 0)
        {
            afficheClient(&Mes_Clients, &Les_Commandes);
            continue;
        }
```

```
        // Appel de la fonction enregistreNewCommande lorsque l'on écrit
'commande ...' dans la console
        if (strcmp(buffer, "commande") == 0)
        {
            enregistreNewCommande(&Les_Commandes, &Mes_Clients);
            continue;
        }

        // Appel de la fonction enregistreTache lorsque l'on écrit 'tache ...' dans la
console
        if (strcmp(buffer, "tache") == 0)
        {
            enregistreTache(&Les_Commandes, &Couts_Specialite,
&Embauche_travailleurs);
            continue;
        }

        // Appel de la fonction enregistreProgressionTravail lorsque l'on écrit
'progression ...' dans la console
        if (strcmp(buffer, "progression") == 0)
        {
            enregistreProgressionTravail(&Les_Commandes,
&Couts_Specialite,&Embauche_travailleurs);
            continue;
        }

        // Appel de la fonction avancementCommande lorsque l'on écrit
'supervision ...' dans la console
        if (strcmp(buffer, "supervision") == 0)
        {
            avancementCommande(&Les_Commandes, &Couts_Specialite);
            continue;
        }

        // Appel de la fonction afficheTacheRestante lorsque l'on écrit 'charge ...'
dans la console
        if (strcmp(buffer, "charge") == 0)
        {
            afficheTacheRestante(&Embauche_travailleurs, &Couts_Specialite,
&Les_Commandes);
            continue;
        }

        // Lorsque l'on écrit 'passe' dans la console
        if (strcmp(buffer, "passe") == 0)
        {
            continue;
        }

        // Appel de la fonction Quitter lorsque l'on écrit 'interruption ...' dans la
console
        if (strcmp(buffer, "interruption") == 0)
        {
            Quitter();
            break;
        }

        // Un message d'erreur comme quoi notre entrée n'est pas reconnu
        printf("!!! instruction inconnue >%s< !!!\n", buffer);
    }
    return 0;
```

```
}



// Instructions après avoir écrit "..." dans la console
// Légende : pt = pointeur

// Lorsque l'on écrit "developpe" dans la console
// Brief : Enregistre le nom de la spécialité et le coût horaire
//         dans le tableau de spécialités proposés par l'entreprise
void enregistreSpecialites(Specialites* pt_liste_specialites)
{
    // Initialisation des variables et Demande d'entrée le nom et le coût horaire
d'une spécialité
    Specialite cout_specialite;
    Mot nom_specialite;
    get_id(nom_specialite);
    int cout_horaire = get_int();

    // Message d'erreur si le nombre de specialité est supérieur aux nombres de
    spécialités autorisés
    assert(pt_liste_specialites->nb_specialites <= MAX_SPECIALITES);

    // Copie le nom et le coût horaire de la spécialité dans "Specialite"
    strcpy(cout_specialite.nom, nom_specialite);
    cout_specialite.cout_horaire = cout_horaire;

    /* Parcours toutes les spécialités pour comparer si le nom de la spécialité
d'entrée est pareille
                               que celui dans la liste des spécialités
*/
    unsigned int x;
    for (x = 0; x < pt_liste_specialites->nb_specialites; x++)
    {
        // Compare le nom de la spécialité avec le nom de spécialité dans le tableau
des spécialités
        if (strcmp(nom_specialite, pt_liste_specialites->tab_specialites[x].nom) == 0)
        {
            // Message d'erreur
            assert(nom_specialite == pt_liste_specialites->tab_specialites[x].nom);

            // Arrête la fonction void enregistreSpecialites grâce au return
            return;
        }
    }
    // Enregistre le nom et le coût horaire d'une spécialité dans "Specialites" (les
tab des spécialités)
    pt_liste_specialites->tab_specialites[pt_liste_specialites->nb_specialites] =
cout_specialite;

    // Incrémente de UN le nombre de spécialité
    pt_liste_specialites->nb_specialites++;
}

// Lorsque l'on écrit "specialites" dans la console
/* Brief : Afficher le prix de facturation d’une heure de travail
           pour toutes les spécialités proposés par l'entreprise      */
void afficheSpecialites(Specialites* pt_specialites)
{
    // Ajoute les spécialités proposés ainsi que leurs coûts horaires à la suite du
MSG ci-dessous
    printf("specialites traitees :");
```

```
    unsigned int virgule = 0;
    unsigned int x;

    // Boucle qui parcours toutes les spécialités ET renvoie le nom ainsi que son prix
de facturation
    for (x = 0; x < pt_specialites->nb_specialites; x++)
    {
        // Ajoute la virgule si il a déjà afficher une fois le nom de la spécialité et
le cout horaire de la spé
        if (virgule != 0)
        {
            printf(",");
        }
        printf(" %s/%d", pt_specialites->tab_specialites[x].nom,
pt_specialites->tab_specialites[x].cout_horaire);
        virgule++;
    }
    printf("\n");
}

// Lorsque l'on écrit "embauche" dans la console
// Brief : Enregistrer les noms et les compétences des travailleurs dans la liste des
travailleurs
void enregistreEmbauche(Travailleurs* pt_list_travailleurs, Specialites*
pt_specialites)
{
    // Initialise et Demande d'entrée le nom et la ou les compétence(s) du travailleur
    Mot nom_travailleur;
    get_id(nom_travailleur);
    Mot competences;
    get_id(competences);

    // Message d'erreur si le nombre de travailleurs est supérieur aux nombres de
travailleurs autorisés
    assert(pt_list_travailleurs->nb_travailleurs <= MAX_TRAVAILLEURS);

    // Initialisation des variables x et y pour la boucle for ET
    // Un indicateur pour savoir qu'un travailleur existe déjà
    unsigned int x, y;
    unsigned int travailleur_existant = 0;

    // Parcours la boucle tant qu'une spécialité est proposé par l'entreprise
    for (x = 0; x < pt_specialites->nb_specialites; x++)
    {
        // Parcours la liste des travailleurs
        for (y = 0; y < pt_list_travailleurs->nb_travailleurs; y++)
        {
            /* Vérifie si le nom de la compétence d'entrée est pareille
               Que le nom de la spécialité dans la LISTE DES SPECIALITES proposés par
l'entreprise */
            if (strcmp(pt_specialites->tab_specialites[x].nom, competences) == 0)
            {

                // Vérifie si le nom du travailleur existe déjà dans la liste des
travailleurs
                if (strcmp(pt_list_travailleurs->tab_travailleurs[y].nom,
nom_travailleur) == 0)
                {
                    // Le travailleur existe déjà ainsi la compétence est vrai pour ce
travailleur
```

```
                    pt_list_travailleurs->tab_travailleurs[y].tags_competences[x] =
VRAI;

                    // Incrémente 'travailleur_existant' de UN, cela veut dire que ce
travailleur existe déjà
                    travailleur_existant++;
                }
            }
        }
        /* Condition qui compare la compétence d'entrée avec le nom de spécialité dans
la liste spécialité ET
              L'indicateur 'travailleur_existant = 0' nous montre que aucun
travailleur n'a été déclaré à ce nom       */

        if (strcmp(pt_specialites->tab_specialites[x].nom, competences) == 0 &&
travailleur_existant == 0)
        {
            // Initialise le nombre d'heure de charge d'un travailleur à 0

pt_list_travailleurs->tab_travailleurs[pt_list_travailleurs->nb_travailleurs].charge =
0;

            // Créer un nouveau travailleur en copiant le nom du travailleur dans le
tableau des travailleurs

strcpy(pt_list_travailleurs->tab_travailleurs[pt_list_travailleurs->nb_travailleurs].n
om, nom_travailleur);

            // Enregistre la compétence du travailleur en booleen càd VRAI/FAUX dans
le tableau des compétences de ce travailleur

pt_list_travailleurs->tab_travailleurs[pt_list_travailleurs->nb_travailleurs].tags_com
petences[x] = VRAI;

            // Incrémente de UN le nombre de travailleurs
            pt_list_travailleurs->nb_travailleurs++;
        }
    }
}

// Lorsque l'on écrit "travailleurs" dans la console
// Brief : Affiche les noms des travailleurs compétents pour la spécialité d'entrée
void afficheTravailleurs(Travailleurs* pt_travailleurs, Specialites* pt_specialites)
{

    // Initialisation de la variable ET
    // Demande d'entrée le nom d'une spécialité
    Mot nom_specialites;
    get_id(nom_specialites);

    // Initialisation des variables x et y pour la boucle for
    unsigned int x, y;

    // Si l'entrée était "travailleurs tous"
    if (strcmp(nom_specialites, "tous") == 0)
    {
        // Indicateur 'condition = FAUX' veut dire que aucun travailleur n'a été
ajouter dans la phrase ci-dessous
        Booleen notFirst = FAUX;

        // Parcours le nombre de spécialités proposés par l'entreprise
        for (x = 0; x < pt_specialites->nb_specialites; x++)
        {
```

```
            // Remplace le %s par le nom de la spécialité
            printf("la specialite %s peut etre prise en charge par :",
pt_specialites->tab_specialites[x].nom);

            // Parcours tous les travailleurs de l'entreprise
            for (y = 0; y < pt_travailleurs->nb_travailleurs; y++)
            {

                // Condition qui vérifie si le travailleur 'y' possède la compétence
                if (pt_travailleurs->tab_travailleurs[y].tags_competences[x] == 1)
                {

                    // Condition qui vérifie si ce n'est pas le 1er travailleur
                    if (notFirst == VRAI && pt_travailleurs->nb_travailleurs > 0)
                    {
                        // Pour les travailleurs après le 1er travailleur on ajoute
une virgule
                        printf(",");
                    }
                    // Affiche le nom des travailleurs
                    printf(" %s", pt_travailleurs->tab_travailleurs[y].nom);
                    notFirst = VRAI;
                }
            }
            // Retour à la ligne pour chaque consultation des travailleurs compétents
à la spécialité
            printf("\n");

            // La condition redevient FAUX avant de retourner dans la boucle qui
parcours les spécialités
            notFirst = FAUX;
        }
    }

    // Condition qui vérifie si au moins un travailleur existe
    else if (pt_travailleurs->nb_travailleurs > 0)
    {
        // Initialisation des variables x et y pour la boucle for
        unsigned int x, y;

        // Indicateur 'condition = FAUX' veut dire que aucun travailleur n'a été
ajouter dans la phrase ci-dessous
        Booleen notFirst = FAUX;

        // Parcour le nombre de spécialités
        for (x = 0; x < pt_specialites->nb_specialites; x++)
        {
            // Condition qui vérifie si le nom de la spécialité d'entrée est pareil
            // Que le nom de la spécialité dans le tableau des spécialités
            if (strcmp(nom_specialites, pt_specialites->tab_specialites[x].nom) == 0)
            {

                // Remplace le %s par le nom de la spécialité
                printf("la specialite %s peut etre prise en charge par :",
pt_specialites->tab_specialites[x].nom);

                // Parcours tous les travailleurs de l'entreprise
                for (y = 0; y < pt_travailleurs->nb_travailleurs; y++)
                {

                    // Condition qui vérifie si le travailleur est compétent(e) pour
cette spécialité
```

```
                    if (pt_travailleurs->tab_travailleurs[y].tags_competences[x] ==
VRAI)
                    {

