Lezione 29 C e assembly 7

mercoledì 28 febbraio 2024 17:54

Andiamo a vedere altro esempio : **lo shifter:** questo programma fa vedere lo shift del valore di una variabile, o meglio del tipo di dato sia in codifica decimale, sia in binario :

```
 GNU nano 4.8                                                    shift.c
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>

#define MAX_BITS (int)(sizeof(unsigned int)*CHAR_BIT)
char string[MAX_BITS+1];
char *print_bits(unsigned int value);

char *print_bits(unsigned int value)
{
        int i;
        char *p;
        p = string + MAX_BITS;
        p[0] = 0;

        for(i = 0; i < MAX_BITS; i++) {
                *--p = value & 1 ? '1' : '0';
                value >>= 1;
        }
        return string;
}

int main()
{
        unsigned int a=256, sa;
        int b=-256,sb;
        sa=a>>2; // qui traslo a dx di 2 posizioni ( divido per 4)-> srl (shift right logical)
        sb=b>>2; // qui traslo a dx di 2 posizioni ( divido per 4)-> srl (shift right logical)
        printf("a\t\t= %03d (%s)\n", a, print_bits(a));
        printf("after shift\t= %03d (%s)\n\n", sa, print_bits(sa));
        printf("b\t\t= %04d (%s)\n", b, print_bits(b));
        printf("after shift\t= %04d (%s)\n", sb, print_bits(sb));
        return 0;
}
```

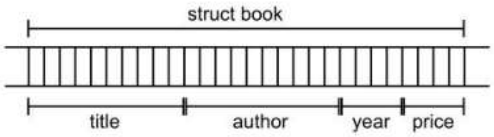
Il quale snippet di codice produce questo output :

```
a               = 256 (00000000000000000000000100000000)
after shift     = 064 (00000000000000000000000001000000)

b               = -256 (11111111111111111111111100000000)
after shift     = -064 (11111111111111111111111111000000)
```

Andiamo ora a vedere un importante costrutto del c : **le struct ,** ovvero aggregazioni logiche di tipi di dati :

```
struct book {
    char title[10];
    char author[10];
    int publication_year;
    float price;
};
```



Mentre ora vediamo un esempio di come poterle usare :

```c
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <string.h>

//qui si dichiara una struttura di tipo persona che contiene nome,eta,peso ed altezza

struct person{
        char *name;
        int age;
        int height;
        int weight;
};

struct person *person_create(struct person *who,char *name, int age, int height,int weight)
{
        //assert se persona non e nulla
        assert(who != NULL);
        who->name = strdup(name);
        who->age = age;
        who->height = height;
        who->weight = weight;
        return who;
}
void person_print(struct person *who)
{
        printf("Name: %s\tAge: %d\tHeight: %d\tWeight: %d\n",who->name,who->age,who->weight, who->weight);
}

int main()
{
        struct person joe;
        (void)person_create(&joe, "Joe Alex", 32, 64, 140);
        printf("Joe is at memory location %p:\n", &joe);
        person_print(&joe);
        joe.age += 20;
        person_print(&joe);
}
```

Con il seguente output :

```
Joe is at memory location 0x7ffc0093f080:
Name: Joe Alex  Age: 32 Height: 140     Weight: 140
Name: Joe Alex  Age: 52 Height: 140     Weight: 140
```

Vediamo ora altro esempio , di come inizializzare la struct :

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

struct student{
        char name[100];
        int roll;
        float marks;
};
// funzione usata per stampare i dati
void print_structure(struct student *p)
{
        printf("Name :%s\tRoll: %d\tMarks :%f\n",p->name,p->roll,p->marks);
        printf("\n");
}

int main()
{
        //modo "1" per inizializzare la struct
        struct student stu1={"Mario",12,79.5};
        //modo "2" per inizializzare la struct
        struct student stu2={.roll=15,.name="Luigi",.marks=80.0};
        print_structure(&stu1);
        print_structure(&stu2);
}
```

Con il seguente ouput:

```
Name :Mario     Roll: 12        Marks :79.500000

Name :Luigi     Roll: 15        Marks :80.000000
```

Analogamente a quanto detto per il C, andiamo ora a vedere le **struct** in assembler : per accedere ad ogni singolo campo della struttura , si usa la modalità di indirizzamento generale, ovvero base+(indice)*spiazzamento . In dettaglio : si calcola spiazzamento a partire da indirizzo base della struct , si carica indirizzo iniziale della struct in un registro , si utilizza operando in memoria composto da spiazzamento + base :

```c
struct student
{
    char  name[100];
    int   roll;
    float marks;
};

struct student stu1;
stu1.roll = 12;
```

```
movq $stu1, %rax
movl $12, 100(%rax)
```

Tornando alle struct , dobbiamo fare attenzione !! In quanto possono esistere struct (union) incomplete !! . Ma attenzione , visto che sono incomplete (o meglio non totalmente definite) non si posso istanziare, ma comunque è possibile definirne un puntatore:

```c
struct thing *pt;
```

Applicando questa dichiarazione, è possibile risolvere i riferimenti circolari tra header differenti per realizzare strutture dati ricorsive , **quindi con questa "tecnica" si nascondono i tipi di dato della struct ,** quindi si ha una sorta di struct astratta.

person.h:

```c
struct person;

struct person *person_create(struct
 person *who, char *name, int age, int
 height, int weight);
```

person.c:

```c
#include “person.h”
struct person {
        char *name;
        int age;
        int height;
        int weight;
};
```

Quindi in base a quanto detto finora, in C (funzioni) il passaggio dei parametri avviene sempre per valore , ricevendo una copia del valore , mentre nel caso di vettori/matrici copia del puntatore , mentre nel caso di struct avviene tramite la copia di tutta la struttura. Andiamo ora a vedere un esempio :

```c
struct huge {
    char member[4096];
};

extern void f1(struct huge s);
extern void f2(struct huge *s);

struct huge glbl;

void f(void) {
    f1(glbl);
}
```

```
pushq   %rbp
movq    %rsp, %rbp
subq    $4096, %rsp
movq    %rsp, %rdi
movq    $glbl, %rsi
movq    $512, %rcx
cld
rep movsq
call    f1
addq    $4096, %rsp
leave
ret
```

Andiamo ad applicare quanto detto finora , o meglio andiamo a vedere il movimento delle stringhe in assembler : per farlo si usano due istruzioni dedicate : **movs e stos** : entrambe utilizzano registri impliciti :

1. RCX: contatore del numero di operazioni da eseguire
2. RSI: indirizzo sorgente
3. RDI: indirizzo destinazione
4. RAX: valore cui impostare la memoria
5. DF (direction flag): identifica operazione da svolgere
    a. Se 0 si va in avanti
    b. Se 1 si va indietro

Vediamo un esempio :

`movs`: move data from string to string

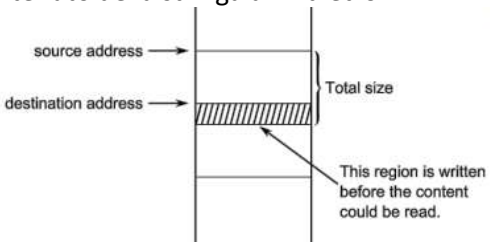
```
movq $source, %rsi
movq $destination, %rdi
movq $size/8, %rcx
cld
movsq
```

`stos`: store string

```
movq $0x0, %rax
movq $destination, %rdi
movq $size/8, %rcx
cld
stosq
```

Attenzione però a "muovere le stringhe" : se nel caso che la src e la dst sono sovrapposte : si ha inconsistenza, quindi si opta per copiare il contenuto della stringa all'indietro :

```
movb $0x800, %rsi
movb $0x810, %rdi
movb $0x20, %rcx
cld
movsb
```



Analogamente in C:

```
#include <string.h>
```

```
void *memcpy(void *restrict dest, const void *restrict src, size_t n);
```

Effettua una copia memoria/memoria in avanti

```
void *memmove(void *dest, const void *src, size_t n);
```

Effettua una copia memoria/memoria all'indietro (i buffer possono sovrapporsi)

```
void *memset(void *s, int c, size_t n);
```

Imposta un'area di memoria al valore `c`