TD - Introduction Département
Cédric Zanni
Pierre-Etienne Moreau

Ascenseur

Utilisation de GitHub

Afin de continuer à vous familiariser avec git, le TD devra être réalisé dans un dépôt git. Des commits réguliers devront être réalisés et il faudra effectuer votre développement dans des branches.
Si vous n'avez pas fini la première partie du TD sur git (avant Divers), faites le. Je vous conseille également de lire la partie relative à gitignore dans la partie Divers. Si vous terminez ce TD en avance, vous pouvez continuez à vous familiariser avec Git en mettant en pratique ce qui est décrit dans la partie Divers.

Mise en place d'un Makefile

Ecrire un fichier Makefile permettant de facilement (re-)compiler un programme, de supprimer les fichiers intermédiaires, de changer le nom de l'exécutable, de choisir les options de compilation…

Il faudra bien entendu mettre à jour ce fichier au fur et à mesure du TD. Je vous recommande de diviser votre travail en plusieurs fichiers.

Installation de ncurses

ncurses est une librairie permettant de faciliter l'affichage d'une interface texte dans un terminal. Il vous faut tout d'abord récupérer le code source à l'adresse suivante :
https://ftp.gnu.org/pub/gnu/ncurses/

Pour le compiler, placez vous dans le répertoire racine de ncurses et exécutez:

```
./configure
make
```

le temps de compilation est assez long (plusieurs librairies et programmes de test), cela montre l'importance de ne pas re-compiler directement l'intégralité de son projet mais seulement les sous parties qui en ont besoin.

Localisez ncurses.h et libncurses.a grâce à la commande `find`.
Depuis le terminal, copier les répertoires *include* et *lib* qui les contiennent vers votre projet.

Vous pouvez également tester certains programmes de test (dans le répertoire *test*) pour vérifier que tout fonctionne correctement. Vous aurez pour cela besoin d'exécuter la commande :

```
export TERM=xterm-color
```

cela définit ce que l'on appelle une variable d'environnement, cette dernière peut être utilisée dans le terminal de la manière suivante :

```
echo $TERM
```

Utilisation de ncurses

Pour vous familiariser avec la librairie, implémentez le déplacement d'un caractère dans la zone d'affichage.

Fonctions d'affichages utiles dans la librairie ncurses:

```
// Initialisation de la librairie
initscr();
// Création de la fenêtre
WINDOW * win = newwin(h, l, offset_y, offset_x);
// Finalisation en fin de programme (libération de ressource, …)
endwin();

// efface le contenu de la fenêtre
?? wclear(WINDOW *);
// met à jour l'affichage
?? wrefresh(WINDOW *);

// dessine une boite autour de la fenêtre
?? box(WINDOW *, 0,0);

// dessine ligne verticale ou horizontal dans la fenêtre
?? wvline(WINDOW *,char,int);
?? whline(WINDOW *,char,int);

// dans les fonctions suivante les deux premiers entier correspond à des coordonnées dans la fenêtre
?? void wmove(WINDOW *,int,int); // déplace le curseur (utile pour wvline et whline)
?? mvwprintw(WINDOW *, int , int, const char*, ...); //affiche une chaîne de caractère sur le même principe que printf
?? mvwaddch(WINDOW *, int, int, char); // affiche un unique caractère
```

Fonction d'interaction avec le clavier :

```
?? keypad(WINDOW *,bool); // autorise l'utilisation des flèches et autre touches spéciales (macro KEY_UP, KEY_DOWN, ....)
int wgetch(WINDOW *win);  // récupère la dernière touche appuyé
void noecho();  // empèche le caractère frappé au clavier d'être afficher
```

Les ascenseurs : enfin ....

ci dessous se trouve le code du simulateur ainsi que les les fichiers *.h définissant les interfaces.

A vous d'écrire le code implantant ces interfaces.
Une fonctions est particulière, elle est appelé régulièrement à partir de main :

- `void stepElevator(Building *b);`
  cette fonction simule le déplacement de l'ascenseur :
  - si l'étage courant de l'ascenseur correspond à sa destination, faire en sorte que les personnes entrent et sortent de la cabine
  - sinon, l'étage courant est incrémenté ou décrémenté de 1 pour se rapprocher de sa destination

Nous vous conseillons d'implanter ces 2 fonctions :

- `PersonList* exitElevator(Elevator *e);`
  lorsque l'ascenseur arrive à un étage cette fonction est appelée (par `stepElevator`)
  cette fonction renvoie la liste des personnes qui sortent de l'ascenseur
  restent dans l'ascenseur les personnes qui ne sont pas sorties

