# I SOTTOPROGRAMMI

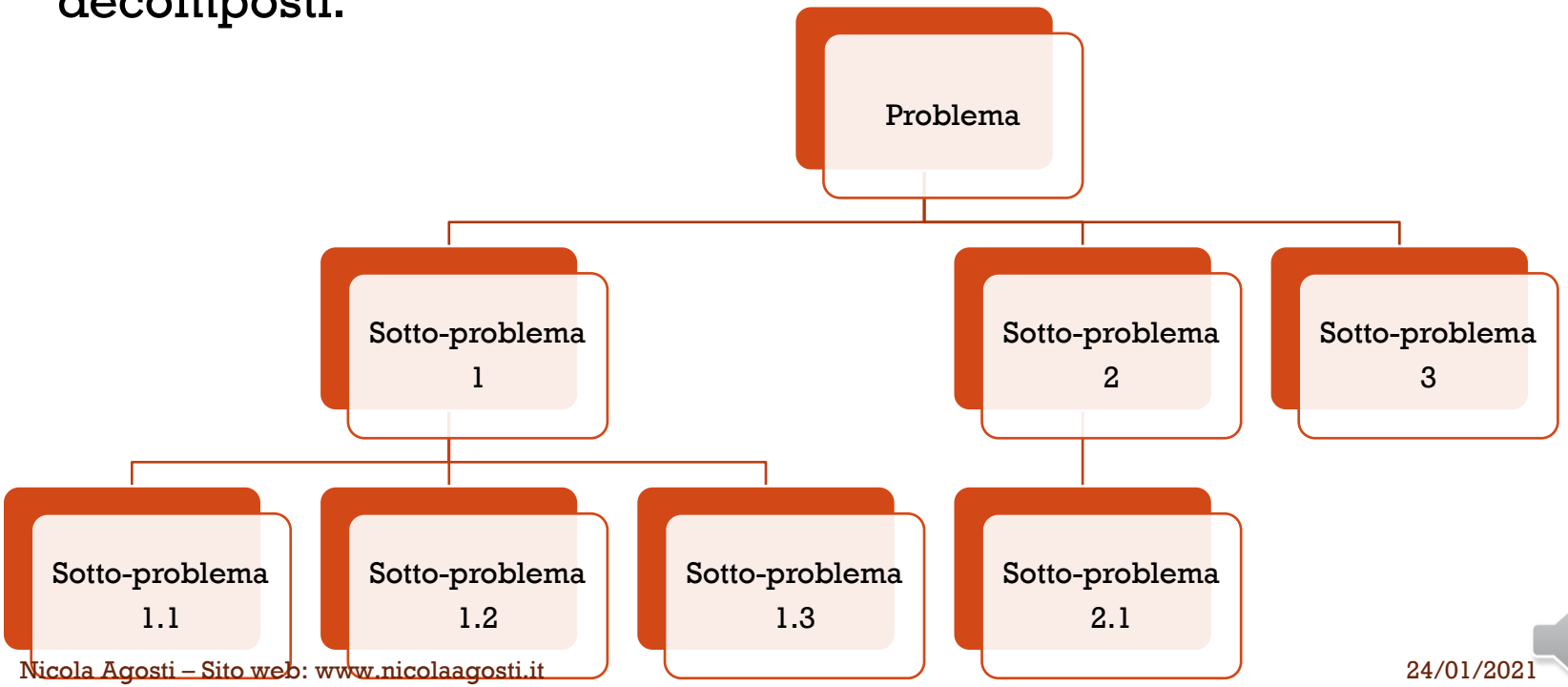
Il linguaggio C

1



# TOP-DOWN

- Quando un problema appare molto complesso, un metodo per affrontarlo è suddividerlo in sotto-problemi più semplici, che possono, se necessario, essere, a loro volta, ulteriormente decomposti.



