Lezione 27 C e assembly 5

mercoledì 29 novembre 2023 17:40

Andiamo a parlare delle **variabili booleane** : variabili che valgono 0/1->false/true. Questo tipo di dato non esiste , o meglio nessuna isa implementa questi che li gestisce. In C per gestire questo tipo di dato si usa un **header : stdbool.h** , il quale header definisce parole chiave tipo true, false, bool. In assembler questi confronti vengono implementati dalla JZ. Facciamo una bella ripassata degli operatori in generale : logici, aritmetici, ecc ecc :

Operatori Aritmetici	
+	somma
-	sottrazione
*	moltiplicazione
/	divisione
%	modulo
++	incremento
--	decremento

Operatori di Confronto	
==	uguaglianza
!=	non uguaglianza
>	maggiore di
<	minore di
>=	maggiore o uguale
<=	minore o uguale

Operatori Logici	
&&	and logico
||	or logico
!	negazione logica
? :	confronto ternario

Operatori Bit a Bit	
&	and binario
|	or binario
^	xor binario
~	complemento a 1
<<	shift a sinistra
>>	shift a destra

Assegnazione	
=	assegnazione *uguale*
+=	assegnazione *più-uguale*
-=	assegnazione *meno-uguale*
*=	assegnazione *per-uguale*
/=	assegnazione *diviso-uguale*
%=	assegnazione *modulo-uguale*
<<=	assegnazione *shift-sinistro-uguale*
>>=	assegnazione *shift-destro-uguale*
&=	assegnazione *and-uguale*
^=	assegnazione *xor-uguale*
|=	assegnazione *or-uguale*

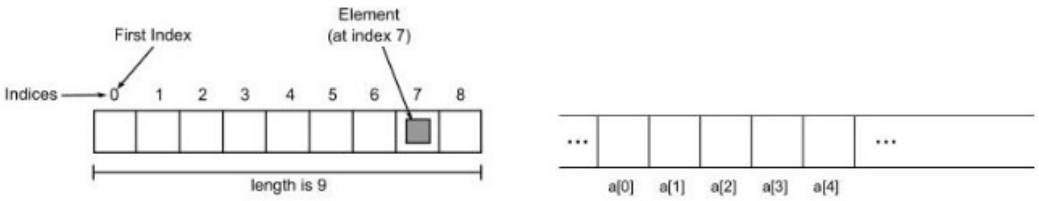
Vediamo un esempio :

```c
#include <stdio.h>

int main(int argc,char *argv[])
{
        int i=26;
        do{
                /*
                decremento
                da notare il momento in cui voglio accedere
                in questo caso POST-DECREMENTO
                prima leggo e poi decremento
                */
                printf("%d, ",--i);
        }while(i);
        /*
        il cursore arretra di due caratteri
        quindi si ha la sovrapposizione dei caratteri
        */
        printf("\b\b \n");

        i=0;
        while(i<26)
        {
                printf("%d",i);
                i++;
        }
        printf("\b\b \n");
        return 0;

}
```

```
daniele@daniele-Aspire-E5-571G:~/Scrivania/programmi_calcolatori/esercizi_c$ gcc bool_while.c -o bool
daniele@daniele-Aspire-E5-571G:~/Scrivania/programmi_calcolatori/esercizi_c$ ./bool
25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
01234567891011121314151617181920212223 24 5
```

Andiamo a vedere un'altra possibile collezione di elementi : **gli array .** Questi elementi in

memoria , vengono rappresentati in modo contiguo (vedasi memoria flat) . In base al tipo di dato che deve immagazzinare, si ha una differente taglia del dato; inoltre ricordiamoci l'architettura little endian.



Quindi un classico esempio di vettore è il seguente, dove le parentesi "[ ]" (spiazzamento) stanno ad indicare il numero di bye che mi devo spostare, sempre in relazione al tipo di dato :

A[i]=a + size*i

Dove size rappresenta la taglia dell'oggetto (ciascun elemento), mentre i rappresenta il numero di elementi di cui mi voglio spostare.
Vediamolo ora in assembly :

```
  movq $0, %rcx # %rcx viene usato come indice
  movq $array, %rax # %rax viene usato come base
loop:
  movq (%rax, %rcx, 8), %rdx # sposta i dati dove serve
  # <processa i dati>
  addq $1, %rcx
  cmpq $size, %rcx
  jnz .loop
```

Da notare che in %rax si ha l'indirizzo di base del vettore; nell'istruzione di spiazzamento si accede in memoria , per poi farci qualunque cosa. Poi per aggiornare l'indice aggiungo 1 al contatore , che in memoria ed essendo/operando con quadword, incremento di 8!!
In alternativa a questo :

```
  movq $0, %rcx # %rcx viene usato come indice
loop:
  movq array(, %rcx, 8), %rdx # sposta i dati dove serve
  # <processa i dati>
  addq $1, %rcx
  cmpq $size, %rcx
  jnz .loop
```

In quanto si usa spiazzamento con operandi nel bss : indirizzi vicini alla sezione dati : risparmio un registro base !!
Altro esempio:

```
 movq $array, %rax # carica indirizzo del primo elemento
 movq $size, %rcx
oop:
 movq (%rax), %rdx # sposta i dati dove serve
 # <processa i dati>
 addq $24, %rax
 subq $1, %rcx
 jnz .loop
```

