Programmazione

Andrea Piroddi

Dipartimento di Informatica, Scienza e Ingegneria

Elaborato 8

L'operatore ->

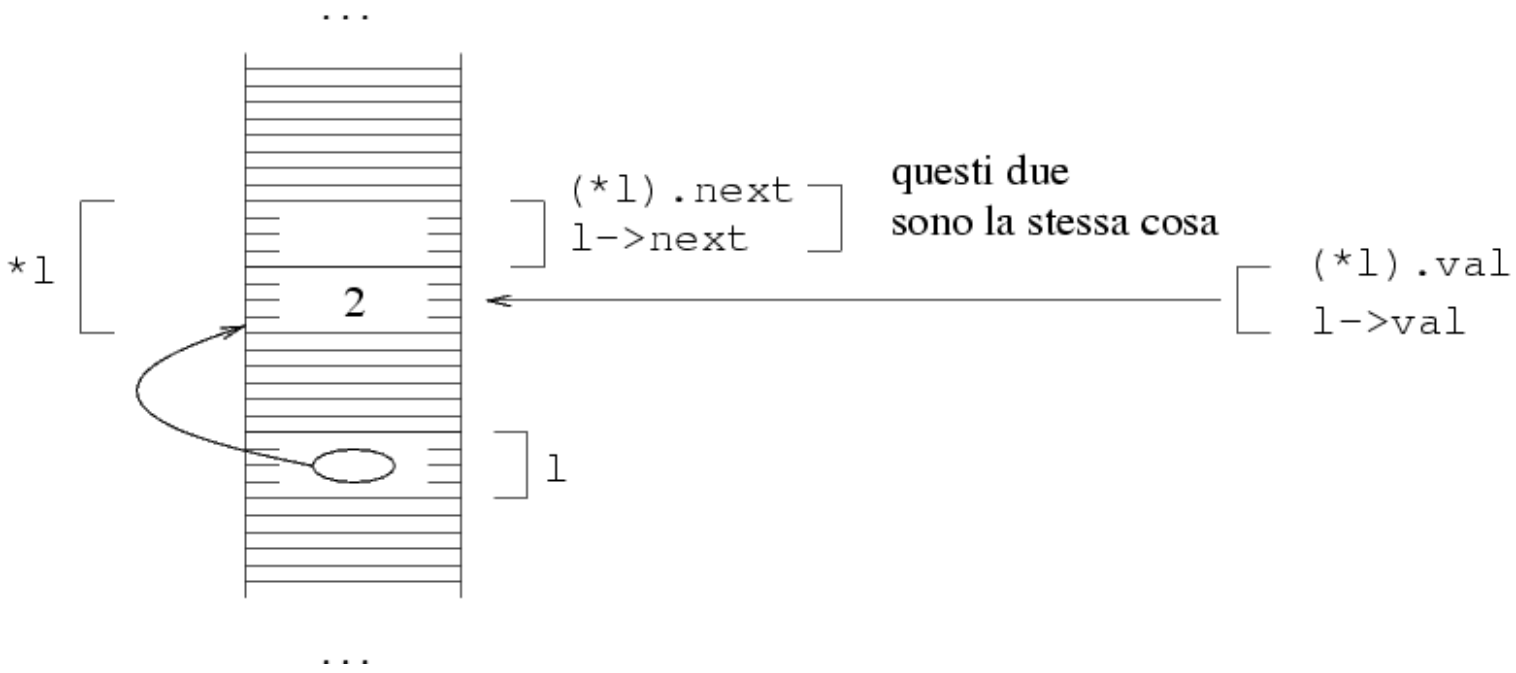
Già dai pochi esempi di programmi sulle liste visti fino a questo momento risulta chiaro che appaiono molto spesso espressioni del tipo

(*1).next In cui 1 è un puntatore a una struttura. Il significato è : prendi la struttura puntata da **1**, e di questa prendi il campo ***next***.

Dal momento che questa espressione si ripete molto spesso, il C ne fornisce una forma abbreviata:

1->next

Questa espressione ha esattamente lo stesso significato della precedente. A sinistra della freccia deve apparire un puntatore a una struttura, e a destra il nome di uno dei campi di questa struttura. Questa espressione individua la sottovariabile della struttura 1 il cui nome è ***next***.



