Data e Ora di sistema

Ottenere la data e l'ora di sistema in C++ non è un compito così banale come può essere in altri linguaggi, dove è anche previsto il tipo di dato apposito e tutte le operazioni sono immediate. In realtà, anche la procedura che andremo ad utilizzare non sarà particolarmente complicata, solo un po' più complessa da realizzare. Per capire questo tutorial è necessario avere una conoscenza, seppur minima delle strutture e dei puntatori a struttura.

1. Introduzione

Attraverso questo tutorial, vedremo come ottenere la data e l'ora di sistema, attraverso un metodo abbastanza semplice e che ha il vantaggio di essere compatibile con l'ANSI C e pertanto con la stragrande maggioranza dei compilatori.

Quello che vogliamo ottenere è un formato a noi, utile, che pertanto non sia una banale stringa, ma una serie di cifre intere che possano in seguito essere utilizzate anche per altri calcoli.

I tipi di dato che ci interessano sono contenuti in `ctime`, `time.h` per i vecchi compilatori.

In particolare questi sono il tipo `time_t`, atto a contenere il tempo trascorso dal **1 gennaio 1970** (in secondi, generalmente) e `tm`, una struttura formata da campi contenenti la data in un formato leggibile. In C++, a differenza di molti altri linguaggi, ottenere la data e l'orario di sistema non è un'operazione così immediata, visto che richiede alcuni passaggi e alcuni concetti non del tutto elementari.

2. Codice necessario

Nel nostro programma, per ragioni che vedremo dopo, ci serve un puntatore a una struttura `tm`.

```
tm *leggibile = NULL
```

L'inizializzazione a `NULL` dice al compilatore che attualmente il puntatore non punta ad alcun indirizzo di memoria. A questo punto dobbiamo creare una variabile di tipo `time_t`. Se si invia l'indirizzo di questa variabile alla funzione `time()`, in seguito questa conterrà esattamente i secondi trascorsi dal 1 gennaio 1970.

```
time_t tempo;
time (&tempo);
```

E' ora il momento di fare la conoscenza della funzione `localtime()`: questa, ricevendo come input l'indirizzo di un `time_t` (opportunamente inizializzato ad un numero di secondi), restituisce un puntatore a `tm` con le informazioni correttamente

suddivise. Questo è il motivo per cui abbiamo precedentemente dichiarato il `tm` come puntatore.

Dopo questa istruzione

```
leggibile = localtime (&date);
```

l'indirizzo puntato da `leggibile` conterrà esattamente le informazioni che ci servono. Tuttavia, per poterle utilizzare, necessitiamo di sapere come è definito il tipo `tm`. Apriamo quindi `time.h` e cerchiamo nel codice questo frammento:

Definizione della struttura tm

```
struct tm
{
int tm_sec; /* Seconds: 0-59 (K&R says 0-61?) */
int tm_min; /* Minutes: 0-59 */
int tm_hour; /* Hours since midnight: 0-23 */
int tm_mday; /* Day of the month: 1-31 */
int tm_mon; /* Months *since* january: 0-11 */
int tm_year; /* Years since 1900 */
int tm_wday; /* Days since Sunday (0-6) */
int tm_yday; /* Days since Jan. 1: 0-365 */
int tm_isdst; /* +1 Daylight Savings Time, 0 No DST,
* -1 don't know */
};
```

Ora sappiamo esattamente tutti i campi. Va solo ricordata la regola sintattica per cui per accedere ai campi di un puntatore a `struct` si deve utilizzare l'operatore freccia (->). Pertanto per avere la data scriveremo (output non formattato).

```
cout << leggibile->tm_mday << leggibile->tm_mon << leggibile-> tm_year;
```

Se i risultati vi sembrano errati osservate che il numero del mese parte da 0 (gennaio) fino a 11 (dicembre), quindi per scriverlo correttamente bisogna incrementarlo di uno. L'anno inoltre è riferito al 1900, quindi per avere la data corretta e formattata scriveremo:

```
cout << leggibile->tm_mday << " " << (leggibile->tm_mon + 1)
<< " " << (leggibile-> tm_year + 1900);
```

3. Scrittura di una funzione

Si può creare una semplice funzione che riceva come argomenti i reference di 7 interi e gli assegni i valori di giorno, mese, anno, giorno della settimana, ora, minuti, secondi della data attuale.

Funzione `now()`

```
void now (int &gm, int &m, int &a, int &gs, int &h, int &min, int &s)
{
  // Determinazione della data e dell'orario correnti
  time_t data;
  struct tm * leggibile = NULL;
```

```
  time (&data);
  leggibile = localtime (&data);

  // Riempimento delle variabili
  // Giorno del mese
  gm = leggibile->tm_mday;

  // Mese
  m = leggibile->tm_mon +1;

  // Anno
  a = leggibile->tm_year+1900;

  // Giorno della settimana (1 = Domenica, 7 = Sabato)
  gs = leggibile->tm_wday+1;

  // Ora
  h = leggibile->tm_hour;

  // Minuti
  min = leggibile->tm_min;

  // Secondi
  s = leggibile->tm_sec;

  // Fine della funzione
  return;
}
```

4. Orologio dinamico

Scriveremo adesso un programma che generi un orologio dinamico, ossia che si aggiorna ogni secondo, dando appunto l'effetto di essere interattivo.

