Il terminale, la console e il linguaggio C

Lo **scopo** dell'**istruzione** è quello di trasformare gli **specchi** in **finestre**.

Sydney J. Harris

La console testuale, la mitica "finestra nera" per i non addetti ai lavori, è da sempre croce e delizia per noi sviluppatori software. Nella sua apparente semplicità, tale ambiente si mostra amichevole o perfido a seconda della conoscenza che noi abbiamo di lui, della sua struttura interna e dei comandi che accetta. Ho usato volutamente il pronome personale "lui" in quanto noi informatici siamo soliti attribuire una personalità agli oggetti con i quali sistematicamente interagiamo nella nostra vita quotidiana.
Ogni sistema software ha da sempre una interfaccia a caratteri, la cosiddetta shell testuale, che ci permette di interagire con il sistema sottostante: nel mondo Unix-Linux essa prende il nome di **terminale** mentre nel contesto Windows è chiamata più spesso **console** o anche **cmd.exe**, dal nome del file eseguibile che la manda in esecuzione.
In questo breve viaggio cercherò di mostrare come interagire, usando il linguaggio C, con la console in questione per realizzare delle semplici operazioni di spostamento del cursore da un punto ad un altro della finestra. Infatti, una delle difficoltà maggiori che abbiamo nella gestione di tale finestra è la sua assoluta rigidità dovuta al fatto che il prompt dei comandi è apparentemente immutabile nella sua acquisizione di un comando da tastiera e nella relativa restituzione dell'output del comando stresso. Se questa situazione è del tutto scontata e naturale in un contesto classico di gestione testuale lo stesso non vale nel momento in cui vogliamo gestire in maniera un po' più "grafica" la nostra console. Mi riferisco qui alla possibilità di realizzare un minimo di interattività all'interno della finestra muovendo, ad esempio, il cursore a piacimento in una specifica posizione al fine di simulare un classico ambiente a menu piuttosto che un rudimentale videogioco pseudo-grafico.

Il terminale e il mondo Unix-like

Nel mondo cosiddetto Unix-like, ovvero nel contesto di derivazione Unix quale Linux e macOS, il nostro terminale può essere gestito al meglio facendo uso della libreria di funzioni **ncurses** il cui sito ufficiale è:

```
https://invisible-island.net/ncurses/
```

Ovviamente ci si dovrà accertare della disponibilità della librerie in questione. A solo titolo di esempio, l'installazione in un sistema Debian-Ubuntu la si realizza con il gestore di pacchetti apt nel seguente modo:

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

Una volta risolti i problemi di installazione si potrà editare il proprio file sorgente con un editor di propria scelta. Io uso per lo più l'editor **nano** e, solo quando costretto dalle circostanze, il mitico **vim**.
Un "salve mondo" con ncurses, ovvero il tipico primo programma che si realizza per un qualsiasi ambiente di programmazione potrebbe essere il seguente:

```
#include <ncurses.h>
int main()
{
   initscr();
   printw("Salve mondo da ncurses!");
   refresh();
   getch();
```

```
    endwin();

    return 0;
}
```

Tale semplicissimo snippet (pezzo di codice sorgente) dovrebbe far comprendere almeno in linea generale la modalità di utilizzo della libreria in questione.
La prima cosa che faccio notare è il fatto che ho inserito il solo include della libreria ncurses.h in quanto la sola riga:

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

Include in automatico altre librerie tra cui anche `stdio.h`.
La prima istruzione che incontriamo, la chiamata alla funzione `initscr()` inizializza l'ambiente ncurses, ambiente che viene poi chiuso con la successiva `endwin()`. Faccio notare che l'istruzione initscr() non effettua la cancellazione dello schermo ma predispone tutta una serie di strutture dati per interfacciare ncirses con l'hardware video del sistema su cu è in esecuzione.
Di seguito incontriamo l'istruzione:

```
printw("Salve mondo da ncurses!");
```

che, senza sorprese predispone la stampa di una stringa di saluto. In realtà, la scrittura avviene preventivamente in uno specifico buffer che verrà poi dirottato in output grazie alla funzione `refresh()` che aggiornerà il nostro display video. Ci sono poi un altro paio di cose che vale la pena di sottolineare. La prima riguarda l'assoluta necessità di usare l'istruzione `getch()` per consentire la visualizzazione dell'output a video in quanto, in caso contrario, scomparirebbe immediatamente alla vista riportando la sola visualizzazione del prompt dei comandi. La seconda cosa riguarda il fatto che la funzione getch() effettua una sorta di aggiornamento implicito e automatico dello schermo e che quindi, teoricamente, in questo caso specifico di esempio minimale anche senza la chiamata della funzione `refresh`, l'output a video verrebbe comunque mostrato.
Per la compilazione useremo il seguente comando:

```
gcc  mycurse.c -lncurses
```

