Université de Technologie de Compiègne – IA01

Devoir 1

Programmation C

Martin Schneider et Océane Bordeau
A21

Exercice 1

Programme 1

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int A = 20, B = 5, C = -10, D = 2;
    printf("%d \n", A && B || !0 && C++ && !D++);
    printf("c=%d d=%d \n", C, D);
    return 0;
}
```

L'exécution nous renvoie :

```
1
c=-10 d=2
```

On étudie d'abord A && B qui vaut 1 car A et B sont différents de 0. Comme A&&B vaut 1, !0 && C++ && !D++ n'est pas évalué car il se situe de l'autre côté de l'opérateur ||. En effet si nous avons 1 || …, ce qui se trouve du côté droit de l'opérateur || n'est pas évalué. Ainsi l'instruction A&&B || !0 && C++ && !D++ renvoie 1. Comme !0 && C++ && !D++ n'a pas été évalué, C vaut -10 et D vaut 2.

Programme 2

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int p[4] = {1, -2, 3, 4};
    int *q = p;
    printf("c=%d\n", *++q * *q++);
    printf("c=%d \n", *q);
    return 0;
}
```

L'exécution nous renvoie :

```
c=4
c=3
```

*q pointe sur le premier élément du tableau soit 1. Ainsi *++q pointe sur le deuxième élément du tableau soit -2. On multiplie alors le deuxième élément du tableau avec lui-même. Ainsi la première ligne renvoie bien -2 * -2 soit 4. Puis le pointeur est post-incrémenté suite à cette instruction, ainsi *q renverra le troisième élément du tableau soit 3.

Programme 3

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int A = 20, B = 5;
    int C = !--A / ++!B;
    printf("A=%d B=%d c=%d \n", A, B, C);
    return 0;
}
```

Cette fonction renvoie une erreur car on cherche à incrémenter une variable binaire !B.

Programme 4

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int a = -8, b = 3;
    a >>= 2 ^ b;
    printf("a=%d \n", a);
    return 0;
}
```

L'exécution nous renvoie :

```
a=-4
```

A l'initialisation a = -8 = 11000 (en complémentation à 2)

2 ^ b = 2 ^ 3 = 10 ^ 11 = 01

a prend donc la valeur de a décalé de 1 bit à droite. Soit 11000 décalé d'un bit à droite donne 11100 (on complète à gauche avec un 1 car les bits sont signés).

11100 donne -4 en complémentation à 2.

Programme 5

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int p[4] = {1, -2, 3, 4};
    int *q = p;
    int d = *q & *q++ | *q++;
    printf("d=%d \n", d);
    printf("q=%d \n", *q);
    return 0;
}
```

L'exécution nous renvoie :

```
d=-1
q=3
```

A l'initialisation q pointe sur le premier élément du tableau soit 1. *q & *q++ renvoie 1 car il correspond à 1 & 1 soit 1. Puis le pointeur q est post incrémenté ainsi *q vaut -2. On évalue ensuite le |, soit 1 | -2, soit en binaire : 001 | 110. Ce qui donne : 111 en complémentation à cela vaut 001 soit -1. Puis le pointeur q est post incrémenté ainsi *q vaut 3. Les résultats sont donc bien d=-1 et q=3.

Exercice 2

Initialiser les notes des étudiants dans le tableau POINTS

```c
int *initpoints(int N)
{
    int *POINTS;
    POINTS = (int *)malloc(N * sizeof(int));
    for (int i = 0; i < N; i++)
    {
        printf("Quelle est la note de l'etudiant %d \n", i + 1);
        int tmp = -1;
        scanf("%d", &tmp);
        while (tmp < 0 || tmp > 60)
        {
            printf("Veuillez saisir une valeur entre 0 et 60.\n");
            scanf("%d", &tmp);
        }
        POINTS[i] = tmp;
    }
    return (POINTS);
}
```

Cette fonction lit les notes de N étudiants de l'UTC dans un devoir de l'UV SR01 et les mémorise dans un tableau POINTS de dimension N.

Cette fonction prend en paramètre le nombre d'étudiants N et renvoie le tableau des notes. On pense à allouer dynamiquement ce tableau de la taille de N * sizeof(int). Puis pour chaque étudiant on demande sa note entre 0 et 60. On veille à vérifier que la note se situe bien entre 0 et 60.

Note maximale du devoir