- `PersonList* enterElevator(Elevator *e, PersonList *waitingList);`
  lorsque l'ascenseur arrive à un étage cette fonction est appelée (par `stepElevator`)
  cette fonction fait entrer dans l'ascenseur les personnes qui attendent (`waitingList`) et renvoie la nouvelle liste d'attente (i.e. la liste initiale moins les personnes qui sont entrées dans l'ascenseur)
  L'ascenseur ne peut pas accueillir plus de personnes que sa capacité maximale

person.h

```
#ifndef PERSON_H
#define PERSON_H

typedef struct _Person {
    int src;
    int dest;
} Person;

typedef struct _PersonList {
    Person *person;
    struct _PersonList *next;
} PersonList;

Person* createPerson(int src, int dest);
PersonList* insert(Person *p, PersonList *list);

#endif
```

elevator.h

```
#ifndef ELEVATOR_H
#define ELEVATOR_H

#include "person.h"

typedef struct _Elevator {
    int capacity;    // capacité maximale de la cabine
    int currentFloor;// étage courant
    int targetFloor; // destination
    PersonList *persons; // personnes dans la cabine
} Elevator;

typedef struct _Building {
    int nbFloor; // nombre d'étage des l'immeuble
    Elevator *elevator; // la cabine d'ascenseur
     PersonList **waitingLists; // array of waiting list (one
per floor)
} Building;

Elevator  *create_elevator(int  capacity,  int  currentFloor,
PersonList *persons);
Building  *create_building(int  nbFloor,  Elevator  *elevator,
PersonList **waitingLists);

PersonList* exitElevator(Elevator *e);
PersonList* enterElevator(Elevator *e, PersonList *list);
void stepElevator(Building *b);

#endif
```

main.c

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

#include "elevator.h"
#include "person.h"

#define HEIGHT 30
#define WIDTH 40
#define PERSON_WIDTH 3

void DisplayPersonList(WINDOW *win, PersonList *list, int level, int offset)
{
  while(list != NULL) {
    // display 25 for a person going from floor 2 to floor 5
    mvwaddch(win, level, offset, '0' + list->person->src);
    mvwaddch(win, level, offset+1, '0' + list->person->dest);
```

```
    list = list->next;
    offset+= PERSON_WIDTH;
  }
}

void DisplayElevator(WINDOW *win, int nbFloor, Elevator *e, int offset) {
  //Display elevator
  // [23 24 31 30 42]

  int level = 3*(nbFloor - e->currentFloor); // 3 lines per level
  mvwaddch(win, level, offset+1, '[');
  DisplayPersonList(win, e->persons, level, offset+2);
  mvwaddch(win, level, offset+2+ (PERSON_WIDTH*e->capacity), ']');
}

void DisplayBuilding(WINDOW *win, Building *b) {
  int offset = 1;

  // display wall
  // |                |
  // |[23 24 31 30 42]| 31 32
  // |                |
  int right_wall = offset + 3 + (PERSON_WIDTH*b->elevator->capacity);
  for(int i=0; i < b->nbFloor; ++i) {
    int level = 3*i+1;
    mvwaddch(win,level,  offset,'|');
    mvwaddch(win,level+1,offset,'|');
    mvwaddch(win,level,  right_wall,'|');
    mvwaddch(win,level+1,right_wall,'|');
  }
  for(int i=offset+1; i < right_wall; i++) {
    mvwaddch(win,3*(b->nbFloor)+1,i,'_');
  }

  DisplayElevator(win, b->nbFloor, b->elevator, offset);

  for(int i=0; i < b->nbFloor; i++) {
    int level = 3*(b->nbFloor - i);
    DisplayPersonList(win,b->waitingLists[i], level, right_wall + 2);
  }
}


int main() {
  srand(time(NULL));   // should only be called once

  // generate list of waiting persons
  int nbFloor = 5;
       PersonList **waitingLists = malloc(nbFloor*sizeof(PersonList*));
  for(int currentFloor=0; currentFloor < nbFloor; currentFloor++) {
    waitingLists[currentFloor] = NULL;
    int nbPerson = 5; // 5 persons in the waiting list
    for(int j=0 ; j<nbPerson ; j++) {
      int dest = rand() % (nbFloor);
```

```
      Person *p = createPerson(currentFloor, dest);
      waitingLists[currentFloor] = insert(p,waitingLists[currentFloor]);
    }
  }

  // Initialize building and elevator
  int capacity = 3;
  int currentFloor = 0;
  Elevator *elevator = create_elevator(capacity, currentFloor , NULL);
  Building *building = create_building(nbFloor, elevator, waitingLists);

  // Initialize ncurse display
  initscr(); // initialize ncurses
  noecho();  // do not display in window the pressed keys
  halfdelay(2);

  WINDOW *win = newwin(HEIGHT, WIDTH, 0, 0);

  // Animation loop
  bool run=true;
  while(run) {
    // Generate people in function of input (or quit if 'q')
    int input = wgetch(win);
    if(input == 'q') {
      run = false;
    } else {
      int level = input - '0';
      if(0 <= level && level < nbFloor) {
       building->elevator->targetFloor = level;
      }
    }

    // Update state machine of elevator !!!!

    stepElevator(building);

    wclear(win);   // clear display area
    box(win, 0,0); // display border of window

    DisplayBuilding(win, building);

    wrefresh(win); // actual display function

  }

  endwin(); // correct ending of ncurses

  return 0;
}
```