

Université d'ORLÉANS

Centre Universitaire
de BOURGES

Laboratoire d'Informatique Fondamentale
d'ORLÉANS

Une introduction au langage *C*

Exercices corrigés

A. ED-DBALI

Exercice 1 : *Calculer la taille des types de données.*

```
#include <stdio.h >
main() {
    printf ("caractere  %d\n", sizeof(char));
    printf ("entier      %d\n", sizeof(int));
    printf ("float       %d\n", sizeof(float));
    printf ("short       %d\n", sizeof(short));
    printf ("long        %d\n", sizeof(long));
    printf ("double      %d\n", sizeof(double));
}
```

Résultat de l'exécution :

```
caractere  1
entier     2
float      4
short     2
long      4
double    8
```

Exercice 2 : *Lire un caractère, si c'est une lettre minuscule la convertir en majuscule.*

```
if (c >= 'a' && c <= 'z') printf("%c", c + 'A' - 'a');
```

Exercice 3 : *Lire deux nombres et un signe opératoire (+, -, *, / ou %) puis effectuer l'opération correspondante.*

```
#include <stdio.h>
main() {
    int i, j;
    char Op;
    scanf("%d %c %d", &i, &Op, &j);
    switch(c) {
        case '+': printf("Resultat : %d\n", i + j); break;
        case '-': printf("Resultat : %d\n", i - j); break;
        case '*': printf("Resultat : %d\n", i * j); break;
        case '/': printf("Resultat : %d\n", i / j); break;
        case '%': printf("Resultat : %d\n", i % j); break;
        default : printf("Operation inconnue\n");
    }
}
```

Exercice 4 : *Lire une suite de caractères et la convertir en un entier.*

```
#include <stdio.h>
main ( ) {
    int i ;
    char c ;
    i = 0 ;
    while ((c=getchar( )) >= '0' && c<= '9')
        i = i * 10 + c - '0';
}
```

```
    printf("%d\n", i);
}
```

ou bien :

```
for (i = 0; (c = getchar( )) >= '0' && c <= '9'; i = i * 10 + c - '0');
```

Si on saisi un entier long, il faut définir i comme suit : long i; et à l'affichage utiliser le format %ld :
printf("%ld\n", i);

(les autres format longs existent aussi : lf et le)

Exercice 5 : Dans un texte, remplacer les tabulations par des espaces. Il ne s'agit pas de remplacer systématiquement une tabulation par huit espaces.

Si on numérote les colonnes à partir de 0 :

- si le caractère tabulation est en colonne 2 il faut le remplacer par 6 espaces pour se retrouver en colonne 8;
- si le caractère tabulation est en colonne 0 il faut le remplacer par 8 espaces pour se retrouver en colonne 8;
- si le caractère tabulation est en colonne 14 il faut le remplacer par 2 espaces pour se retrouver en colonne 16;
etc.

D'où l'algorithme :

- Avancer de $8 - i \% 8$
- Aller à $i + 8 - i \% 8$

```
#include <stdio.h>
main( ) {
    int i = 0, j;
    char c;
    while ((c=getchar( )) != EOF)
        /* Lire jusqu'a la fin du fichier d'entree standard CTRL-D */
        switch(c) {
            case '\t' : for(j = 1 ; j <= 8 - i % 8 ; j++) putchar(' ');
                       i = i + 8 - i % 8 ; break ;
            case '\n' : i = 0 ; putchar(c) ; break ;
            default   : i++ ; putchar(c) ;
        }
}
```

ou bien

```
case '\t' : do { i++ ; putchar ( ' ' ) ; }
           while (i % 8) ; break ; /* la variable j devient inutile */
```

ou encore

```
case '\t' : do putchar ( ' ' )
           while (++i % 8) ; break ;
```