Per risolvere il nostro problema, si potrebbe subito pensare ad un ciclo infinito che continuamente chiama `now()` e stampa l'orario, utilizzando sempre una chiamata a `clrscr()` per pulire lo schermo.

Bandiamo subito questa idea per almeno 2 motivi:

1. La funzione `clrscr()` non è inclusa nella libreria standard e comunque rallenta l'esecuzione
2. Un ciclo infinito non permette di uscire dal programma in modo semplice.

Per risolvere il primo problema, possiamo agire su due strade diverse:

- Utilizzare la funzione system di `stdlib.h`
- Utilizzare il carattere di carriage return `\r` che automaticamente torna all'inizio della riga corrente eliminando l'eventuale contenuto.

Per comodità e velocità di esecuzione, ci conviene decisamente imboccare la seconda strada. A questo punto è necessario vedere come evitare il ciclo infinito: dentro la libreria `conio.h` è disponibile la

funzione `kbhit()` che, non ricevendo parametri, restituisce un valore `VERO` se è stato premuto un tasto e un valore `FALSO` in caso contrario.

Possiamo usare questa funzione come rilevatore di eventi, da inserire in un ciclo come condizione: una volta rilevata la pressione di un tasto, il programma termina. Ci tengo a precisare che questa funzione **non è standardizzata**, ma al momento in cui scrivo non ho trovato altre soluzioni, se non quella di ricorrere al linguaggio Assembly, che mi sembra comunque da scartare a priori. Comunque sia, in ambiente Windows, sarà sufficiente dichiarare il progetto di tipo *Console Application* per avere a disposizione la libreria necessaria.

Il corpo del programma si presenta quindi come segue:

Orologio dinamico

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>

// Prototipi delle funzioni
void now (int&,int&,int&,int&,int&,int&,int&);

int main ()
{
  int g, m, a, gs, h, min, s;

  do {
    now (g, m, a, gs, h, min, s);
    cout << "\r";
    cout << g << "-" << m << "-" << a << " ";
    cout << h << ":" << min << ":" << s;
  } while (!kbhit());

  return 0;
}

void now (int &gm, int &m, int &a, int &gs, int &h, int &min, int &s)
{
  time_t data;
  struct tm * leggibile = NULL;
  time (&data);
  leggibile = localtime (&data);
  gm = leggibile->tm_mday;
  m = leggibile->tm_mon +1;
  a = leggibile->tm_year+1900;
  gs = leggibile->tm_wday+1; // 1 = Domenica - 7 = Sabato
  h = leggibile->tm_hour;
  min = leggibile->tm_min;
  s = leggibile->tm_sec;
  return;
}
```

Eseguendo il programma, esso appare perfettamente funzionante, ma abbiamo ancora un problema, non del tutto indifferente: l'immagine su schermo sembra "traballare" (a seconda della velocità del vostro processore dovreste vedere più o meno questo sfarfallio).

Questo è dovuto al fatto che il computer impiega pochi millisecondi a eseguire un ciclo completo, quindi in un secondo questo viene completato migliaia di volte: decisamente uno spreco. A noi, in fondo,

basta poter calcolare l'ora ogni secondo (ossia ad ogni variazione sostanziale della stampa).

Ci serve una funzione che dica al programma di fermarsi per un secondo prima di proseguire. In questo caso abbiamo seri problemi di compatibilità:

- Su ambiente Windows è possibile utilizzare `Sleep()` (con l'iniziale maiuscola), contenuta in `windows.h`
- Su Turbo C è possibile utilizzare `delay()`, contenuta in `dos.h`

A seconda del vostro compilatore, scegliete quale usare: entrambe ricevono come argomento un intero che indica il numero di millisecondi per il quale il processore deve stare in attesa. Nel nostro caso ci basterà inserire 1000 come parametro dopo la stampa e il gioco sarà fatto.

Ecco il codice completo del programma:

Orologio dinamico – versione corretta

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

// Prototipi delle funzioni
void now (int&,int&,int&,int&,int&,int&,int&);

int main ()
{

  // Dichiarazione delle variabili
  int g, m, a, gs, h, min, s;

  do {

    // Chiamata alla funzione
    now (g, m, a, gs, h, min, s);

    // Pulitura della riga
    cout << "\r";

    // Stampa
    cout << g << "-" << m << "-" << a << " ";
    cout << h << ":" << min << ":" << s;

    // Attesa di 1 secondo
    Sleep (1000);

  } while (!kbhit());

  return 0;
}

void now (int &gm, int &m, int &a, int &gs, int &h, int &min, int &s)
{
  time_t data;
  struct tm * leggibile = NULL;
```

```
  time (&data);
  leggibile = localtime (&data);
  gm = leggibile->tm_mday;
  m = leggibile->tm_mon +1;
  a = leggibile->tm_year+1900;
  gs = leggibile->tm_wday+1; // 1 = Domenica - 7 = Sabato
  h = leggibile->tm_hour;
  min = leggibile->tm_min;
  s = leggibile->tm_sec;
  return;
}
```