Attenzione alla necessità di compilare usando l'opzione `-lncurses` in quanto ncurses non è semplicemente un file di intestazione ma una vera e propria libreria e quindi `-l` ne impone il linking. Ovviamente, per lancoare il nostro eseguibile digiteremo il classico `./a.out`.
Al solo scopo di far comprendere la semplicità d'uso della libreria vi mostro ora una funzione che consente di spostare il punto di editing, il nostro cursore, in una specifica posizione:

```
move(y,x);
```

dove `y` è il valore per la riga e quindi, partendo dall'angolo in alto a sinistra, che ha coordinata (0, 0), aumenta verso il basso mentre `x` è il valore per la colonna che quindi, banalmente, aumenta da sinistra verso destra.
Un'altra funzione semplice e interessante è:

```
mvprintw(y,x,formato,argomenti[...])
```

che consente di stampare una data stringa in una qualsiasi posizione del nostro terminale.
Questo brevissima panoramica sulla libreria ncurses voleva solo sollecitare l'interesse per un sistema software che, se be padroneggiato può dare grandi soddisfazioni come dimostra plasticamente la figura relativa al videogioco Rouge sviluppato appunto sfruttando tale libreria e che ha segnato un'epoca guadagnandosi un posto nella storia dei videogame.

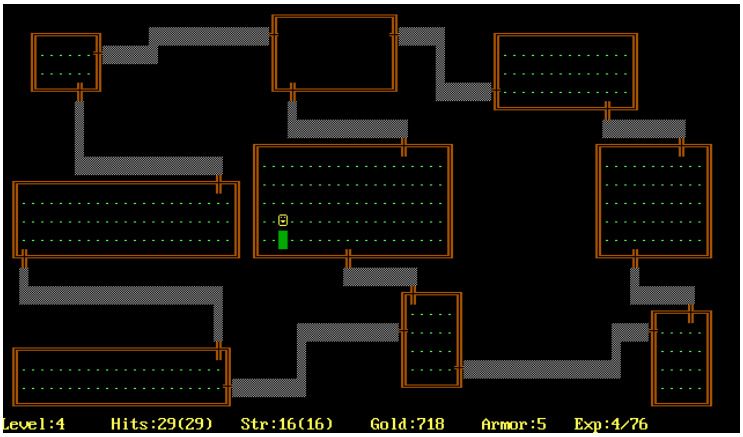

Figura 1 Il videogioco Rouge realizzato con le librerie ncurses.

Cmd.exe e la console testuale come non l'hai mai vista prima

La libreria ncurses è sicuramente uno strumento formidabile relativo al mondo Unix-like. Tuttavia, può essere interessante capire come gestire la console a caratteri in maniera evoluta anche sul sistema di zio Bill. Per farlo, ovviamente, possiamo sfrutta il WSL, ovvero il Sottosistema Windows per Linux all'interno di Windows 10, così come usarlo in ambiente di emulazione Cygwin. Volendo potremmo addirittura sfruttare sotto Windows le librerie ncurses con il linguaggio di programmazione Python. Ma a noi piacciono le cose complesse e ardite per cui di seguito vi mostro come realizzare "a mano" qualcosa di simile alle ncurses direttamente sotto Windows. Pronti? Partiamo!

Sappiamo bene che la console **cmd.exe** è in effetti estremamente rigida ma cercheremo ora di forzarla a essere più duttile rispetto alle nostre necessità e di renderla, in un certo qual modo, "grafica". Ovviamente, la prima cosa che dobbiamo cercare di fare è quella di poter scrivere un certo simbolo in una posizione qualsiasi della console e non quindi semplicemente sul nostro prompt dei comandi. Questa operazione, non propriamente banale, è possibile grazie alla libreria `windows.h` ed alla funzione `SetConsoleCursorPosition` che consente appunto di settare (collocare) il cursore in una data posizione della console. Per rendere il tutto più immediatamente comprensibile vi mostro subito un pezzo di codice minimale che realizza quanto detto.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

void gotoxy(short x, short y);

int main()
{
    gotoxy(10,1);
    printf("X");

    getchar();

