Funzioni – Esercizio 1

Creare una funzione **`float square(float x)`** . La funzione deve restituire il quadrato del parametro **`x`**.

Creare un'altra funzione, di nome **`float cube(float x)`**, che restituisce invece il cubo del valore **`x`**.

Progettare quindi e codificare un programma che legge un float da tastiera e restituisce il suo quadrato ed il suo cubo. Per calcolare il quadrato ed il cubo si devono utilizzare le due funzioni sopra definite.

Funzioni – Soluzione 1 (1)

```
#include <stdio.h>

float quadrato (float x) {
     x=x*x;
     return x;
}

float cubo (float x) {
     x=x*x*x;
     return x;
}
```

Funzioni – Soluzione 1 (2)

```
float leggi () {
     float valore;
     printf("\nInserisci un numero reale: ");
     scanf ("\n%f",&valore);
     return valore;
}

void main () {
     float n, q, c;
     n = leggi();
     q = quadrato(n);
     c = cubo(n);
printf("Elevato al quadrato risulta %f", q);
printf("Elevato al cubo risulta %f\n", c);
}
```

Funzioni – Esercizio 2

Si scriva una funzione

```
int somma_potenze(int a,int n);
```

che dati **a** e **n** deve calcolare $\sum_{i=1}^{n} a^i$

A tal fine si scriva una funzione

```
int potenza(int x,int y);
```

che dati **x** e **y** deve calcolare x^y usando come operazione primitiva il prodotto.

Funzioni – Soluzione 2 (1)

```
#include <stdio.h>
int leggi() {
     int valore;
     scanf("\n%d",&valore);
     return valore;
}

int pot(int x, int y) {
     int i, pot=1;
     for(i=0;i<y;i++)
          pot=pot*x;

     return pot;
}
```

Funzioni – Soluzione 2 (2)

```
int serie (int a,int n) {
      int k, serie=0;
      for (k=1;k<=n;k++)
            serie= serie + pot(a,k);

      return serie;
}

void main() {
      int b, n, risultato;
      printf("\nInserire base: ");
      b=  leggi();
      printf(“Numero di elementi: ");
      n= leggi();

      risultato = serie (b,n);
      printf("\nRisultato: %d\n", risultato);
}
```

Funzioni – Esercizio 3

Codificare in C le funzioni:
`int min_to_sec(int a)` che considera il parametro a come minuti e restituisce il numero di secondi corrispondente;
`int ore_to_sec(int a)` che considera il parametro a come ammontare di ore, e restituisca il numero di secondi corrispondente. Si utilizzi la funzione definita precedentemente.

Definire un possibile main che prende in ingresso tre valori interi, rappresentanti ore, minuti e secondi della durata di un CD Audio. Il programma deve stampare il valore corrispondente in secondi. Utilizzare la struct tempo (ore, minuti, secondi), e la funzione di conversione:
`int tempo_to_sec(tempo t)`

Funzioni – Soluzione 3 (1)

```
#include <stdio.h>
typedef struct {
      int ore;
      int min;
      int sec;
} tempo ;

int min_to_sec(int m) {
      int ms ;
      ms=m*60;
      return ms;
}

int ore_to_sec(int h) {
      int hs, hm;
      hm=h*60;
      hs=min_to_sec(hm);
      return hs;
}
```

Funzioni – Soluzione 3 (2)

```
int tempo_to_sec(tempo t) {
     int s_min, s_ore, s_sec;
     s_sec = t.sec;
     s_min = min_to_sec(t.min);
     s_ore = ore_to_sec(t.ore);
     return s_ore+s_min+s_sec;
}
```

Funzioni – Soluzione 3 (3)

```
void main() {
     int sectot;
     tempo t;
     printf("Inserire ore/min/sec: ");
     scanf("%d:%d:%d",&t.ore,&t.min, &t.sec);
     sectot=tempo_to_sec(t);
     printf("\nSecondi totali: %d", sectot);
}
```

Funzioni – Esercizio 4

Si progetti un programma per la geometria piana che definisca una struct punto (float x,y).
Si crei una funzione “distanza”, che dati in ingresso due punti ne calcoli la distanza euclidea.
Si definisca poi una struct puntopolare (float r,t) per le coordinate polari.
Si creino poi le funzioni per la conversione da coordinate rettangolari a polari e viceversa.
Si utilizzi un main() con opportuni input ed output per dimostrare il funzionamento di tutte le funzioni.

Funzioni – Soluzione 4

```
#include <math.h>

typedef struct { float x,y;} punto;
typedef struct { float r,t;} puntopolare;

float distanza(punto a, punto b)
{      return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));}

struct punto pol2rec(puntopolare p)
{      punto temp;
       temp.x = p.r * cos(p.t);
       temp.y = p.r * sin(p.t);
       return temp;}

struct puntopolare rec2pol(punto p)
{      puntopolare temp;
       temp.r = sqrt(p.x*p.x + p.y*p.y);
       temp.t = atan2(p.y,p.x);
       return temp;}
```

Funzioni – Esercizio 5

Codificare in C la funzione
`int ipotenusa(int a, int b)` che, dati i cateti **`a`** e **`b`** di un triangolo rettangolo, restituisce il valore dell'ipotenusa.

A tal scopo si utilizzi il Teorema di Pitagora:

$$Ipotenusa = \sqrt{a^2 + b^2}$$

Per calcolare la radice quadrata si utilizzi la funzione di libreria **`sqrt(x)`.** Per utilizzare quest'ultima si aggiunga l'istruzione **`#include <math.h>`** in testa al file.

