

RAPPORT PROJET D'IAP



PRÉSENTÉ PAR
EL SAMNY AHMED &
SOFIANE OULAD ITTO
(GROUPE 111)



TABLE DES *matières*

- 3 Présentation du problème
- 4 Suite présentation du problème
- 5 Bilan du projet
- 6 Code source des sprints dans toutes
leurs releases

Présentation du problème:

Nous avons réalisé un logiciel permettant de traiter et d'enregistrer plusieurs commandes sur une console dans une entreprise.

Le projet consiste donc à créer un programme permettant d'assigner des tâches à des travailleurs selon plusieurs critères, allant de leur compétence à leur facturation en passant par leur disponibilité.

Pour commencer, des spécialités peuvent être déclarées et enregistrées lors de l'entrée de l'une d'elles en machine (sur l'ordinateur) d'un membre du personnel. La commande se compose de l'instruction « developpe » puis d'un mot correspondant au nom que prend la spécialité ainsi que d'un entier qui prend comme valeur le coût horaire de cette spécialité. Par exemple, si l'on tape sur la console « developpe reseau 13 », tous les travailleurs de l'entreprise travaillant dans le réseau seront payés à 13€ de l'heure. Il faut savoir que le système permet d'enregistrer au maximum 10 spécialités. Ensuite, pour déclarer un nouveau travailleur on utilise l'instruction « embauche » suivi du nom du nouvel employé ainsi que le nom de sa spécialité. Par exemple « embauche Maxence graphisme » voudrait dire que Maxence est le nouvel employé travaillant dans le domaine du graphisme. La capacité maximale d'enregistrement est limitée à 50 travailleurs. Pour déclarer un client on utilise l'instruction « demarche » suivie du nom du client. Par exemple, « demarche Toto » voudrait dire qu'on déclare Toto comme étant un nouveau client de l'entreprise. 100 clients peuvent être déclarés au maximum. Pour déclarer une nouvelle commande on utilise l'instruction commande suivie du nom de la commande ainsi que du nom du client effectuant la commande. Par exemple, « commande superProduit Amel » voudrait dire qu'on a déclaré la commande superProduit pour la cliente Amel. Pour déclarer une tâche, qui, on le rappelle est nécessaire à la réalisation d'une commande on utilise l'instruction « tache » suivie du nom de la commande ainsi que du nom de la spécialité concernée et d'un nombre qui est le nombre d'heure nécessaire pour réaliser la tâche. Par exemple, « tache superProduit graphisme 12 » voudrait dire qu'on déclare une tâche pour la commande superProduit pour la spécialité graphisme et que le temps nécessaire à la réalisation de cette tâche serait de 12 heures.

Le système affecte automatiquement une tâche à un travailleur en fonction de si le travailleur a reçu peu de tâches, si le premier travailleur de la compétence est connu ou celui qui lui reste une quantité minime de travail à réaliser. Pour déclarer l'avancement d'une tâche on utilise l'instruction « progression » suivi du nom de la commande, de la spécialité et du nombre d'heures à ajouter à l'avancement. Si le mot clé « passe » est entré à la suite cela veut dire qu'il faut réaffecter la tâche.

Concernant l'affichage, si l'on souhaite afficher la liste des spécialités on entre l'instruction « specialites », pour la liste des travailleurs c'est l'instruction « travailleurs » qu'il faut entrer suivi du nom de la spécialité, l'instruction « travailleurs tous » affiche la liste des travailleurs pour toutes les spécialités, l'instruction « client » affiche la liste des commandes effectuées pour ce client. L'instruction « client tous » affiche la liste de toutes les commandes, « supervision » sert à consulter l'avancement des commandes et « charge » sert à consulter la charge de travail. Une facturation est affichée automatiquement par le système lorsqu'une commande est terminée et lorsque toutes les commandes sont terminées le système affiche une facturation globale.

Bilan du projet:

Chaque étape permettait petit à petit d'atteindre ce but, à chaque fois en créant ou en gérant différemment des fonctions et des structures. En effet, une fonction utilisée dans un sprint ne contiendra plus les mêmes instructions dans le sprint suivant.

De ce fait, il fallait alors faire en sorte que les fonctions utilisées soient flexibles, pour pouvoir les modifier aisément au fur et à mesure que l'on avance dans le projet.

Une autre difficulté présente durant le projet, et plus précisément à chaque nouveau sprint, est l'implémentation de nouvelles fonctions et structures.

En effet, lors du sprint 2 par exemple, il fallait ajouter au code plusieurs structures, qui sont présentées dans le guide technique du sprint.

Mais pour pouvoir faire le sprint 2, il fallait d'abord comprendre le rôle de chacune de ces structures, et comment elles fonctionnent, pour qu'on puisse les utiliser efficacement, et les implémenter dans le code, en plus d'ajouter et/ou modifier diverses fonctions.

Ces difficultés ont réellement commencé à apparaître lors du sprint 2, et se sont accentués pour le sprint 4, étant le dernier sprint réalisé pour notre groupe.

Voyant que le sprint 3 suit la même logique que le sprint 2, nous sommes directement passé au sprint 4 car le temps était compté et ainsi éviter de perdre du temps sur le sprint 3 qui fait presque la même chose que le sprint 2.

Nous allons maintenant mettre nos différents sprints dans leur version release, avec les différentes preuves de validation en utilisant l'outil diffchecker, qui permet de comparer deux fichiers et donc de savoir si les sorties sont identiques ou non.

