Il programmatore che c'e' in noi – Lezione 10 – Unioni

Premessa: non dovete avere mal di testa.

Una union e' una zona di memoria **condivisa** tra due o piu' variabili anche di tipo diverso tra di loro, ed e' molto simile ad una struct (di cui abbiamo parlato la volta scorsa) sia nella dichiarazione sia nell'utilizzo.

Per essere chiari, ecco un esempio che compara una struct con una union.

```
//Dichiarazione di una struct

    struct datiProdotto{
     char nome[21]; //Nome del prodotto 20 char
     char tipo[11]; //Tipologia
     double prezzo; //Prezzo
    }unaStrutturaProdotto;

   //In memoria ho a questo punto costruito qualcosa del genere
   //[01234567890123456789][0123456789][12345678]
   // nome                 tipo        prezzo
   //Totale Memoria Occupata = 21+11+8 = 40 BYTE

   //Vediamo se e' vero:
   int nDimStrutturaProdotto=sizeof(struct datiProdotto);
   NSLog(@"Memoria occupata dalla struttura prodotto:
%d\n",nDimStrutturaProdotto); //=40

   //Vediamo invece una union
   union datiProdottoU{
     char nome[21]; //Nome del prodotto 20 char
     char tipo[11]; //Tipologia
     double prezzo; //Prezzo
    }unaUnionProdotto;

   //In memoria questa volta ho costruito qualcosa del genere
   //[01234567890123456789] nome
   //[0123456789] tipo
   //[12345678] prezzo

   //Quindi tutte e tre le variabili utilizzano la stessa zona di memoria
   //Il totale della memoria occupata e' quindi quella della variabile che
occupa piu' memoria
   //e non dalla somma delle quantita

   //Totale Memoria Occupata = 21 BYTE (la variabile che occupa piu' spazio e'
nome di 21 Byte)

   //Vediamo se e' vero:
   int nDimUnionProdotto=sizeof(union datiProdottoU);
   NSLog(@"Memoria occupata dalla union prodotto: %d\n",nDimUnionProdotto);
//Gulp ! = 24 !!!

   //Bene, 24 e' comunque vicino a 21, che e' il valore che avevo previsto, ma
perche' e 24 ?

   //Il computer per ottimizzare la velocita' di esecuzione nelle operazioni che
riguardano
   //la copia o lo spostamento dei dati nella memoria, preferisce avere a che
fare con dimensioni
```

```
   //di dati che siano dei multipli un INT.

   //Tale comportamento relativo alle struct ed alle union e' chiamato "BYTE
ALLIGNED" e puo' essere
   //modificato tramite opportune opzioni del compilatore.
   //Consiglio a chi fosse interessato di consultare il manuale del compilatore

   //Assegnamo dei valori alla union e vediamo come si comporta
   strcpy(unaUnionProdotto.nome,"Lamponi e Fragole");

   //Vediamo cosa contiene la nostra union
   NSLog(@"unaUnionProdotto.nome = '%s'\n",unaUnionProdotto.nome);

   //Ora assegno il valore alla seconda variabile ...
   strcpy(unaUnionProdotto.tipo,"Frutta");

   //Vediamo cosa contiene la nostra union
   NSLog(@"unaUnionProdotto.nome = '%s'\n",unaUnionProdotto.nome);
   //Vediamo cosa contiene la nostra union
   NSLog(@"unaUnionProdotto.tipo = '%s'\n",unaUnionProdotto.tipo);

   //DATO CHE LA ZONA DI MEMORIA E' LA STESSA, QUANDO ASSEGNO IL VALORE A TIPO,
VADO
   //AUTOMATICAMENTE A MODIFICARE IL VALORE DI NOME
   //In realta' in memoria dovrei avere questa situazione
   //0------------------------------24
   //Lamponi e Fragole'\0'               Dopo la prima assegnazione
   //Frutta'\0' e Fragole'\0'            Dopo la seconda assegnazione

   //Dato che sono delle stringhe C, ossia null terminated, chiuse quindi dal
carattere 0x00 (NUL)
   //quando chiedo la stampa del contenuto della variabile con NSLog() ottengo
Frutta

   //Cosa succede se ora assegno un valore numerico a prezzo ? Vediamo.
   unaUnionProdotto.prezzo=1200;

   //Vediamo cosa contiene la nostra union
   NSLog(@"unaUnionProdotto.nome = '%s'\n",unaUnionProdotto.nome); // '@í¿'
   NSLog(@"unaUnionProdotto.tipo = '%s'\n",unaUnionProdotto.tipo); // '@í¿'
   NSLog(@"unaUnionProdotto.prezzo = '%.0f'\n",unaUnionProdotto.prezzo); // 1200

   //Quindi ogni assegnazione che vado a fare ad una variabile, comporta la
modifica del contenuto
   //anche delle altre variabili, questo e' dovuto al fatto che TUTTE le
variabili stanno
   //PUNTANDO alla stessa zona di memoria.
   //Controlliamo se e' vero ...

    NSLog(@"\n\nTutte le variabili della struct puntano ad una locazione propria
della memoria\n");

    NSLog(@"Indirizzo unaStrutturaProdotto.nome   =
0x%x\n",&unaStrutturaProdotto.nome);
    NSLog(@"Indirizzo unaStrutturaProdotto.tipo   =
0x%x\n",&unaStrutturaProdotto.tipo);
    NSLog(@"Indirizzo unaStrutturaProdotto.prezzo =
0x%x\n",&unaStrutturaProdotto.prezzo);

    NSLog(@"\n\nTutte le variabili della union puntano alla stezza locazione di
inizio della memoria\n");

    NSLog(@"Indirizzo unaUnionProdotto.nome   = 0x%x\n",&unaUnionProdotto.nome);
```

```
    NSLog(@"Indirizzo unaUnionProdotto.tipo   = 0x%x\n",&unaUnionProdotto.tipo);
    NSLog(@"Indirizzo unaUnionProdotto.prezzo =
0x%x\n",&unaUnionProdotto.prezzo);
```

