Tecniche della Programmazione, lez. 12 - seconda parte

- caratteri
- reali: Floating Point

E parliamo anche di `espressione condizionale` e *obfuscation* …

Tipo base char

Rappresenta i caratteri (di solito ASCII) in un byte.
Nel byte (la variabile di tipo char) c'è un codice virtualmente compreso nell'intervallo [0, 255]


```
char crt;
      crt = 'f'
      printf ("BLA … %c\n", crt);
```

A(65), B(66), …, Z(90), …, a(97), …, z(122)
0(48), …, 9(57)
!(33), #(35), {(123), }(125), ~(126)

Tipo base char: aritmetica

Apropos della slide precedente, dopo aver verificato che l'output qui sotto corrisponde al codice mostrato, **Vedi Approfondimenti**

```
int main () {
  char crt,crt2='d';
…                       scanf("%c", &crt);
printf ("caro/a utente, …due … sono %c e %c\n", crt+1, crt+2);

printf ("invece i due predecessori …%c e %c\n", crt-2, crt-1);

printf ("moltiplicando per 2 %c si ottiene %c\n", crt, crt*2);

printf ("dividendo per 2 %c si ottiene %c\n", crt, crt/2);
crt=149;
…
printf ("il … divisione di %c per d è %c\n", crt, crt%crt2);
```

```
caro/a utente, dammi un carattere ...
f
caro/a utente, i successivi due caratteri sono g e h
invece i due predecessori immediati sono d e e
moltiplicando per 2 f si ottiene ╠
dividando per 2 f si ottiene 3
il resto della divisione di 149 per 100 e' 49
il resto della divisione di -107 per 100 e' -7
il resto della divisione di ò per d e' ·
FINE
```

Visto che ci siamo … Espressione condizionale

C'è un altro operatore … OPERATORE CONDIZIONALE

? :

è TERNARIO, cioè prende tre operandi, permettendo di ottenere un'espressione condizionale:

EXP_C (condizione) ? (exp1) : (exp2)

Valutazione di EXP_C :

se la condizione vale TRUE, EXP_C ritorna il valore di exp1;

altrimenti EXP_C ritorna il valore di exp2.

```
int maxTra2(int n1, int n2) {

   return (n1 > n2) ? n1 : n2);
}
```

Di seguito, due o tre esercizi sull'espressione condizionale.

Espressione Condizionale - esercizio 1

Aggiungere la definizione della funzione isEven() ... (cioè "è pari") per far funzionare bene il programma proposto.
La funzione riceve un valore intero e restituisce 1 o 0, a seconda se il valore è pari o dispari.
La funzione deve usare un'espressione condizionale per determinare quale valore restituire

```
#include <stdio.h>

/* prototipo isEven() */

int main() {
    int num = 25;

    if (isEven(num)) {
        printf("%d e` pari\n", num);
    } else printf("%d e` dispari\n", num);
    }
printf("\nFINE\n");
return 0;
}

/* definizione isEven() */
```

Vedi Esercizi

Espressione Condizionale - esercizio 2

Il programma qui sotto risolve un'equazione di secondo grado, i cui coeff. sono dati in input. Aggiungere la definizione della funzione soluzioniComplesse(), che riceve i coefficienti e restituisce 1 se sono ammesse soluzioni complesse coniugate, 0 se sono ammesse soluzioni reali. La funzione deve usare un'espressione condizionale per determinare quale valore restituire

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

/* prototipo soluzioniComplesse() */

int main(){
 double a, b, c, delta, x1, x2, parteReale, parteImmag;

  printf("Inserisci i coefficienti dell'equazione ax^2 + bx + c = 0:\n");
  scanf("%lf %lf %lf", &a, &b, &c);

  delta = b * b - 4 * a * c;

  if (soluzioniComplesse(a, b, c)) {
      parteReale = -b / (2 * a);
      parteImmag = sqrt(-delta) / (2*a);
      printf("Le soluzioni sono complesse: %.2lf + %.2lfi e %.2lf - %.2lfi\n",
              parteReale, parteImmag, parteReale, parteImmag);
  } else {
      x1 = (-b + sqrt(delta)) / (2*a);
      x2 = (-b - sqrt(delta)) / (2*a);
      printf("Le soluzioni sono reali: %.2lf e %.2lf\n", x1, x2);
  }

printf"\nFINE);
return 0; }

/* definizione soluzioniComplesse() */
```

Vedi Esercizi

Espressione Condizionale - esercizio 3

Aggiungere la definizione della funzione maxTra3(), che riceve tre numeri interi e restituisce il massimo tra loro.
La funzione deve usare un'espressione condizionale per determinare quale valore restituire.