# SOTTOPROGRAMMI

- Il C consente di dichiarare ed invocare i sottoprogrammi in qualunque punto del flusso di esecuzione del programma principale o all'interno degli stessi sottoprogrammi.
- Si distinguono due tipi di sottoprogrammi:
  - le **procedure** che non restituiscono alcun valore e non possono far parte di espressioni;
  - le **funzioni** che restituiscono un valore, assegnabile eventualmente ad una variabile, al chiamante e possono essere impiegate all'interno di espressioni.

| *void srand(unsigned int seed)* |
| --- |
| *procedura* |

| *int rand(void)* |
| --- |
| *funzione* |

# COME SI DICHIARA UNA PROCEDURA

- Per dichiarare una **procedura** è necessario:
  - indicare che non ritorna alcun valore attraverso l'utilizzo della parola chiave «void»;
  - assegnare un nome univoco alla procedura;
  - elencare, tra parentesi tonde e separati da virgole, l'elenco degli eventuali parametri (e il loro tipo) passati alla procedura.

```
void <NomeProcedura>([<ElencoParametri]>)
{
   <Istruzioni>
}
```



# ESEMPIO - PROCEDURA

```
#include <stdio.h>
#include <stdlib.h>

void stampa(int num)
{
  printf("\nIl numero e': %d", num);
}

int main(int argc, char *argv[])
{
  stampa(10);

  system("PAUSE");
  return 0;
}
```

# COME SI DICHIARA UNA FUNZIONE

- Per dichiarare una **funzione** è necessario:
  - indicare il tipo del valore di ritorno;
  - assegnare un nome univoco alla procedura;
  - elencare, tra parentesi tonde e separati da virgole, l'elenco degli eventuali parametri (e il loro tipo) passati alla procedura.

```
TipoRestituito <NomeFunzione>([<ElencoParametri]>)
{
  <Istruzioni>
}
```



# ESEMPIO - FUNZIONE

```
#include <stdio.h>
#include <stdlib.h>

float quadrato(float num)
{
  return num * num;
}

int main(int argc, char *argv[])
{
   printf("Il quadrato di 5 e': %.2f", quadrato(5));

   system("PAUSE");
   return 0;
}
```

