Programmazione I

Il Linguaggio C

Strutture di Controllo

Daniel Riccio

Università di Napoli, Federico II

13 ottobre 2021

Sommario

- Argomenti
  - Le funzioni matematiche
  - Casting esplicito
  - Soluzione degli esercizi assegnati
  - Problemi/Algoritmi/Programmi

Le funzioni matematiche

#include <**math.h**> è l'header file della libreria standard del C che contiene definizioni di macro, costanti e dichiarazioni di funzioni e tipi usati per le operazioni matematiche.

FUNZIONE MATEMATICA		FUNZIONE C
$\sqrt{x}$	radice quadrata di *x* (restituisce un valore ≥0)	`double sqrt (double x);`
sen x	seno di *x* (con *x* reale)	`double sin (double x);`
cos x	coseno di *x* (con *x* reale)	`double cos (double x);`
tg x	tangente di *x* (con *x* reale)	`double tan (double x);`
arcsen x	arcoseno di *x* (con *x* nell'intervallo [-1 ;+1])	`double asin (double x);`
arccos x	arcoseno di *x* (con *x* nell'intervallo [-1 ;+1])	`double acos (double x);`
arctg x	arcotangente di *x* (con *x* reale)	`double atan (double x);`
e^x	*e* elevato ad *x* (con *x* reale)	`double exp (double x);`
10^x	10 elevato a *x* (con *x* reale)	`double pow10 (double x);`
ln x	logaritmo in base *e* di *x* (con *x* reale ≥0)	`double log (double x);`
log x	logaritmo di base 10 di *x* (*x* reale positivo)	`double log10 (double x);`
|x|	valore assoluto di *x*	`double fabs (double x);`
	calcolo ipotenusa di un triangolo rettangolo i di cateti *x* ed *y* (con *x*,*y* numeri reali)	`double hypot (double x, double y);`
arctg x/y	arcotangente di *y*/*x*. Il valore restituito è compreso fra $-\pi$ e $+\pi$ estremi compresi	`double atan2 (double y, double x);`
y^x	*x* elevato ad *y* (con *x*,*y* numeri reali)	`double pow (double x, double y);`
	calcola mantissa ed esponente di *x*	`double frexp (double x, int* esp);`

Le funzioni matematiche

Tutte le funzioni in C restituiscono dei valori **double** invece che **float**, in modo da garantire la massima precisione

Le due espressioni matematiche seguenti sono tradotte in C

$$y = \frac{ax^3 - 3x^5}{2ab}$$

L'espressione si scrive con l'istruzione di assegnazione seguente:

```
y = (a* pow(x,3) - 3*pow(x,5)) /(2*a*b)
```

Oppure

```
y = (a* pow(x,3) - 3*pow(x,5)) /2/a/b
```

$$z = \frac{\sqrt{x^4 - 3}}{|x - 1|}$$

L'espressione si scrive con l'istruzione di assegnazione seguente:

```
z = sqrt(pow(x,4) - 3) /fabs(x-1)
```

Casting di tipo implicito

Il C esercita un controllo sui **tipi** e dà un messaggio di errore quando si tenta di eseguire operazioni fra operandi di tipo non ammesso

Durante l'esecuzione di un programma ci sono situazioni in cui viene cambiato il tipo di dato (casting implicito):

– quando un valore di un tipo viene assegnato a una variabile di un altro tipo compatibile;
– quando un'operazione contiene due elementi di tipo diverso (per esempio la somma di un intero con un reale);

Questo può provocare degli **errori difficilmente individuabili**

Casting di tipo implicito

I quattro operatori matematici si applicano a qualsiasi tipo standard, ma i tipi dei due operandi **devono essere uguali**

Nel caso di due tipi diversi, il compilatore esegue una conversione di tipo **implicita** su uno dei due operandi, seguendo la regola di promuovere il tipo più semplice (meno ampio in bytes) a quello più complesso (più ampio in bytes)

Il compilatore considera una gerarchia (in ordine crescente di complessità):
char → unsigned char → short → unsigned short → long → unsigned long → float → double → long double

Esempio: 3.4/2 il secondo operando è trasformato in 2.0 e il risultato è correttamente 1.7

Casting di tipo implicito

Nelle assegnazioni, il tipo dell'operando di destra viene sempre convertito implicitamente nel tipo dell'operando di sinistra

Se il tipo dell'espressione ha una precisione minore del tipo della variabile, come in

Variabile-double = espressione-float

allora non c'è alcun problema; se, invece, accade il contrario, come in

Variabile-float = espressione-double

potrebbero nascere due tipi di problemi:

1 - il risultato dell'espressione-double contiene un numero di cifre che il tipo float non gestisce, con una grave **perdita di precisione**;

2 - il risultato dell'espressione-double supera il massimo valore rappresentabile come float: in tal caso il programma potrebbe avere un **comportamento imprevedibile**.