Elaborato 8

Definiamo le strutture di base

```
const struct position UNK_POSITION = {-1,-1}; // Setta la posizione unknown a (UINT_MAX,UINT_MAX)

struct ghost {
          int id;  //questo è l'id del fantasma
  int status;
  int dir;
  struct position pos;
};

struct ghosts {
  char **A;
  unsigned int nrow;
  unsigned int ncol;
          unsigned int n;
  struct ghost *ghost;
};

static struct position ghost_move_normal(struct ghosts *G, struct pacman *P, unsigned int id);

static struct position  ghost_move_scared(struct ghosts *G, struct pacman *P, unsigned int id);

static struct position  ghost_move_eyes(struct ghosts *G, struct pacman *P, unsigned int id);
```

Elaborato 8

- ***ghosts setup():*** *crea la struttura dati ideata per* ***memorizzare*** *le informazioni di num_ghosts fantasmi e ne ritorna un puntatore.*

```
struct ghosts *ghosts_setup(unsigned int num_ghosts) {
        struct ghosts *G = (struct ghosts *)malloc(sizeof(struct ghosts));
        srand(time(NULL));
        if(G != NULL) { //se G non punta a  NULL
                unsigned int i;
                // al campo n di G assegniamo num_ghosts
                G->ghost = (struct ghost *)calloc(num_ghosts,sizeof(struct ghost));//al campo ghost di G assegniamo dinamicamente la memoria
        for(i = 0; i < G->n; i++) { //per ogni ghost presente assegniamo una UNK_POSITION e una direzione casuale
                                                        }
        }

        return G;
}
```

Elaborato 8

- **ghosts destroy(): libera tutta la memoria** allocata dinamicamente per la creazione della struct ghosts.

```
void ghosts_destroy(struct ghosts *G) {

        //se G non punta a  NULL libera la memoria
}
```

Elaborato 8

- **ghosts set arena(): memorizza la matrice di gioco che indica le posizioni in cui i fantasmi potranno muoversi (vedi sotto).**

positions.txt

```
1  xxxxxxxxxxxxxxxxxxxxxxxxxxxx
2  x    G       xx            x
3  x xxxx xxxxx xx xxxxx xxxx x
4  x xxxx xxxxx xx xxxxx xxxx x
5  x xxxx xxxxx xx xxxxx xxxx x
6  x  G                       x
7  x xxxx xx xxxxxxxx xx xxxx x
8  x xxxx xx xxxxxxxx xx xxxx x
9  x      xx    xx    xx      x
10 xxxxxx xxxxx xx xxxxx xxxxxx
11 xxxxxx xxxxx xx xxxxx xxxxxx
12 xxxxxx xx    G     xx xxxxxx
13 xxxxxx xx xx    xx xx xxxxxx
14 xxxxxx xx x      x xx xxxxxx
15  G       x  XX  x
16 xxxxxx xx xG  G Gx xx xxxxxx
17 xxxxxx xx xxxxxxxx xx xxxxxx
18 xxxxxx xx          xx xxxxxx
19 xxxxxx xx xxxxxxxx xx xxxxxx
20 xxxxxx xx xxxxxxxx xx xxxxxx
21 x            xx            x
22 x xxxx xxxxx xx xxxxx xxxx x
23 x xxxx xxxxx xx xxxxx xxxx x
24 x   xx        <       xx   x
25 xxx xx xx xxxxxxxx xx xx xxx
26 xxx xx xx xxxxxxxx xx xx xxx
27 x      xx    xx    xx      x
28 x xxxxxxxxxx xx xxxxxxxxxx x
29 x xxxxxxxxxx xx xxxxxxxxxx x
30 x     G                     x
31 xxxxxxxxxxxxxxxxxxxxxxxxxxxx
32
```

Descrizione dell'arena di gioco passata alla funzione ghosts set arena():

