

Guillaume de La Grandiere, Inès Kettani

# Initiation à l’Algorithmique et à la Programmation

Dossier de développement

---

# Table des matières

Présentation.....	2
Introduction :.....	2
Objectif du projet :.....	2
Entrées et sorties :.....	2
Organisation du développement.....	4
Sprints :.....	4
Tests des sprints : .....	4
Bilan du projet .....	6
Difficultés rencontrées :.....	6
Conclusion :.....	6
Annexe 1 : validation entrée/sortie.....	7
Sprint 1 :.....	7
Sprint 2 :.....	7
Sprint 3 :.....	8
Sprint 4 :.....	8
Sprint 5 :.....	8
Sprint 6 :.....	9
Annexe 2 : code source sprints.....	11
Sprint 1 :.....	11
Sprint 2 :.....	15
Sprint 3 :.....	22
Sprint 4 :.....	29
Sprint 5 :.....	37
Sprint 6 :.....	47

# Présentation

## Introduction :

Les progrès technologiques ont permis un essor du traitement de l'information sous forme numérique. Cette méthode permet de réduire les erreurs potentielles dû à l'imprécision d'un traitement physique sur papier. Quand une entreprise atteint une taille conséquente, elle rencontre la nécessité de créer un système de gestion informatisé. Le but de ce projet sera donc de créer un système de gestion pour cette entreprise.

## Objectif du projet :

Le but de ce projet est de programmer un interpréteur de commandes capables de gérer une base de données sous forme de tableaux. Il doit être capable de gérer toutes les étapes de la complétion d'une commande : enregistrer un client et ses commandes, gérer les travailleurs en fonction de leurs compétences et répartir de manières optimisée les tâches à effectuer ainsi que facturer les commandes. En finissant le sprint 6 release nous avons un programme complet qui peut enregistrer des clients, des spécialités, des commandes ainsi que des travailleurs. Il peut assigner des tâches aux commandes dans différentes spécialités et les assigner aux travailleurs en fonctions de leur quantité de travail à effectuer. Ces tâches peuvent ensuite être avancées, réassignées et terminées. Enfin, les commandes sont facturées une fois terminées, et, quand toutes les commandes sont facturées, le programme doit afficher la liste de tous les clients ainsi que la facture totale qu'ils ont à payer.

## Entrées et sorties :

Le projet comporte 12 entrées, chacune correspondant à une fonction du programme : traite\_demarche, traite\_embauche, traite\_commande, traite\_supervision, traite\_client, traite\_travailleurs, traite\_specialites, traite\_tache, traite\_progression, traite\_charge, traite\_passe et traite\_interruption.

Toutes ces fonctions sont appelées dans le main via une commande de l'opérateur grâce à la fonction `get_id` qui permet de récupérer une chaîne de caractère. Celle-ci est comparée aux différentes commandes possibles grâce à la fonction `strcmp()` qui permet de comparer deux chaînes de caractères, importée de la bibliothèque `<string.h>`.

Les sorties sont effectuées via des printf qui se trouvent dans différentes fonctions :

- traite\_supervision qui affiche l'état des tâches pour chaque commande.
- affiche\_client qui affiche un client et ses différentes commandes.
- affiche\_travailleur qui affiche les travailleur compétents pour une spécialité.
- affiche\_specialite qui affiche une spécialité et son coût horaire.
- traite\_charge qui affiche les taches assignées à un travailleur ainsi que le nombres d'heures de travail restantes pour celles-ci.
- traite\_facturation qui affiche la facture pour une commande avec la facturation de chaque tâche.
- traite\_fin qui affiche la facture totale de chaque client et clos le programme.
- traite\_interruption qui affiche le message d'interruption et clos le programme.

# Organisation du développement

## Sprints :

Le projet est organisé en sprints, permettant une évolution du programme par palliés. Ils sont catégorisés comme suit :

- Sprint 1 : toutes fonctions provoquent uniquement un affichage en fonctions des paramètres d'entrée
- Sprint 2 : le programme doit pouvoir enregistrer les travailleurs, les spécialités ainsi que les clients. Il doit pouvoir les afficher sur demande de l'opérateur
- Sprint 3 : Le programme doit pouvoir enregistrer des commandes et créer des tâches associées à celles-ci. Il doit aussi pouvoir afficher les tâches et leur avancement. Il doit aussi pouvoir avancer et compléter les tâches
- Sprint 4 : Le programme doit pouvoir assigner automatiquement les tâches au travailleur ayant le moins de charge de travail.
- Sprint 5 : Le programme doit pouvoir facturer une commande terminée, et, une fois toutes les commandes facturées, afficher la facturation finale pour chaque client et clore le programme
- Sprint 6 : Le programme doit pouvoir réassigner une tâche non complétée à la demande de l'utilisateur.

## Tests des sprints :

Pour valider les tests, un fichier contenant des entrées doit être lu et exécuté par le programme et donner en sortie un fichier identique à celui fourni. Le fichier « out » peut être obtenu en passant le fichier « in » en entrée lors d'une exécution du programme dans l'invite de commande comme ceci :

```
Microsoft Windows [version 10.0.18363.1139]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Windows\System32>f:

F:>\CD F:\Famille\Documents\Guillaume\DUT\Projet IAP\Sprints\Debug

F:\Famille\Documents\Guillaume\DUT\Projet IAP\Sprints\Debug>Sprint1.exe <in1.release.txt>out1_release.txt

F:\Famille\Documents\Guillaume\DUT\Projet IAP\Sprints\Debug>
```

Le fichier « out » ainsi obtenu peut être alors comparé au fichier « out » fourni pour vérifier la validité du code. Nous avons utilisé le site [diffchecker.com](https://diffchecker.com) pour effectuer la vérification.

The two files are identical

Editor

Clear
Save Diff
Share

Original Text
Changed Text

```

realisees
29 ## pour la commande "superProduit", pour la specialite "reseau" : "3" heures de plus ont ete
realisees
30 ## une reallocation est requise
31 ## pour la commande "produiT0p", pour la specialite "graphisme" : "2" heures de plus ont ete
realisees
32 ## une reallocation est requise
33 ## pour la commande "superProduit", pour la specialite "reseau" : "1" heures de plus ont ete
realisees
34 ## pour la commande "produiT0p", pour la specialite "graphisme" : "1" heures de plus ont ete
realisees
35 ## consultation de l'avancement des commandes
36 ## consultation de la charge de travail de "Toto"
37 ## consultation de la charge de travail de "Titi"
38 ## fin de programme
39

```

# Bilan du projet

## Difficultés rencontrées :

Les difficultés rencontrées ont été progressives : sur le premier sprint l'écriture était uniquement répétitive. Sur le second sprint aucune difficulté particulière n'a été rencontrée. A partir du 3-ème sprint des erreurs sont apparus dû à la complexité des structures du programme. Pour le quatrième sprint les structures ne sont plus fournies et le plus compliqué a été de savoir comment modifier de manière optimisée les structures pour prendre en compte les affectations de tâches et ne pas surcharger la mémoire.

Par la suite le seul réel problème a été l'optimisation du programme une fois le sprint 6 release terminé.

Dans sa version actuelle, le programme n'est pas complètement optimisé et certaines commandes peuvent paraître brouillon. Nous avons fait notre maximum mais le programme n'est pas le plus réduis possible.

## Conclusion :

Ce projet présentait une certaine difficulté de part le fait qu'il s'agissait du premier projet que nous avions à réaliser en C. Cependant la division en sprints permet un travail échelonné et une difficulté progressive.

En supplément les sprints ne donnent pas de consigne précises ce qui nécessite de chercher à travers les « in » et les « out » fournis pour connaître une partie des commandes et des instructions à implémenter.

Ce projet nous a aussi permis d'apprendre à coder en équipe sur un code et à relire le travail des autres pour corriger les erreurs d'inattention ainsi que celle de codage. Cela nous a donné un aperçu du travail dans le monde professionnel.

# Annexe 1 : validation entrée/sortie

## Sprint 1 :

The two files are identical

Editor

Clear
Save Diff
Share

Original Text	Changed Text
<pre> realisees 29 ## pour la commande "superProduit", pour la specialite "reseau" : "3" heures de plus ont ete realisees 30 ## une reallocation est requise 31 ## pour la commande "produitTop", pour la specialite "graphisme" : "2" heures de plus ont ete realisees 28 32 ## une reallocation est requise 33 ## pour la commande "superProduit", pour la specialite "reseau" : "1" heures de plus ont ete realisees 29 34 ## pour la commande "produitTop", pour la specialite "graphisme" : "1" heures de plus ont ete realisees 31 35 ## consultation de l'avancement des commandes 32 36 ## consultation de la charge de travail de "Toto" 37 ## consultation de la charge de travail de "Titi" 33 38 ## fin de programme 39 34 35 ## consultation de la charge de travail de "Toto" 36 ## consultation de la charge de travail de "Titi" 37 ## fin de programme 38 </pre>	<pre> realisees 29 ## pour la commande "superProduit", pour la specialite "reseau" : "3" heures de plus ont ete realisees 30 ## une reallocation est requise 31 ## pour la commande "produitTop", pour la specialite "graphisme" : "2" heures de plus ont ete realisees 32 ## une reallocation est requise 33 ## pour la commande "superProduit", pour la specialite "reseau" : "1" heures de plus ont ete realisees 34 ## pour la commande "produitTop", pour la specialite "graphisme" : "1" heures de plus ont ete realisees 35 ## consultation de l'avancement des commandes 36 ## consultation de la charge de travail de "Toto" 37 ## consultation de la charge de travail de "Titi" 38 ## fin de programme 39 35 ## consultation de la charge de travail de "Toto" 36 ## consultation de la charge de travail de "Titi" 37 ## fin de programme 38 </pre>

## Sprint 2 :

## Sprint 3 :

The two files are identical

Editor
↔
Clear
Save Diff

Original Text	Changed Text
<pre> 32 état des taches pour produiTTop : reseau:0/28, graphisme:21/67 33 état des taches pour superProduit : reseau:5/45, graphisme:7/32 34 état des taches pour produiTTop : reseau:16/28, graphisme:21/67 35 état des taches pour superProduit : reseau:8/45, graphisme:7/32 36 état des taches pour produiTTop : reseau:16/28, graphisme:21/67 37 état des taches pour superProduit : reseau:8/45, graphisme:7/32 38 état des taches pour produiTTop : reseau:16/28, graphisme:23/67 39 état des taches pour superProduit : reseau:9/45, graphisme:7/32 40 état des taches pour produiTTop : reseau:16/28, graphisme:23/67 41 état des taches pour superProduit : reseau:9/45, graphisme:7/32 42 état des taches pour produiTTop : reseau:16/28, graphisme:24/67 43 ## consultation de la charge de travail de "Toto" 44 ## consultation de la charge de travail de "Titi" 45 ## fin de programme 46 </pre>	<pre> 32 état des taches pour produiTTop : reseau:0/28, graphisme:21/67 33 état des taches pour superProduit : reseau:5/45, graphisme:7/32 34 état des taches pour produiTTop : reseau:16/28, graphisme:21/67 35 état des taches pour superProduit : reseau:8/45, graphisme:7/32 36 état des taches pour produiTTop : reseau:16/28, graphisme:21/67 37 état des taches pour superProduit : reseau:8/45, graphisme:7/32 38 état des taches pour produiTTop : reseau:16/28, graphisme:23/67 39 état des taches pour superProduit : reseau:9/45, graphisme:7/32 40 état des taches pour produiTTop : reseau:16/28, graphisme:23/67 41 état des taches pour superProduit : reseau:9/45, graphisme:7/32 42 état des taches pour produiTTop : reseau:16/28, graphisme:24/67 43 ## consultation de la charge de travail de "Toto" 44 ## consultation de la charge de travail de "Titi" 45 ## fin de programme 46 </pre>

## Sprint 4 :

The two files are identical

Editor
↔
Clear
Save Diff

Original Text	Changed Text
<pre> produiTHyper/graphisme/15heure(s) 24 état des taches pour superProduit : reseau:0/45, graphisme:0/32 25 état des taches pour produiTTop : reseau:0/28, graphisme:0/67 26 état des taches pour megaProduit : graphisme:0/100 27 état des taches pour produiTHyper : graphisme:0/15 28 état des taches pour superProduit : reseau:9/45, graphisme:7/32 29 état des taches pour produiTTop : reseau:16/28, graphisme:24/67 30 état des taches pour megaProduit : graphisme:0/100 31 état des taches pour produiTHyper : graphisme:0/15 32 charge de travail pour Toto : superProduit/reseau/36heure(s), produiTTop/reseau/12heure(s),     megaProduit/graphisme/100heure(s) 33 charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTTop/graphisme/43heure(s),     produiTHyper/graphisme/15heure(s) 34 ## fin de programme 35 </pre>	<pre> produiTHyper/graphisme/15heure(s) 24 état des taches pour superProduit : reseau:0/45, graphisme:0/32 25 état des taches pour produiTTop : reseau:0/28, graphisme:0/67 26 état des taches pour megaProduit : graphisme:0/100 27 état des taches pour produiTHyper : graphisme:0/15 28 état des taches pour superProduit : reseau:9/45, graphisme:7/32 29 état des taches pour produiTTop : reseau:16/28, graphisme:24/67 30 état des taches pour megaProduit : graphisme:0/100 31 état des taches pour produiTHyper : graphisme:0/15 32 charge de travail pour Toto : superProduit/reseau/36heure(s), produiTTop/reseau/12heure(s),     megaProduit/graphisme/100heure(s) 33 charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTTop/graphisme/43heure(s),     produiTHyper/graphisme/15heure(s) 34 ## fin de programme 35 </pre>

## Sprint 5 :

The two files are identical

Editor
↔
Clear
Save Diff

Original Text	Changed Text
<pre> 29 état des taches pour produiTTop : reseau:16/28, graphisme:24/67 30 état des taches pour megaProduit : graphisme:0/100 31 état des taches pour produiTHyper : graphisme:0/15 32 charge de travail pour Toto : superProduit/reseau/36heure(s), produiTTop/reseau/12heure(s),     megaProduit/graphisme/100heure(s) 33 charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTTop/graphisme/43heure(s),     produiTHyper/graphisme/15heure(s) 34 facturation superProduit : reseau:810, graphisme:416 35 facturation produiTTop : reseau:504, graphisme:871 36 facturation megaProduit : graphisme:1300 37 charge de travail pour Toto : 38 charge de travail pour Titi : produiTHyper/graphisme/15heure(s) 39 facturation produiTHyper : graphisme:195 40 facturations : Roger:2526, Rodgio:1570 41 </pre>	<pre> 29 état des taches pour produiTTop : reseau:16/28, graphisme:24/67 30 état des taches pour megaProduit : graphisme:0/100 31 état des taches pour produiTHyper : graphisme:0/15 32 charge de travail pour Toto : superProduit/reseau/36heure(s), produiTTop/reseau/12heure(s),     megaProduit/graphisme/100heure(s) 33 charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTTop/graphisme/43heure(s),     produiTHyper/graphisme/15heure(s) 34 facturation superProduit : reseau:810, graphisme:416 35 facturation produiTTop : reseau:504, graphisme:871 36 facturation megaProduit : graphisme:1300 37 charge de travail pour Toto : 38 charge de travail pour Titi : produiTHyper/graphisme/15heure(s) 39 facturation produiTHyper : graphisme:195 40 facturations : Roger:2526, Rodgio:1570 41 </pre>

## Sprint 6 :

The two files are identical

Editor ↗

Original Text

```

52 charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTophy/produit/44heure(s),
produiTophy/produit/15heure(s)
53 etat des taches pour superProduit : reseau:45/45, graphisme:7/32
54 etat des taches pour produiTophy : reseau:28/28, graphisme:24/67
55 etat des taches pour megaProduit : graphisme:100/100
56 etat des taches pour produiTophy : graphisme:0/15
57 charge de travail pour Toto : produiTophy/produit/43heure(s)
58 charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTophy/produit/15heure(s)
59 facturation superProduit : reseau:810, graphisme:416
60 facturation produiTophy : reseau:504, graphisme:871
61 charge de travail pour Toto :
62 charge de travail pour Titi : produiTophy/produit/15heure(s)
63 facturation produiTophy : graphisme:195
64 facturations : Roger:2526, Rodgio:1570
65

```

Changed Text

```

52 charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTophy/produit/44heure(s),
produiTophy/produit/15heure(s)
53 etat des taches pour superProduit : reseau:45/45, graphisme:7/32
54 etat des taches pour produiTophy : reseau:28/28, graphisme:24/67
55 etat des taches pour megaProduit : graphisme:100/100
56 etat des taches pour produiTophy : graphisme:0/15
57 charge de travail pour Toto : produiTophy/produit/43heure(s)
58 charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTophy/produit/15heure(s)
59 facturation superProduit : reseau:810, graphisme:416
60 facturation produiTophy : reseau:504, graphisme:871
61 charge de travail pour Toto :
62 charge de travail pour Titi : produiTophy/produit/15heure(s)
63 facturation produiTophy : graphisme:195
64 facturations : Roger:2526, Rodgio:1570
65

