Il programmatore che c'e' in noi – Lezione 6 – Aritmetica e Puntatori

La volta scorsa abbiamo dato la definizione di una variabile puntatore, di seguito riassunta
"Un puntatore e' una variabile che contiene l'indirizzo di una locazione di memoria"

Vi arriva una busta chiusa, l'aprite, e trovate un bigliettino con un messaggio del tipo "presentati in via Verdi,n.8"
Andate in "via Verdi,n.8" e ci trovate una villa bifamiliare.

Bene, cosa significa ?

Semplice, la busta e' una variabile di tipo puntatore, infatti contiene l'indirizzo di un altro oggetto (la villa bifamiliare) non l'oggetto stesso (ci vorrebbe una busta enorme…)

Si dice quindi che la variabile PUNTA all'oggetto indicato dall'indirizzo.

Se nella stessa busta, sostituisco il bigliettino, con un altro su cui c'e' scritto "presentati in via Rossi n.25", significa che ho cambiato il valore alla mia variabile di tipo puntatore (ossia ho cambiato l'indirizzo dell'oggetto a cui essa punta).

Supponiamo ora che io vi consegni la busta e vi dica anche, andare alla villa successiva a quella indicata nella busta. Come vi comportate ?

La situazione di via Verdi e' questa:
via Verdi 8 – Prima Villa bifamiliare (prima villa)
via Verdi 9 – Prima Villa bifamiliare (seconda villa)

via Verdi 10 – Seconda Villa bifamiliare (prima villa)
via Verdi 11 – Seconda Villa bifamiliare (seconda villa)

Andate alla villa al N° 9 o quella al N° 10 ?

Dipende.

Dipende dal tipo di oggetto a cui punta il puntatore, quindi se l'indirizzo indicato nella nostra busta e' relativo ad una villa bifamiliare, quando dico di andare all'indirizzo successivo, andro alla successiva villa bifamiliare, quindi alla N. 10 di via Verdi.

Se invece, l'indirizzo indicato nella busta, e' relativo (=punta) ad una villa singola, allora la villa successiva e' quella al N° 9 di via Verdi.

Questo e' esattamente quello che avviene quando noi effettuiamo delle operazioni aritmetiche con i puntatori.

Per concludere quella similitudine, tra puntatori e buste con indirizzi… vi segnalo un ulteriore interessante particolare, la busta stessa e' un oggetto; di conseguenza potrei avere una ulteriore busta (variabile puntatore) che contiene l'indirizzo dove trovare una busta che contiene a sua volta l'indirizzo della villa.

In effetti posso avere un puntatore che punta ad un puntatore che punta ad un puntatore … ecc.

Abbiamo gia' incontrato dei particolari caratteri che hanno uno specifico significato proprio per consentire di lavorare con le variabili di tipo puntatore.

& Serve per ricavare l'indirizzo di memoria di un oggetto
* Serve per dichiarare una variabile di tipo puntatore

Lo stesso simbolo * si usa per l'operatore che e' complementare al quello & e serve per ottenere il valore contenuto nell'oggetto indicato dall'indirizzo specificato.

Esempio:

```
//dichiaro le variabili
int nNumeroCivico=0;  //variabile di tipo int
int* pUnNumeroCivico=nil; //variabile puntatore ad un int
int nUnAltroNumeroCivico=0; //variabile di tipo int

nNumeroCivico=8; //Assegno il valore alla variabile int
pUnNumeroCivico = & nNumeroCivico; //ricavo l'indirizzo di memoria della variabile ad un int

nUnAltroNumeroCivico = *pUnNumeroCivico; //ricavo il valore della variabile che si trova
                                         //all'indirizzo contenuto in pUnNumeroCivico
                                         // ossia l'indirizzo di nNumeroCivico e di
                                         // conseguenza il suo valore e cioe' 8.

//a questo punto la variabile nUnAltroNumeroCivico vale 8.

*pUnNumeroCivico = 12; //Cosa succede ?

//Semplice, ho modificato il valore contenuto nella variabile puntata da pUnNumeroCivico ossia
nNumeroCivico, quindi a questo punto nNumeroCivico vale 12.
```

OPERATORI ARITMETICI

Gli operatori aritmerici sono dei particolari simboli che ci consentono di effettuare delle operazioni aritmetiche sui numeri.
In linguaggio C/C++ e Objetive C gli operatori aritmetici sono

- Sottrazione
+ Addizione
* Moltiplicazione
/ Divisione
% Modulo (resto di una divisione tra numeri interi)
-- Decremento
++ Incremento