    return 0;
}

void gotoxy(short x, short y)
{
    COORD pos ={x,y};
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}
```

Analizziamo quindi il codice precedente, osservando innanzitutto, come già anticipato, la necessità di includere la libreria `windows.h`. Subito dopo troviamo l'intestazione di una funzione `gotoxy` che abbiamo scritto con lo specifico scopo di spostare il nostro cursore in una determinata posizione che passeremo come argomento alla funzione stessa. La funzione in questione richiama quindi al suo interno `SetConsoleCursorPosition`. Quest'ultima prende in input il riferimento alla finestra in uso e le coordinate per la nuova posizione del cursore. Per ulteriori dettagli su tale funzione è possibile rifarsi al link:

`https://docs.microsoft.com/it-it/windows/console/setconsolecursorposition`

Qui si può facilmente scoprire che le coordinate da passare alla funzione sono la colonna e la riga di una cella del buffer dello schermo. Ovviamente, le coordinate in questione devono trovarsi all'interno dei limiti del buffer dello schermo della console. Nel caso in cui la funzione ha esito negativo verrà restituito il valore zero. Al momento, per una questione di semplicità non effettuiamo uno specifico controllo di errore che comunque dovrebbe essere sempre fatto con l'ausilio della funzione `GetLastError`.

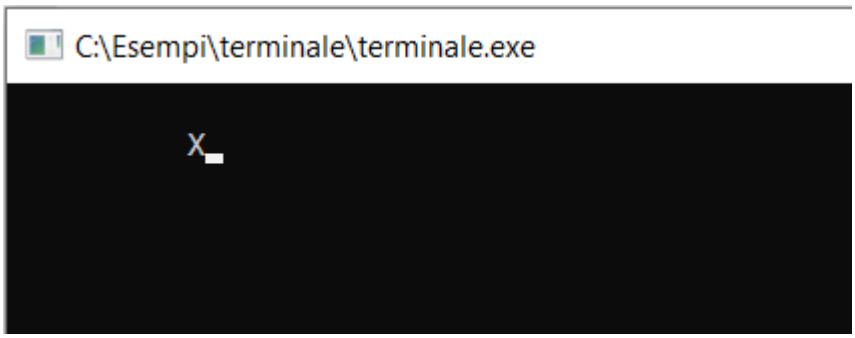

Figura 2 La console dopo lo spostamento del cursore

Come si intuisce, la coordinata 0, 0 è nell'angolo in alto a sinistra e quindi è come se ci trovassimo in un piano cartesiano con x che cresce verso sinistra e y che cresce verso il basso.
Per rendere immediatamente interessante quello che stiamo facendo, possiamo immaginare di far muovere il nostro cursore sulla console usando i mitici e classici tasti freccia. Per farlo dobbiamo innanzitutto poter leggere i tasti della tastiera e quindi solo dopo spostare il cursore in maniera consistente. Per leggere la tastiera possiamo sfruttare le funzioni `kbhit` e `getch`.
È importante precisare che entrambe le funzioni in questione non sono il massimo della standardizzazione e che quindi devono essere usate consapevoli del fatto che sarà necessario verificare gli eventuali vincoli delle specifiche piattaforme. In ogni caso, `kbhit` legge la tastiera e restituisce un valore diverso da zero in caso di pressione di un tasto mentre `getch` restituisce il valore corrispondente al tasto digitato. Vediamo allora il codice che ci permette di leggere la digitazione dei vari tasti e interrompere l'esecuzione in caso di digitazione del tasto `ESC`, escape, che corrisponde al valore 27.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <conio.h>
int main()
{
    printf("Premere ESC per interrompere.\n");
    int ch;
    while (TRUE)
    {
          if ( kbhit() )
          {
            //
            ch = getch();
            if ((ch == 27))
```

```
            {
             break;
             }

            printf("Tasto: %c numero: %d\n", ch, ch);
        }
    }

    printf("Programma terminato!");
    getchar();

    return 0;
}
```

Il programmino in questione presenta, nel più classico dei modi, il **main loop**, ovvero il ciclo principale che si imposta su pressoché qualsiasi software e che consente di iterare le varie operazioni fin quando non se ne richiede in maniera forzata l'uscita. Il codice stesso dovrebbe essere di immediata comprensione: il ciclo itera teoricamente all'infinito (`while (TRUE)`) e terminerà con un `break` non momento in cui il tasto digitato è appunto `ESC`.

Da notare che ho usato la funzione `getch` e non `getchar` in modo da non dover attendere la pressione del tasto `Invio` per confermare la digitazione del singolo tasto.

