02_Decl_vs_Def

Table of contents

Dichiarazione vs Definizione

Tipi di dato

- **Dichiarazione** intesa come costrutto del linguaggio che introduce un nome per una entità (abbiamo visto l'esempio di `cout` in `HelloWorld.cpp`). La struttura non la conosco, non posso creare oggetti di tipo `S`.

```
// dichiarazione pura del tipo S
struct S;
```

- **Definizione** sottintesa come dichiarazione (siccome introduce un nome), per fornire ulteriore elemento per caratterizzare l'entità (implementazione di `foo()`). Di `T` conosco la struttura e posso creare oggetti di quel tipo.

```
// definizione del tipo T
struct T { int a; };
```

Essendo linguaggio denso, C++ insiste sul fatto di usare gli stessi caratteri per indicare situazioni totalmente diverse. Nel caso sotto, il compilatore si può chiedere:

```
nome1 * nome2
```

1. `nome1` è puntato dal puntatore `nome2`?
2. operatore binario (moltiplicazione) tre i due valori?

Al compilatore servono indizi sui tipi delle variabili dichiarate (per default assume sia valore), altrimenti ambiguità sussistono alla compilazione, che non andrebbe a buon fine.

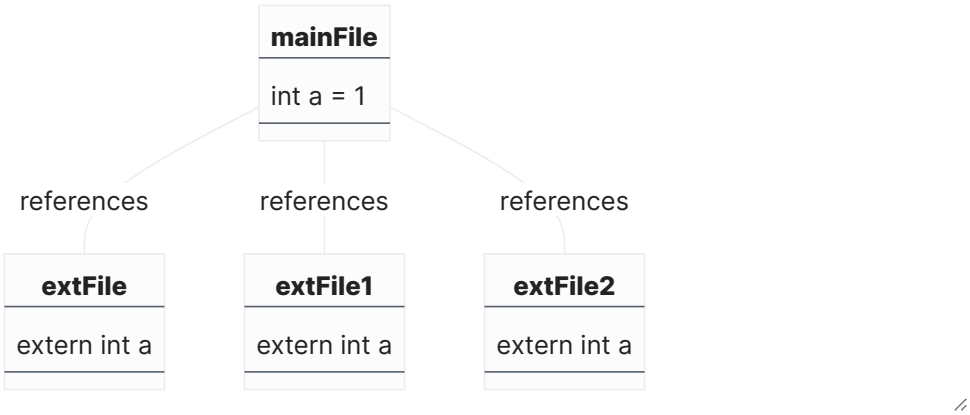
> `enum` del C++ 2011
>
> Con lo standard introdotto nel 2011, il C++ implementa la possibilità di *dichiarare puramente* un tipo di enumerazione che prima necessitava obbligatoriamente la definizione
>
> ```
> enum E : int; // opaque enum declaration
> enum E : int { a, b, c, d }; // definition
> ```

Variabili

- dichiarazione pura

```
extern int a; // dichiarazione pura di variabile (globale)
```

Il compilatore sa che esiste una variabile di tipo `int` a nome `a`, che tuttavia ha forma (verrà creata) in qualche altra parte del codice (unità di traduzione)



`extern "C"`

Una situazione simile con l'uso della parola chiave `extern` è il caso in cui volessimo riferirci a una variabile/funzione, dichiarata in un'altra un'unità di traduzione, che usa la convenzione del linguaggio `"C"`

```
// dichiara variabile globale con C linkage
extern "C" int errno;
```

- definizione

```
int b; // zero-inizialization
int c = 1;
extern int d = 2; // definizione, perché inizializzata
```

Vengono create le variabili accordate, inizializzate se richiesto

Funzioni

- dichiarazione pura

```
void foo(int a);
extern void foo(int a);
```

Commentiamo il fatto che `foo()` ha due dichiarazioni.
Il *numero* e *tipo* di argomenti della funzione, sono identificanti la stessa: questo permette di distinguere univocamente funzioni che hanno lo stesso nome (cosa a cui al compilatore non importa). Detto "bruttamente": le dichiarazioni pure possono ripetersi tante volte, mentre le definizioni sono uniche (One Definition Rule).

- definizione

```
void foo(int a) { // presente il corpo, quindi definizione
  std::cout << a;
}
```

Hanno un corpo

Template

- dichiarazione pura

```
template <typename T> struct S;
```

- definizione di template di classe

```
template <typename T>
struct S {
  T t;
};
```

- dichiarazione pura di template di funzione

```
template <typename T>
T add(T t1, T t2);
```

- definizione di template di funzione

```
template <typename T>
T add(T t1, T t2) {
  return t1 + t2;
}
```

28-02-2023