C: Puntatori

Elementi di Informatica 2018/2019

Puntatori

Dichiarazione di variabile: `int n;`

- nome
- locazione di memoria
- indirizzo della locazione di memoria

Puntatori

L'operatore `&` (vedi utilizzo `scanf`) restituisce l'indirizzo di memoria di una variabile

I **puntatori** sono delle variabili alle quali può essere assegnato un indirizzo di memoria

Puntatori: dichiarazione

```
int *int_pointer;
```

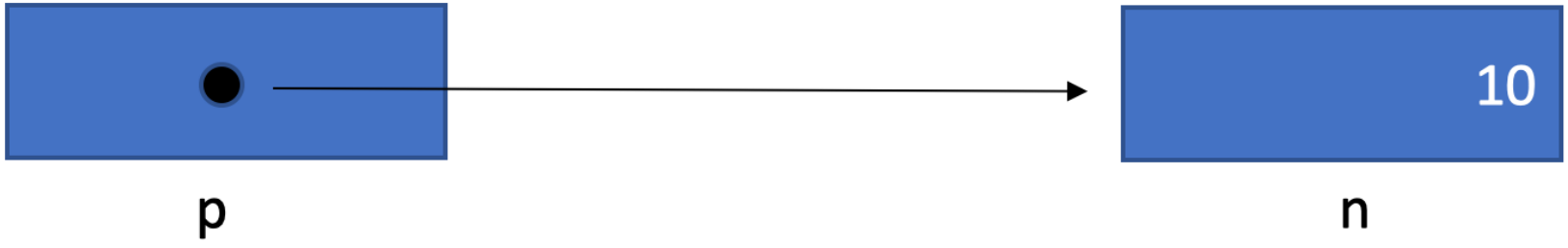
Occorre specificare il tipo base (in questo caso `int`) e il nome della variabile (in questo case `int_pointer`) preceduto dal simbolo `*`

Puntatori: inizializzazione

```
int n=10;
int *p;

p=&n; //non assegna a p il valore di n!
```

In questo caso caso la variabile puntatore ad intero `p` è inizializzata con l'indirizzo di `n`



Puntatori: primo utilizzo

```c
#include <stdio.h>

int main()
{
  int n=10;
  int *p;

  p=&n;
  printf("%d %d",n,*p);
}
```

L'operatore `*` (detto di *indirezione*) applicato ad una variabile puntatore permette di restituire il valore contenuto nella variabile puntata

Puntatori: esempio

```c
#include <stdio.h>

int main()
{
  int n=10,m;
  int *p;

  p=&n;
  m=*p;
  n=20;
  printf("%d %d %d",n,*p,m); //stampa: 20 20 10
}
```

Puntatori: esempio

```c
#include <stdio.h>

int main()
{
  int n=10,m;
  int *p;

  p=&n;
  m=*p;
  n=20;
  printf("%d %d %d",n,*p,m); //stampa: 20 20 10

  *p=30;
  printf("%d %d %d",n,*p,m); //stampa: 30 30 10
}
```

Puntatori: esempio

```c
#include <stdio.h>

int main()
{
  int n=10,m;
  int *p,*q;

  p=&n;
  m=*p;
  n=20;
  printf("%d %d %d",n,*p,m); //stampa: 20 20 10

  *p=30;
  printf("%d %d %d",n,*p,m); //stampa: 30 30 10

  q=p;
  printf("%d %d %d",n,*p,*q); //stampa 30 30 30
}
```

Esercizio.

Scrivere un programma che esegua la somma di due numeri (letti da tastiera) usando i puntatori

Soluzione.

```c
#include <stdio.h>
int main() {

  int n1,n2;
  int *p1,*p2;

  printf("Inserisci un numero: ");
  scanf("%d",&n1);
  printf("Inserisci un altro numero: ");
  scanf("%d",&n2);

  p1=&n1;
  p2=&n2;

  printf("La somma vale %d", *p1 + *p2);
}
```

Esercizio.

Scrivere un programma che definisca il maggiore tra due numeri (letti da tastiera) usando i puntatori

Soluzione.

```c
#include <stdio.h>
int main() {

  int n1,n2,max;
  int *p1,*p2;

  printf("Inserisci un numero: ");
  scanf("%d",&n1);
  printf("Inserisci un altro numero: ");
  scanf("%d",&n2);

  p1=&n1;
  p2=&n2;

  if(*p1>*p2) max=*p1;
  else max=*p2;
  printf("Il valore maggiore e' %d", max);
}
```

Puntatori a strutture

```c
#include <stdio.h>
#define NOME 50

int main() {
    struct anagr
    {
        int matricola;
        char nome[NOME];
        char cognome[NOME];
    };

    struct anagr studente;
    struct anagr *pointer;

    pointer=&studente;
}
```

Puntatori a strutture

```
    printf("\nNome studente: ");
    scanf("%s",studente.nome);
    printf("\nCognome studente: ");
    scanf("%s",studente.cognome);
    printf("\nMatricola: ");
    scanf("%d",&studente.matricola);

    printf("\n\nDati studente: ");
    printf("%s %s - matricola %d\n",(*pointer).nome,
    (*pointer).cognome,(*pointer).matricola);
}
```