```
#include <stdio.h>

/* prototipo maxTra3() */

int main() {
 int numero1=29, numero2=12, numero3=18;
 /* per test
    int numero1=12, numero2=11, numero3=18;
    int numero1=-19, numero2=-22, numero3=-61;
    int numero1=19, numero2=22, numero3=-61;
  */

  print("il massimo tra %d, %d, and %d e`: ", numero1, numero2, numero3);
  print("%d\n", maxTra3(numero1, numero2, numero3));

printf("\nFINE\n");
return 0;
}

/* definizione maxTra3() */
```

Vedi Esercizi

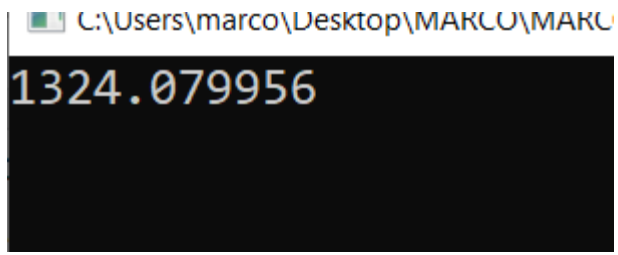
Espressione condizionale per il programma sui caratteri ... e poi obfuscation

ora e` il momento
Vedi Approfondimenti

Tipo base float

Rappresentazione approssimata dei numeri reali

