CS-119(a) – ICC-C Série 14

2024-05-27

Important Les exercices 1-3 portent sur le backtracking. Les exercices 4-6 sont des exercices de révision, jetez-y un coup d'oeil aussi !

Exo1 Les permutations

Utilisez le code vu en cours pour écrire un programme qui lit un entier `N` et affiche toutes les permutations de taille `N` à l'écran.

Solution de l'exercice 1

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void afficher_pn(int *perm, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", perm[i]);
    }
    printf("\n");
}

void enumerer_pn(int *perm, int *vu, int indice, int n)
{
    if (indice == n)
    {
        afficher_pn(perm, n);
        return;
    }

    for (int valeur = 0; valeur < n; valeur++)
```

```
    {
        if (!vu[valeur])
        {
            vu[valeur] = 1;
            perm[indice] = valeur;
            enumerer_pn(perm, vu, indice + 1, n);
            vu[valeur] = 0;
        }
    }
}

int main(int argc, char **argv)
{
    int n;
    if (argc != 2)
    {
        printf(
            "%s n liste toutes les permutations "
            "des nombres entre 0 et n-1.\n",
            argv[0]);
        return 1;
    }

    n = atoi(argv[1]); // parse int
    if (n <= 0)
    {
        printf(
            "%s doit être un entier supérieur à 0\n.",
            argv[1]);
        return 2;
    }

    int *perm = malloc(n * sizeof(int));
    int *vu = malloc(n * sizeof(int));

    memset(vu, 0, 5 * sizeof(int));
    enumerer_pn(perm, vu, 0, n);

    free(perm);
    free(vu);

    return 0;
}
```

Exo2 Yoda-speak

On nous donne un fichier avec une phrase découpée en séquences de mots, avec un bout de phrase par ligne. Chaque bout de phrase occupe moins de 30 caractères et il y a au plus 10 lignes contenant du texte dans le ficher. La dernière ligne est vide. On aimerait reconstituer la phrase de toutes les manières possibles. Par exemple pour l'entrée

```
que tu aies
il faut
patience
mon jeune Padawan
```

on aimerait obtenir les `4! = 24` phrases suivantes (dans n'importe quel ordre) :

```
patience il faut que tu aies mon jeune Padawan
patience il faut mon jeune Padawan que tu aies
patience que tu aies il faut mon jeune Padawan
patience que tu aies mon jeune Padawan il faut
patience mon jeune Padawan il faut que tu aies
patience mon jeune Padawan que tu aies il faut
il faut patience que tu aies mon jeune Padawan
il faut patience mon jeune Padawan que tu aies
il faut que tu aies patience mon jeune Padawan
il faut que tu aies mon jeune Padawan patience
il faut mon jeune Padawan patience que tu aies
il faut mon jeune Padawan que tu aies patience
que tu aies patience il faut mon jeune Padawan
que tu aies patience mon jeune Padawan il faut
que tu aies il faut patience mon jeune Padawan
que tu aies il faut mon jeune Padawan patience
que tu aies mon jeune Padawan patience il faut
que tu aies mon jeune Padawan il faut patience
mon jeune Padawan patience il faut que tu aies
mon jeune Padawan patience que tu aies il faut
mon jeune Padawan il faut patience que tu aies
mon jeune Padawan il faut que tu aies patience
mon jeune Padawan que tu aies patience il faut
mon jeune Padawan que tu aies il faut patience
```

Lisez ligne par ligne depuis le fichier avec la fonction `fgets`, et n'oubliez pas d'enlever le dernier caractère `'\n'` de chaque ligne. Rappel : quand le fichier est terminé la fonction `fgets` retourne `NULL`.

Ecrivez les phrases reconstituées dans le fichier `yoda.txt`

Solution de l'exercice 2

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void afficher_mots(FILE *f,
                   char **phrase,
                   int n)
{
    for (int i = 0; i < n; i++)
    {
        fprintf(f, "%s ", phrase[i]);
    }
    fprintf(f, "\n");
}

void recomposer(char **phrase,
                int *vu,
                char **mots,
                int indice,
                int n,
                FILE *out)
{
    if (indice == n)
    {
        afficher_mots(out, phrase, n);
        return;
    }

    for (int i = 0; i < n; i++)
    {
        if (!vu[i])
        {
            vu[i] = 1;
            phrase[indice] = mots[i];
            recomposer(phrase, vu, mots, indice + 1, n, out);
            vu[i] = 0;
        }
    }
}

#define N 10
```

```c
#define M 32 // 30 + \n + \0
int main()
{
    char *phrase[N];
    int vu[N];
    char buffer[M];
    char *mots[N];

    FILE *in = fopen("mots.txt", "r");
    int n_mots = 0;
    while (fgets(buffer, M - 1, in) != NULL)
    {
        int len = strlen(buffer);

        if (len == 0)
        {
            // ligne vide
            break;
        }

        if (buffer[len - 1] == '\n')
        {
            buffer[len - 1] = 0;
        }

        mots[n_mots] = malloc(len * sizeof(char));
        strcpy(mots[n_mots], buffer);

        n_mots++;
    }

    FILE *out = fopen("yoda.txt", "w");

    memset(vu, 0, N * sizeof(int));
    recomposer(phrase, vu, mots, 0, n_mots, out);
    fclose(out);

    for (int i = 0; i < n_mots; i++)
    {
        free(mots[i]);
    }
}
```