Esempio:

```
int c;
double d;
c = d;
```

(**warning in fase di compilazione**)

Test

Se **n1** ed **n2** sono variabili di tipo **int** , **r** una variabile di tipo **float** e **c** una variabile di tipo **char** dire cosa stampa il computer allorché esegue le seguenti istruzioni:

```
int n1, n2;
float r;
char c;

n1 = 11;
n2 = 3;
```

```
printf("n1*n2=%d \n",n1*n2);
```

n1*n2=33

```
c='H';
n2=n1%n2;
printf("c=%c \n n2=%d \n",c,n2);
```

c=H
n2=2

```
r=n1/n2;
n1=n1/n2;
printf("n1=%d \n r=%f",n1,r);
```

n1=5
r=5.000000

Casting di tipo esplicito

L'**operatore di casting** (o di **conversione esplicita**), serve per forzare una conversione di tipo (ha due operandi)

in C equivale a:

(**tipo**) **variabile**
tipo (**variabile**)

consiste nell'indicazione del nuovo tipo fra parentesi davanti al nome della variabile da trasformare

Tutti i tipi standard consentono il casting, se la variabile da trasformare è l'operando di una certa operazione, il tipo risultante deve essere fra quelli ammissibili (altrimenti viene generato un errore in compilazione)

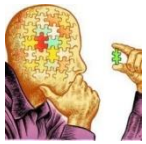
Esempio:
float r;
r =(**float**)**3**/4;

é corretto e assegna ad **r** il valore **0.75**

Esempio:
float(**n**) **%** **3** é errato in quanto l'operatore % ammette solo operandi interi

Esercizio

Scrivere un programma che prende in input tre valori interi positivi **a**, **b**, **c** e stampa a video il valore massimo **max**(**a**, **b**, **c**).
Nota: senza l'uso di costrutti condizionali.



Ragionamento

L'espressione (**a**≥**b**) assume valore **1** se **a** è maggiore di **b**, **0** altrimenti.
L'espressione (**a**≥**c**) assume valore **1** se **a** è maggiore di **c**, **0** altrimenti.

L'espressione (**a**≥**b**)*(**a**≥**c**) assume valore **1** se **a** è il massimo, **0** altrimenti.

L'espressione **a***(**a**≥**b**)*(**a**≥**c**) assume valore **a** se **a** è il massimo, **0** altrimenti.

Idea

Combino le tre espressioni con una somma

max(**a**, **b**, **c**) = **a***(**a**≥**b**)*(**a**≥**c**) + **b***(**b**≥**a**)*(**b**≥**c**) + **c***(**c**≥**a**)*(**c**≥**b**)

Esercizio

Algoritmo

Dati **a**,**b**,**c**
A ← **a***(**a**≥**b**)*(**a**≥**c**)
B ← **b***(**b**≥**a**)*(**b**≥**c**)
B ← **c***(**c**≥**a**)*(**c**≥**b**)
Massimo ← A+B+C
stampa(Massimo)

È corretto?

Esecuzione

```
Il massimo fra 10, 10 e 1 è il valore 20
```

Algoritmo

Dati **a**,**b**,**c**
A ← **a***(**a**≥**b**)*(**a**≥**c**)
B ← **b***(**b**≥**a**)*(**b**≥**c**)*(**a**≠**b**)
B ← **c***(**c**≥**a**)*(**c**≥**b**)*(**c**≠**a**)*(**c**≠**b**)
Massimo ← (A+B+C);
stampa(Massimo)

È corretto?

Esecuzione

```
Il massimo fra 10, 10 e 1 è il valore 10
```

Esercizio

Algoritmo

Dati **a**,**b**,**c**

A ← **a***(**a**≥**b**)*(**a**≥**c**)

B ← **b***(**b**≥**a**)*(**b**≥**c**)*(**a**≠**b**)

B ← **c***(**c**≥**a**)*(**c**≥**b**)*(**c**≠**a**)*(**c**≠**b**)

Massimo ← A+B+C

stampa(Massimo)

È corretto?