```
float r;
r=1324.08;
printf("%f\n", r);
```



Si chiama RAPPRESENTAZIONE IN VIRGOLA MOBILE (Floating Point)

La rappresentazione deriva dalla notazione scientifica, in cui un numero è rappresentato come

$$M \times 10^E$$

M è la ***mantissa***, che rappresenta la parte frazionaria del numero, con (quasi) tutte le sue cifre;
E è ***l'esponente***.

$$223.365 \times 10^6 = 223365000 = 2.23365 \times 10^8 = 0.223365 \times 10^9$$

NB2 Ovviamente useremo solo cifre binarie e potenze di 2.
E la rappresentazione FP è «normalizzata»:
prima del «point» c'è una cifra non nulla (cioè 1)

Tipo base float

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per le cifre della mantissa normalizzata, escluso il primo 1 (M)

Nella locazione di memoria ci sono le seguenti informazioni:

- **segno** s del numero rappresentato (0/positivo, 1/negativo)

- **M** rappresenta la sequenza di cifre (binarie) della **mantissa normalizzata**
  … solo quelle successive al "1." … vedi gli esempi dopo …

- **esponente** E: un numero (binario) compreso tra 0 e 255;
  scritto in "eccesso 127"
  … [0,255] -> [-127, 128]

Tipo base float

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per le cifre della mantissa normalizzata, escluso il primo 1 (M)

Nella locazione di memoria ci sono le seguenti informazioni:

- segno s del numero rappresentato (0/positivo, 1/negativo)

- M rappresenta la sequenza di cifre (binarie) della mantissa normalizzata … solo quelle successive al "1." … vedi gli esempi dopo …

- esponente E: un numero (binario) compreso tra 0 e 255; questo numero è scritto in "eccesso 127", cioè è stato calcolato aggiungendo 127 al suo valore effettivo. Quindi, quando viene usato nel calcolo $M \times 2^{esp}$, gli viene sottratto 127. Questo permette di avere esponenti anche negativi, e quindi di rappresentare numeri reali di ordini di grandezza che vanno da 2^{-127} a 2^{128} … vedi gli esempi dopo …

Tipo base float

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per M

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Nella locazione di memoria ci sono le seguenti informazioni:
- segno s del numero rappresentato (0/positivo, 1/negativo)

- esponente E: un numero (binario) compreso tra 0 e 255; questo numero è scritto in "eccesso 127", cioè è stato calcolato aggiungendo 127 al suo valore effettivo. Quindi, quando viene usato nel calcolo indicato sopra, gli viene sottratto 127. Questo permette di avere esponenti anche negativi, tra -127 a 128, e quindi di rappresentare numeri reali di ordini di grandezza che vanno da 2^{-127} a 2^{128} … vedi gli esempi dopo …

- M rappresenta la sequenza di cifre (binarie) della mantissa normalizzata … solo quelle successive al "1." … vedi gli esempi dopo …

Tipo base float - esempio

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per M

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

$(4.25)_{10}$ = 100.01

→ mantissa normalizzata 1.0001

M = 0001 (solo le cifre della mantissa normalizzata successive al "1.")

→ esponente di 2 per tornare dalla mantissa normalizzata al numero: 2

(a questo esponente va aggiunto 127, per avere la sua rappresentazione in eccesso 127)

$E = 2+127 = 129\ (10000001)_2$

rappr(4.25) = 0 10000001 00010000000000000000000

prova: facciamo il calcolo indicato in alto a destra, per provare che la rappresentazione FP qui sopra è corretta …

$$1.M \times 2^{E-127} = 1.0001 \times 2^2 = 4.25$$

☺ sviluppa il calcolo

Tipo base float - esempio

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Altro esempio

data la rappresentazione

1 10000000 10010010000000000000000

qual è il numero?

segno ?

E = ?

M = ?

mantissa normalizzata = ?

numero =

Tipo base float – esempio

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Altro esempio

data la rappresentazione

1 10000000 10010010000000000000000

qual è il numero?

segno negativo;

E = 128 -127 = 1 cioè E è 128, ma il valore effettivo è 128 in eccesso 127 cioè 1;

M = 1001001

mantissa normalizzata = 1.1001001 = 1 + $\frac{1}{2}$ + 1/16 + 1/128 = 1.5703125

numero = $-1.5703125 \times 2^1 = -3.140625$

Tipo base float – esempio 2

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = …

mantissa normalizzata = …

M =

E =

rappr(5.375) = 0 . . . . . .

☺ Vedi Esercizi

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Esercizio

Dato $(796.25)_{10}$

Rappresentazione in binario = …

mantissa normalizzata = …

M =

E =

rappr(796.25) = 0 . . . . . ☺

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(796.25)_{10}$

Rappresentazione in binario = 1100011100.01

mantissa normalizzata = 1.10001110001

M = 10001110001

E = 9+127 = 136 $(10001000)_2$

rappr(796.25) = 0 10001000 10001110001000000000000

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(796.25)_{10}$

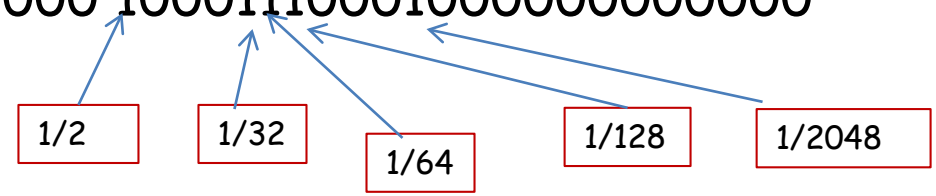
Rappresentazione in binario = 1100011100.01

mantissa normalizzata = 1.10001110001

M = 10001110001

E = 9+127 = 136 $(10001000)_2$

rappr(796.25) = 0 10001000 10001110001000000000000



prova: facciamo il calcolo indicato in alto a destra, per provare che la rappresentazione FP qui sopra è corretta ...

$$1.M \times 2^{E-127} = \ldots\ldots\ldots\ldots\ldots \times 2^{9} = 796.25$$

Tipo base float - sparsity

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

su 64 bit
- E è di 11 bit e M di 52;
- è eccesso 1023
- ordini di grandezza da 2^{-1023} a 2^{1024}

Gli ordini di grandezza dei numeri rappresentati sono ingenti (da 2^{-127} a 2^{128})

ma i numeri rappresentati tendono ad essere sempre più sparsi, man mano che crescono di grandezza:

infatti, per ogni esponente, c'è un numero costante di numerali (e quindi di numeri rappresentati): se M è di 23 bit, i possibili numerali sono 2^{23}

... più i numeri sono grandi, più sono sparsi ...

Esercizi – Floating Point

Calcolare il numero razionale rappresentato dalle seguenti notazioni FP (su 32 bit, 23 di mantissa)

11000100010001110001000000000000 (noto …)
11000100011001110001000000000000 (piccole grandi differenze)
11000000101011000000000000000000 (noto …)
01000100101011000010010000000000
11000010010011101100000000000000

Per i seguenti numeri razionali, scrivere la rappresentazione in FP (32 bit).

128796.70312500
125.1875
796.25 (noto …)
5509.8125

Esercizi – Floating Point

Calcolare il numero razionale rappresentato dalle seguenti notazioni FP (su 32 bit, 23 di mantissa)

```
11000100010001110001000000000000 -796,25
11000100011001110001000000000000 -924,25
11000000101011000000000000000000 -5,375
01000100101011000010010000000000 1377,125
11000010010011101100000000000000 -51,6875
```

Per i seguenti numeri razionali, scrivere la rappresentazione in FP (32 bit).