Exercice 6 : Saisir puis trier un tableau d'entiers (tri par sélection).

```
#define L 20

int t[L], i, debut, IndiceMin, aux ;
```

```

main () {
    for (i = 0 ; i < L ; i++)
        scanf("%d", &t[i]) ; /* remplit le tableau */

    for (debut = 0 ; debut < L-1 ; debut++) {
        /* les elements de t avant debut sont deja a leur place */
        /* recherche du minimum du sous tableau t[debut .. L-1] */
        IndiceMin = debut;
        for (i = debut + 1 ; i < L ; i++)
            if (t[i] < t[IndiceMin]) IndiceMin = i ;
        /* IndiceMin = indice du minimum de t[debut .. L-1] */
        /* Echanger t[IndiceMin] et t[debut] */
        aux = t[debut] ;
        t[debut] = t[IndiceMin] ;
        t[IndiceMin] = aux ;
    }

    /* Affichage du tableau trie */
    for(i = 0 ; i < L ; i++) printf("%d ", t[i]);
    putchar('\n') ; /* passage a la ligne */
}

```

Exercice 7 : Lire une série de nombres et à chaque lecture insérer le nouveau nombre lu au bon endroit (tri par insertion).

```

...
for (i = 0 ; i < L ; i++) {
    scanf("%d", &x);
    for (j = i ; j > 0 && x < t[j-1] ; j--)
        t[j] = t[j-1];
    t[j] = x ;
}
for (i = 0 ; i < L ; i++) printf("%d ", t[i]);

```

En PASCAL, la condition $j > 0 \ \&\& \ x < t[j-1]$ posera un problème car si j est égale à 0, par exemple, $t[-1]$ qui n'existe pas cause un débordement dans l'accès au tableau t ¹. Ici il n'y aura pas de problème si on observe l'ordre des comparaisons dans cette condition. En effet, C évalue d'abord la condition $j > 0$, et si elle est fausse alors toute la condition est fausse (on ne regarde pas si $x < t[j-1]$).

Exercice 8 : Ecrire une fonction `echange(x,y)` qui échange les valeurs des 2 variables x et y .

```

void echange(x,y)
    int *x, *y ; { /* x et y sont des adresses d'entiers (et non pas des entiers) */
    int aux ;      /* La variable auxiliaire pour l'echange */
    aux = *x ;
    *x = *y ;
    *y = aux ;
}
main () {
    int i, j ;
    ...
    echange (&i, &j) /* On transmet les adresses des variables a echanger */
    ...
}

```

¹Le remède serait de remplacer le test $x < t[j-1]$ par un booléen

Problème : Trier un tableau par échanges.

```
#include <stdio.h>
#define L 10
int t[L] ;
main () {
    LireTab(t) ;
    TriTab(t) ;
    EcrireTab(t) ;
}

LireTab(u)
int u[ ] { /* pas besoin de donner la longueur le compilateur doit */
int i ; /* seulement savoir que c'est un tableau d'entiers */
for (i = 0 ; i < L ; i++)
    scanf("%d", &u[i]) ;
}

EcrireTab(u)
int u [ ] {
int i,
for (i = 0 ; i < L ; i++) printf("%d", u[i]) ;
putchar('\n') ;
}

echange(x,y) (cf. plus haut)

int RechMin (u, d, f) /* Cherche l'indice du minimum */
int u[ ],d, f { /* tableau, indice debut, indice fin */
int Imin, j ;
Imin = d ;
for(j = d+1 ; j <= f ; j++)
if (u[j] < u[Imin]) Imin=j ;
return Imin ;
}

TriTab (u) /* trie le tableau u de longueur L */
int u[ ] {
int i ;
for (i=0 ; i < L-1 ; i++)
echange(&u[RechMin(u, i, L-1)], &u[i]) ;
}
```

| |
|--|
| Exercice 9 : Décoder un nombre en une suite d'entiers de 0 à 9. |
|--|

```
decoder(n) int n; {
    int q;
    q = n / 10;
    if (q > 0) decoder(q);
    putchar(n % 10 + '0');
}
```

Exercice 10 : Copier un tableau dans un autre : `copier(t_1, t_2, n)` avec t_1 le tableau source, t_2 le tableau destination et n le nombre d'éléments à copier.

```
copier(t1, t2, n) int t1[ ], *t2, n; { /* n = longueur */
    int i ;
    for (i = 0 ; i < n ; i++) t2[i] = t1[i];
}
ou bien for (i = 0 ; i < n ; i++) *t2++ = *t1++;
ou encore while (n-- > 0) *t2++ = *t1++ ; (la variable i devient inutile).
```

Exercice 11 : Copier un tableau à 2 dimensions dans un autre.

```
copier(t1, t2, d1, d2) int t1[][10], t2[][10], d1, d2; {
    int i, j ;
    for (i = 0 ; i < d1 ; i++)
        for (j = 0 ; j < d2 ; j++)
            t2[i][j] = t1[i][j];
}
```

Exercice 12 : Copier une chaîne dans une autre.

```
copiechaine(S1, S2) char *S1, *S2 ; {
    while (*S2++ = *S1++);
}
```

Dans l'affectation `*S2++ = *S1++` (qui est aussi une condition) le teste se fait après affectation donc on copie dans S2 le `'\0'` qui marque la fin de S1.