Esempi

```
int nAnnoNascita=1966;
int nAnnoCorrente=2006;
int nEta=0;

nEta= nAnnoCorrente – nAnnoNascita; //Uso operatore – per effettuare il calcolo e l'operatore =
per assegnare il risultato (40) alla variabile nEta
```

```
int nNumero=45;
BOOL bIsDispari=FALSE;

bIsDispari = numero % 2; //Uso operatore modulo %, per sapere se un numero e' dispari o meno, l'operatore modulo restituisce il resto della divisione tra interi, quindi se il numero e' pari il resto della divisione per 2 vale 0 altrimenti vale 1.


int nContatore=0;

nContatore++;  //uso l'operatore di incremento per incrementare di 1 il valore della variabile

// nContatore a questo punto vale 1

nContatore++;

// nContatore a questo punto vale 2

nContatore--; //uso l'operatore di decremento per decrementare di 1 il valore della variabile

// nContatore a questo punto vale 1
```

NOTE

In C/C++ e Objetive C posso usare anche gli operatori "composti" ossia operatore di assegnazione e operatore aritmetico

Esempio:

```
nContatore+=5; //uso l'operatore composto per assegnare a nContatore il valore di nContatore + 5
nContatore-=3; //uso l'operatore composto per assegnare a nContatore il valore di nContatore - 3
nContatore*=3; //uso l'operatore composto per assegnare a nContatore il valore di nContatore * 3
nContatore/=3; //uso l'operatore composto per assegnare a nContatore il valore di nContatore / 3
```

Gli operatori di incremento e decremento possono essere posto davanti (prefisso) o dietro (postfisso) la variabile, la differenza e' nella valutazione della valore della variabile.
Nel caso in cui siano posti davanti, prima si applica l'operatore (ossia si incrementa o decrementa il valore) e poi si valuta il valore della variabile, nel caso opposto, prima viene valutato il valore della variabile e poi si procede ad applicare l'operatore.

Esempio:

```
int nContatore=0;
int nValore=0;

//Se scrivo

nValore =++nContatore; // nValore = Vale 1, nContatore Vale 1 (prima viene incrementato il valore di nContatore da 0 a 1, quindi il valore viene assegnato alla variabile nValore)

//Se invece scrivo
```

nValore =nContatore++; // nValore = Vale 0, nContatore Vale 1 (prima viene assegnato il valore di nContatore (0) alla variabile nValore, quindi viene incrementato il valore di nContatore da 0 a 1)

Gli operatori possono essere unary,binary o ternary a seconda se hanno bisogno di un solo operando, di due o di tre.
Gli operatori aritmetici unary sono – (cambio segno) ++ (incremento) -- (decremento) mentre gli altri sono binary.

Per concludere il discorso sugli operatori arimetici, occorre dire che esiste una PRECEDENZA tra operatori, ossia, se in una espressione sono presenti piu' operatori, essi vengono valutati in base alla loro precedenza. E' possibile (anzi, vivamente consigliato…) utilizzare le parentesi tonde () per MODIFICARE la precedenza degli operatori o per meglio chiarire le intenzioni che abbiamo per la valutazione dell'espressione.

PRECEDENZA DEGLI OPERATORI ARITMETICI

Alta
++ --
- (unary cambio segno)
* / %
+ -
Bassa

Gli operatori allo stesso livello sono valutati dal compilatore partendo da sinistra verso destra.

Esempi:

```
int nValore=0;
int nPrimoNumero=5;
int nSecondoNumero=2;
int nTerzoNumero=4;

nValore = nPrimoNumero * ++nSecondoNumero – nTerzoNumero; // Quanto vale ?
//prima viene valutato l'operatore ++ che ha precedenza piu' alta, quindi ++nSecondoNumero = 3
//quindi viene valutato l'operatore * di moltiplicazione nPrimoNumero * 3 = 5*3=15
//Infine viene valutato l'operatore – di sottrazione 15 – nTerzoNumero= 15 – 4 = 11
//Il risultato dell'espressione e' quindi 11

//La seguente espressione fornisce come risultato -5
nValore = nPrimoNumero * (++nSecondoNumero – nTerzoNumero); // Quanto vale ?

//Dato che ci sono le parentesi, la precedenza e' modificata e quindi viene valutata in questo modo
//Prima viene incrementato il valore di nSecondoNumero (da 2 a 3)
//Poi ad esso viene sottratto il valore di nTerzoNumero (4) quindi 3 – 4 = -1
//Infine viene valutata l'espressione nPrimoNumero * -1 = 5 * -1 = -5
```

