Licence d'informatique
Module de C/Unix

Examen – Partie C

Jean-Luc LEVAIRE

Janvier 1999

Les documents de cours et TD sont autorisés.

On rendra deux copies séparées pour la partie Unix et pour la partie langage C. La partie C est à rendre sur une *copie de couleur blanche*.

1 Questions de cours

Question 1
Donner la représentation schématique de `i` et de `p` avant exécution de la boucle `for`. Qu'affiche le programme ? Quel est le contenu du tableau `i` à la fin du programme ?

```
void
f(int **i)
{
  *((*i++)++)++;*((*i)++)--;
}

void
main(void)
{
  int i[2][5] = {{0, 1, 2, 3, 4}, {2, 3, 4, 5, 6}}, *p[2] = {*i, *(i+1)};
  int j;

  for (j=0; j < 5; j++) {
    f(p);
    printf("%d %d\n", *(*p-1), *(*(p+1)-1));
  }
}
```

Question 2
Comment, dans le système Unix, un nom est-il associé à un fichier ? Quelles sont les différences entre entrées-sorties par appel système et entrées-sorties par librairie ?

Question 3
Que signifient les déclarations suivantes :
- `float (*d1[2])(int);`
- `char *(*(*d2)(int))(int);`

2 Du recouvrement des fenêtres

Nous souhaitons manipuler des fenêtres dans un environnement graphique sur une station de travail. Une fenêtre est un rectangle spécifié par la coordonnée (x, y) de son origine (coin supérieur gauche) par rapport à l'origine de l'écran (coin supérieur gauche de l'écran de coordonnées $(0, 0)$), et par ses dimensions (w, h) (sous la forme (largeur, hauteur)).

Id : xamc99.tex,v 1.2 2000/11/18 09:41:32 marquet Exp

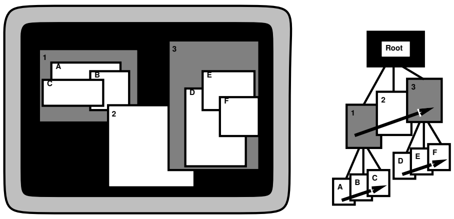

FIG. 1 – La hiérarchie des fenêtres.

Question 4
Proposer un type `Window` pour représenter une fenêtre, et un type `WindowList` pour représenter une liste de fenêtres. Proposer aussi un type `Bool` pour représenter le type booléen.

Question 5
Écrire une fonction

```
Bool Recover(Window w1, Window w2) ;
```

qui teste si les fenêtres `w1` et `w2` se recouvrent (ont une zone commune à l'écran).

Question 6
Écrire une fonction

```
WindowList WindowVisible(Window w1, Window w2) ;
```

qui retourne dans une liste de fenêtres les zones visibles (qui sont des rectangles) de `w1` quand `w2` est en avant-plan de `w1`. Les zones visibles ne doivent pas se recouvrir.

Question 7
Écrire une fonction

```
WindowList WindowListVisible(Window w, WindowList wlist) ;
```

qui retourne dans une liste de fenêtres les zones visibles de `w` quand les fenêtres de la liste `wlist` recouvrent la fenêtre `w`. Les zones visibles ne doivent pas se recouvrir. (Suggestion : appeler la fonction `WindowVisible` sur la premiere fenetre de `wlist`, puis recommencer avec les zones trouvées précédemment avec la seconde fenêtre de `wlist`, etc)

Dans un environnement graphique, les fenêtres sont organisées de manière hiérarchique : la fenêtre Root est la racine de cet arbre, et correspond au fond de l'écran. Puis les filles de Root sont les fenêtres Toplevel, et sont gérées par le window manager (en ce qui concerne leur position et leur taille). Enfin les fenêtres descendantes des Toplevel sont des fenêtres ordinaires gérées directement par l'application liée au Toplevel correspondant. Elles ne peuvent pas déborder de leur fenêtre Toplevel, mais peuvent aussi se recouvrir à l'intérieur de leur Toplevel. La figure 1 présente un exemple d'une telle hiérarchie.

On propose de représenter cette hiérarchie par le type suivant :

```
typdef struct noeud {
  Window *fenetre;
  WindowList visibles; /* Liste des zones visibles */
  struct noeud *soeurs; /* Liste des noeuds des fenetres soeurs */
  struct noeud *filles; /* Liste des noeuds des fenetres filles */
  /* dans l'ordre de leur profondeur dans le recouvrement       */
  struct noeud *parente; /* Noeud de la fenetre parente */
} noeud, *WindowTree ;
```

L'ordre dans la liste des fenêtres sœurs détermine leur profondeur dans le recouvrement (d'arrière-plan vers avant-plan).

Question 8
Écrire une fonction

```
WindowTree Locate(int x, int y, WindowTree wt) ;
```

qui renvoie, pour une hiérarchie de fenêtres `wt`, un pointeur sur le nœud de cette hiérarchie tel que le point de coordonnées (x ,y) est dans une zone visible de la fenêtre pointée par le champ fenêtre de ce nœud. Elle renvoie `NULL` si le point est en dehors de la fenêtre associée à `wt`.

Question 9
Écrire un procédure

```
void ComputeVisible(WindowTree wt) ;
```

qui calcule la liste des zones visibles (champ `visible`) de la fenêtre associée au nœud `wt`. Si une telle liste existait déjà, cette procédure détruit l'ancienne liste.