Code source des différents sprints dans tous leurs release:

Sprint 1 release:

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

typedef enum { FAUX = 0, VRAI = 1 } Booleen;
Booleen EchoActif = FAUX;

// Messages emis par les instructions
-----
#define MSG_DEVELOPPE "## nouvelle specialite \"%s\" ; cout  
horaire \"%d\"\\n"
#define MSG_INTERRUPT "## fin de programme\\n"
#define MSG_EMBAUCHE "## nouveau travailleur \"%s\" competent pour  
la specialite \"%s\"\\n"
#define MSG_DEMARCHE "## nouveau client \"%s\"\\n"
#define MSG_COMMANDE "## nouvelle commande \"%s\", par client  
\"%s\"\\n"
#define MSG_TACHE "## la commande \"%s\" requiere la specialite  
\"%s\" (nombre d'heures \"%d\")\\n"
#define MSG_PROGRESSION "## pour la commande \"%s\", pour la  
specialite \"%s\" : \"%d\" heures de plus ont ete realisees\\n"
#define MSG_SPECIALITES "## consultation des specialites\\n"
#define MSG_TRAVAILLEURS "## consultation des travailleurs  
competents pour la specialite \"%s\"\\n"
#define MSG_CLIENT "## consultation des commandes effectuees par  
\"%s\"\\n"
#define MSG_SUPERVISION "## consultation de l'avancement des  
commandes\\n"
#define MSG_CHARGE "## consultation de la charge de travail de  
\"%s\"\\n"
#define MSG_TRAVAILLEURS_TOUS "## consultation des travailleurs  
competents pour chaque specialite\\n"
#define MSG_CLIENT_TOUS "## consultation des commandes effectuees  
par chaque client\\n"
#define MSG_PROGRESSION_PASSE "## une reallocation est requise\\n"

// Lexemes
-----
--
#define LGMOT 35
#define NBCHIFFREMAX 5
```

```

typedef char Mot[LGMOT + 1];
void get_id(Mot id) {
    scanf("%s", id);
    if (EchoActif) printf(">>echo %s\n", id);
}
int get_int() {
    char buffer[NBCHIFFREMAX + 1];
    scanf("%s", buffer);
    if (EchoActif) printf(">>echo %s\n", buffer);
    return atoi(buffer);
}

// Instructions
-----
// developpe -----
void traite_developpe() {
    Mot nom_specialite;
    get_id(nom_specialite);
    int cout_horaire = get_int();
    printf(MSG_DEVELOPPE, nom_specialite, cout_horaire);
}
// interruption -----
void traite_interruption() {
    printf(MSG_INTERRUPTION);
}

// embauche
-----
void traite_embauche() {
    Mot nom_travailleur;
    get_id(nom_travailleur);
    Mot nom_competence;
    get_id(nom_competence);
    printf(MSG_EMBAUCHE, nom_travailleur, nom_competence);
}

// demarche
-----
void traite_demarche() {
    Mot nom_nouveau_client;
    get_id(nom_nouveau_client);
    printf(MSG_DEMARCHE, nom_nouveau_client);
}

// commande
-----
void traite_commande() {
    Mot nom_commande;

```

```

    get_id(nom_commande);
    Mot nom_client;
    get_id(nom_client);
    printf(MSG_COMMANDE, nom_commande, nom_client);
}

// tache
-----
void traite_tache() {
    Mot nom_commande;
    get_id(nom_commande);
    Mot nom_specialite;
    get_id(nom_specialite);
    int nb_heure = get_int();
    printf(MSG_TACHE, nom_commande, nom_specialite, nb_heure);
}

// progression
-----
void traite_progression() {
    Mot nom_commande;
    get_id(nom_commande);
    Mot nom_specialite;
    get_id(nom_specialite);
    int heure_en_plus = get_int();
    printf(MSG_PROGRESSION, nom_commande, nom_specialite,
heure_en_plus);
}

// progression passe
void traite_progression_passe() {
    printf(MSG_PROGRESSION_PASSE);
}

// specialite
-----
void traite_specialites() {
    printf(MSG_SPECIALITES);
}

// travailleur
-----
void traite_travailleurs() {
    Mot nom_specialite;
    get_id(nom_specialite);
    if(strcmp(nom_specialite, "tous") == 0)
        printf(MSG_TRAVAILLEURS_TOUS);
    else
        printf(MSG_TRAVAILLEURS, nom_specialite);
}

```



```

}

// client
-----
void traite_client() {
    Mot nom_client;
    get_id(nom_client);
    if(strcmp(nom_client, "tous") == 0)
        printf(MSG_CLIENT_TOUS);
    else
        printf(MSG_CLIENT, nom_client);
}

// supervision
-----
void traite_supervision() {
    printf(MSG_SUPERVISION);
}

// charge
-----
void traite_charge() {
    Mot charge_travail;
    get_id(charge_travail);
    printf(MSG_CHARGE, charge_travail);
}

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

        if (strcmp(buffer, "interruption") == 0) {
            traite_interruption();
            break;
        }

        if (strcmp(buffer, "embauche") == 0) {
            traite_embauche();
        }
    }
}

```

```

        continue;
    }

    if (strcmp(buffer, "demarche") == 0) {
        traite_demarche();
        continue;
    }

    if (strcmp(buffer, "commande") == 0) {
        traite_commande();
        continue;
    }

    if (strcmp(buffer, "tache") == 0) {
        traite_tache();
        continue;
    }

    if (strcmp(buffer, "progression") == 0) {
        traite_progression();
        continue;
    }

    if (strcmp(buffer, "specialites") == 0) {
        traite_specialites();
        continue;
    }

    if (strcmp(buffer, "travailleurs") == 0) {
        traite_travailleurs();
        continue;
    }

    if (strcmp(buffer, "client") == 0) {
        traite_client();
        continue;
    }

    if (strcmp(buffer, "supervision") == 0) {
        traite_supervision();
        continue;
    }

    if (strcmp(buffer, "charge") == 0) {
        traite_charge();
        continue;
    }

```

