Le funzioni in C++

Liceo G.B. Brocchi
Classi seconde Scientifico – opzione scienze applicate
Bassano del Grappa, Novembre 2022

Motivazioni

- Immaginate di dover scrivere un programma che deve calcolare diverse volte la potenza *b^e*

```
int main(){
        //need to compute 3^4
        double result = 1;
        double b = 3;
        for (i = 1; i <= 4; i++){
                result = result * 3;
        }
        //do stuff
        //need to compute 4^5
        //too much copying and pasting!
        result = 1;
        for (i = 1; i <= 5; i++){
                result = result * 4;
        }
        //do stuff
        //need to compute 10^3
        //too much copying and pasting!
        result = 1;
        for (i = 1; i <= 3; i++){
                result = result * 10;
        }
}
```

Motivazioni

- Per organizzare meglio il codice (che può diventare di dimensioni notevoli, anche migliaia di righe di codice) sarebbe meglio creare delle piccole porzioni di codice riutilizzabili secondo le necessità del programma
- Ad esempio, vorremmo poter scrivere il codice che calcola b^e una sola volta e poi riutilizzarlo (**richiamarlo**)
- Una cosa che può essere **chiamata** deve avere un nome e chi la chiama deve conoscere almeno la funzionalità che offre, anche se non sa esattamente come la implementa
- Ad esempio la potenza *b^e* può essere calcolata con un ciclo while o con un ciclo for… con delle variabili con un certo nome etc…
- Ma chi ha bisogno di calcolare *b^e* vorrebbe semplicemente una black box che fa il calcolo giusto, anche senza sapere come lo fa

Motivazioni

- I programmi scritti nei linguaggi *procedurali* come il C consistono di tanti piccoli pezzi di codice che si occupano di realizzare funzionalità specifiche
- Si organizza il codice in questo modo perché è molto più facile manutenere tanti piccoli pezzi di codice che fanno cose specifiche rispetto ad un unico blocco (*monolite*) di codice che fa tutto
- Finora abbiamo scritto codice *monolitico*, tutto contenuto nel *main*
- Se le dimensioni del codice crescono, utilizzare solo il *main* diventa ingestibile
- Pensate alle funzioni di un foglio di calcolo:
  - la funzione RADQ di Excel richiede un parametro numerico e restituisce un valore numerico. Ma sappiamo come è realizzata internamente RADQ? **NO**

Motivazioni

- Questi pezzi di codice che fanno una cosa specifica vengono detti **funzioni**
- Le funzioni devono specificare come si comportano, e vanno prima di tutto **dichiarate**
- Una funzione deve dichiarare all'esterno le seguenti cose:
    - il tipo della cosa che restituisce (ritorna) a chi la chiama
    - il proprio nome
    - i parametri che accetta (che costituiscono la **lista dei parametri formali**)

