

I puntatori in C++

1

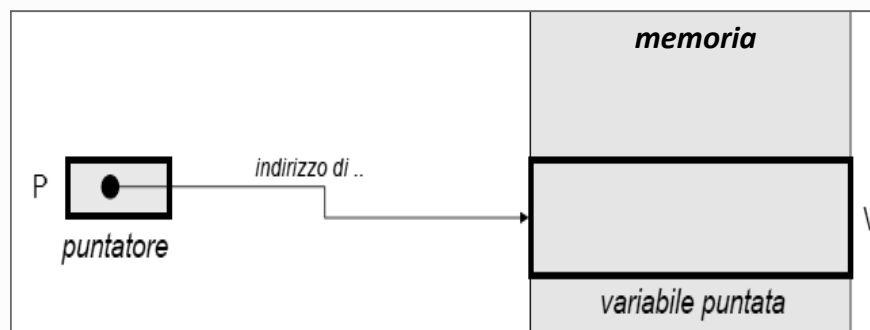
I puntatori in C++

Il **puntatore** è un tipo di dato scalare, che consente di rappresentare gli **indirizzi delle variabili allocate in memoria**.

Dominio:

Il dominio di una variabile di tipo puntatore è un insieme di indirizzi.

Quindi il valore contenuto in una variabile P di tipo puntatore può essere **l'indirizzo di un'altra variabile (detta variabile *puntata*)**.



2

I puntatori in C++

In C++ i puntatori si definiscono mediante **il costruttore ***.

Definizione di una variabile puntatore:

```
<TipoElementoPuntato> *<NomePuntatore>;
```

dove:

<TipoElementoPuntato> e` il tipo della variabile puntata

<NomePuntatore> e` il nome della variabile di tipo puntatore

il simbolo ***** è **il costruttore del tipo puntatore.**

Esempio:

```
int *P;           // P è un puntatore a intero
```

I puntatori in C++

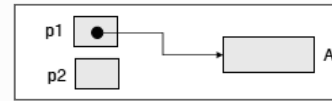
Operatori:

- **Assegnamento (=):** è possibile l'assegnamento tra puntatori (dello stesso tipo).
È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*):** è un operatore unario.
Si applica a un puntatore e restituisce il valore contenuto nella cella puntata.
Serve per accedere alla variabile puntata.
- **Operatori aritmetici: +, -, ++, --**
In generale, possono essere *sommati numeri interi* ai puntatori.
NON ha senso sommare dei puntatori tra loro.
- **Operatori relazionali: >, <, ==, !=**
I puntatori possono essere confrontati tra loro
- **Operatore indirizzo &:** si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

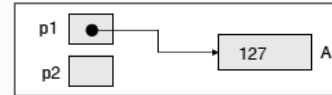
I puntatori in C++

Esempio:

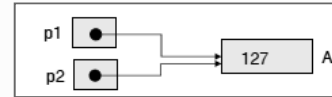
```
int *p1, *p2;
int A;
p1 = &A;
```



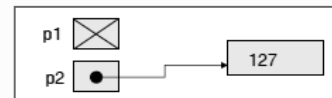
```
*p1 = 127;
```



```
p2 = p1;
```



```
p1 = NULL; // NULL è la costante che vale 0 e denota il puntatore "nullo"
```



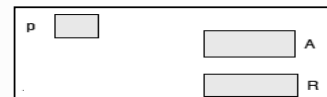
I puntatori in C++

Operatore Indirizzo &: si applica solo ad oggetti che esistono in memoria (quindi, già definiti).
 & non è applicabile ad espressioni.

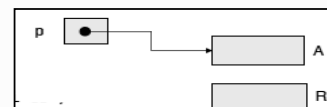
Operatore Dereferencing *: consente di accedere ad una variabile specificandone l'indirizzo
 L'indirizzo rappresenta un modo alternativo al nome (*alias*) per accedere e manipolare la variabile

Esempio:

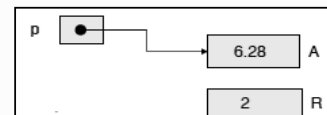
```
float *p;
float R, A;
```



```
p=&A; // *p è un alias di A
```



```
R=2;
*p=3.14*R; // A è modificato
```



I puntatori in C++

Riassumendo

- `&A` vuol dire: l'**indirizzo** della variabile A;
- `*B` vuol dire: il **valore** contenuto all'indirizzo puntato B;
- `*` e `&` si possono vedere come operatori inversi

`&*A` è come scrivere `A`

`*&A==A`

`&*A==A`

7

I puntatori in C++

Esercizio

1. Commentare tutte le istruzioni
2. Completare le istruzioni **cout** scrivendo cosa viene visualizzato.
3. Confrontare i valori visualizzati nelle istruzioni dalla n. 21 alla n. 23.
4. Se le istruzioni n.11 e n.12 fossero state:


```
*p++;
*q--;
```

