Remise à niveau en C
Fiche de TP 1

L3 Informatique 2020-2021

Introduction à la librairie NCURSES

Dans ce TP, nous allons introduire la librairie NCURSES qui permet de développer des applications complètes. Les prochains TP à venir seront des mini projets dans lesquels une application complète sera demandée. Cette dernière devra être programmée de sorte à proposer une interface NCURSES soignée, minimaliste et fonctionnelle.

1 Apprentissage des notions de base

La librairie NCRUSES offre une gestion radicalement performante de la sortie standard. Il devient possible, par exemple, d'écrire à un endroit arbitraire d'une fenêtre. Elle permet d'écrire véritablement des applications robustes et de qualité professionnelle. Voici quelques applications connues basées sur cette librairie :

— `vim` (https://www.vim.org, https://github.com/vim/vim), l'un des meilleurs éditeurs de textes pour programmeurs ;

— `ranger` (https://github.com/ranger/ranger), un explorateur de fichiers très efficace écrit en PYTHON ;

— `htop` (http://hisham.hm/htop, https://github.com/hishamhm/htop), un moniteur et gestionnaire de processus interactif ;

— `lynx` (http://lynx.browser.org), un navigateur internet en mode texte ;

— `moc` (http://moc.daper.net), un lecteur de musique complet et minimaliste.

Pour utiliser NCURSES, il nécessaire d'inclure l'en-tête `ncurses.h`. Un programme `Prog.c` minimal est

```c
#include <ncurses.h>

int main() {
    /* Initialise la fenetre. */
    initscr();

    /* Imprime une chaine de caracteres. */
    printw("Bonjour.");

    /* Rafraichit la fenetre. */
    refresh();

    /* Attend l'appui d'une touche. */
    getch();

    /* Ferme la fenetre. */
    endwin();

    return 0;
}
```

Il se compile par la commande

```
gcc Prog.c -lncurses
```

L'option `-lncurses` permet de lier la librairie NCURSES afin de créer l'exécutable. Le rôle de cette option sera élucidé en détail dans les cours à venir.

Pour apprendre l'utilisation basique d'une librairie, l'une des meilleurs choses à faire consiste à étudier des exemples et à se les approprier en les modifiant légèrement. C'est la stratégie adoptée par les exercices suivants.

Exercice 1. (Faire bonne impression)

1. En étudiant l'exemple

```c
#include <ncurses.h>

int main() {
    initscr();

    printw("1");

    move(2, 10);
    addch('2');
    addch('3');

    move(LINES - 1, COLS - 1);
    addch('4');
    mvaddch(4, 2, '5');
    mvprintw(3, 3, "ABCD");
    printw("**");

    refresh();
    getch();
    endwin();
    return 0;
}
```

   (a) décrire la géométrie de la fenêtre (position de l'origine et façon dont les positions sont indexées) ;

   (b) expliquer ce que représentent les entités `LINES` et `COLS` ;

   (c) décrire le rôle de la fonction `move` ;

   (d) comparer et expliquer les différences entre les fonctions `addch`, `mvaddch`, `printw` et `mvprintw` ;

   (e) écrire un programme dans lequel la chaîne de caractères `"4!+2!"` est imprimée en position `(8, 4)`, signifiant que le 1er caractère de la chaîne occupe cette position. Proposer trois versions : une première utilisant uniquement la fonction `addch` pour l'impression, une autre uniquement la fonction `mvaddch` et une dernière uniquement la fonction `mvprintw`.

2. En étudiant l'exemple

```c
#include <ncurses.h>

int main() {
    initscr();

    attron(A_NORMAL);
    printw("Normal␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣ABCabc012\n");

    attron(A_REVERSE);
    printw("Inverse␣␣␣␣␣␣␣␣␣␣␣␣:␣ABCabc012\n");
    attroff(A_REVERSE);

    attron(A_BOLD);
    printw("Gras␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣ABCabc012\n");
    attroff(A_BOLD);

    attron(A_UNDERLINE);
    printw("Souligne␣␣␣␣␣␣␣␣␣␣␣:␣ABCabc012\n");
    attroff(A_UNDERLINE);

    attron(A_REVERSE | A_BOLD);
    printw("Inverse␣et␣souligne␣:␣ABCabc012\n");
    attron(A_NORMAL);

    refresh();
    getch();
    endwin();

    return 0;
}
```

(a) décrire le rôle de la fonction `attron` et des arguments qu'elle accepte ;

(b) commenter la ligne 11 et en déduire le rôle de la fonction `attroff` ;

(c) écrire un programme dans lequel la chaîne de caractères `"*10*"` est imprimée en position `(0, 0)` en gras et souligné.

