



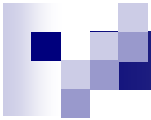
# Linguaggio C++

## Record



# Definizione

- Un **record** è un insieme di dati non necessariamente omogenei, in relazione tra loro come attributi di una stessa unità informativa.
- Esempio:  
PIANETA
  - Nome → stringa
  - Visibilità ad occhio nudo → booleano
  - Diametro → reale
  - Raggio orbitale medio → reale

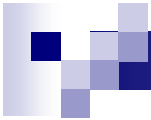


# Dichiarazione e definizione di un record

- Record (o struttura) che descrive un punto nel piano cartesiano:

```
struct posiz{           //dichiarazione tipo record
    double x;
    double y;
};
```

```
struct posiz p1, p2;    //definizione di due var di tipo record
```



# Dichiarazione e definizione di un record

- Riferimento ai membri (campi) della struttura (record):

p1.x = 3.5;      //dot notation

p1.y = 4.9;



# Passaggio per valore di un record

```
void stampaPosizione(struct posiz p) {  
    cout<<"x = "<<p.x<<" y = "<<p.y;  
}
```

Chiamata:

```
stampaPosizione(p1);
```



# Ritorno di un record da una funzione

- Come una struttura può essere passata (per valore ad una funzione), può anche essere ritornata (per valore) da questa alla funzione chiamante:

```
struct posiz posizioneOpposta(struct posiz p) {  
    p.x = - p.x;  
    p.y = - p.y;  
    return p;  
}
```

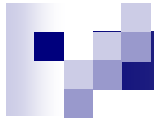
Chiamata:     p2 = posizioneOpposta(p1);  
(dove p2 è una variabile di tipo struct posiz).



# Passaggio di una struttura by reference

```
void posizioneOpposta(struct posiz &p) {  
    p.x = -p.x;  
    p.y = -p.y;  
}
```

Chiamata: `posizioneOpposta(p1);`



# Array di record

```
struct posiz esa[6]; //vertici di un esagono
```

```
double x,y;
```

```
    x = esa[2].x;
```

```
    y = esa[2].y;
```

```
struct posiz p
```

```
    p = esa[5];           //assegnazione diretta fra record
```





# Operazioni di I/O

```
struct posiz{
    double x;
    double y;
};

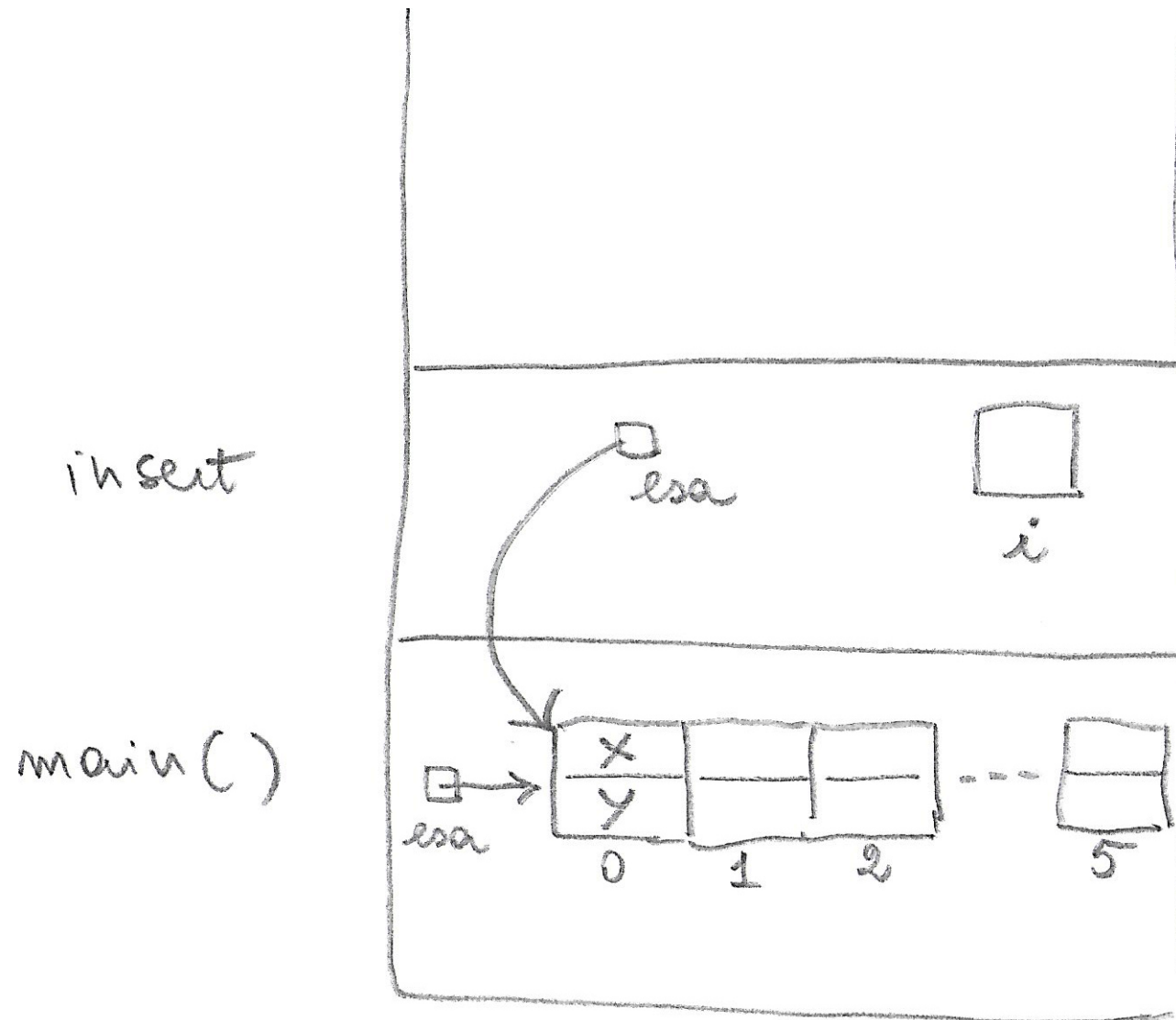
int main() {
    struct posiz esa[6];

        insert(esa);
        stampa(esa);

        return 0;
}
```

```
void insert(struct posiz esa[]) {
    int i;
        for(i=0; i<6; i++) {
            cout<<"...";
            cin>>esa[i].x>>esa[i].y;
        }
}
```

```
void stampa(const struct posiz esa[]) {
    int i;
        for(i=0; i<6; i++){
            cout<<esa[i].x<<"\t"<<esa[i].y<<endl;
        }
}
```