```
128796.70312500     01000111111110111000111001011010
125.1875            01000010111110100110000000000000
796.25              01000100010001110001000000000000
5509.8125           01000101101011000010111010000000
```

Tecniche della Programmazione, lez. 12 - seconda parte

- Approfondimenti

Tipo base char: codici interi

☺ provare questo codice in un programma e verificare l'output ...

```
char crt;
int cod;
  printf ("caro/a utente, dammi un carattere ...\n");
  scanf("%c", &crt);
  printf ("caro/a utente, mi hai dato il carattere %c avente
                                                codice %d\n", crt, crt);
… …    scanf("%d", &cod);
  printf ("caro/a utente, mi hai dato il codice %d che corrisponde
                                              al carattere %c\n", cod, cod);
```

```
caro/a utente, dammi un carattere ...
f
caro/a utente, mi hai dato il carattere f avente codice 102
caro/a utente, dammi un codice di carattere ...
-52
caro/a utente, mi hai dato il codice -52 che corrisponde al carattere
FINE
```

```
caro/a utente, dammi un carattere ...
f
caro/a utente, mi hai dato il carattere f avente codice 102
caro/a utente, dammi un codice di carattere ...
204
caro/a utente, mi hai dato il codice 204 che corrisponde al carattere
FINE
```

Ora vedi prossima slide

Tipo base char

Rappresenta i caratteri (di solito ASCII) in un byte.
Nel byte (la variabile di tipo char) c'è un codice virtualmente compreso nell'intervallo [0, 255]

```
char crt;
      crt = 'f';
      printf ("BLA … %c\n", crt);
```

(Costante carattere, tra apici)

```
A(65), B(66), …, Z(90), …, a(97), …, z(122)     0(48), …, 9(57)
                                !(33), #(35), {(123), }(125), ~(126)
```

E sapere questo ci puo` anche bastare …

Pero` , in effetti,

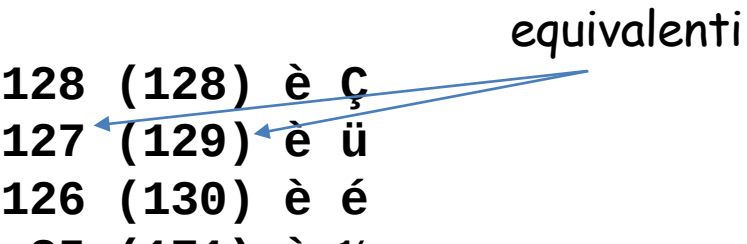
- mentre la **codifica ASCII ha codici da 0 a 255** (cioe` in [0.255]),
- la **codifica usata nel byte char** (cioe` il codice, il numero intero, contenuto nella locazione della variabile di tipo char) **è un po' diversa**, sebbene alla fine, equivalente:

In particolare:

- **i codici [-128, -1]** corrispondono ai **caratteri ASCII di codice [128, 255]**
- **i codici [0, 127]** corrispondono ai **caratteri ASCII di codice [0, 127]**

non agitiamoci … la cosa è trasparente … possiamo anche ignorarla, ma vale!
Ad esempio, con un codice che potete ~~facil~~ … uhm … ~~con ragionevole facil~~ uhm … con un po' di applicazione scrivere (ma lo trovate nei complementi didattici se non accettate la sfida) puo` venire fuori l'output mostrato nella prossima slide …
Ora vedi prossima slide (poi quella successiva, dopo aver provato a fare il programma …

Tipo base char: codici interi (-128??)




```
il carattere corrispondente al codice -128 (128) è Ç
il carattere corrispondente al codice -127 (129) è ü
il carattere corrispondente al codice -126 (130) è é
il carattere corrispondente al codice  -85 (171) è ½
il carattere corrispondente al codice  -84 (172) è ¼
il carattere corrispondente al codice   -1 (255) è
il carattere corrispondente al codice    0 (  0) è
il carattere corrispondente al codice   21 ( 21) è §
il carattere corrispondente al codice   35 ( 35) è #
il carattere corrispondente al codice   36 ( 36) è $
il carattere corrispondente al codice   37 ( 37) è %
il carattere corrispondente al codice   38 ( 38) è &
il carattere corrispondente al codice   39 ( 39) è '
il carattere corrispondente al codice   40 ( 40) è (
il carattere corrispondente al codice   41 ( 41) è )
il carattere corrispondente al codice   48 ( 48) è 0
il carattere corrispondente al codice   64 ( 64) è @
il carattere corrispondente al codice   65 ( 65) è A
il carattere corrispondente al codice   90 ( 90) è Z
il carattere corrispondente al codice   97 ( 97) è a
il carattere corrispondente al codice  121 (121) è y
```

Tipo base char: codici interi (-128?) - come abbiamo fatto