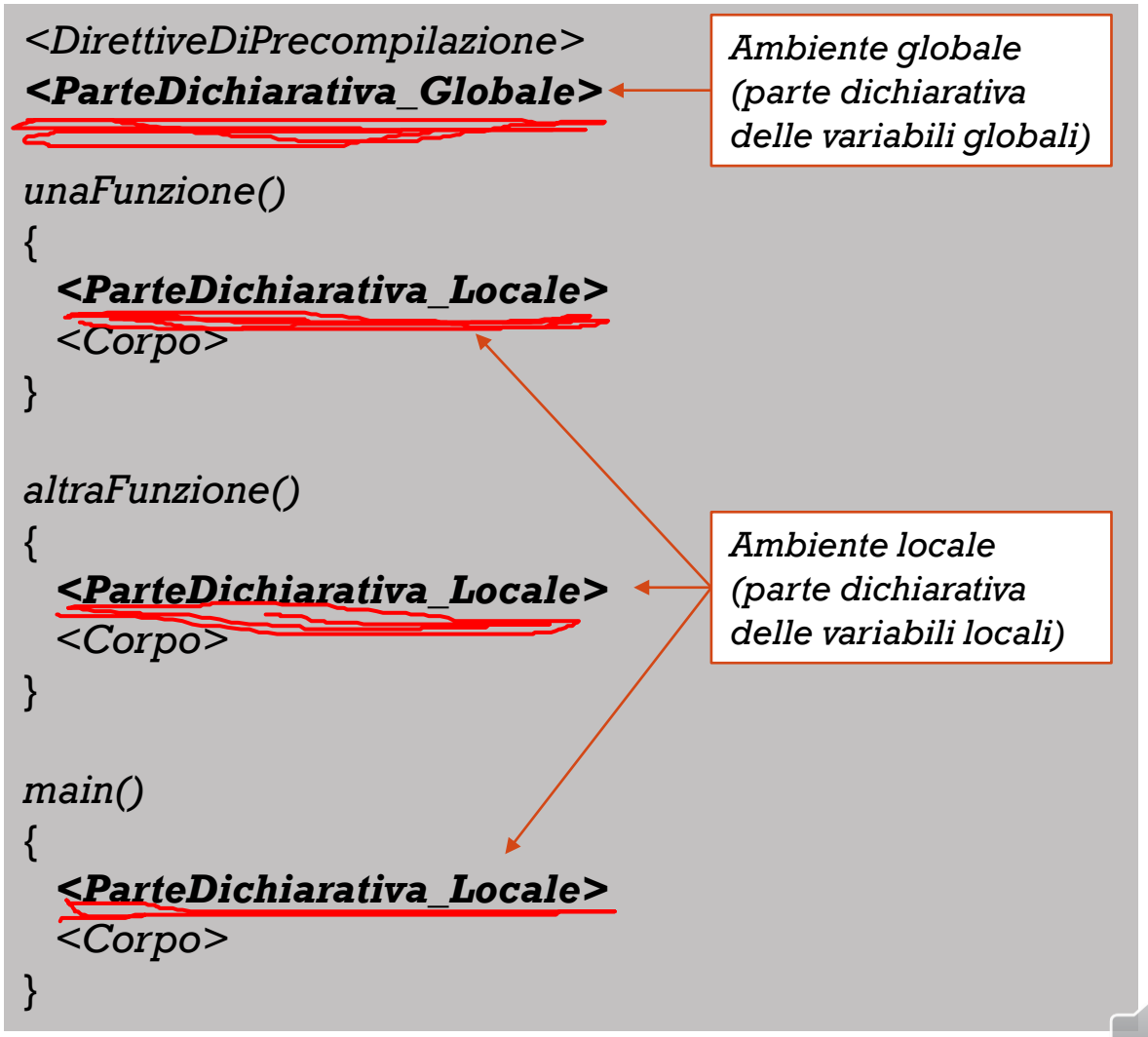
# VISIBILITÀ E PERIODO DI VITA DELLE VARIABILI

- Il **periodo di vita** di una variabile in un programma è l'intervallo di tempo, durante l'esecuzione di quel programma, nel quale la variabile esiste.
- La **visibilità** di una variabile è la porzione di programma nel quale essa può essere chiamata/utilizzata.
- In funzione del periodo di vita e della visibilità le variabili vengono classificate come:
  - **variabili locali**: utilizzabili/visibili solo all'interno della funzione in cui sono state dichiarate;
  - **variabili globali**: hanno visibilità e periodo di vita estesi all'intero programma.

N.B. Da qui in avanti, per semplicità espositiva, si utilizzeranno i termini funzione e procedura come sinonimi.

# VISIBILITÀ E PERIODO DI VITA DELLE VARIABILI

- Il fatto che una variabile sia locale o globale dipende dalla **posizione** nella quale è stata dichiarata all'interno del programma (**ambiente locale o globale**).



# VISIBILITÀ E PERIODO DI VITA DELLE VARIABILI

- Regole di visibilità:
  - una variabile non può essere usata se non è stata dichiarata;
  - le variabili globali sono accessibili a tutte le funzioni;
  - una variabile dichiarata in una funzione ha significato ed è visibile solo al suo interno;
  - le variabili locali sono create quando la funzione viene chiamata e distrutte quando la funzione termina;
  - se si dichiara una variabile locale con lo stesso nome di una variabile globale, durante il suo periodo di vita, un riferimento a tale nome indica la variabile locale e non quella globale (la località maschera la globalità);
  - anche le variabili dichiarate all'interno del *main* sono locali;
  - una variabile globale è visibile solo all'interno del file sorgente in cui è definita, a partire dal punto della definizione e non prima.