Per i piu' intraprendenti …

```
nValore = nPrimoNumero * nSecondoNumero++ – nTerzoNumero; // Quanto vale ?
//In questo caso, la valutazione dell'espressione da come risultato 6 come mai ?
```

Ovviamente il mio personale consiglio e' quello di scrivere sempre il codice (=il programma, il sorgente) nella maniera piu' chiara possibile (=leggibile)

Quindi, l'espressione che ho usato negli esempi precedenti, dovrebbe essere scritta in un modo piu' leggibile, simile a questo

```
int nValore=0;
int nPrimoNumero=5;
int nSecondoNumero=2;
int nTerzoNumero=4;

++nSecondoNumero;

 nValore = (nPrimoNumero * nSecondoNumero) – nTerzoNumero; // Quanto vale ?
```

Bene, siamo ora in grado di parlare della Aritmetica dei Puntatori, come indicato nel titolo di questa lezione…

Supponiamo di avere questa situazione

```
int nEta=10,nOldEta=0; //Dichiarazione multipla di variabili dello stesso tipo NOTARE l'utilizzo del simbolo , (virgola) e' un altro operatore del linguaggio C/C++ e ObjectiveC, che serve per separare una lista di istruzioni

int* pIndirizzoDiEta=nil; //dichiaro la variabile puntatore e la inizializza a nil (nulla)

pIndirizzoDiEta = &nEta; //Calcolo l'indirizzo di memoria della variabile nEta e lo memorizzo
                         //nella variabile puntatore

nOldEta=*pIndirizzoDiEta; //Ricavo il valore della variabile puntata dal puntatore e lo memorizzo
                          //in nOldEta, questa istruzione equivale a scrivere nOldEta = nEta;

pIndirizzoDiEta++;  //Incremento il valore di pIndirizzoDiEta, CHE E' UN indirizzo di memoria
                    //Di quanto e' stato incrementato ?
                    //A cosa punta il nuovo indirizzo ?
```

Per rispondere a queste domande, facciamoci dire quanto vale l'indirizzo contenuto in pIndirizzoDiEta prima e dopo l'incremento usando

```
NSLog(@"Indirizzo contenuto pIndirizzoDiEta=%p",pIndirizzoDiEta);
```

Questo e' il risultato relativo all'esecuzione sul mio mac,
pIndirizzoDiEta=0xbffffa18 (prima)
pIndirizzoDiEta=0xbffffa1c (dopo)

Se ora uso l'applicazione calcolatrice, e effettuo la sottrazione tra i due valori (sono esadecimali) ottengo 0x4 ossia 4 in decimale. QUATTRO ? Il valore del contenuto nella mia variabile puntatore si e' incrementata di un colpo solo di 4 non di 1 come probabilmente pensavamo.
Vi ricordate l'esempio fatto qualche riga piu' in alto delle villette bifamiliari, bene, con l'istruzione ++ abbiamo detto alla nostra variabile puntatore di puntare all'indirizzo successivo ad una variabile di tipo INT (dato che le variabili INT occupano 4 bytes in memoria, lo avevamo gia' verificato con l'uso dell'operatore sizeof() ricordate?) e' quindi corretto che il valore dell'indirizzo si sia

incrementato di 4 bytes. Ecco perche' e' importante sapere a che tipo di variabile punta una variabile puntatore.

Se invece dell'istruzione
pIndirizzoDiEta++;
scrivo
pIndirizzoDiEta+=4; //Il nuovo indirizzo sara' 4 locazioni successive all'indirizzo di partenza

//Dato che pIndirizzoDiEta e' un puntatore che punta ad un int e che un int occupa 4 bytes in
//memoria, con l'istruzione precedente mi sono spostato di 4 * 4 bytes = 16 Bytes.

Se invece dichiaro un puntatore ad un char, dato che un char occupa un solo byte, mi sarei spostato di 4 bytes e non di 16.