Esecuzione

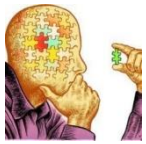
```
Il massimo fra 10, 10 e 1 è il valore 10
```

Programma (Esercizio_004_a.c)

main.c

```c
#include <stdio.h>

int main()
{
  int a = 1;
  int b = 10;
  int c = 1;

  int A = 0;
  int B = 0;
  int C = 0;

  int Massimo = 0;
  int divisore = 0;

  A = a*(a>=b)*(a>=c);
  B = b*(b>=a)*(b>=c)*(b!=a);
  C = c*(c>=a)*(c>=b)*(b!=c)*(a!=c);

  Massimo = (A + B + C);

  printf("Il massimo fra %d, %d e %d è il valore %d\n", a, b, c, Massimo);

  return 0;
}
```

Determinare se un numero è intero



Ragionamento

Un numero intero non ha una parte decimale

Idea

Confrontiamo il numero con la sua parte intera

Algoritmo

Leggo **numero**
Pongo la parte intera di **numero** in una variabile intera **tmp**
Confronto **numero** e **tmp**
stampa(messaggio)

Programma (Esercizio_006_a.c)

main.c

```c
/*************************************************
 *
 * Esercizio_006_a.C
 * verifica se un numero letto in input è intero
 *
 * ************************************************/
#include <stdio.h>
#include <math.h>

int main()
{
  float numero;
  int tmp;

  printf("inserisci un numero: ");
  scanf("%f", &numero);

  tmp = (int)numero;

  if(numero==tmp)
    printf("Il numero %f è un intero\n",numero);
  else
    printf("Il numero %f non è un intero\n",numero);

  return 0;
}
```

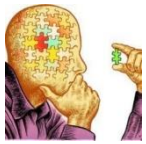
Determinare se un numero è intero

Programma (Esercizio_006_b.c)

main.c

```c
/*************************************************
 *
 * Esercizio_006_b.C
 * verifica se un numero letto in input è intero
 *
 * ***********************************************/
#include <stdio.h>
#include <math.h>

int main()
{
  float numero;
  double tmp;

  printf("inserisci un numero: ");
  scanf("%f", &numero);

  tmp = floor(numero);

  if(numero==tmp)
    printf("Il numero %f è un intero\n",numero);
  else
    printf("Il numero %f non è un intero\n",numero);

  return 0;
}
```

Determinare se un numero intero è pari



Ragionamento

Un numero intero è pari se è divisibile per 2

Idea

Verifico se il resto della divisione per 2 è zero

Algoritmo

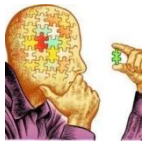
Leggo **numero**
Se resto (**numero**/**2**==0)
  stampa(“Il numero è pari”)
altrimenti
  stampa(“Il numero è dispari”)

Programma (Esercizio_007.c)

main.c

```c
/*************************************************
 *
 * Esercizio_007.C
 * verifica se un numero intero letto in input è pari
 * o dispari
 *
 * ************************************************/
#include <stdio.h>
#include <math.h>

int main()
{
  int numero;

  printf("inserisci un numero: ");
  scanf("%d", &numero);

  if((numero%2)==0)
    printf("Il numero %d è pari\n",numero);
  else
    printf("Il numero %d è dispari\n",numero);

  return 0;
}
```

Stampare l'i-esimo bit di un numero



Ragionamento

Utilizzare gli operatori logici bit a bit permette di mascherare i bit di interesse

Idea

Shiftiamo un **1** di **i** posizioni ed effettuiamo un **AND** logico con il numero in input

Algoritmo

Leggo **numero**
Leggo **i**

Poni **bit_pos** a 1

bit_pos ← 1 << (**i**-1)
bit ← **numero** **AND** **bit_pos**
bit ← 1 >> (**i**-1)

stampa(**bit**)

