Programmazione

Giovanni Da San Martino

Dipartimento of Matematica, Università degli Studi di Padova
giovanni.dasanmartino@unipd.it
A.A. 2021-2022

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Previuosly on Programmazione

- Gli operatori aritmetici ed i comandi sono definiti tra termini dello stesso tipo
- Ma il C effettua automaticamente alcune conversioni tra tipi, le promozioni: (ovvero presi due elementi nella tabella a fianco, la conversione avviene a quello più in alto)
- Il C permette di forzare una conversione
- int x = (int) 3 + 3.9;
- printf("%d", x); // stampa 6!
- Si rischia di perdere informazione, ovvero il C non è type-safe

long double
double
float
long
int , unsigned int
short, char

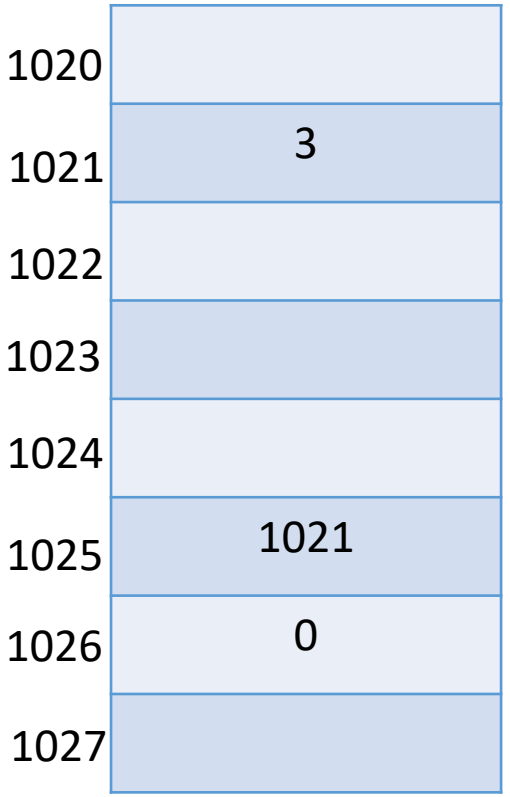
Puntatori

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Definizione

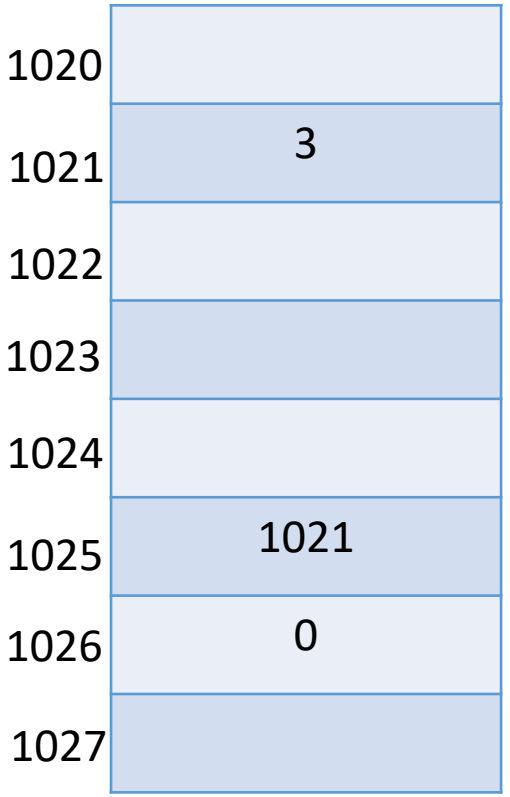
- Gli indirizzi di memoria (l-valori) sono interi positivi.
- Un puntatore è una variabile il cui r-valore è un l-valore
- Es. int x = 3; //l-valore = 1021, r-valore = 3

  la variabile con l-valore 1025 potrebbe essere un puntatore (vedi figura)
- I puntatori sono uno strumento di basso livello: ci permettono di manipolare altre variabili (altre celle di memoria)