3. En étudiant l'exemple

```c
#include <ncurses.h>

int main() {
    initscr();

    start_color();
    init_pair(1, COLOR_RED, COLOR_CYAN);
    init_pair(2, COLOR_YELLOW, COLOR_BLACK);

    curs_set(0);

    attron(COLOR_PAIR(1));
    mvprintw(2, 3, "Abc123␣**␣␣*");
    attroff(COLOR_PAIR(1));

    attron(COLOR_PAIR(2));
    mvprintw(2, 16, "2121");
    attroff(COLOR_PAIR(2));

    refresh();
    getch();
    endwin();
    return 0;
}
```

(a) commenter la ligne 13 et en déduire le rôle de l'appel à fonction `curs_set` avec l'argument `0` ;

(b) modifier le programme de sorte à écrire en vert sur fond bleu la deuxième chaîne de caractères.

Exercice 2. (Dessins animés)

1. En étudiant l'exemple

```c
#include <ncurses.h>
#include <unistd.h>

#define DELAI 50000

int main() {
    int x, y;

    initscr();

    x = 0;
    y = 0;
    while(1) {
        clear();
        mvaddch(y, x, 'o');
        refresh();

        usleep(DELAI);
        x = (x + 1) % COLS;
        y = (y + 1) % LINES;
    }
    endwin();
    return 0;
}
```

(a) commenter la ligne 14 et en déduire le rôle de la fonction `clear` ;

(b) essayer de déplacer l'appel à `clear` au sein de la boucle et observer les différences de comportement obtenues ;

(c) même question que la précédente pour l'appel à `refresh` ;

(d) décrire précisément le rôle de la fonction `usleep` et en particulier l'unité dans laquelle est exprimé son paramètre. Cette fonction est apportée par le fichier d'en-tête `unistd.h`.

2. En étudiant l'exemple

```c
#include <ncurses.h>
#include <unistd.h>

#define DELAI 20000

int main() {
    int x, y;

    initscr();

    x = 0;
    y = 0;
    while(1) {
        clear();

        mvaddch(0, 0, '*');
        mvaddch(0, COLS - 1, '*');
        mvaddch(LINES - 1, 0, '*');
        mvaddch(LINES - 1, COLS - 1, '*');

        mvaddch(y, x, 'o');
        refresh();

        usleep(DELAI);
        x = (x + 1) % COLS;
        y = (y + 1) % LINES;
    }
    endwin();
    return 0;
}
```

(a) énoncer le problème principal que l'on peut visualiser ;

(b) une solution possible au problème précédent consiste à n'effacer que le nécessaire d'un tour de boucle à l'autre. Pour cela, il suffit d'imprimer le caractère espace aux endroits voulus. Proposer une solution utilisant ce procédé. Penser en premier lieu à supprimer l'appel à la fonction `clear` ;

(c) il est aussi possible d'utiliser la fonction `int delch()` ou la fonction `int mvdelch(int y, int x)` qui respectivement efface le caractère à l'endroit courant et qui efface le caractère en position `(y, x)`. Tenter de proposer une solution au phénomène précédent en utilisant l'une ou l'autre de ces fonctions ;

(d) essayer de déplacer l'appel à la fonction `refresh` dans les variantes précédentes et observer les différences de comportements dans les exécutions du programme.

Exercice 3. (Entrez s'il vous plaît !)

1. En étudiant l'exemple

```c
#include <ncurses.h>

int main() {
    char chaine[128];
    int entier;

    initscr();

    getstr(chaine);
    mvprintw(3, 0, "Chaine␣lue␣:␣%s", chaine);

    mvscanw(10, 0, "%d", &entier);

    mvprintw(11, 0, "Entier␣lue␣:␣%d", entier);

    refresh();
    getch();
    endwin();
    return 0;
}
```

(a) décrire le rôle de la fonction `getstr` ;

(b) écrire un programme dans lequel une chaîne de caractères est lue en position `(2, 4)` puis affichée en position `(0, 0)` *(Indice : penser à utiliser la fonction* `move` *avant l'appel à* `getstr` *ou utiliser la fonction* `mvgetstr`*.)* ;

(c) écrire un programme dans lequel un entier est tout d'abord demandé en position `(0, 0)`. Si celui-ci est nul, l'exécution se termine. Sinon, un nouvel entier est demandé en position `(1, 1)`, puis `(2, 2)`, puis `(3, 3)`, *etc.*, jusqu'à ce que l'utilisateur rentre la valeur `0`.

