Il programmatore che c'e' in noi – Lezione 11– Enumerazioni e Tipi dati utente

Una enumerazione e' un elenco di costanti intere a cui, per ciascun valore numerico, e' stato assegnato un nome .

Nella realizzazione di programmi, ci troviamo spesso, per comodita', a dover utilizzare dei valori numerici interi, consecutivi (0,1,2,3...) per rappresentare determinati valori che numerici non sono.

Ad esempio un elenco di colori, un elenco di categorie di libri o l'elenco delle monete relative all'Euro (5,10,20,50 centesimi, 1,2 euro).

Per motivi di LEGGIBILITA' conviene utilizzare una COSTANTE numerica (che ha quindi un NOME) piuttosto che utilizzare il solo valore numerico che rappresenta ciascun elemento dell'elenco.

Nota:
Con il termine Leggibilita' si intende sostanzialmente questo:
- chi legge il sorgente di un programma non e' detto che sia la stessa persona che lo ha scritto
- a distanza di anni potreste riprendere in mano un vecchio programma per modificarlo

Conosciamo gia' un modo per poter fare questo (assegnare un NOME ad un valore numerico), utilizzando la parola chiave #define NOMECOSTANTE valore

Esempio:
#define ROSSO 2
#define VERDE 3

ma esiste un sistema ancora piu' efficace: l'uso della parola chiave enum.

Una enumerazione si definisce nel seguente modo:

```
//Definizione della enum
enum tag{
  nomeelemento,
  nomeelemento,
  nomelemento,
…
};

//Dichiarazione di una variabile di tipo enum
enum tag nomevariabile;
```

oppure, tutto insieme…

```
enum tag{
 nomeelemento,
 nomeelemento,
 nomeelemento…
}listavariabili;
```

Esempio:

Una enumerazione di categorie di libri

```
enum genere{
 fantascienza,
 giallo,
 avventura,
 fantasia,
 informatica
} genereDiLibro;
```

Tramite questa dichiarazione, ogni elemento della enum ora e' rappresentato da un valore numerico intero, a partire da zero e che si incrementa di 1 per ogni elemento successivo della lista.

Quindi fantascienza vale 0, giallo=1, avventura=2 ecc.

Controlliamo se e' vero …

Da XCODE creaimo un nuovo progetto, cocoa application, quindi nel file main.m andiamo a dichiarare la nostra enum e con NSLog vediamo i valori numerici corrispondenti dei vari elementi.

```
int main(int argc, char *argv[])
{
//Dichiarazione dell'enum
enum genere{
 fantascienza, //=0
 giallo,       //=1
 avventura,    //=2
 fantasia,     //=3
 informatica   //=4
}genereDiLibro; //Dichiarazione della variabile

NSLog(@"Valore dell'elemento informatica = %d",informatica);

return NSApplicationMain(argc,  (const char **) argv);
}
```

Eseguiamo ed ecco il risultato …
2006-07-02 10:28:00.048 Enumerazioni[943] Valore dell'elemento informatica = 4

Quanto vale la nostra variabile `genereDiLibro ?`

```
NSLog(@"Valore della variabile genereDiLibro: %d ",genereDiLibro); //Bho ? Non
e' stata assegnata! Quindi dipendera' da quello che trova in memoria
```

Infatti ecco il mio log
2006-07-02 10:30:35.339 Enumerazioni[947] Valore della variabile genereDiLibro: 7468

Bene, inizializziamola

```
genereDiLibro=informatica; //Inizializzo la variabile con un elemento della enum

NSLog(@"Valore della variabile genereDiLibro: %d ",genereDiLibro); //Ora vale
informatica cioe' 4.
```

```
Verifichiamo…
[Session started at 2006-07-02 10:33:01 +0200.]
2006-07-02 10:33:01.488 Enumerazioni[966] Valore dell'elemento informatica = 4
```

```
2006-07-02 10:33:01.488 Enumerazioni[966] Valore della variabile genereDiLibro:
7468
2006-07-02 10:33:01.488 Enumerazioni[966] Valore della variabile genereDiLibro:
4
```

In sintesi, ora e' possibile utilizzare il nome degli elementi della enum, in qualsiasi espressione invece di usare il valore numerico, migliorandone la leggibilità e riducendo la possibilità di errori nel programma.