# VISIBILITÀ E PERIODO DI VITA DELLE VARIABILI

```
#include <stdio.h>
#include <stdlib.h>

int var = 10;

void procedura()
{
  int var = 5;
  printf("Variabile locale in procedura var = %d\n", var);
}

void altraProcedura()
{
  printf("Variabile globale in procedura var = %d\n", var);
}
```

Ambiente globale

Ambiente locale riferito a «procedura»

Questa «var» è quella globale

# VISIBILITÀ E PERIODO DI VITA DELLE VARIABILI

```
int main()
{
  int var = 2;

  printf("Variabile locale al main var = %d\n", var);

  procedura();
  altraProcedura();

  system("PAUSE");
}
```

*Ambiente locale riferito a «main»*

# PARAMETRI DELLE FUNZIONI

- Visto una funzione che riesce a «vedere» solo le variabili globali e le proprie variabili locali, se due funzioni devono condividere uno stesso valore questo deve:
  - essere memorizzato in una variabile globale;
  - passato come parametro alla funzione.
- Quando, all'atto della dichiarazione di una funzione, si indica, tra parentesi tonde, un elenco di «**parametri formali**», in realtà si stanno dichiarando una serie di variabili locali per la funzione in cui verranno memorizzati, al momento della chiamata, i valori che si intende passare alla funzione chiamata («**parametri attuali**»).
- I parametri permettono quindi di «**condividere le variabili locali**».

# ESEMPIO - PARAMETRI

- Calcolo della somma dei soli numeri pari presenti nel vettore con l'utilizzo delle funzioni.

```
#include <stdio.h>
#include <stdlib.h>

int pari(int num)
{
        if(num % 2 == 0)
                return num;
        else
                return 0;
}
```

Parametro formale

# ESEMPIO - PARAMETRI

```
int main(void)
{
	int vettore[] = {23, 56, 38, 71, 51, 57, 44, 92, 6, 9};
	int somma = 0, i;

	for(i = 0; i < 10; i++)
	{
		somma += pari(vettore[i]);
	}

	printf("Somma dei numeri pari = %d", somma);

	system("PAUSE");
	return 0;
}
```

Parametro attuale

# ATTENZIONE

- Se una funzione ha più parametri formali (anche dello stesso tipo), nella dichiarazione della funzione è, comunque, necessario elencare ognuno di essi con il proprio tipo associato.

```
int somma(int x, y);
```

```
int somma(int x, int y);
```

- La modalità utilizzata fin qui per il passaggio dei parametri viene chiamata «**per valore**» e non consente di:
  - passare vettori e stringhe come parametri;
  - utilizzare i parametri per ritornare valori al chiamante;
  - modificare in modo permanente il contenuto delle variabili passate dal chiamante.

# PROTOTIPI DI FUNZIONE

- Il **prototipo** di una funzione non è altro che una dichiarazione di quella funzione in cui vi è solo la parte dell'intestazione e dove viene indicato solo il tipo degli eventuali parametri formali.
- Solitamente vengono inseriti all'inizio del programma, subito dopo gli header, e comunque prima della chiamata alle funzioni stesse.
- La dichiarazione della funzione ed il relativo codice dovrà poi essere scritta in un'altra parte del programma.
- Per indicare l'assenza del valore di ritorno o l'assenza di parametri si usa la parola chiave «**void**»

```
int somma(int, int);
```

```
void stampa(void);
```

# LA RICORSIVITÀ

- Si dice **ricorsiva** un funzione che richiama sé stessa.
- Si dice **iterativa** una funzione che NON richiama sé stessa e che, normalmente, risolve il problema attraverso uno o più cicli.
- Osservazioni:
  - qualsiasi funzione ricorsiva **può essere trasformata** in una iterativa;
  - una funzione ricorsiva spesso risulta **più leggibile** di una iterativa;
  - una funzione ricorsiva è normalmente **meno efficiente**, sia in termini di velocità di esecuzione sia in termini di occupazione di memoria, di una iterativa.