2. En étudiant l'exemple

```c
#include <ncurses.h>

int main() {
    int touche;
    int x, y;
    int x_prec, y_prec;

    initscr();
    noecho();

    x = COLS / 2;
    y = LINES / 2;

    mvaddch(y, x, 'o');

    while (1) {
        x_prec = x;
        y_prec = y;

        touche = getch();
        if (touche == 'q')
            x -= 1;
        if (touche == 'd')
            x += 1;
        if (touche == 'z')
```

```
            y -= 1;
        if (touche == 's')
            y += 1;
        mvaddch(y_prec, x_prec, ' ');
        mvaddch(y, x, 'o');
        refresh();
    }

    getch();
    endwin();
    return 0;
}
```

(a) commenter la ligne 9 et en déduire le rôle de la fonction `noecho` ;

(b) consolider le programme afin que l'objet ne puisse plus sortir de la fenêtre ;

(c) augmenter le programme d'une fonctionnalité permettant de repositionner l'objet en position initiale si la touche `'i'` est pressée ;

(d) augmenter le programme d'une fonctionnalité permettant de déplacer l'objet de plus d'une case à la fois. Pour ce faire, on dispose d'un pas, initialement égal à 1, et celui-ci peut-être incrémenté ou décrémenté respectivement par les touches `'p'` et `'m'`. La valeur du pas doit s'afficher en bas à gauche de la fenêtre sous la forme `"Pas : N"`, où `N` est la valeur du pas courant. L'objet doit maintenant se déplacer fidèlement au pas courant.

3. En étudiant l'exemple

```
#include <ncurses.h>
#include <unistd.h>

int main() {
    int touche, val, delai;

    initscr();
    noecho();
    nodelay(stdscr, TRUE);

    val = 0;
    delai = 1000000;
    mvprintw(0, 0, "Valeur : %3d", val);
    while (1) {
        touche = getch();
        if (touche != ERR) {
            if (touche == 'r')
                val = 0;
            if (touche == 'b')
                delai /= 2;
            if (touche == 't')
                delai *= 2;
        }
        mvprintw(0, 0, "Valeur : %3d", val);
        refresh();

        val = (val + 1) % 128;
        usleep(delai);
    }

    getch();
    endwin();
    return 0;
}
```

(a) Commenter la ligne 9 et en déduire le rôle de l'appel à la fonction `nodelay` avec les arguments `stdscr` (écran standard) et `TRUE`. *(Indice : ceci modifie le mode de comportement de la fonction* `getch`*.)* ;

(b) modifier le programme afin que la valeur entière affichée soit en gras (et uniquement celle-ci : la chaine de caractères `"Valeur : "` doit rester en écriture normale) :

(c) augmenter le programme d'une fonctionnalité permettant de quitter l'exécution par un appui sur la touche `'q'` ;

(d) par défaut, certaines touches du clavier ne sont pas accessibles. Pour rendre utilisables entre autres les flèches directionnelles, il faut incorporer l'appel `keypad(stdscr, TRUE)` au début du programme. De cette façon, remplacer l'utilisation des touches `'b'` et `'t'` respectivement par `KEY_UP` et `KEY_DOWN` ;

(e) par défaut, les touches pressées sont enregistrées dans un tampon d'entrée (exactement comme c'est le cas sur l'entrée standard). Il est possible d'incorporer l'appel à `cbreak()` au début du programme pour ne plus avoir ce comportement. Tester et comparer la version sans cet appel et la version avec cet appel.

Exercice 4. (La souris attrape le chat)