Esempio:
Se devo controllare se un dato libro e' di informatica scrivero' qualcosa del genere …

```
if( genereDiLibro == informatica)
 NSLog(@"Genere di libro: informatica");

che e' molto piu' chiaro di

if( genereDiLibro == 4)
 NSLog(@"Genere di libro: informatica");
```

Per i piu' curiosi
E' possibile far partire l'indice dei valori da assegnare agli elementi della enum con un valore numerico intero qualsiasi (se non viene specificato nulla, parte da 0).

Esempio

```
enum colors{
 rosso=100,
 verde,
 gialloC, //Ho gia' giallo definito nella enum genere devo dare un altro nome !
 marrone=0,
 nero,
 bianco
};

NSLog(@"Rosso: %d Verde: %d Giallo: %d Marrone: %d Nero: %d Bianco: %d",
rosso,verde,gialloC,marrone,nero,bianco);
```

E' questo e' il risultato del log

2006-07-02 10:40:51.803 Enumerazioni[994] Rosso: 100 Verde: 101 Giallo: 102 Marrone: 0 Nero: 1 Bianco: 2

TYPEDEF – Tipi dati utente

Nella vita reale, spesso amiamo dare nomi diversi alle persone o alle cose, per renderle piu' facili da ricordare o per "evidenziarne" delle caratteristiche.

Bene, in informatica possiamo fare anche questo, con i nostri tipi di dato.

Se, ad esempio, invece di usare il nome double per identificare una variabile di tipo numerica reale che poi uso per dichiarare delle variabili relative al mio stipendio, voglio creare un tipo di variabile che si chiami stipendio devo usare la parola chiave typedef per creare un mio tipo di dati personalizzato.

La sintassi e' la seguente
typedef nometipodatobase nuovonome;
quindi, per l'esempio citato sopra:

typedef double stipendio; //Creo un nuovo tipo dato che si chiama stipendio

Per utilizzarlo

```
//Dati definiti da utente
typedef double stipendio;

stipendio meseGennaio=120.0;
stipendio meseFebbraio=180.0;
stipendio meseMarzo=220.0;

stipendio medio=0.0;

medio =( meseGennaio + meseFebbraio + meseMarzo) /3.0;

NSLog(@"Stipendio Medio: %.2f",medio);
```

L'utilizzo di typedef ha un duplice scopo:
1 – migliora la leggibilita' e la comprensione del codice
2 - facilita la migrazione del programma tra i sistemi (es. Mac os x, Unix, Linux, Windows)
questa caratteristica dei programmi si chiama PORTABILITA'.

Se il programma deve funzionare invece che su un Mac su un sistema che non conosce il tipo dati double, ma usa solo float, e' sufficiente cambiare la sola istruzione `typedef double stipendio;` con `typedef float stipendio;` e tutto funzionerà regolarmente.

Se, invece abbiamo scritto

```
double meseGennaio=120.0;
double meseFebbraio=180.0;
double meseMarzo=220.0;
```

dobbiamo cambiare tutte le istruzioni dove compare il tipo dato `double` con il tipo dato `float`, **operazione piu' lunga e piu' soggetta ad errori**.

Normalmente non si procede alla ridefinizione dei tipi di dati base a meno che non si abbia gia' in mente di creare un software portabile su piu' piattaforme.

Un uso molto comune del typedef e' invece quello di assegnare un proprio tipo di dato ad un struttura dati.

In una delle precedenti puntate abbiamo esaminato le struct, che sostanzialmente ci consentono di "raggruppare" delle variabili logicamente legate tra di loro.

