**Primo Blocco di Esercizi di Preparazione alla Prova Scritta**
**Fondamenti di Informatica L-A** (Proff. Paola Mello e Paolo Bellavista)

Questo blocco di esercizi include:

- esercizi di analisi;
- esercizi di sintesi;
- esercizi sulle grammatiche;
- esercizi sulla rappresentazione dei numeri;
- esercizi su record di attivazione;
- esercizi sulle funzioni ricorsive;
- domande varie su argomenti di teoria.

## ESERCIZI DI ANALISI

**ESERCIZIO**

Si indichino i valori stampati dal seguente programma C, motivando la risposta data. Indicare quali sono i blocchi in cui è visibile la variabile N (motivare la risposta).

```
#include <stdio.h>
#define L 7

int N=L;
void  P(char* VET[], int DIM);

int main () {

     char* M[L] = {"pippo","pluto","paperino","qui","quo","qua",
          "minnie"};
     int  i;

     P(M,L-2);
     for (i = L-1; i-->0; ) printf("%s\n",*(M+i));
     return 0;
}

void  P(char* VET[], int DIM) {

     int i;
     N++;
     for (i=0; i<DIM-1; i=i+2)  strcpy(VET[i],VET[i+1]);
     return;
}
```

**Soluzione**

Il programma stampa:
**qua**
**quo**
**qui**
**qui**
**pluto**
**pluto**

La variabile N è visibile in tutti i blocchi, perché è definita nell'ambiente globale.

**Esercizio**
Dato il seguente programma C:

```
#include <stdio.h>
#define Dim 5
char  c = 'a';
char A[Dim]={'c','i','a','o','\0'};
char vet[Dim]={'e','e','e','e','\0'};

void sub(char vet1[], char vet2[]);
void stampa(char vet[]);

int main()
{stampa(A);
 sub(A,vet);
 stampa(vet);
 stampa(A);
 printf("%c",c);
 return 0;
}

void sub(char vet1[Dim], char vet2[Dim])
{int i; char c='b';
 for(i=0; i<Dim-1; i++)
    if (vet2[i]>vet1[i])
       vet1[i]=vet2[i];
       else vet1[i]=c;
 }

void stampa(char vet[])
{printf("Vettore:\n");
 printf("%s",vet);
 printf("\n");
}
```

Che cosa viene stampato dal programma (si motivi opportunamente la risposta)? Si dica inoltre se la variabile **`i`** definita nella procedura **`sub()`** è visibile dalla procedura **`stampa()`** e dal main.

**Soluzione**
La prima stampa produce i valori ciao che corrispondono al vettore **A** inalterato. Dopo di che viene chiamata la procedura **sub** che modifica i valori di **vet** e di **A**. Se l'elemento di **vet** è maggiore alfabeticamente del corrispondente elemento di **A**, quest'ultimo elemento viene sovrascritto dall'elemento di **vet**, altrimenti viene sovrascritto con il carattere 'b'.

Quindi **vet** viene modificato in ebeb. Viene poi stampato il vettore **A**, producendo eeee. Poi viene stampato **vet** ebeb.
Infine viene stampato **c**, che è la variabile definita esternamente al main e non quella definita nella procedura **sub()**.

Quindi, il risultato stampato è:

```
Vettore:
ciao
Vettore:
eeee
Vettore:
ebeb
a
```

la variabile *i* definita nella procedura **sub()** non è visibile nella procedura **stampa()** e nel main.

**Esercizio**
Dato il seguente programma C:

```
#include <stdio.h>
#define Dim 5

int f(int *V, int k)
{int i, s=Dim; for(i=Dim-1;i>=0; i-=k) { V[i]=i;   s+=i;   }
 return s;
}

int main()
{ int A[Dim]={1,1,1,1,1};
     int i;
        for (i=0; i<Dim; i+=2)
          A[i]-=i;
     printf("%d\n", f(A,3));
     for(i=0; i<Dim; i++)
          printf("%d\t", A[i]);
     return 0;
}
```

Qual è l'uscita del programma? La risposta deve essere opportunamente motivata.