                        // Condition qui vérifie si un travailleur a déjà été ajouter
à la phrase ci-dessus ou pas
                        if (notFirst == VRAI && pt_travailleurs->nb_travailleurs > 0)
                        {
                            // Pour les travailleurs après le 1er travailleur on
ajoute une virgule
                            printf(",");
                        }
                        // Affiche le nom des travailleurs
                        printf(" %s", pt_travailleurs->tab_travailleurs[y].nom);

                        // La condition devient VRAI càd un travailleur a déjà été
ajouter
                        notFirst = VRAI;
                    }

                }
                // Retour à la ligne pour chaque consultation des travailleurs
compétents à la spécialité demandé
                printf("\n");

                // La condition redevient FAUX avant de retourner dans la boucle qui
parcours les spécialités
                notFirst = FAUX;
            }
        }
    }
}

// Lorsque l'on écrit "demarche" dans la console
// Brief : Enregistrer le client dans la liste de client "tab_clients"
void enregistreClients(Clients* pt_clients)
{
    // Initialisation des variables et attend une entrée pour le nom du client
    Mot nom_client;
    get_id(nom_client);

    // Message d'erreur si le nombre de clients est supérieur aux nombres de clients
autorisés
    assert(pt_clients->nb_clients <= MAX_CLIENTS );

    // Si c'est le premier client
    if (pt_clients->nb_clients == 0)
    {
        // Copie le nom_client dans le tableau des clients
        strcpy(pt_clients->tab_clients[pt_clients->nb_clients], nom_client);

        // Incrémente de UN le nombre de client
        pt_clients->nb_clients++;
    }
    else
    {
        /* Parcours tous les clients pour comparé si le nom du client est pareil
                    que le nom du client dans le tableau des clients            */

        unsigned int x = 0;
        // Parcours le nombre de clients
        for (x = 0; x < pt_clients->nb_clients; x++)
```

```
        {
            // Compare le nom du client saisi avec le nom du client dans le tableau
des clients
            if (strcmp(pt_clients->tab_clients[x], nom_client) == 0)
            {
                printf("Le client %s est deja declare, veuillez essayer avec un autre
nom. Merci\n", nom_client);

                // Arrête toute la fonction void enregistreClients
                return;
            }
        }

        // Copie le nom du client dans le tableau des clients
        strcpy(pt_clients->tab_clients[pt_clients->nb_clients], nom_client);

        // Incrémente de UN le nombre de client
        pt_clients->nb_clients++;
    }
}

// Lorsque l'on écrit "client" dans la console
// Brief : Afficher la liste des produits que le client a commandé
void afficheClient(Clients* pt_clients, Commandes* pt_commande)
{
    // Initialise la variable et attend une entrée pour le nom du client
    Mot nom_client;
    get_id(nom_client);

    // Si le nom du client est égale à 'tous'
    if (strcmp(nom_client, "tous") == 0)
    {
        /* Affiche tous les commandes de tout les clients
           Parcours le nombre de clients                    */
        unsigned int x;

        // Parcours le nombre de clients
        for (x = 0; x < pt_clients->nb_clients; x++)
        {
            // Remplace %s par le nom du client dans le tableau des clients
            printf("le client %s a commande : ", pt_clients->tab_clients[x]);

            // Initialise les compteurs
            unsigned int y;
            unsigned int existe = 0;

            // Parcours le nombre de commande
            for (y = 0; y < pt_commande->nb_commandes; y++)
            {
                // Vérifie si à l'index 'x' c'est l'index du client pour la commande
'y'
                if (x == pt_commande->tab_commandes[y].idx_client)
                {
                    // Ajoute la virgule si le nom de la 1ère commande est déjà
affiché
                    if (existe != 0)
                    {
                        printf(", ");
                    }
                    printf("%s", pt_commande->tab_commandes[y].nom);
                    existe++;
                }
            }
```

```
            printf("\n");

        }
        // Le compilateur arrêtera de lire la suite grâce au return
        return;
    }
    // Boucle servant à afficher tout les noms de clients dans le tableau client
    // Parcours le nombre de clients
    unsigned int x;
    for (x = 0; x < pt_clients->nb_clients; x++)
    {
        // Vérifie si le nom du client existe dans le tableau des clients
        if (strcmp(nom_client, pt_clients->tab_clients[x]) == 0)
        {
            // Remplace %s par le nom du client dans le tableau des clients
            printf("le client %s a commande : ", pt_clients->tab_clients[x]);

            // Initialise les compteurs
            unsigned int y;
            unsigned int virgule = 0;

            // Parcours le nombre de commande
            for (y = 0; y < pt_commande->nb_commandes; y++)
            {
                if (x == pt_commande->tab_commandes[y].idx_client)
                {
                    // Ajoute la virgule si le nom de la 1ère commande est déjà
affiché
                    if (virgule != 0)
                    {
                        printf(", ");
                    }
                    printf("%s", pt_commande->tab_commandes[y].nom);
                    virgule++;
                }
            }
            printf("\n");
        }
    }
}

// Lorsque l'on écrit "commande" dans la console
// Brief : Enregistrer la liste de la commande du client
void enregistreNewCommande(Commandes* pt_liste_commandes, Clients* pt_client)
{
    // Initialise les variables et Demande de saisir le nom produit et le nom du
client
    Mot nom_produit;
    get_id(nom_produit);
    Mot nom_client;
    get_id(nom_client);

    // Message d'erreur si le nombre de commandes est supérieur aux nombres de
commandes autorisés
    assert(pt_liste_commandes->nb_commandes <= MAX_COMMANDES);

    // Initialise les compteurs x et y
    unsigned int x, y;

    // Parcours le nombre de clients
    for (x = 0; x < pt_client->nb_clients; x++)
    {
```

```
        // Compare si le nom du client est pareil que celui dans la liste des clients
        if (strcmp(nom_client, pt_client->tab_clients[x]) == 0)
        {
            // Passe à l'index du client le nombre 'y' pour savoir à qui la commande
est destinée

pt_liste_commandes->tab_commandes[pt_liste_commandes->nb_commandes].idx_client = x;

            // Copie le nom du produit dans la liste des commandes

strcpy(pt_liste_commandes->tab_commandes[pt_liste_commandes->nb_commandes].nom,
nom_produit);

            // Incrémente de UN le nombre de commande
            pt_liste_commandes->nb_commandes++;
        }
    }

    // Parcours le nombre maximum de specialités
    for (y = 0; y < MAX_SPECIALITES; y++)
    {
        // Initialise les heures effectuées et les heures requises des taches par
spécialité du client 'x' à 0

pt_liste_commandes->tab_commandes[pt_liste_commandes->nb_commandes].taches_par_special
ite[y].nb_heures_effectuees = 0;

pt_liste_commandes->tab_commandes[pt_liste_commandes->nb_commandes].taches_par_special
ite[y].nb_heures_requises = 0;
    }
}

// Lorsque l'on écrit "progression" dans la console
// Brief : Enregistre la progression d'une commande (actualise l'état de la tâche)
void enregistreProgressionTravail(Commandes* pt_commandes, Specialites*
pt_specialites, Travailleurs* pt_travailleur)
{
    // Initialisation des variables et En attente de trois entrées
    Mot nom_commande;
    get_id(nom_commande);
    Mot nom_specialite;
    get_id(nom_specialite);
    int NbHeure_avancé = get_int();

    unsigned int x, y;
    // Parcours le nombre de commande dans la liste des commandes
    for (x = 0; x < pt_commandes->nb_commandes; x++)
    {
        if (strcmp(nom_commande, pt_commandes->tab_commandes[x].nom) == 0)
        {
            // Parcours le nombre de spécialités dans le tableau des spécialités
proposés par l'entreprise
            for (y = 0; y < pt_specialites->nb_specialites; y++)
            {
                if (strcmp(nom_specialite, pt_specialites->tab_specialites[y].nom) ==
0)
                {

pt_commandes->tab_commandes[x].taches_par_specialite[y].nb_heures_effectuees +=
NbHeure_avancé;

pt_travailleur->tab_travailleurs[pt_commandes->tab_commandes[x].taches_par_specialite[
y].idx_travailleur].charge -= NbHeure_avancé;
```

```
                }
            }
        }
    }
}

// Lorsque l'on écrit "supervision" dans la console
// Brief : Affiche l'avancement d'une commande
void avancementCommande(Commandes* pt_commande, Specialites* pt_specialite)
{

    // Initialisation des compteurs x et z
    unsigned int x, z;

    // Parcours toutes les commandes
    for (x = 0; x < pt_commande->nb_commandes; x++)
    {
        // Vérifie si la commande est celle demandé
        if (pt_commande->nb_commandes != 0)
        {
            printf("etat des taches pour %s : ", pt_commande->tab_commandes[x].nom);

            // Initialise le compteur
            unsigned int virgule = 0;

            // Parcours toutes les spécialités qui sont proposés par l'entreprise
            for (z = 0; z < pt_specialite->nb_specialites; z++)
            {
                /*      Rappel "nom_spe : nb_heure_effectuee / nb_heure_requise"
*/

                // Si le nb_heure_effectue est égale à 0 et que le nb_heure_requise
est différent de 0
                if
(pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_effectuees == 0
                    &&
pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_requises != 0)
                {
                    // Si l'avancement de la première commande est déjà faite alors on
ajoute la virgule
                    if (virgule > 0)
                    {
                        printf(", ");
                    }
                    printf("%s:0/%d", pt_specialite->tab_specialites[z].nom,

pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_requises);
                    virgule++;
                }
                // Si le nb_heure_effectue est strictement supérieur à 0 et que le
nb_heure_requise est différent de 0
                else if
(pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_effectuees > 0
                    &&
pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_requises != 0)
                {
                    // Si l'avancement de la première commande est déjà faite alors on
ajoute la virgule
                    if (virgule > 0)
                    {
                        printf(", ");
                    }
```

```
                    printf("%s:%d/%d", pt_specialite->tab_specialites[z].nom,
pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_effectuees,
pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_requises);
                    virgule++;
                }
            }
            printf("\n");
        }
    }
}

// Lorsque l'on écrit "tache" dans la console
// Brief : Enregistrer la tâche d'une commande qui précise la spécialité et le nb
heure nécessaire
void enregistreTache(Commandes* pt_commande, Specialites* pt_tab_specialites,
Travailleurs* pt_travailleur)
{
    // Initialise les variables et Demande de saisir le nom de la commande
    // Et le nom de la spécialité et le nombre d'heure nécessaire pour réaliser la
tâche
    Mot nom_commande;
    get_id(nom_commande);
    Mot nom_specialite;
    get_id(nom_specialite);
    int heure_nécessaire = get_int();

    // Initialise les compteurs
    unsigned int x, i, j;

    // Initialise à 0 les heures total et l'index du travailleur
    unsigned int heure_total = 0;
    unsigned int idx_travailleur = 0;

    // Initialise la condition de départ à faux (aucune heure totale n'a encore été
définie)
    Booleen heuretotalExiste = FAUX;

    // Initialise les nombres heures requises et effectuées
    Tache tache;
    tache.nb_heures_requises = heure_nécessaire;
    tache.nb_heures_effectuees = 0;

    // Parcours le nombre de commandes
    for (x = 0; x < pt_commande->nb_commandes; x++)
    {
        // Vérifie si cette commande est celle qui est demandée
        if (strcmp(nom_commande, pt_commande->tab_commandes[x].nom) == 0)
        {
            break;
        }
    }