```
#include <stdio.h>
int main () {
  char c;
  int cod, c1;

  cod = -128;              /* inizializzazione */
  while (cod <= 127) {
      if (cod<0)
         c1=cod+256;
      else c1 = cod;

      printf ("il car… %4d (%3d) …%c\n", cod, c1, cod);

      cod = cod + 1;              /* modifica variabile di test */
  }

printf("FINE\n");
return 0;
}
```

Con istruzione iterativa WHILE

Nelle prossima slide si vede una versione di questo programma che fa uso di un'espressione condizionale. Quindi … **torna alla lezione**, da lì proseguirai per fare un paio di esercizi sull'espressione condizionale, e poi tornerai qui e proseguirai con la prossima slide.

Espressione condizionale per il programma sui caratteri ... e poi obfuscation

Usiamo una espressione condizionale per abbreviare il codice della slide precedente

```
cod = -128;                /* inizializzazione */
while (cod <= 127) {
    printf ("il carattere corrispondente al
             codice %4d (%3d) è %c\n",
             cod, (cod<0? cod+256 : cod), cod);

    cod = cod + 1;                /* modifica variabile di test */
}
```

dovresti essere qui dopo aver fatto gli esercizi sull'uso dell'espressione condizionale in C.
Se non li hai fatti falli subito.

Poi prova a fare l'esecuzione simulata di alcune iterazioni di questo codice, per vedere che output viene (e se l'output e` uguale a quello precedentemente visto)
Poi vedi la slide successiva: Obfuscation!

Visto che ci siamo … Obfuscation

Se usiamo contemporaneamente l'espressione condizionale e quella di post incremento otteniamo in questo caso un codice molto più compresso (corto) e forse più difficile da comprendere

EXP_C (condizione) ? (exp1) : (exp2)
se la condizione vale TRUE, EXP_C ritorna il valore di exp1;
altrimenti EXP_C ritorna il valore di exp2.

post-incremento cod++
(restituisce il valore di cod prima di incrementarlo)

```
cod = -128;                /* inizializzazione */
while (cod <= 127)
    printf ("il carattere corrispondente al
            codice %4d (%3d) è %c\n",
            cod, (cod<0? cod+256 : cod), cod++);
```

```
/* la modifica della variabile di test avviene nella valutazione del
postincremento */
```

Tecniche della Programmazione, lez. 12 - seconda parte

- Esercizi

Espressione Condizionale - esercizio 1

Aggiungere la definizione della funzione isEven() ... (cioè "è pari") per far funzionare bene il programma proposto.
La funzione riceve un valore intero e restituisce 1 o 0, a seconda se il valore è pari o dispari.
La funzione deve usare un'espressione condizionale per determinare quale valore restituire

```
#include <stdio.h>

/* prototipo isEven() */
int isEven(int);

int main() {
    int num = 25;

    if (isEven(num)) {
        printf("%d e` pari\n", num);
    } else printf("%d e` dispari\n", num);
    }
printf("\nFINE\n");
return 0;
}

/* definizione isEven() */
int isEven(int numero){
  int risult;
  risult = (num%2 == 0) ? 1 : 0;
return risult;
}
```

Espressione Condizionale - esercizio 2

Il programma qui sotto risolve un'equazione di secondo grado, i cui coeff. sono dati in input. Aggiungere la definizione della funzione soluzioniComplesse(), che riceve i coefficienti e restituisce 1 se sono ammesse soluzioni complesse coniugate, 0 se sono ammesse soluzioni reali. La funzione deve usare un'espressione condizionale per determinare quale valore restituire