**Soluzione:**
Il ciclo for nel main modifica le componenti del vettore A di indice pari. Il vettore diventa:
{1,1,-1,1,-3}
Il main chiama poi la funzione f, passando il vettore A per indirizzo e il valore 3.
La funzione, a passi di 3 (k=3), assegna alle componenti del vettore V (che rappresenta l'indirizzo del primo elemento di A) di indice i=4 e i=1 il valore dell'indice stesso.
Il vettore A diventa:
{1,1,-1,1,4}
e somma alla variabile s (inizialmente uguale a 5) tali indici, restituendo il valore di s in uscita (s=5+4+1=10).
Nel main viene stampato il valore restituito dalla funzione:
10
e il vettore A:
1 1 -1 1 4

**Esercizio**
Dato il seguente programma C:

```
#include <stdio.h>
#define D 4
float V[D]={1.5, 2.5, 3.5, 4.5};
float A[D]={0,0,1,1};

float Fun(float V[], int k)
{int i;
 for(i=0;i<D; i+=k)
          V[i]=i;
 return i;
}

int main()
{int i;
 printf("%f\n", Fun(A,2));
 for(i=0; i<D; i++)
      printf("%f\t", V[i]);
 printf("\n");
 for(i=0; i<D; i++)
      printf("%f\t", A[i]);
 return 0;
}
```

Che cosa stampa il programma (si motivi opportunamente la risposta)?

**Soluzione**
Il programma main chiama la funzione Fun, passando il vettore A per indirizzo e il valore 2.
La funzione, a passi di 2 (k=2), assegna alle componenti del parametro formale V (al quale è stato assegnato l'indirizzo del vettore A) di indice i=0,2 il valore dell'indice stesso, e al termine restituisce il valore dell'indice i (=4).
Tale risultato viene stampato dalla printf nel programma main:
4.000000
Terminata la funzione, il main stampa il contenuto del vettore V:
1.500000 2.500000 3.500000 4.500000
e il vettore A:
0.000000 0.000000 2.000000 1.000000
Si noti che il vettore V dichiarato nella parte delle dichiarazioni globali non è stato modificato dalla chiamata della funzione Fun. Infatti a tale funzione viene trasferito per indirizzo il vettore A (modificato). La funzione Fun accede alle componenti di A attraverso il parametro formale V che non è il vettore dichiarato in precedenza, ma un nome (identico al precedente) con il quale si riferisce – all'interno della funzione Fun – il parametro attuale A passato per indirizzo.

**Esercizio**
Dato il seguente programma C:

```
#include<stdio.h>
#define Dim 4
char  single = 'n';
char Primo[Dim]={'a','b','c','\0'};
char Secondo[Dim]={'b','b','b','\0'};

int change(char string1[], char string2[]);
void print(char string[]);

int main()
{int N;
 print(Primo);
 N = change(Primo,Secondo);
 print(Secondo);
 print(Primo);
 printf("%c,%d",single,N);
 return 0;
}

int change(char string1[], char string2[])
{int j; int i=4; char single='f';
 for(j=0;j<Dim-1;j++)
    if (string2[j]>string1[j])
       {string1[j]=string2[j];
        i++;}
        else Primo[j]=single;
```

```
 return i;}

void print(char string[])
{printf("Vettore:\n");
 printf("%s",string);
 printf("\n");
}
```

Che cosa viene stampato dal programma? La risposta deve essere opportunamente motivata. Si dica inoltre se la variabile N definita nel main è visibile anche dalle funzioni/procedure **change** e **print**.

**Soluzione**
La prima stampa produce i valori abc che corrispondono al vettore **Primo** inalterato. Dopo di che viene chiamata la procedura **change** che modifica i valori di **Primo** e di **Secondo**. Se l'elemento di **Secondo** è maggiore alfabeticamente del corrispondente elemento di **Primo**, quest'ultimo elemento viene sovrascritto dall'elemento di **Secondo**, altrimenti il j-esimo elemento di **Primo** viene sovrascritto con il carattere 'f' (la definizione di **single**, interna alla procedura **change**, nasconde la definizione data come variabile globale). Il valore restituito da **change** è il valore iniziale di ***i*** più il numero di volte in cui si è verificato il caso **Secondo[j]** > **Primo[j]**, cioè 4+1.