```
    if (strcmp(buffer, "passe") == 0) {  
        traite_progression_passe();  
        continue;  
    }  
  
    printf("!!! instruction inconnue >%s< !!!\n", buffer);  
}  
return 0;  
}
```

Comparaison diffchecker du sprint 1 release:

The two files are identical

Editor


Clear Save Diff Share

Original Text


Changed Text

1 ## nouvelle specialite "reseau" ; cout horaire "18"
2 ## nouvelle specialite "graphisme" ; cout horaire "13"
3 ## consultation des specialites
4 ## nouveau travailleur "Toto" competent pour la specialite "reseau"
5 ## nouveau travailleur "Titi" competent pour la specialite "graphisme"
6 ## nouveau travailleur "Toto" competent pour la specialite "graphisme"
7 ## consultation des travailleurs competents pour la specialite "reseau"
8 ## consultation des travailleurs competents pour la specialite "graphisme"
9 ## consultation des travailleurs competents pour chaque specialite
10 ## nouveau client "Roger"
11 ## nouveau client "Rodgio"
12 ## consultation des commandes effectuees par "Roger"
13 ## consultation des commandes effectuees par "Rodgio"
14 ## consultation des commandes effectuees par chaque client
15 ## nouvelle commande "superProduit", par client "Roger"
16 ## nouvelle commande "produitTop", par client "Rodgio"

1 ## nouvelle specialite "reseau" ; cout horaire "18"
2 ## nouvelle specialite "graphisme" ; cout horaire "13"
3 ## consultation des specialites
4 ## nouveau travailleur "Toto" competent pour la specialite "reseau"
5 ## nouveau travailleur "Titi" competent pour la specialite "graphisme"
6 ## nouveau travailleur "Toto" competent pour la specialite "graphisme"
7 ## consultation des travailleurs competents pour la specialite "reseau"
8 ## consultation des travailleurs competents pour la specialite "graphisme"
9 ## consultation des travailleurs competents pour chaque specialite
10 ## nouveau client "Roger"
11 ## nouveau client "Rodgio"
12 ## consultation des commandes effectuees par "Roger"
13 ## consultation des commandes effectuees par "Rodgio"
14 ## consultation des commandes effectuees par chaque client
15 ## nouvelle commande "superProduit", par client "Roger"
16 ## nouvelle commande "produitTop", par client "Rodgio"

 Diffchecker Desktop
Run Diffchecker offline, on your computer, with more features!

CHECK IT OUT

 Bibcitation
By Diffchecker: A free online tool to generate citations, reference lists, and bibliographies.
APA, MLA, Chicago, and more.

CHECK IT OUT

Find Difference

Sprint 2 release:

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

typedef enum { FAUX = 0, VRAI = 1 } Booleen;
Booleen EchoActif = FAUX;

// Messages emis par les instructions
-----
#define MSG_INTERRUPTION "## fin de programme\n"
#define MSG_COMMANDE "## nouvelle commande \"%s\", par client\n\"%s\"\n"
#define MSG_TACHE "## la commande \"%s\" requiere la specialite\n\"%s\" (nombre d'heures \"%d\")\n"
#define MSG_PROGRESSION "## pour la commande \"%s\", pour la specialite \"%s\" : \"%d\" heures de plus ont ete realisees\n"
#define MSG_SUPERVISION "## consultation de l'avancement des commandes\n"
#define MSG_CHARGE "## consultation de la charge de travail de\n\"%s\"\n"
#define MSG_PROGRESSION_PASSE "## une reallocation est requise\n"

// Lexemes
-----
--
#define LGMOT 35
#define NBCHIFFREMAX 5
typedef char Mot[LGMOT + 1];

void get_id(Mot id) {
    scanf("%s", id);
    if (EchoActif) printf(">>echo %s\n", id);
}

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

//Specialites
#define MAX_SPECIALITES 10

typedef struct {
```

```

    Mot nom;
    int cout_horaire;
}Specialite;

typedef struct {
    Specialite tab_specialites[MAX_SPECIALITES];
    unsigned int nb_specialites;
}Specialites;

//Travailleurs
#define MAX_TRAVAILLEURS 50

typedef struct {
    Mot nom;
    Booleen tags_competences[MAX_SPECIALITES];
}Travailleur;

typedef struct {
    Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
    unsigned int nb_travailleurs;
}Travailleurs;

//Client
#define MAX_CLIENTS 10

typedef struct {
    Mot tab_clients[MAX_CLIENTS];
    unsigned int nb_clients;
}Clients;

// Instructions
-----
// developpe -----
void traite_developpe(Specialites* tabSpecialites) {

    Mot nom_specialite;
    get_id(nom_specialite);
    int cout_horaire = get_int();

    strcpy(tabSpecialites->tab_specialites[tabSpecialites->nb_specialit
es].nom, nom_specialite);

    tabSpecialites->tab_specialites[tabSpecialites->nb_specialites].cou
t_horaire = cout_horaire;
    tabSpecialites->nb_specialites++;
}

// interruption -----
void traite_interruption() {

```

```

    printf(MSG_INTERRUPTION);
}

// embauche
-----
void traite_embauche(Specialites* tabSpecialites, Travailleurs*
tabTravail) {
    Mot nom_travailleur, nom_competence;
    get_id(nom_travailleur);
    get_id(nom_competence);

    unsigned int count;

    for (count = 0; count < tabSpecialites->nb_specialites; count++)
    {
        if (strcmp(tabSpecialites->tab_specialites[count].nom,
nom_competence) == 0)
            break;
    }

    unsigned int j;
    for (j = 0; j < MAX_TRAVAILLEURS; j++) {
        if (strcmp(nom_travailleur,
tabTravail->tab_travailleurs[j].nom) == 0) {
            tabTravail->tab_travailleurs[j].tags_competences[count]
= VRAI;
            return;
        }
    }

    strcpy(tabTravail->tab_travailleurs[tabTravail->nb_travailleurs].no
m, nom_travailleur);

    tabTravail->tab_travailleurs[tabTravail->nb_travailleurs].tags_comp
etences[count] = VRAI;
    ++tabTravail->nb_travailleurs;
}

// demarche
-----
void traite_demarche(Clients* clientele) {
    Mot nom_nouveau_client;
    get_id(nom_nouveau_client);
    strcpy(clientele->tab_clients[clientele->nb_clients],
nom_nouveau_client);
    clientele->nb_clients += 1;
}