```
#include <stdio.h>
#include <math.h>
          /* prototipo soluzioniComplesse() */
          int soluzioniComplesse(double, double, double);

int main(){  double a, b, c, delta, x1, x2, parteReale, parteImmag;

  printf("Inserisci i coefficienti dell'equazione ax^2 + bx + c = 0:\n");
  scanf("%lf %lf %lf", &a, &b, &c);

  delta = b * b - 4 * a * c;
  if (soluzioniComplesse(a, b, c)) {
      parteReale = -b / (2 * a);
      parteImmag = sqrt(-delta) / (2*a);
      printf("Le soluzioni sono complesse: %.2lf + %.2lfi e %.2lf - %.2lfi\n",
              parteReale, parteImmag, parteReale, parteImmag);
  } else {
      x1 = (-b + sqrt(delta)) / (2*a);
      x2 = (-b - sqrt(delta)) / (2*a);
      printf("Le soluzioni sono reali: %.2lf e %.2lf\n", x1, x2);
  }
printf"\nFINE);
return 0;
}

/* definizione soluzioniComplesse() */
int soluzioniComplesse(double c1, double c2, double c3) {
   double delta = c2*c2 - 4*c1*c3;
return delta < 0 ? 1 : 0;
}
```

Espressione Condizionale - esercizio 3

Aggiungere la definizione della funzione maxTra3(), che riceve tre numeri interi e restituisce il massimo tra loro.
La funzione deve usare un'espressione condizionale per determinare quale valore restituire.

```
#include <stdio.h>
        /* prototipo maxTra3() */
        int maxTra3(int, int, int);

int main() {
  int numero1=29, numero2=12, numero3=18;

  print("il massimo tra %d, %d, and %d e`: ", numero1, numero2, numero3);
  print("%d\n", maxTra3(numero1, numero2, numero3));

printf("\nFINE\n");
return 0;
}

/* definizione maxTra3() */
int maxTra3(int n1, int n2, int n3) {
   int risult = (n1 > n2) ? ((n1 > n3) ? n1 : n3) : ((n2 > n3) ? n2 : n3

return risult;
}
```

E questo era complicato, lo so.
C'e` infatti unpo` di ***obfuscation***.

Tipo base float – esempio 2

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = ...

M =

E =

rappr(5.375) = 0 . . . . . ☺

Tipo base float – esempio 2

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M =

E =

rappr(5.375) = 0 . . . . . ☺

Tipo base float – esempio 2

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit …)

E =

rappr(5.375) = 0 . . . . . ☺

Tipo base float – esempio 2

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit …)

E = 2 + 127 = (☺)$_{10}$ = (☺)$_2$

rappr(5.375) = 0 . . . . . ☺

Tipo base float - esempi

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit ...)

E = 2 + 127 = $(129)_{10}$ = (☺)$_2$

rappr(5.375) = 0 . . . . . ☺

Tipo base float – esempio 2

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit …)

E = 2 + 127 = $(129)_{10}$ = $(10000001)_2$

rappr(5.375) = 0 . . . . . .

☺

Tipo base float – esempio 2

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

$(127_{10} = 01111111_2)$

Esercizio

Dato $(5.375)_{10}$

Rappresentazione in binario = 101.011

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit …)

E = 2 + 127 = $(129)_{10}$ = $(10000001)_2$

rappr(5.375) = 0 10000001 01011000000000000000000

Tipo base float – esempio 2

Su 32 bit si usano
- 1 bit per il segno (s) della mantissa
- 8 bit per l'esponente (E)
- 23 bit per la mantissa (M)

Il numero rappresentato nei 32 bit è calcolato come

$$(-1)^s \times 1.M \times 2^{E-01111111}$$

($127_{10} = 01111111_2$)

Esercizio

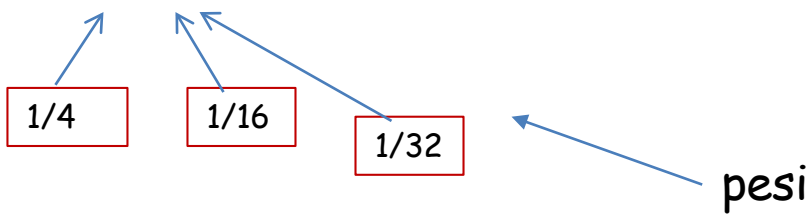
Dato $(5.375)_{10}$

mantissa normalizzata = 1.01011

M = 01011 (la parte che va nei 23 bit …)

E = 2 + 127 = $(129)_{10}$ = $(10000001)_2$

rappr(5.375) = 0 10000001 01011000000000000000000



prova: facciamo il calcolo indicato in alto a destra, per provare che la rappresentazione FP qui sopra è corretta …

$$1.M \times 2^{E-127} = \ldots\ldots\ldots\ldots\ldots = 5.375$$