Università degli Studi di Ferrara

Corso di Laurea in Matematica - A.A. 2021 - 2022

Programmazione
Lezione 11 – Funzioni

Docente: Michele Ferrari - michele.ferrari@unife.it

Nelle lezioni precedenti

- Operatore Ternario: `max = a>b ? a : b;`
- Shift, rotazione, merge di un array
- Ricerca del minimo, ricerca di un valore
- Il problema dell'ordinamento (insertion sort, selection sort, bubble sort)
- Ricerca binaria
- Complessità computazionale
- Array bidimensionali (matrici) e multidimensionali
- Generare numeri casuali

In questa lezione

- Problemi e sotto problemi
- Strategia TOP DOWN
- Strategia BOTTOM UP
- Sottoprogrammi
- Le funzioni
- Funzioni ricorsive

Problemi e sotto problemi

Quando si ha a che fare con problemi semplici come il calcolo del massimo fra due numeri, l'algoritmo è facilmente individuabile.

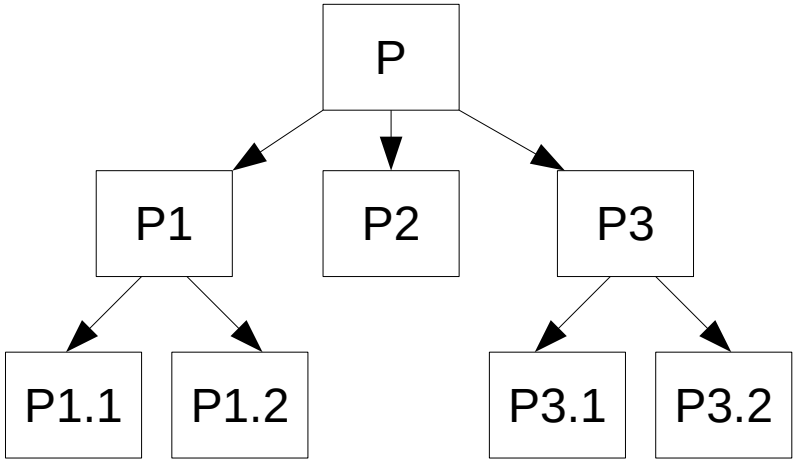
Quando invece si deve risolvere un problema più complesso è buona pratica semplificare il problema suddividendolo in problemi "più semplici" e progettare un algoritmo per ciascuno.

Nel caso il sotto problema risulti essere ancora un problema complesso, è possibile applicare ancora una volta il processo di scomposizione

Aumenta il numero di problemi ma diminuisce la loro complessità

Progettazione Top-Down

La strategia di progettazione TOP-DOWN consiste nel trasformare un problema in una gerarchia di problemi di difficoltà decrescente.



La progettazione TOP-DOWN

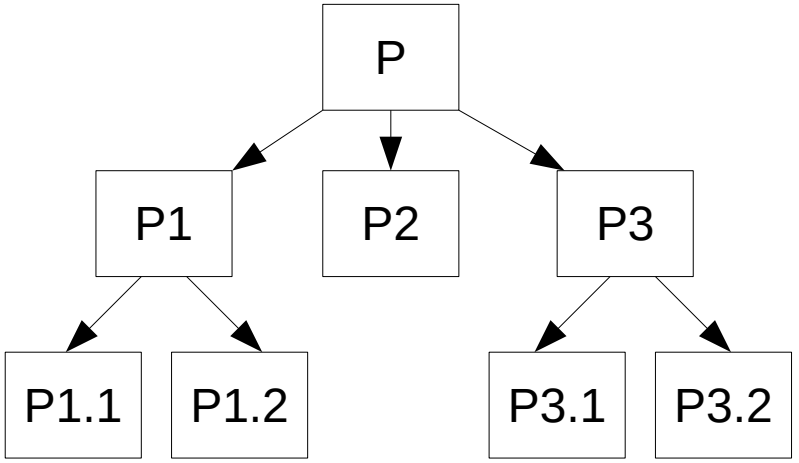
Questo approccio è detto anche “per raffinamenti successivi” grazie ai quali potremo ottenere la “giusta” descrizione del problema.

TOP => Alto livello di descrizione del problema

DOWN => Basso livello di descrizione del problema

Il processo di scomposizione dovrà continuare fino a che il livello di dettaglio raggiunto è sufficientemente espresso in termini di azioni elementari per l’esecutore.