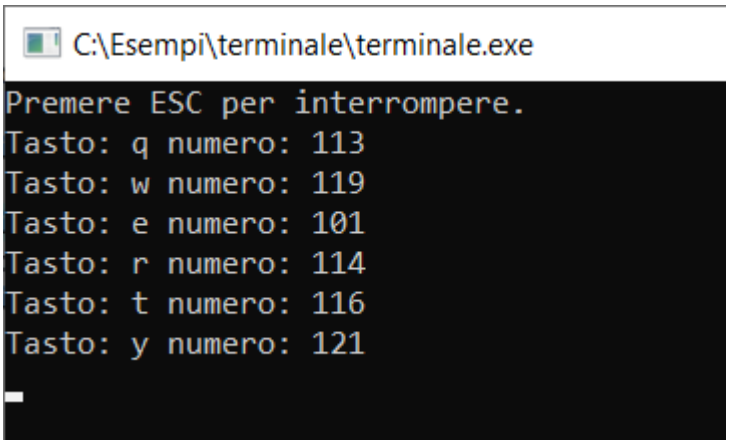

Figura 3- Il programma per la cattura dei valori dei tasti.

Muoviamo il cursore nella console

A questo punto, maturate le competenze per spostare in una certa posizione il nostro cursore e capito come possiamo intercettare i tasti, possiamo immaginare di muovere il nostro cursore nel punto in cui vogliamo tramite specifici tasti. Per semplificare il codice evitiamo di usare i tasti freccia. Infatti, la pressione dei tasti freccia non è particolarmente standard e tra l'altro restituisce 2 differenti valori piuttosto che uno solo, così come per gli altri tasti normali. In ogni caso, non è un gran problema in quanto possiamo immaginare di usare, al posto dei tasti freccia, una combinazione classica nota come WASD. Si tratta dei tasti che hanno appunto queste quattro lettere e che vengono usate nei giochi per sostituire le frecce stesse in quanto questi hanno una organizzazione a T rovesciata del tutto simile.

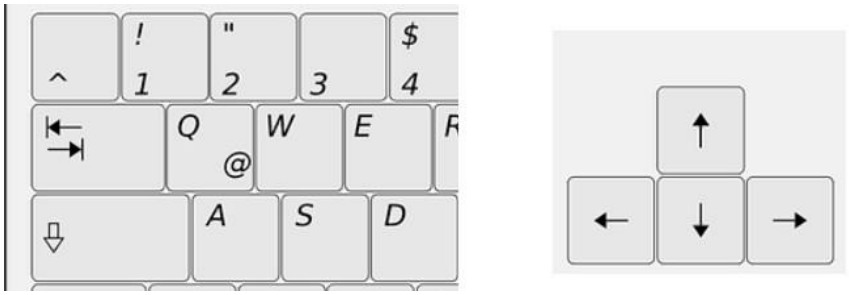

Figura 4 Il confronto tra i tasti freccia e la combinazione WASD

Come si intuisce anche dalla figura, i tasti A e D corrispondono ad avanti e indietro mentre W e S ai tasti sopra e sotto. Vediamo allora come leggere questi tasti e spostare il cursore in logica conseguenza.
Servendoci del precedente codice scopriamo anche immediatamente quali sono i valori corrispondenti ai tasti di nostro interesse e che vi riporto in figura.

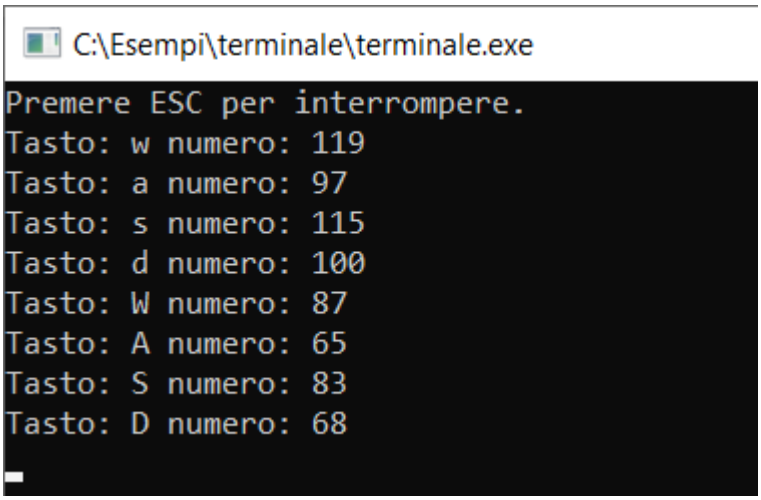

Figura 5 I codici dei tasti della combinazione WASD.

