Costrutti di base per strutture dati

struct, puntatori, array

Lezione 7 di Programmazione 2

Docente: Giuseppe Scollo

Università di Catania, sede di Comiso (RG)
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Studi in Informatica applicata, AA 2006-7

Indice

strutture dati eterogenee: `struct`

struttura dati **eterogenea** : aggregato di valori di tipi possibilmente diversi

costrutti nei linguaggi di programmazione: **`struct`** in C++, **`record`** in Pascal, etc.

in termini matematici, il tipo di dati di tali strutture è un **prodotto cartesiano** degli insiemi di valori dei tipi componenti, con **proiezioni :** funzioni di accesso ai valori dei componenti

nella terminologia OO i componenti di una struttura eterogenea sono detti **membri** della struttura, e si accede al loro valore attraverso il loro nome, che con la **dot notation** designa (in forma postfissa) la funzione di proiezione

ad es., la struttura dati di un punto nello spazio bidimensionale, con coordinate cartesiane, può essere specificata dalla seguente **definizione di tipo :**

```
struct punto2D {float x; float y; };
```

se **`p`** è una variabile di tipo **`punto2D`**, le sue proiezioni cartesiane sono **`p.x`** e **`p.y`**

esempio: metrica cartesiana

una più ricca struttura dati per i punti dello spazio bidimensionale, dotata non solo di proiezioni cartesiane ma anche di una funzione che calcola la distanza fra due punti, è specificata dalla seguente **interfaccia :**

```
struct punto2D {float x; float y; };
float distanza2D (punto2D, punto2D);
```

se **`Punto2D.h`** è il nome del file contenente questa interfaccia, la seguente è una sua **implementazione :**

```
#include <cmath>
#include "Punto2D.h"
float distanza2D(punto2D a, punto2D b)
  { float dx = a.x - b.x, dy = a.y - b.y;
    return sqrt(dx*dx + dy*dy);
  }
```

accesso indiretto ai dati: puntatori

se `t` è un tipo, `t*` designa il tipo dei **puntatori** o **riferimenti** a oggetti di tipo `t`

se `p` è una variabile di tipo `t*`, allora `*p` designa l'oggetto (di tipo `t`) a cui `p` fa riferimento

i puntatori sono di solito implementati come indirizzi di memoria: l'operatore prefisso "`&`" dà l'indirizzo dell'oggetto a cui è applicato, dunque `*&a` e `&*a` equivalgono ad `a`

il **passaggio dei parametri per riferimento** è utile ad almeno **due scopi :**

- permettere ad una funzione di produrre più valori, non nel suo valore di ritorno bensì in variabili condivise con il chiamante (parametri passati per riferimento)
- rendere efficiente il passaggio di parametri aventi struttura grande o complessa, evitandone la copia in variabili locali che si ha nel passaggio di parametri per valore

e in C++ può effettuarsi tecnicamente in **due modi :**

nel chiamante:	nella definizione di funzione:
`op(&a, &b)`	`op (ta *a, tb *b) { ... *a = ...; *b = ...; ... }`
`op(a, b)`	`op (ta& a, tb& b) { ... a = ...; b = ...; ... }`

esempio: coordinate polari

vediamo come si possa codificare una funzione che effettui la conversione da coordinate cartesiane a coordinate polari, con il passaggio dei parametri per riferimento nei due modi suddetti (e le funzioni `sqrt` e `atan2` di libreria):

- passaggio di parametri per **riferimento indiretto** (passaggio per valore di un **puntatore**):

```
polar (punto2D c, float *r, float *theta)
  { *r = sqrt(c.x*c.x + c.y*c.y); *theta = atan2(c.y, c.x); }
```

- per ottenere nelle variabili `float a, b` le coordinate polari del `punto2D p` si invoca

```
polar(p, &a, &b);
```

- passaggio di parametri per **riferimento :**

```
polar (punto2D c, float& r, float& theta)
  { r = sqrt(c.x*c.x + c.y*c.y); theta = atan2(c.y, c.x); }
```