```

« out » sprint 6 release :

```

specialites traitees : reseau/18, graphisme/13
la specialite reseau peut etre prise en charge par : Toto
la specialite graphisme peut etre prise en charge par : Toto, Titi
le client Roger a commande :
le client Rodgio a commande :
le client Roger a commande : superProduit, megaProduit
le client Rodgio a commande : produiTophy, produiTophy
etat des taches pour superProduit :
etat des taches pour produiTophy :
etat des taches pour megaProduit :
etat des taches pour produiTophy :
charge de travail pour Toto : superProduit/reseau/45heure(s)
charge de travail pour Titi :
charge de travail pour Toto : superProduit/reseau/45heure(s)
charge de travail pour Titi : superProduit/graphisme/32heure(s)
charge de travail pour Toto : superProduit/reseau/45heure(s)
charge de travail pour Titi : superProduit/graphisme/32heure(s), produiTophy/produit/67heure(s)
charge de travail pour Toto : superProduit/reseau/45heure(s), megaProduit/graphisme/100heure(s)
charge de travail pour Titi : superProduit/graphisme/32heure(s), produiTophy/produit/67heure(s)
charge de travail pour Toto : superProduit/reseau/45heure(s), megaProduit/graphisme/100heure(s)
charge de travail pour Titi : superProduit/graphisme/32heure(s), produiTophy/produit/67heure(s), produiTophy/produit/15heure(s)
charge de travail pour Toto : superProduit/reseau/45heure(s), produiTophy/produit/28heure(s), megaProduit/graphisme/100heure(s)
charge de travail pour Titi : superProduit/graphisme/32heure(s), produiTophy/produit/67heure(s), produiTophy/produit/15heure(s)
etat des taches pour superProduit : reseau:0/45, graphisme:0/32
etat des taches pour produiTophy : reseau:0/28, graphisme:0/67
etat des taches pour megaProduit : graphisme:0/100
etat des taches pour produiTophy : graphisme:0/15
etat des taches pour superProduit : reseau:5/45, graphisme:7/32
etat des taches pour produiTophy : reseau:16/28, graphisme:21/67
etat des taches pour megaProduit : graphisme:0/100
etat des taches pour produiTophy : graphisme:0/15
charge de travail pour Toto : superProduit/reseau/40heure(s), produiTophy/produit/12heure(s), megaProduit/graphisme/100heure(s)
charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTophy/produit/46heure(s), produiTophy/produit/15heure(s)
etat des taches pour superProduit : reseau:8/45, graphisme:7/32
etat des taches pour produiTophy : reseau:16/28, graphisme:23/67
etat des taches pour megaProduit : graphisme:0/100
etat des taches pour produiTophy : graphisme:0/15
charge de travail pour Toto : superProduit/reseau/37heure(s), produiTophy/produit/12heure(s), megaProduit/graphisme/100heure(s)
charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTophy/produit/46heure(s), produiTophy/produit/15heure(s)
etat des taches pour superProduit : reseau:8/45, graphisme:7/32
etat des taches pour produiTophy : reseau:16/28, graphisme:23/67
etat des taches pour megaProduit : graphisme:0/100
etat des taches pour produiTophy : graphisme:0/15
charge de travail pour Toto : superProduit/reseau/37heure(s), produiTophy/produit/12heure(s), megaProduit/graphisme/100heure(s)
charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTophy/produit/44heure(s), produiTophy/produit/15heure(s)

```

```
| facturation megaProduit : graphisme:1300
etat des taches pour superProduit : reseau:45/45, graphisme:7/32
etat des taches pour produiTTop : reseau:28/28, graphisme:23/67
etat des taches pour megaProduit : graphisme:100/100
etat des taches pour produiTHyper : graphisme:0/15
charge de travail pour Toto :
charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTTop/graphisme/44heure(s), produiTHyper/graphisme/15heure(s)
etat des taches pour superProduit : reseau:45/45, graphisme:7/32
etat des taches pour produiTTop : reseau:28/28, graphisme:24/67
etat des taches pour megaProduit : graphisme:100/100
etat des taches pour produiTHyper : graphisme:0/15
charge de travail pour Toto : produiTTop/graphisme/43heure(s)
charge de travail pour Titi : superProduit/graphisme/25heure(s), produiTHyper/graphisme/15heure(s)
facturation superProduit : reseau:810, graphisme:416
facturation produiTTop : reseau:504, graphisme:871
charge de travail pour Toto :
charge de travail pour Titi : produiTHyper/graphisme/15heure(s)
facturation produiTHyper : graphisme:195
| facturations : Roger:2526, Rodgio:1570
```

# Annexe 2 : code source

## sprints

### Sprint 1 :

```

1  #pragma warning(disable:4996) // on désactive l'avertissement sur les scanf
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  typedef enum { FAUX = 0, VRAI = 1 } Boolean; // on définit le type booléen qui n'est pas inclus dans le C
7  Boolean EchoActif = FAUX; // le mode de débogage est désactivé par défaut
8
9
10 // Messages émis par les instructions -----
11 // on définit ci dessous les messages à afficher selon les fonctions utilisées
12 #define MSG_DEVELOPPE "## nouvelle spécialité \"%s\" ; cout horaire \"%d\"\n"
13 #define MSG INTERRUPTION "## fin de programme\n"
14 #define MSG_EMBAUCHE "## nouveau travailleur \"%s\" compétent pour la spécialité \"%s\"\n"
15 #define MSG DEMARCHE "## nouveau client \"%s\"\n"
16 #define MSG TACHE "## la commande \"%s\" requiert la spécialité \"%s\" (nombre d'heures \"%d\")\n"
17 #define MSG PROGRESSION "## pour la commande \"%s\", pour la spécialité \"%s\" : \"%d\" heures de plus ont été réalisées\n"
18 #define MSG PASSE "## une réallocation est requise\n"
19 #define MSG SPECIALITES "## consultation des spécialités\n"
20 #define MSG TRAVAILLEURS "## consultation des travailleurs compétents pour la spécialité \"%s\"\n"
21 #define MSG CLIENT "## consultation des commandes effectuées par \"%s\"\n"
22 #define MSG SUPERVISION "## consultation de l'avancement des commandes\n"
23 #define MSG CHARGE "## consultation de la charge de travail de \"%s\"\n"
24 #define MSG COMMANDE "## nouvelle commande \"%s\" , par client \"%s\"\n"
25 #define MSG TOUS TRAVAILLEURS "## consultation des travailleurs compétents pour chaque spécialité\n"
26 #define MSG TOUS CLIENT "## consultation des commandes effectuées par chaque client\n"
27
28 // Lexèmes -----
29 #define LGMOT 35 // longueur d'un mot (35 caractères)
30 #define NBCHIFFREMAX 5 // taille d'un nombre (5 chiffres maximums)
31 typedef char Mot[LGMOT + 1]; // définition du type mot
32
33 void get_id(Mot id) { // fonction de récupération d'un mot
34     scanf("%s", id);
35     if (EchoActif) printf(">>echo %s\n", id); // si le mode débogage est activé on affiche le mot entre précédemment
36 }
37
38 int get_int() { // fonction de récupération d'un entier positif
39     char buffer[NBCHIFFREMAX + 1];
40     scanf("%s", buffer);
41     if (EchoActif) printf(">>echo %s\n", buffer); // si le mode débogage est activé on affiche le mot entre précédemment
42     return atoi(buffer);
43 }
44

```

```
45 // Instructions -----
46
47 //commande -----
48 void traite_commande() {
49     Mot nom_commande, nom_client;
50     get_id(nom_commande); // on recuperere aupres de l'utilisateur le nom de la commande
51     get_id(nom_client); // on recuperere aupres de l'utilisateur le nom du client
52     printf(MSG_COMMANDE, nom_commande, nom_client);
53 }
54
55 // Charge -----
56 void traite_charge() {
57     Mot nom_travailleur;
58     get_id(nom_travailleur); // recuperation du nom du travailleur aupres de l'utilisateur
59     printf(MSG_CHARGE, nom_travailleur);
60 }
61
62 // Supervision -----
63 void traite_supervision() {
64     printf(MSG_SUPERVISION);
65 }
66
67 // Client -----
68 void traite_client() {
69     Mot nom_client;
70     get_id(nom_client); // recuperation du nom du client aupres de l'utilisateur
71     if (strcmp(nom_client, "tous") == 0) {
72         printf(MSG_TOUS_CLIENT);
73     }
74     else { // sinon on affiche uniquement le client indique par l'utilisateur
75         printf(MSG_CLIENT, nom_client);
76     }
77 }
78 }
```

```

80 void traite_travailleurs() {
81   Mot nom_specialite;
82   get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
83   if (strcmp(nom_specialite, "tous") == 0) {
84     printf(MSG_TOUS_TRAVAILLEURS);
85   }
86   else { // sinon on affiche uniquement le client indique par l'utilisateur
87     printf(MSG_TRAVAILLEURS, nom_specialite);
88   }
89 }
90
91 // Specialités -----
92 void traite_specialites() {
93   printf(MSG_SPECIALITES);
94 }
95
96 // Progression -----
97 void traite_progression() {
98   Mot nom_commande, nom_specialite;
99   get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
100  get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
101  int nbr_heure = get_int(); // recuperation aupres de l'utilisateur du nombred'heure de la tache a effectuer
102  printf(MSG_PROGRESSION, nom_commande, nom_specialite, nbr_heure);
103 }
104
105 // Passe -----
106 void traite_passe() {
107   printf(MSG_PASSE);
108 }
109
110 // Tâches -----
111 void traite_tache() {
112   Mot nom_commande, nom_specialite;
113   get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
114   get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
115   int nbr_heure = get_int(); // recuperation aupres de l'utilisateur du nombred'heure de la tache a effectuer
116   printf(MSG_TACHE, nom_commande, nom_specialite, nbr_heure);
117 }
118
119 // Demarche -----
120 void traite_demarche() {
121   Mot nom_client;
122   get_id(nom_client); // recuperation aupres de l'utilisateur de la valeur de nom_client
123   printf(MSG_DEMARCHE, nom_client);
124 }
125
126 // Embauche -----
127 void traite_embauche() {
128   Mot nom_travailleur, nom_specialite;
129   get_id(nom_travailleur); // recuperation aupres de l'utilisateur de nom_travailleur
130   get_id(nom_specialite); // recuperation aupres de l'utilisateur de nom_specialite
131   printf(MSG_EMBAUCHE, nom_travailleur, nom_specialite);
132 }
133
134 // developpe -----
135 void traite_developpe() {
136   Mot nom_specialite;
137   get_id(nom_specialite); // recuperation aupres de l'utilisateur de nom_specialite
138   int cout_horaire = get_int(); // recuperation aupres de l'utilisateur du cout horaire de la specialite
139   printf(MSG DEVELOPPE, nom_specialite, cout_horaire);
140 }
141
142 // interruption -----
143 void traite_interruption() {
144   printf(MSG INTERRUPTION);
145 }
146

```

```

147 //Boucle principale -----
148 ~ int main(int argc, char* argv[]) {
149 ~   if (argc >= 2 && strcmp("echo", argv[1]) == 0) { // on verifie si le mot echo a ete mis en parametre du programme lors de l'execution en console
150 |     EchoActif = VRAI; // activation du mode debugage
151 }
152 // declaration des variables utilisees dans le main
153 Mot buffer;
154 Boolean progression = FAUX; // permet de verifier si la derniere valeur de "buffer" etait on non "progression"
155 ~ while (VRAI) {
156     get_id(buffer); // on demande la commande a entrer a l'utilisateur
157     if (progression==VRAI && strcmp(buffer, "passe") == 0) {
158         traite_passe(); // on execute la fonction traite_passe
159         progression = FAUX;
160         continue;
161     }
162     else {
163         progression = FAUX;
164     }
165     if (strcmp(buffer, "commande") == 0) {
166         traite_commande(); // on execute la fonction traite_commande
167         continue;
168     }
169     if (strcmp(buffer, "charge") == 0) {
170         traite_charge(); // on execute la fonction traite_charge
171         continue;
172     }
173     if (strcmp(buffer, "supervision") == 0) {
174         traite_supervision(); // on execute la fonction traite_supervision
175         continue;
176     }
177     if (strcmp(buffer, "client") == 0) {
178         traite_client(); // on execute la fonction traite_client
179         continue;
180     }
181     if (strcmp(buffer, "travailleurs") == 0) {
182         traite_travailleurs(); // on execute la fonction traite_travailleurs
183         continue;
184     }
185     if (strcmp(buffer, "specialites") == 0) {
186         traite_specialites(); // on execute la fonction traite_specialites
187         continue;
188     }
189     if (strcmp(buffer, "progression") == 0) {
190         traite_progression(); // on execute la fonction traite_progression
191         progression = VRAI;
192         continue;
193     }
194
195     if (strcmp(buffer, "tache") == 0) {
196         traite_tache(); // on execute la fonction traite_tache
197         continue;
198     }
199     if (strcmp(buffer, "demarche") == 0) {
200         traite_demarche(); // on execute la fonction traite_demarche
201         continue;
202     }
203     if (strcmp(buffer, "embauche") == 0) {
204         traite_embauche(); // on execute la fonction traite_embauche
205         continue;
206     }
207     if (strcmp(buffer, "developpe") == 0) {
208         traite_developpe(); // on execute la fonction traite_developpe
209         continue;
210     }
211     if (strcmp(buffer, "interruption") == 0) {
212         traite_interruption(); // on execute la fonction traite_interruption
213         break;
214     }
215     printf("!!! instruction inconnue >%s< !!!\n", buffer); // si aucune commande n'as ete reconnue on affiche ce message
216 }
217 return 0;
218 }
219

```

## Sprint 2 :

```

1  #pragma warning(disable:4996) // on désactive l'avertissement sur les scanf
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  typedef enum { FAUX = 0, VRAI = 1 } Boolean; // on définit le type booléen qui n'est pas inclus dans le C
7  Boolean EchoActif = FAUX; // le mode de débogage est désactivé par défaut
8
9  // Messages émis par les instructions -----
10 // on définit ci dessous les messages à afficher selon les fonctions utilisées
11 #define MSG_DEVELOPPE "## nouvelle spécialité \"%s\" ; cout horaire \"%d\"\n"
12 #define MSG INTERRUPTION "## fin de programme\n"
13 #define MSG_EMBAUCHE "## nouveau travailleur \"%s\" compétent pour la spécialité \"%s\"\n"
14 #define MSG DEMARCHE "## nouveau client \"%s\"\n"
15 #define MSG TACHE "## la commande \"%s\" requiert la spécialité \"%s\" (nombre d'heures \"%d\")\n"
16 #define MSG PROGRESSION "## pour la commande \"%s\", pour la spécialité \"%s\" : \"%d\" heures de plus ont été réalisées\n"
17 #define MSG PASSE "## une réallocation est requise\n"
18 #define MSG SPECIALITES "spécialités traitées : "
19 #define MSG SPECIALITES_ERREUR "spécialité inconnue\n"
20 #define MSG TRAVAILLEURS "la spécialité %s peut être prise en charge par : "
21 #define MSG CLIENT "le client %s a commandé : "
22 #define MSG CLIENT_ERREUR "client inconnu\n"
23 #define MSG CLIENT_ID_COMMANDE ""
24 #define MSG SUPERVISION "## consultation de l'avancement des commandes\n"
25 #define MSG CHARGE "## consultation de la charge de travail de \"%s\"\n"
26 #define MSG COMMANDE "## nouvelle commande \"%s\", par client \"%s\"\n"
27
28 // Lexèmes -----
29 #define LGMOT 35 // longueur d'un mot (35 caractères)
30 #define NBCHIFFREMAX 5 // taille d'un nombre (5 chiffres maximums)
31 typedef char Mot[LGMOT + 1]; // définition du type mot
32
33 void get_id(Mot id) { // fonction de récupération d'un mot
34     scanf("%s", id);
35     if (EchoActif) printf(">>echo %s\n", id); // si le mode débogage est activé on affiche le mot entre précédemment
36 }
37
38 int get_int() { // fonction de récupération d'un entier positif
39     char buffer[NBCHIFFREMAX + 1];
40     scanf("%s", buffer);
41     if (EchoActif) printf(">>echo %s\n", buffer); // si le mode débogage est activé on affiche le mot entre précédemment
42     return atoi(buffer);
43 }
44
45 // Données -----
46
47 // spécialités -----
48 #define MAX_SPECIALITES 10 // on définit le nombre maximum de spécialités
49 typedef struct { // une spécialité est composée d'un nom de type mot et d'un cout horaire entier
50     Mot nom;
51     int cout_horaire;
52 } Specialite;
53 typedef struct { // l'ensemble des spécialités est regroupé dans une structure contenant un tableau de spécialités ainsi qu'un entier contenant le nombre de spécialités
54     Specialite tab_specialites[MAX_SPECIALITES];
55     unsigned int nb_specialites;
56 } Specialites;
57
58 // travailleurs -----
59 #define MAX_TRAVAILLEURS 50 // on définit le nombre maximum de travailleurs
60 typedef struct { // un travailleur possède un nom de type mot, des compétences répertoriées dans un tableau de boolean et un nombres d'heures de travailles effectuées entier
61     Mot nom;
62     Boolean tags_competences[MAX_SPECIALITES]; // VRAI ou FAUX avec VRAI indiquant que le travailleur maîtrise la spécialité numéro n stockée à la case n du tableau de spécialités
63     unsigned int nb_heures_travail;
64 } Travailleur;
65
66 typedef struct { // l'ensemble des travailleurs est regroupé dans une structure contenant un tableau de travailleur ainsi qu'un entier contenant le nombre de travailleurs
67     Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
68     unsigned int nb_travailleurs;
69 } Travailleurs;
70
71 // client -----
72 #define MAX_CLIENTS 100 // on définit le nombre maximum de clients
73 typedef struct { // les clients sont répertoriés dans un tableau de mot contenant le nom de chaque client ainsi qu'un entier stockant le nombre de clients
74     Mot tab_clients[MAX_CLIENTS];
75     unsigned int nb_clients;
76 } Clients;

```

```

75 // Instructions -----
76
77 //commande -----
78 void traite_commande() {
79     Mot nom_commande, nom_client;
80     get_id(nom_commande); // on recuperes aupres de l'utilisateur le nom de la commande
81     get_id(nom_client); // on recuperes aupres de l'utilisateur le nom du client
82     printf(MSG_COMMANDE, nom_commande, nom_client);
83 }
84
85 // Charge -----
86 void traite_charge() {
87     Mot nom_travailleur;
88     get_id(nom_travailleur); // recuperation du nom du travailleur aupres de l'utilisateur
89     printf(MSG_CHARGE, nom_travailleur);
90 }
91
92 // Supervision -----
93 void traite_supervision() {
94     printf(MSG_SUPERVISION);
95 }
96
97 // Affichage Client -----
98 void affiche_clients(const Clients* rep_cli, int i){
99     printf(MSG_CLIENT, rep_cli->tab_clients[i]);
100    printf(MSG_CLIENT_ID_COMMANDE);
101    printf("\n");
102 }
103
104 // Client -----
105 void traite_client(const Clients* rep_cli) {
106     Mot nom_client;
107     get_id(nom_client); // recuperation du nom du client aupres de l'utilisateur
108     unsigned int i = 0;
109     Boolean suivant = FAUX;
110     if (strcmp(nom_client, "tous") == 0) { // si tous es entré on affiche tous les clients
111         while (i < rep_cli->nb_clients) { // parcours de l'ensemble des clients
112             affiche_clients(rep_cli, i); // affichage du client d'index i et de ses commandes
113             i++;
114         }
115     }
116     else { // sinon on affiche uniquement le client indiqué par l'utilisateur
117         while (i < rep_cli->nb_clients) { // parcourt de l'ensemble des clients
118             if (strcmp(nom_client, rep_cli->tab_clients[i]) == 0) {
119                 affiche_clients(rep_cli, i); // affichage du client d'index i et de ses commandes
120                 return; // interruption de la fonction
121             }
122             i++;
123         }
124     }
}