Dichiarazione

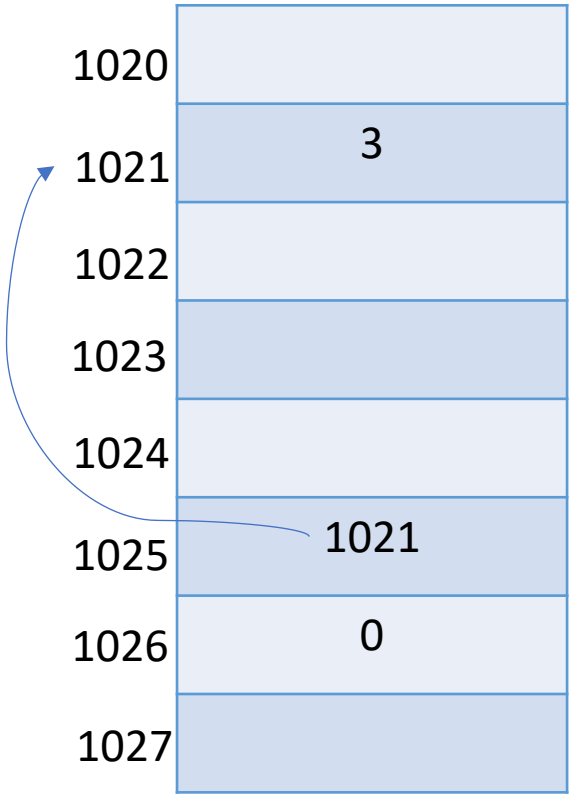
- Dichiarazione: tipo *nome;
- Es. int *ptr = NULL; // l-valore = 1026
- ptr è una variabile di tipo puntatore ad una variabile di tipo intero.



- Per ogni Tipo di variabile, esiste il corrispondente tipo “puntatore a Tipo”
  - float *x; char *c; long *l, ...
  - Perché?
- Esistono pure puntatori a funzioni

Operatori unari su variabili: & *

- &x (dove x è necessariamente una variabile) restituisce l'l-valore di x
- *p restituisce la variabile indicata dal r-valore del puntatore p
- & e *
  - hanno precedenza minore di () [], ma maggiore degli operatori aritmetici
  - sono associativi da destra a sinistra



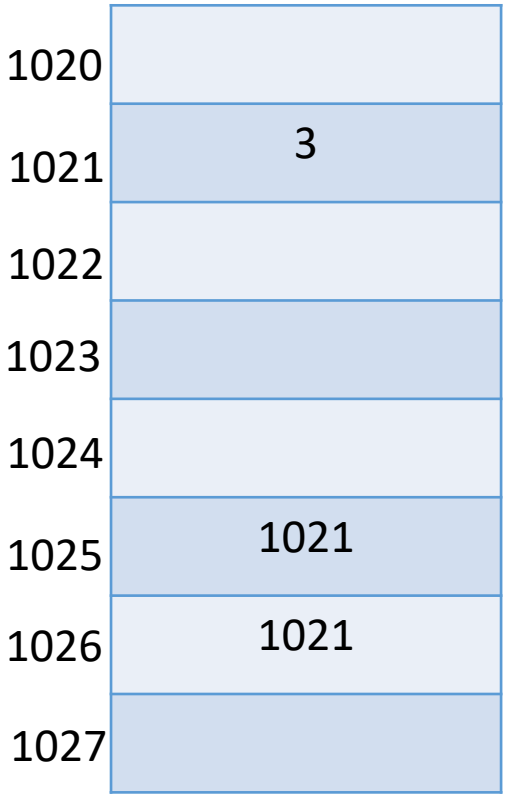

```
int *xptr = &x; // l-valore di xptr = 1025
printf("%p,%d,%p", xptr, *xptr, &x);
// stampa 1021, 3, 1021
```

