Programmazione I

a.a 2008-2009

docente: Carmine Gravino

Progettazione di Algoritmi

Presentazione realizzata dal Prof. Andrea De Lucia

Progettazione di Algoritmi

- *Verificare se un numero naturale è primo*
- *Calcolare il massimo comune divisore di due numeri naturali*

Numeri primi: Analisi e Specifica

- **Dati di ingresso: *n***
- **Precondizione: *n* > 1**
- **Dati di uscita: *b***
- **Postcondizione: se *n* è primo allora *b* = TRUE, altrimenti *b* = FALSE. Un numero è primo se è divisibile solo per 1 e per se stesso.**

Attributo	*Tipo*	*Descrizione*
n	intero	Numero naturale in ingresso
b	booleano	Valore booleano in uscita

Algoritmo: 1ª soluzione

1. Dividere *n* per tutti i numeri compresi tra 1 ed *n* e contare il numero di divisori.
2. Se il numero di divisori è 2 allora *n* è primo

```
do  scanf("%d", &n);  //  test precondizione
while(n <=1);
divisori = 0;                                  } conta divisori
for (i = 1; i <= n; i++)
     if(n % i == 0)  divisori++;
if(divisori == 2)
    printf("è un numero primo \n");
else printf("non è un numero primo \n");
```

... aggiungere divisori e i al dizionario dati ...

Correttezza dell'algoritmo

- **Inizialmente *divisori* è 0. Poiché ad ogni passo *i* del ciclo, $1 \leq i \leq n$, *divisori* viene incrementato se *i* è un divisore di *n*, allora al termine del passo *i*-esimo *divisori* sarà il numero di divisori di *n* compresi tra 1 ed *i*. Di conseguenza al termine del passo *n* (uscita dal ciclo) *divisori* sarà il numero di divisori di *n*.**
- **La proprietà "al termine del passo *i*-esimo *divisori* sarà il numero di divisori di *n* compresi tra 1 ed *i*" viene detta invariante di ciclo.**
- **... non sempre è facile dimostrare la correttezza di un algoritmo ... in alternativa lettura del codice passo passo e testing**
  - ☞ **nel nostro caso, provare per i seguenti valori di n: -5, 1, 11, 15 (... perché ...)**

Migliorare l'efficienza dell'algoritmo

- Prima osservazione: non serve dividere per tutti i valori di ***i***. Posso fermarmi quando incontro un divisore ***i*** diverso da 1 ed ***n***.

```
do  scanf("%d", &n);
while(n <=1);
primo = 1;  i = 2;
while (primo && i < n)
      if(n % i == 0)  primo = 0;
      else i++;
if(primo)
printf("è un numero primo \n");
else printf("non è un numero primo \n");
```

Ancora miglioramenti

- Seconda osservazione: posso verificare fuori dal ciclo i numeri pari ed evitare di dividere per i multipli di 2.
- Terza osservazione: se un numero ***n*** non è primo, allora ha almeno due divisori: ***n*** = ***d1*** * ***d2***. Uno dei due è sicuramente minore o uguale della radice quadrata di n.
  Quindi, se ***n*** è primo, posso uscire dal ciclo quando ***i*** > sqrt(***n***), o equivalentemente quando
  i * ***i*** > ***n***

Programma completo

```
#include <stdio.h>
main() {
   int n, i, primo;

  do { printf("Inserisci numero positivo > 1: ");
       scanf("%d", n); }
  while(n <=1);
  if(n %  2 == 0 && n != 2)  primo = 0;
  else { primo = 1;  i = 3;
         while (primo && i * i <= n)
              if(n % i == 0)  primo = 0;
              else i += 2; }
  if(primo)   printf("%d e' un numero primo \n", n);
  else printf("%d non e' un numero primo \n", n); }
```

Massimo Comune Divisore: Analisi e Specifica

- **Dati di ingresso: *x*, *y***
- **Precondizione: $x > 1$ AND $y > 1$**
- **Dati di uscita: *gcd***
- **Postcondizione: *gcd* è il massimo comune divisore di *x* e *y*. (*gcd* è un divisore di *x* e *y*. Per ogni altro divisore z di *x* e *y*, $gcd \geq z$).**

Attributo	*Tipo*	*Descrizione*
x	intero	Numero naturale in ingresso
y	intero	Numero naturale in ingresso
gcd	intero	Massimo comune divisore tra x e y

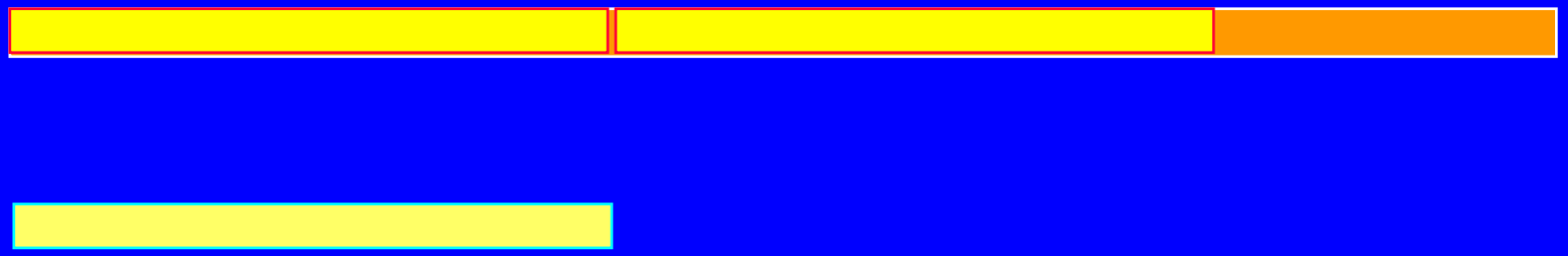
Algoritmo: 1ª soluzione

1. Inizialmente porre *gcd* al minimo tra *x* e *y*
2. Decrementa ***gcd*** finché ***gcd*** non divide sia ***x*** che ***y***

```
do { scanf("%d", &x);
     scanf("%d", &y);}
while(x <= 1 || y <= 1);
gcd = x < y ? x : y;          // espressione condizionale
while(x %  gcd != 0   ||  y % gcd != 0)
   gcd--;
printf("%d", gcd);
```

Algoritmo di Euclide

- Difficile la dimostrazione di correttezza.
- Intuizione: il massimo comune divisore di x e y è anche un divisore del resto della divisione del più grande per il più piccolo tra x e y.



Algoritmo di Euclide

```
#include <stdio.h>
main() {
   int x, y, r, done;
  do { printf( "Inserisci 1^ numero positivo > 1: ");
        scanf("%d", &x); }
  while(x <=1);
  do { printf("Inserisci 2^ numero positivo > 1: ");
        scanf("%d", &y); }
  while(y <=1);
  done = 0;
  while(! done) {
       r = x % y;
       if (r == 0)  done = 1;
       else { x = y;   y = r; } }
  printf("il massimo comune divisore e': %d \n", y);
}
```

Realizzazione di algoritmi: caratteristiche da considerare

- Correttezza
  - **se non funziona serve a poco ...**
- Efficienza
  - **i risultati devono essere prodotti in tempo utile ...**
- Comprensibilità (Chiarezza)
  - **l'efficienza non deve andare a scapito della chiarezza (qualcun altro potrebbe aver necessità di capire ...)**
- Manutenibilità
  - **se il programma sarà usato a lungo, prima o poi dovrò modificarlo ...**