Per evitare problemi dovremo ovviamente intercettare sia il codice per le lettere minuscole sia quelle per le maiuscole. Di seguito vi riporto una possibile e semplice implementazione di quanto ci siamo proposti di fare.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <conio.h>

#define UP 1
#define DOWN 2
#define LEFT 3
#define RIGHT 4

#define MAX_X 80
#define MAX_Y 25


void gotoxy(short x, short y);
short pos_x, pos_y;

int main()
{
    short pos_x=0;
    short pos_y=0;


    gotoxy(0, MAX_Y+1);
    printf("Premere ESC per interrompere.\n");
    gotoxy(0,0);
```

```
    int ch;
    while (TRUE)
    {
          if ( kbhit() )
          {
            //
            ch = getch();
            if ((ch == 27))
            {
             break;
             }
             else
             {
                 switch(read_key(ch))
                 {
                     case UP:
                         pos_y--;
                         break;

                     case DOWN:
                         pos_y++;
                         break;

                     case LEFT:
                         pos_x--;
                         break;

                     case RIGHT:
                         pos_x++;
                         break;
                 }
                 //evito che valori diventino negativi o superiori al max consentito
                 if (pos_x <0) pos_x=0;
                 if (pos_y <0) pos_y=0;
                 if (pos_x > MAX_X) pos_x=MAX_X;
                 if (pos_y > MAX_Y) pos_y=MAX_Y;

                 gotoxy(pos_x, pos_y);
                 printf("X");
             }

        }
    }

    printf("Game over ;) ");
    getchar();

    return 0;
}

//
int read_key(int ch)
{
    int direction=0;
    if (ch==119 || ch == 87)  //tasto W - sopra
    {
        direction=UP;
    }
    else if (ch==97 || ch == 65) //tasto A - sinistra
    {
        direction=LEFT;
    }
    else if (ch==115 || ch == 83) //tasto S - sotto
    {
        direction=DOWN;
    }
    else if (ch==100 || ch == 68) //tasto D - destra
    {
        direction=RIGHT;
    }
```

```
    return direction;
}

void gotoxy(short x, short y)
{
    COORD pos ={x,y};
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}
```

Cercherò ora di commentare gli elementi più salienti di quanto proposto. Innanzitutto, definiamo alcune costanti utile per semplificare la leggibilità del codice stesso, iniziando con quelle utili a definire la direzione per i nostri tasti.

```
#define UP 1
#define DOWN 2
#define LEFT 3
#define RIGHT 4
```

Successivamente definiamo il limiti del nostro campo di gioco con le seguenti:

```
#define MAX_X 80
#define MAX_Y 25
```

Ovvero, in questo caso, 25 righe per 80 colonne. Per gestire la posizione corrente usiamo, invece, le seguenti variabili globali:

```
short pos_x, pos_y;
```

Detto questo, il codice è abbastanza semplice da interpretare considerando quanto già visto in precedenza. Cicliamo, dunque, fino alla pressione del tasto escape e con il seguente switch effettuiamo, modificando le variabili `pos_x` e `pos_y` il movimento corrispondente alla scelta effettuata:

```
        switch(read_key(ch))
                {
                    case UP:
                        pos_y--;
                        break;

                    case DOWN:
                        pos_y++;
                        break;

                    case LEFT:
                        pos_x--;
                        break;

                    case RIGHT:
                        pos_x++;
                        break;
                }
```

Come si può facilmente comprende abbiamo creato una specifica funzione `read_key` per capire in quale direzione andare in quanto la funzione in questione legge i codici dei tasti e restituisce la costante della direzione da prendere.
Infine, vale forse la pena di notare le righe di codice:

```
gotoxy(0, MAX_Y+1);
printf("Premere ESC per interrompere.\n");
gotoxy(0,0);
```

che rappresentano, a livello minimale, un modo per rendere l'interfaccia più pulita spostando le istruzioni per l'uscita dal gioco in una zona esterna al nostro campo di azione.
Ovviamente, il tutto è assolutamente embrionale e vuole solo dare indicazioni di massima su come organizzare un tipico gioco. Possiamo infatti immaginare, a solo titolo di esempio, come potremmo generare in una posizione casuale del campo di gioco un qualsiasi elemento e poi far raggiungere al nostro giocatore quella posizione, ad esempio, calcolando quanto tempo ci mette per raggiungerla e restituire un punteggio in base alla sua velocità di azione. Come al solito, l'unico limite è la fantasia!

Carlo A. Mazzone