    // Parcours le nombre de spécialités
    for (j = 0; j < pt_tab_specialites->nb_specialites; j++)
    {
        // Vérifie si cette commande est celle qui est demandée
        if (strcmp(nom_specialite, pt_tab_specialites->tab_specialites[j].nom) == 0)
        {
            break;
        }
    }
```

```
    // Parcours le nombre de travailleurs pour déterminer la position du travailleur
    for (i = 0; i < pt_travailleur->nb_travailleurs; i++)
    {
        // Vérifie si la compétence du travailleur existe
        if (pt_travailleur->tab_travailleurs[i].tags_competences[j] == VRAI)
        {
            // Condition pour initier l'heure total sinon il y a rien au début
            if (heuretotalExiste == FAUX)
            {
                heure_total = pt_travailleur->tab_travailleurs[i].charge;
                idx_travailleur = i;
                heuretotalExiste = VRAI;
            }
            else
            {
                if (heure_total > pt_travailleur->tab_travailleurs[i].charge)
                {
                    heure_total = pt_travailleur->tab_travailleurs[i].charge;
                    idx_travailleur = i;
                }
            }
        }
    }
    // Affecte l'index du travailleur à l'index travailleur dans tache
    tache.idx_travailleur = idx_travailleur;

    // Initialise les heures requises et effectuée à la commande 'x' de la tache 'j'
    pt_commande->tab_commandes[x].taches_par_specialite[j] = tache;

    // Ajoute à la charge du travailleur en question le nombre d'heure requise
    pt_travailleur->tab_travailleurs[idx_travailleur].charge +=
tache.nb_heures_requises;
}



// Lorsque l'on écrit "charge" dans la console
/* Brief : Pour le travailleur choisi, affiche le nombre d'heures restantes pour
           l'accomplissement de chacune des tâches qui lui sont affectées.       */
void afficheTacheRestante(Travailleurs* pt_travail, Specialites* pt_spe, Commandes*
pt_commande)
{
    // Initialise la variable et demande de saisir le nom du travailleur
    Mot nom_travailleur;
    get_id(nom_travailleur);

    // Représente l'index travailleur du travailleur qu'on a saisi en Entrée
    unsigned int x;

    // Parcours tous les travailleurs
    for (x = 0; x < pt_travail->nb_travailleurs; x++)
    {
        // Enregistre l'index 'x' qui correspond au nom du travailleur dans la liste
des travailleurs
        if (strcmp(nom_travailleur, pt_travail->tab_travailleurs[x].nom) == 0)
        {
            break;
        }
    }

    // Initialise les compteurs
```

```
    unsigned int y, spe;
    unsigned int compteur = 0;

    // Vérifie si le nom du travailleur existe dans la liste des travailleurs
    if (strcmp(nom_travailleur, pt_travail->tab_travailleurs[x].nom) == 0)
    {
        // Affichage par défaut
        printf("charge de travail pour %s : ", nom_travailleur);
    }

    // Parcours le nombre de commandes
    for (y = 0; y < pt_commande->nb_commandes; y++)
    {
        // Parcours le nombre de spécialités
        for (spe = 0; spe < pt_spe->nb_specialites; spe++)
        {
            if
(pt_commande->tab_commandes[y].taches_par_specialite[spe].idx_travailleur == x)
            {
                if
((pt_commande->tab_commandes[y].taches_par_specialite[spe].nb_heures_requises
                    -
pt_commande->tab_commandes[y].taches_par_specialite[spe].nb_heures_effectuees) > 0)
                {
                    // Un compteur pour ajouter la virgule lorsque la première tâche
est déjà affichée
                    if (compteur != 0)
                    {
                        printf(", ");
                    }
                    printf("%s/%s/%dheure(s)", pt_commande->tab_commandes[y].nom,
pt_spe->tab_specialites[spe].nom,

(pt_commande->tab_commandes[y].taches_par_specialite[spe].nb_heures_requises
                            -
pt_commande->tab_commandes[y].taches_par_specialite[spe].nb_heures_effectuees));
                    compteur++;
                }
            }
        }
    }
    printf("\n");
}

// Lorsque l'on écrit "interruption" dans la console
void Quitter()
{
    printf(MSG_INTERRUPTION);
}
}

#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

/*                                          PROJETS IAP Sprint 1 RELEASE
                                            Auteurs : Enes Inac ET MAXIME WANG
                                            Date de création : 12/10/2020

*/
```

```
                                                    // Toutes les définitions constantes
typedef enum
{
    FAUX = 0, VRAI = 1
}Booleen;
Booleen EchoActif = FAUX;

// Messages emis par les instructions ----------------------------------------
#define MSG_DEVELOPPE "## nouvelle specialite \"%s\" ; cout horaire \"%d\"\n"
#define MSG_SPECIALITES "## consultation des specialites\n"

#define MSG_EMBAUCHE "## nouveau travailleur \"%s\" competent pour la specialite
\"%s\"\n"
#define MSG_TRAVAILLEURS "la specialite \%s\ peut etre prise en charge par :"
#define MSG_TRAVAILLEURS_TOUS "## consultation des travailleurs competents pour chaque
specialite\n"

#define MSG_DEMARCHE "## nouveau client \"%s\"\n"

#define MSG_NOUVELLE_COMMANDE "## nouvelle commande \"%s\", par client \"%s\"\n"
#define MSG_COMMANDE_TOUS  "## consultation des commandes effectuees par chaque
client\n"
#define MSG_CONSULTATION_AVANCEMENT "## consultation de l'avancement des commandes\n"


#define MSG_TACHE "## la commande \"%s\" requiere la specialite \"%s\" (nombre
d'heures \"%d\")\n"
#define MSG_CONSULTATION_AVANCEMENT "## consultation de l'avancement des commandes\n"
#define MSG_CHARGE "## consultation de la charge de travail de \"%s\"\n"

#define MSG_PROGRESSION_TRAVAIL "## pour la commande \"%s\", pour la specialite
\"%s\" : \"%d\" heures de plus ont ete realisees\n"
#define MSG_PASSE "## une reallocation est requise\n"
#define MSG_SUPERVISION "## consultation de l'avancement des commandes\n"
#define MSG_CONSULTATION_AVANCEMENT "## consultation de l'avancement des commandes\n"
#define MSG_CHARGE "## consultation de la charge de travail de \"%s\"\n"

#define MSG_INTERRUPTION "## fin de programme\n"

#define MSG_CONSULTATION_COMMANDE "le client \"%s\" a commande :\n"

// Lexemes -------------------------------------------------------------------
#define LGMOT 35
#define NBCHIFFREMAX 5
#define max_size 100

// specialites ---------------------
#define MAX_SPECIALITES 10
#define MAX_TRAVAILLEURS 50
#define MAX_CLIENTS 10
#define MAX_COMMANDES 500

typedef char Mot[LGMOT + 1];

// Donnees -------------------------------------------------

// Enregistre le nom et cout horaire d'une specialite
typedef struct {
    Mot nom;
    unsigned int cout_horaire;
} Specialite;
```

```c
// Affiche toutes les spécialités proposés et le nombre de spécialité proposés en
entier -------
typedef struct {
    Specialite tab_specialites[MAX_SPECIALITES];
    unsigned int nb_specialites;
} Specialites;

// Enregistre les compétences d'un travailleur --------------------
typedef struct Travailleur {
    Mot nom;
    Booleen tags_competences[MAX_SPECIALITES];
    unsigned int nb_competences;
} Travailleur;

// Affiche tous les travailleurs et le nombre de travailleur en entier
--------------------
typedef struct {
    Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
    unsigned int nb_travailleurs;
} Travailleurs;

// Enregistre une tableau de clients avec leurs noms -----------------
typedef struct {
    Mot tab_clients[MAX_CLIENTS];
    unsigned int nb_clients;
} Clients;

// Initialisation des variables pour Taches des commandes ----------------------
typedef struct {
    unsigned int nb_heures_requises;
    unsigned int nb_heures_effectuees;
} Tache;

// Enregistre un tableau de spécialité, le nom et l'index du client
typedef struct {
    Mot nom;
    unsigned int idx_client;
    Tache taches_par_specialite[MAX_SPECIALITES]; // nb_heures_requises==0 <=> pas
detache pur cette specialite
} Commande;

// Enregistre une liste de commandes et le nombre de commandes
typedef struct {
    Commande tab_commandes[MAX_COMMANDES];
    unsigned int nb_commandes;
} Commandes;

// Fonction de type void ne retourne aucune valeur mais excute les instructions -->
// Nom de la fonction get_id / Paramètre de la fonction Mot id, Mot defini ci-dessus
et id est un string de capacité de mémoire maximum de 35 caractères
void get_id(Mot id)
{
    scanf("%s", id);
    if (EchoActif) printf(">>echo %s\n", id);
}

// fonction de type int
int get_int()
{
    char buffer[NBCHIFFREMAX + 1];
    scanf("%s", buffer);
    if (EchoActif) printf(">>echo %s\n", buffer);
    return atoi(buffer);
```

```
}

// Instructions ----------------------------------------------------------------

// developpe----------------------------
// Objectif : une reallocation est requise
void traite_passe()
{
    printf(MSG_PASSE);
}

// Lorsque l'on écrit "developpe" dans la console
// Brief : Affiche le nom de la spécialité et le coût horaire
//         dans le tableau de spécialités proposés par l'entreprise
void enregistreSpecialites()
{
    // Initialisation des variables et Demande d'entrée le nom et le coût horaire
d'une spécialité
    Mot nom_specialite;
    get_id(nom_specialite);
    int cout_horaire = get_int();
    printf(MSG_DEVELOPPE, nom_specialite, cout_horaire);
}

// Lorsque l'on écrit "specialites" dans la console
/* Brief : Afficher le prix de facturation d’une heure de travail
           pour toutes les spécialités proposés par l'entreprise      */
void affiche_specialites()
{
    printf(MSG_SPECIALITES);
}

// Lorsque l'on écrit "embauche" dans la console
// Brief : Affiche les noms et les compétences des travailleurs dans la liste des
travailleurs
void enregistreEmbauche()
{
    // Initialise et Demande d'entrée le nom et la ou les compétence(s) du travailleur
    Mot nom_travailleur;
    get_id(nom_travailleur);
    Mot competences;
    get_id(competences);
    printf(MSG_EMBAUCHE, nom_travailleur, competences);
}

// Lorsque l'on écrit "travailleurs" dans la console
// Brief : Affiche les noms des travailleurs compétents pour la spécialité d'entrée
void afficheTravailleurs()
{
    // Initialisation de la variable ET
    // Demande d'entrée le nom d'une spécialité
    Mot nom_specialites;
    get_id(nom_specialites);
    if (strcmp(nom_specialites, "tous") == 0) {
        printf(MSG_TRAVAILLEURS_TOUS);
    }
    else {
        printf("## consultation des travailleurs competents pour la specialite
\"%s\"\n", nom_specialites);
    }
}
```

```
// Lorsque l'on écrit "demarche" dans la console
// Brief : Affiche le client dans la liste de client "tab_clients"
void enregistreClients()
{
    // Initialisation des variables et attend une entrée pour le nom du client
    Mot nom_client;
    get_id(nom_client);

    printf(MSG_DEMARCHE, nom_client);
}

// Lorsque l'on écrit "client" dans la console
// Brief : Afficher la liste des produits que le client a commandé
void afficheClient()
{
    // Initialise la variable et attend une entrée pour le nom du client
    Mot nom_client;
    get_id(nom_client);
    if (strcmp(nom_client, "tous") == 0) {
        printf(MSG_COMMANDE_TOUS);
    }
    else {
        printf("## consultation des commandes effectuees par \"%s\"\n", nom_client);
    }
}

