I tipi strutturati e i record

Riferimenti:

Alla scoperta dei Fondamenti dell'Informatica, paragrafo 5.4

Che C serve..per iniziare a programmare, capitolo 4

Gli array: dichiarazione

- Un array è una struttura composta da un insieme di elementi tutti dello stesso *tipo* e con lo stesso *nome*. Il numero di elementi dell'array è detto dimensione ma anche *cardinalità* dell'array.
- Per dichiarare un array si devono specificare il tipo dei dati in esso contenuti, il nome dell'array e la sua cardinalità

```
<Tipo> nome_array[dimensione];
```

- La dimensione dell'array deve essere un valore costante per consentire al compilatore di fissarne l'allocazione in memoria prima della esecuzione del programma. Gli elementi vengono allocati in locazioni di memoria consecutive. La dichiarazione è senza inizializzazione, quindi il contenuto degli elementi dell'array v non è definito.
- **Esempio**:

int vettore[5];

- Indica un array di 5 locazioni di tipo intero, non inizializzate

- l'etichetta *vettore* corrisponde all'indirizzo del primo elemento dell'array

Gli array: inizializzazione

- L' inizializzazione di un array si effettua indicando i valori degli elementi tra parentesi graffe separati da virgole
- **Esempio**

```
float v[3] = {0.5, -13.25, 12.2};
char lettera[5]={'a', 'b', 'c', 'd', 'e'};
int vett[3] = {0}; //inizializza tutti i valori a zero
```

Gli array: accedere agli elementi

- Per accedere ad uno specifico elemento dell'array si usa il nome dell'array e la posizione dell'elemento fra parentesi quadre
- Gli elementi di un array **partono dalla posizione 0**; se N è la cardinalità, gli elementi hanno indice fra 0 e N-1
- Per indicare la posizione di un elemento dell'array si possono usare valori costanti, espressioni o variabili, ma tutti devono essere di tipo intero

Esempio:

- **vettore[3]** indica l'elemento in posizione 3 (il quarto) dell'array *vettore*
- **vettore[i]** indica l'elemento in posizione i-esima
- **vettore[c+3]** indica l'elemento in posizione c+3

Gli array multidimensionali

- Si possono anche dichiarare array multidimensionali.

```
<Tipo> nome_array[dim1] [dim2]… [dimn];
```

- Essi sono strutture in cui sono necessarie più posizioni per identificare un elemento (es. matrici)

- **Esempio**

int mat[2][3]; //dichiara una matrice mat di 2 righe e 3 colonne

mat[i][j] indica l'elemento di posto (i,j) in mat

- Anche gli array bidimensionali vengono disposti in locazioni di memoria consecutive. In particolare vengono disposte le righe una dopo l'altra
- **Esempio**

int mat[2][3] = {1,2,3,4,5,6}; si può scrivere anche

int mat[2][3] = {{1,2,3},{4,5,6}};

int mat[3][3] = {0}; inizializza tutti i valori a zero

Definire un nuovo tipo array

- Con **typedef** è possibile definire nuovi tipi a partire da quelli semplici e strutturati

- ***Esempio 1:***

#define N 10

typedef float arrayDiReali[N];

Definisce un nuovo tipo che si chiama arrayDiReali con cui è possibile dichiarare delle variabili:

arrayDiReali vettore;//vettore è un array di N elementi di tipo float

- ***Esempio 2*** (multidimensionali)

#define N 2

#define M 3

typedef int matrice[N][M];

matrice mat;//istanzia una variabile di tipo matrice, ossia un array bidimensionale 2x3 di interi

Le stringhe di caratteri

- Una stringa è una sequenza di caratteri che può essere definita come array di **char**

<char nome_stringa[dim];>

- La stringa viene conclusa con il **carattere terminatore '\0'** (ASCII NULL) che occupa la posizione *dim-1*
- Una stringa può essere inizializzata come un qualsiasi array elencando i caratteri che devono farne parte, incluso il terminatore
- Esempio:

char stringa[10]={'c', 'i', 'a', 'o', '\0'); *oppure*

char stringa[10]="ciao"; *in questo caso il terminatore viene aggiunto automaticamente in fondo alla stringa*

char stringa[]="ciao"; *in questo caso la dimensione viene determinata automaticamente dalla lunghezza della costante stringa più uno*

Attenzione! Se si creano stringhe più lunghe di quanto dichiarato, non ci saranno messaggi di errore ma il programma invade aree di memoria non previste

Le stringhe di caratteri: input e output

- L'inizializzazione e le procedure di input e output possono essere effettuate sull'intera stringa invece di dover operare su un carattere alla volta:
  - **Esempio**

    char nome[10]="Antonio";

    nome="Angelo";

    cout<<nome; *stampa tutti i caratteri fino al terminatore*

    cin>>nome; *carica nel vettore nome tutti i caratteri immessi da tastiera (fino ad un eventuale spazio) e aggiunge automaticamente il terminatore*