```c
int nmax(int *tab, int N)
{
    int max = -1;
    for (int i = 0; i < N; i++)
    {
        if (tab[i] > max)
        {
            max = tab[i];
        }
    }
    return (max);
}
```

Cette fonction nous permettra de renvoyer la note maximale du devoir mais aussi la valeur maximale dans le tableau NOTES pour afficher les graphiques.

On initialise max à -1 de sorte à ce que pour chaque élément du tableau, on vérifie si la valeur est supérieure à la variable max, si oui alors max devient cette nouvelle valeur sinon on passe à la valeur suivante.

On retourne ce max.

Note minimale du devoir

```c
int nmin(int *tab, int N)
{
    int min = 100;
    for (int i = 0; i < N; i++)
    {
        if (tab[i] < min)
        {
            min = tab[i];
        }
    }
    return (min);
}
```

Cette fonction nous permettra de renvoyer la note minimale du devoir.

On initialise la valeur min à 100 (cette valeur doit être supérieure à la plus grande note possible soit 60). Pour chaque élément du tableau, on vérifie si la valeur est inférieure à la variable min, si oui alors min devient cette nouvelle valeur sinon on passe à la valeur suivante.

On retourne ce min.

Moyenne des notes du devoir

```c
float moyenne(int *tab, int N)
{
    float total = 0;
    for (int i = 0; i < N; i++)
    {
        total += tab[i];
    }
    return (total / N);
}
```

Cette fonction retournera la moyenne des notes du devoir.

On initialise un float total qui nous permettra de sommer les notes du devoir. En effet, pour chaque élément du tableau, on ajoute à total la valeur de l'élément actuel du tableau. On retourne ainsi total/N qui est bien un float.

Initialiser le tableau NOTES

```c
int *initnotes(int *POINTS, int N)
{
    int *NOTES;
    NOTES = (int *)malloc(7 * sizeof(int));
    for (int i = 0; i < 7; i++)
    {
        NOTES[i] = 0;
    }

    for (int i = 0; i < N; i++)
    {
        if (POINTS[i] >= 0 && POINTS[i] <= 9)
        {
            NOTES[0]++;
        }
        else if (POINTS[i] >= 10 && POINTS[i] <= 19)
        {
            NOTES[1]++;
        }
        else if (POINTS[i] >= 20 && POINTS[i] <= 29)
        {
            NOTES[2]++;
        }
        else if (POINTS[i] >= 30 && POINTS[i] <= 39)
        {
            NOTES[3]++;
        }
        else if (POINTS[i] >= 40 && POINTS[i] <= 49)
        {
            NOTES[4]++;
        }
        else if (POINTS[i] >= 50 && POINTS[i] <= 59)
        {
            NOTES[5]++;
        }
        else if (POINTS[i] == 60)
        {
            NOTES[6]++;
        }
    }
    return (NOTES);
}
```

Cette fonction nous permet de compter dans chaque intervalle [0 ; 9], [10 ; 19], [20 ; 29], [30 ; 39], [40 ; 49], [50 ; 59], {60}, le nombre de notes du devoir s'y situant.

Ainsi on initialise un tableau NOTES de dimension 7, on initialise chacun de ses éléments à 0. Pour chaque élément du tableau POINTS passé en paramètres on vérifie dans quel intervalle il se situe et on incrémente alors dans la bonne case du tableau correspondant à cet intervalle. On renvoie bien ce tableau NOTES.

Graphique nuage de points

