



Individuazione di sottoproblemi

- Quando il problema è complesso conviene partire con una individuazione di sottoproblemi
- Scriviamo un algoritmo contenente azioni o condizioni complesse per l'esecutore che dettaglieremo e raffineremo in passaggi successivi per ottenere un algoritmo direttamente eseguibile
- Ognuno dei sottoproblemi potrà essere risolto da un algoritmo a parte che potremo riutilizzare, quando sarà necessario, nella soluzione di ulteriori problemi complessi.



Individuazione di sottoproblemi: vantaggi

- I dettagli delle diverse soluzioni sono descritti negli algoritmi dei sottoproblemi
- In generale, uno stesso sottoproblema deve essere risolto più volte nella soluzione di un problema principale
- Dagli algoritmi derivano programmi, quindi si possono raccogliere librerie di software da riutilizzare in nuovi programmi



Esempio: ripetizione di codice

- Se in un algoritmo fosse prevista la lettura di due numeri positivi in ingresso, allora la parte di codice Javascript che traduce questa parte dell'algoritmo potrebbe essere

Esempio: ripetizione di codice

```
<SCRIPT>
<!-- Inizio script JavaScript
var a0, b0, r;

a0 = window.prompt("Inserisci il primo numero");
while (isNaN(a0) || a0 <= 0 || a0 == null ||
      ((a0 - parseInt(a0)) != 0)) {
    window.alert("Il valore " + a0 + " non va bene,\n" +
                "inserire un numero intero positivo.");
    a0 = window.prompt("Inserisci il primo numero");
}
a0 = a0 * 1;
b0 = window.prompt("Inserisci il secondo numero");
while (isNaN(b0) || b0 <= 0 || b0 == null ||
      ((b0 - parseInt(b0)) != 0)) {
    window.alert("Il valore " + b0 + " non va bene,\n" +
                "inserire un numero intero positivo.");
    b0 = window.prompt("Inserisci il primo numero");
}
b0 = b0 * 1;
var a = a0, b = b0;
[...]
// Fine script -->
</SCRIPT>
```



Esempio: ripetizione di codice

- Desideriamo controllare che l'input inserito sia effettivamente un numero intero positivo
- Se non è un intero positivo il numero inserito si segnala l'errore e si richiede una nuova immissione
- Si noti `parseInt(.)`, `isNaN(.)` e `alert(.)`



Esempio: ripetizione di codice

- Il precedente esempio contiene due blocchi di istruzioni simili per la richiesta e il controllo dell'input, una per ogni valore richiesto all'utente
- I due blocchi di istruzioni differiscono per:
 - la variabile su cui è memorizzato il valore in input (a0 e b0)
 - il messaggio che viene visualizzato nella finestra "prompt" ("Inserisci il primo/secondo numero")
- Se l'algoritmo prevedesse l'inserimento di 12 numeri da memorizzare in altrettante variabili avremmo scritto per 12 volte lo stesso codice!!!
- Ci piacerebbe poter disporre di una nuova istruzione del tipo
 - `promptNumero(messaggio)`
- La nuova istruzione dovrebbe essere come prompt ma con il controllo che il valore immesso sia un intero positivo

Esempio: ripetizione del codice

- Se dovessimo scrivere un algoritmo che presi tredici numeri determini se sono tutti e tredici dispari o meno allora questo stralcio di codice dovrebbe essere scritto 13 volte per 13 variabili diverse (con un array risparmieremmo un po')

```
var N1, N1_dispari;
N1 = window.prompt("Inserisci il numero");
N1 = N1 * 1;
while (N1 > 1) {
    N1 = N1 - 2;
}
if (N1==0)
    { N1_dispari = true; }
else
    { N1_dispari = true; }
```



Esempio: ripetizione del codice

- Ci piacerebbe poter disporre di una nuova istruzione del tipo
 - `NumeroDispari(numero)`
- La nuova istruzione dovrebbe dire true se il numero è dispari e false altrimenti così potremmo scrivere

```
var N1, N1_dispari;  
N1 = window.prompt("Inserisci il numero");  
N1 = N1 * 1;  
N1_dispari = NumeroDispari(N1);
```



