Corso di Elementi di Informatica

A.A. 2017-18

Docente: Alessandro Amirante

alessandro.amirante@unina.it

Università degli Studi di Napoli Federico II

Agenda

- Dati strutturati
- Tipi strutturati
- Array
- Array in C++
- La parola chiave typedef
- Record
- Record in C++

Dati strutturati

- Una variabile può contenere un solo valore alla volta (una variabile appartenente ad un tipo semplice contiene un unico valore scelto nell'ambito del tipo).
- Utilizzando i costruttori di tipo è possibile creare variabili che contengono valori non atomici, e cioè utilizzando i costruttori di tipo è possibile aggregare tipi (semplici) per ottenere un unico tipo strutturato
- Unità e scomponibilità dei dati strutturati
  - nome unico del dato strutturato
  - funzione di accesso (operatore)
  - ciascun componente si comporta come una normale variabile
  - specificità legate alla tipologia e al linguaggio

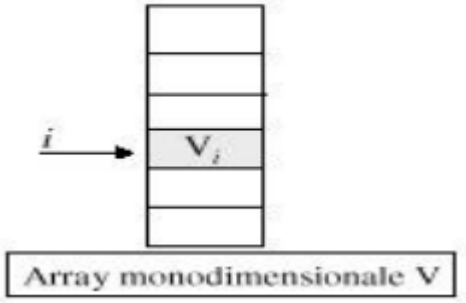
I tipi strutturati

- Essi sono caratterizzati da
  - Tipi componenti
    - Semplici
    - Strutturati
  - Modalità di assemblaggio
    - Prodotto cartesiano
    - Sequenza
  - Funzione di accesso
    - Per posizione
    - Per nome
  - Costruttore di tipo
  - Costruttore dei valori

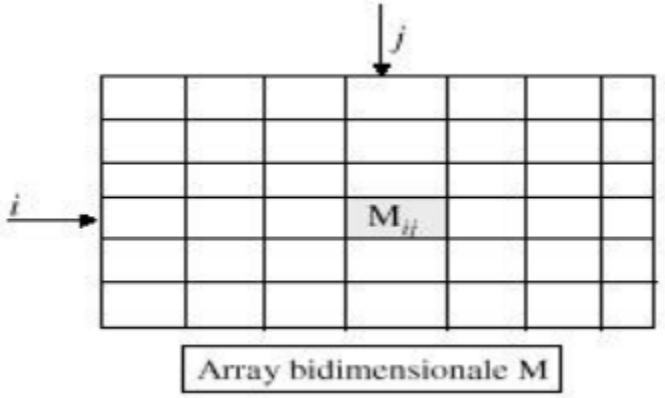
Array

- Un *array* è costituito da più tipi componenti, tutti del medesimo tipo
- L'accesso avviene a mezzo di un indice (eventualmente calcolato)

- Array monodimensionale (*Vettore*)



- Array bidimensionale (*Matrice*)



Array monodimensionali

- Sequenza finita di valori omogenei
- Il valore di un array come unica variabile è quindi una n-pla di valori componenti:
  - Es. (2, 9, 0, -1, 24, 0, 9, 6)
- Funzione di accesso
  - I componenti sono in corrispondenza biunivoca con i numeri naturali
  - I numeri associati ai componenti si dicono *indici*
  - Operandi della funzione di accesso sono l'array stesso e l'indice

Dichiarazione di array

- La dichiarazione di un array ha lo scopo di definire le seguenti informazioni fondamentali:
  - Il tipo degli elementi componenti
  - Il numero degli indici
  - Il valore minimo e massimo di ciascun indice

Accesso ad array

- La funzione d'accesso ai componenti dell'array è del tipo:
  - **`nomeArray [espressioneIndice]`**
  - **`nomeArray [espressioneIndice, espessioneIndice]`**
- *espressioneIndice* è una espressione che in generale può assumere uno dei valori dell'indice
- Esempi:
  - **`M[i]`**
  - **`M[i][j]`**

Esempi di accesso ad array

- La funzione di accesso usa il simbolo [ ] con notazione *infissa*:

```
a[3] = s[1] + x;

if ( a[4] > s[1]+3 )
  s[2] = a[2]+a[1];

x = a[i];

a[i] = a[i] + 1

a[i*x] = s[a[j+1] - 3] * (y - a[y]);
```

Operazioni definite sull'Array

- Occorre distinguere tra operatori applicabili agli elementi dell'array ed all'array in quanto struttura
- Ai dati componenti sono applicabili tutti gli operatori previsti per il tipo componente
- Sulla struttura array, in genere, i linguaggi di tipo general-purpose definiscono solo alcune limitate operazioni specifiche quali operazioni di lettura e scrittura

Allocazione di array

- Allocazione statica
  - Le dimensioni sono definite attraverso costanti al tempo di compilazione e, pertanto, l'occupazione di memoria risulta fissa
  - Occorre definire la dimensione con riferimento al valore massimo occorrente per l'insieme ammissibile dei valori
  - Per ciascuna esecuzione le dimensioni effettive non devono eccedere le dimensioni massime
- Allocazione dinamica
  - Le dimensioni sono definite dinamicamente al tempo di esecuzione
  - Non occorre più distinguere tra dimensioni massime e dimensioni effettive, in quanto per ciascuna esecuzione può essere allocato il numero di elementi strettamente necessario

