Puntatori e passaggio per riferimento

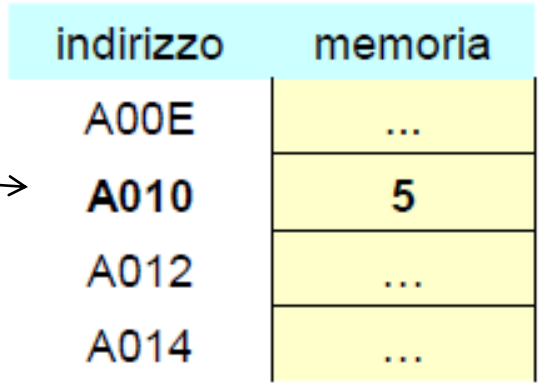
Lezione frontale #3

Argomenti:
-Puntatori
-Procedure e funzioni (avanzato)
-I/O (scanf)
-Memoria dinamica (base)

PARTE 1

Variabili tradizionali vs. Puntatori

- Variabili tradizionali
  - *Esempio:* ***int a = 5;***
- Proprietà della variabile a:
  - nome: a
  - tipo: int
  - valore: 5
  - **indirizzo: A010**



- Finora abbiamo usato solo le prime tre proprietà
- Come si usa l'indirizzo?
  - **&a**
  - **operatore indirizzo "&" applicato alla variabile a**
  - valore 0xA010 (esadecimale)

Variabili tradizionali vs. Puntatori

- Gli indirizzi si utilizzano nelle variabili di tipo puntatore, dette anche **puntatori**
- *Variabili di tipo puntatore*
- *Esempio:* ***int *p;***
- Proprietà della variabile p:
  - nome: p
  - tipo: **puntatore a intero**
    - (ovvero, indirizzo di una cella di memoria per un num intero)
  - valore: **inizialmente casuale (<- attenzione!)**
  - indirizzo: **fissato una volta per tutte**

Sintassi della dichiarazione di variabili puntatore

- **<tipo> *<identificatore>;**
- *Esempio:*
  - **int *p1;**
  - **int* p2, i, *p3, j;**
  - **char *p4, c, *p5;**
- p1, p2, p3 sono di tipo puntatore a int
- i, j sono di tipo int
- p4, p5 sono di tipo puntatore a char
- c è di tipo char

NOTA:

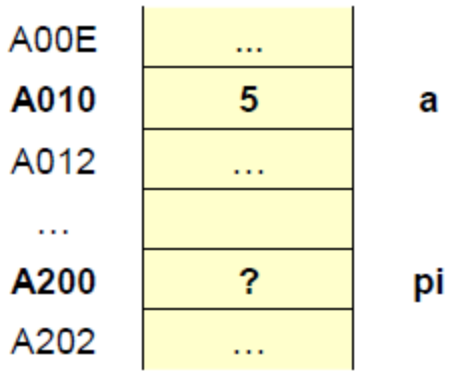
```
int *pi;
int * pi;
int* pi;
```

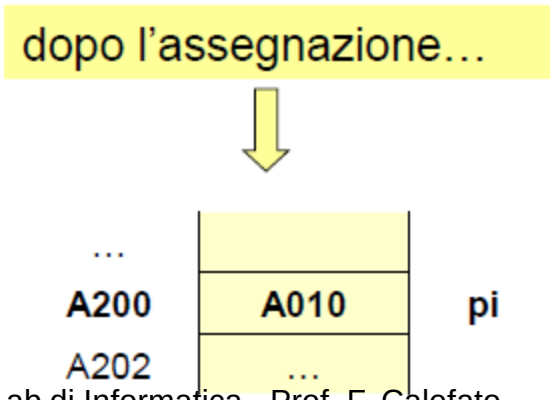
Sono tutte scritture equivalenti!