Dans l'exemple

```c
#include <ncurses.h>
#include <stdlib.h>
#include <time.h>

void dessiner_chat(int y, int x) {
    mvprintw(y, x, "*****");
    mvprintw(y + 1, x, "* * *");
    mvprintw(y + 2, x, "*****");
}

void effacer_chat(int y, int x) {
    mvprintw(y, x, "     ");
    mvprintw(y + 1, x, "     ");
    mvprintw(y + 2, x, "     ");
}

int main() {
    int touche;
    int chat_x, chat_y;
    int souris_x, souris_y;
    MEVENT ev;

    srand(time(NULL));

    initscr();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    nodelay(stdscr, TRUE);
    curs_set(0);
    mousemask(ALL_MOUSE_EVENTS
        | REPORT_MOUSE_POSITION, NULL);

    chat_x = rand() % (COLS - 4);
    chat_y = rand() % (LINES - 2);
    while (1) {
        touche = getch();
        if (touche == KEY_MOUSE
                && getmouse(&ev) == OK) {
            souris_x = ev.x;
            souris_y = ev.y;
            if ((chat_x <= souris_x)
                    && (souris_x <= chat_x + 4)
                    && (chat_y <= souris_y)
                    && (souris_y <= chat_y + 2)) {
                effacer_chat(chat_y, chat_x);
                chat_x = rand() % (COLS - 4);
                chat_y = rand() % (LINES - 2);
            }
        }
        dessiner_chat(chat_y, chat_x);
        refresh();
    }

    getch();
    endwin();
    return 0;
}
```

l'utilisateur doit cliquer sur le chat avec la souris. Un nouveau chat apparaît ensuite aléatoirement dans la fenêtre.

1. Modifier le programme afin qu'il affiche, lorsque la souris attrape le chat, le message `"Attrape !"` au centre de la fenêtre. Après exactement 500 ms, le message disparaît et un nouveau chat apparaît ensuite.

2. Modifier le programme de sorte que lorsque la souris touche l'un des deux yeux du chat, celui-ci se change en un `'X'`. Le chat n'est pas considéré comme attrapé dans ce cas. Les deux yeux du chat peuvent être ainsi touchés.

2 Notions dans la pratique

Exercice 5. (Damier)

Écrire un programme qui affiche un damier de dimensions 10×10 dont le coin inférieur gauche est au centre de la fenêtre. Les cases alternent entre la couleur rouge et verte.

Exercice 6. (Triangle)

Écrire un programme qui demande en position `(0, 0)` un entier positif n et qui affiche ensuite à partir de la position `(1, 0)` un triangle fait de lignes de `'*'` de longueur 1, puis 2, ..., jusqu'à n. Par exemple, si $n = 3$, les lignes affichées sont

```
*
**
***
```

Exercice 7. (Marche aléatoire)

Écrire un programme dans lequel `'o'` est affiché au centre de la fenêtre. Toutes les secondes, une case voisine orthogonalement au `'o'` est sélectionnée aléatoirement et le `'o'` se déplace sur cette nouvelle case. L'ancienne case occupée est au passage remplacée par un `'x'`. Par exemple, un exécution possible pourrait produire au bout de 28 secondes d'exécution l'affichage

```
xxx
  x
o xx
xxxxxxxx
   x   x
 xxx   x
 xxxxxxx
```

L'utilisateur peut à tout moment doubler la vitesse d'exécution en appuyant sur la flèche du haut, diviser la vitesse d'exécution de moitié en appuyant sur la flèche du bas ou mettre l'exécution en pause en appuyant sur la touche entrée. Un nouvel appui sur la touche entrée relance l'exécution.

Exercice 8. (Clics)

Écrire un programme qui se lance sur une fenêtre remplie de `'0'`. Lorsque l'utilisateur clique sur un case, la valeur de celle-ci est incrémentée. Si une case contient `'9'`, un clic sur cette dernière la remplace par un carré bleu. Les carrés bleus ne peuvent plus être modifiés par la suite.

3 Préparations pour la suite

Exercice 9. (Récapitulons !)

Reprendre toutes les nouvelles fonctions vues dans les exercices précédents en faisant une table dans laquelle apparaît leur prototype, rôle des paramètres, valeur renvoyée et une documentation succincte. L'objectif est de se constituer un aide-mémoire personnalisé pour gagner du temps pour les TP suivants.

Exercice 10. (Ouvertures)

1. Ouvrir la page du manuel de NCURSES (commande `man ncurses`) et la parcourir. Une version interactive se trouve à

   https://www.mkssoftware.com/docs/man3/ncurses.3.asp

Lire les premières parties pour avoir un apperçu officiel de la librairie.

2. Se rendre sur les pages

   https://www.gnu.org/software/ncurses/ncurses-intro.html

   http://www.tldp.org/HOWTO/NCURSES-Programming-HOWTO/

   http://www.cs.ukzn.ac.za/~hughm/os/notes/ncurses.html

Celles-ci contiennent de nombreux exemples et informations qui couvrent presque l'intégralité des besoins pour les TP suivants.