Exercice 13 : Concaténer deux chaînes S_1 et S_2 dans une troisième chaîne S_3 en rendant la longueur de S_3 .

```
int concat(S1, S2, S3) char S1[], S2[], S3[]; {
    int l = 0;
    while (*S1 != '\0') { *S3++ = *S1++; l++;}
    while (*S3++ = *S2++) l++;
    return l ;
}
```

Exercice 14 : Concaténer S_1 et S_2 dans S_3 sans déclarer S_3 à l'avance mais en rendant son adresse.

```
char * concat(S1, S2) char *S1, *S2; {
    int l = 0 ;
    char *p = S1, *q = S2;
    while (*p++) l++;          /* calcule la longueur de S1 */
    while (*q++) l++;          /* ajoute la longueur de S2 */
    q = p = malloc(l+1);      /* reservation dynamique */
    while (*S1) *p++ = *S1++; /* copie S1 dans p */
    while (*p++ = *S2++);     /* copie S2 a la suite */
    return q;                  /* pointeur sur la tete de la chaine resultat */
}
```

```

}

main( ) {
    char S1[80], S2[80]; /* ce sont des tableaux      */
    char * S3;          /* c'est seulement un pointeur */
    ...
    S3 = concat (S1, S2);
    printf("%s", S3);
    ...
}

```

Exercice 15 : Saisir une liste d'individus puis les trier par ordre alphabétique en échangeant des pointeurs plutôt que des individus.

```

#include <stdio.h>
#include <string.h>

typedef struct { /* DATE devient un synonyme de struct {...} */
    int j, a; char mois[15];
} DATE;

typedef struct {
    char nom [50], prenom [50]; DATE dnaiss;
}INDIVIDU ; /* designe la structure qui vient d'etre definie */

#define L10
INDIVIDU tab [L] /* tableau d'individus*/
INDIVIDU *(tp[L]) ; /* tableau de pointeurs sur des individus */

main () {
    remplir(); /* remplir tab */
    initialiser(); /* initialiser tp */
    trier();
    editer();
}

initialiser() {
    int i;
    for (i = 0 ; i < L ; i++) tp[i] = tab + i;
        /* tp[i] est un pointeur vers le ieme element de tab */
}

trier() { /* tri a bulles */
    int fin ;
    for (fin = L-1 ; fin > 0 ; fin--) bulle(fin);
}

bulle(fin) int fin; { /* fait monter jusqu'a fin le */
                    /* maximum de tp[0..fin] */
    for (i = 1 ; i <= fin ; i++)
        if (superieur(*(tp[i-1]), *(tp[i])))
            echanger(tp + (i-1), tp + i); /* au depart *tp[i] = tab[i] */
}

echanger(x, y) INDIVIDU **x, **y;
                /* echange des deux pointeurs qui se trouvent
                dans le tableau tp : se sont des pointeurs

```

```

        vers des pointeurs */
    INDIVIDU * aux;
    aux = *x;
    *x = *y;
    *y = aux;
}

superieur(i1, i2) INDIVIDU i1, i2; { /*comparer i1.nom et i2.nom */
    return strcmp (i1.nom, i2.nom) > 0 ? 1 : 0;
} /* strcmp est une fonction de la bibliotheque qui donne un
    resultat (> 0), (= 0) ou (< 0) selon que la 1ere chaine
    est superieure, egale ou inferieure a la seconde */

remplir() {
    int i;
    for (i = 0 ; i < L ; i++) lire_individu(tab + i);
}

Lire_individu (pi) INDIVIDU *pi; {
    scanf("%s", &(pi -> nom));
    scanf("%s", &(pi -> prenom));
    scanf("%d %s %d", &(pi -> dnaiss.j), &(pi -> dnaiss.m), &(pi -> dnaiss.a));
}

editer() {
    int i;
    for (i=0 ; i < L ; i++) {
        print f("%s\n", tp[i] -> nom); /* ou bien (*(tp[i])).nom */
        print f("%d %s %d\n", tp[i] -> dnaiss.j,
                tp[i] -> dnaiss.m,
                tp[i] -> dnaiss.a);
    }
}

```

Exercice 16 : Trier, par ordre croissant, une suite d'entiers (utilisation de l'arbre binaire de recherche).