```

```

122     i++;
123   }
124   printf(MSG_CLIENT_ERREUR);
125 }
126 }
127
128 // Affiche travailleurs -----
129 void affiche_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav, int i) {
130   Boolean suivant = FAUX;
131   printf(MSG_TRAVAILLEURS, rep_spe->tab_specialites[i].nom);
132   for (unsigned int j = 0; j < rep_trav->nb_travailleurs; j++) { // parcours de l'ensemble des travailleurs
133     if (rep_trav->tab_travailleurs[j].tags_competences[i] == VRAI) {
134       if (suivant)printf(" ");
135       else suivant = VRAI;
136       printf("%s", rep_trav->tab_travailleurs[j].nom);
137     }
138   }
139   printf("\n");
140   return;
141 }
142
143 // Travailleurs -----
144 void traite_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav) {
145   Mot nom_specialite;
146   get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
147   unsigned int i = 0;
148   Boolean suivant = FAUX;
149   if (strcmp(nom_specialite, "tous") == 0) { // si tous es entré on affiche tous les travailleurs
150     while (i < rep_spe->nb_specialites) { //parcours de l'ensemble des specialites
151       affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs competatents pour la specialite d'index i
152       i++;
153     }
154   }
155   else { // sinon on affiche uniquement le client indiqué par l'utilisateur
156     while (i < rep_spe->nb_specialites) { //parcours de l'ensemble des specialites
157       if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
158         affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs compétatents pour la specialite d'index i
159         return;
160       }
161       i++;
162     }
163   }
164   printf(MSG_SPECIALITES_ERREUR);
165 }
166

```

```

168 // Specialités -----
169 void traite_specialites(const Specialites* rep_specialites) {
170     printf(MSG_SPECIALITES);
171     if (rep_specialites->nb_specialites == 0) {
172         printf("\n");
173         return;
174     }
175     for (int i = 0; i < rep_specialites->nb_specialites;i++) {
176         printf("%s/%d",rep_specialites->tab_specialites[i].nom, rep_specialites->tab_specialites[i].cout_horaire);
177         if (i != rep_specialites->nb_specialites - 1)printf(", ");
178     }
179     printf("\n");
180 }
181
182 // Progression -----
183 void traite_progression() {
184     Mot nom_commande, nom_specialite;
185     get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
186     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
187     int nbr_heure = get_int(); // recuperation aupres de l'utilisateur du nombred'heure de la tache a effectuer
188     printf(MSG_PROGRESSION, nom_commande, nom_specialite, nbr_heure);
189 }
190
191 // Passe -----
192 void traite_passe() {
193     printf(MSG_PASSE); // affiche MSG_PASSE
194 }
195
196 // Tâches -----
197 void traite_tache() {
198     Mot nom_commande, nom_specialite;
199     get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
200     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
201     int nbr_heure = get_int(); //recuperation du nombre d'heure de la tache a effectuer
202     printf(MSG_TACHE, nom_commande, nom_specialite, nbr_heure);
203 }
204
205 // Demarche -----
206 void traite_demarche(Clients* rep_cli) {
207     get_id(rep_cli->tab_clients[rep_cli->nb_clients]);
208     rep_cli->nb_clients += 1;
209 }

```

```

211 // Embauche -----
212 void traite_embauche(Travailleurs* rep_trav, const Specialites* rep_spe) {
213     Mot nom_specialite;
214     Travailleur travailleur;
215     unsigned int j = 0;
216     get_id(travailleur.nom); // on recupere le nom du travailleur
217     get_id(nom_specialite); // on recupere le nom de la specialite maîtrisée
218     for (j = 0; j < rep_trav->nb_travailleurs; j++) { // on parcourt l'ensemble des travailleurs enregistrés
219         if (strcmp(travailleur.nom, rep_trav->tab_travailleurs[j].nom) == 0) {
220             for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble des spécialités enregistrées
221                 if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
222                     rep_trav->tab_travailleurs[j].tags_competences[i] = VRAI; // on passe la compétence d'index i à VRAI pour le travailleur d'index j
223                     return;
224                 }
225             }
226         }
227     }
228     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble des spécialités enregistrées
229         if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
230             travailleur.tags_competences[i] = VRAI;
231             break;
232         }
233     }
234     rep_trav->tab_travailleurs[rep_trav->nb_travailleurs] = travailleur;
235     rep_trav->nb_travailleurs += 1;
236 }
237
238 // Developpe -----
239 void traite_developpe(Specialites* rep_spe) {
240     Specialite spe;
241     get_id(spe.nom); // on récupère le nom de la spécialité auprès de l'utilisateur
242     spe.cout_horaire = get_int(); // on récupère le cout horaire de la spécialité auprès de l'utilisateur
243     rep_spe->tab_specialites[rep_spe->nb_specialites] = spe;
244     rep_spe->nb_specialites += 1;
245 }
246
247 // Interruption -----
248 void traite_interruption() {
249     printf(MSG INTERRUPTION);
250 }
251

```

```

252 //Boucle principale -----
253 int main(int argc, char* argv[]) {
254     if (argc >= 2 && strcmp("echo", argv[1]) == 0) {
255         EchoActif = VRAI;
256     }
257     Mot buffer;
258     Booleen progression = FAUX; // permet de verifier si la derniere valeur de "buffer" etait on non "progression"
259     Specialites rep_specialites; // repertoire des specialites
260     Travailleurs rep_travailleurs; // repertoire des travailleurs
261     Clients rep_clients; // repertoire des clients
262
263     rep_clients.nb_clients = 0;
264     rep_specialites.nb_specialites = 0;
265     rep_travailleurs.nb_travailleurs = 0;
266     while (VRAI) {
267         get_id(buffer); // on demande la commande a entrer a l'utilisateur
268         if (progression == VRAI && strcmp(buffer, "passe") == 0) {
269             traite_passe(); // on execute la fonction traite_passe
270             progression = FAUX;
271             continue;
272         }
273     else {
274         progression = FAUX;
275     }
276     if (strcmp(buffer, "commande") == 0) {
277         traite_commande(); // on execute la fonction traite_commande
278         continue;
279     }
280     if (strcmp(buffer, "charge") == 0) {
281         traite_charge(); // on execute la fonction traite_charge
282         continue;
283     }
284     if (strcmp(buffer, "supervision") == 0) {
285         traite_supervision(); // on execute la fonction traite_supervision
286         continue;
287     }
288     if (strcmp(buffer, "client") == 0) {
289         traite_client(&rep_clients); // on execute la fonction traite_client
290         continue;
291     }
292     if (strcmp(buffer, "travailleurs") == 0) {
293         traite_travailleurs(&rep_specialites, &rep_travailleurs); // on execute la fonction traite_travailleurs
294         continue;
295     }
296     if (strcmp(buffer, "specialites") == 0) {
297         traite_specialites(&rep_specialites); // on execute la fonction traite_specialites
298         continue;
299     }
}

```

```
300     if (strcmp(buffer, "progression") == 0) {
301         traite_progression(); // on execute la fonction traite_progression
302         progression = VRAI;
303         continue;
304     }
305     if (strcmp(buffer, "tache") == 0) {
306         traite_tache(); // on execute la fonction traite_tache
307         continue;
308     }
309     if (strcmp(buffer, "demarche") == 0) {
310         traite_demarche(&rep_clients); // on execute la fonction traite_demarche
311         continue;
312     }
313     if (strcmp(buffer, "embauche") == 0) {
314         traite_embauche(&rep_travailleurs, &rep_specialites); // on execute la fonction traite_embauche
315         continue;
316     }
317     if (strcmp(buffer, "developpe") == 0) {
318         traite_developpe(&rep_specialites); // on execute la fonction traite_developpe
319         continue;
320     }
321     if (strcmp(buffer, "interruption") == 0) {
322         traite_interruption(); // on execute la fonction traite_interruption
323         break;
324     }
325     printf("!!! instruction inconnue >%s< !!!\n", buffer);
326 }
327 return 0;
328 }
```

## Sprint 3 :

```

1  #pragma warning(disable:4996) // on désactive l'avertissement sur les scanf
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <assert.h>
6
7  typedef enum { FAUX = 0, VRAI = 1 } Boolean; // on définit le type booléen qui n'est pas inclus dans le C
8  Boolean EchoActif = FAUX; // le mode de débogage est désactivé par défaut
9
10 // Messages émis par les instructions -----
11 // on définit ci dessous les messages à afficher selon les fonctions utilisées
12 #define MSG_DEVELOPPE "## nouvelle spécialité \"%s\" ; cout horaire \"%d\"\n"
13 #define MSG INTERRUPTION "## fin de programme\n"
14 #define MSG_EMBAUCHE "## nouveau travailleur \"%s\" compétent pour la spécialité \"%s\"\n"
15 #define MSG DEMARCHE "## nouveau client \"%s\"\n"
16 #define MSG TACHE "## la commande \"%s\" requiert la spécialité \"%s\" (nombre d'heures \"%d\")\n"
17 #define MSG PROGRESSION "## pour la commande \"%s\", pour la spécialité \"%s\" : \"%d\" heures de plus ont été réalisées\n"
18 #define MSG PASSE "## une réallocation est requise\n"
19 #define MSG SPECIALITES "spécialités traitées : "
20 #define MSG SPECIALITES_ERREUR "spécialité inconnue\n"
21 #define MSG TRAVAILLEURS "la spécialité %s peut être prise en charge par : "
22 #define MSG CLIENT "le client %s a commandé : "
23 #define MSG_CLIENT_ERREUR "client inconnu\n"
24 #define MSG_CLIENT_ID_COMMANDE "%s"
25 #define MSG SUPERVISION "état des tâches pour %s : "
26 #define MSG CHARGE "## consultation de la charge de travail de \"%s\"\n"
27 #define MSG COMMANDE "## nouvelle commande \"%s\", par client \"%s\"\n"
28
29 // Lexèmes -----
30 #define LGMOT 35 // longueur d'un mot (35 caractères)
31 #define NBCHIFFREMAX 5 // taille d'un nombre (5 chiffres maximums)
32 typedef char Mot[LGMOT + 1]; // définition du type mot
33
34 void get_id(Mot id) { // fonction de récupération d'un mot
35     scanf("%s", id);
36     if (EchoActif) printf(">>echo %s\n", id); // si le mode débogage est activé on affiche le mot entre précédemment
37     assert(strlen(id) > 1); // on vérifie que le mot fait plus d'un caractère
38 }
39 int get_int() { // fonction de récupération d'un entier positif
40     char buffer[NBCHIFFREMAX + 1];
41     scanf("%s", buffer);
42     if (EchoActif) printf(">>echo %s\n", buffer); // si le mode débogage est activé on affiche le mot entre précédemment
43     assert(atoi(buffer) >= 0); // on vérifie que le nombre entre est un entier positif
44     return atoi(buffer);
45 }
46

```

```

49 // specialites -----
50 #define MAX_SPECIALITES 10 // on definie le nombre maximum de specialites
51 typedef struct { //une specialite est composee d'un nom de type mot et d'un cout horaire entier
52   Mot nom;
53   int cout_horaire;
54 } Specialite;
55 typedef struct { //l'ensemble des specilites est regroupe dans une structure contenant un tableau de specilites ainsi qu'un entier contenant le nombre de specialites
56   Specialite tab_specialites[MAX_SPECIALITES];
57   unsigned int nb_specialites;
58 } Specialites;
59 // travailleurs -----
60 #define MAX_TRAVAILLEURS 50 // on definie le nombre maximum de travailleurs
61 typedef struct { // un travailleur possede un nom de type mot, des competences repertoriees dans un tableau de boolean et un nombres d'heures de travailles a effectues entier
62   Mot nom;
63   Boolean tags_competences[MAX_SPECIALITES]; // VRAI ou FAUX avec VRAI indiquant que le travailleur maîtrise la specialites numero n stockee a la case n du tableau de specialites
64 } Travailleur;
65 typedef struct { //l'ensemble des travailleurs est regroupe dans une structure contenant un tableau de travailleur ainsi qu'un entier contenant le nombre de travailleurs
66   Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
67   unsigned int nb_travailleurs;
68 } Travailleurs;
69 // client -----
70 #define MAX_CLIENTS 100 // on definie le nombre maximum de clients
71 typedef struct { // les clients sont repertoriés dans un tableau de mot contenant le nom de chaque client ainsi qu'un entier stockant le nombre de clients
72   Mot tab_clients[MAX_CLIENTS];
73   unsigned int nb_clients;
74 } Clients;
75 // commandes -----
76 #define MAX_COMMANDES 500 // on definie le nombre maximum de commandes
77 typedef struct { // une tache est definie pas sont nombre d'heures requises ainsi que son nombre d'heures effectues (les deux étais des entiers naturels)
78   unsigned int nb_heures_requeses;
79   unsigned int nb_heures_effectuees;
80 } Tache;
81 // une commande est definie par son nom (mot), les client auquelle elle est associee (entier naturel),
82 // les taches qui lui sont associeesla travailleurs en charge des taches ainsi que la facture qui lui est associee
83 typedef struct {
84   Mot nom;
85   unsigned int idx_client;
86   Tache taches_par_specialite[MAX_SPECIALITES]; /* nb_heures_requeses==0 <-> pas de tache pour cette specialite,
87   tache numero n est associee a la specialite n du tableau des specialites
88 */
89 } Commande;
90 /*l'ensemble des commandes est regroupe dans une structure contenant un tableau de travailleur ainsi qu'un entier contenant le nombre de travailleurs
91 et qu'un entier contenant les nombre de commandes facturees*/
92 typedef struct {
93   Commande tab_commandes[MAX_COMMANDES];
94   unsigned int nb_commandes;
95 } Commandes;
96 // Instructions -----
97 // Commande -----
98 // Commande -----
100 void traite_commande(Commandes* rep_com, const Clients* rep_cli) {
101   Commande cmd;
102   Mot nom_client;
103   get_id(cmd.nom); // on recupere aupres de l'utilisateur le nom de la commande
104   get_id(nom_client); // on recupere aupres de l'utilisateur le nom du client
105   for (unsigned int i = 0; i < MAX_SPECIALITES; i++) { // on parcourt l'ensemble du tableau tache_par_specialite de cmd
106     cmd.taches_par_specialite[i].nb_heures_effectuees = 0;
107     cmd.taches_par_specialite[i].nb_heures_requeses = 0;
108   }
109   for (unsigned int i = 0; i < rep_cli->nb_clients; i++) { // on parcours l'ensemble des clients enregistres
110     if (strcmp(nom_client, rep_cli->tab_clients[i]) == 0) {
111       cmd.idx_client = i;
112       break;
113     }
114   }
115   rep_com->tab_commandes[rep_com->nb_commandes] = cmd;
116   rep_com->nb_commandes += 1;
117 }
118
119 // Charge -----
120 void traite_charge() {
121   Mot nom_travailleur;
122   get_id(nom_travailleur); // recuperation du nom du travailleur aupres de l'utilisateur
123   printf(MSG_CHARGE, nom_travailleur);
124 }
125
126

```

```

127 // Supervision -----
128 void traite_supervision(const Specialites* rep_spe, const Commandes* rep_com) {
129   Boolean suivant = FAUX;
130   int requis, effectuees; // on declare deux entier qui correspondent aux nombres d'heures requises et effectuees pour une tache donnee pour simplifier la lisibilité
131   if (rep_com->nb_commandes > 0) {
132     for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // on parcours l'ensemble des commandes
133       printf(MSG_SUPERVISION, rep_com->tab_commandes[i].nom);
134       for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
135
136         requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requises;
137         effectuees = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees;
138
139         if (requis != 0) {
140           if (suivant)printf(" ");
141           else suivant = VRAI;
142           printf("%s:%d/%d", rep_spe->tab_specialites[j].nom, effectuees, requis);
143         }
144       }
145       printf("\n");
146       suivant = FAUX;
147     }
148   }
149 }
150
151 // Affichage Client -----
152 void affiche_clients(const Clients* rep_cli, const Commandes* rep_com, int i){
153   Boolean suivant = FAUX;
154   printf(MSG_CLIENT, rep_cli->tab_clients[i]);
155   for (unsigned int j = 0; j < rep_com->nb_commandes; j++) { // parcours de l'ensemble des commandes
156     if (rep_com->tab_commandes[j].idx_client == i) { // on verifie si l'index du client associe a la commande j est celui du client indique en parametre
157       if (suivant)printf(" ");
158       else suivant = VRAI;
159       printf(MSG_CLIENT_ID_COMMANDE, rep_com->tab_commandes[j].nom);
160     }
161   }
162   printf("\n");
163   return;
164 }
165
166 // Client -----
167 void traite_client(const Clients* rep_cli, const Commandes* rep_com) {
168   Mot nom_client;
169   get_id(nom_client); // recuperation du nom du client aupres de l'utilisateur
170   unsigned int i = 0;
171   Boolean suivant = FAUX;
172
173   if (strcmp(nom_client, "tous") == 0) { // si tous est entre on affiche tous les clients
174     while (i < rep_cli->nb_clients) { //parcours de l'ensemble des clients
175       affiche_clients(rep_cli, rep_com, i);
176       i++;
177     }
178   }
179   else { // sinon on affiche uniquement le client indique par l'utilisateur
180     while (i < rep_cli->nb_clients) { // parcourt de l'ensemble des clients
181       if (strcmp(nom_client, rep_cli->tab_clients[i]) == 0) {
182         affiche_clients(rep_cli, rep_com, i);
183         return;
184       }
185       i++;
186     }
187   }
188   printf(MSG_CLIENT_ERREUR); // si client non enregistre
189 }
190 }
191
192
193 // Affiche travailleurs -----
194 void affiche_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav, int i) {
195   Boolean suivant = FAUX;
196   printf(MSG_TRAVAILLEURS, rep_spe->tab_specialites[i].nom);
197   for (unsigned int j = 0; j < rep_trav->nb_travailleurs; j++) { // parcours de l'ensemble des travailleurs
198     if (rep_trav->tab_travailleurs[j].tags_competences[i] == VRAI) { // on verifie si le travailleur d'index j maîtrise la competence d'index i
199       if (suivant)printf(" ");
200       else suivant = VRAI;
201       printf("%s", rep_trav->tab_travailleurs[j].nom);
202     }
203   }
204   printf("\n");
205   return; // fin de la fonction
206 }
207