```c
void graph_nuage(int *NOTES)
{
    for (int i = nmax(NOTES, 7); i > 0; i--)
    {
        printf("%d > ", i);
        for (int j = 0; j < 7; j++)
        {
            if (NOTES[j] == i)
            {
                printf("    o    ");
            }
            else
            {
                printf("         ");
            }
        }
        printf("\n");
        printf("\n");
    }
    printf("    +");
    for (int j = 0; j < 7; j++)
    {
        if (NOTES[j] == 0)
        {
            printf("----o---+");
        }
        else
        {
            printf("--------+");
        }
    }
    printf("\n");
    printf("    |  0 - 9 | 10 - 19| 20 - 29| 30 - 39| 40 - 49| 50 - 59|   60   |");
}
```

Cette fonction permet d'afficher à l'aide d'un graphique en nuages le tableau NOTES.

En effet, on affiche x lignes, x représentant la valeur maximale du tableau NOTES. Pour chaque ligne on affiche d'abord son numéro puis on parcourt chaque élément du tableau NOTES, s'il est égal au numéro de la ligne on affiche un « o » sinon on affiche des espaces. Entre chaque ligne, on effectue un retour à la ligne.

Pour la ligne 0, il faut vérifier si un élément du tableau NOTES est égal à zéro, si c'est le cas il faut afficher un « o » sur la ligne en tiret, sinon on affiche seulement les tirets.

Graphique en bâtons

```c
void graph_batons(int *NOTES)
{
    for (int i = nmax(NOTES, 7); i > 0; i--)
    {
        printf("%d > ", i);
        for (int k = 0; k < 7; k++)
        {
            if (NOTES[k] >= i)
            {
                printf(" ####### ");
            }
            else
            {
                printf("         ");
            }
        }
        printf("\n\n");
    }
    printf("    +--------+--------+--------+--------+--------+--------+--------+\n");
    printf("    | 0 - 9  | 10 - 19| 20 - 29| 30 - 39| 40 - 49| 50 - 59|   60   |");
}
```

Cette fonction nous permet d'afficher un graphique en bâtons du tableau NOTES.

Nous résolvons ce problème de la même façon que pour afficher le graphique nuage de points à l'exception que nous n'avons rien à afficher si un élément du tableau NOTES est égal à 0.

Fonction main

```c
int main(int argc, char const *argv[])
{
    int N = -1;
    printf("Combien d'etudiants ont fait le devoir de SR01 ?\n");
    scanf("%d", &N);
    while (N <= 0)
    {
        printf("Le nombre d'étudiants doit etre positif.\n\n");
        scanf("%d", &N);
    }
    int *POINTS = initpoints(N);
    int *NOTES = initnotes(POINTS, N);
    printf("La note maximale du devoir est de %d. \n", nmax(POINTS, N));
    printf("La note minimale du devoir est de %d. \n", nmin(POINTS, N));
    printf("La note moyenne du devoir est de %.2f. \n\n", moyenne(POINTS, N));
    graph_nuage(NOTES);
    graph_batons(NOTES);
    free(POINTS);
    free(NOTES);
    return 0;
}
```

La fonction `main` nous permet de balayer toutes les fonctions programmées précédemment, on demande à l'utilisateur combien d'étudiants ont fait le devoir pour passer ce nombre en paramètres de certaines fonctions.

On peut ainsi visualiser ceci lors de l'exécution du programme.

```
PS D:\GI01\SR01\devoir1> gcc -o ex2 ex2.c
PS D:\GI01\SR01\devoir1> .\ex2
Combien d'etudiants ont fait le devoir de SR01 ?
10
Quelle est la note de l'etudiant 1
0
Quelle est la note de l'etudiant 2
5
Quelle est la note de l'etudiant 3
8
Quelle est la note de l'etudiant 4
15
Quelle est la note de l'etudiant 5
20
Quelle est la note de l'etudiant 6
25
Quelle est la note de l'etudiant 7
34
Quelle est la note de l'etudiant 8
38
Quelle est la note de l'etudiant 9
53
Quelle est la note de l'etudiant 10
60
```

```
La note maximale du devoir est de 60.
La note minimale du devoir est de 0.
La note moyenne du devoir est de 25.80.