Definire un possibile main che legga da tastiera due valori che rappresentino i cateti di un triangolo rettangolo, e stampi il valore dell'ipotenusa.

Funzioni – Soluzione 5 (1)

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

double ipotenusa(int a, int b) {
     double ipotenusa;
     double radice;

     radice = a*a + b*b;

     ipotenusa = sqrt(radice);

     return ipotenusa;
}
```

Funzioni – Soluzione 5 (2)

```
void main(void) {
     int x,y;
     double ipo;

     printf("Inserisci un lato: ");
     scanf("\n%d",&x);
     printf("\nInserisci l'altro lato: ");
     scanf("\n%d",&y);

     ipo = ipotenusa(x,y);

     printf("\n\nIpotenusa: %.2lf\n", ipo);

}
```

Funzioni – Esercizio 6

Codificare in C la funzione
`int perimetro(int a, int b, int c)` che, dati i lati a,b,c di un triangolo, ne calcola il perimetro.

Codificare in C la funzione
`float area(int a, int b, int c)`
che restituisce l'area di un triangolo i cui lati misurano a, b, c. A tal scopo si usi la formula di Erone:

$$Area = \sqrt{p(p-a)(p-b)(p-c)}$$

Dove p è la metà del perimetro. A tal scopo si includa l'header **`<math.h>`** e si utilizzi la funzione **`sqrt(x)`**.

Definire un possibile main che prende in ingresso i tre lati di un triangolo e stampa perimetro ed area.

Funzioni – Soluzione 6 (1)

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

int perimetro(int a, int b, int c) {
     return a+b+c;
}

float area(int a, int b, int c) {
     double p = perimetro(a,b,c)/2.0;
     double radice;
     float result;
     radice = p*(p-a)*(p-b)*(p-c);
     result = (float)sqrt(radice);
     return result;
}
```

Funzioni – Soluzione 6 (2)

```
void main(void) {
     int x,y,z;
     int p;
     float a;

     printf("Inserisci il primo lato: ");
     scanf("\n%d",&x);
     printf("Inserisci il primo lato: ");
     scanf("\n%d",&y);
     printf("Inserisci il primo lato: ");
     scanf("\n%d",&z);
     p = perimetro(x,y,z);
     a = area(x,y,z);
     printf("\n\nPerimetro:\t%d", p);
     printf("\nArea:\t\t%.2f\n", a);
}
```

Funzioni – Esercizio 7

Codificare in C la funzione **`int primo(int x)`** che restituisce:

1 se x è un numero primo
0 altrimenti.

Si utilizzi a tal proposito l'operatore modulo (%).

Si progetti un programma che legge da tastiera un numero N, e stampa a video tutti i numeri primi compresi tra 0 e N.

Funzioni – Soluzione 7 (1)

Primo livello di specifica:

I numeri < 4 sono primi;

Un numero $N > 3$ è primo se non è divisibile per 2, né per uno dei numeri dispari compresi fra 3 e $\sqrt{N}$

Funzioni – Soluzione 7 (2)

Secondo livello di specifica:

Verifico se $N < 4$: in tal caso N è primo.

Verifico se N è pari: in tal caso N non è primo.

Considero un ciclo, in cui ho un indice i che assume i valori dispari fra 3 e $\sqrt{N}$. Ad ogni iterazione, controllo se N è divisibile per i. Se è divisibile, esco (N non è primo). Se non è divisibile, provo col prossimo numero. Se ho provato tutti i valori, vuol dire che N è primo

Funzioni – Soluzione 7 (3)

Terzo livello di specifica:

Verifico se $N < 4$: in tal caso N è primo.

Verifico se N è pari: in tal caso N non è primo.

Inizializzo un indice $i = 3$.

Eseguo un ciclo in cui la condizione di uscita è:

- o ho dimostrato che N non è primo
- oppure ho provato tutti i valori da 3 a $\sqrt{N}$

All'interno del ciclo incremento i.

All'uscita del ciclo, controllo quale delle 2 condizioni è vera.

Funzioni – Soluzione 7 (4)

Quarto livello di specifica:

```
#define FALSE 0
#define TRUE 1
int primo(int N)
{ int i;
  if (N<4) { return TRUE; }
  if ((N % 2) == 0) {return FALSE; }
  i = 3;
  while ((i<sqrt(N)) && (N % i) != 0)
     { i += 2; }
  return (N % i) != 0;
}
```

Funzioni – Esercizio 8

Si scriva un programma che prenda in ingresso i coefficienti a, b, c di un equazione di secondo grado e che restituisca i risultati dell'equazione.
A tal fine si scriva una funzione

```
int delta_positivo(float a,float b,float c);
```

che dati a, b, c controlli se esiste il delta dell'equazione,
e una funzione

```
float delta(float a, float b, float c);
```

che dati a, b, c restituisca il delta dell'equazione.

Funzioni – Soluzione 8 (1)

```
int delta_positivo(float a,float b,float c) {
     return ((b * b) >= (4 * a * c));
}


float delta(float a, float b, float c) {
     return ((b * b) - (4 * a * c));
}
```

Funzioni – Soluzione 8 (2)

```
main()
{float a, b, c;
 double x1, x2;
 printf("Inserisci i tre coefficienti: ");
 scanf("%f %f %f", &a, &b, &c);
 if (delta_positivo(a,b,c))
   {x1=(-b + sqrt(delta(a,b,c)))/(2*a);
    x2=(-b - sqrt(delta(a,b,c)))/(2*a);
    printf("I risultati sono x1=%g, x2=%g.",
            x1, x2);
   }
 else printf("Delta minore di 0");
}
```