Esempi


```
int x = 3;
int *xp;   // l-valore=1025; r-valore=indefinito
xp = &x; // r-valore=1021

*xp = 4; printf("%d",x); // stampa 4
*xp = *xp+1; printf("%d",x); // stampa 5

int *xp2 = &x; //ok, ad xp2 si assegna un l-valore
                //equivale a int *xp2; xp2 = &x;
int *xp3 = x; // errore di tipo
```



Esempi

int x = 3; int *xp2;

*xp2 = x; // xp2 è stata dichiarata ma non inizializzata, per cui l' r-valore di xp2, *xp2, punta ad una cella di memoria "casuale" nella quale cerchiamo di scrivere un intero, perciò ci aspettiamo (e speriamo in) un errore di **Segmentation Fault**.

- Come evitare il problema?

```
int *xp2 = NULL; //equivale a int *xp2 = 0, l-valore=1026
if (!xp2) {
        // puntatore non inizializzato
}
```

Variabili – Manuale d'Uso

1. Non usare una variabile non inizializzata.
2. Controllare che il tipo della variabile sia corretto per l'operazione dove si sta per usare.
3. &* sono associativi da destra &*p = &(*(p)).
4. Memorizzare alla perfezione il significato degli operatori: per esempio * restituisce la variabile (un oggetto che ha un nome, tipo, l-valore, r-valore) non il suo r-valore.
5. * è usato con due significati diversi 1) nella dichiarazione e 2) nel corpo di una funzione.

Puntatori ed Espressioni

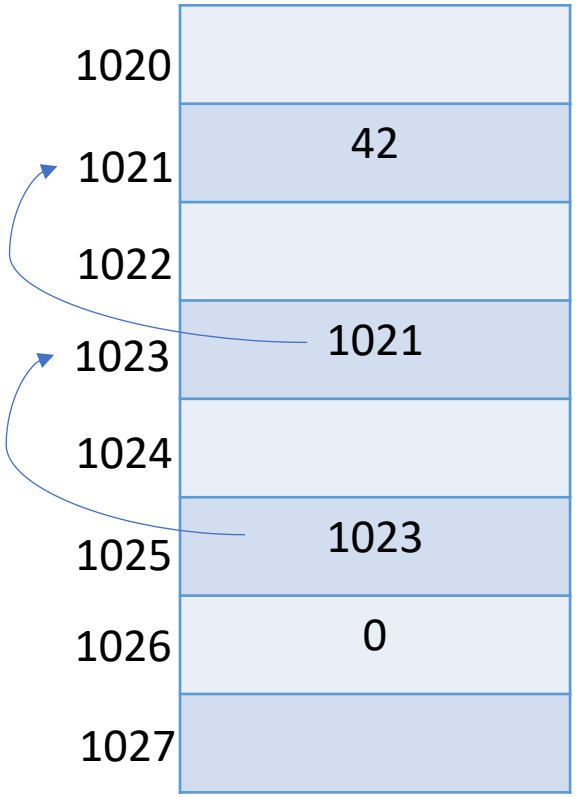
- Se un puntatore è stato dichiarato ed inizializzato, può essere utilizzato in un espressione

```
int y, x=3, *p1=&x, *p2=*p1;
y = *p1 * *p2; //  (*p1) * (*p2)
y = x + *p2;
y = 5* - *p2/ *p1; // se non ci fosse stato lo spazio tra / e *?
```

Puntatori a Puntatori a Puntatori a....

- Si può definire un puntatore ad una variabile puntatore utilizzando più volte il simbolo * nella dichiarazione

```
int x, *p1;
int **p2; // p2 è un puntatore ad una variabile
          //puntatore a variabile intera. L-valore: 1025
x = 42; p1 = &x;
p2 = &p1;
printf("%d", **p2);
```



Funzioni: Passaggio di Parametri

- Passaggio per valore: i valori dei parametri attuali vengono assegnati ai parametri formali al momento dell'invocazione della funzione.
  - La funzione (f) non modifica le variabili parametri attuali (y)

```
void f(int x) {
        x = x + 1;
}

int y = 2;
f(y);
printf("%d", x); // stampa 2
```

