Il programmatore che c'e' in noi – Lezione 16 – Ingrassare e dimagrire a piacimento

Sara' che ho bevuto tre bicchieri di Barbera d'Asti Cascina Ferro (scusate la soletta pubblicitaria), sara' che arrivo da una crisi esistenziale (ahh… l'ammmore) insomma il questo e' il titolo che mi e' venuto in mente per questa puntata sull'allocazione dinamica.
Torno a scrivere dopo una lunga assenza, dovuta alla crisi di cui sopra, non che la crisi sia superata ma, si deve andare avanti; ed eccomi qui.

Allocazione dinamica, dicevamo appunto. Immaginare di prendere una casa, inizialmente per voi, probabilmente vi basta un monolocale, ci mettete dentro un letto, una cucina un frigo e la tv, ed ovviamente (se mi state leggendo) un Mac.

Poi, ad un certo punto, vi dice bene e conoscete una donna (o un uomo), decidete di vivere insieme e il monolocale non vi basta piu'; bene, allochiamo un'altra stanza (o meglio affittiamo un'altra stanza) ed il nostro nuovo nido e' in grado di contentere piu' "roba"; facciamo un salotto ed una libreria.

Poi, magari dalla vostra relazione nasce un figlio/a e quindi occorre allocare nuovamente (affittare) un'altra stanza, per poi arredarla con letto e scrivania.

Bene, poi il figlio cresce e se ne va; allora possiamo deallocare la sua stanza; vendere o regalare il suo lettino e la sua scrivania e dismettere l'affitto della stanza.

Poi, si sa', l'ammmore finisce, e ci si separa; allora nuova deallocazione, si libera il salotto; non serve e si dismette anche l'affitto di tale stanza.

In pratica, la nostra dimora puo' essere vista come una allocazione dinamica delle stanze a seconda delle necessita', noi sappiamo di aver bisogno di stanze ma non sappiamo ancora di quante e magari una volta che le abbiamo utilizzare non ne abbiamo piu' bisogno e quidi le "deallochiamo".

Per la memoria del nostro computer vale la stessa regola; ci sono casi in cui sappiamo di aver bisogno di riservare della memoria per contenere i nostri dati, ma semplicemente non sappiamo quanta.

Inoltre, essendo la memoria RAM, una risorsa comunque scarsa e prezionsa, se non abbiamo piu' bisogno di riservare della memoria meglio liberarla, deallocarla appunto.

In linguaggio C si usano delle funzioni standard per allocare, riallocare e liberare la memoria del proprio computer esse sono **calloc**() **malloc**() per l'allocazione, **realloc**() per la riallocazione (ossia cambiare la porzione di memoria in precedenza allocata, come se potessimo cambiare la dimensione della stanza che abbiamo affittato in precedenza) la **free**() per liberare (deallocare) la memoria.

In Cocoa ogni oggetto deve essere allocato dinamicamente in memoria, per fare cio' esiste un metodo che si chiama (guarda caso) **alloc**.

Ogni oggetto allocato, deve essere deallocato, liberato o per dirlo in Cocoa, rilasciato; per fare cio' si usa il metodo **release**.

Qualsiasi elemento in Cocoa (oggetto) deve essere allocato e poi rilasciato.

Facciamo un esempio con l'oggetto NSString, la classe Cocoa per manipolare le stringhe di caratteri.

Con IB creiamo una semplice interfaccia che contiene due campi di input di testo (NSTextField) ed un pulsante (NSButton)

Creiamo la classe che controlla i due campi di input ed il pulsante

```
#import <Cocoa/Cocoa.h>

@interface theController : NSObject {
    IBOutlet NSTextField *txtInput;
    IBOutlet NSTextField *txtOutput;

    NSString *strInput;
    NSString *strOutput;

}
- (IBAction)btnElabora:(id)sender;
@end
```

E definiamo due variabili (oggetti) di tipo NSString strInput e strOutput che come visibile nella definizione sono solo dei puntatori (*)

Nella definizione della classe, andiamo ad intercettare due metodi init e dealloc della classe appena creata

Questi due metodi sono chiamati in automatico dal Framework di cocoa durante la creazione dell'oggetto (init) e prima della sua distruzione (dealloc).

```
#import "theController.h"

@implementation theController

- (id) init
{
    self = [super init];
    if (self != nil) {
      strInput = [NSString alloc];
      strOutput=[NSString alloc];
    }
    return self;
}

- (void) dealloc
{

    [strInput release];
    [strOutput release];
```

```
    [super dealloc];
}

- (IBAction)btnElabora:(id)sender {

}
@end
```

In pratica durante la fase di creazione dell'oggetto, sto allocando anche lo spazio di memoria per gli oggetti di tipo NSString che abbiamo in precedenza definito.

Viceversa, in fase di distruzione dell'oggetto, devo rilasciare (deallocare) la memoria degli oggetti NSString da me in precedenza allocati.

Una volta che l'oggetto NSString e' allocato e' possibile invocare uno dei suo metodi di inizializzazione, dal semplice init oppure ad initWithFormat ecc.

Per assegnare un valore iniziale al nostro campo di input presente nella finestra di dialogo creata in precedenza in IB, ci conviene intercettare un ulteriore metodo dell'oggetto che abbiamo creato (theController) ossia awakeFromNib.