3 >     o

2 >                     o       o

1 >             o                               o       o

    +--------+--------+--------+--------+----o---+--------+--------+
    |  0 - 9 | 10 - 19| 20 - 29| 30 - 39| 40 - 49| 50 - 59|   60   |

3 >  #######

2 >  #######          #######  #######

1 >  #######  #######  #######  #######          #######  #######

    +--------+--------+--------+--------+--------+--------+--------+
    | 0 - 9  | 10 - 19| 20 - 29| 30 - 39| 40 - 49| 50 - 59|   60   |
```

Exercice 3

Introduction

```c
typedef struct Voiture
{
    char modele[20];
    char immat[9];
    int km;
    enum Etat
    {
        dispo,
        louee
    } etat;
} Voiture;
```

Chaque véhicule est représenté sous la forme d'une structure décrite dans le header. Cette structure contient 2 tableaux de char de tailles fixes pour le modèle et l'immatriculation, un int pour le kilométrage et un état (loué ou dispo) représenté par un enum. Pour gérer le parc automobile nous avons décidé d'instancier chaque voiture dans un tableau. En effet, l'interface ne permettant pas d'ajouter ou de supprimer de véhicules, on supposera que la taille du parc reste constante, on peut donc initialiser dès le début un tableau de taille fixe à l'aide de la fonction init.

Fonction init

```c
Voiture *init(int n)
{
    Voiture *parc = (Voiture *)malloc(n * sizeof(Voiture));
    int choix = -1;
    printf("Avez-vous deja un fichier sauvegarde ? Oui : 1, Non : 0\n");
    scanf("%d", &choix);
    viderBuffer();
    if (choix == 0)
    {
        for (int i = 0; i < n; i++)
        {
            printf("Quel est le modele de la voiture %d ?\n", i + 1);
            lire(parc[i].modele, 21);

            printf("Quelle est l'immatriculation de la voiture %d ?\n", i + 1);
            lire(parc[i].immat, 9);

            printf("Quelle est le kilometrage de la voiture %d ?\n", i + 1);
            int km = -1;
            scanf("%d", &km);
            viderBuffer();
            while (km < 0)
            {
                printf("Veuillez indiquer un kilometrage positif ou nul");
                scanf("%d", &km);
                viderBuffer();
            }
            parc[i].km = km;

            printf("Quelle est l'etat de la voiture %d ? 0 : dispo, 1 louee\n", i + 1);
            int choix_etat = -1;
            scanf("%d", &choix_etat);
            viderBuffer();
```

```
        while (choix_etat != 1 && choix_etat != 0)
        {
            printf("Veuillez indiquer un etat de 0 ou 1\n");
            scanf("%d", &choix_etat);
            viderBuffer();
        }
        parc[i].etat = choix_etat;
    }
}
```

On récupère en paramètres un entier n correspondant à la taille du parc de voitures. On commence par allouer dynamiquement un espace de la taille de n fois la taille de la structure Voiture. On passe ensuite aux entrées utilisateurs pour saisir les informations des n véhicules du parc. Nous avons pris soin de sécuriser ces entrées à l'aide de deux fonctions : `lire` et `viderBuffer`. Ces fonctions assurent que les entrées textuelles ne dépassent pas la taille des variables (à la manière de `fgets`) et vide le flux `stdin` si l'utilisateur a saisi trop de caractères.

```
int lire(char *chaine, int longueur)
{
    char *positionEntree = NULL;
    if (fgets(chaine, longueur, stdin) != NULL)
    {
        positionEntree = strchr(chaine, '\n');
        if (positionEntree != NULL)
        {
            *positionEntree = '\0';
        }
        else
        {
            viderBuffer();
        }
        return 1;
    }
    else
    {
        viderBuffer();
        return 0;
    }
}
```

```
void viderBuffer()
{
    int c = 0;
    while (c != '\n' && c != EOF)
    {
        c = getchar();
    }
}
```

Une fois toutes les informations saisies et vérifiées, on instancie un élément du tableau précédemment alloué avec l'ensemble des champs obtenus.

La fonction `init` retourne enfin l'adresse du tableau pour qu'il puisse être utilisé par la suite.

Nous avons par la suite étendu cette fonction d'initialisation pour initialiser le parc avec un fichier binaire sauvegardé[1].

[1] Voir Extension de la fonction init pour plus d'informations

Fonction louer