// le 1er mot input "commande" ------------------------
void enregistreNewCommande()
{
    Mot produit;
    get_id(produit);
    Mot nom_client;
    get_id(nom_client);
    printf(MSG_NOUVELLE_COMMANDE, produit, nom_client);
}

// le 1er mot input "tache" ------------------------
void enregistreTache()
{
    Mot produit;
    get_id(produit);
    Mot nom_specialite;
    get_id(nom_specialite);
    int nb_heure = get_int();
    printf(MSG_TACHE, produit, nom_specialite, nb_heure);
}

// le 1er mot input "progression" ------------------------
void enregistreProgressionTravail()
{
    Mot produit;
    get_id(produit);
    Mot nom_specialite;
    get_id(nom_specialite);
    int nb_heure = get_int();
    printf(MSG_PROGRESSION_TRAVAIL, produit, nom_specialite, nb_heure);
}

// supervision ------------------------------
void avancementCommande() {
    printf(MSG_SUPERVISION);
}
// le 1er mot input "charge" ------------------------
```

```
void afficheTacheRestante()
{
    Mot nom_travailleur;
    get_id(nom_travailleur);
    printf(MSG_CHARGE, nom_travailleur);
}

// le 1er mot input "interruption" ------------------------
void Quitter()
{
    printf(MSG_INTERRUPTION);
}

// Boucle principale -------------------------------------------------------------
int main(int argc, char* argv[])
{
    if (argc >= 2 && strcmp("echo", argv[1]) == 0)
    {
        EchoActif = VRAI;
    }

    // Buffer de type Mot et ne peut contenir au max 35 caratères
    Mot buffer;

    // Une boucle à l'infini
    while (VRAI)
    {
        // Attente d'une entrée dans la console ------------
        get_id(buffer);

        // Appel de la fonction traite_developpe lorsque l'on écrit 'developpe ...'
dans la console --------------------------------
        if (strcmp(buffer, "developpe") == 0)
        {
            enregistreSpecialites();
            continue;
        }

        // Appel de la fonction traite_specialites lorsque l'on écrit 'specialite ...'
dans la console --------------------------
        if (strcmp(buffer, "specialites") == 0)
        {
            affiche_specialites();
            continue;
        }

        // Appel de la fonction traite_embauche lorsque l'on écrit 'embauche ...' dans
la console ---------------------------------
        if (strcmp(buffer, "embauche") == 0)
        {
            enregistreEmbauche();
            continue;
        }

        // Appel de la fonction traite_travailleurs lorsque l'on écrit
'travailleur ...' dans la console -------------------------
        if (strcmp(buffer, "travailleurs") == 0)
        {
            afficheTravailleurs();
            continue;
        }
```

```
        // Appel de la fonction traite_demarche lorsque l'on écrit 'demarche ...' dans
la console ---------------------------------
        if (strcmp(buffer, "demarche") == 0)
        {
            enregistreClients();
            continue;
        }

        // Appel de la fonction traite_client lorsque l'on écrit 'client ...' dans la
console --------------------------
        if (strcmp(buffer, "client") == 0) {
            afficheClient();
            continue;
        }


        // Appel de la fonction enregistreNewCommande lorsque l'on écrit
'commande ...' dans la console
        if (strcmp(buffer, "commande") == 0)
        {
            enregistreNewCommande();
            continue;
        }

        // Appel de la fonction enregistreTache lorsque l'on écrit 'tache ...' dans la
console
        if (strcmp(buffer, "tache") == 0)
        {
            enregistreTache();
            continue;
        }

        // Appel de la fonction enregistreProgressionTravail lorsque l'on écrit
'progression ...' dans la console
        if (strcmp(buffer, "progression") == 0)
        {
            enregistreProgressionTravail();
            continue;
        }

        // Appel de la fonction avancementCommande lorsque l'on écrit
'supervision ...' dans la console
        if (strcmp(buffer, "supervision") == 0)
        {
            avancementCommande();
            continue;
        }

        // Appel de la fonction afficheTacheRestante lorsque l'on écrit 'charge ...'
dans la console
        if (strcmp(buffer, "charge") == 0)
        {
            afficheTacheRestante();
            continue;
        }

        // Lorsque l'on écrit 'passe' dans la console
        if (strcmp(buffer, "passe") == 0)
        {
            printf("## une reallocation est requise\n");
            continue;
        }
```

```
        // Appel de la fonction Quitter lorsque l'on écrit 'interruption ...' dans la
console
        if (strcmp(buffer, "interruption") == 0)
        {
            Quitter();
            break;
        }

        // Un message d'erreur comme quoi notre entrée n'est pas reconnu
        printf("!!! instruction inconnue >%s< !!!\n", buffer);
    }
    return 0;
}

#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

/*                                              PROJETS IAP Sprint 2 RELEASE
                                                Auteurs : Enes Inac ET MAXIME WANG
                                                Date de création : 16/10/2020

*/

// Toutes les définitions constantes
typedef enum
{
    FAUX = 0, VRAI = 1
}Booleen;
Booleen EchoActif = FAUX;

// Messages emis par les instructions ----------------------------------------
#define MSG_DEVELOPPE "## nouvelle specialite \"%s\" ; cout horaire \"%d\"\n"
#define MSG_SPECIALITES "## consultation des specialites\n"

#define MSG_EMBAUCHE "## nouveau travailleur \"%s\" competent pour la specialite
\"%s\"\n"
#define MSG_TRAVAILLEURS "la specialite \%s\ peut etre prise en charge par :"
#define MSG_TRAVAILLEURS_TOUS "## consultation des travailleurs competents pour chaque
specialite\n"

#define MSG_DEMARCHE "## nouveau client \"%s\"\n"

#define MSG_NOUVELLE_COMMANDE "## nouvelle commande \"%s\", par client \"%s\"\n"
#define MSG_COMMANDE_TOUS  "## consultation des commandes effectuees par chaque
client\n"
#define MSG_CONSULTATION_AVANCEMENT "## consultation de l'avancement des commandes\n"


#define MSG_TACHE "## la commande \"%s\" requiere la specialite \"%s\" (nombre
d'heures \"%d\")\n"
#define MSG_CONSULTATION_AVANCEMENT "## consultation de l'avancement des commandes\n"
#define MSG_CHARGE "## consultation de la charge de travail de \"%s\"\n"

#define MSG_PROGRESSION_TRAVAIL "## pour la commande \"%s\", pour la specialite
\"%s\" : \"%d\" heures de plus ont ete realisees\n"
#define MSG_PASSE "## une reallocation est requise\n"
#define MSG_SUPERVISION "## consultation de l'avancement des commandes\n"
#define MSG_CONSULTATION_AVANCEMENT "## consultation de l'avancement des commandes\n"
#define MSG_CHARGE "## consultation de la charge de travail de \"%s\"\n"
```

```c
#define MSG_INTERRUPTION "## fin de programme\n"

#define MSG_CONSULTATION_COMMANDE "le client \"%s\" a commande :\n"

// Lexemes ------------------------------------------------------------------
#define LGMOT 35
#define NBCHIFFREMAX 5
#define max_size 100

// specialites ----------------------
#define MAX_SPECIALITES 10
#define MAX_TRAVAILLEURS 50
#define MAX_CLIENTS 10
#define MAX_COMMANDES 500

typedef char Mot[LGMOT + 1];

// Donnees ------------------------------------------------------

// Enregistre le nom et cout horaire d'une specialite
typedef struct {
    Mot nom;
    unsigned int cout_horaire;
} Specialite;

// Affiche toutes les spécialités proposés et le nombre de spécialité proposés en
entier -------
typedef struct {
    Specialite tab_specialites[MAX_SPECIALITES];
    unsigned int nb_specialites;
} Specialites;

// Enregistre les compétences d'un travailleur --------------------
typedef struct Travailleur {
    Mot nom;
    Booleen tags_competences[MAX_SPECIALITES];
    unsigned int nb_competences;
} Travailleur;

// Affiche tous les travailleurs et le nombre de travailleur en entier
--------------------
typedef struct {
    Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
    unsigned int nb_travailleurs;
} Travailleurs;

// Enregistre une tableau de clients avec leurs noms -----------------
typedef struct {
    Mot tab_clients[MAX_CLIENTS];
    unsigned int nb_clients;
} Clients;

// Initialisation des variables pour Taches des commandes -----------------------
typedef struct {
    unsigned int nb_heures_requises;
    unsigned int nb_heures_effectuees;
} Tache;

// Enregistre un tableau de spécialité, le nom et l'index du client
typedef struct {
    Mot nom;
    unsigned int idx_client;
```

```
    Tache taches_par_specialite[MAX_SPECIALITES]; // nb_heures_requises==0 <=> pas
detache pur cette specialite
} Commande;

// Enregistre une liste de commandes et le nombre de commandes
typedef struct {
    Commande tab_commandes[MAX_COMMANDES];
    unsigned int nb_commandes;
} Commandes;
// Fonction de type void ne retourne aucune valeur mais excute les instructions -->
// Nom de la fonction get_id / Paramètre de la fonction Mot id, Mot defini ci-dessus
et id est un string de capacité de mémoire maximum de 35 caractères
void get_id(Mot id)
{
    scanf("%s", id);
    if (EchoActif) printf(">>echo %s\n", id);
}

// fonction de type int
int get_int()
{
    char buffer[NBCHIFFREMAX + 1];
    scanf("%s", buffer);
    if (EchoActif) printf(">>echo %s\n", buffer);
    return atoi(buffer);
}

// Instructions ----------------------------------------------------------------

// developpe----------------------------
// Objectif : une reallocation est requise
void traite_passe()
{
    printf(MSG_PASSE);
}

// Lorsque l'on écrit "developpe" dans la console
// Brief : Enregistre le nom de la spécialité et le coût horaire
//         dans le tableau de spécialités proposés par l'entreprise
void enregistreSpecialites(Specialites* pt_liste_specialites)
{
    // Initialisation des variables et Demande d'entrée le nom et le coût horaire
d'une spécialité
    Specialite cout_specialite;
    Mot nom_specialite;
    get_id(nom_specialite);
    int cout_horaire = get_int();

    // Copie le nom et le coût horaire de la spécialité dans "Specialite"
    strcpy(cout_specialite.nom, nom_specialite);
    cout_specialite.cout_horaire = cout_horaire;

    /* Parcours toutes les spécialités pour comparer si le nom de la spécialité
d'entrée est pareille
                                que celui dans la liste des spécialités
*/
    unsigned int x;
    for (x = 0; x < pt_liste_specialites->nb_specialites; x++)
    {
        // Compare le nom de la spécialité avec le nom de spécialité dans le tableau
des spécialités
        if (strcmp(nom_specialite, pt_liste_specialites->tab_specialites[x].nom) == 0)
```

```
        {
            // Message d'erreur
            assert(nom_specialite == pt_liste_specialites->tab_specialites[x].nom);

            // Arrête la fonction void enregistreSpecialites grâce au return
            return;
        }
    }
    // Enregistre le nom et le coût horaire d'une spécialité dans "Specialites" (les
tab des spécialités)
    pt_liste_specialites->tab_specialites[pt_liste_specialites->nb_specialites] =
cout_specialite;

    // Incrémente de UN le nombre de spécialité
    pt_liste_specialites->nb_specialites++;
}