- L'arena viene passata come matrice di caratteri.
- Tale matrice `e ricavata dal file positions.txt e *processata* in arena.c: vengono rimossi i simboli corrispondenti a pacman e fantasmi e vengono calcolati i percorsi per ritornare nella home.
- Ogni cella contiene un carattere che indica se tale cella `e attraversabile oppure no (contiene un muro).
- Ogni cella attraversabile contiene una lettera che indica la direzione da seguire per poter ritornare velocemente nella casa dei fantasmi (vedi file path.txt nel pacchetto sorgente):
  - Walls: x (XWALL SYM)
  - Home: X (HOME SYM)
  - Up: U (UP SYM)
  - Left: L (LEFT SYM)
  - Right: R (RIGHT SYM)
  - Down: D (DOWN SYM)

```
void ghosts_set_arena(struct ghosts *G, char **A, unsigned int nrow, unsigned int ncol) {
        // se G è NON nullo
                G->A = A; // assegniamo al campo A di G la matrice di gioco
                // assegniamo al campo numero di righe il numero di righe e cosi per le colonne


}
```

Elaborato 8

- **ghosts set position(): setta la posizione per il fantasma con identificativo id.**

void ghosts_set_position(struct ghosts *G, unsigned int id, struct position pos) {

if(G != NULL && id < G->n) G->ghost[id].pos = pos; \\ se G non punta a NULL e il suo identificativo è valido assegniamo al fantasma con quell'id la posizione

}

Elaborato 8

- **ghosts set status(): setta lo stato (NORMAL, SCARED NORMAL, SCARED BLINKING, EYES) del fantasma con identificativo id.**

```
void ghosts_set_status(struct ghosts *G, unsigned int id, enum ghost_status status) {

        // se G non punta a  NULL e l'id è valido, assegniamo al ghost con quell'id uno stato

}
```

Elaborato 8

- **ghosts get number(): ritorna il numero totale di fantasmi**.

```
unsigned int ghosts_get_number(struct ghosts *G) {

        //restituisce il numero di fantasmi
}
```

Elaborato 8

- **ghosts get position(): ritorna la posizione nell'arena del fantasma con identificativo id.**

struct position ghosts_get_position(struct ghosts *G, unsigned int id) {

return G != NULL && id < G->n ? G->ghost[id].pos : UNK_POSITION; // restituisce la posizione del fantasma con identificativo id facendo un opportuno controllo sugli id

}

Elaborato 8

- **ghosts get status(): ritorna lo stato del fantasma con identificativo id.**

enum ghost_status ghosts_get_status(struct ghosts *G, unsigned int id) {

// restituisce lo stato dell fantasma con identificativo id facendo un opportuno controllo sugli id

}

Elaborato 8

- **ghosts_get_number()**: **ritorna n se G non punta a NULL.**

```
unsigned int ghosts_get_number(struct ghosts *G) {
        return G != NULL ? G->n : 0;
}
```

Elaborato 8

- **ghosts move(): sposta il fantasma con identificativo id secondo i seguenti vincoli:**

```
struct position ghosts_move(struct ghosts *G, struct pacman *P, unsigned int id) {
        switch(ghosts_get_status(G,id)) {
                case NORMAL:
                        return ghost_move_normal(G,P,id);
                case SCARED_NORMAL:
                        …
                case SCARED_BLINKING:
                        …
                case EYES:
                        …
                case UNK_GHOST_STATUS:
                        …;
        }
}
```

Il movimento di un fantasma può essere random (gioco noioso), a patto che rispetti i vincoli definiti nelle specifiche.

L'obiettivo dovrebbe essere quello di cercare di implementare un movimento non banale, seguendo i seguenti principi:

- se lo stato del fantasma `e **NORMAL** allora il movimento deve cercare di portarlo vicino a pacman;
- se lo stato del fantasma `e **SCARED NORMAL** oppure **SCARED BLINKING** allora il movimento deve cercare di portarlo lontano da pacman;
- se lo stato del fantasma `e **EYES** allora il movimento deve cercare di riportarlo in una delle posizioni HOME (le posizioni nella matrice marcate con X) in modo che possa riprendere la forma normale. In quest'ultimo caso `e sufficiente scegliere la direzione di movimento suggerita nella posizione corrente della matrice arena.