Riprendo l'esempio della carta di identita'…

```
struct cartaIdentita{
  char nome[101]; //stringa per contenere al massimo 100 caratteri (+1 = NUL)
  char cognome[101];
  char comuneResidenza[51];
  int nAnnoNascita; //Anno aaaa
  int nMeseNascita; // Mese mm
```

```
  int nGiornoNascita; //Giorno gg
  double nAltezza; //Altezza in metri,centimetri
};
```

Questa e' la definizione della nostra struttura dati da gestire, come abbiamo visto la volta scorsa, ogni volta che mi serve una variabile per contenere tale struttura dati, devo fare una dichiarazione di questo tipo

```
struct cartaIdentita CartaIdentitaFranco; //Creo la variabile
```

A questo punto posso lavorare sulla variabile `CartaIdentitaFranco`.

L'uso dell'istruzione typedef, ci consente di creare un tipo dati

```
typedef struct cartaIdentita cIdentita;
```

quindi ora per creare la variabile di tipo cartaIdentita, e' necessario indicare solo

```
cIdentita CartaIdentitaFranco;
```

Riepilogo la differenza:

1) senza usare typedef

```
struct cartaIdentita CartaIdentitaFranco; //Creo la variabile
struct cartaIdentita CartaIdentitaRossi; //Creo la variabile
struct cartaIdentita CartaIdentitaMarco; //Creo la variabile

   //Assegno i valori …
   strcpy(CartaIdentitaFranco.nome,"Francesco");
   …
```

2) usando typedef

```
cIdentita CartaIdentitaFranco;
cIdentita CartaIdentitaRossi;
cIdentita CartaIdentitaMarco;

   //Assegno i valori …
   strcpy(CartaIdentitaFranco.nome,"Francesco");
   …
```

Il vantaggio che abbiamo e' quello di scrivere meno (non devo ripetere struct ogni volta) e di rendere un codice piu' leggibile.

Generalmemente nel momento stesso un cui si definisce la struttura, si assegna il nome con typedef, cosi' risparmio ulteriori istuzioni.

```
typedef struct cartaIdentita{
  char nome[101]; //stringa per contenere al massimo 100 caratteri (+1 = NUL)
  char cognome[101];
  char comuneResidenza[51];
  int nAnnoNascita; //Anno aaaa
  int nMeseNascita; // Mese mm
  int nGiornoNascita; //Giorno gg
  double nAltezza; //Altezza in metri,centimetri
} cIdentita;

cIdentita fg;
```

```
strcpy(fg.nome,"Francesco");
NSLog(@"nome: %s",fg.nome);
```

Concludo segnalando la possilita' di utilizzare typedef anche per le enum e le union.

Questo e' un esempio preso pari-pari dalla classe NSButtonCell, che definisce i possibili tipi di pulsanti utilizzabili tramite Cocoa.

```
typedef enum _NSButtonType {
    NSMomentaryLightButton      = 0,    // was NSMomentaryPushButton
    NSPushOnPushOffButton       = 1,
    NSToggleButton          = 2,
    NSSwitchButton          = 3,
    NSRadioButton           = 4,
    NSMomentaryChangeButton     = 5,
    NSOnOffButton           = 6,
    NSMomentaryPushInButton     = 7,    // was NSMomentaryLight

    /* These constants were accidentally reversed so that NSMomentaryPushButton
lit and
       NSMomentaryLight pushed. These names are now deprecated */

    NSMomentaryPushButton       = 0,
    NSMomentaryLight            = 7

} NSButtonType;
```

Questi significa che ogni volta che nel programma voglio impostare un tipo di pulsante potro' utilizzare il tipo dato NSButtonType.

Esempio:

```
 NSButtonType tipoDiMioPulsante; //Dichiaro la variabile tipoDiMioPulsante
                                 //Di tipo enum _NSButtonType


 tipoDiMioPulsante = NSMomentaryLight; //Assegno il valore alla mia variabile
                                       //Uso uno dei valori previsti dalla Apple

 NSLog(@"tipoDiMioPulsante: %d",tipoDiMioPulsante); //vale 7.
```

Con questa puntata si conclude il ciclo relativo ai dati ed alle variabili, dalla prossima iniziamo a parlare delle istruzioni del linguaggio. (STATEMENTS).

Saluti.