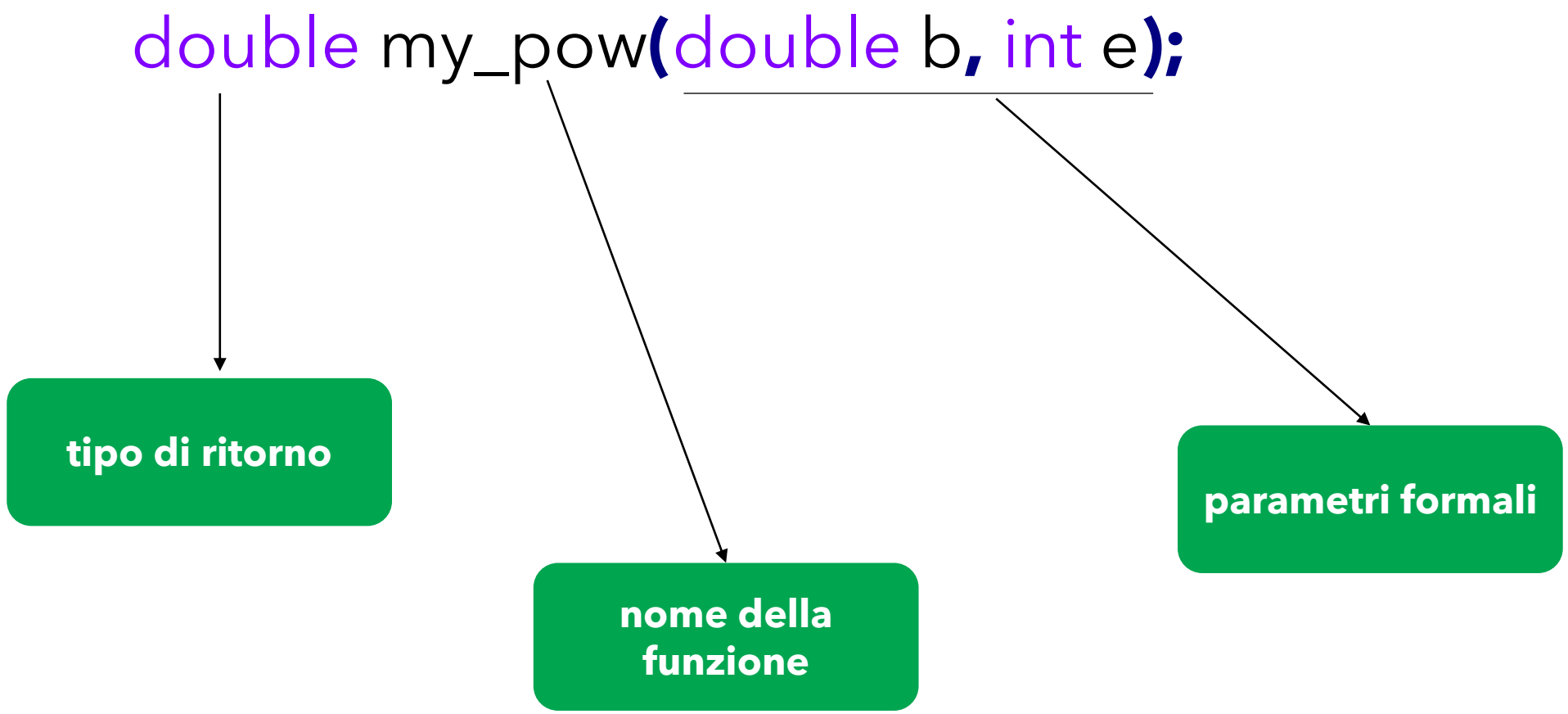
- Qual è il tipo del risultato restituito da una funzione che calcola la potenza *b*^*e*?
- Che nome le diamo?
- Quali sono i parametri che accetta?