```

```

// commande
-----
void traite_commande() {
    Mot nom_commande;
    get_id(nom_commande);
    Mot nom_client;
    get_id(nom_client);
    printf(MSG_COMMANDE, nom_commande, nom_client);
}

// tache
-----
void traite_tache() {
    Mot nom_commande;
    get_id(nom_commande);
    Mot nom_specialite;
    get_id(nom_specialite);
    int nb_heure = get_int();
    printf(MSG_TACHE, nom_commande, nom_specialite, nb_heure);
}

// progression
-----
void traite_progression() {
    Mot nom_commande;
    get_id(nom_commande);
    Mot nom_specialite;
    get_id(nom_specialite);
    int heure_en_plus = get_int();
    printf(MSG_PROGRESSION, nom_commande, nom_specialite,
heure_en_plus);
}

// progression passe
void traite_progression_passe() {
    printf(MSG_PROGRESSION_PASSE);
}

// specialite
-----
void traite_specialites(Specialites* tab_spe) {
    printf("specialites traitees : ");
    unsigned int i = 0;
    for (; i < tab_spe->nb_specialites; ++i) {
        if (i == tab_spe->nb_specialites - 1) {
            printf("%s/%d\n", tab_spe->tab_specialites[i].nom,
tab_spe->tab_specialites[i].cout_horaire);
        }
    }
}

```



```

        else {
            printf("%s/%d, ", tab_spe->tab_specialites[i].nom,
tab_spe->tab_specialites[i].cout_horaire);
        }
    }
}

// travailleur
-----
void traite_travailleurs(Travailleurs* tab_travailleurs,
Specialites* tab_specialites) {
    Mot nom_specialite;
    get_id(nom_specialite);
    if (strcmp(nom_specialite, "tous") == 0) {
        unsigned int i;
        for (i = 0; i < tab_specialites->nb_specialites; ++i) {
            printf("la specialite %s peut etre prise en charge par
: ", tab_specialites->tab_specialites[i].nom);
            unsigned int k;
            int count = 0;
            for (k = 0; k < tab_travailleurs->nb_travailleurs; ++k)
            {
                if
(tab_travailleurs->tab_travailleurs[k].tags_competences[i] == VRAI)
            {
                if (count > 0) {
                    printf(", %s",
tab_travailleurs->tab_travailleurs[k].nom);
                    count++;
                }
                else if(count == 0) {
                    printf("%s",
tab_travailleurs->tab_travailleurs[k].nom);
                    count++;
                }
            }
        }
        printf("\n");
    }
    else {
        unsigned int c;
        int count = 0;
        printf("la specialite %s peut etre prise en charge par : ",
nom_specialite);
        for (c = 0; c < tab_specialites->nb_specialites; ++c) {
            if (strcmp(tab_specialites->tab_specialites[c].nom,
nom_specialite) == 0) {

```

```

        break;
    }
}

unsigned int m;
for (m = 0; m < tab_travailleurs->nb_travailleurs; ++m) {
    if
(tab_travailleurs->tab_travailleurs[m].tags_competences[c] == VRAI)
{
    if (count > 0) {
        printf(", %s",
tab_travailleurs->tab_travailleurs[m].nom);
        count++;
    }
    else if (count == 0) {
        printf("%s",
tab_travailleurs->tab_travailleurs[m].nom);
        count++;
    }
}
}

```

```

    }
    printf("\n");
}
}

```

```

// client
-----
void traite_client(Clients* clientele) {
    Mot nom_client;
    get_id(nom_client);
    if (strcmp(nom_client, "tous") == 0)
    {
        for (int count = 0; count < clientele->nb_clients; count++)
            printf("le client %s a commande : \n",
clientele->tab_clients[count]);
    }
    else
        printf("le client %s a commande : \n", nom_client);
}

```

```

// supervision
-----
void traite_supervision() {
    printf(MSG_SUPERVISION);
}

```

```

// charge
-----

```

```

void traite_charge() {
    Mot charge_travail;
    get_id(charge_travail);
    printf(MSG_CHARGE, charge_travail);
}

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

    Specialites tabSpecialites;
    tabSpecialites.nb_specialites = 0;
    Travailleurs tabTravail;
    tabTravail.nb_travailleurs = 0;
    Clients clientele;
    clientele.nb_clients = 0;

    if (argc >= 2 && strcmp("echo", argv[1]) == 0) {
        EchoActif = VRAI;
    }
    Mot buffer;
    while (VRAI) {
        get_id(buffer);
        if (strcmp(buffer, "developpe") == 0) {
            traite_developpe(&tabSpecialites);
            continue;
        }

        if (strcmp(buffer, "interruption") == 0) {
            traite_interruption();
            break;
        }

        if (strcmp(buffer, "embauche") == 0) {
            traite_embauche(&tabSpecialites, &tabTravail);
            continue;
        }

        if (strcmp(buffer, "demarche") == 0) {
            traite_demarche(&clientele);
            continue;
        }

        if (strcmp(buffer, "commande") == 0) {
            traite_commande();
            continue;
        }
    }
}

```

```

        if (strcmp(buffer, "tache") == 0) {
            traite_tache();
            continue;
        }

        if (strcmp(buffer, "progression") == 0) {
            traite_progression();
            continue;
        }

        if (strcmp(buffer, "specialites") == 0) {
            traite_specialites(&tabSpecialites);
            continue;
        }

        if (strcmp(buffer, "travailleurs") == 0) {
            traite_travailleurs(&tabTravail, &tabSpecialites);
            continue;
        }

        if (strcmp(buffer, "client") == 0) {
            traite_client(&clientele);
            continue;
        }

        if (strcmp(buffer, "supervision") == 0) {
            traite_supervision();
            continue;
        }

        if (strcmp(buffer, "charge") == 0) {
            traite_charge();
            continue;
        }

        if (strcmp(buffer, "passe") == 0) {
            traite_progression_passe();
            continue;
        }

        printf("!!! instruction inconnue >%s< !!!\n", buffer);
    }
    return 0;
}