Esempio

```
int nEta=10;
char *pIndirizzoUnByte=nil; //Variabile puntatore a char
int* pIndirizzoDiEta=nil; //Variabile puntatore a int

pIndirizzoDiEta=&nEta; //Calcolo indirizzo e lo metto in pIndirizzoDiEta

pIndirizzoUnByte = pIndirizzoDiEta; //Assegno il valore di pIndirizzoDiEta anche a
                                    //pIndirizzoUnByte (vedi nota)

pIndirizzoDiEta++; //Indirizzo ora e' stato incrementato di 4 (un int vale 4 bytes)
pIndirizzoUnByte ++; //Indirizzo ora e' stato incrementato di 1 (un char vale 1 byte)
```

NOTA
L'istruzione
pIndirizzoUnByte = pIndirizzoDiEta;
genera un warning del compilatore, in quanto si stanno assegnando dati di tipi diversi (un valore di puntatore ad interi viene assegnato ad un puntatore a char). E' comunque una operazione lecita, per evitare la segnalazione occorre utilizzare un "CAST" di tipo, ossia una operazione di tipo di dato (nel nostro caso da tipo puntatore ad intero a puntatore a char).
L'istruzione nella forma corretta e' quindi
pIndirizzoUnByte = (char*) pIndirizzoDiEta; //uso del cast (char*) per convertire il tipo di dato

---

Se invece di incrementare l'indirizzo di una variabile puntatore, volessi incrementare il contenuto della variabile a cui punta l'indirizzo si procede nel seguente modo

```
(*pIndirizzoDiEta)++; //Incremento il valore puntato dall'indirizzo contenuto in pIndirizzoDiEta

(*pIndirizzoDiEta) += 4; //Sommo 4 al valore puntato dall'indirizzo contenuto in pIndirizzoDiEta
```

NOTA
Occorre usare le parentesi, altrimenti causa precedenza degli operatori, invece di incrementare il valore puntato dall'indirizzo incremento l'indirizzo stesso, come visto in precedenza.
*pIndirizzoDiEta++; //Incrementa l'indirizzo e poi ricava il contenuto a cui punta il nuovo indirizzo

Concludo con un rapido riepilogo di quanto visto.

```
int nEta=10; //Variabile di tipo int
int nValore=0; //Variabile di tipo int
int* pIndirizzoDiEta=nil; //dichiaro la variabile puntatore e la inizializza a nil (nulla)

pIndirizzoDiEta = &nEta; //Calcolo l'indirizzo di memoria della variabile nEta e lo memorizzo
                         //nella variabile puntatore

(*pIndirizzoDiEta)++; // Incremento il valore della variabile puntata da pIndirizzoDiEta quindi
                      // nEta, che passa da 10 a 11.

(*pIndirizzoDiEta)+=5; // Incremento il valore della variabile puntata da pIndirizzoDiEta quindi
                       // nEta, che passa da 11 a 16.

(*pIndirizzoDiEta)=25; // Assegno il valore della variabile puntata da pIndirizzoDiEta quindi
                       // nEta, che passa da 16 a 25.

nValore = (*pIndirizzoDiEta); //Assegno a nValore il valore puntato da pIndirizzoDiEta
                              //ossia nEta che attualmente vale 25

pIndirizzoDiEta++; //Incremento l'indirizzo di memoria contenuto in pIndirizzoDiEta che
//passa ad esempio da 0xbffffbac a 0xbffffbb0 (+4Bytes)

nValore = (*pIndirizzoDiEta); //Assegno a nValore il valore puntato da pIndirizzoDiEta
                              // !!!! ATTENZIONE !!! Ora non punta piu' a nEta, quindi non
                              //siamo in grado di sapere a cosa punta e di conseguenza a quale e' il
                              //valore contenuto alla //locazione di memoria indicata

(*pIndirizzoDiEta)=67;        //!!! ATTENZIONE !!! Operazione MOLTO pericolosa, nel senso
                              //che stiamo scrivendo in una zona di memoria di cui non conosciamo
                              //il significato di conseguenza si rischia di "sporcare"
                              //sino a rischiare il crash (=il blocco dell'applicazione, la chiusura
                              // inaspettata…)
                              //come l'esempio del log del crash della mia applicazione
                              //quando ho cercato di eseguire queste istruzioni
                              // pIndirizzoDiEta-=0xbffffbc4;
                              // (*pIndirizzoDiEta)=789;

Exception:  EXC_BAD_ACCESS (0x0001)
Codes:      KERN_INVALID_ADDRESS (0x0001) at 0xc0000d78
```

Spero, di essere riuscito a essere chiaro ed esauriente in questa spiegazione.
Le prossime lezioni, saranno di sicuro meno "pesanti", la parte piu' difficile ed ostica e' proprio questa sui puntatori, quindi sforzatevi di comprenderne appieno il funzionamento, dopo di che' tutto il resto e' una passeggiata (o quasi…).

Saluti, e a presto.