// Lorsque l'on écrit "specialites" dans la console
/* Brief : Afficher le prix de facturation d'une heure de travail
           pour toutes les spécialités proposés par l'entreprise      */
void affiche_specialites(Specialites* pt_specialites)
{

    // Ajoute les spécialités proposés ainsi que leurs coûts horaires à la suite du
MSG ci-dessous
    printf("specialites traitees :");

    unsigned int virgule = 0;
    unsigned int x;

    // Boucle qui parcours toutes les spécialités ET renvoie le nom ainsi que son prix
de facturation
    for (x = 0; x < pt_specialites->nb_specialites; x++)
    {
        // Ajoute la virgule si il a déjà afficher une fois le nom de la spécialité et
le cout horaire de la spé
        if (virgule != 0)
        {
            printf(",");
        }
        printf(" %s/%d", pt_specialites->tab_specialites[x].nom,
pt_specialites->tab_specialites[x].cout_horaire);
        virgule++;
    }
    printf("\n");
}
// Lorsque l'on écrit "embauche" dans la console
// Brief : Enregistrer les noms et les compétences des travailleurs dans la liste des
travailleurs
void enregistreEmbauche(Travailleurs* pt_list_travailleurs, Specialites*
pt_specialites)
{
    // Initialise et Demande d'entrée le nom et la ou les compétence(s) du travailleur
    Mot nom_travailleur;
    get_id(nom_travailleur);
    Mot competences;
    get_id(competences);

    // Message d'erreur si le nombre de travailleurs est supérieur aux nombres de
travailleurs autorisés
    assert(pt_list_travailleurs->nb_travailleurs <= MAX_TRAVAILLEURS);
```

```
    // Initialisation des variables x et y pour la boucle for ET
    // Un indicateur pour savoir qu'un travailleur existe déjà
    unsigned int x, y;
    unsigned int travailleur_existant = 0;

    // Parcours la boucle tant qu'une spécialité est proposé par l'entreprise
    for (x = 0; x < pt_specialites->nb_specialites; x++)
    {
        // Parcours la liste des travailleurs
        for (y = 0; y < pt_list_travailleurs->nb_travailleurs; y++)
        {
            /* Vérifie si le nom de la compétence d'entrée est pareille
               Que le nom de la spécialité dans la LISTE DES SPECIALITES proposés par
l'entreprise */
            if (strcmp(pt_specialites->tab_specialites[x].nom, competences) == 0)
            {

                // Vérifie si le nom du travailleur existe déjà dans la liste des
travailleurs
                if (strcmp(pt_list_travailleurs->tab_travailleurs[y].nom,
nom_travailleur) == 0)
                {
                    // Le travailleur existe déjà ainsi la compétence est vrai pour ce
travailleur
                    pt_list_travailleurs->tab_travailleurs[y].tags_competences[x] =
VRAI;

                    // Incrémente 'travailleur_existant' de UN, cela veut dire que ce
travailleur existe déjà
                    travailleur_existant++;
                }
            }
        }
        /* Condition qui compare la compétence d'entrée avec le nom de spécialité dans
la liste spécialité ET
              L'indicateur 'travailleur_existant = 0' nous montre que aucun
travailleur n'a été déclaré à ce nom       */

        if (strcmp(pt_specialites->tab_specialites[x].nom, competences) == 0 &&
travailleur_existant == 0)
        {
            // Créer un nouveau travailleur en copiant le nom du travailleur dans le
tableau des travailleurs

strcpy(pt_list_travailleurs->tab_travailleurs[pt_list_travailleurs->nb_travailleurs].n
om, nom_travailleur);

            // Enregistre la compétence du travailleur en booleen càd VRAI/FAUX dans
le tableau des compétences de ce travailleur

pt_list_travailleurs->tab_travailleurs[pt_list_travailleurs->nb_travailleurs].tags_com
petences[x] = VRAI;

            // Incrémente de UN le nombre de travailleurs
            pt_list_travailleurs->nb_travailleurs++;
        }
    }
}

// Lorsque l'on écrit "travailleurs" dans la console
// Brief : Affiche les noms des travailleurs compétents pour la spécialité d'entrée
void afficheTravailleurs(Travailleurs* pt_travailleurs, Specialites* pt_specialites)
```

```
{
    // Initialisation de la variable ET
    // Demande d'entrée le nom d'une spécialité
    Mot nom_specialites;
    get_id(nom_specialites);

    // Initialisation des variables x et y pour la boucle for
    unsigned int x, y;

    // Si l'entrée était "travailleurs tous"
    if (strcmp(nom_specialites, "tous") == 0)
    {
        // Indicateur 'condition = FAUX' veut dire que aucun travailleur n'a été
ajouter dans la phrase ci-dessous
        Booleen notFirst = FAUX;

        // Parcours le nombre de spécialités proposés par l'entreprise
        for (x = 0; x < pt_specialites->nb_specialites; x++)
        {
            // Remplace le %s par le nom de la spécialité
            printf("la specialite %s peut etre prise en charge par :",
pt_specialites->tab_specialites[x].nom);

            // Parcours tous les travailleurs de l'entreprise
            for (y = 0; y < pt_travailleurs->nb_travailleurs; y++)
            {

                // Condition qui vérifie si le travailleur 'y' possède la compétence
                if (pt_travailleurs->tab_travailleurs[y].tags_competences[x] == 1)
                {
                    // Condition qui vérifie si ce n'est pas le 1er travailleur
                    if (notFirst == VRAI && pt_travailleurs->nb_travailleurs > 0)
                    {
                        // Pour les travailleurs après le 1er travailleur on ajoute
une virgule
                        printf(",");
                    }
                    // Affiche le nom des travailleurs
                    printf(" %s", pt_travailleurs->tab_travailleurs[y].nom);
                    notFirst = VRAI;
                }
            }
            // Retour à la ligne pour chaque consultation des travailleurs compétents
à la spécialité
            printf("\n");

            // La condition redevient FAUX avant de retourner dans la boucle qui
parcours les spécialités
            notFirst = FAUX;
        }
    }
    // Condition qui vérifie si au moins un travailleur existe
    else if (pt_travailleurs->nb_travailleurs > 0)
    {
        // Initialisation des variables x et y pour la boucle for
        unsigned int x, y;

        // Indicateur 'condition = FAUX' veut dire que aucun travailleur n'a été
ajouter dans la phrase ci-dessous
        Booleen notFirst = FAUX;

        // Parcour le nombre de spécialités
```

```
        for (x = 0; x < pt_specialites->nb_specialites; x++)
        {
            // Condition qui vérifie si le nom de la spécialité d'entrée est pareil
            // Que le nom de la spécialité dans le tableau des spécialités
            if (strcmp(nom_specialites, pt_specialites->tab_specialites[x].nom) == 0)
            {

                // Remplace le %s par le nom de la spécialité
                printf("la specialite %s peut etre prise en charge par :",
pt_specialites->tab_specialites[x].nom);

                // Parcours tous les travailleurs de l'entreprise
                for (y = 0; y < pt_travailleurs->nb_travailleurs; y++)
                {
                    // Condition qui vérifie si le travailleur est compétent(e) pour
cette spécialité
                    if (pt_travailleurs->tab_travailleurs[y].tags_competences[x] ==
VRAI)
                    {

                        // Condition qui vérifie si un travailleur a déjà été ajouter
à la phrase ci-dessus ou pas
                        if (notFirst == VRAI && pt_travailleurs->nb_travailleurs > 0)
                        {
                            // Pour les travailleurs après le 1er travailleur on
ajoute une virgule
                            printf(",");
                        }
                        // Affiche le nom des travailleurs
                        printf(" %s", pt_travailleurs->tab_travailleurs[y].nom);

                        // La condition devient VRAI càd un travailleur a déjà été
ajouter
                        notFirst = VRAI;
                    }
                }
                // Retour à la ligne pour chaque consultation des travailleurs
compétents à la spécialité demandé
                printf("\n");

                // La condition redevient FAUX avant de retourner dans la boucle qui
parcours les spécialités
                notFirst = FAUX;
            }
        }
    }
}
// Lorsque l'on écrit "demarche" dans la console
// Brief : Enregistrer le client dans la liste de client "tab_clients"
void enregistreClients(Clients* pt_clients)
{
    // Initialisation des variables et attend une entrée pour le nom du client
    Mot nom_client;
    get_id(nom_client);

    // Message d'erreur si le nombre de clients est supérieur aux nombres de clients
autorisés
    assert(pt_clients->nb_clients <= MAX_CLIENTS);

    // Si c'est le premier client
    if (pt_clients->nb_clients == 0)
    {
```

```
        // Copie le nom_client dans le tableau des clients
        strcpy(pt_clients->tab_clients[pt_clients->nb_clients], nom_client);

        // Incrémente de UN le nombre de client
        pt_clients->nb_clients++;
    }
    else
    {
        /* Parcours tous les clients pour comparé si le nom du client est pareil
                    que le nom du client dans le tableau des clients            */

        unsigned int x = 0;
        // Parcours le nombre de clients
        for (x = 0; x < pt_clients->nb_clients; x++)
        {
            // Compare le nom du client saisi avec le nom du client dans le tableau
des clients
            if (strcmp(pt_clients->tab_clients[x], nom_client) == 0)
            {
                printf("Le client %s est deja declare, veuillez essayer avec un autre
nom. Merci\n", nom_client);

                // Arrête toute la fonction void enregistreClients
                return;
            }
        }

        // Copie le nom du client dans le tableau des clients
        strcpy(pt_clients->tab_clients[pt_clients->nb_clients], nom_client);

        // Incrémente de UN le nombre de client
        pt_clients->nb_clients++;
    }
}

// Lorsque l'on écrit "client" dans la console
// Brief : Afficher la liste des produits que le client a commandé
void afficheClient(Clients* clients)
{
    // Initialise la variable et attend une entrée pour le nom du client
    Mot nom_client;
    get_id(nom_client);

    // Si le nom du client est égale à 'tous'
    if (strcmp(nom_client, "tous") == 0)
    {
        /* Affiche tous les commandes de tout les clients
           Parcours le nombre de clients                    */
        for (unsigned int x = 0; x < clients->nb_clients; x++)
        {
            // Remplace %s par le nom du client dans le tableau des clients
            printf("le client %s a commande : \n", clients->tab_clients[x]);
        }
    }
    else
    {
        // Boucle servant à afficher tout les noms de clients dans le tableau client
        // Parcours le nombre de clients
        for (unsigned int x = 0; x < clients->nb_clients; x++)
        {
            // Condition vérifiant si le nom du client saisi au départ
            // correspond à l'élément indice x dans le tableau des clients
```

```
            if (strcmp(nom_client, clients->tab_clients[x]) == 0)
            {
                // La condition est valide donc ça nous affiche la commande du client
                printf("le client %s a commande : \n", clients->tab_clients[x]);
            }
        }
    }
}
// le 1er mot input "commande" ------------------------
void enregistreNewCommande()
{
    Mot produit;
    get_id(produit);
    Mot nom_client;
    get_id(nom_client);
    printf(MSG_NOUVELLE_COMMANDE, produit, nom_client);
}

// le 1er mot input "tache" ------------------------
void enregistreTache()
{
    Mot produit;
    get_id(produit);
    Mot nom_specialite;
    get_id(nom_specialite);
    int nb_heure = get_int();
    printf(MSG_TACHE, produit, nom_specialite, nb_heure);
}