Individuazione di sottoproblemi: vantaggi

- Per risolvere il problema del calcolo del MCD bisogna risolvere il sottoproblema della divisibilità
- Per risolvere il problema del pari o dispari bisogna risolvere il sottoproblema della divisibilità
- Per risolvere il problema del numero primo bisogna risolvere il sottoproblema della divisibilità
- La suddivisione in sottoproblemi serve a risolvere un sottoproblema con un algoritmo, codificarlo in un linguaggio di programmazione, riusare il codice scritto per la risoluzione di altri problemi (pensate al problema di calcolare la radice quadrata di un numero)

Le funzioni: definizione

- In altre parole: si vorrebbe definire una **funzione**, cioè *una parte di codice utilizzabile in più parti di uno stesso programma*
- In JavaScript questo è possibile farlo utilizzando la parola chiave **function** :

```
function nome_della_funzione ( arg1, arg2, ..., argn) {  
    "definizione della funzione"  
}
```

nome della funzione

"definizione della funzione"

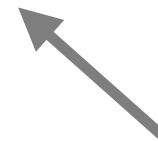
parentesi graffe!

parametri "formali" della funzione, usati nella definizione della funzione

Le funzioni: richiamo

- La definizione di una funzione è una sequenza di istruzioni, un blocco di istruzioni
- Le istruzioni contenute in una funzione non vengono eseguite quando definite ma solo al momento del richiamo della funzione:

- `nome_della_funzione(val1, val2, ..., valn)`



parametri attuali
della funzione

- Quando l'interprete incontra un richiamo di una funzione passa ad eseguire il codice contenuto nella definizione della funzione, dopo aver sostituito i parametri formali con quelli attuali



Ambito di validità delle variabili

- Variabili locali: i parametri formali e le dichiarazioni di variabili interne ad una funzione
- Variabili globali: le dichiarazioni di variabili esterne alle funzioni, a livello più alto
- Le variabili locali sono visibili solo all'interno della funzione in cui sono dichiarate ma mai all'interno di altre funzioni o a livello principale
- Le dichiarazioni locali nascondono quelle principali con lo stesso identificatore



Le funzioni: restituire un risultato

- All'interno di una funzione si può usare l'istruzione `return` per restituire dei valori, di solito il risultato prodotto dalla funzione stessa
 - `return <espressione>;` : restituirà il valore computato dall'espressione
 - `return ;` : restituirà `undefined`
- In entrambi i casi si esce dalla funzione e l'interprete JavaScript passa ad eseguire l'istruzione che segue il richiamo della funzione

Esempio

```
<SCRIPT>
<!-- Inizio script JavaScript

function promptNumero(messaggio) {
    var numero = window.prompt(messaggio);
    while (isNaN(numero) || numero <= 0 ||
           numero == null ||
           ((numero - parseInt(numero)) != 0)) {
        alert("Il valore " + numero + " non va bene,\n"
+
            "inserire un numero intero positivo.");
        numero = prompt(messaggio);
    }
    return numero * 1;
}

var a0, b0, r;

a0 = promptNumero("Inserisci il primo numero");
b0 = promptNumero("Inserisci il secondo numero");

var a = a0, b = b0;
[...]
// Fine script -->
</SCRIPT> -->
```