Quindi **Primo** diventa bff. Viene poi stampato il vettore **Secondo**, producendo bbb. Poi viene stampato **Primo**: bff.
Infine viene stampato **single**, che è la variabile definita esternamente al main e non quella definita nella procedura **change** e il valore restituito da **change**.

Quindi, il risultato stampato è:

```
Vettore:
abc
Vettore:
bbb
Vettore:
bff
n,5
```

la variabile ***N*** definita nel main non è visibile nelle procedure/funzioni.

---

**Esercizio**
Dato il seguente programma C:

```
#include <stdio.h>
#define Dim 5

int calc(int *N, int p)
{int i, k=Dim;
 for(i=0; i<Dim; i=i+p)
```

```
        {       N[i]=p-N[i];
                k=k-i;
        }
p = p + 1;
return k;
}

int main()
{ int Quad[Dim]={1,4,9,16,25};
  int i, j = 2;
  for (i=0; i<Dim; i++)
      Quad[i] = i+j - Quad[i];
  printf("%d\n", calc(Quad,j));
  for(i=0; i<Dim; i++)
        printf("%d\t", Quad[i]);
  printf("\n%d\n", j);
  return 0;
}
```

dire che cosa viene stampato dal programma, con le opportune motivazioni.
Si dica poi se la variabile `k` è visibile dal `main()`.

**Soluzione**

Dopo il primo ciclo for del main, il vettore Quad contiene i seguenti valori

```
{1, -1, –5, –11, -19}
```

Dopo di che viene chiamata la funzione calc che riceve come parametro attuale (per indirizzo) il vettore Quad e la variabile intera j = 2. Nel ciclo for della funzione calc il vettore viene modificato nel seguente modo:

- Al primo passo l'elemento di indice 0 viene sottratto al valore 2 e la variabile k rimane uguale a 5
- Al secondo passo, l'elemento di indice 2 viene sottratto a 2 e la variabile k diventa uguale a 3
- Al terzo passo, l'elemento di indice 4 viene sottratto a 2 e la variabile k diventa uguale a -1.

k viene restituita dalla funzione calc e quindi la prima stampa del programma main fornisce in uscita **-1**. Viene poi stampato il vettore Quad modificato (perché passato per indirizzo) ed infine j che non viene modificata dalla funzione ric.

Risultato stampato dal programma

```
-1
1    -1   7    -11  21
2
```

**Esercizio**
Dato il seguente programma:

```
#include <stdio.h>
#define DIM 6

int p(int a)
{ if (a%2==0)
      return 0;
  else return a+1;
}

int f(int *a, int b)
{
      if (a[b]!=0)
            return a[b]=5;
      else return p(b+1)+b;
}



int main()
{     int A[DIM]={0,0,0,0,0,0};
      int i;
      for(i=0; i<DIM; i+=2)
            A[i]=i;
      printf("%d\n", f(A,0));
      for(i=0; i<DIM; i++)
            printf("%d\t",A[i]);
      return 0;
}
```

Si indichino, nel giusto ordine, i valori stampati dal programma.

**Soluzione:**

```
2
0          0          2          0          4          0
```

## ESERCIZI DI SINTESI

**Esercizio**
Dato un numero, stabilire se esso è un numero "*perfetto*" o no.
In particolare, realizzare una funzione perfetto(int x) che restituisca true se e solo se il numero x è perfetto. Si ricordi che un numero è "*perfetto*" se risulta essere la somma di tutti i suoi divisori (escluso sé stesso).
Progettare poi un programma main che, utilizzando tale funzione, chieda ad un utente un numero, e calcoli tutti i numeri perfetti compresi tra 1 e tale numero.