L'idée est de mettre cette suite dans un arbre binaire telque pour chaque nœud d'étiquette l'entier N on ait les éléments inférieurs ou égaux à N dans le sous arbre gauche et les éléments supérieurs à N dans le sous arbre droit.

```

typedef struct element {
    int Objet;
    struct element * SousArbreGauche, * SousArbreDroit;
} Noeud, *Arbre;
Arbre construire() { /*lit des entiers et les met dans un arbre binaire de recherche */
    int x ;
    Arbre A = NULL; /* Pointeur sur la racine de l'arbre jusqu'alors construit. */
    while (scanf("%d", &x) != EOF) {
        if (A == NULL) /* Le premier element vient d'etre lu. */
            A = creer_element(x);
        else /* L'arbre n'est pas vide => placer x convenablement dans A. */
            placer(x, A);
    }
    return A; /* Le resultat est l'adresse de la racine. */
}
Arbre creer_element(x) int x; { /* cree un noeud en y mettant la valeur de x */
    Arbre p, /* Pointeur pour accueillir l'entier lu. */
    p = (Arbre) malloc(sizeof(Noeud));
    p -> Objet = x;
    p -> SousArbreGauche = p -> SousArbreDroit = NULL;
    return p;
}

void placer(x, r) int x; Arbre r; { /* Placer x dans un arbre (non vide) de racine r. */

```

```

if (x <= r -> Objet)
    if (r -> SousArbreGauche == NULL) /* Sous arbre gauche vide */
        r -> SousArbreGauche = creer_element(x);
    else /* Sous arbre gauche non vide */
        placer(x, r -> SousArbreGauche);
else /* faire la meme chose avec le sous arbre droit */
    if (r -> SousArbreDroit == NULL)
        r -> SousArbreDroit = creer_element(x);
    else
        placer(x, r -> SousArbreDroit);
}
int Afficher(r) Arbre r; { /* Affiche l'arbre trie et compte ses elements. */
    int NbrElements = 0;
    if (r -> SousArbreGauche != NULL)
        NbrElements += Afficher(r -> SousArbreGauche); /* Parcourt le sous-arbre gauche. */
    printf("%d ", r -> Objet); /* Affiche et */
    NbrElements++; /* compte l'entier de la racine */
    if (r -> SousArbreDroit != NULL)
        NbrElements += Afficher(r -> SousArbreDroit); /* Parcourt le sous-arbre droit. */
    return NbrElements;
}
main () {
    print f("Nombre d'elements : %d\n", afficher(construire()));
}

```

Exercice 17 : *Ecrire une fonction `evaluer(Exp)` qui évalue une expression. `Exp` est une expression représentée par un arbre binaire dont les nœuds sont des opérateurs binaires et les feuilles des nombres. Exemple : L'expression $3 * (4 + 5) / 7 - 9$ sera représentée par l'arbre dont l'écriture termale est : $-(/*(3, +(4, 5)), 7), 9)$.*

```

typedef enum { operation, nombre } TYPE_NOEUD;
typedef struct noeud {
    TYPE_NOEUD TypeNoeud;
    union {
        char Operation;
        float Nombre;
    } operation_ou_nombre;
    struct noeud * ExpGauche, * ExpDroite;
} NOEUD, * Expression;

#define Op    operation_ou_nombre.Operation
#define Nbr   operation_ou_nombre.Nombre

float evaluer(Exp) Expression Exp; {
    if (Exp -> TypeNoeud == nombre) /* Si c'est un nombre, le resultat est ce nombre */
        return Exp -> Nbr;
    else
        switch (Exp -> Op) {
            case '+' : return evaluer(Exp -> ExpGauche) + evaluer(Exp -> ExpDroite);
            case '-' : return evaluer(Exp -> ExpGauche) - evaluer(Exp -> ExpDroite);
            case '*' : return evaluer(Exp -> ExpGauche) * evaluer(Exp -> ExpDroite);
            case '/' : return evaluer(Exp -> ExpGauche) / evaluer(Exp -> ExpDroite);
        }
}