```

Comparaison diffchecker du sprint 2 release:

The two files are identical

Editor


Clear Save Diff Share


Original Text

Changed Text

```
1 specialites traitees : reseau/18, graphisme/13
2 la specialite reseau peut etre prise en charge par : Toto
3 la specialite graphisme peut etre prise en charge par : Toto, Titi
4 la specialite reseau peut etre prise en charge par : Toto
5 la specialite graphisme peut etre prise en charge par : Toto, Titi
6 le client Roger a commande :
7 le client Rodgio a commande :
8 le client Roger a commande :
9 le client Rodgio a commande :
10 ## nouvelle commande "superProduit", par client "Roger"
11 ## nouvelle commande "produitTop", par client "Rodgio"
12 le client Roger a commande :
13 le client Rodgio a commande :
14 le client Roger a commande :
15 le client Rodgio a commande :
16 ## consultation de l'avancement des commandes
```

```
1 specialites traitees : reseau/18, graphisme/13
2 la specialite reseau peut etre prise en charge par : Toto
3 la specialite graphisme peut etre prise en charge par : Toto, Titi
4 la specialite reseau peut etre prise en charge par : Toto
5 la specialite graphisme peut etre prise en charge par : Toto, Titi
6 le client Roger a commande :
7 le client Rodgio a commande :
8 le client Roger a commande :
9 le client Rodgio a commande :
10 ## nouvelle commande "superProduit", par client "Roger"
11 ## nouvelle commande "produitTop", par client "Rodgio"
12 le client Roger a commande :
13 le client Rodgio a commande :
14 le client Roger a commande :
15 le client Rodgio a commande :
16 ## consultation de l'avancement des commandes
```

 **Diffchecker Desktop**
Run Diffchecker offline, on your computer, with more features!

 **Bibcitation**
By Diffchecker: A free online tool to generate citations, reference lists, and bibliographies.
APA, MLA, Chicago, and more.

Find Difference

Check It Out

Check It Out

Sprint 4 release:

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

typedef enum { FAUX = 0, VRAI = 1 } Booleen;
Booleen EchoActif = FAUX;

// Messages emis par les instructions
-----
#define MSG_INTERRUPT "## fin de programme"

// Lexemes
-----
--
#define LGMOT 35
#define NBCHIFFREMAX 5
typedef char Mot[LGMOT + 1];

void get_id(Mot id) {
    scanf("%s", id);
    if (EchoActif) printf(">>echo %s\n", id);
}

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

// DONNEES
-----

// Specialites
#define MAX_SPECIALITES 10

typedef struct {
    Mot nom;
    int cout_horaire;
} Specialite;

typedef struct {
    Specialite tab_specialites[MAX_SPECIALITES];
    unsigned int nb_specialites;
} Specialites;
```

```

// Travaillleurs
#define MAX_TRAVAILLEURS 50

typedef struct {
    Mot nom;
    Booleen tags_competences[MAX_SPECIALITES];
    unsigned int nb_heures_restantes;
} Travailleur;

typedef struct {
    Travailleur tab_travailleurs[MAX_TRAVAILLEURS];
    unsigned int nb_travailleurs;
} Travaillleurs;

// Client
#define MAX_CLIENTS 10

typedef struct {
    Mot tab_clients[MAX_CLIENTS];
    unsigned int nb_clients;
} Clients;

// Commande
#define MAX_COMMANDES 500

typedef struct {
    unsigned int nb_heures_requises;
    unsigned int nb_heures_effectuees;
    unsigned int travailleur_st;
} Tache;
typedef struct {
    Mot nom;
    unsigned int idx_client;
    Tache taches_spe[MAX_SPECIALITES];
} Commande;
typedef struct {
    Commande tab_commandes[MAX_COMMANDES];
    unsigned int nb_commandes;
} Commandes;

// Instructions
-----
// Developpe
// Fonction qui prend en paramètre la structure tabSpecialites de
type Specialites
void traite_developpe(Specialites* tabSpecialites) {

```

```

    Mot nom_specialite;
    get_id(nom_specialite);
    int cout_horaire = get_int();
    if (tabSpecialites->nb_specialites < MAX_SPECIALITES)
    {

strcpy(tabSpecialites->tab_specialites[tabSpecialites->nb_specialit
es].nom, nom_specialite);

tabSpecialites->tab_specialites[tabSpecialites->nb_specialites].cou
t_horaire = cout_horaire;
        tabSpecialites->nb_specialites += 1;
    }
    else
    {
        printf("Le nombre de specialite a atteint son maximum");
    }
}

// Interruption
void traite_interruption() {
    printf(MSG_INTERRUPTION);
}

// Embauche
/*
* Cette fonction sert à s'occuper des différentes embauches pour
les
* travailleurs,
* afin d'embaucher ces derniers de manière convenable selon s'ils
ont déjà été
* embauché ou non
* */
void traite_embauche(Specialites tabSpecialites, Travailleurs*
tabTravail) {
    Mot nom_travailleur, nom_competence;
    get_id(nom_travailleur);
    get_id(nom_competence);
    for (int count = 0; count < MAX_TRAVAILLEURS; ++count)
    {
        if (strcmp(nom_travailleur,
tabTravail->tab_travailleurs[count].nom) == 0)
        {
            int i;
            for (i = 0; i < tabSpecialites.nb_specialites; ++i)
            {
                if (strcmp(nom_competence,

```