```c
void louer(Voiture *voitures, int n)
{
    printf("Veuillez entrer l'immatriculation du vehicule a louer\n");
    char immat[9];
    lire(immat, 9);

    int voiture_index = -1;
    for (int i = 0; i < n; i++)
    {
        if (strcmp(voitures[i].immat, immat) == 0)
        {
            voiture_index = i;
        }
    }

    if (voiture_index != -1)
    {
        if (voitures[voiture_index].etat == dispo)
        {
            voitures[voiture_index].etat = louee;
            printf("Le vehicule immatricule %s est maintenant loue ! \n", voitures[voiture_index].immat);
        }
        else
        {
            printf("Cette voiture est deja louee.\n");
        }
    }
    else
    {
        printf("Aucun vehicule ne correspond a cette immatriculation.\n");
    }
}
```

On commence par demander l'immatriculation du véhicule recherché. On boucle ensuite sur le tableau parc en comparant chaque immatriculation pour récupérer uniquement le véhicule recherché.

Si l'immatriculation existe et que la voiture est disponible, il suffit de modifier son état (de disponible à loué) sinon on affiche que le véhicule est déjà loué ou que l'immatriculation ne correspond à aucun véhicule du parc.

Fonction retour

```c
void retour(Voiture *voitures, int n)
{
    char immat[9];
    int ok = 0, km = -1;
    printf("Veuillez entrer l'immatriculation du vehicule a retourner.\n");
    lire(immat, 9);
    for (int i = 0; i < n; i++)
    {
        if (strcmp(voitures[i].immat, immat) == 0)
        {
            ok = 1;
            if (voitures[i].etat == dispo)
            {
                printf("La voiture n'etait pas en location.\n");
            }
            else
            {
                printf("Combien de kilometres avez-vous fait ?\n");
                scanf("%d", &km);
                viderBuffer();
                while (km < 0)
                {
                    printf("Veuillez entrer un kilometrage positif.\n");
                    scanf("%d", &km);
                    viderBuffer();
                }
                voitures[i].km += km;
                voitures[i].etat = dispo;
                printf("La voiture est de nouveau disponible avec %d km au compteur.\n", voitures[i].km);
            }
        }
    }
    if (ok == 0)
    {
        printf("Aucun vehicule ne correspond a cette immatriculation.\n");
    }
}
```

Cette fonction est très similaire à la fonction `louer`, il suffit simplement de vérifier que la voiture était bien en location, de demander le kilométrage parcouru et d'incrémenter en conséquence le kilométrage total de la voiture.

Fonction etat

```c
void etat(Voiture *voitures, int n)
{
    char immat[9];
    int ok = 0;
    printf("Veuillez entrer l'immatriculation du vehicule a afficher.\n");
    lire(immat, 9);
    for (int i = 0; i < n; i++)
    {
        if (strcmp(voitures[i].immat, immat) == 0)
        {
            ok = 1;
            printf("La voiture est une %s\n", voitures[i].modele);
            printf("L'immatriculation de la voiture est : %s\n", voitures[i].immat);
            printf("La voiture a %d km au compteur\n", voitures[i].km);
            if (voitures[i].etat == dispo)
            {
                printf("La voiture est disponible a la location.\n");
            }
            else
            {
                printf("La voiture est deja louee.\n");
            }
        }
    }
    if (ok == 0)
    {
        printf("Aucun vehicule ne correspond a cette immatriculation.\n");
    }
}
```

Comme pour les fonctions précédentes, on boucle sur le tableau parc en comparant les immatriculations avec celle saisie par l'utilisateur, une fois trouvée on affiche simplement les différents champs contenus dans la structure Voiture.

Fonction etatParc

```c
void etatParc(Voiture *voitures, int n)
{
    printf("Le parc automobile est actuellemnt compose de %d vehicules.\n", n);
    int nb_loc = 0;
    int sum_km = 0;
    for (int i = 0; i < n; i++)
    {
        if (voitures[i].etat == louee)
            nb_loc++;
        sum_km += voitures[i].km;
    }
    printf("%d vehicules sont loues.\n", nb_loc);
    printf("%d vehicules sont disponibles.\n", n - nb_loc);
    printf("Le kilometrage moyen de l'ensemble des voitures est de %.1f km.\n", (float)sum_km / (float)n);
}
```

Cette fonction utilise deux compteurs : le nombre de voitures louées et le kilométrage total. On boucle ensuite sur le tableau parc en incrémentant les compteurs.

On affiche enfin les informations relatives au parc en utilisant les compteurs obtenus. Sachant km moyen = km total / nb voitures.

Fonction free_parc

