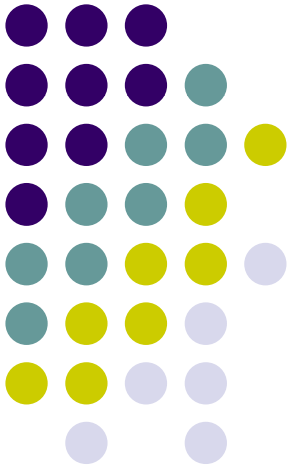
Università di Napoli Federico II - Dip. di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Corso di Laurea in Ingegneria Informatica

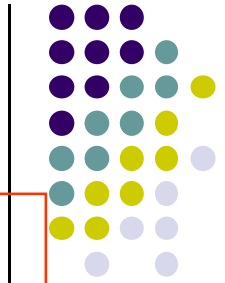
DIETI

Corso di Programmazione I

Allocazione Dinamica



Operatore new

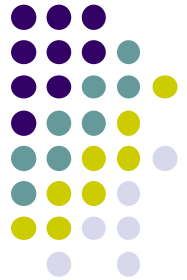


```
T* p = new T;        //alloca memoria sufficiente a contenere un
                     // oggetto di tipo T e restituisce puntatore

T* p1 =new T[n];     //alloca memoria sufficiente a contenere n
                     // elementi di tipo T e restituisce puntatore
```

In fase di esecuzione, new alloca un'area di memoria sufficiente ad ospitare un valore del tipo T e restituisce il puntatore a tale area;

il tipo T definisce implicitamente l'ampiezza dell'area di memoria occorrente.

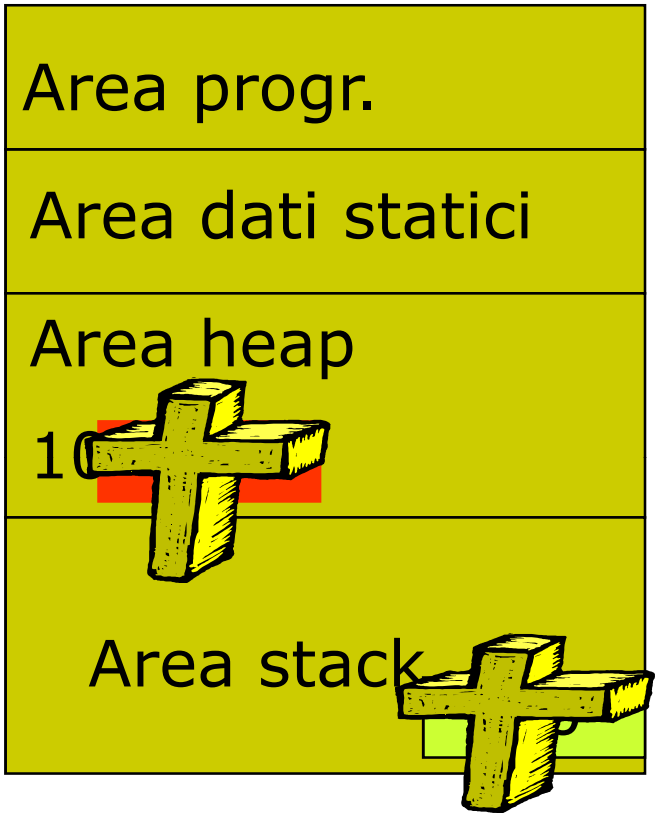
Operatore delete



```
delete p;     //dealloca area puntata da p

delete [] p1; //dealloca tutto l'array precedentemente
allocato
```

Produce la deallocazione dell'area di memoria puntata dalla variabile p, cioè annulla l'allocazione, rendendo nuovamente disponibile lo spazio di memoria prima occupato.

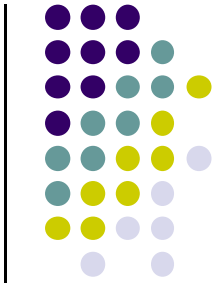
Allocazione della memoria




```
void main() {

     int *p;

    p=new int;

    *p=20;

    delete p;

}
```

Il puntatore p è una variabile automatica, quindi allocata in area stack.
Dopo l'istruzione new, p punta ad una locazione nello *heap* atta a contenere un intero

Allocazione dinamica di un vettore



Con l'allocazione dinamica di un vettore il puntatore restituito è quello all'indirizzo del primo elemento del vettore, pertanto è di tipo T* se T è il tipo degli elementi del vettore

```
float*  a= new float[n];
```

