Strutture Dati

Luca Abeni

Dati e Tipi di Dati

- Una variabile è caratterizzata dal suo Tipo
  - Specifica i valori che può assumere (Insieme di Definizione)
  - Specifica a quanti byte di memoria la variabile è associata
  - Specifica come interpretare il valore di tali byte
- Fino ad ora abbiamo considerato insiemi di definizione "semplici"
  - Numeri naturali/interi
  - Approssimazioni di numeri reali
  - Caratteri

Tipi Fondamentali

- Tipi di dato "base" definiti dal linguaggio
- Se un linguaggio prevede la definizione di altri tipi di dato, si definiscono in termini di tipi base
- In C: `int`, `char`, `float`, `double`
- Per ogni tipo fondamentale, il linguaggio definisce una serie di operazioni...
  - Il tipo specifica anche come interpretare i bit che codificano un dato...
  - ...quindi specifica anche come operare su di essi!

Operazioni Fondamentali

- Operatori fondamentali definiti su variabili:
  - Assegnamento (=)
  - Confronto (==)
    - Per alcuni tipi di dato, oltre all'uguaglianza == è definito anche l'ordinamento ($<$, $\leq$, $>$, $\geq$)
    - Tipi di dato *scalari*
- Tipo di dato scalare
  - Composto da un unico valore
  - Si contrappone a tipi "strutturati"

Esempio: Ordinamento di Numeri

- A cosa servono i tipi di dato strutturati? Non bastano i tipi scalari?
  - Capiamolo con un esempio...
- Problema: dato un insieme di numeri naturali, ordinarli
- Possibile soluzione: Ordinamento per Inserimento Diretto
  1. Scorrere tutti i numeri dal secondo all'ultimo
  2. Per ogni numero considerato, inserirlo nel giusto ordine fra i precedenti
- Descrizione molto informale...
  - "*Scorrere tutti i numeri*"?
  - "*Per ogni numero considerato*"?
  - "*Inserire nel giusto ordine*"?

Ordinamento per Inserimento - Esempio

- Cerchiamo di capire meglio l'ordinamento per inserimento...
- ...tramite un esempio!

44	55	12	42	94	18	6	67
44	55	12	42	94	18	6	67
44	55	12	42	94	18	6	67
12	44	55	42	94	18	6	67
12	42	44	55	94	18	6	67
12	42	44	55	94	18	6	67
12	18	42	44	55	94	6	67
6	12	18	42	44	55	94	67
6	12	18	42	44	55	67	94

Formalizziamo l'Algoritmo - 2

- Contatore `i`: posizione del numero che stiamo considerando
- Per ogni valore di `i` (da 2 a n), considera i numeri con posizione $<$ `i`
  - Altro contatore `j`, che va da `i - 1` a 1 (conta indietro)
- Se il numero in posizione `j` è $>$ del numero in posizione `i`, sposta tale numero a destra
- Se il numero in posizione `j` è $<$ del numero in posizione `i`, ferma il contatore `j` ed inserisci in posizione `j + 1` il numero che era in posizione `i`

Formalizziamo l'Algoritmo - 1

- Variabili: i, j. Insieme di definizione: numeri naturali
- Inoltre, appoggio (intero) e numeri da ordinare...
- `i = 2`
- mentre `i` $\leq n$
  - `appoggio` = numero in posizione `i`
  - `j = i - 1`
  - mentre `j` > 0 e numero in posizione `j` $>$ `appoggio`
    - numero in posizione `j + 1` = numero in posizione `j`
    - `j = j - 1`
  - `i = i + 1`
  - numero in posizione `j + 1` = `appoggio`

Formalizziamo l'Algoritmo - 3

- "numero in posizione `i` / `j`"?
  - Formalizziamo un po' meglio questa cosa...
- Usiamo un nome per identificare tutti i numeri...
  - Se un numero ha insieme di definizione $\mathcal{Z}$, questa variabile ha insieme di definizione $\mathcal{Z}^n$
- ...ed identifichiamo il singolo numero tramite un indice
  - In matematica, $\texttt{numero}_i$, $\texttt{numero}_j$
  - Ma usiamo qualche simbolo che sia meglio rappresentabile sui computer
  - `numero[i]`, `numero[j]`

Formalizziamo l'Algoritmo - 4

- Variabili: `i`, `j` $\in \mathcal{N}$; `appoggio` $\in \mathcal{Z}$ e `numero` $\in \mathcal{Z}^n$
- `i = 2`
- mentre `i` $\leq n$
  - `appoggio` $=$ `numero[i]`
  - `j = i - 1`
  - mentre `j` > 0 e `numero[j]` $>$ `appoggio`
    - `numero[j + 1] = numero[j]`
    - `j = j - 1`
  - `i = i + 1`
  - `numero[j + 1] = appoggio`