 Quali valori sarebbero stati visualizzati nelle istruzioni **cout**?
(Fare le opportune considerazioni sui diversi risultati ottenuti rispetto a prima della modifica)

```

1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {int *p, *q, *t, a, b;
6 a=50;
7 b=10;
8 p=&a;
9 q=&b;
10
11 (*p)++;
12 (*q)--;|
13
14 cout<<"  "<<a<<endl;
15 cout<<"  "<<b<<endl;
16
17 cout<<"  "<<*p<<endl;
18 cout<<"  "<<*q<<endl;
19
20 cout<<"  "<<p<<endl;
21 cout<<"  "<<q<<endl;
22 cout<<"  "<<&p<<endl;
23 cout<<"  "<<&q<<endl;
24
25 cout<<"  "<<*p+1<<endl;
26 system("pause");
27 }

```

8

Puntatori a puntatori

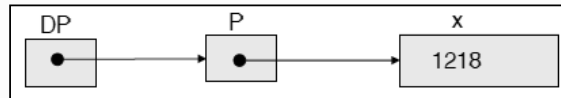
Un puntatore può *puntare* a variabili di tipo qualunque (*semplici* o *strutturate*)

→ può *puntare anche a un puntatore*

```
<TipoElementoPuntato> **< NomePuntatore>;
```

Esempio:

```
#include <iostream>
using namespace std;
int main()
{
    int x, *P, **DP;
    P = &x;
    DP = &P;
    **DP=1218;
    cout<<"x= "<<x<<" **DP= "<< **DP<<" *P= "<<*P<<endl;
    cout<<"DP= "<<DP<<" *DP= "<< *DP<<" P= "<<P<<endl;
    system ("pause");
}
```



9

Puntatori: passaggio parametri per riferimento

I puntatori permettono di realizzare il **passaggio dei parametri alle funzioni per riferimento** (o per indirizzo).

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

int main()
{
    int s,t;
    s=7;
    t=8;
    scambia( &s, &t);
    cout <<"s vale: "<<s<<" t vale: " << t<<endl;
    System ("pause");
}
```

10

Puntatori: passaggio parametri per riferimento

Esercizi

1. Realizzare una funzione che riceva tre parametri per riferimento e un quarto per valore e modifica i primi tre moltiplicandoli per il quarto. Il programma principale dopo aver richiamato la funzione deve visualizzare i tre valori modificati e i loro indirizzi di memoria.
2. Realizzare una funzione che riceva come parametro passato per riferimento un carattere dell'alfabeto inglese e se è minuscolo lo trasformi nel successivo maiuscolo in caso contrario nel precedente minuscolo (il successivo di 'z' sia 'A' e il precedente di 'A' sia 'z'). Il programma principale dopo aver richiamato la funzione deve visualizzare il carattere modificato e l'indirizzo di memoria in cui si trova.

Puntatori

Esercizi

Nei seguenti esercizi tutte le istruzioni devono essere scritte utilizzando solo i puntatori a tutte le variabili dichiarate.

3. Acquisire 10 valori interi (ciascuno inferiore a 8) e comunicare il loro prodotto (senza usare un vettore). Tutte le istruzioni devono essere scritte utilizzando solo i puntatori alle variabili dichiarate.
4. Dati N valori da tastiera comunicare la somma e la media.

11

Puntatori e vettori

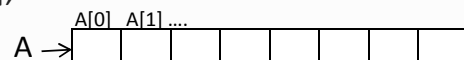
Vettori e puntatori sono strettamente correlati:

```
int x[10]; // x è ora un puntatore ad intero!
           // x corrisponde all'indirizzo del
           // primo elemento del vettore
```

x e &x[0] sono la stessa cosa
 x[0] e *x sono la stessa cosa
 x[1] e *(x+1) sono la stessa cosa

Più in generale: x[n] e *(x+n) sono la stessa cosa...

Quindi, quando dichiariamo un vettore: int A[8];



il compilatore:

- Riserva una zona della memoria per 8 componenti intere;
- Considera **A** come un **puntatore ad intero** e gli assegna l'indirizzo del **primo elemento (&A[0])**.
 (Questo ci fa comprendere meglio perché il **passaggio di un vettore nelle funzioni avvenga per indirizzo**).

12

Aritmetica dei puntatori

Le operazioni concesse sono: $++$, $--$, $+$, $-$

L'operazione di **addizione** nell'aritmetica dei puntatori **incrementa il puntatore** in base al suo tipo base.

Incrementare di 1 un puntatore significa far saltare il puntatore alla prossima locazione corrispondente ad un elemento di memoria il cui tipo coincide con quello base.

- gli operatori unari di **incremento** $++$ e **decremento** $--$ unitari
 - in forma prefissa: $++p$, $--p$;
 - in forma postfissa: $p++$, $p--$;
- l'operatore di **somma o differenza** tra un puntatore ed un intero
 - p. es.: $p+i$ punta all'elemento che segue di i posizioni quello puntato da p ;

Esempio:

Tipo $*p$;

$p=p+1$;

operazione
logica

$p=p+sizeof(Tipo)$

operazione
algebraica

13

Aritmetica dei puntatori

In generale:

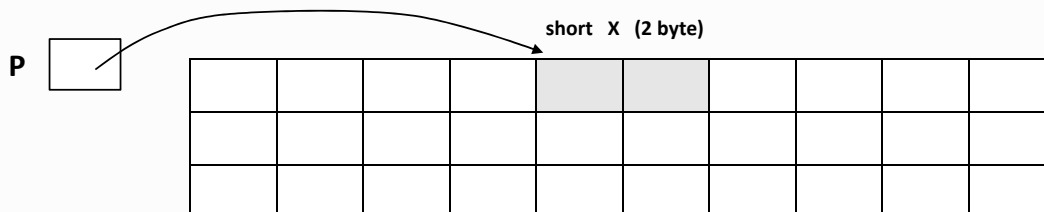
Tipo $*p$;

$p = p + k$;

operazione
logica

$p = p + sizeof(Tipo) * k$

operazione
algebraica



Esempio:

short $*P$, X ;

$P = \&X$;

14

Aritmetica dei puntatori

In generale:

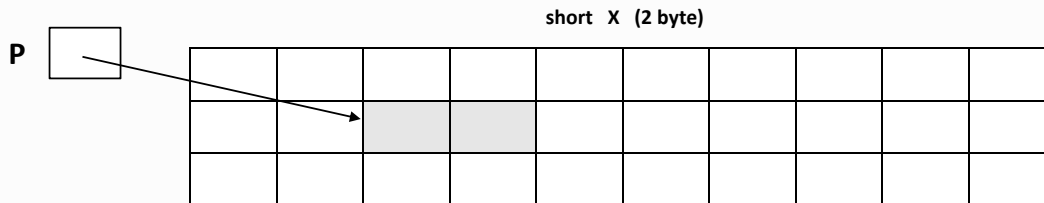
Tipo *p;

$$p = p + k;$$

operazione
logica

$$p = p + \text{sizeof(Tipo)} * k$$

operazione
algebraica



Esempio:

short *P, X;

P=&X;

P = P + 4;

P = P + 2*4

Questa tecnica è particolarmente utile per i puntatori ad array.

Aritmetica dei puntatori e vettori

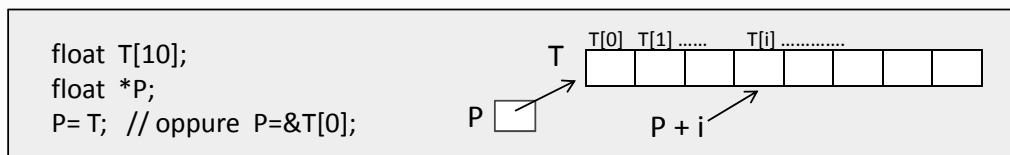
Le operazioni concesse sono: +, -, ++, --

L'operazione di **addizione** nell'aritmetica dei puntatori **incrementa il puntatore in base al suo tipo base**.

Incrementare di 1 un puntatore ad un vettore significa far saltare il puntatore all'elemento successivo del vettore che si troverà a distanza di tanti byte dal precedente elemento del vettore, quanti ne occupa il *tipo* dichiarato.

Se **P** è l'indirizzo di una variabile di tipo vettore T,

ossia P è &T[0], P+1 è &T[1] → allora per definizione **P+i** è &T[i]



```
float T[10];
float *P;
P= T; // oppure P=&T[0];
```

La differenza tra P e T è la seguente: P è una **variabile** di tipo puntatore
T è una **costante** di tipo puntatore

Aritmetica dei puntatori e vettori

Conoscendo l'indirizzo del primo elemento e la dimensione dell'elemento, è possibile calcolare l'indirizzo di qualunque elemento del vettore:

Operatori aritmetici (*somma e sottrazione*) su puntatori a vettori

Se **P** e **Q** sono puntatori ad elementi di vettori ed **i** è un intero:

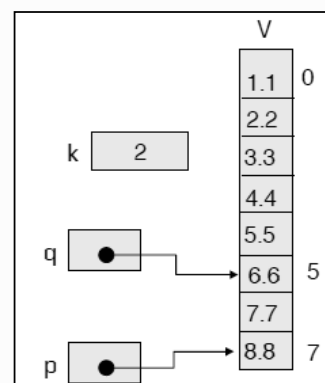
- **(P + i)** restituisce l'indirizzo dell'elemento spostato di **i posizioni** in **avanti** rispetto a quello puntato da P;
- **(P - i)** restituisce l'indirizzo dell'elemento spostato di **i posizioni** all'**indietro** rispetto a quello puntato da P;
- **(P - Q)** restituisce l'intero che rappresenta il numero di elementi compresi tra P e Q.

17

Aritmetica dei puntatori e vettori

Esempio