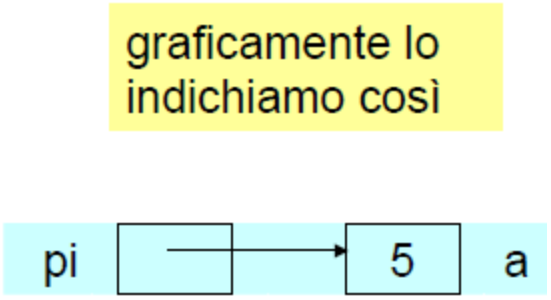
NOTA:
È meglio avere dichiarazioni su righe distinte per tipi diversi

Inizializzazione puntatori

- Una variabile puntatore può essere inizializzata usando l'operatore di indirizzo
- *Esempio:* ***pi = &a;***
  - il valore di pi viene posto pari all'indirizzo di a
  - ovvero, pi **punta ad a**
  - ovvero, a è l'**oggetto puntato da pi**







Operatore di dereferenziamento

- Operatore di dereferenziamento *
- Applicato ad una variabile puntatore fa riferimento all'oggetto puntato
- *Esempio:*
  - **int *pi;** // dichiarazione di un puntatore ad intero
  - **int a = 5, b;**
  - **pi = &a;** // pi punta ad a => *pi è un altro modo di denotare a
  - **b = *pi;** // assegna a b il valore della variabile puntata
// da pi, ovvero il valore di a, ovvero 5
  - ***pi = 9;** // assegna 9 alla variabile puntata da pi, ovvero ad a */

  - N.B. Se **pi è di tipo int *, allora *pi è di tipo int**
  - ***pi** è spesso letto come **"il contenuto di pi"**

Operatori di dereferenziamento * e di indirizzo &

- **“*” è associativo a destra**
  - **p è equivalente a *(*p)
- **“&” può essere applicato solo a una variabile**
  - &a non è una variabile quindi &(&a) **è sbagliato**

- **“*” e “&” sono uno l’inverso dell’altro**
- data la dichiarazione int a;
  - *&a è un alias per a
- data la dichiarazione int *pi;
  - &*pi ha valore uguale a pi

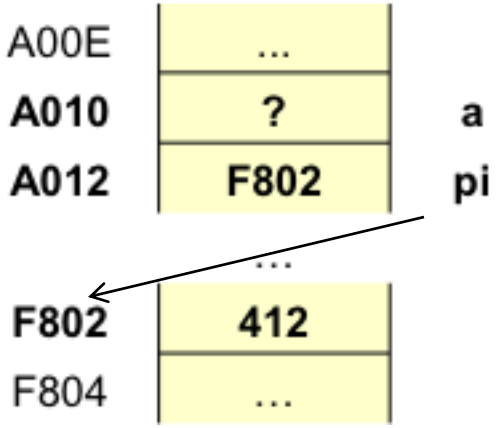
I due usi di *

- **ATTENZIONE:**
- Non bisogna confondere le due occorrenze di *
  - * in una dichiarazione serve per dichiarare una variabile di tipo puntatore
    - Es.: int *pi;
  - * in una espressione è l'operatore di dereferenziamento
    - Es.: b = *pi;

Inizializzazione di puntatori

- I puntatori (come tutte le altre variabili) devono essere inizializzati prima di poter essere usati
- È un errore dereferenziare una variabile puntatore non inizializzata che puo' dare origine a un errore a run-time
- Esempio:

```
int a;
int * pi; // = NULL!!
a = *pi;
*pi = 500;
```



ad a viene assegnato il valore 412

Scrive 500 nella cella memoria di indirizzo F802
… ma non sappiamo a cosa corrisponde questa cella di memoria!
… la memoria puo essere corrotta!
… **ERRORE** a run-time!

