

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

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
                           delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;

    for(i=0; i<MAXPAROLA; i++)
        freq[i]=0;

    if(argc != 2)
    {
        fprintf(stderr, "ERRORE, serve un parametro con il nome del file\n");
        exit(1);
    }
    f = fopen(argv[1], "r");
    if(f==NULL)
    {
        fprintf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
        exit(1);
    }

    while( fgets(riga, MAXRIGA, f) != NULL )
    {
        inizio = 0;
        lunghezza = strlen(riga);
        for(i=0; i<lunghezza; i++)
            if(isspace(riga[i]))
                inizio = i+1;
            else
                break;
        if(inizio == lunghezza)
            continue;
        riga[inizio] = '\0';
        lunghezza -= inizio;
        if(lunghezza > MAXPAROLA)
            lunghezza = MAXPAROLA;
        freq[lunghezza]++;
    }
}
```

Parametri “by reference”

Introduzione

Passaggio dei parametri

- Il linguaggio C prevede il passaggio di parametri “by value”
 - Il chiamato non può modificare le variabili del chiamante
 - Il parametro formale viene inizializzato con una copia del valore del parametro attuale

Problemi

- Il passaggio “by value” risulta inefficiente qualora le quantità di dati da passare fossero notevoli
 - Nel caso del passaggio di vettori o matrici, il linguaggio C non permette il passaggio “by value”, ma copia solamente l’indirizzo di partenza
 - Esempio: `strcmp`
- Talvolta è necessario o utile poter modificare il valore di una variabile nel chiamante
 - Occorre adottare un meccanismo per permettere tale modifica
 - Esempio: `scanf`

Soluzione

- La soluzione ad entrambi i problemi è la stessa:
 - Nel passaggio di vettori, ciò che viene passato è solamente l'indirizzo
 - Per permettere di modificare una variabile, se ne passa l'indirizzo, in modo che il chiamato possa modificare direttamente il suo contenuto in memoria
- Viene detto passaggio “by reference” dei parametri
 - Definizione impropria, in quanto gli indirizzi sono, a loro volta, passati “by value”

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

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
                           delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;

    for(i=0; i<MAXPAROLA; i++)
        freq[i]=0;

    if(argc != 2)
    {
        fprintf(stderr, "ERRORE, serve un parametro con il nome del file\n");
        exit(1);
    }
    f = fopen(argv[1], "r");
    if(f==NULL)
    {
        fprintf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
        exit(1);
    }

    while( fgets(riga, MAXRIGA, f) != NULL )
```

Parametri “by reference”

Operatori & e *

Operatori sugli indirizzi

- Per gestire il passaggio “by reference” dei parametri occorre
 - Conoscere l’indirizzo di memoria di una variabile
 - Operatore &
 - Accedere al contenuto di una variabile di cui si conosce l’indirizzo ma non il nome
 - Operatore *
- Prime nozioni della aritmetica degli indirizzi, che verrà approfondita in Unità successive

Operatore &

- » L'operatore **indirizzo-di** restituisce l'indirizzo di memoria della variabile a cui viene applicato
 - `&a` è l'indirizzo 1012
 - `&b` è l'indirizzo 1020

int a → 1012	37
int b → 1020	-4

Osservazioni

- L'indirizzo di una variabile viene deciso dal compilatore
- L'operatore & si può applicare solo a variabili singole, non ad espressioni
 - Non ha senso &(a+b)
 - Non ha senso &(3)
- Conoscere l'indirizzo di una variabile permette di leggerne o modificarne il valore senza conoscerne il nome

Variabili “puntatore”

