puntatori

Attributi di un oggetto

nome o identificatore;

tipo o classe;

valore o valori;

indirizzo;

Indirizzo

Consideriamo la dichiarazione con inizializzazione:

```
string name="My Name";
```

Questa definisce un oggetto con attributi

identificatore: name

tipo: string

valore: "My Name"

indirizzo: &name, noto solo dopo che il codice compilato sia stato caricato in memoria

Puntatori

- I puntatori sono oggetti di tipo int usati per memorizzare gli indirizzi di altri oggetti (che possono essere a loro volta interi ma anche char, double, string, o di qualunque altro tipo o classe).

- I puntatori vengono utilizzati per allocare e deallocare la memoria durante la fase di esecuzione del programma, ovvero dinamicamente.

Dichiarazione di un puntatore

Un puntatore ad un oggetto intero si definisce con:

```
int  *  p_int;
```

Un puntatore ad un oggetto double si definisce con:

```
double  *  p_double;
```

Un puntatore ad un oggetto di tipo mio_oggetto si definisce con:

```
mio_oggetto  *  p_mio;
```

La posizione di * è irrilevante, posso scrivere anche

```
mio_oggetto   *p_mio;
mio_oggetto*   p_mio;
```

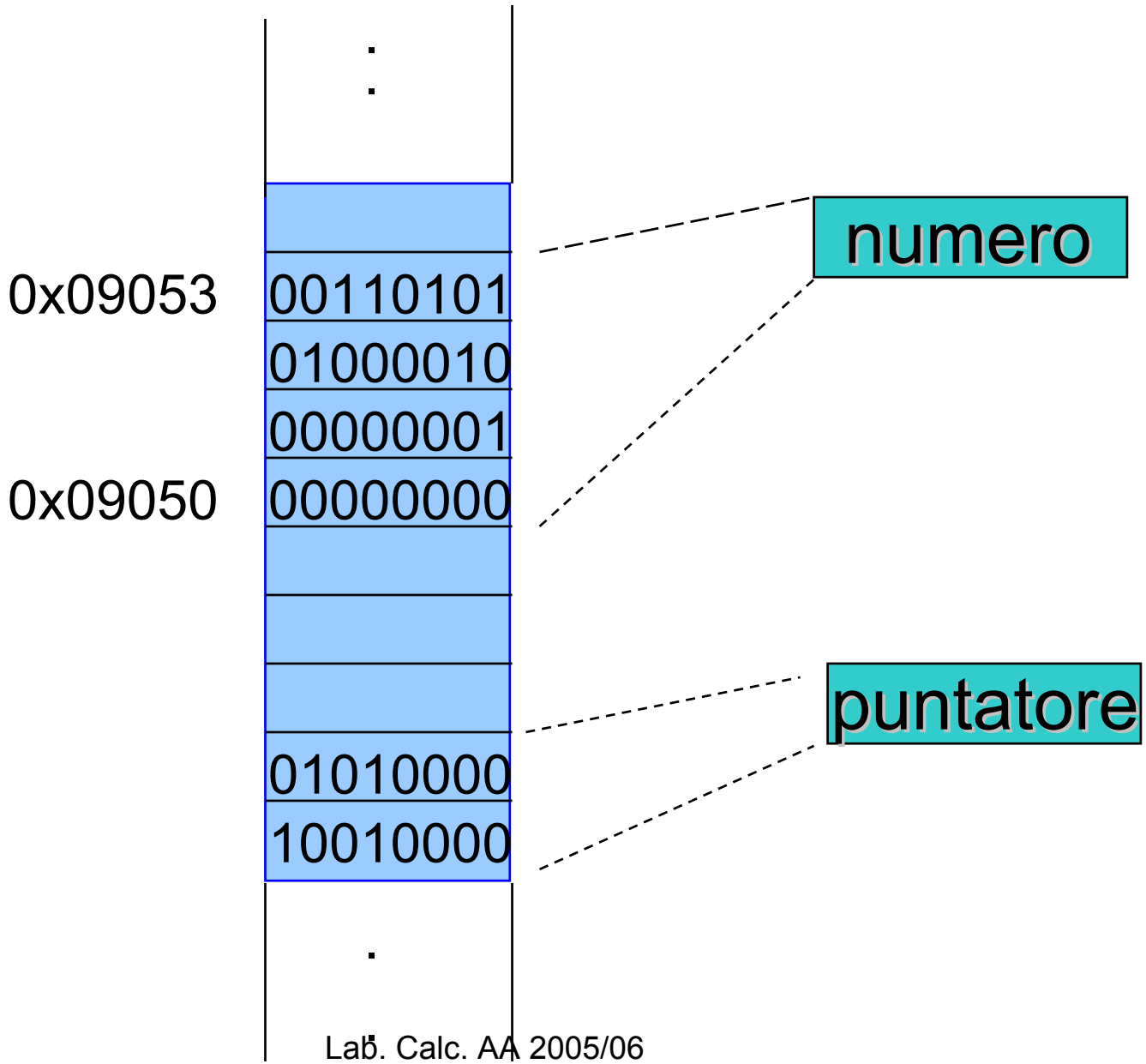
Sintassi

```
        tipo   *   identificatore;
// identificatore è un puntatore ad un oggetto
// di tipo tipo (tipo o classe qualsiasi)
        tipo lvalue  = *identificatore;
// in questa espressione il contenuto della
   locazione
// di memoria che ha come indirizzo identificatore
// viene copiato nell'oggetto lvalue.
```