Tale metodo e' invocato in modo automatico dal Framework di Cocoa, prima della visualizzazione del NIB (ossia dell'interfaccia grafica) della nostra applicazione.

Nel esempio di seguito indicato la stringa strInput e' inizializzata con il testo "prova" e tale stringa e' passata come argomento per modificare il contenuto dell'oggetto NSTextField

```
-(void)awakeFromNib{
    strInput=[strInput initWithString:@"prova"];
    [txtInput setStringValue:strInput];
}
```

Dato che spesso si alloca e si inizializza nello stesso momento l'oggetto, conviene normalmente procedere in questo modo

```
      strInput = [[NSString alloc] initWithString:@"prova"] ;
```

Quindi riepilogando, nel metodo init della nostra classe theController

```
- (id) init
{
    self = [super init];
    if (self != nil) {
      strInput = [[NSString alloc] initWithString:@"prova"] ;
      strOutput= [[NSString alloc] initWithString:@""];

    }
    return self;
}
```

mentre nella

```
-(void)awakeFromNib{
    [txtInput setStringValue:strInput];
}
```

Una volta che la nostra variabile e' allocata essa possiede un indirizzo in memoria, che possiamo conoscere usando la funzione NSLog

```
      NSLog(@"\nstrInput  Indirizzo: %p",strInput);
      NSLog(@"\nstrOutput Indirizzo: %p",strOutput);
```

nel mio Mac ad esempio, ottengo

```
2008-12-28 23:01:59.304 AllocaAlloca[1214:10b]
strInput  Indirizzo: 0x2030
2008-12-28 23:01:59.305 AllocaAlloca[1214:10b]
strOutput Indirizzo: 0x2040
```

Gli oggetti NSString sono in Cocoa, degli oggetti immutabili, ossia una volta assegnato loro un valore il valore non puo' essere cambiato occorre distruggere il vecchio oggetto, crearne uno nuovo ed assegnare ad esso il nuovo valore.

Quindi se per esempio voglio modificare il valore della stringa strOutput in modo che mi fornisca il valore "rovesciato" della stringa in input, devo creare un metodo che mi restituisca il nuovo oggetto NSString da assegnare al puntatore *strOutput della mia classe.

Dato che l'oggetto NSString e' immutabile, utilizzo una sua classe derivata (presente in Cocoa) che e' la NSMutableString, che come dice il nome e' invece mutabile, di conseguenza, alloco la memoria dell'oggetto, poi scrivo la stringa di input al contrario, un carattere alla volta.
Quando ho finito restituisco il puntatore della mutable string appena creata.

Il lavoro e' svolto da questo metodo

```
-(NSString*) reverse:(NSString *) strInInput{

    NSMutableString *reversedStr;
    int len = [strInInput length];

    reversedStr = [[NSMutableString alloc] initWithString:@""];

    while (len > 0)
     [reversedStr appendString:[NSString stringWithFormat:@"%C", [strInInput
characterAtIndex:--len]]];

    return reversedStr;
}
```

Dato che la NSMutableString e' una classe derivata da NSString posso tranquillamente assegnare il puntatore alla variabile *strOutput.

Infatti nell'implementazione del metodo

```
- (IBAction)btnElabora:(id)sender
```

effettuo le seguenti operazioni:

1) libero la memoria (rilascio) occupata dalla variabile NSString *strInput
2) assegno ad essa il puntatore della variabile stringa che mi viene restituta dal oggetto NSTextField, utilizzo il metodo stringValue.
3) Dato che l'oggetto NSTextField mi restituisce una istanza allocata da lui del puntatore alla stringa, invio un messaggio retain al mio oggetto NSString, in modo da garantirmi che la memoria non venga liberata in modo casuale ed autonomo ma solo quando eseguiro' il metodo release.
4) A questo punto, rilascio la memoria utilizzata dall'oggetto NSString *strOutput e chiamo il metodo reverse:
5) Il metodo reverse: per come e' stato realizzato, alloca ad ogni chiamata un nuovo oggetto, di conseguenza non devo effettuare il retain sulla stringa NSString *strOutput.

Il tutto e' sintetizzato in questo metodo

```
- (IBAction)btnElabora:(id)sender {
 [strInput release]; //Libero la memoria allocata in precedenza
 strInput = [[txtInput stringValue] retain]; //Ricavo la nuova stringa e invoco
retain perche' la variabile e' in autorelease da NSTextField

 [strOutput release]; //Libero la memoria allocata in precedenza
 strOutput = [self reverse:strInput];    //Crea la nuova stringa reversed, non
devo fare retain perche' il metodo reverse: alloca solo.
 [txtOutput setStringValue:strOutput]; //Aggiorna interfaccia utente
}
```

In pratica dato che l'oggetto NSString e' immutabile, ogni qual volta devo assegnare un nuovo valore devo prima rilasciare la memoria allocata dal precedente oggetto, quindi ri allocarla per in nuovo oggetto.
Se avessi usato delle NSMutableString, sarebbe stato sufficiente fare l'allocazione iniziale (nel metodo init) e quindi la release nel metodo dealloc; ovviamente non mi avrebbe consentito di cercare di spiegare il funzionamento dell'alloc/release.

Alla prossima per approfondire il discorso.

Saluti.