Tipi di variabili puntatore

- **Ricordate!**
- Il tipo di una variabile puntatore è **puntatore a tipo**
  - char * pc; // variabile ptr di tipo puntatore a char
  - int * pi; // variabile ptr di tipo puntatore a int
  - float * pf; // variabile ptr di tipo puntatore a float
  - …
- I tipi puntatore sono indirizzi (non interi, nè caratteri, float etc…)
  - Cioè il loro valore è SEMPRE un indirizzo

Tipi di variabili puntatore

- Due variabili puntatore a tipi diversi non sono compatibili tra loro
- Esempio:
  - int x;
  - int *pi;
  - float *pf;

  - x = pi; // assegnazione di **int*** a **int**
  - pf = x; // assegnazione di **int** a **float***
  - pi = pf; // assegnazione **float*** a **int***

Esercizio 1

1. Realizzare un programma che definisca e inizializzi due variabili di tipo intero (i1e i2) e due variabili di tipo float (f1e f2)
2. Stampare i valori delle variabili i1, i2, f1e f2
3. Definire ed inizializzare un puntatore per ciascuna di queste variabili (siano pi1, pi2, pf1, pf2), associando i puntatori alla corrispondente variabile
4. Stampare attraverso i puntatori i valori delle variabili i1, i2, f1e f2
5. Cambiare attraverso i puntatori i valori delle variabili i1, i2, f1e f2
6. Stampare (senza usare i puntatori) i valori delle variabili i1, i2, f1e f2
7. Stampare i valori delle variabili puntate da pi1, pi2, pf1, pf2
8. Stampare gli indirizzi esadecimali di ciascuna delle variabili puntatore (printf con %p)

PARTE 2

Passaggio di valori

- In C i parametri delle funzioni sono passati per valore
  - Il parametro formale è una nuova variabile locale alla funzione
  - Al momento dell'attivazione il valore del parametro attuale è copiato nel parametro formale
- Perché per valore?
  - E' il metodo più sicuro per evitare "modifiche accidentali" alle variabili
- A volte non è sufficiente però…
  - Occorre usare il **passaggio dei parametri per riferimento**
  - **In C non è disponibile** ed è dunque necessario **simularlo** tramite il **passaggio di variabili puntatore**, detto **passaggio per indirizzo**

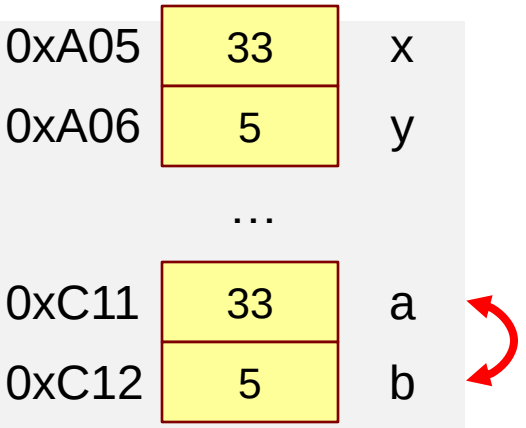
Procedura di scambio tra due variabili con passaggio per valore

```
void scambia(int a, int b) {
      int t;
      t = a;
      a = b;
      b = t;
}

main() {
      int x = 33, y = 5;
      scambia(x, y);
      printf("x = %d, y = %d\n", x, y);
}
```

Parametri formali

Stampa sempre x = 33, y = 5

Parametri attuali



Procedura di scambio tra due variabili con passaggio per indirizzo

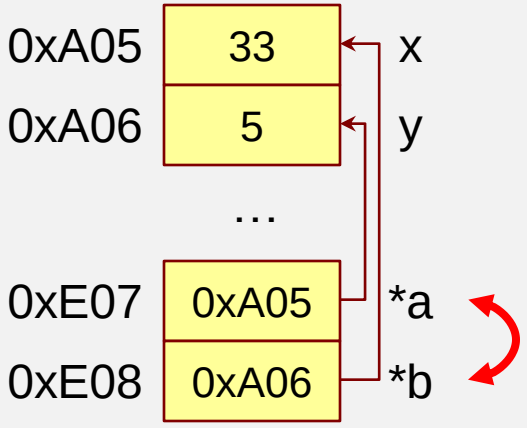
```
void scambia(int* a, int* b)
{
      int t;
      t = *a;
      *a = *b;
      *b = t;
}

main()
{
      int x=33, y=5;
      scambia(&x, &y);
      printf("x = %d, y = %d\n", x, y);
}
```