Array

- `numero` $\in \mathcal{Z}^n$
  - Variabile con insieme di definizione $\mathcal{Z}^n$
  - Composta da n variabili con insieme di definizione $\mathcal{Z}$ (variabili intere)
  - Singoli elementi accessibili come `numero[i]`
- Abbiamo appena scoperto gli Array!
- Array: tipo *strutturato* composto da variabili di un tipo semplice ripetute più volte
- Tutti gli elementi dell'array hanno lo stesso tipo

Tipi Strutturati

- Tipi di dato strutturati (o aggregati)
  - Composizioni di tipi di dato più semplice
  - I tipi "composti" in un tipo strutturato possono essere tipi fondamentali o di altro genere...
  - ...Anche tipi a loro volta strutturati!
  - Ma andiamo con ordine
- Vari modi di comporre tipi di dato più semplici in tipi più complessi
  - Array: collezioni di variabili dello stesso tipo
  - Record / Strutture: collezioni di variabili ti tipo diverso
  - Record o strutture possono essere varianti

Tipi Strutturati - Perché?

Algorithms + Data Structures = Programs (Algoritmi + Strutture Dati = Programmi)

- Un programma non è solo una sequenza di istruzioni...
- ...Ma è anche caratterizzato dai dati che tali istruzioni modificano!
  - E chiaramente dal tipo di tali dati...
- Come la programmazione strutturata cerca di "fare ordine" nell'algoritmo...
- ...Così può essere utile "fare ordine" fra i dati
  - Organizzandoli tramite tipi più complessi

Definire un Tipo di dato Strutturato

- Tipo di dato strutturato:
  - Descrizione di come combinare i tipi di dato più semplici
    - Array, Struttura, ...
  - Operazioni per manipolare tali “combinazioni di dati”
    - Modificare il valore
    - Leggere il valore
    - ...
- Accedere direttamente ai vari “componenti” del dato potrebbe comprometterne la strutturazione

Array

- Array: insieme ordinato (n-upla) di elementi *omogenei*
  - Tutti dello stesso tipo
  - n: numero di elementi che compongono l'array
- Caratteristiche:
  - Omogeneità
    - Visibile come n variabili (elementi dell'array) uguali fra loro
  - Ordinamento
    - Ogni elemento è identificato da un numero intero (indice)
- Ogni elemento dell'array è accessibile (leggibile / modificabile) indipendentemente dagli altri...
- Visibile come un gruppo di variabili aventi un nome parametrizzato (parametro: indice i)

Array e Spazi Vettoriali

- Array: simile al concetto matematico di Vettore
  - Tipo base con insieme di definizione $\mathcal{I} \rightarrow$ un array di n elementi ha insieme di definizone $\mathcal{I}^n$
- Array di dimensioe n: $v \in \mathcal{I}^n$
  - Insieme di definizione: spazio vettoriale di dimensione n
  - Valore assunto dalla variabile: vettore $(v_1, v_2, \ldots v_n)$
- Notazione informatica: `v[i]` $\equiv v_i$

Array e Variabili

- Ricordi? Variabile = coppia (nome, valore)
  - In caso di array, coppia (nome, lista di valori)
  - Ogni valore è identificato non solo dal nome, ma da nome e indice
- In memoria, nome associato ad una o più locazioni che contengono il valore...
  - In caso di array, nome associato ad una ripetizione di n zone di memoria che contengono un valore

$$\mathtt{n} \in \mathcal{N}\colon \mathtt{n} \rightarrow \boxed{\text{valore}}$$

$$\mathtt{n} \in \mathcal{N}^d\colon \mathtt{n} \rightarrow \boxed{\text{v1}}\boxed{\text{v2}}\boxed{\text{v3}}\cdots\boxed{\text{vd}}$$

Matrici

- Si possono definire array di array $\rightarrow$ matrici
- Due indici i e j invece di uno solo

$$\texttt{v[]} \rightarrow (v_1, v_2, \cdots v_d)$$

$$\texttt{m[][]} \rightarrow \begin{pmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,d} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ m_{k,1} & m_{k,2} & \cdots & m_{k,d} \end{pmatrix}$$

Record / Strutture

- Array: collezione di variabili dello stesso tipo
  - Omogeneità, ordinamento, ...
- Come aggregare logicamente dati di tipo diverso fra loro?
  - Esempio: variabile che rappresenta una persona:
    - nome
    - cognome
    - età
    - sesso
    - ...
  - Strutture! (o Record, dipende dalla terminologia)

Strutture: Perché?