Ora che abbiamo appreso come funziona una union, la domanda e' "a cosa serve ?"

Bene, diciamo che e' molto utile per fare delle conversioni dati, quando si lavora con i singoli byte o bit.

Esempio:
Ho un numero intero, tale numero e' rappresentato in memoria (sul nostro Mac) da 4 BYTE (lo abbiamo visto ad inizio corso), voglio conoscere il valore dei singoli byte.
Una possibile soluzione consiste nell'utilizzare una union.
In pratica "sovrappongo" una variabile int e un array di 4 unsigned char.

```
    //Dichiaro una union come composta da un int e da 4 unsigned char
    union convInt{
     int nValore;
     unsigned char byte[4];
    }dato;

    //Assegno il valore all'intero
    dato.nValore=168987384;
    //Leggo il valore dell'intero
    NSLog(@"Valore di  nValore = %d\n",dato.nValore);

    //Leggo il valore dell'intero come rappresentato dai 4 BYTE in memoria
    NSLog(@"Valore di  nValore = %d BYTE0: %d BYTE1: %d BYTE2: %d BYTE3: %d
\n",dato.nValore,dato.byte[0],dato.byte[1],dato.byte[2],dato.byte[3]);

A questo punto il problema che ci eravamo posti e' stato risolto.

E se volessi calcolare il valore di un int conoscendo i valori dei singoli byte
che lo rappresentano ?

Per convertire il valore rappresentato in BYTE in un int e' sufficiente
moltiplicare il valore di ciascun byte per le potenze di 256 descrescenti a
partire da 3

    //quindi valoreIntero= byte0 * 256^3 + byte1*256^2+byte2*256^1+byte3*256^0

    //semplificando (in quanto 256^1 = 256 e 256^0=1)

    //valoreIntero= byte0 * 256^3 + byte1*256^2+byte2*256+byte3

    int aValoreRappresentato=(dato.byte[0] * pow(256,3)) + (dato.byte[1] *
pow(256,2)) + (dato.byte[2] * pow(256,1)) + (dato.byte[3] * pow(256,0));

     NSLog(@"valore rappresentato: %d",aValoreRappresentato);
```

Bene, ovviamente se modifico il valore di un byte della union, cambiera' il valore della variabile intero in essa presente.

```
//Se cambio il valore di un byte, cambia anche il valore dell'intero presente
nella union
    dato.byte[0]=0;
    dato.byte[1]=0;
    dato.byte[2]=2;
```

```
dato.byte[3]=128;

NSLog(@"Valore di  nValore = %d\n",dato.nValore);
```

ora vale 640 (provate a calcolarlo con il metodo sopra descritto…)

Per i piu' curiosi e volenterosi
Se dovessimo rappresentare il valore binario di un carattere, come possiamo fare, usando le struct e le union esiste una semplice soluzione, che trovate nell'esempio legato a questa lezione.
Provare a farlo senza guardare la soluzione!
Chi vuole potrebbe implementare il programma CheCarattere creato in una delle scorse puntate aggiungendo la possibilita' di visualizzare anche il valore binario dei caratteri e renderlo disponibile anche agli altri.

Saluti.