- Per memorizzare gli indirizzi di memoria, occorre definire opportune variabili di tipo “indirizzo di...”
- Nel linguaggio C si chiamano **puntatori**
- Un puntatore si definisce con il simbolo *
 - `int *p ; /* puntatore ad un valore intero */`
 - `float *q ; /* puntatore ad un valore reale */`

```

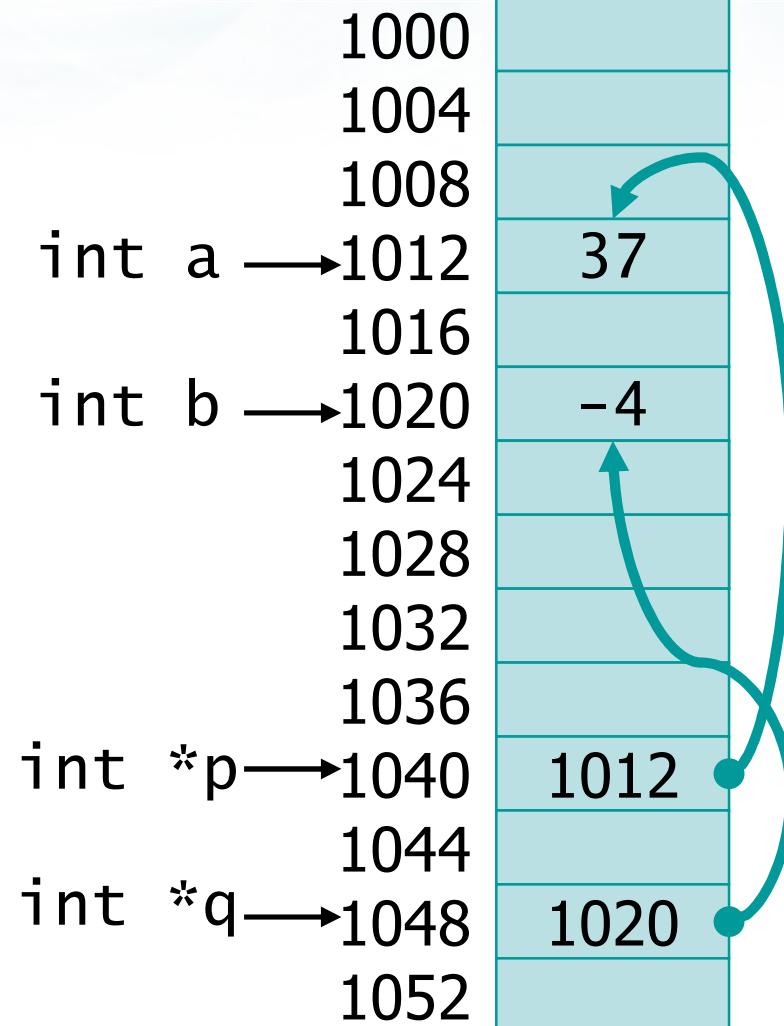
int main(void)
{
    int a, b ;
    int *p, *q ;

    a = 37 ;
    b = -4 ;

    p = &a ;
    /* p "punta a" a */

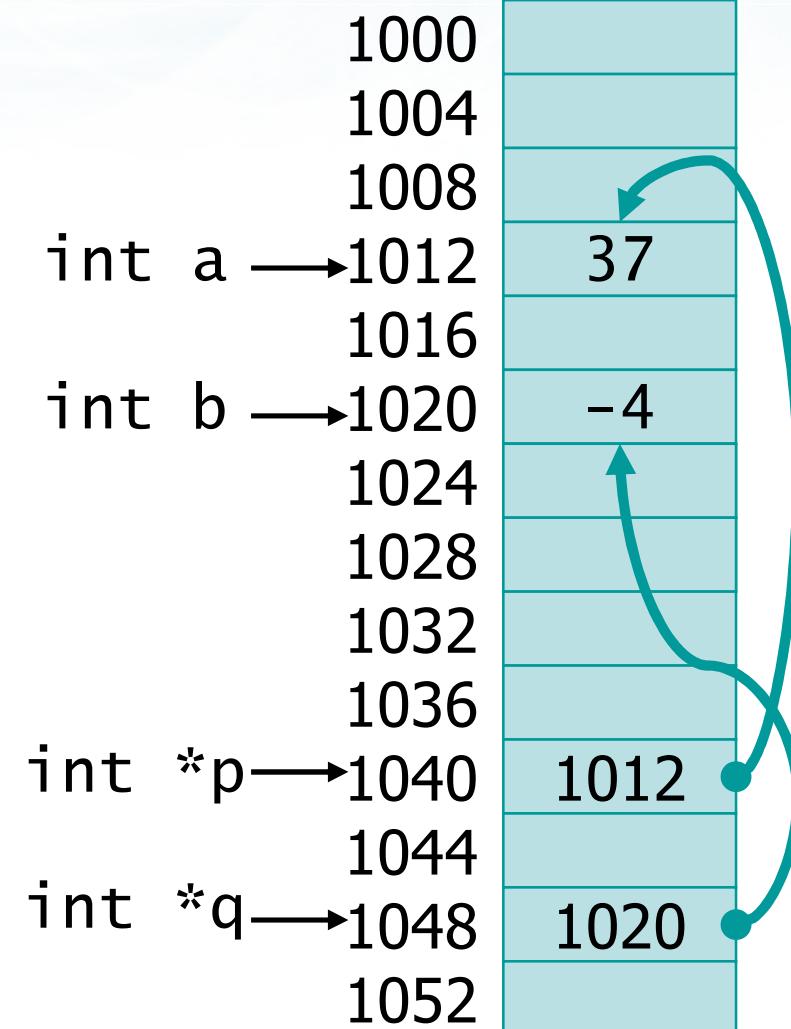
    q = &b ;
    /* q "punta a" b */
}