```

tabSpecialites.tab_specialites[i].nom) == 0)
{

tabTravail->tab_travailleurs[count].tags_competences[i] = VRAI;
return;
}
}

}
}

strcpy(tabTravail->tab_travailleurs[tabTravail->nb_travailleurs].no
m,
nom_travailleur);

tabTravail->tab_travailleurs[tabTravail->nb_travailleurs].nb_heures
restantes = 0;
for (int count = 0; count < MAX_SPECIALITES; ++count)
{
tabTravail->tab_travailleurs[tabTravail->nb_travailleurs].
tags_competences[count] = FAUX;
}
for (int i = 0; i < tabSpecialites.nb_specialites; ++i)
{
if (strcmp(nom_competence,
tabSpecialites.tab_specialites[i].nom) == 0)
{

tabTravail->tab_travailleurs[tabTravail->nb_travailleurs].
tags_competences[i] = VRAI;
break;
}
}
tabTravail->nb_travailleurs += 1;

}

// Demarche
/*
* Cette fonction prend comme paramètre la structure clientele de
type Clients,
* elle sert à donner à la variable nom_nouveau_client le nom du
client
* donné dans le Input du programme, que l'on retrouve dans
* clientele->tab_clients[clientele->nb_clients] */
void traite_demarche(Clients* clientele) {
Mot nom_nouveau_client;
get_id(nom_nouveau_client);

```

```

    strcpy(clientele->tab_clients[clientele->nb_clients],
nom_nouveau_client);
    clientele->nb_clients += 1;
}

// Commande
void traite_commande(Clients clientele, Commandes* tabCommande) {
    Mot nom_commande, nom_client_cmd;
    get_id(nom_commande);
    get_id(nom_client_cmd);
    for (int i = 0; i < MAX_SPECIALITES; ++i)
    {
        tabCommande->tab_commandes[tabCommande->nb_commandes].
taches_spe[i].nb_heures_requises = 0;
        tabCommande->tab_commandes[tabCommande->nb_commandes].
taches_spe[i].nb_heures_effectuees = 0;
    }
    for (int count = 0; count < MAX_CLIENTS; ++count)
    {
        if (strcmp(nom_client_cmd, clientele.tab_clients[count]) ==
0)
        {

strcpy(tabCommande->tab_commandes[tabCommande->nb_commandes].nom,
        nom_commande);

tabCommande->tab_commandes[tabCommande->nb_commandes].idx_client
        = count;
        tabCommande->nb_commandes += 1;
        break;
    }
}

// Tache
void traite_tache(Specialites tabSpecialites, Commandes*
tabCommande,
        Travailleurs* tabTravail) {
    Mot nom_tache, nom_specialite;
    get_id(nom_tache);
    get_id(nom_specialite);
    int nb_heure = get_int();
    int sp = 0;
    int cm = 0;
    int count = 0;
    int j = count;
    unsigned int travailleur_st = 0;
    unsigned int min_heure_travail = 0;

```

```

while (sp < tabSpecialites.nb_specialites)
{
    if (strcmp(tabSpecialites.tab_specialites[sp].nom,
nom_specialite) == 0)
    {
        break;
    }
    ++sp;
}
while (cm < tabCommande->nb_commandes)
{
    if (strcmp(tabCommande->tab_commandes[cm].nom, nom_tache) ==
0)
    {

tabCommande->tab_commandes[cm].taches_spe[sp].nb_heures_requises =
nb_heure;
        break;
    }
    ++cm;
}
while (count < tabTravail->nb_travailleurs - 1)
{
    if (tabTravail->tab_travailleurs[count].tags_competences[sp]
== VRAI)
    {
        min_heure_travail = tabTravail->tab_travailleurs[count].
nb_heures_restantes;
        break;
    }
    ++count;
}
while (j < tabTravail->nb_travailleurs)
{
    if (tabTravail->tab_travailleurs[j].tags_competences[sp] ==
VRAI)
    {
        if (tabTravail->tab_travailleurs[j].nb_heures_restantes
<
min_heure_travail)
        {
            min_heure_travail = tabTravail->tab_travailleurs[j].
nb_heures_restantes;
            travailleur_st = j;
        }
    }
    ++j;
}

```

```

    }
    tabCommande->tab_commandes[cm].taches_spe[sp].travailleur_st =
        travailleur_st;
    tabTravail->tab_travailleurs[travailleur_st].nb_heures_restantes
+= nb_heure;

}

// Progression
void traite_progression(Specialites tabSpecialites, Commandes*
tabCommande,
                        Travailleurs* tabTravail) {
    Mot progression_cm, progression_spe;
    get_id(progression_cm);
    get_id(progression_spe);
    int nb_heure_realise = get_int();
    int count = 0;
    int i = 0;

    while (count < tabSpecialites.nb_specialites)
    {
        if (strcmp(tabSpecialites.tab_specialites[count].nom,
                    progression_spe) == 0)
        {
            break;
        }
        ++count;
    }

    while (i < tabCommande->nb_commandes)
    {
        if (strcmp(tabCommande->tab_commandes[i].nom,
                    progression_cm) == 0)
        {
            tabCommande->tab_commandes[i].taches_spe[count].
            nb_heures_effectuees += nb_heure_realise;

            tabTravail->tab_travailleurs[tabCommande->tab_commandes[i].
            taches_spe[count].travailleur_st].nb_heures_restantes
            -= nb_heure_realise;
        }
        ++i;
    }
}

// Progression Passe
void traite_progression_passe() {
}

// Specialites

```

```

void traite_specialites(Specialites tabSpecialites) {
    printf("specialites traitees : ");
    unsigned int count = 0;
    for (; count < tabSpecialites.nb_specialites; ++count)
    {
        if (count == tabSpecialites.nb_specialites - 1)
            printf("%s/%d\n",
tabSpecialites.tab_specialites[count].nom,

tabSpecialites.tab_specialites[count].cout_horaire);
        else
            printf("%s/%d, ",
tabSpecialites.tab_specialites[count].nom,

tabSpecialites.tab_specialites[count].cout_horaire);
    }
}

// Travailleurs
/*
* Ctte fonction sert à assigner chaque travailleurs à leur
spécialité
* */
void traite_travailleurs(Specialites tabSpecialites, Travailleurs*
toutTravailleur) {
    Mot travailleur_diff;
    get_id(travailleur_diff);

    if (strcmp(travailleur_diff, "tous") == 0)
    {
        for (int count = 0; count < tabSpecialites.nb_specialites;
++count)
        {
            int k = 0;
            printf("la specialite %s peut etre prise en charge par
: ",
tabSpecialites.tab_specialites[count].nom);
            for (int i = 0; i < toutTravailleur->nb_travailleurs;
++i)
            {
                if (toutTravailleur->tab_travailleurs[i]
.tags_competences[count] == VRAI)
                {
                    if (k == 0)
                    {
                        printf("%s",
toutTravailleur->tab_travailleurs[i].nom);
                        k += 1;

```