Il conteggio dei bit comincia da 0

Stampare l'i-esimo bit di un numero

Programma (Esercizio_008.c)

main.c

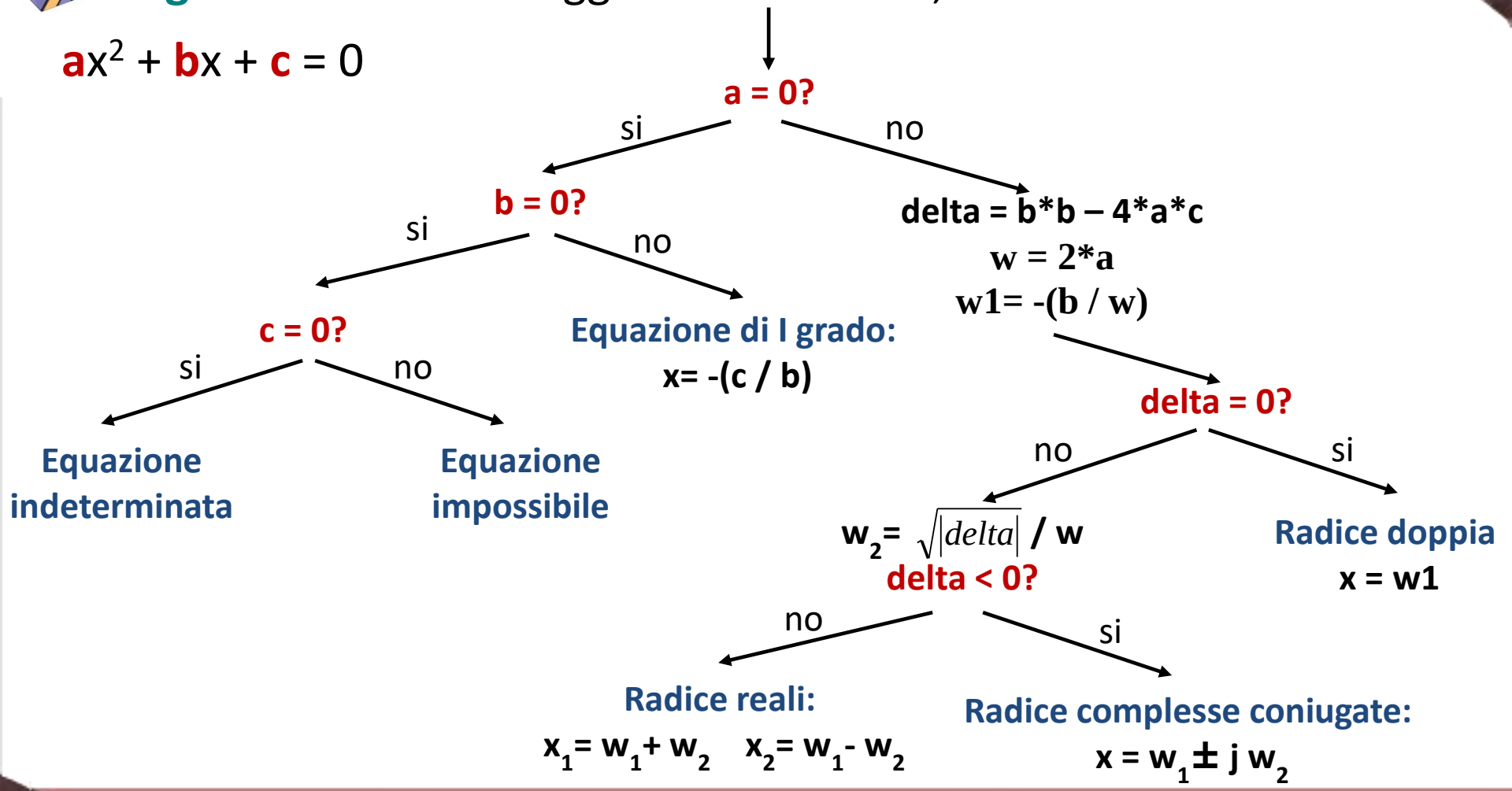
```c
/*************************************************
 *
 * Esercizio_008.C
 * Stampa il bit in posizione i-esima di un numero
 * intero N
 *
 * ************************************************/
#include <stdio.h>

int main()
{
  int numero;
  int i;
  int bit, bit_pos;

  printf("inserisci un numero: ");
  scanf("%d", &numero);
  printf("inserisci la posizione: ");
  scanf("%d", &i);

  if(i>(8*sizeof(int)))
    printf("La posizione indicata eccede il numero di bit di un intero\n");
  else {
    bit_pos = 1 << (i-1);
    bit = numero & bit_pos;
    bit = bit >> (i-1);
    printf("L'i-esimo bit di %d è %d\n",numero, bit);
  }

  return 0;
}
```

In modo equivalente
bit >>= (i-1);

