Tipi definiti dall'utente

Anna Corazza

aa 2023/24

Dove studiare

- Str'13, sezione 2.3, 8

Str'13 Bjarne Stroustrup, The C++ Programming Language (4th edition), 2013
`https://www.stroustrup.com/4th.html`

Tipi di dato definiti dall'utente

- Finora abbiamo visto tipi **built-in**, che sono di livello piuttosto basso, vicino al livello macchina, un po' scomodi per programmare.
- **Meccanismi di astrazione** per costruire dei tipi più vicini alle esigenze del programmatore: i tipi **definiti dall'utente** (user-defined types).

Strutture

struct

```
struct Vector{
  int sz;  // numero di elementi
  double* elem; // punt. ad array di elementi
};
```

- È una prima versione di un tipo per una miglior gestione dei vettori.
- Possiamo dichiarare una variabile:

```
Vector v;
```

Strutture

Inizializzazione

```
struct Vector{
  int sz;  // numero di elementi
  double* elem; // punt. ad array di elementi
};
```

- Definire una funzione per inizializzare l'array di elementi:

```
void vector_init(Vector& v, int s){
  v.elem = new double[s];
  v.sz = s;
}
```

- Allochiamo l'array `elem` sullo heap.
- Si noti che il primo argomento (`Vector& v`) è passato per riferimento, in modo da poterlo modificare.

Strutture

Utilizzo del vettore

```
double read_and_sum(int s){
  // legge s interi da cin e restituisce la
     somma
  // s positivo
  Vector v;
  vector_init(v,s);
  for(int i=0; i<s; i++)
    cin >> v.elem[i];

    double sum=0;
    for(int i=0; i<s; i++)
       sum+=v.elem[i];
    return sum;
```

- Il programmatore deve conoscere la **struttura interna** dell'oggetto!

Classi

- Una classe ha un insieme di membri: dati, funzioni o tipi.
- La sua interfaccia è definita dai suoi membri pubblici:
- i membri privati sono accessibili solo tramite l'interfaccia

```
class Vector{
  public:
     Vector(int s) :elem{new double[s]}, sz{s} {
         } // costruttore
     double& operator[](int i) { return elem[i];}
     int size() { return sz; }
  private:
     double* elem;
     int sz;
};

Vector v(6);
```

Classi

Discussione

- In un certo senso, l'oggetto `Vector` è una specie di maniglia (handle) che permette di elaborare l'array `elem` congiuntamente alla sua dimensione `sz`.
- Istanze diverse possono avere un diverso numero di elementi, e ogni istanza può avere un diverso numero di elementi in punti diversi del programma.
- Tuttavia, tutti gli oggetti `Vector` avranno la stessa dimensione
- Tecnica generale: una "maniglia" con una dimensione fissa che si riferisce a dei dati di dimensione variabili allocati "altrove" (in questo caso sul free store via `new`).

Classi

Utilizzo del vettore

```
double read_and_sum(int s){
  // legge s interi da cin e restituisce la
     somma
  // s positivo
  Vector v(s);
  for(int i=0; i<s; i++)
    cin >> v[i];

    double sum=0;
    for(int i=0; i<s; i++)
       sum+=v[i];
    return sum;
```

- La struttura interna del vettore rimane nascosta.
- Notate come l'utilizzo del costruttore risolve il problema delle variabili non inizializzate.

Enumerazioni

```
enum class Color {red,blue,green};
enum class Traffic_light {green,yellow,red};

Color col=Color::red;
Traffic_light light=Traffic_light::red;
```

- Le enumerazioni vengono usate per rappresentare piccoli insiemi di valori interi.
- `red`: enumeratore
- Vivono all'interno dello scope (ambito di definizione) della loro `enum class`, cosicché lo stesso valore può venir usato in `enum class`es senza confusione.

Enumerazioni

enum class

```
enum class Color {red,blue,green};
enum class Traffic_light {green,yellow,red};
```

- La parola chiave `class` che segue `enum` specifica che:
  - questa enumerazione è fortemente tipata
  - i suoi enumeratori hanno un ambito di definizione

```
Color x=red;          // NO: quale red?
Color y=Traffic_light::red; // NO: tipo
    sbagliato!
Color z=Color::red;   // OK
```

- Analogamente non si possono mescolare i valori in `Color` con gli interi:

```
int i=Color::red;   // No, tipo sbagliato
Color c=2;          // No, tipo sbagliato
```

Enumerazioni

`enum` semplice

- Se tolgo `class`, allora non ho più queste proprietà:
  - i valori diventano semplici interi
  - i nomi non sono qualificati

Enumerazioni

`enum class`: operatori

- Per default, un `enum class` ha solo assegnamento, inizializzazione e confronto (==,<,...)
- Tuttavia, essendo un tipo definito dall'utente, si possono definire anche dei nuovi operatori:

```
Traffic_light& operator++(Traffic_light&
    t){
  // prefix increment ++
  switch(t){
    case Traffic_light::green: return
        t=Traffic_light::yellow;
    case Traffic_light::yellow: return
        t=Traffic_light::red;
    case Traffic_light::red: return
        t=Traffic_light::green;
  }
}
```