**Soluzione**

```
int perfetto(int x)
{
     int somma = 0;
     int i;
     for(i=1; i<x; i++)
     {
          if ( (x%i)==0 )
               somma = somma + i;
     }
     return somma==x;
}


int main()
{
     int num;
     int i;
     printf(”Inserire limite: ”);
     scanf("%d", &num);
     for(i=1; i<= num; i++)
          if (perfetto(i))
            printf("%d e’ un numero perfetto\n", i);
     return 0;
}
```

**Esercizio**
Realizzare un programma che calcoli il numero PI-Greco con precisione arbitraria.
In particolare, realizzare una funzione pigreco(int x) che riceva in ingresso un intero x (che rappresenterà la precisione), e restituisca in uscita il valore di PI-Greco, calcolato secondo la seguente formula:

$$\pi = \sum_{i=0}^{x} (-1)^i \frac{4}{2i+1}$$

Si realizzi poi un programma main() che, almeno una volta, chieda all'utente di immettere un valore x. Se tale valore è uguale a -1, il programma termini immediatamente; se il valore è maggiore o uguale a 0, allora il programma calcoli pi-greco con tale precisione e chieda un nuovo numero all'utente. Qualora il valore non sia ammissibile (nel nostro caso, minore di -1), il programma ne richieda uno corretto all'utente.

**Soluzione**

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

double pigreco(int x) {
     double somma = 0; int i;
     for (i=0; i<=x; i++)
          somma = somma + pow(-1, i)*(4.0/(2*i +1));

     return somma;
}

int main()
{
     int num;
     double pi;

     do {
          printf("Inserire numero: ");
          scanf("%d", &num);

          if (num >=0) {
               pi = pigreco(num);
               printf("Pi Greco vale: %f\n\n", pi);
          }
     } while (num != -1);
     return 0;
}
```

**Esercizio**

a) Si definisca una funzione iterativa long fact (long n) che calcoli in maniera iterativa il fattoriale del parametro n. Si definisca inoltre una funzione ricorsiva float power (float base, long exp) che calcoli in maniera ricorsiva la potenza del parametro base elevato ad exp.

b) Si definisca una funzione **float newton(float a, float b, long n)** che calcoli il seguente valore:

$$\sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^{k}$$

dove il simbolo del binomio è così definito:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

c) Si scriva infine un programma main che chieda all'utente di inserire tre valori, **a** e **b** (float), e **n** (int). Il programma controlli, in particolare, che **n** sia maggiore o uguale a zero (in caso contrario si continui a chiedere all'utente un nuovo valore finché questi non soddisfa la condizione posta). Il programma stampi infine il valore ottenuto invocando la funzione **newton()** con parametri **a**, **b**, **n** e poi termini.

**Soluzione**

```
long fact(long n) {
     long result = 1;
     for ( ; n>0; n--)
          result = result * n;
     return result;
}


float power(float base, long esp) {
     if (esp == 0)
          return 1;
     else
          return base * power(base, esp-1);
}


long binomio(long n, long k) {
     return fact(n)/(fact(k) * fact(n-k));
}


float newton(float a, float b, long n) {
     long i;
     float result = 0;
```

```
     for (i=0; i<=n; i++)
          result = result + binomio(n, i) * power(a, n-i) * power(b,
i);
     return result;
}


int main() {
     float a;
     float b;
     long n;

     printf("Inserire a, b e n: ");
     scanf("%f %f %d", &a, &b, &n);
     while (n<0) {
          printf("Re-inserire n: ");
          scanf("%d", &n);
     }
     printf("(%f + %f)^%d = %f\n", a, b, n, newton(a, b, n));

     return 0;
}
```

**Esercizio**

Si definisca una struttura mese, caratterizzata dai campi nome_mese (stringa, non più di 16 caratteri compreso il terminatore), e da lunghezza_mese (intero, rappresentante i giorni di tale mese).

Si scriva poi una funzione seleziona che, ricevuto come parametro di ingresso un array di strutture mese ed un array rappresentatne la lunghezza di tale array, stampi a video i nomi dei mesi che hanno 31 giorni.

Si definisca poi un programma main che chieda all'utente di inserire il nome di un mese ed il numero di giorni che lo compongono (chieda tali informazioni al massimo 12 volte), memorizzi le risposte in un array preparato precedentemente, e stampi a video i mesi di 31 giorni tramite la funzione seleziona.

**Soluzione**

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

struct mese {
    char nome_mese[16];
    int lunghezza_mese;
```

```
};


void seleziona(struct mese meseArray[], int lung) {
    int i;

    for (i=0; i< lung; i++)
      if (meseArray[i].lunghezza_mese == 31)
            printf("%s ha 31 giorni.\n", meseArray[i].nome_mese);
}


int main() {
    struct mese mesiLetti[12];
    int i;

    for (i=0; i<12; i++) {
      printf("Inserire primo mese: ");
      scanf("%s", mesiLetti[i].nome_mese);
      printf("Inserire giorni: ");
      scanf("%d", &(mesiLetti[i].lunghezza_mese) );
    }

    seleziona(mesiLetti, 12);
    return 0;
}
```