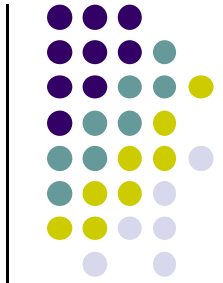
Il riferimento agli elementi del vettore viene espresso a mezzo della consueta notazione con indice

```
a[i] = 10.0
```

```
*(a+i)=10.0
```

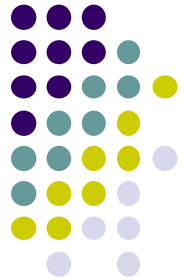
```
delete [] a; //cancella l'intero vettore precedentemente allocato
```

Esempio:



```
// allocazione dinamica del vettore (con deallocazione)

#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

int main(void) {

   int  *vett;
   int   n = 10;

    vett = new int[n];
    cout << "allocati " <<  n*sizeof(int) << " byte" << endl;

   delete [] vett;
    cout << "deallocati " << n*sizeof(int) << " byte" << endl;
   }
   return 0;
}
```

Allocazione dinamica di una stringa

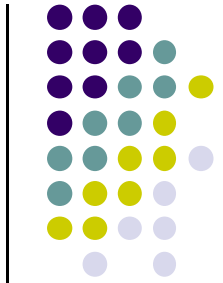


- Con l'allocazione dinamica di una stringa il puntatore restituito è quello all'indirizzo del primo elemento della stringa. Pertanto è di tipo char *

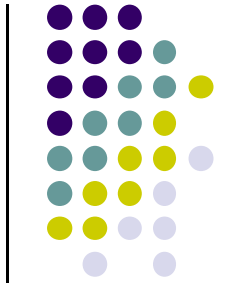
```
char*  s= new char[n+1];
```

- L'istruzione precedente alloca spazio per una stringa di n caratteri **più il terminatore**
- Valgono le stesse considerazioni fatte per i vettori

Esempio

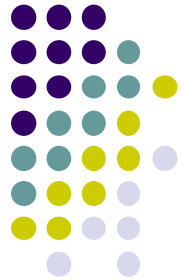


- Un programma che alloca spazio in area heap per una stringa di *n* caratteri e la inizializza con dati letti da tastiera.



```
#include <iostream>
#include <stdlib.h>

using namespace std;

int main(int argc, char *argv[])
{
  char * stringa;
  int n;

  cout << "\n quanto e' lunga la stringa che vuoi inserire?";
  cin >> n;

  stringa = new char[n+1];

  cout << "\n inserisci la stringa: ";
  cin.ignore();

  cin.getline(stringa,n+1);   //inserisce il terminatore

  cout << "\n hai inserito: ";
  cout << stringa;

  delete [] stringa;
  system("PAUSE");
  return 0;
}
```

Allocazione dinamica di un record

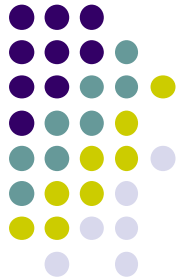


L'allocazione dinamica di un record avviene analogamente attraverso l'uso dell'operatore **new**

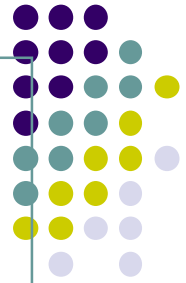
Indicato con **R** un tipo record e con **r** un puntatore ad **R**, si ha:

```
R* r = new R;
```

Esempio

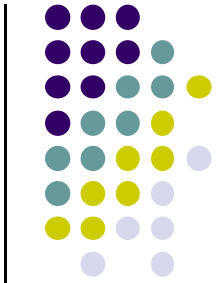


- Dichiarazione di un tipo (record) Punto, allocazione dinamica di una variabile di tipo Punto e sua inizializzazione.



```
#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

struct Punto {
   float X;
   float Y;
};

int main(void) {

   Punto P1; // allocazione statica
   P1.X=0.0;
   P1.Y=0.0;

   Punto * PuntP;
   PuntP=new Punto;
   PuntP->X=3;  // (*PuntP).X
   PuntP->Y=4;  // (*PuntP).Y

   cout << "\n le coordinate del punto P1: " << P1.X << "," << P1.Y;
   cout << "\n le coordinate del punto allocato dinamicamente: " << PuntP->X << "," << PuntP->Y;
   delete PuntP;
   system("PAUSE");
   return 0;
}
```

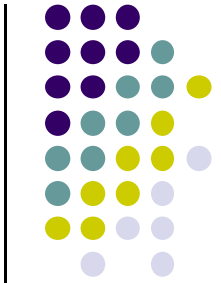
Un caso particolare di assegnazione tra variabili ….