Allocazione statica vs. dinamica

- Non tutti i linguaggi di programmazione prevedono entrambe le modalità
- L'allocazione statica è caratterizzata da:
  - Un numero fisso predefinito di componenti
  - Un'allocazione non ottimale della memoria disponibile
  - Semplicità e velocità di esecuzione
- L'allocazione dinamica è caratterizzata da:
  - Un numero variabile di componenti
  - Uso efficiente della memoria
  - Un appesantimento al tempo di esecuzione

Gli array in C++

```
tipo nome_array[dimensione]
```

- Esempio: dichiarazione di un array di dieci elementi interi

```
int sample[10];
```

  - È possibile accedere ad un singolo elemento dell'array mediante un indice, che descrive la posizione di un elemento nell'array stesso

```
sample[0]; //primo elemento
sample[j]; //elemento j+1-esimo
sample[9]; //ultimo elemento
```

Gli array in C++

- La sintassi per la generica dichiarazione di un array è quindi:

  `<NomeTipo> <Identificatore> [<NumeroElementi>];`
- Tipo può essere sia un tipo primitivo che uno definito dal programmatore
- Identificatore è un nome scelto dal programmatore per identificare l'array
- **`NumeroElementi`** deve essere un intero positivo e indica il numero di singole variabili che compongono l'array
- Il generico elemento dell'array viene selezionato con la notazione **`Identificatore[Espressione]`**, dove **`Espressione`** può essere una qualsiasi espressione che produce un valore intero
- Il primo elemento di un array è sempre **`Identificatore[0]`**, e di conseguenza l'ultimo è **`Identificatore[NumeroElementi-1]`**

Gli array in C++

- Esempio: array monodimensionale

```
float Pippo[10];  // Dichiara un array di 10
                  // elementi di tipo float
float Pluto;      // Dichiara una variabile di
                  // tipo float
Pippo[0] = 13.5;  // Assegna 13.5 al primo
                  // elemento
Pluto = Pippo[9]; // Seleziona l'ultimo elemento
                  // di Pippo e lo assegna a
                  // Pluto
```

Gli array in C++: inizializzazione aggregata

- È possibile specificare i valori iniziali dei singoli elementi dell'array mediante *l'inizializzazione aggregata*:

```
int Pippo[5] = { 10, -5, 6, 110, -96 };
```

- Nel caso di array monodimensionali, è possibile omettere l'indice. Il compilatore, infatti, è in grado determinare il numero di elementi in base al numero di valori forniti per l'inizializzazione:

```
float Pluto[ ] = { 1.1, 3.5, -10.5 }
```

Caricamento e visualizzazione di array in C++

- È possibile utilizzare il ciclo for…

```
int vett[10];

//caricamento vettore
for (int i=0; i<10; i++) {
   cin >> vett[i];
}

//stampa vettore
for (int i=0; i<10; i++) {
   cout << vett[i] << " " << endl;
}
```

Memorizzazione di array in C++

- In C++ tutti gli array consistono di locazioni di memoria contigue. Tutti gli elementi di un array, pertanto, si trovano in memoria uno accanto all'altro
- numero di byte occupati = numero byte tipo base x numero elementi dell'array

Array in C++: errori frequenti

- In C++ non è possibile assegnare un array ad un altro
  - La seguente istruzione è illegale:

```
int a[10], b[10];
…
a = b; // errore!
```

- Il C++ non effettua alcun controllo sui limiti degli array
  - Non c'è nulla che eviti lo *sconfinamento* alla fine di un array
  - Il programmatore deve aggiungere un controllo d'errore adeguato a seconda delle necessità

Variabili array e Tipo array

- L’istruzione **`int a[10];`** dichiara una variabile di tipo "array di 10 interi" di nome "a"
- All’atto della dichiarazione della variabile viene allocato in memoria uno spazio pari a quello necessario per contenere 10 variabili di tipo intero
- Se si vuole dichiarare una nuova variabile "b" (array di 10 interi) si deve utilizzare la medesima istruzione: **`int b[10]`**

La parola chiave typedef ed il tipo array

- È possibile, utilizzando la parola chiave **`typedef`**, definire un tipo "array di interi"

```
typedef int TArrayDiInteri[10];
```

- Si definisce un nuovo tipo, chiamato TArrayDiInteri, costituito da variabili di tipo array di 10 interi
- All'atto di questa istruzione non avviene alcuna allocazione di memoria. Si sta solo definendo un "alias" per un tipo
- Volendo dichiarare le due variabili a e b (array di 10 interi), si userà l'istruzione

```
TArrayDiInteri a, b;
```

La parola chiave typedef

- In C++ esiste la possibilità di dichiarare un *alias* per un altro tipo (non un nuovo tipo) utilizzando la parola chiave typedef

```
typedef < Tipo > < Alias > ;
```

