Funzioni di libreria

KP pp.112-115, 471-474, 573-575, 589, 598-601

DD pp.144-153, 307-313, 563-565, 588-591

Funzioni matematiche (1) KP p. 112,573

Il linguaggio C non contiene funzioni matematiche predefinite. Funzioni come:

`sqrt()`, `pow()`, `exp()`, `log()`, `sin()`, `cos()`, `tan()`

sono disponibili nella libreria matematica `math.h`.

Tutte queste funzioni, eccetto pow(), ricevono un singolo parametro di tipo double e restituiscono un risultato di tipo double.

La funzione pow() riceve due parametri di tipo double e restituisce un risultato di tipo double.

Funzioni matematiche (2)

Esempio.

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

    int main()
    { double x;
      scanf("%lf",&x);
      if (x>=0.0)        /*altrimenti non posso eseguire sqrt*/
         printf("x=%lf, rad%lf, pot=%lf",x, sqrt(x), pow(x,x));
     else printf("il numero deve essere non negativo");
    }
```

Eseguendo il programma ed inserendo 2 (oppure 2.0, oppure 2e0, oppure 0.2e1) si ottiene:
x=2.000000e+00, rad=1.414213e+00, pot=4.000000e+00

N.B. nel codice C, 2 e 2.0 non sono equivalenti, ma nella lettura da input sì!

Funzioni matematiche (3)

Il *dominio* di una funzione matematica è l'insieme dei valori per cui essa è definita. Si verifica un errore di dominio quando una funzione mat. viene richiamata con un parametro non appartenente al suo dominio. Si verifica un errore di intervallo quando il valore che deve essere restituito dalla funzione risulta matematicamente definito, ma non può essere rappresentato (overflow o underflow).

Esempi.

Per la funzione `asin()` si verifica un errore di dominio se il parametro non si trova nell'intervallo [-1,1].

Per la funzione `log()` si verifica un errore di dominio se il parametro è negativo, ed un errore di intervallo se il parametro è 0, perché il logaritmo di 0 non può essere rappresentato (il suo limite vale ∞).

Funzioni matematiche (4)

`double exp(double x);`

restituisce e^x

`double log(double x); double log10(double x);`

restituiscono il logaritmo naturale (base e) ed in base 10 di x

N.B. Per calcolare il log in base 2? tener presente che $\log_a b = \log_c b / \log_c a$, e porre a=2 e c=e oppure c=10.

`double ceil(double x); double floor(double x);`

restituiscono la parte intera superiore ed inferiore di x. Attenzione: anche se il risultato potrebbe essere intero, il risultato è memorizzato in un double per evitare overflow.

`double fabs(double x); int abs(int x);`

restituiscono il valore assoluto di x. Attenzione: `abs` è in `stdlib.h`, mentre `fabs` in `math.h`

Esercizi

Un'applicazione della funzione floor è l'arrotondamento di un valore all'intero più vicino: y=floor(x+.5); scrivere un programma che legga un numero da input e lo arrotondi all'intero più vicino.
Similmente all'esercizio precedente, scrivere un programma che legga un numero reale da input e, utilizzando floor, lo arrotondi al decimo o al centesimo (hint: usare le formule y=floor(x*10+.5)/10; e y=floor(x*100+.5)/100;
Scrivere una funzione che prenda in input una coppia di interi e determini se il secondo sia un multiplo del primo.
Scrivere una funzione che prenda in input un intero e restituisca l'intero che ha le cifre invertite (ad es. 1487 -> 7841)

Funzioni per i caratteri

Le funzioni per i caratteri si trovano nella libreria `ctype.h`

`int isspace(int c);`

restituisce 1 se c è un carattere di spaziatura (newline '\n', spazio ' ', tabulazione '\t', ecc.) e 0 in caso contrario

`int isdigit(int c); int isalpha(int c);`

restituiscono 1 se c è una cifra (risp. una lettera), 0 in caso contrario.

`int islower(int c); int isupper(int c);`

restituiscono 1 se c è una lettera minuscola (risp. maiuscola) e 0 in caso contrario

`int tolower(int c); int toupper(int c);`

restituiscono una lettera minuscola (risp. maiuscola) se c è maiuscola (risp. minuscola), e c stesso in caso contrario

Manipolazione di caratteri (1)

```
int isdigit(int c)
{
  return ((c>='0')&&(c<='9'));
}
```

```
int tolower(int c)
{
  if isupper( c )
     c=c-'A'+'a';
  return c;
}
```

```
int isalpha(int c)
{
  return (((c>='a')&&(c<='z')) ||
          ((c>='A')&&(c<='Z')));
}
```

Manipolazione di caratteri (2)

Esempio: studio della funzione islower

```
#include<stdio.h>
#include<ctype.h>
int main
 {
    printf("%s%s\n%s%s\n%s%s",
       islower('p')?"p è":"p non è",
       "un carattere minuscolo",
       islower('5')?"5 è":"5 non è",
      "un carattere minuscolo",
      islower('!')?"! è":"! non è"
      "un carattere minuscolo");
}
```

Ricorda:

```
(espl)?esp2:esp3;
```

è equivalente a:

```
if (espl)
   esp2;
else esp3;
```

Output:

p è un carattere min.
5 non è un car. min.
! non è un car. min.

Esercizi

- Provare a scrivere in C tutte le principali funzioni di manipolazioni di caratteri

Funzioni data e ora (1) KP p. 471,598

```
long time(long *p);
```

restituisce un valore intero ottenuto dal clock interno, pari al numero di secondi trascorsi dal 1/1/1970 (ma sono possibili anche altre unità di tempo ed altri punti di partenza).
Se il parametro non è NULL, il valore restituito viene assegnato anche alla variabile puntata.

Per verificare la durata di un programma:

```
#include <stdio.h>
#include <time.h>
int main()
{
  long inizio, fine;
  inizio=time(NULL);
  ...
  fine=time(NULL);
  printf("tempo: %ld", fine-inizio);
  return 0;
}
```

N.B. funziona solo per programmi molto lenti (dell'ordine dei secondi!!)

Funzioni data e ora (2)

`char *ctime(long *ptime);`

ptime deve essere l'indirizzo di una variabile che memorizza un tempo ritornato da time; ctime converte tale valore in una stringa fornita dal sistema.
Esempio: Tue Oct 17 14:33:50 2006

`CLOCKS_PER_SEC`

dipendente dalla macchina; definisce il numero di "tic" del clock della CPU al secondo. Costante definita in `time.h`

`long clock(void)`

restituisce un'approssimazione del numero di "tic" del clock della CPU utilizzati dal programma fino al punto della chiamata. Il valore restituito può essere diviso per `CLOCKS_PER_SEC` per essere convertito in secondi.
Anche questa funzione può essere usata per valutare il tempo di esecuzione (N.B. non vengono computati i tempi di attesa dell'utente)

Funzioni data e ora (3)

```
#include <stdio.h>
#include <time.h>
#define N 100000000

int main()
{
float a=3.333, b=5.555, x;
double c=3.333, d=5.555, y;
int i; long tempo1,tempo2;

for (i=1; i<N; i++)
  x=a*b;
tempo1=clock();
for (i=1; i<N; i++)
  y=c*d;
tempo2=clock();
printf("tempi float: %ld\n", tempo1);
printf("tempi double: %ld\n", tempo2-tempo1);
return 0;
}
```

Esempio: Confronta i tempi di esecuzione della moltiplicazione in singola precisione con quella in doppia precisione

Sorprendentemente, l'output è stato:
tempi float: 119
tempi double: 110
(forse perché i float vengono promossi in double e si perde tempo nella conversione…)

Numeri pseudocasuali (1) KP p.589

PREMESSA: generare numeri casuali è impossibile, perciò parliamo di numeri pseudocasuali…

`int rand(void); void srand(unsigned x);`

Queste funzioni sono in `stdlib.h`

ogni chiamata di `rand()` genera e restituisce un intero; chiamate ripetute generano una sequenza di interi che risultano uniformemente distribuiti nell'intervallo [0, RAND_MAX]. RAND_MAX dipende dalla macchina, ma è tipicamente 32767.

La sequenza prodotta da `rand` è sempre la stessa, poiché la sequenza è pseudocasuale. La funzione `srand()` inizializza il generatore di numeri casuali, in modo che più sequenze generate da `rand()` inizino da un valore diverso. Se non specificato, `rand()` agisce come se fosse stata chiamata `srand(1)`.

L'istruzione `srand(time(NULL));` si utilizza per inizializzare il generatore casuale con un valore differente a ogni esecuzione del programma.

Numeri pseudocasuali (2) DD p.143

ATTENZIONE: L'intervallo dei valori prodotti da `rand` è spesso diverso da quello che serve nelle specifiche applicazioni. Ad esempio, se si simula il lancio di una moneta: [0,1]; se si simula il lancio di un dado: [0,5];…

Per ovviare a questo inconveniente si può usare la funzione `rand` in congiunzione con l'operatore `%`

Esempio.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int i;
  for (i=1; i<=20; i++)
     printf("%d ", 1+(rand()%6));
  return 0;
}
```

Numeri pseudocasuali (3)

Esempio: Verifichiamo la pseudocasualità dei numeri generati dalla funzione `rand` con una statistica.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int i, uscita;
  int f1=0; int f2=0; int f3=0; int f4=0; int f5=0; int f6=0;

  for (i=1; i<=1000; i++)
    {uscita= 1+(rand()%6);
     switch (uscita) {
     case 1:
        ++f1;
        break;
     case 2:
        ++f2;
        break;
     ...
     }
  }
```

```
  printf("1: %d\n", f1);
  printf("2: %d\n", f2);
  ...
  return 0;
}
```

```
1: 966
2: 1017
3: 988
4: 1060
5: 971
6: 998
```

Esercizi

- Dichiarare la variabile median=RAND_MAX/2.0, scrivere un programma che:
  - chiami rand 500 volte in un ciclo for
  - che conti il numero di volte in cui il valore è, rispettivamente, maggiore o minore di median
  - che stampi ad ogni iterazione la differenza delle due frequenze. Tale valore dovrebbe oscillare intorno allo 0, man mano che il numero di iterazioni cresce.
- Scrivere una singola istruzione che visualizzi un numero casuale tratto dall'insieme {2,4,6,8,10}.