- Definito un tipo record “Persona” si vuole eseguire il seguente programma:

```
int main(int argc, char *argv[]) {

  Persona r1, r2;

  Inizializza_Persona(r1);

  r2=r1;

  void * ptr;

  cout << "\n indirizzi degli elementi della stringa r1.Nome: " << endl;
  for (int i=0; r1.Nome[i]!='\0'; i++)
    {ptr=&(r1.Nome[i]); cout << ptr << endl;}
  cout << "\n indirizzi degli elementi della stringa r2.Nome: " << endl;
  for (int i=0; r2.Nome[i]!='\0'; i++)
    {ptr=&(r2.Nome[i]); cout << ptr << endl;}
  cout << "\n";
  system("PAUSE");
  return 0;
}
```

Due diverse definizioni di “Persona”



- **Definizione 1:**

```
const int dim=100;
struct Persona {
    char Nome[dim];
    int eta;
};
```

- La dimensione della stringa è nota al tempo di compilazione

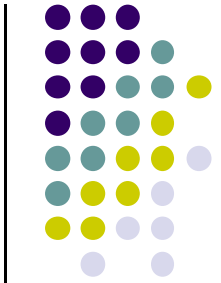
- **Definizione 2:**

```
struct Persona {
    char * Nome;
    int eta;
};
```

Stringa allocata dinamicamente

- La dimensione della stringa NON è nota al tempo di compilazione

La funzione Inizializza_Persona



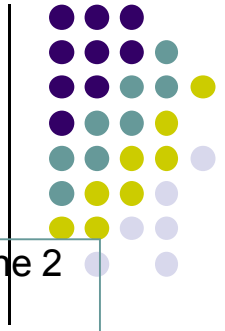
- Nel caso della Definizione 1

```
void Inizializza_Persona(Persona & P) {

   cout << "\n inserire Nome e Cognome: ";
   cin.getline(P.Nome, dim-1);
   cout << "\n inserire l'eta': ";
   cin >> P.eta;
}
```

- Nel caso della Definizione 2

```
void Inizializza_Persona(Persona & P) {

 char Buffer[dim];
   cout << "\n inserire Nome e Cognome: ";
   cin.getline(Buffer, dim-1);
   P.Nome=new char [strlen(Buffer)+1];
   strcpy(P.Nome,Buffer);
   cout << "\n inserire l'eta': ";
   cin >> P.eta;
}
```

…in esecuzione



- Output ottenuto nel caso della Definizione 1

inserire Nome e Cognome: Mario Loy

inserire l'eta': 56

indirizzi degli elementi della stringa r1.Nome:
0x23fd60
0x23fd61
0x23fd62
0x23fd63
0x23fd64
0x23fd65
0x23fd66
0x23fd67
0x23fd68

indirizzi degli elementi della stringa r2.Nome:
0x23fdd0
0x23fdd1
0x23fdd2
0x23fdd3
0x23fdd4
0x23fdd5
0x23fdd6
0x23fdd7
0x23fdd8

Premere un tasto per continuare . . .

- Output ottenuto nel caso della Definizione 2

inserire Nome e Cognome: Mario Loy

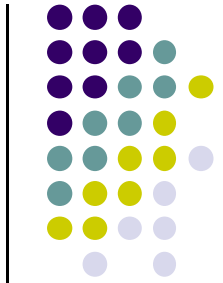
inserire l'eta': 56

indirizzi degli elementi della stringa r1.Nome:
0x671450
0x671451
0x671452
0x671453
0x671454
0x671455
0x671456
0x671457
0x671458

indirizzi degli elementi della stringa r2.Nome:
0x671450
0x671451
0x671452
0x671453
0x671454
0x671455
0x671456
0x671457
0x671458

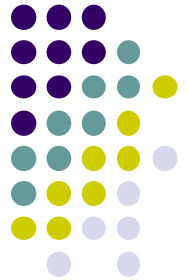
Premere un tasto per continuare . . .