Esempi di dichiarazione di funzione

```
double my_pow(double b, int e);
```

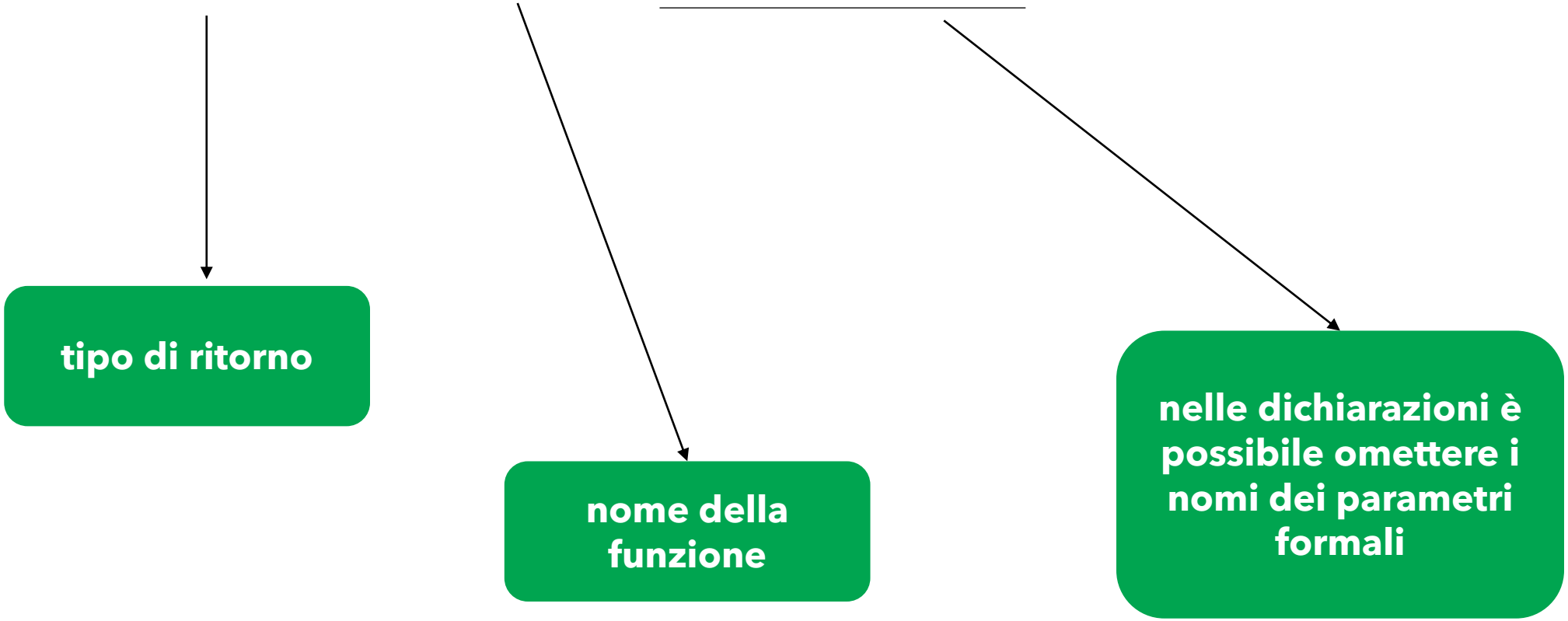
questa scrittura viene detta prototipo di funzione (o intestazione di funzione)
NB: non abbiamo ancora definito da nessuna parte le istruzioni
Stiamo solo specificando le informazioni che serviranno al chiamante per utilizzare my_pow

Esempi di dichiarazione di funzione



Esempi di dichiarazione di funzione

```
double my_pow(double, int);
```



Esempi di dichiarazione di funzione

```
double my_pow(double, int);
```

Vi sembrerà strano, ma possiamo dire che anche le funzioni hanno un «tipo»:
my_pow ha tipo:
- **ritorna (o «restituisce») double**
- **primo parametro double**
- **secondo parametro int**

Questo strano concetto di «tipo di una funzione» è un po' ostico da comprendere, ma ci permetterà di fare cose molto interessanti

Esempi di definizione di una funzione – l'algoritmo di Euclide, iterativo

```
int gcd_euclid(int a, int b) {
        while (b != 0) {
                int remainder = a % b;
                a = b;
                b = remainder;
        }

        return a;
}
```

a e **b** vengono detti **parametri formali** della funzione *gcd_euclid*

Esempi di chiamata di funzione – l'algoritmo di Euclide, iterativo

```
int main() {
    gcd_euclid(252, 105);

    return 0;
}
```

252 e 105 sono i **parametri attuali** della chiamata di funzione

Esempi di definizione di una funzione – stampa di n asterischi

```
void print_n_stars(int n) {
        for (int i = 1; i <= n; i++) {
                cout << '*';
        }
        cout << endl;
}
```

Esempi di chiamata di funzione – stampa di n asterischi

```
int main() {
    print_n_stars(7);

    return 0;
}
```