Attenzione se elemento più grande di quadword (elementi di scala >8) : in questo caso devo incrementare del numero di byte che mi serve , a patto che parto da indirizzo di base. Notiamo che a posto di cmp usiamo la sub : bit di flag testato dento qualunque operazione .Andiamo a vedere ora un'altra rappresentazione in memoria : **le Stringhe:**
Ovvero una rappresentazione in memoria di caratteri ; non si conosce a priori la memoria , quindi alla fine si inserisce il **terminatore di stringa ('\0'),** per evitare undefined behaviour .
Vediamo un esempio :

Andiamo a vedere un esempio pratico :

```c
#include <stdio.h>
int main(int atrg,char *argv[])
{
        int nums[4]={0};
        char name[4]={'a','p'};

        /*
        qui nel primo caso , del vettore di interi
        stampa tutto il vettore inizializzato a 0;
        Nel secondo caso invece , anche avendo un vettore di 4 elementi
        ne stampa solo 2 : gli altri due sono i terminatori di stringa ;
        Nel terzo caso outpu simile , perchè il compilatore va in memoria
        e scorre tutto l'array finche non trova il primo terminatore
        */
        printf("nums : %d%d%d%d\n",nums[0],nums[1],nums[2],nums[3]);
        printf("name each :%c%c%c%c \n",name[0],name[1],name[2],name[3]);
        printf("name: %s\n",name);
}
```

```
nums : 0000
name each :ap
name: ap
```

Inizializziamo ora i vettori :

```c
#include <stdio.h>
int main(int atrg,char *argv[])
{
        int nums[4]={0};
        char name[4]={'a','p'};

        /*
        qui nel primo caso , del vettore di interi
        stampa tutto il vettore inizializzato a 0;
        Nel secondo caso invece , anche avendo un vettore di 4 elementi
        ne stampa solo 2 : gli altri due sono i terminatori di stringa ;
        Nel terzo caso outpu simile , perchè il compilatore va in memoria
        e scorre tutto l'array finche non trova il primo terminatore
        */
        printf("nums : %d%d%d%d\n",nums[0],nums[1],nums[2],nums[3]);
        printf("name each :%c%c%c%c \n",name[0],name[1],name[2],name[3]);
        printf("name: %s\n",name);

        /*
        andiamo ora ad "popolare" i vettori
        e ristampiamo i valori
        */
        nums[0] = 1;
        nums[1] = 2;
        nums[2] = 3;
        nums[3] = 4;

        name[0] = 'A';
        name[1] = 'l';
        name[2] = 'e';
        name[3] = '\0';

        printf("nums: %d %d %d %d\n", nums[0], nums[1], nums[2], nums[3]);
        printf("name each: %c %c %c %c\n", name[0], name[1], name[2], name[3]);
        printf("name: %s\n", name);
}
```

```
nums : 0000
name each :ap
name: ap
nums: 1 2 3 4
name each: A l e
name: Ale
```

Attenzione che questo programma è logicamente corretto , ma è sbagliato in quanto questa è prima invocazione di funzione : **lo stack è vuoto. In determinate circostanze può portare ad undefined behaviour.** Torniamo al discorso della lunghezza di un tipo o di una variabile (restituito in multipli di char) : **SIZEOF : nel caso di char vale sempre 1**

Vediamo ora come codificare elementi e mapparli su un determinato tipo : **enum :** ci possiamo anche differenziare i vari tipi di casi (come per esempio switch), ed assegnargli un valore numerico:

```c
#include<stdio.h>

/*
il compilatore in base al numero degli elementi
comincia ad assegnargli un valore crescente
partendo da 0 :
        CLUB=0
        DIAMONDS=1
        HEARTS=2
        SPADES=3
*/
enum suit{
        CLUB,
        DIAMONDS,
        HEARTS=20,
        SPADES=3
}card;

//qui non passiamo argomenti : argc=0 e argv nullo
int main (void)
{
        card = CLUB;
        // se stampiamo il valore con %d il compilatore da warning
        // stampa il numero di interi
        printf("Size of enum variable = %ld bytes\n", sizeof(card));
        return 0;
}
```

```
Size of enum variable = 4 bytes
```

Vediamo ora come combinare entrambi i concetti : creiamo un array dinamico : **array a lunghezza variabile** : istanziati su stack a tempo di compilazione:

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

//sequenza di numeri pseudo casuali
//compeso tra 0 e max (rand(max))
void generate_and_print(int n)
{
        int i;
        double vals[n];
        for (i=0;i<n;i++)
        {
                vals[i]=(double)rand()/RAND_MAX*50;
        }
        for (i=0;i<n;i++)
        {
                //stampo il double (precisione  singola)
                //mi fermo a 3 cifre decimali
                printf("%.03f",vals[i]);
        }
        printf("\b\b\n");
}

int main(void)
{
        srand(time(NULL));
        generate_and_print(3);
        generate_and_print(12);
        generate_and_print(1);
}
```

```
4.9296.5401.358
2.07623.05122.27640.58022.1118.25615.2815.02811.7557.00022.5883.229
8.872
```