Puntatori a strutture

In alternativa: `pointer->nome` ...

```
    printf("\nNome studente: ");
    scanf("%s",studente.nome);
    printf("\nCognome studente: ");
    scanf("%s",studente.cognome);
    printf("\nMatricola: ");
    scanf("%d",&studente.matricola);

    printf("\n\nDati studente: ");
    printf("%s %s - matricola %d\n",pointer->nome,
    pointer->cognome,pointer->matricola);
}
```

Puntatori a strutture

E ovviamente...

```
    pointer=&studente;

    printf("\nNome studente: ");
    scanf("%s",pointer->nome);
    printf("\nCognome studente: ");
    scanf("%s",pointer->cognome);
    printf("\nMatricola: ");
    scanf("%d",&pointer->matricola);

    printf("\n\nDati studente: ");
    printf("%s %s - matricola %d\n",studente.nome,
    studente.cognome,studente.matricola);
}
```

Strutture contenenti puntatori

```c
#include <stdio.h>
int main() {
    struct st
    {
        int *p1;
        int *p2;
    };

    struct st st_pointers;
    int n1=10, n2;

    st_pointers.p1=&n1;
    st_pointers.p2=&n2;
    *st_pointers.p1=20;
    *st_pointers.p2=*st_pointers.p1 * 2;
    printf("%d %d",n1,n2);
}
```

Cosa stampa a video questo programma?

Liste concatenate lineari

Le liste lineari concatenate sono costituite da *serie di elementi omogenei* che occupano in memoria posizioni non adiacenti

Puntatori a strutture e *strutture contenenti puntatori* sono di fondamentale importanza per la creazione e gestione delle liste concatenate lineari

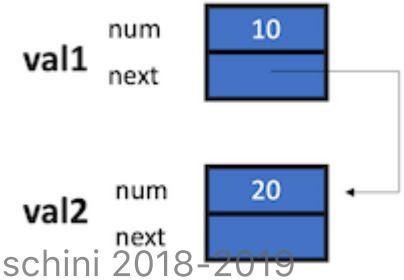
Liste concatenate lineari (statiche)

```c
#include <stdio.h>
int main() {
    struct lista_valori
    {
        int num;
        struct lista_valori *next;
    };
    struct lista_valori val1,val2;

    val1.num=10;
    val2.num=20;

    val1.next=&val2;

    printf("%d",val1.next->num);
}
```



Liste concatenate lineari

```c
#include <stdio.h>
int main() {
    struct lista_valori
    {
        int num;
        struct lista_valori *next;
    };
    struct lista_valori val1,val2,val3;

    val1.num=10;
    val2.num=20;
    val3.num=30;

    val1.next=&val2;
    val2.next=&val3;

    printf("%d ",val1.next->num);
    printf("%d ",val2.next->num);
}
```

Liste concatenate lineari

Alcune interessanti proprietà:

- eliminare elementi
- inserire elementi

Eliminare `val2` dalla lista:

```
val1.next = val2.next;
```

Inserire nuovo elemento (`val4`) tra `val2` e `val3` :

```
val4.next = val2.next;
val2.next = &val4;
```

Questo è possibile grazie al fatto che gli elementi di una lista non sono memorizzati in sequenza. *Cosa succede negli array?*

Liste concatenate lineari

- Puntatore esterno al primo elemento della lista
  - `struct lista_valori *punt_lista = &val1;`
- Puntatore a NULL (fine lista)
  - `val3.next = NULL;`

Esempio: scorri lista

```c
#include <stdio.h>
int main() {
    struct lista_valori {
        int num;
        struct lista_valori *next;
    };
    struct lista_valori val1,val2,val3;
    struct lista_valori *punt_lista;

    val1.num=10;
    val1.next=&val2;
    val2.num=20;
    val2.next=&val3;
    val3.num=30;
    val3.next=NULL;
    punt_lista=&val1;

    while(punt_lista != NULL) {
      printf("%d ",punt_lista->num);
      punt_lista=punt_lista->next;
    }
}
```

Liste concatenate lineari: NOTE

```
struct valori *punt_lista=&n1;
```

oppure

```
struct valori *punt_lista;
punt_lista=&n1;
```

Liste concatenate lineari: NOTE

- Liste statiche
- Liste dinamiche

Il programma ha necessità di allocare la memoria per ogni nuovo elemento della lista!

Allocazione dinamica della memoria

Puntatori e Array

Puntatori e array sono strettamente correlati

E’ possibile definire un puntatore per accedere agli elementi di un array

Puntatori e Array

Il nome di un array rappresenta un puntatore *costante* al suo primo elemento: punta sempre al primo elemento e non è possibile assegnare l'indirizzo di un'altra cella di memoria

```
int vett[N];
int *vett_ptr;

vett_ptr = &vett[0];
```

equivale a

```
vett_ptr = vett; //non compare &
```

E' quindi possibile accedere agli elementi di un array utilizzando un puntatore.