Elaborato 8

- **legal_position()**: **ritorna n se G non punta a NULL.**

```
static int legal_position(struct ghosts *G, struct pacman *P, struct position pos, enum ghost_status status) {
        if(IS_WALL(G->A,pos)) {
                return 0;
        } else {
                unsigned int i;
                struct position p = pacman_get_position(P);

                // check pacman intersect
                if(status != NORMAL && pos.i == p.i && pos.j == p.j)
                        return 0;

                // check ghost intersect
                …
                }
                return 1;
        }
}
```

Elaborato 8

- **ghost_move()**

```
static struct position ghost_move(struct ghosts *G, struct pacman *P, unsigned int id, enum direction dir[]) {
  struct position pos = G->ghost[id].pos;
  int d;

  for(d = 0; d < 4; d++) {
                    struct position new = new_position(pos,dir[d],G->nrow,G->ncol);
    if(legal_position(G,P,new,G->ghost[id].status)) {
                                G->ghost[id].pos = new;
                                G->ghost[id].dir = dir[d];
      return new;
    }
  }
          return pos;
}
```

Elaborato 8

- **setup_remaining_dir()**

```
static void setup_remaining_dir(struct ghosts *G, unsigned int id, enum direction dir[]) {
            int tmp[4] = {0}, d, i;

            dir[1] = dir[2] = dir[3] = UNK_DIRECTION;
            tmp[dir[0]] = 1; // keeps track of the already selected directions

            // Higher priority to the current direction
if(dir[0] !=  G->ghost[id].dir) {
                        dir[1] =  G->ghost[id].dir;
                        tmp[dir[1]] = 1;       }
            // Lower priority to the direction opposite to the current one
            …
             }

            // Random selection of the  remaining directions
            for(i = 0; i <= 3; i++)
                        if(dir[i] == UNK_DIRECTION) {
                                    do {d = rand() % 4;} while(tmp[d] == 1);
                                    dir[i] = d;
                                    tmp[d] = 1;
                        }
}
```

Elaborato 8

- **nearby_home()**

```
static int nearby_home(char **A, unsigned int nrow, unsigned int ncol, struct position pos) {
        int n = 2, a, b, i = pos.i, j=pos.j;
        for(a=-n; a<=n; a++)
                if(i+a >=0 && i+a<nrow)
                        for(b=-n; b<=n; b++)
                                if(j+b >=0 && j+b<ncol)
                                        if(A[i+a][j+b] == HOME_SYM)
                                                return 1;
        return 0;
}
```

Elaborato 8

- **`select_dir_towards()`**

```
static void select_dir_towards(struct ghosts *G, struct pacman *P, unsigned int id, enum direction dir[]) {
        struct position g = ghosts_get_position(G,id);
        struct position p = pacman_get_position(P);

        if(nearby_home(G->A,G->nrow,G->ncol,g))
                dir[0] = UP;
        else if(g.i == p.i)
                …
        else if(g.j == p.j)
                …
        else
                dir[0] = G->ghost[id].dir;

        setup_remaining_dir(G,id,dir);
}
```

Elaborato 8

- **select_dir_away()**

```
static void select_dir_away(struct ghosts *G, struct pacman *P, unsigned int id, enum direction dir[]) {
        struct position g = ghosts_get_position(G,id);
        struct position p = pacman_get_position(P);

        if(nearby_home(G->A,G->nrow,G->ncol,g))
                dir[0] = DOWN;
        else if(g.i == p.i)
                …
        else if(g.j == p.j)
                …
        else
                dir[0] = G->ghost[id].dir;

  setup_remaining_dir(G,id,dir);
}
```

Elaborato 8

- **`select_dir_home()`**

```
static void select_dir_home(struct ghosts *G, unsigned int id, enum direction dir[]) {
        int i  = G->ghost[id].pos.i;
        int j  = G->ghost[id].pos.j;
        char c = G->A[i][j];

        dir[0] = c == 'L' ? LEFT: c == 'R' ? RIGHT : c == 'U' ? UP : DOWN;
        setup_remaining_dir(G,id,dir);
}
```

Elaborato 8

- **`ghost_move_normal()`**

```
static struct position ghost_move_normal(struct ghosts *G, struct pacman *P, unsigned int id) {
…
return ghost_move(G,P,id,dir);
}
```

Elaborato 8

- **`ghost_move_scared()`**

```
static struct position ghost_move_scared(struct ghosts *G, struct pacman *P, unsigned int id) {
  …
  return ghost_move(G,P,id,dir);
}
```

Elaborato 8

- **`ghost_move_eyes()`**

```
static struct position ghost_move_eyes(struct ghosts *G, struct pacman *P, unsigned int id) {
  ...
  return ghost_move(G,P,id,dir);
}
```