Documentazione file action - v0.1

Luigi DE SARLO

February 10, 2010

Questa è la documentazione del tipo di dati (si direbbe classe in un linguaggio orientato agli oggetti) `action` che è la base del funzionamento interno del software BEC3.

Cominciamo dal file header (`action.h`) in cui sono dichiarate tutte le funzionalità del tipo Saltando le prime righe la prima cosa che troviamo sono una serie di macro:

```
#define PARAM_N 4
#define PROTOTYPE_N 200
#define DBFILE "d:\\BEC3\\BEC3.dbase"
#define ERRSTREAM "d:\\BEC3\\errfile.dat"
#define NAMEMAXLEN 20 //warning: this value is coded also
    elsewhere!
#define LINEMAXLEN 200
```

Queste macro permettono di parametrizzare alcune caratteristiche del tipo `action`. In particolare `PARAM_N` indica il numero di parametri che un'azione può accettare. Altra cosa da notare è il commento accanto a `NAMEMAXLEN` che dice che il numero in questione (la lunghezza massima del nome di un'azione) è codificato anche da qualche altra parte. Questo è un errore da parte mia ed errori di questo tipo possono generare bugs, bisogna quindi stare all'erta...

```
typedef union _ActParam {
        int iParam;
    float fParam;
    char sParam[2];
} ActParam;
```

Qui entriamo nel vivo della vicenda: si definisce un nuovo tipo di variabile, `ActParam` per descrivere il parametro di un'azione. Si tratta di una unione: un medesimo pezzo di memoria che può rappresentare un intero, un numero in virgola mobile o una coppia di caratteri (l'idea è che così facendo la taglia in memoria di una azione è sempre la stessa). Da questa definizione si ottiene che un parametro di un'azione è

1. un intero
2. un numero reale (singola precisione, 23bit)
3. una coppia di caratteri

nessuna altro tipo (che so, un boolean) è possibile.

```
typedef struct _Action {
        char * Name;
        char addr[10]; //l'indirizzo
        double time; //il tempo
        ActParam Param[PARAM_N];
        char ParamType[PARAM_N];
        char * ParamName[PARAM_N];
        struct _Action * Hp;
        struct _Action * Lp;
} Action;
```

Questo blocco di codice definisce il tipo `Action` propriamente detto: questo consiste in un array di caratteri (`Name`) che contiene il nome dell'azione, un'array di caratteri che rappresenta l'indirizzo (`addr`), un numero in virgola mobile a doppia precisione (`time`) che rappresenta il tempo associato all'azione stessa, un array di lunghezza `PARAM_N` di variabili di tipo `ActParam` (`Param`) che rappresenta i parametri dell'azione stessa, un array di caratteri in cui è codificato il tipo di parametro (`ParamType`), un array di puntatori che contiene i nomi dei differenti parametri (`ParamName`) e due puntatori ad altre due azioni (`Hp` e `Lp`).

Qualche spiegazione:

- Supponiamo di avere un'azione che contiene due parametri: un valore iniziale e un valore finale di una rampa analogica. Questi si troveranno rispettivamente in `Param[0]` e `Param[1]`. Siccome però `ActParam` è una unione, bisogna specificare come si deve interpretare i bit contenuti in queste locazioni di memoria. Nel nostro caso, visto che si tratta di due numeri reali occorre chiamare `Param[i].fParam`, dove evidentemente $i = 0, 1$. Per tener traccia di questo occorre scrivere un carattere in `ParamType[i]`. Questo carattere è stabilito per convenzione nel blocco di commento che segue nel codice: 's' per un parametro a due caratteri, 'i' per un intero, 'f' per un *float*.

- I due puntatori `Hp` e `Lp` permettono di mantenere un albero di azioni in forma di *linked list* che è il tipo di lista in cui è più facile aggiungere (o togliere) elementi in posizione random.

Saltando il blocco di commenti la definizione successiva nel codice è la seguente:

```
typedef struct _parsed {
        Action * actp;
        double elapsed;
        int n;
} parsed;
```

qui si definisce un tipo `parsed` che serve per tradurre l'albero degli oggetti in un buffer da inviare alla scheda DIO64. In pratica `parsed` incapsula un puntatore all'azione che si deve tradurre (`actp`), il tempo trascorso (`elapsed`) e il numero di azioni tradotte fino a `*actp` esclusa (`n`).

Le definizioni seguenti sono quelle delle funzioni più importanti che si possono utilizzare con il tipo `action`:

```
int PrDBLoad(Action *); //carica il dbase dal file    OK
```

```
Action * copyAction(Action * s, Action * d);  //copia un'
    azione OK

int regAction(Action *, Action *); //registra un'az nel
    dbase OK

Action * ActFind(Action *, char *); //cerca per nome la
    pos nel

int PrDBFree(Action *); //libera le risore??

long getrecord(Action *, FILE *);

long setrecord(Action *, FILE *);

int WalkTree(Action *, Action *, int dir);

int isSpecial(Action *);
```