Esempio

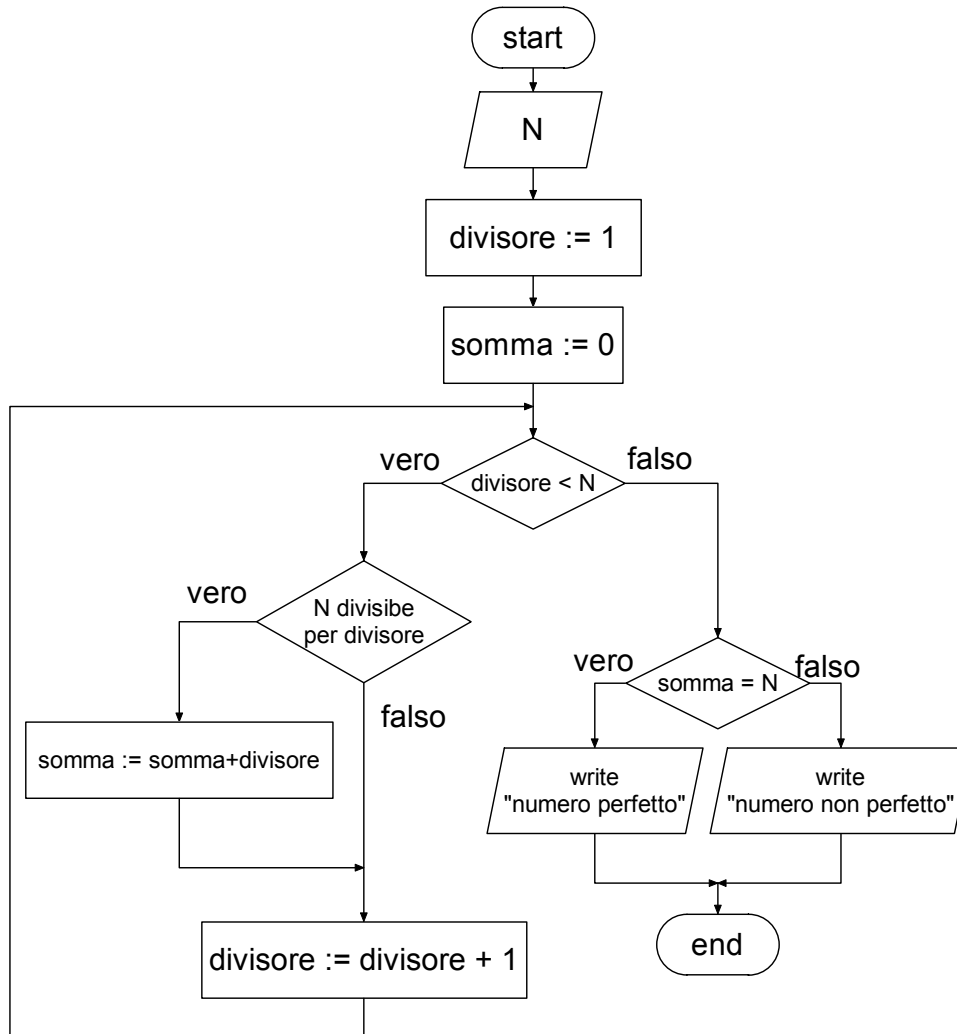
- La funzione `promptNumero(.)` richiede un numero intero in input, verifica che il valore immesso lo sia
- Il valore restituito è il numero intero positivo inserito dall'utente, dopo averlo convertito in `numero(!)`
- Il parametro attuale è il messaggio da visualizzare nella richiesta



Individuazione di sottoproblemi: esempio

- Scrivere un algoritmo che determina se un numero è perfetto.
- Un numero si dice *perfetto* quando è uguale alla somma di tutti i suoi divisori escluso se stesso.
- Ad esempio, il numero 28, divisibile per 1, 2, 4, 7, 14 è un numero perfetto ($28 = 1 + 2 + 4 + 7 + 14$).
- Per scrivere questo algoritmo dovremmo scrivere un algoritmo per la divisibilità ed il relativo codice Javascript

Esempio: numero perfetto





Codice Javascript

```
<html>
<head>
  <title>Esercizio sui numeri perfetti</title>
</head>
  <body>
<script>
<!-- Inizio script JavaScript

function isDividabile(dividendo,divisore)
{
  if(dividendo%divisore==0)
    return true;
  else
    return false;
}

function promptNumero(messaggio) {
  var numero = window.prompt(messaggio);
  while (isNaN(numero) || numero <= 0 ||
    numero == null ||
    ((numero - parseInt(numero)) != 0)) {
    window.alert("Il valore " + numero + " non va bene,\n" +
      "inserire un numero intero positivo.");
    numero = window.prompt(messaggio);
  }
  return numero * 1;
}
```



Codice Javascript

```
var N;
var divisore;
var somma;

N = promptNumero("Inserisci il numero");
divisore = 1;
somma = 0;
while (divisore < N)
{
    if(isDividabile(N,divisore)==true)
        somma = somma + divisore;
    divisore = divisore + 1;
}
if(somma==N)
    window.alert("Il numero " + N + " è perfetto");
else
    window.alert("Il numero " + N + " non è perfetto");

// Fine script -->

</script>
</body>
</html>
```