Equazione di secondo grado

Algoritmo

$ax^2 + bx + c = 0$



Lo swap di due interi

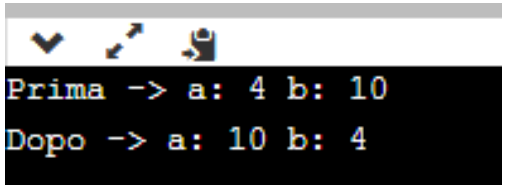
Dati due numeri interi n1 e n2 scrivere un programma che scambi i due valori

Programma

main.c

```c
#include <stdio.h>

int main()
{
  int a=4, b=10;
  int tmp;

  printf("Prima -> a: %d b: %d\n", a, b);

  tmp = a;
  a = b;
  b = tmp;

  printf("Dopo -> a: %d b: %d\n", a, b);

  return 0;
}
```

Esecuzione



Ordinamento di tre interi

Dati tre numeri interi n1, n2 e n3, stampare i tre numeri in ordine crescente.

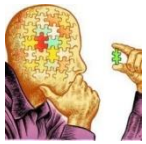
Programma

main.c

```c
#include <stdio.h>

int main()
{
  int n1, n2, n3;
  int tmp;

  printf("Inserisci il primo numero: ");
  scanf("%d", &n1);
  printf("Inserisci il secondo numero: ");
  scanf("%d", &n2);
  printf("Inserisci il terzo numero: ");
  scanf("%d", &n3);

  printf("Prima -> n1: %d n2: %d n3: %d\n", n1, n2, n3);

  // primo scambio
  if(n1>n2){
    tmp = n1;
    n1 = n2;
    n2 = tmp;
  }

  // secondo scambio
  if(n1>n3){
    tmp = n1;
    n3 = n1;
    n1 = tmp;
  }

  // terzo scambio
  if(n2>n3){
    tmp = n2;
    n2 = n3;
    n3 = tmp;
  }

  printf("Dopo -> n1: %d n2: %d n3: %d\n", n1, n2, n3);

  return 0;
}
```

Ordinamento di tre interi

Dati tre numeri interi n1, n2 e n3, stampare i tre numeri in ordine crescente senza copiare o modificare il contenuto delle tre variabili che li contengono.



Ragionamento

I puntatori corrispondono alla cella di memoria che contiene una variabile

Idea

Definiamo dei puntatori alle variabili originarie e scambiamo i puntatori

Algoritmo

Leggo **n1**, **n2**, **n3**
Dichiariamo tre puntatori **p1**, **p2**, **p3**
p1 ← **&n1**
p2 ← **&n2**
p3 ← **&n3**
if(***p1**>***p2**)
  scambia(**p1**,**p2**)
if(***p1**>***p3**)
  scambia(**p1**,**p3**)
if(***p2**>***p3**)
  scambia(**p2**,**p3**)

stampa(***p1**, ***p2**, ***p3**)

Ordinamento di tre interi

Dati tre numeri interi n1, n2 e n3, stampare i tre numeri in ordine crescente senza modificare il contenuto delle tre variabili che li contengono.

Programma

main.c

```c
#include <stdio.h>

int main()
{
  int n1, n2, n3;
  int *p1, *p2, *p3;
  int *tmp;

  printf("Inserisci il primo numero: ");
  scanf("%d", &n1);
  printf("Inserisci il secondo numero: ");
  scanf("%d", &n2);
  printf("Inserisci il terzo numero: ");
  scanf("%d", &n3);

  p1 = &n1;
  p2 = &n2;
  p3 = &n3;

  printf("Prima -> n1: %d n2: %d n3: %d\n", n1, n2, n3);

  // primo scambio
  if(*p1>*p2){
    tmp = p1;
    p1 = p2;
    p2 = tmp;
  }

  // secondo scambio
  if(*p1>*p3){
    tmp = p1;
    p3 = p1;
    p1 = tmp;
  }

  // terzo scambio
  if(*p2>*p3){
    tmp = p2;
    p2 = p3;
    p3 = tmp;
  }

  printf("Dopo con puntatori -> n1: %d n2: %d n3: %d\n", *p1, *p2, *p3);
  printf("Dopo con variabili -> n1: %d n2: %d n3: %d\n", n1, n2, n3);

  return 0;
}
```

Esecuzione

```
Inserisci il primo numero: 7
Inserisci il secondo numero: 1
Inserisci il terzo numero: 5
Prima -> n1: 7 n2: 1 n3: 5
Dopo -> n1: 1 n2: 5 n3: 7
Dopo variabili -> n1: 7 n2: 1 n3: 5
```

Esercizi

1. Dati i tre lati di un triangolo, calcolare i tre angoli del triangolo