Un puntatore è una variabile destinata a contenere un indirizzo

***** operatore di dereferenziamento
***a** cioè il valore contenuto dall'indirizzo di memoria puntato da a

Stampa
x = 5, y = 33

& operatore estrazione di indirizzo



Scambio ***a** con ***b** cioè i valori negli indirizzi di memoria puntato da a e b

Esercizio 2

- Creare la seguente procedura di scambio
- *void scambia_caratteri(char * c1, char * c2, char * c3, char * c4)*
- che scambi c1 con c4 e c2 con c3

Esercizio 3

- Calcolare il fattoriale mediante una procedura con passaggio per indirizzo
- Dimostrare il corretto funzionamento mediante casi di test opportuni in CUnit

Esercizio: calcolo del fattoriale mediante funzione o procedura

Funzione

```
int fattoriale(int num)
{
        int i, fatt = 1;
        i=num;
        while(i>1) {
             fatt = fatt * i;
             i--;
             }
        return fatt;
}
main()
{
        int f;
        f = fattoriale(15);
        printf("15! = %d", f);
}
```

Procedura

```
void fattoriale(int num, int *fatt)
{
      int i;
      *fatt = 1;
            i=num;
        while(i>1) {
            *fatt = fatt * i;
            i--;
            }
}
main()
{
int f;
fattoriale(15, &f);
printf("15! = %d", f);
}
```

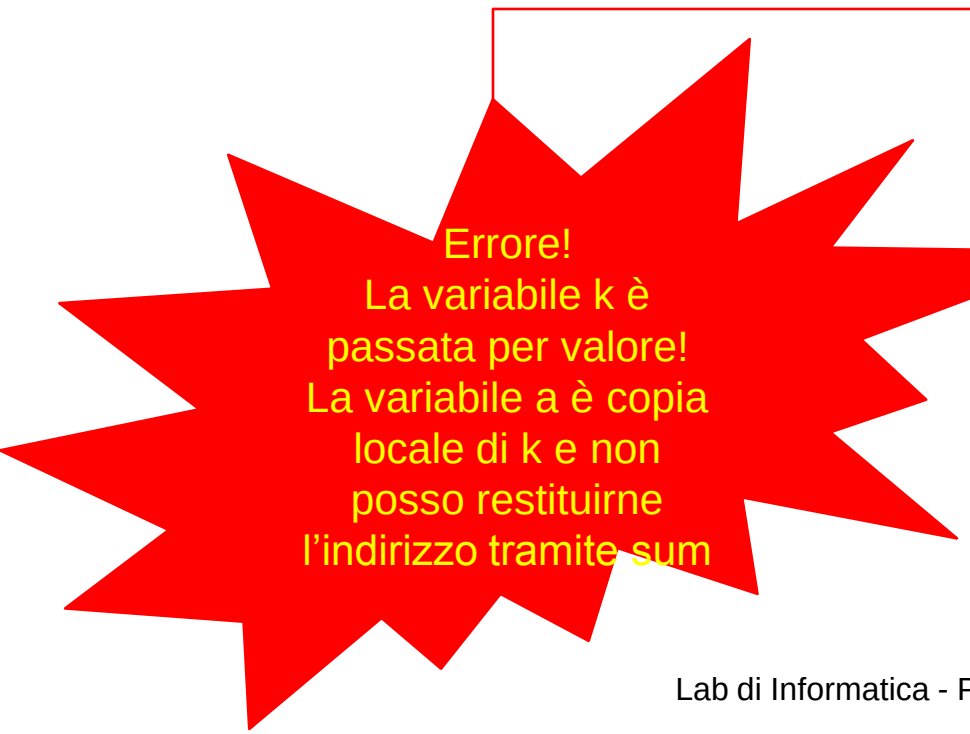
Esercizio 4 (assegnazione per casa)