// le 1er mot input "progression" ------------------------
void enregistreProgressionTravail()
{
    Mot produit;
    get_id(produit);
    Mot nom_specialite;
    get_id(nom_specialite);
    int nb_heure = get_int();
    printf(MSG_PROGRESSION_TRAVAIL, produit, nom_specialite, nb_heure);
}

// supervision ------------------------------
void avancementCommande() {
    printf(MSG_SUPERVISION);
}
// le 1er mot input "charge" ------------------------
void afficheTacheRestante()
{
    Mot nom_travailleur;
    get_id(nom_travailleur);
    printf(MSG_CHARGE, nom_travailleur);
}

// le 1er mot input "interruption" ------------------------
void Quitter()
{
    printf(MSG_INTERRUPTION);
}
// Boucle principale ----------------------------------------------------------
int main(int argc, char* argv[])
{

    // Initialisation des variables
```

```
    Specialites couts_specialite = { " " , 0 };

    Travailleurs Enregistrement_Embauche = { " " , 0 };

    Clients Tout_Mes_Clients = { " " , 0 };


    if (argc >= 2 && strcmp("echo", argv[1]) == 0)
    {
        EchoActif = VRAI;
    }

    // Buffer de type Mot et ne peut contenir au max 35 caratères
    Mot buffer;

    // Une boucle à l'infini
    while (VRAI)
    {
        // Attente d'une entrée dans la console ------------
        get_id(buffer);

        // Appel de la fonction traite_developpe lorsque l'on écrit 'developpe ...'
dans la console --------------------------------
        if (strcmp(buffer, "developpe") == 0)
        {
            enregistreSpecialites(&couts_specialite);
            continue;
        }

        // Appel de la fonction traite_specialites lorsque l'on écrit 'specialite ...'
dans la console --------------------------
        if (strcmp(buffer, "specialites") == 0)
        {
            affiche_specialites(&couts_specialite);
            continue;
        }

        // Appel de la fonction traite_embauche lorsque l'on écrit 'embauche ...' dans
la console ---------------------------------
        if (strcmp(buffer, "embauche") == 0)
        {
            enregistreEmbauche(&Enregistrement_Embauche, &couts_specialite);
            continue;
        }

        // Appel de la fonction traite_travailleurs lorsque l'on écrit
'travailleur ...' dans la console --------------------------
        if (strcmp(buffer, "travailleurs") == 0)
        {
            afficheTravailleurs(&Enregistrement_Embauche, &couts_specialite);
            continue;
        }

        // Appel de la fonction traite_demarche lorsque l'on écrit 'demarche ...' dans
la console ---------------------------------
        if (strcmp(buffer, "demarche") == 0)
        {
            enregistreClients(&Tout_Mes_Clients);
            continue;
        }
```

```
        // Appel de la fonction traite_client lorsque l'on écrit 'client ...' dans la
console -------------------------
        if (strcmp(buffer, "client") == 0) {
            afficheClient(&Tout_Mes_Clients);
            continue;
        }

        // Appel de la fonction enregistreNewCommande lorsque l'on écrit
'commande ...' dans la console
        if (strcmp(buffer, "commande") == 0)
        {
            enregistreNewCommande();
            continue;
        }

        // Appel de la fonction enregistreTache lorsque l'on écrit 'tache ...' dans la
console
        if (strcmp(buffer, "tache") == 0)
        {
            enregistreTache();
            continue;
        }

        // Appel de la fonction enregistreProgressionTravail lorsque l'on écrit
'progression ...' dans la console
        if (strcmp(buffer, "progression") == 0)
        {
            enregistreProgressionTravail();
            continue;
        }

        // Appel de la fonction avancementCommande lorsque l'on écrit
'supervision ...' dans la console
        if (strcmp(buffer, "supervision") == 0)
        {
            avancementCommande();
            continue;
        }

        // Appel de la fonction afficheTacheRestante lorsque l'on écrit 'charge ...'
dans la console
        if (strcmp(buffer, "charge") == 0)
        {
            afficheTacheRestante();
            continue;
        }

        // Lorsque l'on écrit 'passe' dans la console
        if (strcmp(buffer, "passe") == 0)
        {
            printf("## une reallocation est requise\n");
            continue;
        }

        // Appel de la fonction Quitter lorsque l'on écrit 'interruption ...' dans la
console
        if (strcmp(buffer, "interruption") == 0)
        {
            Quitter();
            break;
        }
```

```
        // Un message d'erreur comme quoi notre entrée n'est pas reconnu
        printf("!!! instruction inconnue >%s< !!!\n", buffer);
    }
    return 0;
}
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

/*                                              PROJETS IAP Sprint 3 RELEASE
                                                Auteurs : Enes Inac ET MAXIME WANG
                                                Date de création : 18/10/2020
*/

// Toutes les définitions constantes
typedef enum
{
    FAUX = 0, VRAI = 1
}Booleen;
Booleen EchoActif = FAUX;

// Messages emis par les instructions ----------------------------------------
// Definie Les Messages
#define MSG_CHARGE "## consultation de la charge de travail de \"%s\"\n"

#define MSG_INTERRUPTION "## fin de programme\n"

// Lexemes -------------------------------------------------------------------
#define LGMOT 35
#define NBCHIFFREMAX 5
#define max_size 100

// specialites ---------------------
#define MAX_SPECIALITES 10
#define MAX_TRAVAILLEURS 50
#define MAX_CLIENTS 100
#define MAX_COMMANDES 500

typedef char Mot[LGMOT + 1];

// Donnees -------------------------------------------------------

// Enregistre le nom et cout horaire d'une specialite
typedef struct {
    Mot nom;
    unsigned int cout_horaire;
} Specialite;

// Affiche toutes les spécialités proposés et le nombre de spécialité proposés en
entier -------
typedef struct {
    Specialite tab_specialites[MAX_SPECIALITES];
    unsigned int nb_specialites;
} Specialites;

// Enregistre les compétences d'un travailleur --------------------
typedef struct Travailleur {
    Mot nom;
    Booleen tags_competences[MAX_SPECIALITES];
    unsigned int nb_competences;
```

```
    unsigned int charge;
} Travailleur;

// Affiche tous les travailleurs et le nombre de travailleur en entier
--------------------
typedef struct {
    Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
    unsigned int nb_travailleurs;
} Travailleurs;

// Enregistre une tableau de clients avec leurs noms -----------------
typedef struct {
    Mot tab_clients[MAX_CLIENTS];
    unsigned int nb_clients;
} Clients;

// Initialisation des variables pour Taches des commandes ------------------------
typedef struct {
    int nb_heures_effectuees;
    unsigned int nb_heures_requises;
    unsigned int idx_travailleur;
} Tache;

// Enregistre un tableau de spécialité, le nom et l'index du client
typedef struct {
    Mot nom;
    unsigned int idx_client;
    Tache taches_par_specialite[MAX_SPECIALITES]; // nb_heures_requises == 0 <=> pas
de tache pour cette specialite
} Commande;

// Enregistre une liste de commandes et le nombre de commandes
typedef struct {
    Commande tab_commandes[MAX_COMMANDES];
    unsigned int nb_commandes;
} Commandes;

// Fonction de type void ne retourne aucune valeur mais excute les instructions -->
// Nom de la fonction get_id / Paramètre de la fonction Mot id,
// 'Mot' defini ci-dessus et id est un string de capacité de mémoire maximum de 35
caractères
void get_id(Mot id)
{
    scanf("%s", id);
    if (EchoActif) printf(">>echo %s\n", id);
}

// fonction de type int
int get_int()
{
    char buffer[NBCHIFFREMAX + 1];
    scanf("%s", buffer);
    if (EchoActif) printf(">>echo %s\n", buffer);
    return atoi(buffer);
}
// Instructions ------------------------------------------------------------

// Lorsque l'on écrit "developpe" dans la console
// Brief : Enregistre le nom de la spécialité et le coût horaire
//         dans le tableau de spécialités proposés par l'entreprise
void enregistreSpecialites(Specialites* pt_liste_specialites)
{
```

```
    // Initialisation des variables et Demande d'entrée le nom et le coût horaire
d'une spécialité
    Specialite cout_specialite;
    Mot nom_specialite;
    get_id(nom_specialite);
    int cout_horaire = get_int();

    // Copie le nom et le coût horaire de la spécialité dans "Specialite"
    strcpy(cout_specialite.nom, nom_specialite);
    cout_specialite.cout_horaire = cout_horaire;

    /* Parcours toutes les spécialités pour comparer si le nom de la spécialité
d'entrée est pareille
                              que celui dans la liste des spécialités
*/
    unsigned int x;
    for (x = 0; x < pt_liste_specialites->nb_specialites; x++)
    {
        // Compare le nom de la spécialité avec le nom de spécialité dans le tableau
des spécialités
        if (strcmp(nom_specialite, pt_liste_specialites->tab_specialites[x].nom) == 0)
        {
            // Message d'erreur
            assert(nom_specialite == pt_liste_specialites->tab_specialites[x].nom);

            // Arrête la fonction void enregistreSpecialites grâce au return
            return;
        }
    }
    // Enregistre le nom et le coût horaire d'une spécialité dans "Specialites" (les
tab des spécialités)
    pt_liste_specialites->tab_specialites[pt_liste_specialites->nb_specialites] =
cout_specialite;

    // Incrémente de UN le nombre de spécialité
    pt_liste_specialites->nb_specialites++;
}


// Lorsque l'on écrit "specialites" dans la console
/* Brief : Afficher le prix de facturation d'une heure de travail
           pour toutes les spécialités proposés par l'entreprise      */
void afficheSpecialites(Specialites* pt_specialites)
{
    // Ajoute les spécialités proposés ainsi que leurs coûts horaires à la suite du
MSG ci-dessous
    printf("specialites traitees :");

    unsigned int virgule = 0;
    unsigned int x;

    // Boucle qui parcours toutes les spécialités ET renvoie le nom ainsi que son prix
de facturation
    for (x = 0; x < pt_specialites->nb_specialites; x++)
    {
        // Ajoute la virgule si il a déjà afficher une fois le nom de la spécialité et
le cout horaire de la spé
        if (virgule != 0)
        {
            printf(",");
        }
```

```
        printf(" %s/%d", pt_specialites->tab_specialites[x].nom,
pt_specialites->tab_specialites[x].cout_horaire);
        virgule++;
    }
    printf("\n");
}


// Lorsque l'on écrit "embauche" dans la console
// Brief : Enregistrer les noms et les compétences des travailleurs dans la liste des
travailleurs
void enregistreEmbauche(Travailleurs* pt_list_travailleurs, Specialites*
pt_specialites)
{
    // Initialise et Demande d'entrée le nom et la ou les compétence(s) du travailleur
    Mot nom_travailleur;
    get_id(nom_travailleur);
    Mot competences;
    get_id(competences);

    // Message d'erreur si le nombre de travailleurs est supérieur aux nombres de
travailleurs autorisés
    assert(pt_list_travailleurs->nb_travailleurs <= MAX_TRAVAILLEURS);

    // Initialisation des variables x et y pour la boucle for ET
    // Un indicateur pour savoir qu'un travailleur existe déjà
    unsigned int x, y;
    unsigned int travailleur_existant = 0;