Allocazione dinamica di una variabile puntatore



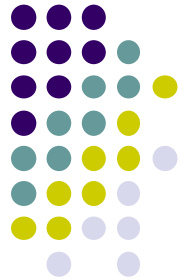
- Una variabile di tipo puntatore appartiene – come ogni altra variabile – ad una classe di memorizzazione che ne definisce la visibilità e stabilisce dove la variabile viene allocata
- Una variabile puntatore può essere pertanto in generale allocata in area stack, in area dati statici, in area heap ed anche essere dichiarata costante.

Errori comuni con puntatori e variabili dinamiche



- Tipici errori quando si usa l'allocazione dinamica sono:
  - Dimenticare di allocare un dato nello *heap* e usare il puntatore come se lo stesse riferendo (produce un effetto impredicibile)
  - Dimenticare di "restituire" la memoria allocata quando non serve più (rischio di *memory leak*)
  - Dimenticare di riassegnare ad un valore valido puntatori che puntavono ad aree di memoria ormai rilasciate (*dangling reference*)
  - Tentare di usare dati dinamici dopo che sono stati deallocati

Leak di memoria (Memory Leak)

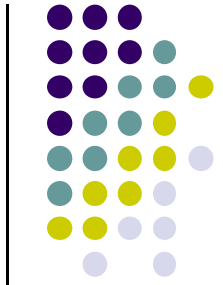


- Quando una variabile allocata dinamicamente non serve più (il suo ciclo di vita si è esaurito all'interno del programma) deve essere distrutta dal programmatore mediante l'operatore ***delete***
- Se questa operazione non viene effettuata, **l'area di memoria heap da essa occupata in memoria non viene rilasciata** fino al termine dell'esecuzione del programma
- Se poi il puntatore che puntava a tale area viene assegnato ad un altro indirizzo, l'area occupata risulta irraggiungibile
- Si avrà quindi **uno spazio in memoria *non utilizzato o non più utilizzabile***
- *Questo fenomeno è noto con l'espressione memory leak*

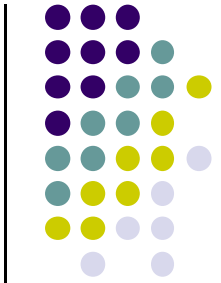
Esempio:

```
// allocazione del vettore con deallocazione ed esempio di memory leak

#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

int main(void) {

   int  *vett;
   int   n = 1000000;

   while (true) {
     vett = new int[n];
    cout << "allocati " <<  n*sizeof(int) << " byte" << endl;

   // delete [] vett;  Commentando questa istruzione non dealloco mai...
    cout << "deallocati " << n*sizeof(int) << " byte" << endl;
   }
   return 0;
}
```

Dangling reference

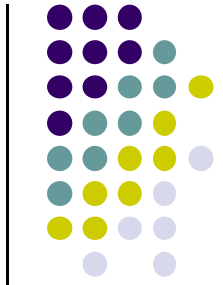


- Una volta rilasciata la memoria mediante l'operatore ***delete*** il puntatore che si riferiva a quell'area di memoria *continua a contenere l'indirizzo relativo*
- *Pertanto è necessario assegnare tale puntatore a 0 o assegnarlo ad un indirizzo valido per evitare il fenomeno noto come dangling reference: cioè la presenza di* ***puntatori che erroneamente si riferiscono ad indirizzi di memoria che sono stati rilasciati*** *e di cui quindi il programmatore non ha più il controllo*



```
// esempio di dangling reference

#include <iostream>
#include <stdlib.h>

using namespace std;

int main(int argc, char *argv[])
{
  int * intptr;

  intptr=new int;
  *intptr=23;
  cout << "il numero intero cui punta intptr e':" << *intptr;
  cout << "il suo indirizzo in memoria e': " << intptr;

  delete intptr;
  cout << "Lo spazio allocato e' stato rilasciato" << endl;
  *intptr=3200; // GRAVE!!!!
  cout << "il numero intero cui punta intptr e':" << *intptr;
  cout << "il suo indirizzo in memoria e': " << intptr;

  system("PAUSE");
  return 0;
}
```

Riferimenti



- Deitel&Deitel: C++ Fondamenti di programmazione
  - Cap. 10, § 10.6
- B.Stroustrup, C++ … principi di programmazione
  - Cap. 11, § 11.2 escluso 11.2.4
- Deitel&Deitel: C++ How to program 8° ed.
  - Cap. 11, §11.9
- Deitel&Deitel: C++ How to program 10° ed.
  - Cap. 10, §10.9