- per ottenere nelle variabili `float a, b` le coordinate polari del `punto2D p` si invoca

```
polar(p, a, b);
```

strutture dati omogenee: `array`

struttura dati **omogenea** : aggregato di valori dello stesso tipo

gli `array` sono strutture dati ad **accesso diretto**, diffuse nei linguaggi di programmazione

in termini matematici, il tipo di dati di tali strutture è quello delle **funzioni finite** (viste come oggetti), con essenzialmente due operazioni:

lettura dell'elemento di dato indice
cioè **applicazione** della funzione in un punto del dominio

scrittura dell'elemento di dato indice
cioè **modifica** della funzione in un punto del dominio

di solito il tipo indice è **totalmente ordinato**, per il qual motivo si pensa spesso (alquanto impropriamente) agli array come a strutture dati sequenziali

in C++ il tipo indice è il tipo elementare predefinito `int`, limitato a valori **non negativi**, e le locazioni di memoria degli elementi sono contigue, in ordine d'indice

sintassi: `t a[N]` è la dichiarazione di un array `a` di al più `N` elementi di tipo `t`
`a[i]` è l'elemento di indice `i` in tale array, per `0 ≤ i < N`

esempio: crivello di Eratostene

algoritmo, risalente al **III secolo a.C**, che "setaccia" i numeri primi minori di **N**

come funziona: per **1 < i < N**, si eliminano i multipli di **i** minori di **N**, a partire da i^2

```
#include <iostream>
using namespace std;
static const int N = 1000;
int main()
  { int i, a[N];
    for (i=2; i < N; i++) a[i] = 1; // inizializzazione
    for (i=2; i < N; i++)
      if (a[i])
         for (int j = i; j*i < N ; j++) a[j*i] = 0; // eliminazione non primi
    for (i=2; i < N; i++)
      if (a[i]) cout << " " << i; // output i primo
    cout << endl;
  }
```

complessità dell'algoritmo: proporzionale a **N lg N**

si esaminano **N-2** elementi dell'array e se ne azzerano all'incirca **N/2 + N/3 + N/5 + ...** : la somma è approssimabile per eccesso con $N H_N$, cioè all'incirca **N ln N**, come si è visto in una lezione precedente

array e allocazione dinamica di memoria

nel programma precedente l'allocazione di memoria all'array è **statica**, cioè nota a tempo di **compilazione**: ciò ha qualche vantaggio, però... se si vuole generare un maggior numero di primi, occorre modificare il valore della costante **N** nel sorgente e ricompilare :(

in C++ è possibile l'allocazione **dinamica** della memoria per un array, dove ad es. la dimensione dell'array è determinata da linea di comando a tempo di **esecuzione**

a tal scopo si può adoperare l'operatore `new t [d]` che genera un array di `d` elementi di tipo `t` e restituisce un **puntatore** al(primo elemento del)l'array

> ciò è coerente con il fatto che, in C++, il nome di un array è un puntatore al suo primo elemento (di indice `0`); tenendo conto della contiguità di memorizzazione e della "aritmetica dei puntatori", in C++ le espressioni `*(a+i)` e `a[i]` sono equivalenti

una modifica del programma precedente per il dimensionamento dinamico dell'array consiste nel sostituire le tre righe che precedono il ciclo di inizializzazione con le seguenti:

```
int main (int argc, char *argv[])
  { int i, N = atoi(argv[1]);
    int *a = new int[N];
```

esercizi

1. definire interfaccia e implementazione del tipo di dato dei punti nello spazio cartesiano tridimensionale, con una funzione che calcoli la distanza di due punti
2. definire, con una `struct`, interfaccia e implementazione del tipo di dato dei punti nel piano in coordinate polari, con una funzione che calcoli la distanza di due punti

   > (l'implementazione può ben essere client del tipo di dato che specifica i punti in coordinate cartesiane e le usa per il calcolo della distanza)
3. definire interfaccia e implementazione del tipo di dato dei punti nello spazio cartesiano a **N** dimensioni, con una funzione che calcoli la distanza fra due punti, dove **N** sia un parametro del tipo di dato; inoltre, definire un client che accetta **N** da linea di comando, quindi genera due punti con coordinate pseudocasuali nell'intervallo [0, 1] e produce in output coordinate e distanza dei due punti

   > (nell'interfaccia si rappresenti il punto come una ***struct*** con due membri: **N** e un puntatore alle coordinate; nell'implementazione si assuma che queste siano memorizzate in ***array***; il client allochi dinamicamente la memoria per esse)
4. il testo (Sedgewick, 2003) propone un programma per la simulazione del lancio di monete (Programma 3.7, p. 88), che però contiene un errore: trovarlo.