```

```

209 // Travailleurs -----
210 void traite_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav) {
211     Mot nom_specialite;
212     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
213     unsigned int i = 0; // initialisation du compteur i a 0
214     Boolean suivant = FAUX;
215     if (strcmp(nom_specialite, "tous") == 0) { // si tous es entré on affiche tous les travailleurs
216         while (i < rep_spe->nb_specialites) { //parcours de l'ensemble des specialites
217             affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs competatents pour la specialite d'index i
218             i++; // incrementation de i
219         }
220     }
221     else { // sinon on affiche uniquement le client indique par l'utilisateur
222         while (i < rep_spe->nb_specialites) { //parcours de l'ensemble des specialites
223             if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) { // on verifie si la specialite renseignee est la meme que celle d'index i
224                 affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs competatents pour la specialite d'index i
225                 return;
226             }
227             i++; // incrementation de i
228         }
229         printf(MSG_SPECIALITES_ERREUR);
230     }
231 }
232
233 // Specialités -----
234 void traite_specialites(const Specialites* rep_spe) {
235     printf(MSG_SPECIALITES);
236     if (rep_spe->nb_specialites == 0) { // on verifie si le nombre de specialites enregistrees est nul
237         printf("\n");
238         return;
239     }
240     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // parcours de l'ensemble des specialites
241         printf("%s/%d", rep_spe->tab_specialites[i].nom, rep_spe->tab_specialites[i].cout_horaire);
242         if (i != rep_spe->nb_specialites - 1)printf(", ");
243     }
244     printf("\n");
245 }
246
247 // Progression -----
248 v void traite_progression(const Specialites* rep_spe, Commandes* rep_com) {
249     Mot nom_commande, nom_specialite;
250     get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
251     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
252     int nbr_heure = get_int(),diff, requis, effectuees;
253     for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // parcours de l'ensemble des commandes
254         if (strcmp(rep_com->tab_commandes[i].nom, nom_commande) == 0) {
255
256             for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
257                 requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requeses;
258                 if (strcmp(rep_spe->tab_specialites[j].nom, nom_specialite) == 0 && requis!=0) {
259                     // on verifie si la specialite renseignee est la meme que celle d'index j et que requis est non nul
260
261                     rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees += nbr_heure;
262                     requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requeses;
263                     effectuees = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees;
264                     diff = requis - effectuees;
265
266                     if (diff<=0) {
267                         rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees = requis; // mise a niveau du nb d'heures effectuees (en cas de depassement)
268                     }
269                     break;
270                 }
271             }
272             break;
273         }
274     }
275 }
276
277 // Passe -----
278 v void traite_passe() {
279     return;
280 }

```

```

282 // Tâches -----
283 void traite_tache(const Specialites* rep_spe, Commandes* rep_com) {
284     Mot nom_commande, nom_specialite;
285     get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
286     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
287     int nbr_heure = get_int();
288     for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // parcours de l'ensemble des commandes
289         if (strcmp(rep_com->tab_commandes[i].nom, nom_commande) == 0) {
290             for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
291                 if (strcmp(rep_spe->tab_specialites[j].nom, nom_specialite) == 0) {
292                     rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requises = nbr_heure;
293                     break;
294                 }
295             }
296             break;
297         }
298     }
299 }
300
301 // Demarche -----
302 void traite_demarche(Clients* rep_cli) {
303     get_id(rep_cli->tab_clients[rep_cli->nb_clients]);
304     rep_cli->nb_clients += 1;
305 }
306
307 // Embauche -----
308 void traite_embauche(Travailleurs* rep_trav, const Specialites* rep_spe) {
309     Mot nom_specialite;
310     Travailleur travailleur;
311     unsigned int j = 0;
312     get_id(travailleur.nom); // on recupere le nom du travailleur aupres de l'utilisateur
313     get_id(nom_specialite); // recuperation de la specialite a maitriser
314     for (j = 0; j < rep_trav->nb_travailleurs; j++) { // on parcourt l'ensemble des travailleurs enregistres
315         if (strcmp(travailleur.nom, rep_trav->tab_travailleurs[j].nom) == 0) {
316             for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble de specialites enregistrees
317                 if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
318                     rep_trav->tab_travailleurs[j].tags_competences[i] = VRAI;
319                     return;
320                 }
321             }
322         }
323     }
324     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble des specialites enregistrees
325         if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
326             travailleur.tags_competences[i] = VRAI;
327             break;
328         }
329     }
330     rep_trav->tab_travailleurs[rep_trav->nb_travailleurs] = travailleur;
331     rep_trav->nb_travailleurs += 1;
332 }
333
334
335 // developpe -----
336 void traite_developpe(Specialites* rep_spe) {
337     Specialite spe;
338     get_id(spe.nom); // on recupere le nom de la specialite aupres de l'utilisateur
339     spe.cout_horaire = get_int(); // on recuperate le cout horaire de la specialite aupres de l'utilisateur
340     rep_spe->tab_specialites[rep_spe->nb_specialites] = spe;
341     rep_spe->nb_specialites += 1;
342 }
343
344
345 // interruption -----
346 void traite_interruption() {
347     printf(MSG INTERRUPTION);
348 }
349

```

---

```

350 //Boucle principale -----
351 int main(int argc, char* argv[]) {
352 if (argc >= 2 && strcmp("echo", argv[1]) == 0) { // on verifie si le mot echo a ete mis en parametre du programme lors de l'execution en console
353     EchoActif = VRAI; // activation du mode debuggage
354 }
355 // declaration des variables utilisees dans le main
356 Mot buffer;
357 Boolean progression = FAUX; // permet de verifier si la derniere valeur de "buffer" etait on non "progression"
358 Specialites rep_specialites; // repertoire des specialites
359 Travailleurs rep_travailleurs; // repertoire des travailleurs
360 Clients rep_clients; // repertoire des clients
361 Commandes rep_commandes; // repertoire des commandes
362
363 rep_clients.nb_clients = 0;
364 rep_specialites.nb_specialites = 0;
365 rep_travailleurs.nb_travailleurs = 0;
366 rep_commandes.nb_commandes = 0;
367 while (VRAI) { // boucle infinie
368     get_id(buffer); // on demande la commande a entrer a l'utilisateur
369     if (progression == VRAI && strcmp(buffer, "passe") == 0) {
370         traite_passe();
371         progression = FAUX;
372         continue;
373     }
374     else {
375         progression = FAUX;
376     }
377     if (strcmp(buffer, "commande") == 0) {
378         traite_commande(&rep_commandes, &rep_clients); // on execute la fonction traite_commande
379         continue;
380     }
381     if (strcmp(buffer, "charge") == 0) {
382         traite_charge(); // on execute la fonction traite_charge
383         continue;
384     }
385     if (strcmp(buffer, "supervision") == 0) {
386         traite_supervision(&rep_specialites, &rep_commandes); // on execute la fonction traite_supervision
387         continue;
388     }
}

```

```

389     if (strcmp(buffer, "client") == 0) {
390         traite_client(&rep_clients, &rep_commandes); // on execute la fonction traite_client
391         continue;
392     }
393     if (strcmp(buffer, "travailleurs") == 0) {
394         traite_travailleurs(&rep_specialites, &rep_travailleurs); // on execute la fonction traite_travailleurs
395         continue;
396     }
397     if (strcmp(buffer, "specialites") == 0) {
398         traite_specialites(&rep_specialites); // on execute la fonction traite_specialites
399         continue;
400     }
401     if (strcmp(buffer, "progression") == 0) {
402         traite_progression(&rep_specialites, &rep_commandes); // on execute la fonction traite_progression
403         progression = VRAI; // passage de progression a VRAI
404         continue;
405     }
406     if (strcmp(buffer, "tache") == 0) {
407         traite_tache(&rep_specialites, &rep_commandes); // on execute la fonction traite_tache
408         continue;
409     }
410     if (strcmp(buffer, "demarche") == 0) {
411         traite_demarche(&rep_clients); // on execute la fonction traite_demarche
412         continue;
413     }
414     if (strcmp(buffer, "embauche") == 0) {
415         traite_embauche(&rep_travailleurs, &rep_specialites); // on execute la fonction traite_embauche
416         continue;
417     }
418     if (strcmp(buffer, "developpe") == 0) {
419         traite_developpe(&rep_specialites); // on execute la fonction traite_developpe
420         continue;
421     }
422     if (strcmp(buffer, "interruption") == 0) {
423         traite_interruption(); // on execute la fonction traite_interruption
424         break;
425     }
426     printf("!!! instruction inconnue >%s< !!!\n", buffer);
427 }
428 return 0;
429 }

```

## Sprint 4 :

```

1 #pragma warning(disable:4996) // on désactive l'avertissement sur les scanf
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <assert.h>
6
7 typedef enum { FAUX = 0, VRAI = 1 } Boolean; // on définit le type booléen qui n'est pas inclus dans le C
8 Boolean EchoActif = FAUX; // le mode de débogage est désactivé par défaut
9
10 // Messages émis par les instructions -----
11 // on définit ci dessous les messages à afficher selon les fonctions utilisées
12 #define MSG_DEVELOPPE "## nouvelle spécialité \"%s\" ; cout horaire \"%d\"\n"
13 #define MSG INTERRUPTION "## fin de programme\n"
14 #define MSG_EMBAUCHE "## nouveau travailleur \"%s\" compétent pour la spécialité \"%s\"\n"
15 #define MSG DEMARCHE "## nouveau client \"%s\"\n"
16 #define MSG TACHE "## la commande \"%s\" requiert la spécialité \"%s\" (nombre d'heures \"%d\")\n"
17 #define MSG PROGRESSION "## pour la commande \"%s\" , pour la spécialité \"%s\" : \"%d\" heures de plus ont été réalisées\n"
18 #define MSG PASSE "## une réallocation est requise\n"
19 #define MSG SPECIALITES "spécialités traitées : "
20 #define MSG SPECIALITES_ERREUR "spécialité inconnue\n"
21 #define MSG TRAVAILLEURS "la spécialité %s peut être prise en charge par : "
22 #define MSG CLIENT "le client %s a commandé : "
23 #define MSG CLIENT_ERREUR "client inconnu\n"
24 #define MSG CLIENT_ID_COMMANDE "%s"
25 #define MSG SUPERVISION "état des tâches pour %s : "
26 #define MSG CHARGE "charge de travail pour %s : "
27 #define MSG COMMANDE "## nouvelle commande \"%s\" , par client \"%s\"\n"
28 #define MSG CHARGE_TACHE "%s/%s/%dheure(s)"
29
30 // Lexèmes -----
31 #define LGMOT 35 // longueur d'un mot (35 caractères)
32 #define NBCHIFFREMAX 5 // taille d'un nombre (5 chiffres maximums)
33 typedef char Mot[LGMOT + 1]; // définition du type mot
34 void get_id(Mot id) { // fonction de récupération d'un mot
35     scanf("%s", id);
36     if (EchoActif) printf(">>echo %s\n", id); // si le mode débogage est activé on affiche le mot entre précédemment
37     assert(strlen(id) > 1); // on vérifie que le mot fait plus d'un caractère
38 }
39 int get_int() { // fonction de récupération d'un entier positif
40     char buffer[NBCHIFFREMAX + 1];
41     scanf("%s", buffer);
42     if (EchoActif) printf(">>echo %s\n", buffer); // si le mode débogage est activé on affiche le mot entre précédemment
43     assert(atoi(buffer) >= 0); // on vérifie que le nombre entre est un entier positif
44     return atoi(buffer);
45 }
46

```

```

49 // specialites -----
50 #define MAX_SPECIALITES 10 // on definie le nombre maximum de specialites
51 typedef struct { //une specialite est composee d'un nom de type mot et d'un cout horaire entier
52     Mot nom;
53     int cout_horaire;
54 } Specialite;
55 typedef struct { //l'ensemble des specilites est regroupe dans une structure contenant un tableau de specilites ainsi qu'un entier contenant le nombre de specialites
56     Specialite tab_specialites[MAX_SPECIALITES];
57     unsigned int nb_specialites;
58 } Specialites;
59 // travailleurs -----
60 #define MAX_TRAVAILLEURS 50 // on definie le nombre maximum de travailleurs
61 typedef struct { // un travailleur possede un nom de type mot, des competences repertoriees dans un tableau de booleen et un nombres d'heures de travailles a effectues entier
62     Mot nom;
63     Boolen tags_competences[MAX_SPECIALITES]; // VRAI ou FAUX avec VRAI indiquant que le travailleur maîtrise la specialites numero n stockee a la case n du tableau de specialites
64     unsigned int nb_heures_travail;
65 } Travailleur;
66 typedef struct { //l'ensemble des travailleurs est regroupe dans une structure contenant un tableau de travailleur ainsi qu'un entier contenant le nombre de travailleurs
67     Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
68     unsigned int nb_travailleurs;
69 } Travailleurs;
70 // client -----
71 #define MAX_CLIENTS 100 // on definie le nombre maximum de clients
72 typedef struct { // les clients sont repertoriés dans un tableau de mot contenant le nom de chaque client ainsi qu'un entier stockant le nombre de clients
73     Mot tab_clients[MAX_CLIENTS];
74     unsigned int nb_clients;
75 } Clients;
76 // commandes -----
77 #define MAX_COMMANDES 500 // on definie le nombre maximum de commandes
78 typedef struct { // une tache est definie pas sont nombre d'heures requises ainsi que son nombre d'heures effectues (les deux étais des entiers naturels)
79     unsigned int nb_heures_requises;
80     unsigned int nb_heures_effectuees;
81 } Tache;
82 // une commande est definie par son nom (mot), les client auquelle elle est associee (entier naturel)
83 // les taches qui lui sont associeesla travailleurs en charge des taches ainsi que la facture qui lui est associee
84 typedef struct []
85 {
86     Mot nom;
87     unsigned int idx_client;
88     Tache taches_par_specialite[MAX_SPECIALITES];
89     // nb_heures_requises==0 <=> pas de tache pour cette specialite,
90     // tache numero n est associee a la specialite n du tableau des specialites
91     int idx_trav_tache[MAX_SPECIALITES]; // pour la tache numero n est associe l'index du travailleur en charge de la tache
92 }
93 //l'ensemble des commandes est regroupe dans une structure contenant un tableau de travailleur ainsi qu'un entier contenant le nombre de travailleurs
94 //et qu'un entier contenant les nombre de commandes facturees
95 typedef struct {
96     Commande tab_commandes[MAX_COMMANDES];
97     unsigned int nb_commandes;
98 } Commandes;
99 // déclaration des fonctions -----
100 void traite_developpe(Specialites* rep_spe); // permet des renseigner une nouvelle specialite
101 void traite_embauche(Travailleurs* rep_trav, const Specialites* rep_spe); // permet des renseigner un nouveau travailleur
102 void traite_demande(Clients* rep_cli); // permet de renseigner un nouveau client
103 void traite_commande(Commandes* rep_com, const Clients* rep_cli); // permet de renseigner une nouvelle commande
104 void traite_supervision(const Specialites* rep_spe, const Commandes* rep_com); // affiche l'état de toutes les taches pour toutes les commandes
105 void traite_client(const Clients* rep_cli, const Commandes* rep_com); // recuper le nom du client a afficher (ou la commande "tous" pour afficher tout les clients)
106 void affiche_clients(const Clients* rep_cli, const Commandes* rep_com, int i); // affiche les commandes effectuées par un client
107 // recuper le nom du travailleur a afficher (ou la commande "tous" pour afficher tout les travailleurs)
108 void traite_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav);
109 void affiche_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav, int i); // affiche les travailleurs maîtrisant la spécialité demandée
110 void traite_specialites(const Specialites* rep_spe); // affiche les specilites traitees
111 //cree une nouvelle tache dans la commande demandee pour la specialite demandee, la tache est ensuite assignee a un travailleur
112 void traite_tache(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav);
113 // fait progresser la tache demande du nombre d'heure specifiee, si laache est completee on enclanche la facturation
114 void traite_progression(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav);
115 // affiche les tache assignees au travailleur demande ainsi que lenmbre d'heure qu'il reste a effetuer pour les tache en question
116 void traite_charge(const Travailleurs* rep_trav, const Commandes* rep_com, const Specialites* rep_spe);
117 void traite_passe(); //affiche un message
118 void traite_assignment(const int idx_com, const int idx_spe, Commandes* rep_com, Travailleurs* rep_trav); // la tache donnee en arametre est assignee a un travailleur
119 void traite_interruption(); // interromp le programme avant une fin complete du fonctionnement

```

```

123 //Boucle principale -----
124 ~ int main(int argc, char* argv[]) {
125 ~ | if (argc >= 2 && strcmp("echo", argv[1]) == 0) { // on verifie si le mot echo a ete mis en parametre du programme lors de l'execution en console
126 | | EchoActif = VRAI; // activation du mode debugage
127 }
128 // declaration des variables utilisees dans le main
129 Mot buffer;
130 Booleen progression = FAUX; // permet de verifier si la derniere valeur de "buffer" etait on non "progression"
131 Specialites rep_specialites; // repertoire des specialites
132 Travailleurs rep_travailleurs; // repertoire des travailleurs
133 Clients rep_clients; // repertoire des clients
134 Commandes rep_commandes; // repertoire des commandes
135
136 rep_clients.nb_clients = 0;
137 rep_specialites.nb_specialites = 0;
138 rep_travailleurs.nb_travailleurs = 0;
139 rep_commandes.nb_commandes = 0;
140 ~ while (VRAI) {
141 | get_id(buffer); // on demande la commande a entrer a l'utilisateur
142 ~ | if (progression == VRAI && strcmp(buffer, "passe") == 0) {
143 | | traite_passe();
144 | | progression = FAUX;
145 | | continue;
146 }
147 ~ | else {
148 | | progression = FAUX;
149 }
150 ~ | if (strcmp(buffer, "commande") == 0) {
151 | | traite_commande(&rep_commandes, &rep_clients);
152 | | continue;
153 }
154 ~ | if (strcmp(buffer, "charge") == 0) {
155 | | traite_charge(&rep_travailleurs, &rep_commandes, &rep_specialites);
156 | | continue;
157 }
158 ~ | if (strcmp(buffer, "supervision") == 0) {
159 | | traite_supervision(&rep_specialites, &rep_commandes);
160 | | continue;
161 }
162 ~ | if (strcmp(buffer, "client") == 0) {
163 | | traite_client(&rep_clients, &rep_commandes);
164 | | continue;
165 }
166 ~ | if (strcmp(buffer, "travailleurs") == 0) {
167 | | traite_travailleurs(&rep_specialites, &rep_travailleurs);
168 | | continue;
169 }
170 ~ | if (strcmp(buffer, "tache") == 0) {
171 | | traite_tache(&rep_specialites, &rep_commandes, &rep_travailleurs);
172 | | continue;
173 }
174 ~ | if (strcmp(buffer, "demarche") == 0) {
175 | | traite_demarche(&rep_clients);
176 | | continue;
177 }
178 ~ | if (strcmp(buffer, "embauche") == 0) {
179 | | traite_embauche(&rep_travailleurs, &rep_specialites);
180 | | continue;
181 }
182 ~ | if (strcmp(buffer, "developpe") == 0) {
183 | | traite_developpe(&rep_specialites);
184 | | continue;
185 }
186 ~ | if (strcmp(buffer, "interruption") == 0) {
187 | | traite_interruption();
188 | | break;
189 }
190 ~ | printf("!!! instruction inconnue >%s< !!!\n", buffer);
191 }
192 }
193 return 0;
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }

```

```

206 // Instructions -----
207
208 // developpe -----
209 void traite_developpe(Specialites* rep_spe) {
210     Specialite spe;
211     get_id(spe.nom); // on recupere le nom de la specialite aupres de l'utilisateur
212     spe.cout_horaire = get_int(); // on recupere le cout horaire de la specialite aupres de l'utilisateur
213     rep_spe->tab_specialites[rep_spe->nb_specialites] = spe;
214     rep_spe->nb_specialites += 1;
215 }
216
217 // Embauche -----
218 void traite_embauche(Travailleurs* rep_trav, const Specialites* rep_spe) {
219     Mot nom_specialite;
220     Travailleur travailleur;
221     unsigned int j = 0;
222     get_id(travailleur.nom); // on recupere le nom du travailleur aupres de l'utilisateur
223     get_id(nom_specialite); // recuperation de la specialite a maîtriser
224     for (j = 0; j < rep_trav->nb_travailleurs; j++) { // on parcourt l'ensemble des travailleurs enregisitres
225         if (strcmp(travailleur.nom, rep_trav->tab_travailleurs[j].nom) == 0) {
226             for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble de specialites enregistrees
227                 if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
228                     rep_trav->tab_travailleurs[j].tags_competences[i] = VRAI;
229                     return;
230                 }
231             }
232         }
233     }
234     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble des specialites enregistrees
235         if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
236             travailleur.tags_competences[i] = VRAI;
237             break;
238         }
239     }
240     travailleur.nb_heures_travail = 0;
241     rep_trav->tab_travailleurs[rep_trav->nb_travailleurs] = travailleur;
242     rep_trav->nb_travailleurs += 1;
243 }
244
245 // Demarche -----
246 void traite_demarche(Clients* rep_cli) {
247     get_id(rep_cli->tab_clients[rep_cli->nb_clients]);
248     rep_cli->nb_clients += 1;
249 }
250

```

```

251 // Commande -----
252 void traite_commande(Commandes* rep_com, const Clients* rep_cli) {
253     Commande cmd;
254     Mot nom_client;
255     get_id(cmd.nom); // on recupere aupres de l'utilisateur le nom de la commande
256     get_id(nom_client); // on recupere aupres de l'utilisateur le nom du client
257     for (unsigned int i = 0; i < MAX_SPECIALITES; i++) { // on parcourt l'ensemble du tableau tache_par_specialite de cmd
258         cmd.taches_par_specialite[i].nb_heures_effectuees = 0;
259         cmd.taches_par_specialite[i].nb_heures_requesees = 0;
260         cmd.idx_trav_tache[i] = -1; // on initialise ma valeur de l'index du travailleur associe a la tache i a -1 (non assignee)
261     }
262     for (unsigned int i = 0; i < rep_cli->nb_clients; i++) { // on parcours l'ensemble des clients enregistres
263         if (strcmp(nom_client, rep_cli->tab_clients[i]) == 0) {
264             cmd.idx_client = i;
265             break;
266         }
267     }
268     rep_com->tab_commandes[rep_com->nb_commandes] = cmd;
269     rep_com->nb_commandes += 1;
270 }
271
272 // Supervision -----
273 void traite_supervision(const Specialites* rep_spe, const Commandes* rep_com) {
274     Boolean suivant = FAUX;
275     int requis, effectuees; // on declare deux entier qui correspondent aux nombres d'heures requesees et effectuees pour une tache donnee pour simplifier la lisibilite
276     if (rep_com->nb_commandes > 0) {
277         for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // on parcours l'ensemble des commandes
278             printf(MSG_SUPERVISION, rep_com->tab_commandes[i].nom);
279             for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
280
281                 requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requesees;
282                 effectuees = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees;
283
284                 if (requis != 0) {
285                     if (suivant)printf(", ");
286                     else suivant = VRAI;
287                     printf("%s:%d/%d", rep_spe->tab_specialites[j].nom, effectuees, requis);
288                 }
289             }
290             printf("\n");
291             suivant = FAUX;
292         }
293     }
294 }
295
296 // Client -----
297 void traite_client(const Clients* rep_cli, const Commandes* rep_com) {
298     Mot nom_client;
299     get_id(nom_client); // recuperation du nom du client aupres de l'utilisateur
300     unsigned int i = 0;
301     Boolean suivant = FAUX;
302
303     if (strcmp(nom_client, "tous") == 0) { // si tous est entre on affiche tous les clients
304         while (i < rep_cli->nb_clients) { //parcours de l'ensemble des clients
305             affiche_clients(rep_cli, rep_com, i);
306             i++;
307         }
308     }
309     else { // sinon on affiche uniquement le client indique par l'utilisateur
310         while (i < rep_cli->nb_clients) { // parcourt de l'ensemble des clients
311             if (strcmp(nom_client, rep_cli->tab_clients[i]) == 0) {
312                 affiche_clients(rep_cli, rep_com, i);
313                 return;
314             }
315             i++;
316         }
317         printf(MSG_CLIENT_ERREUR); // si client non enregistre
318     }
319 }
320
321 // Affichage Client -----
322 void affiche_clients(const Clients* rep_cli, const Commandes* rep_com, int i){
323     Boolean suivant = FAUX;
324     printf(MSG_CLIENT, rep_cli->tab_clients[i]);
325     for (unsigned int j = 0; j < rep_com->nb_commandes; j++) { // parcours de l'ensemble des commandes
326         if (rep_com->tab_commandes[j].idx_client == i) { // on verifie si l'index du client associe a la commande j est celui du client indique en parametre
327             if (suivant)printf(", ");
328             else suivant = VRAI;
329             printf(MSG_CLIENT_ID_COMMANDE, rep_com->tab_commandes[j].nom);
330         }
331     }
332     printf("\n");
333 }
334
335

```

```

336 // Travailleurs -----
337 void traite_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav) {
338     Mot nom_specialite;
339     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
340     unsigned int i = 0; // initialisation du compteur i a 0
341     Boolean suivant = FAUX;
342     if (strcmp(nom_specialite, "tous") == 0) { // si tous es entré on affiche tous les travailleurs
343         while (i < rep_spe->nb_specialites) { //parcours de l'ensemble des specialites
344             affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs competatents pour la specialite d'index i
345             i++; // incrementation de i
346         }
347     }
348     else { // sinon on affiche uniquement le client indiqué par l'utilisateur
349         while (i < rep_spe->nb_specialites) { //parcours de l'ensemble des specialites
350             if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) { // on verifie si la specialite renseignee est la même que celle d'index i
351                 affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs competatents pour la specialite d'index i
352                 return;
353             }
354             i++; // incrementation de i
355         }
356         printf(MSG_SPECIALITES_ERREUR);
357     }
358 }
359
360 // Affiche travailleurs -----
361 void affiche_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav, int i) {
362     Boolean suivant = FAUX;
363     printf(MSG_TRAVAILLEURS, rep_spe->tab_specialites[i].nom);
364     for (unsigned int j = 0; j < rep_trav->nb_travailleurs; j++) { // parcours de l'ensemble des travailleurs
365         if (rep_trav->tab_travailleurs[j].tags_competences[i] == VRAI) { // on verifie si le travailleur d'index j maîtrise la compétence d'index i
366             if (suivant)printf(" ");
367             else suivant = VRAI;
368             printf("%s", rep_trav->tab_travailleurs[j].nom);
369         }
370     }
371     printf("\n");
372     return; // fin de la fonction
373 }
374
375 // Specialités -----
376 void traite_specialites(const Specialites* rep_spe) {
377     printf(MSG_SPECIALITES);
378     if (rep_spe->nb_specialites == 0) { // on verifie si le nombre de specialites enregistrees est nul
379         printf("\n");
380         return;
381     }
382     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // parcours de l'ensemble des specialites
383         printf("%s/%d", rep_spe->tab_specialites[i].nom, rep_spe->tab_specialites[i].cout_horaire);
384         if (i != rep_spe->nb_specialites - 1)printf(" ");
385     }
386     printf("\n");
387 }
388
389
390 // Tâches -----
391 void traite_tache(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav) {
392     Mot nom_commande, nom_specialite;
393     get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
394     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
395     int nbr_heure = get_int();
396     for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // parcours de l'ensemble des commandes
397         if (strcmp(rep_com->tab_commandes[i].nom, nom_commande) == 0) {
398             for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
399                 if (strcmp(rep_spe->tab_specialites[j].nom, nom_specialite) == 0) {
400                     rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requises = nbr_heure;
401                     traite_assignment(i, j, rep_com, rep_trav); // affectation de la tache d'index j pour la commande d'index i à un travailleur
402                     break;
403                 }
404             }
405             break;
406         }
407     }
408 }
409

```

```

410 // Progression -----
411 void traite_progression(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav) {
412   Mot nom_commande, nom_specialite;
413   get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
414   get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
415   int nbr_heure = get_int(),diff, requis, effectuees;
416   for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // parcours de l'ensemble des commandes
417     if (strcmp(rep_com->tab_commandes[i].nom, nom_commande) == 0) {
418
419       for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
420         requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requeses;
421
422         // on verifie si la specialite renseignee est la meme que celle d'index j et que requis est non nul
423         if (strcmp(rep_spe->tab_specialites[j].nom, nom_specialite) == 0 && requis!=0) {
424
425           rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees += nbr_heure; //on ajoute nbr_heures au nombre d'heures effectuees pour la tache d'index j
426           rep_trav->tab_traveilleurs[rep_com->tab_commandes[i].idx_trav_tache[j]].nb_heures_travail -= nbr_heure;
427
428           requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requeses;
429           effectuees = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees;
430           diff = requis - effectuees;
431
432           if (diff<=0) {
433             // compensation du nombres d'heures de travail a effectuer si depassement (effectuees>requis)
434             rep_trav->tab_traveilleurs[rep_com->tab_commandes[i].idx_trav_tache[j]].nb_heures_travail += diff;
435             rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees = requis; // mise a niveau du nb d'heures effectuees (en cas de depassement)
436             rep_com->tab_commandes[i].idx_trav_tache[j] = -1; // reinitialisation de l'assigntion
437           }
438         }
439       }
440     }
441   }
442 }
443 }
444 }

445 // Charge -----
446 void traite_charge(const Travailleurs* rep_trav, const Commandes* rep_com, const Specialites* rep_spe) {
447   Mot nom_traveilleur;
448   get_id(nom_traveilleur); // recuperation du nom du travailleur aupres de l'utilisateur
449   int requis, effectuees, diff;
450   Booleen suivant = FAUX;
451   if (rep_trav->nb_traveilleurs == 0) return; // si il n'y a aucun travailleur enregistre la fonction d'interrompt
452   for (unsigned int i = 0; i < rep_trav->nb_traveilleurs; i++) { // parcours de l'ensemble des travailleurs
453     if (strcmp(nom_traveilleur, rep_trav->tab_traveilleurs[i].nom) == 0) {
454       printf(MSG_CHARGE, nom_traveilleur);
455       if (rep_com->nb_commandes == 0) {
456         printf("\n");
457         return;
458       }
459       for (unsigned int j = 0; j < rep_com->nb_commandes; j++) { // parcours de l'ensemble des commandes enregistrees
460         for (unsigned int k = 0; k < rep_spe->nb_specialites; k++) { // parcours de l'ensemble des specialites enregistrees
461           if (rep_com->tab_commandes[j].idx_trav_tache[k] == i) {
462             if (suivant)printf(", ");
463             else suivant = VRAI;
464             requis = rep_com->tab_commandes[j].taches_par_specialite[k].nb_heures_requeses;
465             effectuees = rep_com->tab_commandes[j].taches_par_specialite[k].nb_heures_effectuees;
466             diff = requis - effectuees; // on affecte a diff la difference entre requis et effectuees
467             printf(MSG_CHARGE_TACHE, rep_com->tab_commandes[j].nom, rep_spe->tab_specialites[k].nom, diff);
468           }
469         }
470       }
471     }
472     printf("\n");
473     suivant = FAUX;
474   }
475 }
476 }

477 // Passe -----
478 void traite_passe() {
479   return;
480 }
481 }

```

```

483 // Assigination -----
484 // assigne a un travailleur la tache d'index idx_spe de la commande d'index idx_com
485 void traite_assigination(const int idx_com, const int idx_spe, Commandes* rep_com, Travailleurs* rep_trav) {
486     Boolean suivant = FAUX;
487     int affecter = -1, requis, effectuees, diff;
488     for (unsigned int i = 0; i < rep_trav->nb_travailleurs; i++) { // parcours des travailleurs
489         if (rep_trav->tab_travailleurs[i].tags_competences[idx_spe] == VRAI) {
490             if (suivant) {
491                 if (rep_trav->tab_travailleurs[i].nb_heures_travail < rep_trav->tab_travailleurs[affecter].nb_heures_travail) {
492                     affecter = i;
493                 }
494             }
495             else {
496                 affecter = i;
497                 suivant = VRAI;
498             }
499         }
500     }
501     assert(affecter >= 0);
502     requis = rep_com->tab_commandes[idx_com].taches_par_specialite[idx_spe].nb_heures_requeses;
503     effectuees = rep_com->tab_commandes[idx_com].taches_par_specialite[idx_spe].nb_heures_effectuees;
504     diff = requis - effectuees;
505     rep_com->tab_commandes[idx_com].idx_trav_tache[idx_spe] = affecter; // on affecte a la tache le travailleur selectionne precedemment
506     rep_trav->tab_travailleurs[affecter].nb_heures_travail += diff; // on ajoute au travailleur en charge de la tache le nombre d'heure a effectuer pour la terminer
507 }
508
509 // interruption -----
510 void traite_interruption() {
511     printf(MSG INTERRUPTION); // affichage du message MSG INTERRUPTION
512 } // fin de traite_interruption

```

## Sprint 5 :

```

1  #pragma warning(disable:4996) // on désactive l'avertissement sur les scanf
2  include <stdio.h>
3  include <stdlib.h>
4  include <string.h>
5  include <assert.h>
6
7  typedef enum { FAUX = 0, VRAI = 1 } Boolean; // on definie le type booléen qui n'es pas inclus dans le C
8  Boolean EchoActif = FAUX; // le mode de débbugage est désactivé par défaut
9
10 // Messages emis par les instructions -----
11 // on definie ci dessous les messages a afficher selon les fonctions utilisées
12 #define MSG_DEVELOPPE "## nouvelle specialite \"%s\" ; cout horaire \"%d\"\n"
13 #define MSG INTERRUPTION "## fin de programme\n"
14 #define MSG_EMBAUCHE "## nouveau travailleur \"%s\" competent pour la specialite \"%s\"\n"
15 #define MSG DEMARCHE "## nouveau client \"%s\"\n"
16 #define MSG TACHE "## la commande \"%s\" requiere la specialite \"%s\" (nombre d'heures \"%d\")\n"
17 #define MSG PROGRESSION "## pour la commande \"%s\", pour la specialite \"%s\" : \"%d\" heures de plus ont ete realisees\n"
18 #define MSG PASSE "## une reallocation est requise\n"
19 #define MSG SPECIALITES "specialites traitees : "
20 #define MSG SPECIALITES_ERREUR "specialitee inconnue\n"
21 #define MSG TRAVAILLEURS "la specialite %s peut etre prise en charge par : "
22 #define MSG CLIENT "le client %s a commande : "
23 #define MSG_CLIENT_ERREUR "client inconnu\n"
24 #define MSG_CLIENT_ID_COMMANDE "%s"
25 #define MSG SUPERVISION "etat des taches pour %s : "
26 #define MSG CHARGE "charge de travail pour %s : "
27 #define MSG COMMANDE "## nouvelle commande \"%s\", par client \"%s\"\n"
28 #define MSG CHARGE_TACHE "%s/%s/%dheure(s)"
29 #define MSG FACTURATION "facturation %s : "
30 #define MSG FACTURATION_FINALE "facturations : "
31
32 // Lexemes -----
33 #define LGMOT 35 //longueur d'un mot (35 caracteres)
34 #define NBCHIFFREMAX 5 // taille d'un nombre (5 chiffres maximums)
35 typedef char Mot[LGMOT + 1]; // définition du type mot
36 void get_id(Mot id) { //fonction de récupération d'un mot
37     scanf("%s", id);
38     if (EchoActif) printf(">>echo %s\n", id); // si le mode débbugage est activé on affiche le mot entre precedemment
39     assert(strlen(id) > 1); // on verifie que le mot fait plus d'un caractere
40 }
41 int get_int() { // fonction de récupération d'un entier positif
42     char buffer[NBCHIFFREMAX + 1];
43     scanf("%s", buffer);
44     if (EchoActif) printf(">>echo %s\n", buffer); // si le mode débbugage est activé on affiche le mot entre precedemment
45     assert(atoi(buffer) >= 0); // on verifie que le nombre entre est un entier positif
46     return atoi(buffer);
47 }
48