Passaggio per Riferimento

- Il passaggio per riferimento permette di modificare all'interno della funzione i parametri attuali (y)
- In C è implementato solamente il passaggio per valore
- Il passaggio per riferimento è ottenuto

```
void f(int x) {
        x = x + 1;
}

int y = 2;
f(y)
printf("%d", x); // stampa 2
```

Passaggio per Riferimento: Esempi

- Scambio di variabili

```
void scambio(int *x, int *y) {
        int temp = *x;
        *x = *y;
        *y = temp;
}
int main() {
        int x=2, y=5;
        scambio(&x, &y);
        printf("x=%d, y=%d\n", x,y); //stampa x=5, y=2
}
```

Array e Puntatori

- int x[] = {1,2,3,4,5}
- x è un puntatore **costante** al primo elemento dell'array:
- x == &x[0]
- int *p = x; // corretto, equivale a &x[0]

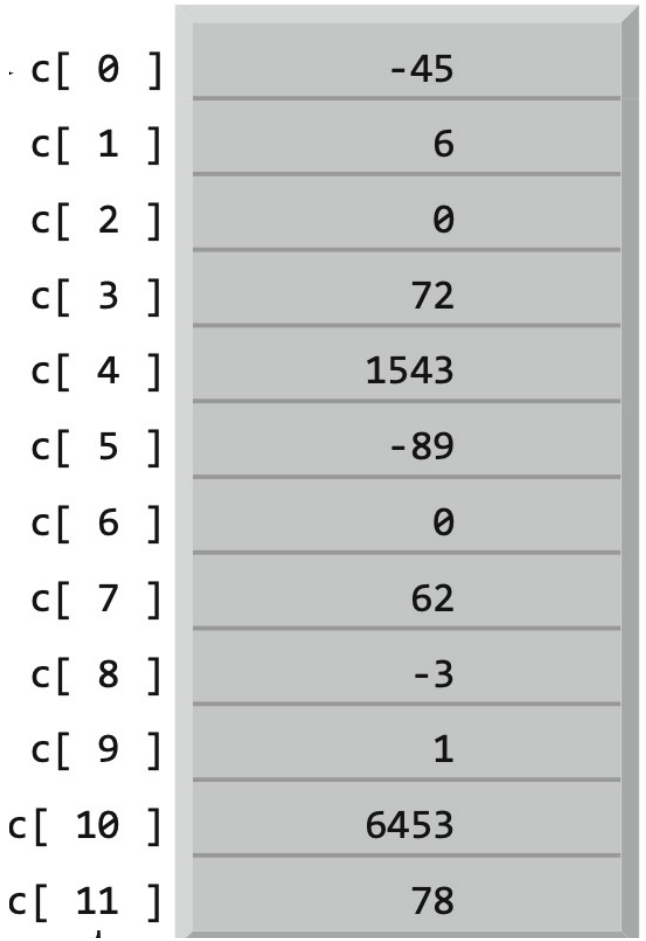
//assumendo che l-valore di x sia 1022

&x[0] == p == 1022

&x[1] == p+1 == 1022 + 1*sizeof(int) == 1026

&x[2] == p+2 == 1022 + 2*sizeof(int) == 1030

&x[3] == p+3 == 1022 + 3*sizeof(int) == 1034



Aritmetica dei Puntatori

- Si può sommare o sottrarre un intero ad un puntatore
  - int x, *p = &x; p+2 equivale a p+2*sizeof(int)
- Si può sottrarre due puntatori (ha senso se si riferiscono allo stesso array)
  - p1-p2
- Si possono confrontare due puntatori:
  - p1>p2, p1==p2, p1 != p2
- Non è ammesso utilizzare puntatori in moltiplicazioni o divisioni
  - p1/3, p1/p2, p1*p2
- Non è ammesso sommare due puntatori
  - p1+p2

Esempio: Stampa Array


```
int a[] = {1,2,3,4};
int i = 0, *p = a;
    while(i<4) {
        printf("%d\n", *p);
        i+=1;
        p = p+1;
    }
```