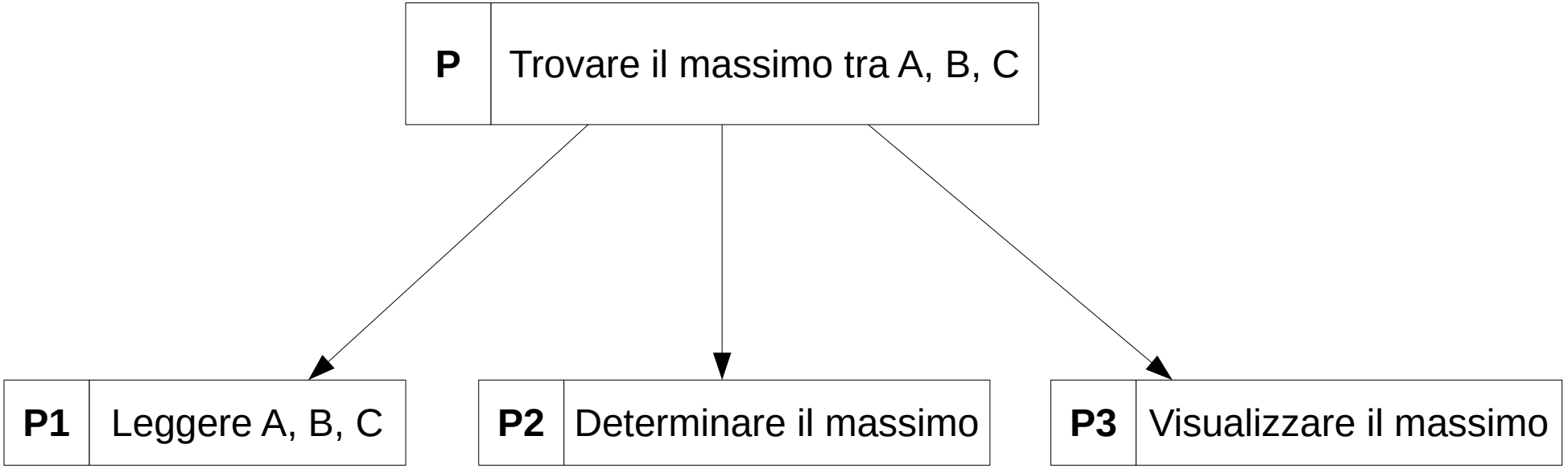
La progettazione TOP-DOWN



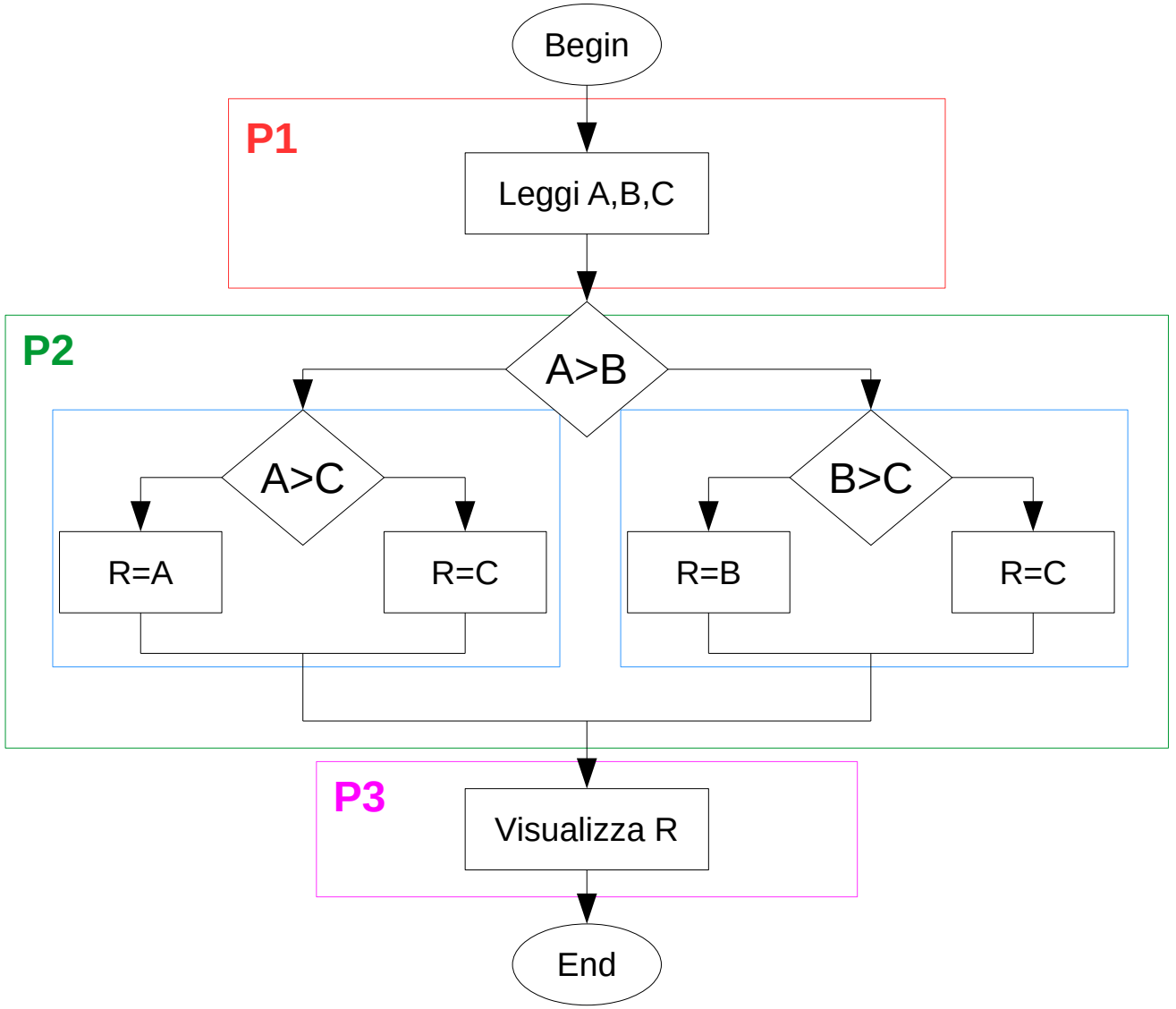
I sottoproblemi P1, P2, P3 sono INDIPENDENTI (possono essere risolti con algoritmi distinti) e COOPERANO alla risoluzione del problema P

Persone diverse potrebbero occuparsi di sviluppare gli algoritmi per risolvere problemi INDIPENDENTI

La progettazione TOP-DOWN



La progettazione TOP-DOWN



La progettazione BOTTOM-UP

Al contrario della progettazione TOP-DOWN, nella progettazione BOTTOM-UP si parte da sottoprogrammi già pronti che vengono assemblati per costruire la nuova elaborazione

"... si può affermare che, nella costruzione di un nuovo algoritmo, è dominante il processo top-down, mentre nell'adattamento (a scopi diversi) di un programma già scritto, assume una maggiore importanza il metodo bottom-up." (N. Wirth)

Scomposizione in sottoprogrammi

Partendo da un problema complesso, una volta individuati i vari sotto problemi elementari, è necessario, nella fase di codifica dell'algoritmo, preparare per ognuno di essi un sotto programma in grado di risolvere quel determinato sotto problema.

Scomposizione in sottoprogrammi

Seguendo il procedimento per scomposizioni successive si arriva alla fine ad avere:

- Un programma principale, il cui scopo è quello di richiamare i vari sotto programmi coordinando l'esecuzione del programma. L'esecuzione del programma principale viene sospesa ogni volta che un sottoprogramma viene chiamato e riprende quando un sotto programma termina la sua esecuzione.
- Uno o più sottoprogrammi. L'esecuzione di un sottoprogramma inizia ogni volta che questo viene chiamato (dal programma principale o da un altro sottoprogramma) e una volta terminata l'esecuzione del sottoprogramma l'esecuzione riprende dal punto successivo a cui il sottoprogramma è stato chiamato.

Scomposizione in sottoprogrammi

Ogni sottoprogramma riceve i propri input, cioè l'insieme delle informazioni necessarie per portare a termine il proprio compito, dai sottoprogrammi precedenti.

Ogni sottoprogramma fornisce i propri output (cioè l'insieme delle informazioni prodotte dalla sua esecuzione) al sottoprogramma chiamante.

Tipi di sottoprogrammi

Abbiamo parlato di sottoprogrammi in modo generico per evidenziare le proprietà comuni a tutti i linguaggi di programmazione.