Exo3 Les N reines

On lit un entier `N > 1`. Sur un échiquier de taille `N x N` vous devez placer `N` reines de telle façon qu'aucune reine n'attaque aucune autre. Si c'est possible, alors affichez un échiquier possible avec des `'.'` aux emplacements vides et des `'*'` aux emplacements correspondant aux positions des reines. Sinon affichez le mot `"Impossible"`.

Par exemple, pour `N = 5` une réponse possible est

```
* . . . .
. . * . .
. . . . *
. * . . .
. . . * .
```

Indices Il existe des solutions pour tous les nombres naturels, sauf 2 et 3.

Les reines doivent être placées sur des lignes, des colonnes, et des diagonales différentes. Vous pouvez donc décrire la position des reines par un tableau unidimensionnel `col` de taille `N` qui est tel que la `i`-ème reine est placée à la ligne `i` et à la colonne `col[i]`. On peut construire ce tableau commençant par la position `0` avec la méthode "backtracking".

En explorant la `j`-ème position de ce tableau (rempli correctement jusqu'à la position `j-1`) vous pouvez vérifier que la nouvelle reine sur la ligne `j` n'attaque aucune des reines placées aux positions `(1, col[1]), ..., (j-1, col[j-1])`.

Solution de l'exercice 3

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void afficher(int *col, int n)
{
    for (int r = 0; r < n; r++)
    {
        for (int c = 0; c < n; c++)
        {
            if (c == col[r])
            {
                printf(" *");
            }
            else
            {
                printf(" .");
            }
        }
        printf("\n");
    }
}

int valide(int *col, int r, int c)
{
    for (int i = 0; i < r; i++)
    {
        if (col[i] == c)
        {
            // colonne
            return 0;
        }
        if (c - col[i] == r - i || c - col[i] == i - r)
        {
            // diagonale
            return 0;
        }
    }
    return 1;
}
```

```c
int reines(int *col, int *vu, int indice, int n)
{
    if (indice == n)
    {
        afficher(col, n);
        return 1;
    }

    for (int c = 0; c < n; c++)
    {
        if (valide(col, indice, c))
        {
            vu[c] = 1;
            col[indice] = c;
            if (reines(col, vu, indice + 1, n))
            {
                return 1;
            }
            vu[c] = 0;
        }
    }

    return 0;
}

int main(int argc, char **argv)
{
    int n;
    if (argc != 2)
    {
        printf(
            "%s n dessine un échiquier de taille n x n qui
   contient n reines qui ne s'attaquent pas réciproquement.\n"
   ,
            argv[0]);
        return 1;
    }

    n = atoi(argv[1]); // parse int
    if (n <= 0)
    {
        printf(
            "%s doit être un entier supérieur à 0\n.",
```

```
            argv[1]);
        return 2;
    }

    int *col = malloc(n * sizeof(int));
    int *vu = malloc(n * sizeof(int));

    memset(vu, 0, 5 * sizeof(int));
    if (!reines(col, vu, 0, n))
    {
        printf("Impossible\n");
    }

    free(col);
    free(vu);

    return 0;
}
```

Exo4 Stars

Qu'affiche ce code ?