```

```

51 // specialites -----
52 #define MAX_SPECIALITES 10 // on definie le nombre maximum de specialites
53 typedef struct { //une specialite est composee d'un nom de type mot et d'un cout horaire entier
54     Mot nom;
55     int cout_horaire;
56 } Specialite;
57 typedef struct { //l'ensemble des specilites est regroupe dans une structure contenant un tableau de specilites ainsi qu'un entier contenant le nombre de specialites
58     Specialite tab_specialites[MAX_SPECIALITES];
59     unsigned int nb_specialites;
60 } Specialites;
61 // travailleurs -----
62 #define MAX_TRAVAILLEURS 50 // on definie le nombre maximum de travailleurs
63 typedef struct { // un travailleur possede un nom de type mot, des competences repertoriees dans un tableau de booleen et un nombres d'heures de travailles a effectues entier
64     Mot nom;
65     Boolean tags_competences[MAX_SPECIALITES]; // VRAI ou FAUX avec VRAI indiquant que le travailleur maîtrise la specialites numero n stockee a la case n du tableau de specialites
66     unsigned int nb_heures_travail;
67 } Travailleur;
68 typedef struct { //l'ensemble des travailleurs est regroupe dans une structure contenant un tableau de travailleur ainsi qu'un entier contenant le nombre de travailleurs
69     Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
70     unsigned int nb_travailleurs;
71 } Travailleurs;
72 // client -----
73 #define MAX_CLIENTS 100 // on definie le nombre maximum de clients
74 typedef struct { // les clients sont repertoriés dans un tableau de mot contenant le nom de chaque client ainsi qu'un entier stockant le nombre de clients
75     Mot tab_clients[MAX_CLIENTS];
76     unsigned int nb_clients;
77 } Clients;
78 // commandes -----
79 #define MAX_COMMANDES 500 // on definie le nombre maximum de commandes
80 typedef struct { // une tache est definie pas sont nombre d'heures requises ainsi que son nombre d'heures effectuees (les deux étais des entiers naturels)
81     unsigned int nb_heures_requises;
82     unsigned int nb_heures_effectuees;
83 } Tache;
84 // une commande est definie par son nom (mot), les client auquelle elle est associee (entier naturel)
85 // les taches qui lui sont associeesla travailleurs en charge des taches ainsi que la facture qui lui est associee
86 typedef struct [
87     Mot nom;
88     unsigned int idx_client;
89     Tache taches_par_specialite[MAX_SPECIALITES];
90     // nb_heures_requises==0 <=> pas de tache pour cette specialite
91     // tache numero n est associee a la specialite n du tableau des specialites
92     int idx_trav_tache[MAX_SPECIALITES]; // pour la tache numero n est associe l'index du travailleur en charge de la tache, idx=-1 <=> tache non assignee
93     long facture; // facture=-1 <=> facture non calculee
94 ] Commande;
95 //l'ensemble des commandes est regroupe dans une structure contenant un tableau de travailleur ainsi qu'un entier contenant le nombre de travailleurs
96 //et qu'un entier contenant les nombre de commandes facturees
97 typedef struct {
98     Commande tab_commandes[MAX_COMMANDES];
99     unsigned int nb_commandes;
100    unsigned int nb_facturations;
101 } Commandes;

```

```

103 // déclaration des fonctions -----
104 void traite_developpe(Specialites* rep_spe); // permet des renseigner une nouvelle specialite
105 void traite_embauche(Travailleurs* rep_trav, const Specialites* rep_spe); // permet des renseigner un nouveau travailleur
106 void traite_demande(Clients* rep_cli); // permet de renseigner un nouveau client
107 void traite_commande(Commandes* rep_com, const Clients* rep_cli); // permet de renseigner une nouvelle commande
108 void traite_supervision(const Specialites* rep_spe, const Commandes* rep_com); // affiche l'état de toutes les taches pour toutes les commandes
109 void traite_client(const Clients* rep_cli, const Commandes* rep_com); // recuper le nom du client a afficher (ou la commande "tous" pour afficher tout les clients)
110 void affiche_clients(const Clients* rep_cli, const Commandes* rep_com, int i); // affiche les commandes effectuées par un client
111 // recuper le nom du travailleur a afficher (ou la commande "tous" pour afficher tout les travailleurs)
112 void traite_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav);
113 void affiche_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav, int i); // affiche les travailleurs maîtrisant la spécialité demandée
114 void traite_specialites(const Specialites* rep_spe); // affiche les spécialités traitées
115 // cree une nouvelle tache dans la commande demandée pour la spécialité demandée, la tache est ensuite assignée à un travailleur
116 void traite_tache(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav);
117 // fait progresser la tache demandée du nombre d'heure spécifiée, si la tache est complétée on enclanche la facturation
118 void traite_progression(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav, const Clients* rep_cli);
119 // affiche les tâches assignées au travailleur demandé ainsi que le nombre d'heure qu'il reste à effectuer pour les tâches en question
120 void traite_charge(const Travailleurs* rep_trav, const Commandes* rep_com, const Specialites* rep_spe);
121 void traite_passe(); // inutile
122 void traite_assignment(const int idx_com, const int idx_spe, Commandes* rep_com, Travailleurs* rep_trav); // la tâche donnée en paramètre est assignée à un travailleur
123 // vérifie que toutes les tâches d'une commande sont terminées et, si c'est le cas, calcule et enregistre la facture
124 void traite_facturation(int idx_com, const Specialites* rep_spe, Commandes* rep_com, const Clients* rep_cli);
125 // vérifie que toutes les commandes ont bien été facturées. Si c'est le cas affiche la liste des clients ainsi que le prix à payer pour chaque client avant de terminer le programme
126 void traite_fin(const Commandes* rep_com, const Clients* rep_cli);
127 void traite_interruption(); // interrompt le programme avant une fin complète du fonctionnement
128

131 //Boucle principale -----
132 int main(int argc, char* argv[]) {
133     if (argc >= 2 && strcmp("echo", argv[1]) == 0) { // on vérifie si le mot echo a été mis en paramètre du programme lors de l'exécution en console
134         EchoActif = VRAI; // activation du mode débogage
135     }
136     // déclaration des variables utilisées dans le main
137     Mot buffer;
138     Boolean progression = FAUX; // permet de vérifier si la dernière valeur de "buffer" était ou non "progression"
139     Specialites rep_specialites; // répertoire des spécialités
140     Travailleurs rep_travailleurs; // répertoire des travailleurs
141     Clients rep_clients; // répertoire des clients
142     Commandes rep_commandes; // répertoire des commandes
143
144     rep_clients.nb_clients = 0;
145     rep_specialites.nb_specialites = 0;
146     rep_travailleurs.nb_travailleurs = 0;
147     rep_commandes.nb_commandes = 0;
148     rep_commandes.nb_facturations = 0;
149     while (VRAI) {
150         get_id(buffer); // on demande la commande à entrer à l'utilisateur
151         if (progression == VRAI && strcmp(buffer, "passe") == 0) {
152             traite_passe();
153             progression = FAUX;
154             continue;
155         }
156         else {
157             progression = FAUX;
158         }
159         if (strcmp(buffer, "commande") == 0) {
160             traite_commande(&rep_commandes, &rep_clients);
161             continue;
162         }
163         if (strcmp(buffer, "charge") == 0) {
164             traite_charge(&rep_travailleurs, &rep_commandes, &rep_specialites);
165             continue;
166         }
167         if (strcmp(buffer, "supervision") == 0) {
168             traite_supervision(&rep_specialites, &rep_commandes);
169             continue;
170         }
171         if (strcmp(buffer, "client") == 0) {
172             traite_client(&rep_clients, &rep_commandes);
173             continue;
174         }
    }
}

```

```
175     if (strcmp(buffer, "travailleurs") == 0) {
176         traite_travailleurs(&rep_specialites, &rep_travailleurs);
177         continue;
178     }
179     if (strcmp(buffer, "specialites") == 0) {
180         traite_specialites(&rep_specialites);
181         continue;
182     }
183     if (strcmp(buffer, "progression") == 0) {
184         traite_progression(&rep_specialites, &rep_commandes, &rep_travailleurs, &rep_clients);
185         progression = VRAI;
186         continue;
187     }
188     if (strcmp(buffer, "tache") == 0) {
189         traite_tache(&rep_specialites, &rep_commandes, &rep_travailleurs);
190         continue;
191     }
192     if (strcmp(buffer, "demarche") == 0) {
193         traite_demarche(&rep_clients);
194         continue;
195     }
196     if (strcmp(buffer, "embauche") == 0) {
197         traite_embauche(&rep_travailleurs, &rep_specialites);
198         continue;
199     }
200     if (strcmp(buffer, "developpe") == 0) {
201         traite_developpe(&rep_specialites);
202         continue;
203     }
204     if (strcmp(buffer, "interruption") == 0) {
205         traite_interruption();
206         break;
207     }
208     printf("!!! instruction inconnue >%s< !!!\n", buffer);
209 }
210 return 0;
211 }
```

```

214 // Instructions -----
215
216 // developpe -----
217 void traite_developpe(Specialites* rep_spe) {
218     Specialite spe;
219     get_id(spe.nom); // on recupere le nom de la specialite aupres de l'utilisateur
220     spe.cout_horaire = get_int(); // on recupere le cout horaire de la specialite aupres de l'utilisateur
221     rep_spe->tab_specialites[rep_spe->nb_specialites] = spe;
222     rep_spe->nb_specialites += 1;
223 }
224
225 // Embauche -----
226 void traite_embauche(Travailleurs* rep_trav, const Specialites* rep_spe) {
227     Mot nom_specialite;
228     Travailleur travailleur;
229     unsigned int j = 0;
230     get_id(travailleur.nom); // on recupere le nom du travailleur aupres de l'utilisateur
231     get_id(nom_specialite); // recuperation de la specialite a maîtriser
232     for (j = 0; j < rep_trav->nb_travailleurs; j++) { // on parcourt l'ensemble des travailleurs enregistres
233         if (strcmp(travailleur.nom, rep_trav->tab_travailleurs[j].nom) == 0) {
234             for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble de specialites enregistrees
235                 if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
236                     rep_trav->tab_travailleurs[j].tags_competences[i] = VRAI;
237                     return;
238                 }
239             }
240         }
241     }
242     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble des specialites enregistrees
243         if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
244             travailleur.tags_competences[i] = VRAI;
245             break;
246         }
247     }
248     travailleur.nb_heures_travail = 0;
249     rep_trav->tab_travailleurs[rep_trav->nb_travailleurs] = travailleur;
250     rep_trav->nb_travailleurs += 1;
251 }
252
253 // Demarche -----
254 void traite_demarche(Clients* rep_cli) {
255     get_id(rep_cli->tab_clients[rep_cli->nb_clients]);
256     rep_cli->nb_clients += 1;
257 }
258

```

```

259 // Commande -----
260 void traite_commande(Commandes* rep_com, const Clients* rep_cli) {
261     Commande cmd;
262     Mot nom_client;
263     get_id(cmd.nom); // on recuperes aupres de l'utilisateur le nom de la commande
264     get_id(nom_client); // on recuperes aupres de l'utilisateur le nom du client
265     cmd.facture = -1;
266     for (unsigned int i = 0; i < MAX_SPECIALITES; i++) { // on parcourt l'ensemble du tableau tache_par_specialite de cmd
267         cmd.taches_par_specialite[i].nb_heures_effectuees = 0;
268         cmd.taches_par_specialite[i].nb_heures_requises = 0;
269         cmd.idx_trav_tache[i] = -1; // on initialise ma valeur de l'index du travailleur associe a la tache i a -1 (non assignee)
270     }
271     for (unsigned int i = 0; i < rep_cli->nb_clients; i++) { // on parcours l'ensemble des clients enregistres
272         if (strcmp(nom_client, rep_cli->tab_clients[i]) == 0) {
273             cmd.idx_client = i;
274             break;
275         }
276     }
277     rep_com->tab_commandes[rep_com->nb_commandes] = cmd;
278     rep_com->nb_commandes += 1;
279 }
280
281 // Supervision -----
282 void traite_supervision(const Specialites* rep_spe, const Commandes* rep_com) {
283     Boolean suivant = FAUX;
284     int requis, effectuees; // on declare deux entier qui correspondent aux nombres d'heures requises et effectuees pour une tache donnee pour simplifier la lisibilite
285     if (rep_com->nb_commandes > 0) {
286         for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // on parcours l'ensemble des commandes
287             printf(MSG_SUPERVISION, rep_com->tab_commandes[i].nom);
288             for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
289                 requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requises;
290                 effectuees = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees;
291
292                 if (requis != 0) {
293                     if (suivant)printf(" ");
294                     else suivant = VRAI;
295                     printf("%s:%d/%d", rep_spe->tab_specialites[j].nom, effectuees, requis);
296                 }
297             }
298             printf("\n");
299         }
300     }
301     suivant = FAUX;
302 }
303 }

304 // Client -----
305 void traite_client(const Clients* rep_cli, const Commandes* rep_com) {
306     Mot nom_client;
307     get_id(nom_client); // recuperation du nom du client aupres de l'utilisateur
308     unsigned int i = 0;
309     Boolean suivant = FAUX;
310
311     if (strcmp(nom_client, "tous") == 0) { // si tous est entre on affiche tous les clients
312         while (i < rep_cli->nb_clients) { //parcours de l'ensemble des clients
313             affiche_clients(rep_cli, rep_com, i);
314             i++;
315         }
316     }
317     else { // sinon on affiche uniquement le client indique par l'utilisateur
318         while (i < rep_cli->nb_clients) { // parcourt de l'ensemble des clients
319             if (strcmp(nom_client, rep_cli->tab_clients[i]) == 0) {
320                 affiche_clients(rep_cli, rep_com, i);
321                 return;
322             }
323             i++;
324         }
325     }
326     printf(MSG_CLIENT_ERREUR); // si client non enregistre
327 }
328 }

329 // Affichage Client -----
330 void affiche_clients(const Clients* rep_cli, const Commandes* rep_com, int i){
331     Boolean suivant = FAUX;
332     printf(MSG_CLIENT, rep_cli->tab_clients[i]);
333     for (unsigned int j = 0; j < rep_com->nb_commandes; j++) { // parcours de l'ensemble des commandes
334         if (rep_com->tab_commandes[j].idx_client == i) { // on verifie si l'index du client associe a la commande j est celui du client indique en parametre
335             if (suivant)printf(" ");
336             else suivant = VRAI;
337             printf(MSG_CLIENT_ID_COMMANDE, rep_com->tab_commandes[j].nom);
338         }
339     }
340     printf("\n");
341     return;
342 }
343 }

344

```

```

345 // Travailleurs -----
346 void traite_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav) {
347     Mot nom_specialite;
348     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
349     unsigned int i = 0; // initialisation du compteur i a 0
350     Boolean suivant = FAUX;
351     if (strcmp(nom_specialite, "tous") == 0) { // si tous es entré on affiche tous les travailleurs
352         while (i < rep_spe->nb_specialites) { //parcours de l'ensembl des specialites
353             affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs competatents pour la specialite d'index i
354             i++; // incrementation de i
355         }
356     }
357     else { // sinon on affiche uniquement le client indique par l'utilisateur
358         while (i < rep_spe->nb_specialites) { //parcours de l'ensembl des specialites
359             if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) { // on verifie si la specialite renseignee est la même que celle d'index i
360                 affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs competatents pour la specialite d'index i
361                 return;
362             }
363             i++; // incrementation de i
364         }
365         printf(MSG_SPECIALITES_ERREUR);
366     }
367 }
368
369 // Affiche travailleurs -----
370 void affiche_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav, int i) {
371     Boolean suivant = FAUX;
372     printf(MSG_TRAVAILLEURS, rep_spe->tab_specialites[i].nom);
373     for (unsigned int j = 0; j < rep_trav->nb_travailleurs; j++) { // parcours de l'ensemble des travailleurs
374         if (rep_trav->tab_travailleurs[j].tags_competences[i] == VRAI) { // on verifie si le travailleur d'index j maîtrise la compétence d'index i
375             if (suivant)printf(", ");
376             else suivant = VRAI;
377             printf("%s", rep_trav->tab_travailleurs[j].nom);
378         }
379     }
380     printf("\n");
381     return; // fin de la fonction
382 }
383
384 // Specialités -----
385 void traite_specialites(const Specialites* rep_spe) {
386     printf(MSG_SPECIALITES);
387     if (rep_spe->nb_specialites == 0) { // on verifie si le nombre de specialites enregistrees est nul
388         printf("\n");
389         return;
390     }
391     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // parcours de l'ensemble des specialites
392         printf("%s/%d", rep_spe->tab_specialites[i].nom, rep_spe->tab_specialites[i].cout_horaire);
393         if (i != rep_spe->nb_specialites - 1)printf(", ");
394     }
395     printf("\n");
396 }
397
398 // Tâches -----
399 void traite_tache(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav) {
400     Mot nom_commande, nom_specialite;
401     get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
402     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
403     int nbr_heure = get_int();
404     for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // parcours de l'ensemble des commandes
405         if (strcmp(rep_com->tab_commandes[i].nom, nom_commande) == 0) {
406             for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
407                 if (strcmp(rep_spe->tab_specialites[j].nom, nom_specialite) == 0) {
408                     rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requeses = nbr_heure;
409                     traite_assignation(i, j, rep_com, rep_trav); // affectation de la tache d'index j pour la commande d'index i a un travailleur
410                     break;
411                 }
412             }
413             break;
414         }
415     }
416 }
417

```

```

418 // Progression -----
419 void traite_progression(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav, const Clients* rep_cli) {
420   Mot nom_commande, nom_specialite;
421   get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
422   get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
423   int nbr_heure = get_int(),diff, requis, effectuees;
424   for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // parcours de l'ensemble des commandes
425     if (strcmp(rep_com->tab_commandes[i].nom, nom_commande) == 0 && rep_com->tab_commandes[i].facture<0) {
426
427       for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) [] // parcours de l'ensemble des specialites
428         requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requises;
429
430       // on verifie si la specialite renseignee est la meme que celle d'index j et que requis est non nul
431       if (strcmp(rep_spe->tab_specialites[j].nom, nom_specialite) == 0 && requis!=0) {
432
433         rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees += nbr_heure; // on ajoute nbr_heure au nombre d'heures effectuees pour la tache d'index j
434         rep_trav->tab_traveilleurs[rep_com->tab_commandes[i].idx_trav_tache[j]].nb_heures_travail -= nbr_heure;
435
436         requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requises;
437         effectuees = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees;
438         diff = requis - effectuees;
439
440       if (diff<=0) {
441         // compensation du nombres d'heures de travail a effectuer si depassement (effectuees>requis)
442         rep_trav->tab_traveilleurs[rep_com->tab_commandes[i].idx_trav_tache[j]].nb_heures_travail += diff;
443         rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees = requis; // mise a niveau du nb d'heures effectuees (en cas de depassement)
444         rep_com->tab_commandes[i].idx_trav_tache[j] = -1; // reinitialisation de l'assigntion
445         traite_facturation(i, rep_spe, rep_com, rep_cli); // facturation de la commande d'index i
446       }
447     }
448   }
449 }
450 }
451 }
452 }
453 }

455 // Charge -----
456 void traite_charge(const Travailleurs* rep_trav, const Commandes* rep_com, const Specialites* rep_spe) {
457   Mot nom_traveilleur;
458   get_id(nom_traveilleur); // recuperation du nom du travalleur aupres de l'utilisateur
459   int requis, effectuees, diff;
460   Booleen suivant = FAUX;
461   if (rep_trav->nb_traveilleurs == 0) return; // si il n'y a aucun travalleur enregistré la fonction d'interrompt
462   for (unsigned int i = 0; i < rep_trav->nb_traveilleurs; i++) { // parcours de l'ensemble des travalleurs
463     if (strcmp(nom_traveilleur, rep_trav->tab_traveilleurs[i].nom) == 0) {
464       printf(MSG_CHARGE, nom_traveilleur);
465       if (rep_com->nb_commandes == 0) {
466         printf("\n");
467         return;
468       }
469       for (unsigned int j = 0; j < rep_com->nb_commandes; j++) { // parcours de l'ensemble des commandes enregistrees
470         for (unsigned int k = 0; k < rep_spe->nb_specialites; k++) { // parcours de l'ensemble des specialites enregistrees
471           if (rep_com->tab_commandes[j].idx_trav_tache[k] == i) {
472             if (suivant)printf(", ");
473             else suivant = VRAI;
474             requis = rep_com->tab_commandes[j].taches_par_specialite[k].nb_heures_requises;
475             effectuees = rep_com->tab_commandes[j].taches_par_specialite[k].nb_heures_effectuees;
476             diff = requis - effectuees; // on affecte a diff la difference entre requis et effectuees
477             printf(MSG_CHARGE_TACHE, rep_com->tab_commandes[j].nom, rep_spe->tab_specialites[k].nom, diff);
478           }
479         }
480       }
481       printf("\n");
482       suivant = FAUX;
483     }
484   }
485 }
486

487 // Passe -----
488 void traite_passe() {
489   return;
490 }
491