In genere si fa distinzione fra due tipologie di sottoprogrammi:

1. Sottoprogrammi che restituiscono al programma chiamante un valore: la chiamata ad una funzione produce un valore che potrà essere assegnato ad una variabile.
2. Sottoprogrammi che non restituiscono alcun valore.

Funzionalità di calcolo

Supponiamo di voler scrivere un programma che, dati in ingresso due numeri interi a e b, visualizzi il risultato di:

1) a^b

2) b^a

Il seguente codice può essere utilizzato per calcolare x^y

```
int count;
int r = 1;
for (count=0; count < y; count++) {
  r = r * x;
}
```

Alla fine del ciclo for, la variabile r conterrà il risultato dell'elevamento a potenza.

Funzionalità di calcolo

Utilizzando le nozioni che abbiamo, possiamo risolvere il nostro problema in questo modo:

```
int main(void)
{
  int r,a,b,count;

  printf(“Inserisci il valore di a: ”);
  scanf(“%d”, &a);

  printf(“Inserisci il valore di b: ”);
  scanf(“%d”, &b);

  r = 1;
  for (count=0; count < b; count++)
    r = r * a;
  printf(“Il valore di a elevato b è: %d”, r);

  r = 1;
  for (count=0; count < a; count++)
    r = r * b;
  printf(“Il valore di b elevato a è: %d”, r);

  return 0;
}
```

Funzionalità di calcolo

Nella soluzione proposta, la funzionalità di elevamento a potenza è replicata due volte (cambiano solo i dati su cui opera).

E se dovessimo calcolare 100 elevamenti a potenza?

La dovremmo replicare 100 volte!

Inoltre, pensiamo ad un esempio più complesso...

Cosa succederebbe se si scoprisse un errore nella funzionalità? o se si desiderasse modificarla?

Infine, dovremo leggere attentamente il programma per capire cosa sta succedendo esattamente.

Funzionalità di calcolo

L'ideale sarebbe poter RIUTILIZZARE la porzione di codice scritta per calcolare sia a^b che b^a, qualcosa che funzioni circa così:

```
...
base = a;
exp = b;
// calcola base^exp
base = b;
exp = a;
// calcola base^exp
...
```

calcola base^exp

```
r = 1;
for (count=0; count < exp; count++)
  r = r * base;
```

Le funzioni

Le funzioni sono blocchi di codice che realizzano una funzionalità di calcolo riutilizzabile e sono caratterizzate da:

- Un nome usato per identificare la funzione
- Una serie di valori in ingresso (parametri) su cui la funzione opera
- Un blocco di istruzioni che viene eseguito ogni volta che la funzione viene richiamata
- Un valore di ritorno, il risultato della funzione di calcolo restituito al chiamante

Le funzioni

La sintassi per definire una funzione è la seguente:

```
 <tipo> nome(<tipo-parametro_1> nome-parametro_1, ... )
```

```
int nome_funzione(int parametro_1, int parametro_2)
{
  ...
  return r;
}
```

Se una funzione non ha un valore di ritorno, (esempio “stampa”) allora il tipo della funzione può essere dichiarato come void

Le funzioni

Per risolvere in modo opportuno il nostro problema, possiamo racchiudere la nostra porzione di codice in una funzione che riceve come parametri 2 valori interi e restituisce un valore intero.

```
int potenza(int x, int y) {
  int r, count;
  r = 1;
  for (count=0; count < y; count++) {
    r = r * x;
  }
  return r;
}
```

Richiamando la stessa funzione 2 volte possiamo risolvere il nostro problema in maniera più efficiente e più elegante

Prototipi delle funzioni

In C le funzioni vanno definite (scritte) nel codice al di fuori della funzione main() (in generale al di fuori di qualsiasi funzione) e soprattutto, come le variabili, vanno definite prima di essere richiamate.

In ANSI C è stata introdotta la possibilità di inserire una dichiarazione della funzione (prototipo) e di rimandare la sua definizione.

Chiamare le funzioni

Per poter eseguire il codice racchiuso in una funzione, è sufficiente usare il nome della funzione accompagnato dall'elenco dei parametri necessari.

I parametri passati dal chiamante sono chiamati argomenti o parametri attuali mentre gli stessi parametri, presenti nella definizione della funzione sono chiamati parametri formali.