Uso dei puntatori

In C++ puntatori e indirizzi sono sinonimi

```
   int  numero = 82485;
// un oggetto intero a 32 bit
// che in binario si scrive
// 00000000 00000001 01000010
   00110101
   int * puntatore;
// puntatore ad un oggetto intero
   puntatore = &numero;
// puntatore a numero
// (posto uguale all'indirizzo di numero)
```


0x09053
00110101
01000010
00000001
0x09050
00000000
01010000
10010000
numero
puntatore

Vettori e puntatori

int a[25]; a è un puntatore a interi

double x[25]; x è un puntatore a double

char ch[25]; ch è un puntatore a char

mentre a[0], x[5] o ch[10] sono rispettivamente un intero, un double ed un char.

Le dichiarazioni

```
char st[] = "A string";
char *st="A string";
```

sono equivalenti, infatti:

st[0] ha valore 'A', *st ha valore 'A'
st[7] ha valore 'g', *(st+7) ha valore 'g'
st[8] ha valore '\0', *(st+8) ha valore '\0'

Indirizzamento

- &st[0] è l'indirizzo di st[0], ovvero è la stessa cosa di st
- &st[1] è l'indirizzo di st[1], ovvero la voce successiva a quella cui punta st, è la stessa cosa di st + 1
- &st[k] è la stessa cosa di st + k per ogni k valido

Inizializzazione

Sappiamo di poter dimensionare e inizializzare un vettore di numeri con la dichiarazione

int list[] = {2,3,4,5};

Va però osservato che non è possibile farlo mediante l'istruzione

int * plist ={2,3,4,5};

Infatti plist è un puntatore all'indirizzo da cui si iniziano a memorizzare uno o più numeri interi (nel nostro caso 2,3,4 e 5) e non un vettore di interi!

list[3] è equivalente a *(plist+3)

Esempio

Cosa stamperà un programma contenente le seguenti istruzioni?

```
int   x = 16;
int * intptr ;
intptr = &x;
*intptr = *intptr + 5;
cout << x<<endl;
```

Esempio

```
        int   x = 16;  // x posto uguale a 16
        int * intptr ; // intptr è un puntatore a int
        intptr = &x; // intptr punta a x
        *intptr = *intptr + 5;
// *intptr non è altri che x, quindi x = x +5
        cout << x<<endl;
// il programma stampa 21
```

Promemoria

```
    tipo idOggetto;
// definisce idOggetto di tipo tipo
    tipo * pOggetto;
// definisce pOggetto, puntatore ad oggetto di
// tipo tipo
    pOggetto = &idOggetto;
// rispettivamente puntatore e indirizzo
    *pOggetto = idOggetto;
// l'oggetto cui pOggetto punta e l'oggetto
// idOggetto
```

Argomenti di funzioni

- In C e C++ esistono due modi principali di passare argomenti a delle funzioni
- Passaggio by value, si dà l'oggetto come argomento:

  double sqrt(double a), in questo caso la variabile che viene passata alla funzione viene usata solo in lettura dalla funzione che non la può modificare in alcun modo
- Passaggio by reference: si dà come argomento un indirizzo (o un puntatore)

  void swap(double & a, double & b), in questo caso la funzione è abilitata a restituire a e b modificati (infatti li scambia tra di loro)

Allocazione dinamica della memoria

L'operatore new alloca la quantità di memoria richiesta per un oggetto di tipo o classe tipo e restituisce l'indirizzo dell'area di memoria riservata .

Forma generale:

new tipo

Esempi

```
int * intptr = new int;
char * chptr = new char;
double * dbptr;
dbptr = new double;
```

O anche, nel caso di più oggetti

```
int * intptr;
intptr = new int[50];
```

Dimensionamento dinamico

int * intptr = new int[50];

è equivalente a int intptr[50];

ma ci sono dei casi, qualora la dimensione di un vettore non sia nota a priori, in cui si può usare solo la prima forma.Possiamo infatti scrivere

```
int n;
int * intptr;
cin << n;
intptr = new int[n];
```

ma non

```
cin << n;
int intptr[n];   // errore!
```

Delete

Per liberare memoria allocata dinamicamente si usa la keyword delete, ovvero

```
int * intPtr;
intPtr = new int;
        :
delete intPtr;
```

Esempio

```
int * intPtr;
intPtr = new int[50];
        :
delete[]  intPtr;  // o delete[50] intPtr
```

Errori frequenti

1. Oggetti inacessibili: qualora li si creino con new e ci si perda il puntatore ad essi

   if(condizione) {int * pint = new int; }

   // dopo la chiusura di questa parentesi non

   // posso più usare pint e quindi non posso

   // accedere all'oggetto , che però non è stato cancellato!
2. Puntatori che non puntino più a nulla: qualora si sia cancellato con delete l'oggetto in memoria e si provi ad utilizzarne il puntatore

Regole

- Il programma deve contenere un'istruzione delete per ogni istruzione new.
- L'operazione di delete va effettuata alla fine del blocco di codice che ha allocato la memoria, in un punto ben visibile, per essere sicuri di non utilizzare più quel puntatore.