```

```

492 // Assigmentation -----
493 void traite_assigmentation(const int idx_com, const int idx_spe, Commandes* rep_com, Travailleurs* rep_trav) {
494     // assigne a un travailleur la tache d'index idx_spe de la commande d'index idx_com
495     |
496     Boolean suivant = FAUX;
497     int affecter = -1, requis, effectuees, diff;
498     for (unsigned int i = 0; i < rep_trav->nb_travailleurs; i++) { // parcours des travailleurs
499         if (rep_trav->tab_travailleurs[i].tags_competences[idx_spe] == VRAI) {
500             if (suivant) {
501                 if (rep_trav->tab_travailleurs[i].nb_heures_travail < rep_trav->tab_travailleurs[affecter].nb_heures_travail) {
502                     affecter = i;
503                 }
504             } else {
505                 affecter = i;
506                 suivant = VRAI;
507             }
508         }
509     }
510     assert(affecter >= 0);
511     requis = rep_com->tab_commandes[idx_com].taches_par_specialite[idx_spe].nb_heures_requeses;
512     effectuees = rep_com->tab_commandes[idx_com].taches_par_specialite[idx_spe].nb_heures_effectuees;
513     diff = requis - effectuees;
514     rep_com->tab_commandes[idx_com].idx_trav_tache[idx_spe] = affecter; // on affecte a la tache le travailleur selectionne precedemment
515     rep_trav->tab_travailleurs[affecter].nb_heures_travail += diff; // on ajoute au travailleur en charge de la tache le nombre d'heure a effectuer pour la terminer
516
517 }
518

519 // Facturation -----
520 void traite_facturation(int idx_com, const Specialites* rep_spe, Commandes* rep_com, const Clients* rep_cli) {
521     long facture = 0;
522     int requis, cout_horaire;
523     Boolean suivant = FAUX;
524     if (rep_com->tab_commandes[idx_com].facture > 0) return; // si la facture a deja ete calculee la fonction s'interrompt
525     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // parcours de l'ensemble des specialites
526         requis = rep_com->tab_commandes[idx_com].taches_par_specialite[i].nb_heures_requeses;
527         if (rep_com->tab_commandes[idx_com].taches_par_specialite[i].nb_heures_effectuees == requis) {
528             cout_horaire = rep_spe->tab_specialites[i].cout_horaire;
529             facture += cout_horaire * requis;
530         } else {
531             return;
532         }
533     }
534     printf(MSG_FACTURATION, rep_com->tab_commandes[idx_com].nom);
535     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // parcours des specialites
536         requis = rep_com->tab_commandes[idx_com].taches_par_specialite[i].nb_heures_requeses;
537         if (requis > 0) {
538             if (suivant) printf(", ");
539             else suivant = VRAI;
540             cout_horaire = rep_spe->tab_specialites[i].cout_horaire;
541             printf("%s:%ld", rep_spe->tab_specialites[i].nom, cout_horaire * requis);
542         }
543     }
544     printf("\n");
545     rep_com->tab_commandes[idx_com].facture = facture;
546     rep_com->nb_facturations += 1;
547     traite_fin(rep_com, rep_cli); // execution de la commande de facturation finale
548 }
549 }
550

```

```
551 // Fin du programme -----
552 void traite_fin(const Commandes* rep_com,const Clients* rep_cli) {
553     int fact = 0; //
554     Boolean suivant = FAUX;
555     if (rep_com->nb_commandes == rep_com->nb_facturations) {// on verifie si toutes les fonctions ont bien ete facturees
556         printf(MSG_FACTURATION_FINALE);
557         for (unsigned int i = 0; i < rep_cli->nb_clients; i++) { // parcours de tout les clients enregistres
558             for (unsigned int j = 0; j < rep_com->nb_commandes; j++) { // parcours de toutes les commandes
559                 if (rep_com->tab_commandes[j].idx_client == i) {
560                     fact += rep_com->tab_commandes[j].facture;
561                 }
562             }
563             if (suivant)printf(" , ");
564             else suivant = VRAI;
565             printf("%s:%ld", rep_cli->tab_clients[i], fact);
566             fact = 0;
567         }
568         printf("\n");
569         exit(0); // sortie du programme
570     }
571 }
572
573 // interruption -----
574 void traite_interruption() {
575     printf(MSG INTERRUPTION);
576 }
577
```

## Sprint 6 :

```

1  #pragma warning(disable:4996) // on désactive l'avertissement sur les scanf
2  ~#include <stdio.h>
3  ~#include <stdlib.h>
4  ~#include <string.h>
5  ~#include <assert.h>
6
7  typedef enum { FAUX = 0, VRAI = 1 } Boolean; // on definie le type booléen qui n'es pas inclus dans le C
8  Boolean EchoActif = FAUX; // le mode de débbugage est désactivé par défaut
9
10 ~// Messages emis par les instructions -----
11 // on définie ci dessous les messages a afficher selon les fonctions utilisées
12 #define MSG_DEVELOPPE "## nouvelle specialite \"%s\" ; cout horaire \"%d\"\n"
13 #define MSG INTERRUPTION "## fin de programme\n"
14 #define MSG_EMBAUCHE "## nouveau travailleur \"%s\" competent pour la specialite \"%s\"\n"
15 #define MSG DEMARCHE "## nouveau client \"%s\"\n"
16 #define MSG TACHE "## la commande \"%s\" requiere la specialite \"%s\" (nombre d'heures \"%d\")\n"
17 #define MSG PROGRESSION "## pour la commande \"%s\", pour la specialite \"%s\" : \"%d\" heures de plus ont ete realisees\n"
18 #define MSG PASSE "## une reallocation est requise\n"
19 #define MSG SPECIALITES "specialites traitees : "
20 #define MSG SPECIALITES_ERREUR "specialitee inconnue\n"
21 #define MSG TRAVAILLEURS "la specialite %s peut etre prise en charge par : "
22 #define MSG CLIENT "le client %s a commande : "
23 #define MSG_CLIENT_ERREUR "client inconnu\n"
24 #define MSG_CLIENT_ID_COMMANDE "%s"
25 #define MSG SUPERVISION "etat des taches pour %s : "
26 #define MSG CHARGE "charge de travail pour %s : "
27 #define MSG_COMMANDE "## nouvelle commande \"%s\", par client \"%s\"\n"
28 #define MSG_CHARGE_TACHE "%s/%s/%dheure(s)"
29 #define MSG_FACTURATION "facturation %s : "
30 #define MSG_FACTURATION_FINALE "facturations : "
31
32 // Lexemes -----
33 #define LGMOT 35 //longueur d'un mot (35 caracteres)
34 #define NBCHIFFREMAX 5 // taille d'un nombre (5 chiffres maximums)
35 typedef char Mot[LGMOT + 1]; // définition du type mot
36 ~void get_id(Mot id) { //fonction de récupération d'un mot
37     scanf("%s", id);
38     if (EchoActif) printf(">>echo %s\n", id); // si le mode débbugage est activé on affiche le mot entre precedemment
39     assert(strlen(id) > 1); // on verifie que le mot fait plus d'un caractere
40 }
41 ~int get_int() { // fonction de récupération d'un entier positif
42     char buffer[NBCHIFFREMAX + 1];
43     scanf("%s", buffer);
44     if (EchoActif) printf(">>echo %s\n", buffer); // si le mode débbugage est activé on affiche le mot entre precedemment
45     assert(atoi(buffer) >= 0); // on verifie que le nombre entre est un entier positif
46     return atoi(buffer);
47 }

```

```

48 // Donnees -----
49
50 // specialites -----
51 #define MAX_SPECIALITES 10 // on definie le nombre maximum de specialites
52 typedef struct { //une specialite est composee d'un nom de type mot et d'un cout horaire entier
53     Mot nom;
54     int cout_horaire;
55 } Specialite;
56 typedef struct { //l'ensemble des specialites est regroupe dans une structure contenant un tableau de specialites ainsi qu'un entier contenant le nombre de specialites
57     Specialite tab_specialites[MAX_SPECIALITES];
58     unsigned int nb_specialites;
59 } Specialites;
60 // travailleurs -----
61 #define MAX_TRAVAILLEURS 50 // on definie le nombre maximum de travailleurs
62 typedef struct { // un travailleur possede un nom de type mot, des competences repertoriees dans un tableau de booleen et un nombres d'heures de travailles a effectues entier
63     Mot nom;
64     Boolean tags_competences[MAX_SPECIALITES]; // VRAI ou FAUX avec VRAI indiquant que le travailleur maitrise la specialites numero n stockee a la case n du tableau de specialites
65     unsigned int nb_heures_travail;
66 } Travailleur;
67 typedef struct { //l'ensemble des travailleurs est regroupe dans une structure contenant un tableau de travailleur ainsi qu'un entier contenant le nombre de travailleurs
68     Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
69     unsigned int nb_travailleurs;
70 } Travailleurs;
71 // client -----
72 #define MAX_CLIENTS 100 // on definie le nombre maximum de clients
73 typedef struct { // les clients sont repertoriees dans un tableau de mot contenant le nom de chaque client ainsi qu'un entier stockant le nombre de clients
74     Mot tab_clients[MAX_CLIENTS];
75     unsigned int nb_clients;
76 } Clients;
77 // commandes -----
78 #define MAX_COMMANDES 500 // on definie le nombre maximum de commandes
79 typedef struct { // une tache est definie par son nom (mot), les client auquelle elle est associee (entier naturel)
80     // les taches qui lui sont associees la travailleurs en charge des taches ainsi que la facture qui lui est associee
81     unsigned int nb_heures_requeses;
82     unsigned int nb_heures_effectuees;
} Tache;

84 // une commande est definie par son nom (mot), les client auquelle elle est associee (entier naturel)
85 // les taches qui lui sont associees la travailleurs en charge des taches ainsi que la facture qui lui est associee
86 typedef struct {
87     Mot nom;
88     unsigned int idx_client;
89     Tache taches_par_specialite[MAX_SPECIALITES];
90     // nb_heures_requeses==0 <=> pas de tache pour cette specialite
91     // tache numero n est associee a la specialite n du tableau des specialites
92     int idx_trav_tache[MAX_SPECIALITES]; // pour la tache numero n est associe l'index du travailleur en charge de la tache, idx=-1 <=> tache non assignee
93     long facture; // facture=-1 <=> facture non calculee
94 } Commande;
95 // l'ensemble des commandes est regroupe dans une structure contenant un tableau de travailleur ainsi qu'un entier contenant le nombre de travailleurs
96 // et qu'un entier contenant les nombre de commandes facturees
97 typedef struct {
98     Commande tab_commandes[MAX_COMMANDES];
99     unsigned int nb_commandes;
100    unsigned int nb_facturations;
101 } Commandes;

```

```

103 // déclaration des fonctions -----
104 void traite_developpe(Specialites* rep_spe); // permet des renseigner une nouvelle specialite
105 void traite_embauche(Travailleurs* rep_trav, const Specialites* rep_spe); // permet des renseigner un nouveau travailleur
106 void traite_demande(Clients* rep_cli); // permet de renseigner un nouveau client
107 void traite_commande(Commandes* rep_com, const Clients* rep_cli); // permet de renseigner une nouvelle commande
108 void traite_supervision(const Specialites* rep_spe, const Commandes* rep_com); // affiche l'état de toutes les taches pour toutes les commandes
109 void traite_client(const Clients* rep_cli, const Commandes* rep_com); // recuper le nom du client a afficher (ou la commande "tous" pour afficher tout les clients)
110 void affiche_clients(const Clients* rep_cli, const Commandes* rep_com, int i); // affiche les commandes effectuées par un client
111
112 // recuper le nom du travailleur a afficher (ou la commande "tous" pour afficher tout les travailleurs)
113 void traite_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav);
114 void affiche_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav, int i); // affiche les travailleurs maîtrisant la spécialité demandée
115 void traite_specialites(const Specialites* rep_spe); // affiche les spécialités traitées
116
117 // cree une nouvelle tache dans la commande demandée pour la spécialité demandée, la tache est ensuite assignée à un travailleur
118 void traite_tache(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav);
119
120 // fait progresser la tache demandée du nombre d'heure spécifiée, si la tache est complétée on enclanche la facturation
121 void traite_progression(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav, const Clients* rep_cli, int* idx_spe_passe, int* idx_com_passe);
122
123 // affiche les tâches assignées au travailleur demandé ainsi que le nombre d'heure qu'il reste à effectuer pour les tâches en question
124 void traite_charge(const Travailleurs* rep_trav, const Commandes* rep_com, const Specialites* rep_spe);
125
126 // réassigne la tâche à un autre travailleur à moins un "traite_prgression" uniquement
127 void traite_passe(const int idx_com, const int idx_spe, Commandes* rep_com, Travailleurs* rep_trav);
128 void traite_assignment(const int idx_com, const int idx_spe, Commandes* rep_com, Travailleurs* rep_trav); // la tâche donnée en paramètre est assignée à un travailleur
129
130 // vérifie que toutes les tâches d'une commande sont terminées et, si c'est le cas, calcule et enregistre la facture
131 void traite_facturation(int idx_com, const Specialites* rep_spe, Commandes* rep_com, const Clients* rep_cli);
132
133 // vérifie que toutes les commandes ont bien été facturées. Si c'est le cas affiche la liste des clients ainsi que le prix à payer pour chaque client avant de terminer le programme
134 void traite_fin(const Commandes* rep_com, const Clients* rep_cli);
135 void traite_interruption(); // interrompt le programme avant une fin complète du fonctionnement
136
137 //Boucle principale -----
138
139 int main(int argc, char* argv[]) {
140   if (argc >= 2 && strcmp("echo", argv[1]) == 0) { // on vérifie si le mot echo a été mis en paramètre lors de l'exécution en console
141     EchoActif = VRAI; // activation du mode debugage
142   }
143   // déclaration des variables utilisées dans le main
144   Mot buffer;
145   Boolean progression = FAUX; // permet de vérifier si la dernière valeur de "buffer" était ou non "progression"
146   Specialites rep_specialites; // répertoire des spécialités
147   Travailleurs rep_travailleurs; // répertoire des travailleurs
148   Clients rep_clients; // répertoire des clients
149   Commandes rep_commandes; // répertoire des commandes
150
151   int idx_spe_passe, idx_com_passe; // permettent de récupérer les informations sur la tâche modifiée dans la fonction progression
152   rep_clients.nb_clients = 0;
153   rep_specialites.nb_specialites = 0;
154   rep_travailleurs.nb_travailleurs = 0;
155   rep_commandes.nb_commandes = 0;
156   rep_commandes.nb_facturations = 0;
157

```

```

159     while (VRAI) {
160         get_id(buffer); // on demande la commande a entrer a l'utilisateur
161         if (progression == VRAI && strcmp(buffer, "passe") == 0) {
162             traite_passe(idx_com_passe, idx_spe_passe, &rep_commandes, &rep_travailleurs);
163             progression = FAUX;
164             continue;
165         }
166         else {
167             progression = FAUX;
168         }
169         if (strcmp(buffer, "commande") == 0) {
170             traite_commande(&rep_commandes, &rep_clients);
171             continue;
172         }
173         if (strcmp(buffer, "charge") == 0) {
174             traite_charge(&rep_travailleurs, &rep_commandes, &rep_specialites);
175             continue;
176         }
177         if (strcmp(buffer, "supervision") == 0) {
178             traite_supervision(&rep_specialites, &rep_commandes);
179             continue;
180         }
181         if (strcmp(buffer, "client") == 0) {
182             traite_client(&rep_clients, &rep_commandes);
183             continue;
184         }
185         if (strcmp(buffer, "travailleurs") == 0) {
186             traite_travailleurs(&rep_specialites, &rep_travailleurs);
187             continue;
188         }
189         if (strcmp(buffer, "specialites") == 0) {
190             traite_specialites(&rep_specialites);
191             continue;
192         }
193         if (strcmp(buffer, "progression") == 0) {
194             traite_progression(&rep_specialites, &rep_commandes, &rep_travailleurs, &rep_clients,&idx_spe_passe,&idx_com_passe);
195             progression = VRAI;
196             continue;
197         }
198         if (strcmp(buffer, "tache") == 0) {
199             traite_tache(&rep_specialites, &rep_commandes, &rep_travailleurs);
200             continue;
201         }
202     }
203     if (strcmp(buffer, "demarche") == 0) {
204         traite_demarche(&rep_clients);
205         continue;
206     }
207     if (strcmp(buffer, "embauche") == 0) {
208         traite_embauche(&rep_travailleurs, &rep_specialites);
209         continue;
210     }
211     if (strcmp(buffer, "developpe") == 0) {
212         traite_developpe(&rep_specialites);
213         continue;
214     }
215     if (strcmp(buffer, "interruption") == 0) {
216         traite_interruption();
217         break;
218     }
219     printf("!!! instruction inconnue >%s< !!!\n", buffer);
220 }
221 return 0;