Stampa:

1

2

3

4

Esempio: Stampa Array (versioni 2 - 3)


```
int a[] = {1,2,3,4};
int i = 0, *p = a;
while(i<4) {
        printf("%d\n", *(p+i));
        i+=1;
}
// Oppure
while(i<4) {
        printf("%d\n", *(a+i));
        i+=1;
} //ma a+=i sarebbe sbagliato!
```

Entrambi Stampano:

1

2

3

4

Array Come Parametri di Funzione

- Quando passiamo un array ad una funzione, in realtà passiamo il puntatore al primo elemento dell'array
- In C gli array sono sempre passati per riferimento

```
void successivo(int array[], int N) {
    for(int i=0; i<N; i+=1) {
        array[i] += 1;
    }
}
```

```
int main (void) {
        int a[]={1,2,3};
        successivo(a,3);
        //a={2,3,4}
}
```

Dangling Reference

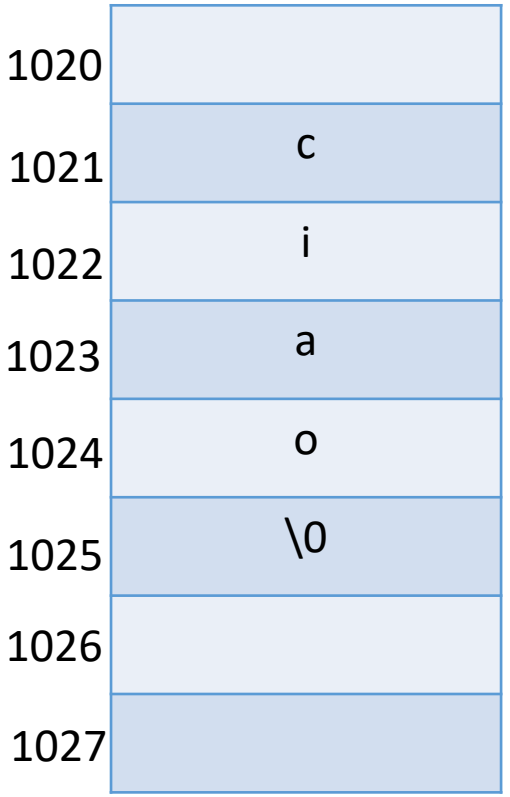
- Una funzione può restituire un puntatore
- Bisogna stare però attenti: le variabili create all'interno della funzione vengono deallocate (distrutte) al termine dell'esecuzione
- anche min viene deallocata (il contenuto della cella di memoria potrebbe essere utilizzato per creare altre variabili)

```
int a[] = {1,2,3,4}
p = min(a, 4);
printf("\n%d\n", *p);
```

```
int * min(int*X, int dimX) {
    int i=1, min=X[0];
    while(i < dimX) {
        if(X[i] < min)
            min=X[i];
        i=i+1;
    }
    return &min;
}
```

Stringhe (Sequence di Caratteri)

- Una sequenza di caratteri viene anche chiamata stringa. Es.
- char string1[] = ”ciao";
- In memoria viene memorizzato come “ciao\0”;
- il carattere \0 indica la fine della stringa
- il primo elemento di string1 è un char *



Variabili Globali

- Variabili definite fuori da ogni funzione sono dette globali, sono visibili in tutto il file da dove sono definite in poi (a meno che non si definisca una variabile con lo stesso nome in una funzione – in questo caso la variabile globale non è visibile nella funzione).
- Le variabili globali sono da in generale da evitare (ogni funzione dovrebbe modificare solamente variabili locali e parametri). Si usano nel caso una variabile debba essere usata in molte funzioni e si debba mantenerne il valore tra ogni chiamata

```
#include <stdio.h>
int x=10;


int main(void) {
        //int x = 8;
        printf("%d\n", x); //se tolgo il commento all'istruzione precedente stampa 8
}
```