```
    int v[6] = {5, 3, 0, 4, 2, 1};
    for (int *p = v; *p; p = v + *p)
    {
        printf("%d ", *p);
    }
    printf("\n");
```

Solution de l'exercice 4

Suivons la suite des instructions exécutées :

1. Le pointeur `p` pointe vers `v[0]`
2. On teste si `*p` est nul – le 1er élément de `v` est `5`, donc ce n'est pas le cas
3. On affiche `*p`, donc `5`
4. Ensuite `p` reçoit la valeur `v + *p`, donc il pointe vers `v + 5`, donc vers le dernier élément du tableau `v`
5. Ce dernier vaut `1`, donc toujours pas nul
6. On affiche `1`
7. Ensuite `p` vaut `v + 1`, donc pointe vers le deuxième élément de `v`
8. Il est aussi non-nul (car égal à `3`)
9. On affiche `3`
10. Pareil, `p` pointe maintenant vers le quatrième élément
11. On affiche `4`
12. Idem, on affiche `2`
13. Enfin `p` pointe vers de troisième élément de `v`
14. Celui-ci vaut `0`, donc on sort de la boucle et on affiche un retour à la ligne.

Le code affiche `5 1 3 4 2` (et un retour à la ligne).

Exo5 Bug mystère

Votre collègue a écrit le code suivant pour lister les `N` premiers nombres pairs positifs. Il a rencontré un problème en testant ce code et vous demande votre avis.

```
#include <stdio.h>

int* quelques_nombres_pairs(int n)
{
    int tableau[n];
```

```
    for (int i=0; i<n; i++)
    {
        tableau[i] = 2*i;
    }
    return tableau;
}

int main()
{
    int combien;
    scanf("%d", &combien);
    int *nombres_pairs = quelques_nombres_pairs(combien);

    for (int i=0; i<combien; i++)
    {
        printf("%d ", nombres_pairs[i]);
    }
    printf("\n");
}
```

Malheureusement, pour l'entrée `25` on voit s'afficher

```
0 32760 0 0 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48
```

Pourquoi ? Comment pourrait-on corriger ce code ?

Solution de l'exercice 5

La fonction `quelques_nombres_pairs` retourne un pointeur vers une variable locale qui **n'est plus valide** après la fin de l'appel à cette fonction. Quelques nombres pairs restent, mais il n'y a aucune garantie sur le contenu du tableau retourné.

Une solution possible est :

```
int* quelques_nombres_pairs_2(int n)
{
    int *tableau = malloc(n * sizeof(int));
    for (int i=0; i<n; i++)
    {
        tableau[i] = 2*i;
    }
    return tableau;
}
```

Il faut aussi se souvenir de libérer la mémoire dans `main` avec `free(nombres_pairs)`.

Exo6 Odd

Souvenez-vous de la cellule d'une liste chaînée :

```
typedef struct _cell_t
{
    int contenu;
    struct _cell_t *next;
} noeud_t;
```

Qu'arrive-t-il à la liste `1 -> 2 -> 4 -> 4 -> 3 -> -5 -> -2` si on appelle la fonction suivante avec le premier élément de la liste en argument ?

```
void pr(cell_t *pcell)
{
    if (pcell == NULL || pcell->next == NULL)
    {
        return;
    }

    if (pcell->next->contenu % 2)
    {
        pr(pcell->next);
    }
    else
    {
        cell_t *a = pcell->next;
        pcell->next = pcell->next->next;
        free(a);

        pr(pcell);
    }
}
```

Solution de l'exercice 6

La fonction vérifie si l'élément suivant est pair. Si oui, alors elle l'efface et elle s'appelle elle-même avec la même cellule en argument. Si non, alors elle avance à la cellule suivante. Donc cette fonction efface tous les éléments pairs de la liste (sauf le tout premier élément).

La liste finale sera `1 -> 3 -> -5`.