Laboratorio 20/06/03

Preparazione verifica in itinere

Es. 1

```
#include <stdio.h>

void caricavalore(int a)
{
  printf("Immetti un numero ");
  scanf("%d",&a);
}

void main()
{
  int v=0;
  caricavalore(v);
  printf("Il valore inserito e' %d\n",v);
}
```

Verificare la correttezza sintattica e correggere gli eventuali errori logici.

Es. 1b

```
#include <stdio.h>
int f(int *h)
{ int *k, b;
  k = h;
  *k = 7;
  b = *k - *h;
  return(b);}
void main()
  {int a, *c, i;
   a = 8;    c = &a;
   for(i=1;i<10;i++)
     *c = *c + f(c);
  printf("%d\t%d\t%d\n",a,*c,i);  }
```

Dopo aver verificato la correttezza sintattica e corretto gli eventuali errori, indicare i valori stampati dal programma.

ES. 2

```
#include <stdio.h>

void maiuscole(char*str, char c)
{
  int i;
  for(i=0;i<strlen(str);i++)
    str[i]=toupper(str[i]);
  c=toupper(c);
}

void main()
{
  char * s="proVA";
  char c="x";
  maiuscole(s,c);
  printf("dopo: %s %c\n",s,c);
}
```

Il programma deve convertire in maiuscolo e stampare. Correggere eventuali errori sintattici e verificare. Che cosa stampa? Perche'?

ES. 2 (soluzione)

```
#include <stdio.h>

void maiuscole(char*str, char * c)
{
  int i;
  for(i=0;i<strlen(str);i++)
    str[i]=toupper(str[i]);
  *c=toupper(*c);
}

void main()
{
  char * s="proVA";
  char c='x';
  maiuscole(s,&c);
  printf("dopo: %s %c\n",s,c);
}
```

ES. 2b

```
#include <stdio.h>
#include <math.h>
typedef int VETT[8];
double media(VETT v)
{int i, somma;
   for(i=0, somma=0;i<8; i++)
       somma = somma + v[i];
   return(somma/8);}
int Pos(VETT v, double m)
{int i, p=0;
 for(i=0; i<8; i++)
   if(fabs(v[i]-m) > fabs(v[p]-m))
      p = i;
  return(p); }
void main()
{int i; VETT vt; double m;
 for(i=0;i<8;i++)
    scanf("%d",&vt[i]);
 printf("%f\t%d\n",m, Pos(vt,(m=media(vt))));
}
```

il programma deve stampare
l'elemento il cui valore è più vicino
al valor medio del vettore

ES. 2b (soluzione)

```
#include <stdio.h>
#include <math.h>
typedef int VETT[8];
double media(VETT v)
{int i, somma;
   for(i=0, somma=0;i<8; i++)
       somma = somma + v[i];
   return(somma/8.0);}
int Pos(VETT v, double m)
{int i, p=0;
 for(i=0; i<8; i++)
   if(fabs(v[i]-m) < fabs(v[p]-m))
      p = i;
  return(p); }
void main()
{int i; VETT vt; double m;
 for(i=0;i<8;i++)
    scanf("%d",&vt[i]);
 printf("%f\t%f\n",m,vt[Pos(vt,(m=media(vt)))]);
}
```

Es 3

- Scrivere un programma che **legge un file** contenente righe di testo e **ne scrive un altro** contenente le stesse righe di testo, pero' con l'ordine delle righe invertito.
- Si puo' assumere che una riga di testo non superi mai i 99 caratteri di lunghezza.
- Il nome del file da invertire viene passato sulla riga di comando come primo parametro.
- Il nome del file invertito viene passato sulla riga di comando come secondo parametro.
- Se il numero di parametri passati non e' corretto o se non e' possibile aprire entrambi i files, il programma deve arrestarsi.
- **Utilizzare un algoritmo ricorsivo (non iterativo)** per implementare la funzionalita' richiesta.
- **Suggerimento**: *per invertire un file di N righe, prima inverto le ultime N-1 righe (se ce ne sono), poi stampo la prima riga...*