```

---

```

224 // Instructions -----
225
226 // developpe -----
227 void traite_developpe(Specialites* rep_spe) {
228     Specialite spe;
229     get_id(spe.nom); // on recupere le nom de la specialite aupres de l'utilisateur
230     spe.cout_horaire = get_int(); // on recupere le cout horaire de la specialite aupres de l'utilisateur
231     rep_spe->tab_specialites[rep_spe->nb_specialites] = spe;
232     rep_spe->nb_specialites += 1;
233 }
234
235 // Embauche -----
236 void traite_embauche(Travailleurs* rep_trav, const Specialites* rep_spe) {
237     Mot nom_specialite;
238     Travailleur travailleur;
239     unsigned int j = 0;
240     get_id(travailleur.nom); // on recupere le nom du travailleur aupres de l'utilisateur
241     get_id(nom_specialite); // recuperation de la specialite a maitriser
242     for (j = 0; j < rep_trav->nb_travailleurs; j++) { // on parcourt l'ensemble des travailleurs enregistres
243         if (strcmp(travailleur.nom, rep_trav->tab_travailleurs[j].nom) == 0) {
244             for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble de specialites enregistrees
245                 if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
246                     rep_trav->tab_travailleurs[j].tags_competences[i] = VRAI;
247                     return;
248                 }
249             }
250         }
251     }
252     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // on parcourt l'ensemble des specialites enregistrees
253         if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) {
254             travailleur.tags_competences[i] = VRAI;
255             break;
256         }
257     }
258     travailleur.nb_heures_travail = 0;
259     rep_trav->tab_travailleurs[rep_trav->nb_travailleurs] = travailleur;
260     rep_trav->nb_travailleurs += 1;
261 }
262
263 // Demarche -----
264 void traite_demarche(Clients* rep_cli) {
265     get_id(rep_cli->tab_clients[rep_cli->nb_clients]);
266     rep_cli->nb_clients += 1;
267 }

```

---

```

269 // Commande -----
270 void traite_commande(Commandes* rep_com, const Clients* rep_cli) {
271   Commande cmd;
272   Mot nom_client;
273   get_id(cmd.nom); // on recuperer aupres de l'utilisateur le nom de la commande
274   get_id(nom_client); // on recuperer aupres de l'utilisateur le nom du client
275   cmd.facture = -1;
276   for (unsigned int i = 0; i < MAX_SPECIALITES; i++) { // on parcourt l'ensemble du tableau tache_par_specialite de cmd
277     cmd.taches_par_specialite[i].nb_heures_effectuees = 0;
278     cmd.taches_par_specialite[i].nb_heures_requises = 0;
279     cmd.idx_trav_tache[i] = -1; // on initialise ma valeur de l'index du travailleur associe a la tache i a -1 (non assignee)
280   }
281   for (unsigned int i = 0; i < rep_cli->nb_clients; i++) { // on parcours l'ensemble des clients enregistres
282     if (strcmp(nom_client, rep_cli->tab_clients[i]) == 0) {
283       cmd.idx_client = i;
284       break;
285     }
286   }
287   rep_com->tab_commandes[rep_com->nb_commandes] = cmd;
288   rep_com->nb_commandes += 1;
289 }
290
291 // Supervision -----
292 void traite_supervision(const Specialites* rep_spe, const Commandes* rep_com) {
293   Boolean suivant = FAUX;
294   int requis, effectuees; // on declare deux entier qui correspondent aux nombres d'heures requises et effectuees pour une tache donnee pour simplefier la lisibilite
295   if (rep_com->nb_commandes > 0) {
296     for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // on parcours l'ensemble des commandes
297       printf(MSG_SUPERVISION, rep_com->tab_commandes[i].nom);
298       for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
299
300         requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requises;
301         effectuees = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees;
302
303         if (requis != 0) {
304           if (suivant)printf(" ", " ");
305           else suivant = VRALI;
306           printf("%s:%d/%d", rep_spe->tab_specialites[j].nom, effectuees, requis);
307         }
308       }
309       printf("\n");
310       suivant = FAUX;
311     }
312   }
313 }

```

```

315 // Client -----
316 void traite_client(const Clients* rep_cli, const Commandes* rep_com) {
317     Mot nom_client;
318     get_id(nom_client); // recuperation du nom du client aupres de l'utilisateur
319     unsigned int i = 0;
320     Boolean suivant = FAUX;
321
322     if (strcmp(nom_client, "tous") == 0) { // si tous est entré on affiche tous les clients
323         while (i < rep_cli->nb_clients) { //parcours de l'ensemble des clients
324             affiche_clients(rep_cli,rep_com, i);
325             i++;
326         }
327     }
328     else { // sinon on affiche uniquement le client indiqué par l'utilisateur
329         while (i < rep_cli->nb_clients) { // parcourt de l'ensemble des clients
330             if (strcmp(nom_client, rep_cli->tab_clients[i]) == 0) {
331                 affiche_clients(rep_cli, rep_com, i);
332                 return;
333             }
334             i++;
335         }
336         printf(MSG_CLIENT_ERREUR); // si client non enregistre
337     }
338 }
339
340 // Affichage Client -----
341 void affiche_clients(const Clients* rep_cli, const Commandes* rep_com, int i){
342     Boolean suivant = FAUX;
343     printf(MSG_CLIENT, rep_cli->tab_clients[i]);
344     for (unsigned int j = 0; j < rep_com->nb_commandes; j++) { // parcours de l'ensemble des commandes
345         if (rep_com->tab_commandes[j].idx_client == i) { // on vérifie si l'index du client associé à la commande j est celui du client indiqué en paramètre
346             if (suivant)printf(", ");
347             else suivant = VRAI;
348             printf(MSG_CLIENT_ID_COMMANDE, rep_com->tab_commandes[j].nom);
349         }
350     }
351     printf("\n");
352     return;
353 }
354
355 // Travailleurs -----
356 void traite_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav) {
357     Mot nom_specialite;
358     get_id(nom_specialite); // recuperation du nom de la spécialité auprès de l'utilisateur
359     unsigned int i = 0; // initialisation du compteur i à 0
360     Boolean suivant = FAUX;
361     if (strcmp(nom_specialite, "tous") == 0) { // si tous es entré on affiche tous les travailleurs
362         while (i < rep_spe->nb_specialites) { //parcours de l'ensemble des spécialités
363             affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs compétents pour la spécialité d'index i
364             i++; // incrementation de i
365         }
366     }
367     else { // sinon on affiche uniquement le client indiqué par l'utilisateur
368         while (i < rep_spe->nb_specialites) { //parcours de l'ensemble des spécialités
369             if (strcmp(nom_specialite, rep_spe->tab_specialites[i].nom) == 0) { // on vérifie si la spécialité renseignée est la même que celle d'index i
370                 affiche_travailleurs(rep_spe, rep_trav, i); // affiche les travailleurs compétents pour la spécialité d'index i
371                 return;
372             }
373             i++; // incrementation de i
374         }
375         printf(MSG_SPECIALITES_ERREUR);
376     }
377 }
378
379 // Affiche travailleurs -----
380 void affiche_travailleurs(const Specialites* rep_spe, const Travailleurs* rep_trav, int i) {
381     Boolean suivant = FAUX;
382     printf(MSG_TRAVAILLEURS, rep_spe->tab_specialites[i].nom);
383     for (unsigned int j = 0; j < rep_trav->nb_travailleurs; j++) { // parcours de l'ensemble des travailleurs
384         if (rep_trav->tab_travailleurs[j].tags_competences[i] == VRAI) { // on vérifie si le travailleur d'index j maîtrise la compétence d'index i
385             if (suivant)printf(", ");
386             else suivant = VRAI;
387             printf("%s", rep_trav->tab_travailleurs[j].nom);
388         }
389     }
390     printf("\n");
391     return; // fin de la fonction
392 }

```

```

394 // Specialités -----
395 void traite_specialites(const Specialites* rep_spe) {
396     printf(MSG_SPECIALITES);
397     if (rep_spe->nb_specialites == 0) { // on verifie si le nombre de specialites enregistrees est nul
398         printf("\n");
399         return;
400     }
401     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // parcours de l'ensemble des specialites
402         printf("%s/%d", rep_spe->tab_specialites[i].nom, rep_spe->tab_specialites[i].cout_horaire);
403         if (i != rep_spe->nb_specialites - 1)printf(" , ");
404     }
405     printf('\n');
406 }
407
408 // Tâches -----
409 void traite_tache(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav) {
410     Mot nom_commande, nom_specialite;
411     get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
412     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
413     int nbr_heure = get_int();
414     for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // parcours de l'ensemble des commandes
415         if (strcmp(rep_com->tab_commandes[i].nom, nom_commande) == 0) {
416             for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
417                 if (strcmp(rep_spe->tab_specialites[j].nom, nom_specialite) == 0) {
418                     rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requeses = nbr_heure;
419                     traite_assignation(i, j, rep_com, rep_trav); // affectation de la tache d'index j pour la commande d'index i a un travailleur
420                     break;
421                 }
422             }
423             break;
424         }
425     }
426 }
427
428 // Progression -----
429 void traite_progression(const Specialites* rep_spe, Commandes* rep_com, Travailleurs* rep_trav, const Clients* rep_cli, int* idx_spe_passe, int* idx_com_passe) {
430     Mot nom_commande, nom_specialite;
431     get_id(nom_commande); // recuperation du nom de la commande aupres de l'utilisateur
432     get_id(nom_specialite); // recuperation du nom de la specialite aupres de l'utilisateur
433     int nbr_heure = get_int(),diff, requis, effectuees;
434     for (unsigned int i = 0; i < rep_com->nb_commandes; i++) { // parcours de l'ensemble des commandes
435         if (strcmp(rep_com->tab_commandes[i].nom, nom_commande) == 0 && rep_com->tab_commandes[i].facture<0) {
436
437             for (unsigned int j = 0; j < rep_spe->nb_specialites; j++) { // parcours de l'ensemble des specialites
438                 requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requeses;
439                 if (strcmp(rep_spe->tab_specialites[j].nom, nom_specialite) == 0 && requis!=0) {
440                     // on verifie si la specialite renseignee est la meme que celle d'index j et que requis est non nul
441
442                     rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees += nbr_heure;
443                     //on ajoute nbr_heures au nombre d'heures effectuees pour la tache d'index j
444
445                     rep_trav->tab_traveilleurs[rep_com->tab_commandes[i].idx_trav_tache[j]].nb_heures_travail -= nbr_heure;
446
447                     requis = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_requeses;
448                     effectuees = rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees;
449                     diff = requis - effectuees;
450
451                     if (diff<=0) {
452                         rep_trav->tab_traveilleurs[rep_com->tab_commandes[i].idx_trav_tache[j]].nb_heures_travail += diff;
453                         // compensation du nombres d'heures de travail a effectuer si depassement (effectuees>requis)
454
455                         rep_com->tab_commandes[i].taches_par_specialite[j].nb_heures_effectuees = requis; // mise a niveau du nb d'heures effectuees (en cas de depassement)
456                         rep_com->tab_commandes[i].idx_trav_tache[j] = -1; // reinitialisation de l'asssignation
457                         traite_facturation(i, rep_spe, rep_com, rep_cli); // facturation de la commande d'index i
458
459                     }
460                     *idx_com_passe = i;
461                     *idx_spe_passe = j;
462                     break;
463                 }
464             }
465         }
466     }
467 }

```

```

469 // Charge -----
470 void traite_charge(const Travailleurs* rep_trav, const Commandes* rep_com, const Specialites* rep_spe) {
471   Mot nom_travailleur;
472   get_id(nom_travailleur); // recuperation du nom du travailleur aupres de l'utilisateur
473   int requis, effectuees, diff;
474   Boolean suivant = FAUX;
475   if (rep_trav->nb_traveilleurs == 0) return; // si il n'y a aucun travailleur enregistre la fonction d'interrompt
476   for (unsigned int i = 0; i < rep_trav->nb_traveilleurs; i++) { // parcours de l'ensemble des travailleurs
477     if (strcmp(nom_travailleur, rep_trav->tab_traveilleurs[i].nom) == 0) {
478       printf(MSG_CHARGE, nom_travailleur);
479       if (rep_com->nb_commandes == 0) {
480         printf("\n");
481         return;
482       }
483       for (unsigned int j = 0; j < rep_com->nb_commandes; j++) { // parcours de l'ensemble des commandes enregistrees
484         for (unsigned int k = 0; k < rep_spe->nb_specialites; k++) { // parcours de l'ensemble des specialites enregistrees
485           if (rep_com->tab_commandes[j].idx_trav_tache[k] == i) {
486             if (suivant)printf(", ");
487             else suivant = VRAI;
488             requis = rep_com->tab_commandes[j].taches_par_specialite[k].nb_heures_requeses;
489             effectuees = rep_com->tab_commandes[j].taches_par_specialite[k].nb_heures_effectuees;
490             diff = requis - effectuees;
491             printf(MSG_CHARGE_TACHE, rep_com->tab_commandes[j].nom, rep_spe->tab_specialites[k].nom, diff);
492           }
493         }
494       }
495       printf("\n");
496       suivant = FAUX;
497     }
498   }
499 }

501 // Passe -----
502 void traite_passe(const int idx_com, const int idx_spe, Commandes* rep_com, Travailleurs* rep_trav) { //reassigne la tache d'index idx_spe pour la commande d'index idx_com
503   int idx_trav = rep_com->tab_commandes[idx_com].idx_trav_tache[idx_spe];
504   int requis = rep_com->tab_commandes[idx_com].taches_par_specialite[idx_spe].nb_heures_requeses;
505   int effectuees = rep_com->tab_commandes[idx_com].taches_par_specialite[idx_spe].nb_heures_effectuees;
506   int nb_heures = requis - effectuees;
507   if (nb_heures == 0) return; // si la tache est terminee la fonction s'interrompt
508
509   // on deduis le nombres d'heures restantes sur la tache au nombres d'heures a effectuer pour le travailleur en charge
510   rep_trav->tab_traveilleurs[idx_trav].nb_heures_travail -= nb_heures;
511   rep_com->tab_commandes[idx_com].idx_trav_tache[idx_spe] = -1; // reinitialisation de l'assignation de la tache
512   traite_assignment(idx_com, idx_spe, rep_com, rep_trav); // nouvelle assignation de la tache
513 }

514 // Assignation -----
515
516 // assigne a un travailleur la tache d'index idx_spe de la commande d'index idx_com
517 void traite_assignment(const int idx_com, const int idx_spe, Commandes* rep_com, Travailleurs* rep_trav) {
518   Boolean suivant = FAUX;
519   int affecter = -1, requis, effectuees, diff;
520   for (unsigned int i = 0; i < rep_trav->nb_traveilleurs; i++) { // parcours des travailleurs
521     if (rep_trav->tab_traveilleurs[i].tags_competences[idx_spe] == VRAI) {
522       if (suivant) {
523         if (rep_trav->tab_traveilleurs[i].nb_heures_travail < rep_trav->tab_traveilleurs[affecter].nb_heures_travail) {
524           affecter = i;
525         }
526       }
527     }
528     else {
529       affecter = i;
530       suivant = VRAI;
531     }
532   }
533   assert(affecter >= 0);
534   requis = rep_com->tab_commandes[idx_com].taches_par_specialite[idx_spe].nb_heures_requeses;
535   effectuees = rep_com->tab_commandes[idx_com].taches_par_specialite[idx_spe].nb_heures_effectuees;
536   diff = requis - effectuees;
537   rep_com->tab_commandes[idx_com].idx_trav_tache[idx_spe] = affecter; // on affecte a la tache le travailleur selectionne precedemment
538   rep_trav->tab_traveilleurs[affecter].nb_heures_travail += diff; // on ajoute au travailleur en charge de la tache le nombre d'heure a effectuer pour la terminer
539
540
541

```

```

542 // Facturation -----
543 void traite_facturation(int idx_com, const Specialites* rep_spe, Commandes* rep_com, const Clients* rep_cli) {
544     long facture = 0;
545     int requis, cout_horaire;
546     Booleen suivant = FAUX;
547     if (rep_com->tab_commandes[idx_com].facture > 0) return; // si la facture a déjà été calculée la fonction s'interrompt
548     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // parcours de l'ensemble des spécialités
549         requis = rep_com->tab_commandes[idx_com].taches_par_specialite[i].nb_heures_requises;
550         if (rep_com->tab_commandes[idx_com].taches_par_specialite[i].nb_heures_effectuées == requis) {
551             cout_horaire = rep_spe->tab_specialites[i].cout_horaire;
552             facture += cout_horaire * requis;
553         }
554         else {
555             return;
556         }
557     }
558     printf(MSG_FACTURATION, rep_com->tab_commandes[idx_com].nom);
559     for (unsigned int i = 0; i < rep_spe->nb_specialites; i++) { // parcours des spécialités
560         requis = rep_com->tab_commandes[idx_com].taches_par_specialite[i].nb_heures_requises;
561         if (requis > 0) {
562             if (suivant)printf(", ");
563             else suivant = VRAI;
564             cout_horaire = rep_spe->tab_specialites[i].cout_horaire;
565             printf("%s:%ld", rep_spe->tab_specialites[i].nom, cout_horaire * requis);
566         }
567     }
568     printf("\n");
569     rep_com->tab_commandes[idx_com].facture = facture;
570     rep_com->nb_facturations += 1;
571     traite_fin(rep_com, rep_cli); // exécution de la commande de facturation finale
572 }
573

```

```

574 // Fin du programme -----
575 void traite_fin(const Commandes* rep_com, const Clients* rep_cli) {
576     int fact = 0; //
577     Booleen suivant = FAUX;
578     if (rep_com->nb_commandes == rep_com->nb_facturations) { // on vérifie si toutes les fonctions ont bien été facturées
579         printf(MSG_FACTURATION_FINALE);
580         for (unsigned int i = 0; i < rep_cli->nb_clients; i++) { // parcours de tous les clients enregistrés
581             for (unsigned int j = 0; j < rep_com->nb_commandes; j++) { // parcours de toutes les commandes
582                 if (rep_com->tab_commandes[j].idx_client == i) {
583                     fact += rep_com->tab_commandes[j].facture;
584                 }
585             }
586             if (suivant)printf(", ");
587             else suivant = VRAI;
588             printf("%s:%ld", rep_cli->tab_clients[i], fact);
589             fact = 0;
590         }
591         printf("\n");
592         exit(0); // sortie du programme
593     }
594 }
595
596 // Interruption -----
597 void traite_interruption() {
598     printf(MSG INTERRUPTION);
599 }

```