- Trasformare in procedure tutte le altre funzioni matematiche realizzate nel modulo precedente, usando il passaggio per indirizzo
- Aggiornate i casi di test di conseguenza (create un altro progetto, non sovrascrivete quello esistente)

Domanda: E' ammissibile una funzione del genere?

```
int sum, k, z;
...
somma(k, z, &sum);
…
```

```
void somma(int a, int b, int* sum)
{
int i=0;
sum = &a;
if(b<0) {
        while (i>b) {
                (*sum)--;
                i--;
                }
        }
else {
        i=0;
        while (i<b) {
                (*sum)++;
                i++;
                }
        }
}
```



Puntatori e funzioni

- L'uso del passaggio per indirizzo con le funzioni permette di ottenere una funzione che "restituisca" più valori e non solo uno
- Es.
  - int scambia(int* a, int* b)
  - Funzione che scambia a con b e restituisce la differenza a - b

Domanda

- E’ ammissibile una funzione del genere?
- Somma a con b in una nuova variabile locale sum e ne restituisce l’indirizzo

```
int* somma (int a, int b) {
    int sum = a+b;
    return &sum;
}
```

- Presto o tardi la locazione di memoria di sum servirà daccapo e sarà sovrascritta

Soluzione: usare la memoria dinamica

- La memoria dinamica è allocata tramite funzione malloc()
  - Sfrutta una porzione di memoria apposita chiamata “heap”
- Una variabile allocata dinamicamente può essere deallocata solo dal programmatore tramite funzione free()

malloc() e free() in stdlib.h

- void *malloc(size)
  - Alloca size byte di memoria
- E se non voglio ricordare quanti byte occupa un char, un int, etc.?
  - Si usa sizeof()
- Es.
  - int* i = (int*) malloc(sizeof(int));
- void free(void *ptr)
  - Libera la memoria puntata da ptr
- Es.
  - free(i);

Soluzione alla domanda

Codice chiamante

```
#include <stdlib.h>
…
int a, b;
int* c;
c = somma(a,b);
…
free(c);
…
```

Codice chiamato

```
#include <stdlib.h>
…
int* somma (int a, int b) {
    int* sum = NULL;
    sum = (int*)
            malloc(sizeof(int));
    if (sum != NULL)
        *sum = a+b;
    return sum;
}
```

Esercizio 5

- Scrivere le procedure di differenza e prodotto con passaggio per indirizzo e memoria dinamica (malloc e free)
- Dimostrarne la correttezza tramite opportuni casi di test

scanf() in <stdio.h>

- int scanf (char* format, ...);
- Funzione che legge dati dallo standard input secondo il formato specificato
- Restituisce il numero di elementi (%) effettivamente letti
- Es.

  int n;

  scanf("%d", &n);
  - Legge un intero e lo salva nella locazione di memoria di n
  - %d è la direttiva per indicare che ci si attende un intero decimale (numero)
  - Restituisce 1 se si digita un valore e si preme invio

scanf()

- Es.

```
…
 while (scanf("%d %d", &n, &m) != 2)
…
```

  - Se si preme invio senza aver digitato un numero scanf restituisce 0
  - Se si preme invio dopo aver digitato un solo numero scanf restituisce 1
  - Il ciclo non termina se si preme invio senza aver digitato due numeri separati da uno spazio

scanf()

- Direttive per i tipi primitivi più comuni
  - '%c' : Legge un carattere
  - '%f' : Legge un floating-point
  - '%d' : Legge un intero decimale con segno
  - '%u' : Legge un intero decimale senza segno
  - …

- NB.
  - Non imparateli tutti a memoria, soli i più comuni
  - Gli altri li trovate sui manuali secondo necessità

Bibliografia

1. Davide Patti, “Puntatori in C”
   http://www.diit.unict.it/~dpatti/FONDINF-LAB/08-C-Puntatori.pdf