# LA RICORSIVITÀ

- Per attivare un processo ricorsivo:
  1. il problema deve essere **scomponibile in sotto-problemi dello stesso tipo**, ognuno dipendente dall'altro in base ad una scala gerarchica;
  2. si devono conoscere le relazioni funzionali che legano il problema ai sotto-problemi simili (**caso generale**);
  3. è necessario conoscere la soluzione di (almeno) un caso particolare del problema, indispensabile per terminare il problema stesso (**condizione di terminazione**).

$$Potenza\ n-esima\ di\ un\ numero\ a\ \neq 0$$
$$\begin{cases} a^n = 1\ se\ n = 0\ e\ a \neq 0 \\ a^n = a * a^{n-1} \quad se\ n > 0 \end{cases}$$

# LA RICORSIVITÀ

- Elementi caratteristici dei problemi ricorsivi:
  1. una **condizione di terminazione**, che permette di determinare se si è di fronte ad un caso particolare risolvibile banalmente o se è necessario ripetere la chiamata alla funzione;
  2. il procedimento di soluzione del **caso generale**, che contiene una o più chiamate ricorsive;
  3. la soluzione di almeno un **caso particolare**, che termina le chiamate ricorsive.

$$Potenza\ n-esima\ di\ un\ numero\ a\ \neq 0$$
$$\begin{cases} a^n = 1\ se\ n = 0\ e\ a \neq 0 \\ a^n = a * a^{n-1} \quad se\ n > 0 \end{cases}$$

# ESEMPIO - RICORSIVITÀ

- Calcolo della potenza n-esima di un numero intero positivo utilizzando una funzione ricorsiva.

```
[…]
int potenza(int, int);

int main(void)
{
        int a, n;

        printf("\nInserisci la base: ");
        scanf("%d", &a);
        printf("\nInserisci l'esponente: ");
        scanf("%d", &n);

        printf("\n%d elevato a %d = %d\n", a, n, potenza(a, n));
        […]
}
```

Prototipo della funzione ricorsiva

Chiamata alla funzione ricorsiva

# ESEMPIO - RICORSIVITÀ

```
int potenza(int a, int n)
{
    if(n == 0)
    {
        return 1;
    }
    return a * potenza(a, n-1);
}
```

*Condizione di terminazione*

*Soluzione «banale»*

*Chiamata a sé stessa (ricorsione)*

# ESERCIZI

- **Esercizio 1:** scrivere un programma che calcoli il fattoriale di un numero naturale ($n \geq 0$) utilizzando:
  a) un algoritmo recursivo;
  b) un algoritmo iterativo.
- **N.B.** Il fattoriale di un numero naturale N è dato dal prodotto dei primi N numeri naturali ed è rappresentato dal simbolo «!».
- **N.B.** Il fattoriale di 0 è uguale a 1.

$$Fattoriale\ di\ un\ numero\ naturale$$
$$\begin{cases} n! = 1 & se\ n = 0 \\ n! = n * (n-1)! & se\ n > 0 \end{cases}$$

# ESERCIZI

- **Esercizio 2:** scrivere un programma che implementi il cosiddetto «Gioco della vita», i cui dettagli sono descritti qui: https://it.wikipedia.org/wiki/Gioco_della_vita
- Suggerimenti:
  - Scomponete il programma in un numero opportuno di sotto-programmi.
  - Definite la matrice (il campo di gioco) come una variabile globale in quando, ad ora, non abbiamo ancora visto come passare vettori e matrici come parametri alle funzioni.
  - Dopo aver sviluppato un'implementazione di base correttamente funzionante, complicate il programma a piacimento inserendo, ad esempio, il riconoscimento di situazioni statiche, cicliche, ecc.