    // Parcours la boucle tant qu'une spécialité est proposé par l'entreprise
    for (x = 0; x < pt_specialites->nb_specialites; x++)
    {
        // Parcours la liste des travailleurs
        for (y = 0; y < pt_list_travailleurs->nb_travailleurs; y++)
        {
            /* Vérifie si le nom de la compétence d'entrée est pareille
               Que le nom de la spécialité dans la LISTE DES SPECIALITES proposés par
l'entreprise */
            if (strcmp(pt_specialites->tab_specialites[x].nom, competences) == 0)
            {

                // Vérifie si le nom du travailleur existe déjà dans la liste des
travailleurs
                if (strcmp(pt_list_travailleurs->tab_travailleurs[y].nom,
nom_travailleur) == 0)
                {
                    // Le travailleur existe déjà ainsi la compétence est vrai pour ce
travailleur
                    pt_list_travailleurs->tab_travailleurs[y].tags_competences[x] =
VRAI;

                    // Incrémente 'travailleur_existant' de UN, cela veut dire que ce
travailleur existe déjà
                    travailleur_existant++;
                }
            }
        }
        /* Condition qui compare la compétence d'entrée avec le nom de spécialité dans
la liste spécialité ET
              L'indicateur 'travailleur_existant = 0' nous montre que aucun
travailleur n'a été déclaré à ce nom       */
```

```
        if (strcmp(pt_specialites->tab_specialites[x].nom, competences) == 0 &&
travailleur_existant == 0)
        {
            // Initialise le nombre d'heure de charge d'un travailleur à 0

pt_list_travailleurs->tab_travailleurs[pt_list_travailleurs->nb_travailleurs].charge =
0;

            // Créer un nouveau travailleur en copiant le nom du travailleur dans le
tableau des travailleurs

strcpy(pt_list_travailleurs->tab_travailleurs[pt_list_travailleurs->nb_travailleurs].n
om, nom_travailleur);

            // Enregistre la compétence du travailleur en booleen càd VRAI/FAUX dans
le tableau des compétences de ce travailleur

pt_list_travailleurs->tab_travailleurs[pt_list_travailleurs->nb_travailleurs].tags_com
petences[x] = VRAI;

            // Incrémente de UN le nombre de travailleurs
            pt_list_travailleurs->nb_travailleurs++;
        }
    }
}
// Lorsque l'on écrit "travailleurs" dans la console
// Brief : Affiche les noms des travailleurs compétents pour la spécialité d'entrée
void afficheTravailleurs(Travailleurs* pt_travailleurs, Specialites* pt_specialites)
{

    // Initialisation de la variable ET
    // Demande d'entrée le nom d'une spécialité
    Mot nom_specialites;
    get_id(nom_specialites);

    // Initialisation des variables x et y pour la boucle for
    unsigned int x, y;

    // Si l'entrée était "travailleurs tous"
    if (strcmp(nom_specialites, "tous") == 0)
    {
        // Indicateur 'condition = FAUX' veut dire que aucun travailleur n'a été
ajouter dans la phrase ci-dessous
        Booleen notFirst = FAUX;

        // Parcours le nombre de spécialités proposés par l'entreprise
        for (x = 0; x < pt_specialites->nb_specialites; x++)
        {
            // Remplace le %s par le nom de la spécialité
            printf("la specialite %s peut etre prise en charge par :",
pt_specialites->tab_specialites[x].nom);

            // Parcours tous les travailleurs de l'entreprise
            for (y = 0; y < pt_travailleurs->nb_travailleurs; y++)
            {

                // Condition qui vérifie si le travailleur 'y' possède la compétence
                if (pt_travailleurs->tab_travailleurs[y].tags_competences[x] == 1)
                {

                    // Condition qui vérifie si ce n'est pas le 1er travailleur
                    if (notFirst == VRAI && pt_travailleurs->nb_travailleurs > 0)
```

```
                    {
                        // Pour les travailleurs après le 1er travailleur on ajoute
une virgule
                        printf(",");
                    }
                    // Affiche le nom des travailleurs
                    printf(" %s", pt_travailleurs->tab_travailleurs[y].nom);
                    notFirst = VRAI;
                }
            }
            // Retour à la ligne pour chaque consultation des travailleurs compétents
à la spécialité
            printf("\n");

            // La condition redevient FAUX avant de retourner dans la boucle qui
parcours les spécialités
            notFirst = FAUX;
        }
    }

    // Condition qui vérifie si au moins un travailleur existe
    else if (pt_travailleurs->nb_travailleurs > 0)
    {
        // Initialisation des variables x et y pour la boucle for
        unsigned int x, y;

        // Indicateur 'condition = FAUX' veut dire que aucun travailleur n'a été
ajouter dans la phrase ci-dessous
        Booleen notFirst = FAUX;

        // Parcour le nombre de spécialités
        for (x = 0; x < pt_specialites->nb_specialites; x++)
        {
            // Condition qui vérifie si le nom de la spécialité d'entrée est pareil
            // Que le nom de la spécialité dans le tableau des spécialités
            if (strcmp(nom_specialites, pt_specialites->tab_specialites[x].nom) == 0)
            {

                // Remplace le %s par le nom de la spécialité
                printf("la specialite %s peut etre prise en charge par :",
pt_specialites->tab_specialites[x].nom);

                // Parcours tous les travailleurs de l'entreprise
                for (y = 0; y < pt_travailleurs->nb_travailleurs; y++)
                {

                    // Condition qui vérifie si le travailleur est compétent(e) pour
cette spécialité
                    if (pt_travailleurs->tab_travailleurs[y].tags_competences[x] ==
VRAI)
                    {

                        // Condition qui vérifie si un travailleur a déjà été ajouter
à la phrase ci-dessus ou pas
                        if (notFirst == VRAI && pt_travailleurs->nb_travailleurs > 0)
                        {
                            // Pour les travailleurs après le 1er travailleur on
ajoute une virgule
                            printf(",");
                        }
                        // Affiche le nom des travailleurs
                        printf(" %s", pt_travailleurs->tab_travailleurs[y].nom);
```

```
                        // La condition devient VRAI càd un travailleur a déjà été
ajouter
                        notFirst = VRAI;
                    }

                }
                // Retour à la ligne pour chaque consultation des travailleurs
compétents à la spécialité demandé
                printf("\n");

                // La condition redevient FAUX avant de retourner dans la boucle qui
parcours les spécialités
                notFirst = FAUX;
            }
        }
    }
}
// Lorsque l'on écrit "demarche" dans la console
// Brief : Enregistrer le client dans la liste de client "tab_clients"
void enregistreClients(Clients* pt_clients)
{
    // Initialisation des variables et attend une entrée pour le nom du client
    Mot nom_client;
    get_id(nom_client);

    // Message d'erreur si le nombre de clients est supérieur aux nombres de clients
autorisés
    assert(pt_clients->nb_clients <= MAX_CLIENTS);

    // Si c'est le premier client
    if (pt_clients->nb_clients == 0)
    {
        // Copie le nom_client dans le tableau des clients
        strcpy(pt_clients->tab_clients[pt_clients->nb_clients], nom_client);

        // Incrémente de UN le nombre de client
        pt_clients->nb_clients++;
    }
    else
    {
        /* Parcours tous les clients pour comparé si le nom du client est pareil
                    que le nom du client dans le tableau des clients            */

        unsigned int x = 0;
        // Parcours le nombre de clients
        for (x = 0; x < pt_clients->nb_clients; x++)
        {
            // Compare le nom du client saisi avec le nom du client dans le tableau
des clients
            if (strcmp(pt_clients->tab_clients[x], nom_client) == 0)
            {
                printf("Le client %s est deja declare, veuillez essayer avec un autre
nom. Merci\n", nom_client);

                // Arrête toute la fonction void enregistreClients
                return;
            }
        }

        // Copie le nom du client dans le tableau des clients
        strcpy(pt_clients->tab_clients[pt_clients->nb_clients], nom_client);
```

```
        // Incrémente de UN le nombre de client
        pt_clients->nb_clients++;
    }
}

// Lorsque l'on écrit "client" dans la console
// Brief : Afficher la liste des produits que le client a commandé
void afficheClient(Clients* pt_clients, Commandes* pt_commande)
{
    // Initialise la variable et attend une entrée pour le nom du client
    Mot nom_client;
    get_id(nom_client);

    // Si le nom du client est égale à 'tous'
    if (strcmp(nom_client, "tous") == 0)
    {
        /* Affiche tous les commandes de tout les clients
           Parcours le nombre de clients                    */
        unsigned int x;

        // Parcours le nombre de clients
        for (x = 0; x < pt_clients->nb_clients; x++)
        {
            // Remplace %s par le nom du client dans le tableau des clients
            printf("le client %s a commande : ", pt_clients->tab_clients[x]);

            // Initialise les compteurs
            unsigned int y;
            unsigned int existe = 0;

            // Parcours le nombre de commande
            for (y = 0; y < pt_commande->nb_commandes; y++)
            {
                // Vérifie si à l'index 'x' c'est l'index du client pour la commande
'y'
                if (x == pt_commande->tab_commandes[y].idx_client)
                {
                    // Ajoute la virgule si le nom de la 1ère commande est déjà
affiché
                    if (existe != 0)
                    {
                        printf(", ");
                    }
                    printf("%s", pt_commande->tab_commandes[y].nom);
                    existe++;
                }
            }
            printf("\n");

        }
        // Le compilateur arrêtera de lire la suite grâce au return
        return;
    }
    // Boucle servant à afficher tout les noms de clients dans le tableau client
    // Parcours le nombre de clients
    unsigned int x;
    for (x = 0; x < pt_clients->nb_clients; x++)
    {
        // Vérifie si le nom du client existe dans le tableau des clients
        if (strcmp(nom_client, pt_clients->tab_clients[x]) == 0)
        {
```

```
            // Remplace %s par le nom du client dans le tableau des clients
            printf("le client %s a commande : ", pt_clients->tab_clients[x]);

            // Initialise les compteurs
            unsigned int y;
            unsigned int virgule = 0;

            // Parcours le nombre de commande
            for (y = 0; y < pt_commande->nb_commandes; y++)
            {
                if (x == pt_commande->tab_commandes[y].idx_client)
                {
                    // Ajoute la virgule si le nom de la 1ère commande est déjà
affiché
                    if (virgule != 0)
                    {
                        printf(", ");
                    }
                    printf("%s", pt_commande->tab_commandes[y].nom);
                    virgule++;
                }
            }
            printf("\n");
        }
    }
}
// Lorsque l'on écrit "commande" dans la console
// Brief : Enregistrer la liste de la commande du client
void enregistreNewCommande(Commandes* pt_liste_commandes, Clients* pt_client)
{
    // Initialise les variables et Demande de saisir le nom produit et le nom du
client
    Mot nom_produit;
    get_id(nom_produit);
    Mot nom_client;
    get_id(nom_client);

    // Message d'erreur si le nombre de commandes est supérieur aux nombres de
commandes autorisés
    assert(pt_liste_commandes->nb_commandes <= MAX_COMMANDES);

    // Initialise les compteurs x et y
    unsigned int x, y;

    // Parcours le nombre de clients
    for (x = 0; x < pt_client->nb_clients; x++)
    {
        // Compare si le nom du client est pareil que celui dans la liste des clients
        if (strcmp(nom_client, pt_client->tab_clients[x]) == 0)
        {
            // Passe à l'index du client le nombre 'y' pour savoir à qui la commande
est destinée

pt_liste_commandes->tab_commandes[pt_liste_commandes->nb_commandes].idx_client = x;