```
int main()
{ float V[8]={1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8};
  int k;
  float *p, *q;
  p=V+7;
  q=p-2;
  k=p-q;
  cout <<"*p = " << *p << " *q = " << *q << " k = " << k << endl;
  system ("pause");
}
```



18

Aritmetica dei puntatori e vettori

Durante l'esecuzione di ogni programma C++, ogni riferimento ad un elemento di un vettore viene tradotto in un puntatore dereferenziato.

Per esempio:

V[0]	viene tradotto in	*V
V[1]	viene tradotto in	*(V + 1)
V[i]	viene tradotto in	*(V + i)
V[expr]	viene tradotto in	*(V + expr)

Esempio:

```
int main()
{ char A[ ] = "0123456789"; // A è un vettore di 10 char
  int i = 5;
  cout<< A[i]<<" "<< A[5] <<" "<< i[A] <<" "<< 5[A] <<" "<< (i-1)[A]<< endl; // ..... !!!
  system("pause");
}
```

Cosa stampa?

NB: Per il compilatore A[i] e i[A] sono lo stesso elemento, perché viene sempre eseguita la conversione: A[i] =>*(A+i) (senza eseguire alcun controllo né su A, né su i).

19

Puntatori e vettori

Esercizi

Nei seguenti esercizi tutte le istruzioni devono essere scritte utilizzando solo i puntatori a tutte le variabili dichiarate.

1. Scrivere un programma che dato un vettore A di 20 numeri reali, calcoli la somma di tutti gli elementi di A memorizzati in posizione dispari, la somma di tutti gli elementi di A memorizzati in posizione pari e la media delle due somme.
2. Scrivere un programma che ordina n numeri casuali con il metodo bubblesort, in modo che due elementi del vettore vengano scambiati con l'utilizzo dei puntatori.
3. Definire una matrice costituita da 3 righe e 5 colonne. Stampare l'indirizzo di ciascun elemento della matrice e dedurre come gli elementi di una matrice vengano organizzati dal compilatore nella memoria del calcolatore.

20

Puntatori a strutture

E' possibile utilizzare i **puntatori** per accedere a **variabili di tipo struct**, tenendo conto che il *punto della notazione postfissa* ha la precedenza sull'*operatore di dereferencing* *.

Esempio:

```

struct studente
{ string Cognome;
  int Eta;
};
.....
studente S, *P;
P = &S;
(*P).Cognome="Rossi"; // Assegna la stringa "Rossi" al campo Cognome della
                        // struct puntata da P (è necessario usare le parentesi)
(*P).Eta=16;         // Assegna la costante 16 al campo Eta della
                        // struct puntata da P

```

21

Puntatori a strutture

Operatore ->

L'operatore -> consente di **accedere ad un campo** di una struttura referenziata da un puntatore in modo più sintetico, per esempio: **P->Eta=16;**

Esempio: con uso di (*P)

```

struct studente
{ string Cognome;
  int Eta;
};
.....
studente S, *P;
P = &S;
(*P).Cognome="Rossi";
(*P).Eta=16;

```



Esempio: con uso di P->

```

struct studente
{ string Cognome;
  int Eta;
};
.....
studente S, *P;
P = &S;
P->Cognome="Rossi";
P->Eta=16;

```

22

Puntatori a strutture

Esercizi

1. Definire la struttura **citta** con i seguenti campi: nome, provincia, n° abitanti anno prec., n° abitanti anno in corso, estensione in kmq. Istanziare una variabile di tipo **citta** e un puntatore ad essa. Utilizzando il puntatore: acquisire tutti i dati di una città, calcolare la densità di popolazione e l'incremento o il decremento percentuale della popolazione, stampare: tutti i dati compresi quelli calcolati, l'indirizzo della variabile istanziata e l'indirizzo di ciascun campo.
2. Scrivere un programma che consenta di inserire i nomi di 50 città italiane e i dati relativi alle precipitazioni nei 30 giorni di un mese (supporre che il mese sia di 30 giorni). I dati sono organizzati in una struttura dati (tabella).
La funzione `carica_citta` riceve come parametro un puntatore ad una struttura di tipo record (singola città) e acquisisce i valori dei campi inseriti da tastiera usando il puntatore.
La funzione `tot_pioggia_citta` riceve come parametro un puntatore ad una struttura di tipo record (singola città) e restituisce il totale dei mm di pioggia caduti nel mese in quella città.
Il programma deve calcolare la media giornaliera delle precipitazioni per ogni singola città.