```c
void free_parc(Voiture *parc, int n)
{
    free(parc);
}
```

La fonction `free_parc` nous permet de libérer la mémoire allouée dynamiquement à l'initialisation du tableau de voitures grâce à la fonction free.

Fonction save

```c
void save(Voiture *voitures, int n)
{
    FILE *file = NULL;
    char fichier[30];
    printf("Quel est le nom du fichier ?\n");
    lire(fichier, 30);
    if ((file = fopen(fichier, "w")) == NULL)
    {
        perror("fopen");
        exit(1);
    }
    fwrite(voitures, sizeof(Voiture), n, file);
    fclose(file);
}
```

La fonction `save` nous permet de sauvegarder le parc de voiture dans un fichier binaire. On demande alors à l'utilisateur le nom du fichier. On ouvre le fichier en écriture, on s'assure que le fichier est bien ouvert et on y écrit toutes les voitures du parc avant de fermer le fichier. On utilise pour cela la fonction d'écriture binaire `fwrite` avec en argument le nombre de voiture contenues dans le parc et la taille de la structure voiture.

Fonction menu

```c
void menu(Voiture *parc, int n)
{
    int choix;
    int continuer = 1;
    while (continuer == 1)
    {
        printf("\n-----------------MENU-----------------\n");
        printf("1: Louer une voiture\n");
        printf("2: Retour d'une voiture\n");
        printf("3: Etat d'une voiture\n");
        printf("4: Etat du parc de voitures\n");
        printf("5: Sauvegarde de l'etat du parc\n");
        printf("0: fin du programme\n");

        scanf("%d", &choix);
        viderBuffer();
        printf("\n");
        switch (choix)
        {
        case 1:
            louer(parc, n);
            break;
        case 2:
            retour(parc, n);
            break;
        case 3:
            etat(parc, n);
            break;
        case 4:
            etatParc(parc, n);
            break;
        case 5:
            save(parc, n);
            break;
        case 0:
            free_parc(parc, n);
            continuer = 0;
            break;
        default:
            printf("Veuillez saisir un choix valide. \n");
            break;
        }
    }
}
```

La fonction `menu` sert seulement à afficher le menu et appeler les fonctions correspondantes au choix entré par l'utilisateur.

Main

```c
int main(int argc, char const *argv[])
{
    Voiture *parc = init(TAILLE);
    menu(parc, TAILLE);
    return 0;
}
```

On initialise d'abord un parc de voiture de taille TAILLE (constante définie dans le header) puis on appelle la fonction menu sur ce parc.

Extension de la fonction init

```
    else
    {
        FILE *file = NULL;
        char fichier[30];
        printf("Quel est le nom du fichier ?\n");
        lire(fichier, 30);
        if ((file = fopen(fichier, "r")) == NULL)
        {
            perror("fopen");
            exit(1);
        }
        fread(parc, sizeof(Voiture), n, file);
        fclose(file);
        printf("Le parc a bien ete cree.\n");
        printf("Les immatriculations des voitures du parc sont les suivantes:\n");
        for (int i = 0; i < n; i++)
        {
            printf("- %s \n", parc[i].immat);
        }
    }
    return parc;
}
```

Comme il est désormais possible d'enregistrer les données d'un parc sous un fichier binaire, la fonction `init` a été modifiée pour pouvoir initialiser le parc avec un fichier binaire préalablement sauvegardé. On demande donc à l'utilisateur dans la fonction `init` s'il souhaite initialiser le parc à l'aide d'un fichier, si c'est le cas :

On demande à l'utilisateur le nom de ce fichier, on l'ouvre et on vérifie que cela s'est bien passé. Puis on lit le fichier à l'aide de `fread` et on entre les caractéristiques des voitures dans le parc préalablement alloué. On effectue un affichage des immatriculations des voitures du parc pour que l'utilisateur vérifie bien que toutes ses voitures ont été initialisées.