```



Operatore *

- L'operatore di **accesso indiretto** permette di accedere, in lettura o scrittura, al valore di una variabile di cui si conosce l'indirizzo
 - `*p` equivale ad `a`
 - `*q` equivale a `b`
 - `*p = 0 ;`
 - `if(*q > 0) ...`



Costrutti frequenti

Costrutto	Significato
int x ;	x è una variabile intera
int *p ;	p è un puntatore a variabili intere
p = &x ;	p punta ad x
*p = 0 ;	Azzera la variabile puntata da p (cioè x)
b = *p ;	Leggi il contenuto della variabile puntata da p e copialo in b

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

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
                           delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;

    for(i=0; i<MAXPAROLA; i++)
        freq[i]=0;

    if(argc != 2)
    {
        fprintf(stderr, "ERRORE, serve un parametro con il nome del file\n");
        exit(1);
    }
    f = fopen(argv[1], "r");
    if(f==NULL)
    {
        fprintf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
        exit(1);
    }

    while( fgets(riga, MAXRIGA, f) != NULL )
```



Parametri “by reference”

Passaggio “by reference”

Passaggio “by reference”

- Obiettivo: passare ad una funzione una variabile, in modo tale che la funzione la possa modificare
- Soluzione:
 - Definire un parametro attuale di tipo puntatore
 - Al momento della chiamata, passare l'indirizzo della variabile (anziché il suo valore)
 - All'interno del corpo della funzione, fare sempre accesso indiretto alla variabile di cui è noto l'indirizzo

Esempio: “Azzera”

- Scrivere una funzione azzera, che riceve un parametro di tipo intero (by reference) e che azzera il valore di tale parametro

Soluzione

```
void azzera( int *v ) ;
```

```
int main( void )
{
    int x ;
    ...
    azzera(&x) ;
    ...
}
```

```
void azzera( int *v )
{
    *v = 0 ;
}
```

Esempio: “Scambia”

- Scrivere una funzione `scambia`, che riceve due parametri di tipo intero (by reference) e che scambia tra di loro i valori in essi contenuti

Soluzione

```
void scambia( int *p, int *q ) ;
```

```
int main( void )
{
    int a,b ;
    ...
    scambia(&a, &b) ;
    ...
}
```

```
void scambia( int *p, int *q )
{
    int t ;
    t = *p ;
    *p = *q ;
    *q = t ;
}
```

Osservazione

- Il meccanismo di passaggio by reference spiega (finalmente!) il motivo per cui nella funzione scanf è necessario specificare il carattere & nelle variabili lette
- Le variabili vengono passate by reference alla funzione scanf, in modo che questa possa scrivervi dentro il valore immesso dall'utente