- Ad esempio:

```
typedef unsigned short int TPiccoloIntero;
typedef long double TArrayDiReali[20];
```

- Questo modo di procedere rende più intuitiva la programmazione e in taluni casi facilita il compito del programmatore

Dati strutturati

- Gli array permettono di raccogliere sotto un unico nome più variabili omogenee e sono solitamente utilizzati quando bisogna operare su più valori dello stesso tipo contemporaneamente
- Tuttavia in generale per rappresentare entità complesse è necessario memorizzare informazioni di diversa natura
  - Ad esempio per rappresentare uno studente può non bastare una stringa per il nome ed il cognome, ma potrebbe essere necessario memorizzare anche la matricola e altre informazioni
- Avere variabili distinte per le singole informazioni non è certamente una buona pratica, diventa difficile capire qual è la relazione tra le varie componenti
- La soluzione consiste nel raccogliere le variabili che modellano i singoli aspetti in un'unica entità che consenta, però, ancora di accedere ai singoli elementi

Il tipo Record

- Il tipo Record è una struttura dati ad accesso casuale costituita da un numero fisso di componenti, detti anche campi del record, che possono essere di tipo semplice o a loro volta strutturato
- Svolge una generica funzione di raggruppamento di più attributi relativi alla medesima entità
- Ciascuno dei campi del record ha un proprio nome ed è singolarmente accessibile mediante composizione del suo nome con il nome del record (funzione di accesso)

Record in C++

- La sintassi per la dichiarazione di una struttura (Record) in C++ è la seguente:

```
struct < NomeTipo > {
< Tipo > < NomeCampo > ;
/* ... */
< Tipo > < NomeCampo > ;
};
```

- Si osservi che la parentesi graffa finale deve essere seguita da un punto e virgola
- Esempio:

```
struct Persona {
char Nome[20];
unsigned short int Eta;
char CodiceFiscale[16];
};
```

Funzione di accesso al Record

- I singoli campi di una variabile di tipo struttura sono selezionabili tramite l'operatore di selezione “.” (punto)
- Le operazioni applicabili al tipo sono in genere quelle di inizializzazione ed assegnazione di valore

Record in C++: esempio 1

```
struct Persona {
  char Nome[20];
  unsigned short int Eta;
  char CodiceFiscale[6];
};

Persona Pippo = { "Pippo", 40, "PPP718" };
Persona AmiciDiPippo[2] = { {"Pluto", 40,
"PLT712"}, {"Minnie", 35, "MNN431"} };

// esempi di uso di strutture:
Pippo.Eta = 41;
unsigned short int Var = AmiciDiPippo[0].Eta;
```

Record in C++: esempio 1

- Si dichiara il tipo record Persona
- Si dichiara la variabile Pippo di tale tipo
- Si inizializza la variabile con una inizializzazione aggregata del tutto simile a quanto si fa per gli array, eccetto che i valori forniti devono essere compatibili con il tipo dei campi e dati nell'ordine definito nella dichiarazione
- Viene mostrata anche la dichiarazione di un array i cui elementi sono di tipo struttura, e il modo in cui eseguire una inizializzazione fornendo i valori necessari all'inizializzazione dei singoli campi di ciascun elemento dell'array
- Si accede ai campi di una variabile di tipo struttura
  - Si noti che prima viene selezionato l'elemento dell'array e poi il campo Nome di tale elemento

Record in C++: esempio 2

```
struct Data {
   unsigned short int Giorno, Mese;
   unsigned int Anno;
};

struct Persona {
   char Nome[20];
   Data DataNascita;
};

Persona Pippo = { "pippo", {10, 9, 1950} };

Pippo.Nome[0] = 'P';
Pippo.DataNascita.Giorno = 15;
unsigned short int UnGiorno = Pippo.DataNascita.Giorno;
```

Il tipo Record: assegnamento

- Per le strutture, a differenza degli array, è definito l'operatore di assegnamento

```
struct Data {
   unsigned short int Giorno, Mese;
   unsigned Anno;
};

Data Oggi = { 10, 11, 1996 };
Data UnaData = { 1, 1, 1995};

UnaData = Oggi; //OK!
```

Il tipo Record: assegnamento

- Se due strutture sono dello stesso tipo, è possibile assegnare il contenuto di una struttura all'altra
- L'assegnamento è ovviamente possibile solo tra variabili dello stesso tipo struttura, ma quello che di solito sfugge è che due tipi struttura che differiscono solo per il nome sono considerati diversi!

```
// con riferimento al tipo Data visto prima:
struct DT {
unsigned short Giorno, Mese;
unsigned int Anno;
};
Data Oggi = { 10, 11, 1996 };
DT Ieri;
Ieri = Oggi; // Errore di tipo!
```

Il tipo Record: la parola chiave typedef

- Anche nel caso del tipo Record è possibile utilizzare la parola chiave typedef
- In particolare l'utilizzo del typedef con la struct ha il seguente significato: prima si definisce una struttura anonima e poi l'uso di typedef crea un alias per quel tipo struttura

```
typedef struct {
   long double ParteReale;
   long double ParteImmaginaria;
} TComplesso;
```

Domande?