INV(A B C D) = INV (B C D) ***A*** *= INV (C D)* ***B A*** *= INV (D)* ***C B A*** *=* ***D C B A***

Es 3 (parte 1 di 2)

```
void main(int argc, char ** argv)
{
  FILE * infile, * outfile;
  if(argc!=3)exit(1);
  infile=fopen(argv[1],"r");
  if(!infile)exit(1);
  outfile=fopen(argv[2],"w");
  if(!outfile)exit(1);
  inverti(infile,outfile);
  fclose(infile); fclose(outfile);
}
```

Es 3 (parte 2 di 2)

```
void inverti(FILE * r, FILE * w)
{
   char buffer[100];
   if(!fgets(buffer,99,r))return;
   inverti(r,w);
   fprintf(w,"%s",buffer);
}
```

Es 3b

- Siano dati due file di testo f e g denominat1 ‘n1.txt’ e ‘n2.txt’ contenenti due sequenze di numeri interi disposti uno per riga. Costruire un programma in C (Pascal) che scrive sul file di testo h denominato 'risultato.txt’, tutti gli elementi di f che non sono in g.

Esempio:

n1.txt	n2.txt	risultato.txt
5	3	5
15	7	6
3	15	
6	24	
	9	
	32	

Es 3b

```
#include <stdio.h>
#include <stdlib.h>
void main() {
  FILE *f, *g, *h;
  int a, b;
if((f=fopen("n1.txt","r"))==NULL)   exit(-1);
if((g=fopen("n2.txt","r"))==NULL)   exit(-2);
if((h=fopen("risultato.txt","w"))==NULL)exit(-3);
  while (fscanf(f,"%d",&a) != EOF)
      { cont = 0;
        while(fscanf(g,"%d",&b) != EOF)
           if (a != b)
                fprintf(h,"%d\n",a);
      }
      fclose(f); fclose(g); fclose(h);}
```

Es 3b (soluzione)

```
#include <stdio.h>
#include <stdlib.h>
void main() {
  FILE *f, *g, *h;
  int a, b, cont;
if((f=fopen("n1.txt","r"))==NULL)   exit(-1);
if((g=fopen("n2.txt","r"))==NULL)   exit(-2);
if((h=fopen("risultato.txt","w"))==NULL)exit(-3);
  while (fscanf(f,"%d",&a) != EOF)
      { cont = 0;
        while(fscanf(g,"%d",&b) != EOF)
                if (a == b) cont++;
        if (cont == 0)
           fprintf(h,"%d\n",a);
        rewind(g);}
      fclose(f); fclose(g); fclose(h);}
```

Es. 4

- Per approssimare ex si usa la seguente formula:
- $e^{x}_{(F)} = \sum_{n=0}^{F} x^{n}/(n!)$
- Si da' il valore della precisione cercata (prec) e quindi si calcola la sommatoria per valori crescenti di F finché

$$| e^{x}_{(F)} - e^{x}_{(F-1)} | < prec$$

Es. 4

```
#include <stdio.h>
#include <math.h>
double Esp(double x, double prec)
{ double att, pre; int i=1;
  att=1.0; pre=0.0;
  do{pre = att; att = att + pre*x/i; i++;
  } while(fabs(att-pre) > prec);
  return(att);}
void main()
{ double  x;
  scanf("%lf",&x);
  printf("%f\n",Esp(x,0.0001));    }
```

Es. 4 (soluzione)

```
#include <stdio.h>
#include <math.h>
double Esp(double x, double prec)
{ double coef, att, pre; int i=1;
  coef=1.0; att=1.0; pre=0.0;
  do{ coef = coef * x / i;
      pre = att; att = att + coef; i++;
  } while(fabs(att-pre) > prec);
  return(att);}
void main()
{ double  x;
  scanf("%lf",&x);
  printf("%f\n",Esp(x,0.0001));    }
```

Es. 5

Dato l'assegnamento C

```
x[2].a = x.b;
```

indicare i tipi, se esistono, che lo rendono possibile; altrimenti spiegare perché non esistono.

Es. 6

Dato l'assegnamento C

```
y.a[2] = y.a;
```

indicare i tipi, se esistono, che lo rendono possibile; altrimenti spiegare perché non esistono.