La libreria string.h

- Il linguaggio mette a disposizione la libreria **string.h** in cui sono predefinite le funzioni di gestione delle stringhe. Per usare la libreria, è necessario includerla con la direttiva:

  #include <string.h>

- strlen: per calcolare la lunghezza di una strnga;
- strcpy: per effettuare la copia di una stringa in un'altra;
- strncpy: per effettuare la copia di n caratteri di una stringa in un'altra;
- strcat: per accodare una stringa ad un'altra;
- strncat: per accodare n caratteri di una stringa ad un'altra;
- strcmp: per il confronto tra due stringhe;
- strcnmp: per confrontare i primi n caratteri di due stringhe;
- strchr: per cercare un carattere in una stringa;
- strstr: per cercare una stringa in un'altra;

La libreria string.h

- **strlen**(v): calcola la lunghezza di v (escluso il terminatore)
- **strcpy**(v_dest, v_src): *copia v_src in v_dest*
- **strncpy**(v_dest, v_src, n): *copia i primi n caratteri di v_src in v_dest*
- **strcat**(v_dest, v_src): *concatena v_src a v_dest (sovrascrive il carattere terminatore di v_dest e aggiunge un nuovo terminatore alla fine)*
- **strncat**(v_dest, v_src, n): *concatena i primi n caratteri di v_src a v_dest*
- **strcmp**(v1,v2): *confronta v1 e v2 e ritorna un valore **<0**(v1<v2), **=0**(v1=v2), **>0***
- **strncmp**(v1,v2,n): *confronta i primi n caratteri di v2 con v1*
- **strchr**(nome_vettore, carattere) *ritorna la prima posizione in cui ha trovato il «carattere»; se non è presente ritorna il valore **null***
- **strstr**(v1,v2) cerca la stringa v2 dentro v1 e *ritorna il puntatore alla posizione in cui è presente; se non è presente ritorna il valore **null***

I record: dichiarazione

- Il **record**, detto anche **struttura (struct)**, è una collezione di elementi che possono essere eterogenei ma sono logicamente collegati (es. le informazioni su uno studente)
- Gli elementi della struttura sono detti *campi* od anche *membri*, possono essere tutti di uno stesso tipo, ma anche di tipo diverso.
- La dichiarazione della struttura prevede che ne venga fissato il nome e che ne siano elencati i campi specificando per ognuno di essi il tipo di appartenenza.

```
struct nome_record
{   tipo_campo1   nome_campo1;
    tipo_campo2   nome_campo2;
                  ……..
    tipo_campoN   nome_campoN;
                  };
```

I record: definizione di variabili

- Per la definizione di variabili si usa il nome del tipo seguito dai nomi delle variabili come di consueto:

```
struct   nome_record  record1, rcord2;
```

```
struct nome_record  record = {valore1, valore2, …., 
                              valoreN};
```

- Si può anche usare un'unica dichiarazione di tipo e di variabili come l'esempio seguente mostra.
  - In una tale definizione si può anche omettere il nome della struttura introducendo un tipo struct senza nome non più utilizzabile in seguito.

```
struct  nome_tipo_record
    { tipo_campo1  nome_campo1;
      tipo_campo2  nome_campo2;
                 …....
      tipo_campoN  nome_campoN;
        } record1, record2;
```

I record: accedere ai campi

- Una volta definita una variabile di tipo struct, si può solo operare con i suoi campi accedendo ad essi indicandone il nome preceduto dal nome del record adeguatamente separato mediante un punto (*dot notation*).

```
nome_record.nome_campo
```

- Esempio:

```
struct tipo_data
       { int giorno;
         char mese[15];  // stringa per gestire il nome del mese
         int anno;
       };
```

```
struct tipo_data data, nascita, matrimonio, presa_di_servizio;
```

```
typedef struct tipo_data
        { int giorno;
          char mese[15];
          int anno;
        } Data;
```

I record: osservazioni

- Due dichiarazioni struct diverse, anche se comprendenti gli stessi campi, introducono variabili che vengono considerate di tipo **diverso**.
- Sui record le uniche operazioni ammesse sono quelle definite sui singoli campi e la *copia* di un intero record in un altro dello stesso tipo.

- Esempio:

```
Data data, nascita, matrimonio, presa_di_servizio;
```

```
cout <<  "\nInserisci giorno :";
cin  >>  data.giorno;
cout <<  "\nInserisci mese   :";
cin  >>  data.mese;
cout <<  "\nInserisci anno   :";
cin  >>  data.anno;
```

```
nascita = data;
```