Puntatori e Array

```c
#include <stdio.h>
int main() {

  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]); //1 2 3

  vett_ptr=vett; //vett_ptr punta dove punta vett
  //vett_ptr = &vett[0];
  *(vett_ptr+1)=10;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]); //cosa stampa?
}
```

Puntatori e Array

```c
#include <stdio.h>
int main() {

  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]); //1 2 3

  vett_ptr=vett; //vett_ptr punta dove punta vett
  *(vett_ptr+1)=10;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]); //stampa 1 10 3
}
```

Puntatori e Array

Avrei potuto scrivere anche...

```c
#include <stdio.h>
int main() {

  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]);

  vett_ptr=vett;
  *(vett_ptr+1)=10;

  for(i=0;i<3;i++)
    printf("%d ",*(vett_ptr+i)); //stampa 1 10 3
}
```

Puntatori e Array

Ma anche...

```c
#include <stdio.h>
int main() {

  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]);

  vett_ptr=vett;
  *(vett_ptr+1)=10;

  for(i=0;i<3;i++)
    printf("%d ",*(vett_ptr++));
}
```

Puntatori e Array

Domanda...

Che differenza c'è tra:

```
   for(i=0;i<3;i++)
     printf("%d ",*(vett_ptr++));
}
```

e

```
   for(i=0;i<3;i++)
     printf("%d ",*(vett_ptr+i));
}
```

Puntatori e Array

Risposta

Provate a stampare `*vett_ptr` dopo il primo e dopo il secondo ciclo `for`!

```
  for(i=0;i<3;i++)
    printf("%d ",*(vett_ptr++));
}
```

e

```
  for(i=0;i<3;i++)
    printf("%d ",*(vett_ptr+i));
}
```

Aritmetica dei puntatori

Incremento: permette di passare ad un indirizzo successivo

`p++`, `p+1`, `p+i`

Decremento: permette di passare ad un indirizzo precedente

`p--`, `p-1`, `p-i`

Lo spostamento dipende dal **tipo base**!

Puntatori e Array: ATTENZIONE

Differenza tra `vett_ptr++` e `++vett_ptr`

Puntatori e Array

Ricapitolando...

```c
#include <stdio.h>
int main() {
  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]);

  vett_ptr=vett;

  for(i=0;i<3;i++)
    printf("%d ",*(vett+i));

  for(i=0;i<3;i++)
    printf("%d ",*(vett_ptr+i));
}
```

... sono equivalenti!

Puntatori e Array

Ma attenzione...

```c
#include <stdio.h>
int main() {
  int i,vett[3]={1,2,3};

  for(i=0;i<3;i++)
    printf("%d ",*(vett+i));

  for(i=0;i<3;i++)
    printf("%d ",*(vett++)); //ERRORE!!!
}
```

... `vett` è una costante!

Esercizio

Scrivere un programma in C che, utilizzando i puntatori, carichi N valori interi (letti da tastiera) in un array e successivamente stampi i valori inseriti.

Soluzione

Senza puntatori...

```c
#include <stdio.h>
#define DIM 10
int main() {

  int i,n,vett[DIM],*vett_ptr;

  vett_ptr=vett;
  do {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci i %d elementi:\n",n);
  for(i=0;i<n;i++) {
      printf("elementodi indice - %d : ",i);
      scanf("%d",&vett[i]);
  }
```

Soluzione

Con i puntatori...

```c
#include <stdio.h>
#define DIM 10
int main() {

  int i,n,vett[DIM],*vett_ptr;

  vett_ptr=vett;
  do {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci i %d elementi:\n",n);
  for(i=0;i<n;i++) {
      printf("elementodi indice - %d : ",i);
      scanf("%d",vett_ptr+i);
  }
```

Soluzione

Ma anche...

```c
#include <stdio.h>
#define DIM 10
int main() {

  int i,n,vett[DIM],*vett_ptr;

  vett_ptr=vett;
  do {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci i %d elementi:\n",n);
  for(i=0;i<n;i++) {
      printf("elementodi indice - %d : ",i);
      scanf("%d",vett+i); //non sto cambiando vett!
  }
```

Soluzione

Senza puntatori...

```
  for(i=0;i<n;i++)
    printf("%d ",vett[i]);
```

Soluzione

Con i puntatori...

```
  for(i=0;i<n;i++)
    printf("%d ",*(vett+i));
}
```

```
  for(i=0;i<n;i++)
    printf("%d ",*(vett_ptr+i));
}
```

```
  for(i=0;i<n;i++)
    printf("%d ",*(vett_ptr++));
}
```

Esercizio

Scrivere un programma in C che calcoli la somma degli elementi di un array usando i puntatori

Soluzione

```c
#include <stdio.h>
#define DIM 10
int main() {

  int i,n,vett[DIM],*vett_ptr,somma=0;

  vett_ptr=vett;
  do {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("\nInserisci i %d elementi: \n",n);
  for(i=0;i<n;i++) {
      printf("\nelementodi indice - %d : ",i);
      scanf("%d",vett+i);
  }
```

Soluzione

```
   printf("\nCalcolo la somma...");
   for(i=0;i<n;i++) {
       somma += *vett_ptr;
       vett_ptr++;
   }
   printf("\nLa somma vale: %d\n",somma);
}
```