# Osservazione

- I campi di un record possono essere di qualunque tipo (int, char, double, float), vettori, matrici, stringhe e per finire di tipo record.



# Strutture annidate:

strutture con campi di tipo struct

- Voglio memorizzare una data.
- Ci sono varie soluzioni, vediamo due:

- ☐ Stringa: `char data[9];`      `//GGMMAAAA`

- ☐ Record

```
struct data{  
    int giorno;  
    int mese;  
    int anno;  
};
```



# Strutture annidate:

## strutture con campi di tipo struct

```
struct data{  
    int giorno;  
    int mese;  
    int anno;  
};
```

```
struct studente{  
    char cognome[25];  
    char nome [21];  
    struct data dataNascita;  
    double peso;  
    double altezza;  
};
```

```
int main() {  
    struct studente classe[MAX_STUD];  
    int numStudenti;  
    ....  
  
    strcpy(classe[i].nome, nome);  
    ...  
    classe[i].peso = 70.5;  
    ...  
    classe[i].dataNascita.giorno = 12;  
    ...  
}
```



# Strutture annidate:

strutture con campi di tipo struct

```
struct studente classe[ ] = {  
    {"Pelizzoni", "Paolo", 10, 9, 1962, 72.5, 1.72},  
    ...  
};
```



# typedef

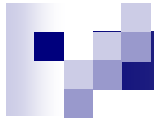
- Con **typedef** si definisce un sinonimo (alias) e non un nuovo tipo.

**typedef** <tipoVecchio> <tipoNuovo>;

Esempi:

typedef struct automobile car;	// car è sinonimo di struct automobile
typedef int elem;	// elem è sinonimo di int
typedef char nome[25];	// nome è sinonimo di char [25]

Vantaggio: leggibilità



# Return di un record

Esempio:

```
#include <stdlib>
```

```
div_t div (int num, int den);    //divide due interi
```

```
typedef struct {  
    int quot;  
    int rem;  
} div_t;
```






# Return di un record

Esempio:

```
div_t x;
```

```
    x = div(10,3);
```

```
    cout<<x.quot<<" "<<x.rem;
```



# Implementazione di un record in memoria

```
struct nomeRecord{  
    int campo1;           //4 byte  
    double campo2;        //8 byte  
    char campo3;          //1 byte  
};
```

```
struct nomeRecord var;
```

Indirizzo campo i-esimo = Indirizzo iniziale + offset  
offset = distanza che separa il campo dall'inizio del record

Sia I = indirizzo iniziale

$I_1 = I + O_1$	$O_1 = 0$
$I_2 = I + O_2$	$O_2 = 4$
$I_3 = I + O_3$	$O_3 = 12$



# Tipo enumerativo: enum

- Un'enumerazione è un insieme di costanti intere definite con un nome.

## Esempio:

```
enum colore{NERO, BLU,VERDE, ROSSO, MAGENTA, BIANCO};
```

```
enum colore sfondo;           //definizione della variabile sfondo di tipo enum colore
```

N.B. I valori simbolici delle variabili di un tipo enumerativo non possono essere acquisiti da tastiera né visualizzati sullo schermo.

Vantaggi: leggibilità



# union

- Una union permette di memorizzare, in momenti diversi dell'esecuzione, variabili di tipo diverso:

```
union id{                                //dichiarazione tipo
    char name[21];
    long serialNumber;
};
```



# union

- Il compilatore riserva lo spazio necessario per l'allocazione in memoria della variabile più grande (“name” in questo caso).

**union** id who; //definizione variabile

- Sono validi entrambi gli assegnamenti:

who.serialNumber = 10023;

...

strcpy(who.name, “xyz”);



# Esempio di utilizzo di **union**: record variant

```
#define MAX_FIGURE 4
```

```
enum tipoCurve{PUNTO,LINEA,CERCHIO};
```

```
struct tipoPunto{  
    double x, y;  
};
```

```
struct tipoLinea{  
    double coeffX,coeffY,termNoto;  
};
```

```
struct tipoCerchio{  
    tipoPunto centro;  
    double raggio;  
};
```



# Esempio di utilizzo di **union**: record variant

```
union tipoCurva {  
    tipoPunto punto;  
    tipoLinea linea;  
    tipoCerchio cerchio;  
};
```

```
struct figura{  
    tipoCurve tipo;  
    tipoCurva curva;  
};
```



# Esempio di utilizzo di **union**: record variant

```
void insert (figura disegni[], int &numFigure);  
void stampa (const figura disegni[], int numFigure);
```

```
int main() {  
    figura disegni[MAX_FIGURE];  
    int numFigure=0;  
  
    //cout<<sizeof(disegni)<<endl;  
  
    insert(disegni,numFigure);  
    stampa(disegni,numFigure);  
  
    return 0;  
}
```

La codifica è lasciata per esercizio.