- Persona: descrivibile tramite variabili distinte:
  - Nome $\in \mathcal{C}^n$ (dove $\mathcal{C}$ è l'insieme dei caratteri ($\mathcal{C}^n$ è un array di n caratteri)
  - Cognome $\in \mathcal{C}^n$
  - Anni $\in \mathcal{N}$
  - ...
- Cosa fare se devo rappresentare più persone?
  - Ogni variabile diventa un array!
  - Rischio di fare grossa confusione...

Persona Come Struttura

- Invece di descrivere una persona tramite più variabili distinte...
- Si può usare un'unica variabile, che aggrega più campi
- `Amico` $\in \mathcal{C}^n \times \mathcal{C}^n \times \mathcal{N} \times ...$
  - $\times$: prodotto fra insiemi
  - Variabile `Amico`: composta da un campo stringa (di n caratteri), un altro campo stringa, un numero naturale, ...
  - Ogni campo è identificato da un nome
- E se devo rappresentare più persone?
  - Ho più di un amico...
  - La variabile `Amico` diventa un array!
  - Il rischio di fare confusione è molto minore...

Strutture e Array

- Array: elementi omogenei ordinati
  - Ciascun elemento è accessibile tramite il suo indice
  - Esempio: `v[i]` è l'`i`-esimo elemento dell'array `v`
- Struttura: campi eterogenei
  - I campi non sono ordinati
  - Ciascun campo della struttura ha un nome
  - I campi sono accessibili singolarmente tramite i loro nomi
  - `persona.anni` è il campo chiamato `anni` della struttura `persona`
- Utile dare un nome ad una struttura composta da determinati campi
  - Esempio: `Persona` $= \mathcal{C}^n \times \mathcal{C}^n \times \mathcal{N} \times ...$, `Amico` $\in$ `Persona`

Definizione di Strutture

- Per poter usare una variabile di tipo struttura, devo definire come tale struttura è fatta
  - Esempio: la variable `Amico` è una struttura `Persona`
  - Devo definire prima la struttura `Persona`
- Definizione di struttura: si definiscono tutti i suoi campi, specificandone nome e tipo
  - Struttura `Persona` =
    - Nome $\in \mathcal{C}^n$
    - Cognome $\in \mathcal{C}^n$
    - Anni $\in \mathcal{N}$
    - ...

Definizione di Strutture in C

- Usiamo la sintassi del linguaggio C
  - `struct` *identificatore* { *definizione dei campi* };
  - Definizione dei campi: come "normali" variabili
- Struttura `persona` composta da due campi stringa (array di 20 caratteri) e tre campi interi positivi:

```
struct persona {
    char nome[20];
    char cognome[20];
    unsigned int anno_di_nascita;
    unsigned int mese_di_nascita;
    unsigned int giorno_di_nascita;
};
```

Uso di Strutture in C

- Definizione di variabile di tipo struttura:
  - `struct` *identificatore struttura identificatore variabile*;
  - Esempio:
    - `struct persona amico;`
  - Definisce una variabile "`amico`" di tipo "`struct persona`" (struttura definita nella slide precedente)
  - La struttura "`struct persona`" deve essere stata definita prima di poter definire la variabile "`amico`"
- Il tipo di dato è "`struct` *identificatore struttura*"
- Come al solito, dichiarazione (`struct persona;`) vs definizione (`struct persona { ...};`)

Esempio

```c
#include <stdio.h>
#include <math.h>

struct punto {
  double x;
  double y;
};

int main()
{
  struct punto p1, p2;
  double distanza;

  p1.x = 1; p1.y = 1;
  p2.x = 5; p2.y = -2;

  distanza = sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
  printf("Distanza fra (%f,%f) e (%f,%f): %f\n", p1.x, p1.y, p2.x, p2.y, distanza);

  return 0;
}
```

Array di Strutture

- Le strutture sono spesso usate in array:

```
struct persona {
    char nome[20];
    char cognome[20];
    unsigned int anno_di_nascita;
    unsigned int mese_di_nascita;
    unsigned int giorno_di_nascita;
};
...
int main()
{
  struct persona amici[20];
  unsigned int i;

  ...
  i = 0;
  while (i < 20) {
    printf("Nome dell'amico numero %d: %s\n", i, amici[i].nome);
  }i

  ...
}
```