```

    }
    else if (k > 0 && k <
toutTravailleur->nb_travailleurs)
    {
        printf(", %s",
toutTravailleur->tab_travailleurs[i].nom);
        k += 1;
    }
    }
    }
    printf("\n");
}
}
else
{
    printf("la specialite %s peut etre prise en charge par : ",
travailleur_diff);
    int i;
    int virgule = 0;
    for (i = 0; i < tabSpecialites.nb_specialites; ++i)
    {
        if (strcmp(travailleur_diff,
tabSpecialites.tab_specialites[i].nom)
== 0)
        {
            break;
        }
    }
    for (int count = 0; count <
toutTravailleur->nb_travailleurs; ++count)
    {
        if
(toutTravailleur->tab_travailleurs[count].tags_competences[i] ==
VRAI)
        {
            if (virgule == 0)
            {
                printf("%s",
toutTravailleur->tab_travailleurs[count].nom);
                virgule += 1;
            }
            else if (virgule > 0)
            {
                printf(", %s",
toutTravailleur->tab_travailleurs[count].nom);
                virgule += 1;
            }
        }
    }
}

```

```

    }
    printf("\n");
}
}

// Client
/*
 * Cette fonction répoertorie les commandes des clients
 */
void traite_client(Clients clientele, Commandes* tabCommande) {
    Mot nom_client;
    get_id(nom_client);

    if (strcmp(nom_client, "tous") == 0)
    {
        for (unsigned int count = 0; count < clientele.nb_clients;
        ++count)
        {
            printf("le client %s a commande : ",
            clientele.tab_clients[count]);
            int virgule = 0;
            for (unsigned int i = 0; i < tabCommande->nb_commandes;
            ++i)
            {
                if (tabCommande->tab_commandes[i].idx_client ==
                count)
                {
                    if (virgule == 0)
                    {
                        printf("%s",
                        tabCommande->tab_commandes[i].nom);
                        virgule += 1;
                    }
                    else if (virgule > 0)
                    {
                        printf(", %s",
                        tabCommande->tab_commandes[i].nom);
                        virgule += 1;
                    }
                }
            }
            printf("\n");
        }
    }
    else
    {
        printf("le client %s a commande : ", nom_client);
    }
}

```

```

    int v = 0;
    int count;
    for(count = 0; count < clientele.nb_clients; count++) {
        if (strcmp(nom_client, clientele.tab_clients[count]) ==
0) {
            for (unsigned int i = 0; i <
tabCommande->nb_commandes; ++i) {
                if (tabCommande->tab_commandes[i].idx_client ==
count) {
                    if (v == 0) {
                        printf("%s",
tabCommande->tab_commandes[i].nom);
                        v += 1;
                    } else if (v > 0) {
                        printf(", %s",
tabCommande->tab_commandes[i].nom);
                        v += 1;
                    }
                }
            }
            printf("\n");
        }
    }
}

```

```

// Supervision
/*Cette commande répertorie l'état des tâches selon la commande
effectué,
* le nombre d'heure effectué ou requise et la spécialité.
* */
void traite_supervision(Commandes tabCommande, Specialites
tabSpecialites) {

    for (unsigned int count = 0; count < tabCommande.nb_commandes;
++count)
    {
        int cpt = 0;
        if (tabCommande.nb_commandes != 0)
        {
            printf("etat des taches pour %s : ", tabCommande.
tab_commandes[count].nom);
            for (int i = 0; i < MAX_SPECIALITES; ++i)
            {
                if (tabCommande.tab_commandes[count].taches_spe[i].
nb_heures_requises != 0 && cpt > 0)
                {
                    printf(", %s:%d/%d",

```



```

tabSpecialites.tab_specialites[i].nom,

tabCommande.tab_commandes[count].taches_spe[i].
    nb_heures_effectuees,

tabCommande.tab_commandes[count].taches_spe[i]
    .nb_heures_requises);
    cpt += 1;
}
else if
(tabCommande.tab_commandes[count].taches_spe[i].nb_heures_requises
!= 0 && cpt == 0)
{
    printf("%s:%d/%d",
tabSpecialites.tab_specialites[i].nom,

tabCommande.tab_commandes[count].taches_spe[i].
    nb_heures_effectuees,

tabCommande.tab_commandes[count].taches_spe[i].
    nb_heures_requises);
    cpt += 1;
}
}
}
printf("\n");
}
}
}
// Charge
/*
* Cette fonction permet de savoir la charge de travail effectué
par le travailleur,
* elle prend donc en paramètre la spécialité, la commande effectué
et le travailleur.
*
* */
void traite_charge(Specialites tabSpecialites, Commandes*
tabCommande,
    Travailleurs* tabTravail) {

    Mot charge_travail;
    get_id(charge_travail);
    int j;
    for (j = 0; j < tabTravail->nb_travailleurs; ++j)
    {
        if (strcmp(tabTravail->tab_travailleurs[j].nom,
charge_travail) == 0)
        {

```