Il sottoprogramma chiamante può utilizzare il valore restituito dalla funzione come:

- Valore da assegnare ad una variabile
- Parametro per chiamare un'altra funzione

Le funzioni

Sia la seguente, la definizione della funzione potenza()

```
int potenza(int a, int b)
{
...
```

Consideriamo la chiamata alla funzione potenza()

```
risultato = potenza(2,3);
```

1. Si sostituiscono i parametri formali con quelli attuali, cioè si sostituisce ad “a” il valore 2 e a “b” il valore 3
2. Si esegue il codice della funzione
3. L’istruzione return termina l’esecuzione della funzione e restituisce il valore 8
4. 8 è assegnato alla variabile risultato

esempio_completo.c

Esempio bis

Visibilità delle variabili

Si possono distinguere 2 tipi di variabili:

• Variabili globali:

Vengono dichiarate al di fuori di ogni funzione, sono quindi accessibili da tutti i sottoprogrammi.

• Variabili locali:

Vengono dichiarate all'interno di una funzione, sono quindi accessibili solamente dalla funzione che le dichiara.

In generale l'utilizzo di variabili globali è sconsigliato, si fa quindi più spesso uso di variabili locali e se qualche sottoprogramma chiamato ha bisogno di accedere a qualche variabile dichiarata nei sottoprogrammi chiamanti, queste vengono passate come parametro (argomento)

Passaggio di parametri

Si dice che un parametro è passato per valore (dal chiamante al chiamato) se il chiamante comunica al chiamato il valore che è contenuto in quel momento in una sua variabile.

Il chiamato predispone una propria variabile locale che conterrà una copia di tale valore.

esempio_passaggio_valore.c

Passaggio di parametri per valore

Cosa è successo?

All’interno di scambia() sono state create delle copie dei parametri passati come argomento, cioè delle variabili locali, pertanto queste sono diverse da quelle passate e sono state distrutte al termine dell’esecuzione della funzione.

Funzioni ricorsive

Una funzione si dice ricorsiva se all'interno del proprio codice essa richiama se stessa (direttamente o indirettamente).

Le funzioni ricorsive sono un'alternativa alle strutture iterative come i cicli while e for.

Le funzioni ricorsive in alcuni casi sono una soluzione più elegante, più chiara e più sintetica rispetto ai cicli, ma presentano sempre alcuni "contro" da tenere bene in considerazione:

- Il punto di terminazione della ricorsione
- Ad ogni chiamata vengono create delle nuove variabili locali
- Se la funzione è particolarmente complicata potrebbe essere difficoltoso il debug

Funzioni ricorsive

```
double fatt(int n) {
  if (n==0)
    return (1);
  else
    return (n*fatt(n-1));
}

int main() {
  int n, i;
  double fattoriale;

  printf("Inserisci un numero intero: ");
  scanf("%d", &n);

  fattoriale = fatt(n);

  printf("Il fattoriale di %d e\' %.0f \n\n", n, fattoriale);
}
```

Ricorsione

1
```
long int fatt(n=3){
  if (n==0)
    return 1;
  else
    return (3*fatt(2));
}
```

2
```
long int fatt(n=2){
  if (n==0)
    return 1;
  else
    return (2*fatt(1));
}
```

3
```
long int fatt(n=1){
  if (n==0)
    return 1;
  else
    return (1*fatt(0));
}
```

4
```
long int fatt(n=0){
  if (n==0)
    return 1;
}
```

Ricorsione

1
```
long int fatt(n=3){
  if (n==0)
    return 1;
  else
    return (3* 2 );
}
```
fatt(2)

2
```
long int fatt(n=2){
  if (n==0)
    return 1;
  else
    return (2* 1 );
}
```
fatt(1)

3
```
long int fatt(n=1){
  if (n==0)
    return 1;
  else
    return (1* 1 );
}
```
fatt(0)

4
```
long int fatt(n=0){
  if (n==0)
    return 1;
}
```

Grazie per l’attenzione

Riferimenti

Il corso di programmazione per il primo anno della Laurea Triennale in Matematica nasce con l'intento di unire ai principi di programmazione una conoscenza basilare di uno degli strumenti software più diffusi nell'ambito matematico: Matlab.

La prima parte del corso pertanto è una rielaborazione del programma di Programmazione per la Laurea Triennale in Informatica 15/16 (in particolare) e 16/17.

Parte del materiale originale da cui ho ricavato il percorso didattico, alcuni approfondimenti ed integrazioni possono essere trovati in:

- Lezioni di Programmazione 15/16 CdS Informatica - Giacomo Piva
- Lezioni di Programmazione 16/17 CdS Informatica - Marco Alberti