            // Copie le nom du produit dans la liste des commandes

strcpy(pt_liste_commandes->tab_commandes[pt_liste_commandes->nb_commandes].nom,
nom_produit);

            // Incrémente de UN le nombre de commande
            pt_liste_commandes->nb_commandes++;
```

```
        }
    }

    // Parcours le nombre maximum de specialités
    for (y = 0; y < MAX_SPECIALITES; y++)
    {
        // Initialise les heures effectuées et les heures requises des taches par
spécialité du client 'x' à 0

pt_liste_commandes->tab_commandes[pt_liste_commandes->nb_commandes].taches_par_special
ite[y].nb_heures_effectuees = 0;

pt_liste_commandes->tab_commandes[pt_liste_commandes->nb_commandes].taches_par_special
ite[y].nb_heures_requises = 0;
    }
}

// Lorsque l'on écrit "tache" dans la console
// Brief : Enregistrer la tâche d'une commande qui précise la spécialité et le nb
heure nécessaire
void enregistreTache(Commandes* pt_commande, Specialites* pt_tab_specialites,
Travailleurs* pt_travailleur)
{
    // Initialise les variables et Demande de saisir le nom de la commande
    // Et le nom de la spécialité et le nombre d'heure nécessaire pour réaliser la
tâche
    Mot nom_commande;
    get_id(nom_commande);
    Mot nom_specialite;
    get_id(nom_specialite);
    int heure_nécessaire = get_int();

    // Initialise les compteurs
    unsigned int x, i, j;

    // Initialise à 0 les heures total et l'index du travailleur
    unsigned int heure_total = 0;
    unsigned int idx_travailleur = 0;

    // Initialise la condition de départ à faux (aucune heure totale n'a encore été
définie)
    Booleen heuretotalExiste = FAUX;

    // Initialise les nombres heures requises et effectuées
    Tache tache;
    tache.nb_heures_requises = heure_nécessaire;
    tache.nb_heures_effectuees = 0;

    // Parcours le nombre de commandes
    for (x = 0; x < pt_commande->nb_commandes; x++)
    {
        // Vérifie si cette commande est celle qui est demandée
        if (strcmp(nom_commande, pt_commande->tab_commandes[x].nom) == 0)
        {
            break;
        }
    }

    // Parcours le nombre de spécialités
    for (j = 0; j < pt_tab_specialites->nb_specialites; j++)
    {
        // Vérifie si cette commande est celle qui est demandée
```

```
        if (strcmp(nom_specialite, pt_tab_specialites->tab_specialites[j].nom) == 0)
        {
            break;
        }
    }

    // Parcours le nombre de travailleurs pour déterminer la position du travailleur
    for (i = 0; i < pt_travailleur->nb_travailleurs; i++)
    {
        // Vérifie si la compétence du travailleur existe
        if (pt_travailleur->tab_travailleurs[i].tags_competences[j] == VRAI)
        {
            // Condition pour initier l'heure total sinon il y a rien au début
            if (heuretotalExiste == FAUX)
            {
                heure_total = pt_travailleur->tab_travailleurs[i].charge;
                idx_travailleur = i;
                heuretotalExiste = VRAI;
            }
            else
            {
                if (heure_total > pt_travailleur->tab_travailleurs[i].charge)
                {
                    heure_total = pt_travailleur->tab_travailleurs[i].charge;
                    idx_travailleur = i;
                }
            }
        }
    }
    // Affecte l'index du travailleur à l'index travailleur dans tache
    tache.idx_travailleur = idx_travailleur;

    // Initialise les heures requises et effectuée à la commande 'x' de la tache 'j'
    pt_commande->tab_commandes[x].taches_par_specialite[j] = tache;

    // Ajoute à la charge du travailleur en question le nombre d'heure requise
    pt_travailleur->tab_travailleurs[idx_travailleur].charge +=
tache.nb_heures_requises;
}
// Lorsque l'on écrit "progression" dans la console
// Brief : Enregistre la progression d'une commande (actualise l'état de la tâche)
void enregistreProgressionTravail(Commandes* pt_commandes, Specialites*
pt_specialites, Travailleurs* pt_travailleur)
{
    // Initialisation des variables et En attente de trois entrées
    Mot nom_commande;
    get_id(nom_commande);
    Mot nom_specialite;
    get_id(nom_specialite);
    int NbHeure_avancé = get_int();

    unsigned int x, y;
    // Parcours le nombre de commande dans la liste des commandes
    for (x = 0; x < pt_commandes->nb_commandes; x++)
    {
        if (strcmp(nom_commande, pt_commandes->tab_commandes[x].nom) == 0)
        {
            // Parcours le nombre de spécialités dans le tableau des spécialités
proposés par l'entreprise
            for (y = 0; y < pt_specialites->nb_specialites; y++)
            {
```

```
                if (strcmp(nom_specialite, pt_specialites->tab_specialites[y].nom) ==
0)
                {

pt_commandes->tab_commandes[x].taches_par_specialite[y].nb_heures_effectuees +=
NbHeure_avancé;

pt_travailleur->tab_travailleurs[pt_commandes->tab_commandes[x].taches_par_specialite[
y].idx_travailleur].charge -= NbHeure_avancé;
                }
            }
        }
    }

}

// Lorsque l'on écrit "supervision" dans la console
// Brief : Affiche l'avancement d'une commande
void avancementCommande(Commandes* pt_commande, Specialites* pt_specialite)
{

    // Initialisation des compteurs x et z
    unsigned int x, z;

    // Parcours toutes les commandes
    for (x = 0; x < pt_commande->nb_commandes; x++)
    {
        // Vérifie si la commande est celle demandé
        if (pt_commande->nb_commandes != 0)
        {
            printf("etat des taches pour %s : ", pt_commande->tab_commandes[x].nom);

            // Initialise le compteur
            unsigned int virgule = 0;

            // Parcours toutes les spécialités qui sont proposés par l'entreprise
            for (z = 0; z < pt_specialite->nb_specialites; z++)
            {
                /*      Rappel "nom_spe : nb_heure_effectuee / nb_heure_requise"
*/

                // Si le nb_heure_effectue est égale à 0 et que le nb_heure_requise
est différent de 0
                if
(pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_effectuees == 0
                    &&
pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_requises != 0)
                {
                    // Si l'avancement de la première commande est déjà faite alors on
ajoute la virgule
                    if (virgule > 0)
                    {
                        printf(", ");
                    }
                    printf("%s:0/%d", pt_specialite->tab_specialites[z].nom,

pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_requises);
                    virgule++;
                }
                // Si le nb_heure_effectue est strictement supérieur à 0 et que le
nb_heure_requise est différent de 0
```

```
                else if
(pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_effectuees > 0
                    &&
pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_requises != 0)
                {
                    // Si l'avancement de la première commande est déjà faite alors on
ajoute la virgule
                    if (virgule > 0)
                    {
                        printf(", ");
                    }
                    printf("%s:%d/%d", pt_specialite->tab_specialites[z].nom,

pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_effectuees,

pt_commande->tab_commandes[x].taches_par_specialite[z].nb_heures_requises);
                    virgule++;
                }
            }
            printf("\n");
        }
    }
}
// Lorsque l'on écrit "charge" dans la console
void afficheTacheRestante()
{
    Mot nom_travailleur;
    get_id(nom_travailleur);
    printf(MSG_CHARGE, nom_travailleur);
}

// Lorsque l'on écrit "interruption" dans la console
void Quitter()
{
    printf(MSG_INTERRUPTION);
}


// Boucle principale ----------------------------------------------------------
int main(int argc, char* argv[])
{
    // Initialisation des variables et stocke des valeurs
    Specialites Couts_Specialite = { " " , 0 };

    Travailleurs Embauche_travailleurs = { " " , 0 };

    Clients Mes_Clients = { " " , 0 };

    Commandes Les_Commandes = { " " , 0 };

    if (argc >= 2 && strcmp("echo", argv[1]) == 0)
    {
        EchoActif = VRAI;
    }

    // Buffer de type Mot et ne peut contenir au max 35 caratères
    Mot buffer;

    // Une boucle à l'infini
    while (VRAI)
    {
        // Attente d'une entrée dans la console ------------
```

```
        get_id(buffer);

        // Appel de la fonction enregistreSpecialites lorsque l'on écrit
'developpe ...' dans la console
        if (strcmp(buffer, "developpe") == 0)
        {
            enregistreSpecialites(&Couts_Specialite);
            continue;
        }

        // Appel de la fonction afficheSpecialites lorsque l'on écrit 'specialite ...'
dans la console
        if (strcmp(buffer, "specialites") == 0)
        {
            afficheSpecialites(&Couts_Specialite);
            continue;
        }

        // Appel de la fonction enregistreEmbauche lorsque l'on écrit 'embauche ...'
dans la console
        if (strcmp(buffer, "embauche") == 0)
        {
            enregistreEmbauche(&Embauche_travailleurs, &Couts_Specialite);
            continue;
        }

        // Appel de la fonction afficheTravailleurs lorsque l'on écrit
'travailleur ...' dans la console
        if (strcmp(buffer, "travailleurs") == 0)
        {
            afficheTravailleurs(&Embauche_travailleurs, &Couts_Specialite);
            continue;
        }

        // Appel de la fonction enregistreClients lorsque l'on écrit 'demarche ...'
dans la console
        if (strcmp(buffer, "demarche") == 0)
        {
            enregistreClients(&Mes_Clients);
            continue;
        }

        // Appel de la fonction afficheClient lorsque l'on écrit 'client ...' dans la
console
        if (strcmp(buffer, "client") == 0)
        {
            afficheClient(&Mes_Clients, &Les_Commandes);
            continue;
        }

        // Appel de la fonction enregistreNewCommande lorsque l'on écrit
'commande ...' dans la console
        if (strcmp(buffer, "commande") == 0)
        {
            enregistreNewCommande(&Les_Commandes, &Mes_Clients);
            continue;
        }

        // Appel de la fonction enregistreTache lorsque l'on écrit 'tache ...' dans la
console
        if (strcmp(buffer, "tache") == 0)
        {
```

```
            enregistreTache(&Les_Commandes, &Couts_Specialite,
&Embauche_travailleurs);
            continue;
        }

        // Appel de la fonction enregistreProgressionTravail lorsque l'on écrit
'progression ...' dans la console
        if (strcmp(buffer, "progression") == 0)
        {
            enregistreProgressionTravail(&Les_Commandes, &Couts_Specialite,
&Embauche_travailleurs);
            continue;
        }

        // Appel de la fonction avancementCommande lorsque l'on écrit
'supervision ...' dans la console
        if (strcmp(buffer, "supervision") == 0)
        {
            avancementCommande(&Les_Commandes, &Couts_Specialite);
            continue;
        }

        // Appel de la fonction afficheTacheRestante lorsque l'on écrit 'charge ...'
dans la console
        if (strcmp(buffer, "charge") == 0)
        {
            afficheTacheRestante();
            continue;
        }

        // Lorsque l'on écrit 'passe' dans la console
        if (strcmp(buffer, "passe") == 0)
        {
            continue;
        }

        // Appel de la fonction Quitter lorsque l'on écrit 'interruption ...' dans la
console
        if (strcmp(buffer, "interruption") == 0)
        {
            Quitter();
            break;
        }

        // Un message d'erreur comme quoi notre entrée n'est pas reconnu
        printf("!!! instruction inconnue >%s< !!!\n", buffer);
    }
    return 0;
}
```