```

        break;
    }
}

int v = 0;
printf("charge de travail pour %s : ",
tabTravail->tab_travailleurs[j].nom);
for (int i = 0; i < tabCommande->nb_commandes; ++i)
{
    for (int k = 0; k < tabSpecialites.nb_specialites; ++k)
    {
        if
(tabCommande->tab_commandes[i].taches_spe[k].travailleur_st == j) {
            if
((tabCommande->tab_commandes[i].taches_spe[k].nb_heures_requises -
tabCommande->tab_commandes[i].taches_spe[k].
nb_heures_effectuees) != 0) {
                if (v == 0) {
                    printf("%s/%s/%dheure(s)",
tabCommande->tab_commandes[i].nom,
tabSpecialites.tab_specialites[k].nom,
(tabCommande->tab_commandes[i].taches_spe[k].
nb_heures_requises -
tabCommande->tab_commandes[i].taches_spe[k].
nb_heures_effectuees));
                }
                else if (v > 0) {
                    printf(", %s/%s/%dheure(s)",
tabCommande->tab_commandes[i].nom,
tabSpecialites.tab_specialites[k].nom,
(tabCommande->
tab_commandes[i].taches_spe[k].nb_heures_requises -
tabCommande->tab_commandes[i].taches_spe[k].
nb_heures_effectuees));
                }
                v += 1;
            }
        }
    }
}

printf("\n");
}

```

```

//Boucle principale
-----
int main(int argc, char* argv[])
{
    /*
     * Initialisation des différentes variables nécessaires dans
     l'ensemble du code,
     * à savoir des variables que l'on va utiliser dans différentes
     fonctions .
     * */
    Specialites tabSpecialites;
    tabSpecialites.nb_specialites = 0;
    Clients clientele;
    clientele.nb_clients = 0;
    Travailleurs nouveau_travailleur;
    nouveau_travailleur.nb_travailleurs = 0;
    Commandes tabCommande;
    tabCommande.nb_commandes = 0;
    tabCommande.tab_commandes->idx_client = 501;

    if (argc >= 2 && strcmp("echo", argv[1]) == 0) {
        EchoActif = VRAI;
    }
    Mot buffer;
    /*
     * Forêt de "if" pour naviguer d'une fonction à une autre selon
     l'input donné.
     * */
    while (VRAI) {
        get_id(buffer);
        if (strcmp(buffer, "developpe") == 0) {
            traite_developpe(&tabSpecialites);
            continue;
        }
        if (strcmp(buffer, "interruption") == 0) {
            traite_interruption();
            break;
        }
        if (strcmp(buffer, "embauche") == 0) {
            traite_embauche(tabSpecialites, &nouveau_travailleur);
            continue;
        }
        if (strcmp(buffer, "demarche") == 0) {
            traite_demarche(&clientele);
            continue;
        }
        if (strcmp(buffer, "commande") == 0) {
            traite_commande(clientele, &tabCommande);

```

```

        continue;
    }
    if (strcmp(buffer, "tache") == 0) {
        traite_tache(tabSpecialites, &tabCommande,
&nouveau_travailleur);
        continue;
    }
    if (strcmp(buffer, "progression") == 0) {
        traite_progression(tabSpecialites, &tabCommande,
&nouveau_travailleur);
        continue;
    }
    if (strcmp(buffer, "passe") == 0) {
        traite_progression_passe();
        continue;
    }
    if (strcmp(buffer, "specialites") == 0) {
        traite_specialites(tabSpecialites);
        continue;
    }
    if (strcmp(buffer, "travailleurs") == 0) {
        traite_travailleurs(tabSpecialites,
&nouveau_travailleur);
        continue;
    }
    if (strcmp(buffer, "client") == 0) {
        traite_client(clientele, &tabCommande);
        continue;
    }
    if (strcmp(buffer, "supervision") == 0) {
        traite_supervision(tabCommande, tabSpecialites);
        continue;
    }
    if (strcmp(buffer, "charge") == 0) {
        traite_charge(tabSpecialites, &tabCommande,
&nouveau_travailleur);
        continue;
    }
    printf("!!! instruction inconnue >%s< !!!\n", buffer);
    return 0;
}
}

```

Comparaison diffchecker du sprint 4 release:

The two files are identical

Editor

Original Text

Changed Text

1 specialites traitees : reseau/18, graphisme/13

2 la specialite reseau peut etre prise en charge par : Toto

3 la specialite graphisme peut etre prise en charge par : Toto, Titi

4 le client Roger a commande :

5 le client Rodgio a commande :

6 le client Roger a commande : superProduit, megaProduit

7 le client Rodgio a commande : produITop, produIThyper

8 etat des taches pour superProduit :

9 etat des taches pour produITop :

10 etat des taches pour megaProduit :

11 etat des taches pour produIThyper :

12 charge de travail pour Toto : superProduit/reseau/45heure(s)

13 charge de travail pour Titi :

14 charge de travail pour Toto : superProduit/reseau/45heure(s)

15 charge de travail pour Titi : superProduit/graphisme/32heure(s)

16 charge de travail pour Toto : superProduit/reseau/45heure(s)

1 specialites traitees : reseau/18, graphisme/13

2 la specialite reseau peut etre prise en charge par : Toto

3 la specialite graphisme peut etre prise en charge par : Toto, Titi

4 le client Roger a commande :

5 le client Rodgio a commande :

6 le client Roger a commande : superProduit, megaProduit

7 le client Rodgio a commande : produITop, produIThyper

8 etat des taches pour superProduit :

9 etat des taches pour produITop :

10 etat des taches pour megaProduit :

11 etat des taches pour produIThyper :


12 charge de travail pour Toto : superProduit/reseau/45heure(s)

13 charge de travail pour Titi :

14 charge de travail pour Toto : superProduit/reseau/45heure(s)


15 charge de travail pour Titi : superProduit/graphisme/32heure(s)

16 charge de travail pour Toto : superProduit/reseau/45heure(s)

Diffchecker Desktop

Run Diffchecker offline, on your computer, with more features!

CHECK IT OUT

Bibcitation

By Diffchecker: A free online tool to generate citations, reference lists, and bibliographies. APA, MLA, Chicago, and more.

CHECK IT OUT

Find Difference

37

Voici donc ce rapport terminé, nous voulons vous remercier de l'avoir lu.

Sofiane Oulad Itto
Ahmed El Samny
Groupe 111.