```

Exercice 18 : *Ecrire un programme qui lit les noms de deux fichiers d'entiers puis réalise la copie de l'un vers l'autre.*

```

#include <stdio.h>
main () {
    FILE *fichier_source, *fichier_destination;
    char nom_fs[40], nom_fd[40];
    int i;
    scanf("%s %s", &nom_fs, &nom_fd);
    if ((fichier_source = fopen(nom_fs, "r")) == NULL) {
        fprintf(stderr, "%s : open failed !", nom_fs);
    }
}

```

```

    exit(1); /* 1 : pour dire par exemple qu'une ouverture
              en lecture ne s'est pas bien deroulee */
}
if ((fichier_destination = fopen(nom_fd, "w")) == NULL) {
    fprintf(stderr, "%s : open failed !", nom_fd);
    exit(2); /* 2 : pour dire par exemple qu'une ouverture
              en ecriture ne s'est pas bien deroulee */
}
while (fread((char *) &i, sizeof(int), 1, fichier_source) != EOF)
    fwrite((char *) &i, sizeof(int), 1, fichier_destination);
fclose(fichier_source); fclose(fichier_destination);
return 0; /* 0 : pour dire que la copie s'est bien passee */
}

```

Exercice 19 : Réécrire en C la commande UNIX `cp` qui réalise la copie d'un ou plusieurs fichiers dans un autre le dernier fichier passé en paramètre est le fichier récepteur. (passage de paramètres à un programme).

```

#include <stdio.h>
main(argc, argv)
    int argc;          /* Nombre de parametres - 1 (argc toujours > 1) */
    char **argv; {     /* tableau de chaines de caracteres */
    FILE *in, *out; /* deux descripteurs de fichiers */
    int i;             /* Indice de boucle */
    if (argc==1)      /* aucun parametre :                               */
        copie(stdin, stdout); /* copier l'entree standard dans la sortie standard */
    else {            /* Au moins un parametre :                               */
        ouvrir le fichier recepteur                                     */
        if ((out = fopen(argv[argc - 1], "w")) == NULL) exit(1);
        if (argc == 2) /* Un seul param\etre :                               */
            copie(stdin, out); /* entree = clavier                               */
        else
            for(i = 1 ; i < argc - 1 ; i++) { /* Parcourir la liste des fichiers source */
                if ((in = fopen(argv[i], "r")) == NULL) exit(2) ;
                copie (in, out) ;
                fclose (in) ;
            }
        fclose (out) ;
    }
    return 0 ; /* Si tout va bien, le code de retour est 0 */
}

copie (fs,fd) /* copier fs dans fd */
    FILE *fs, *fd; {
    char c;
    while ((c = getc(fs)) != EOF) putc(c, fd);
}

```

Exercice 20 : Ecrire un programme qui crée un fichier d'individus (comme défini plus haut).

```

#include <stdio.h>

typedef struct individu {
    ...} INDIVIDU; /* voir plus haut */

lire_individu(pi) /* voir plus haut */
    INDIVIDU *pi; {
    ...
}

```

```
}

main () {
    INDIVIDU i; FILE * f;
    if ((f = fopen("individus", "wb")) == NULL) /* "wb" pour ecriture en binaire */
        exit(1);
    while (!feof(stdin)) { /* Tant que la fin du fichier stdin (Ctrl-D)
                            n'est pas attiente */
        lire_individu(&i);
        fwrite ((char *) (&i) , sizeof(INDIVIDU), 1, f);
    }
    fclose(f);
}
```

Pour relire le fichier.

```
main() {
    INDIVIDU i; FILE * f;
    if ((f = fopen("individus", "rb")) == NULL) /* "rb" pour lecture en binaire */
        exit(1);
    while (fread((char *) (&i), sizeof(INDIVIDU), 1, f) == 1)
        /* Tant que fread lit un individu complet. */
        editer(&i); /* Fonction definie plus haut. */
}
```