**Esercizio**

Si scriva una procedura che legga da input una serie di numeri positivi (massimo 10) e termini con un numero negativo o nullo. Tale procedura restituisce in uscita (come parametri passati per riferimento) un vettore **V** (di dimensione fisica 10 dichiarato nel programma chiamante) contenente i numeri letti e il numero effettivo di elementi letti **N** (ossia la dimensione logica del vettore).

```
void leggi(int V[], int *N);
```

Si mostri anche un esempio di chiamata della procedura.

**Soluzione**

```
#include <stdio.h>

void leggi(int V[], int *N)
{
    int i = 0;
    int Num;
    printf("inserisci un intero positivo (0 o neg per terminare): ");
    scanf("%d", &Num);
    while (Num > 0)
    { V[i] = Num;
```

```
      i++;
      printf(“inserisci un intero positivo (0 o neg per terminare): ”);
      scanf(“%d”, &Num);
    }
 *N = i;
}
```

chiamata:

```
int main(){
    int VETTORE[10];
    int DimLogica;

    leggi(VETTORE, &DimLogica);
    printf("Numero effettivo di elementi nel vettore: %d",DimLogica);
    return 0;
}
```

**Esercizio "La banca"**

Una banca vuole realizzare un programma di simulazione di gestione conto-corrente.

A tal scopo il programma simula l'esistenza di un conto corrente, inizialmente vuoto, e offre un menù all'utente in cui egli può scegliere di compiere 4 operazioni diverse: **deposito, prelievo, visualizzazione del saldo, calcolo degli interessi composti**. L'utente sceglie l'operazione specificando un numero (o un carattere); sempre tramite un numero o un carattere segnala l'intenzione di terminare la simulazione. Dopo ogni operazione il programma ristampa a video l'elenco delle operazioni possibili ed attende un nuovo comando dall'utente.

- L'operazione di deposito consiste nel chiedere all'utente una somma, controllare che sia > 0, e depositarla nel conto.
- L'operazione di prelievo invece consiste sempre nel chiedere la somma all'utente, ma è ovviamente necessario controllare che sia > 0 e che ci siano fondi a sufficienza nel conto corrente.
- L'operazione di visualizzazione saldo consiste semplicemente nel visualizzare il saldo del conto corrente. Per comodità questa operazione deve essere eseguita automaticamente al termine delle operazioni di deposito e prelievo.
- L'operazione di calcolo degli interessi consiste invece nel calcolare a quanto ammonterebbe il capitale finale qualora si avesse il capitale iniziale pari ai soldi depositati sul conto, per **N** anni con un tasso di interesse pari a **r**. Il programma deve quindi chiedere all'utente il tasso di interesse ed il numero di anni su cui simulare. Ovviamente è necessario controllare che i dati inseriti corrispondano a valori plausibili (tasso d'interesse >= 0 e <= 100) (anni > 0).

La formula per il calcolo degli interessi composti, dove r è il tasso di rendimento e N è il numero di anni a cui gli interessi sono applicati, è la seguente:

$$Cfin = Cin * \left(1 + \frac{r}{100}\right)^{N}$$

**Soluzione**

```
#include <stdio.h>

#define DEPOSITO 1
#define PRELIEVO 2
#define SALDO 3
#define INTERESSI 4
#define FINE 0

void stampa_menu();
void f_deposito(float * soldi);
void f_prelievo(float * soldi);
void f_saldo(float soldi);
void f_interessi(float soldi);
float interessi(float c_in, float tasso, int anni);
```

```
int main() {
   int op;
   float soldi = 0;

   printf("Programma di simulazione conto corrente\n\n");
   stampa_menu();

   printf("Scegliere operazione: ");
   scanf("%d", &op);
   while (op != FINE) {
      switch (op) {
         case DEPOSITO:
            f_deposito(&soldi); break;
         case PRELIEVO:
            f_prelievo(&soldi); break;
         case SALDO:
            f_saldo(soldi); break;
         case INTERESSI:
            f_interessi(soldi); break;
         default:
            printf("Operazione non consentita");
      }
      stampa_menu();
      printf("Scegliere operazione: ");
         scanf("%d", &op);
   }
   return 0;
}

void stampa_menu() {
      printf("Operazioni disponibili:\n");
      printf("%d.\tDeposito\n", DEPOSITO);
      printf("%d.\tPrelievo\n", PRELIEVO);
      printf("%d.\tSaldo\n", SALDO);
      printf("%d.\tCalcolo interessi\n", INTERESSI);
      printf("%d.\tFine\n", FINE);
}

void f_deposito(float * soldi) {
      float cash;

      printf("Digitare l'ammontare del deposito: ");
      scanf("%f", &cash);
      while (cash < 0) {
            printf("Ammontare negativo.\n");
            printf("Digitare l'ammontare del deposito: ");
            scanf("%f", &cash);
      }
      *soldi = *soldi + cash;
      f_saldo(*soldi);
}

void f_prelievo(float * soldi) {
      float cash;

      printf("Digitare l'ammontare del prelievo: ");
      scanf("%f", &cash);
      while (cash < 0) {
```

```
            printf("Ammontare negativo.\n");
            printf("Digitare l'ammontare del prelievo: ");
            scanf("%f", &cash);
      }
      while (cash > *soldi) {
            printf("Nel conto non ci sono cosi' tanti soldi.\n");
            printf("Digitare l'ammontare del prelievo: ");
            scanf("%f", &cash);
      }
      *soldi = *soldi - cash;
      f_saldo(*soldi);
}

void f_saldo(float soldi) {
      printf("Disponibilita' nel conto corrente: %6.2f\n", soldi);
}

void f_interessi(float soldi) {
      float tasso, c_fin;
      int anni;

      printf("Inserire il tasso d'interesse: ");
      scanf("%f", &tasso);
      while ((tasso<0) || (tasso>100)) {
            printf("Tasso errato\n");
            printf("Inserire il tasso d'interesse: ");
            scanf("%f", &tasso);
      }
      printf("Inserire anni: ");
      scanf("%d", &anni);
      while (anni<0) {
            printf("Anni negativi.\n");
            printf("Inserire anni: ");
            scanf("%d", &anni);
      }

      c_fin = interessi(soldi, tasso, anni);
      printf("Interessi simulati, con:\n");
      printf("Capitale iniziale: %6.2f\n", soldi);
      printf("Tasso d'interesse: %6.2f\n", tasso);
      printf("Anni di applicazione degli interessi: %d\n", anni);
      printf("Capitale finale: %6.2f\n", c_fin);
}

float interessi(float c_in, float tasso, int anni)
{
      float tasso_perc, tasso_tot, result=1;
      int i;

      tasso_perc = tasso/100;
      tasso_tot = 1 + tasso_perc;

      for (i=0; i<anni; i++)
            result=result*tasso_tot;

      return c_in*result;
}
```

**Esercizio**

Si scriva un programma C che tramite tre funzioni, *leggi*, *media*, *stampa*:

**a)** Legga da terminale una prima sequenza di numeri terminati dal valore 0 (un numero su ogni linea) e li inserisca in un vettore A;

**b)** Legga da terminale una seconda sequenza di numeri terminati dal valore 0 e li inserisca in un altro vettore B;

**c)** Sia in grado di calcolare la media degli elementi di un vettore con la funzione *media*;

**d)** Stampi a video il vettore (A oppure B) la cui media è maggiore.

NOTA: Si ipotizzi una dimensione massima di 10 per i vettori

**Esempio:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Vettore A: | 3 | 5 | 7 | 8 | 2 | | Media=25/5=5 |
| Vettore B: | 2 | 6 | 10 | 2 | 3 | 15 | Media=38/6=6.333 |
| Vettore Max: | 2 | 6 | 10 | 2 | 3 | 15 | |

**Soluzione**

```
#include<stdio.h>
#define MAX 10

int leggi(int vet[],char nome);
float media(int vet[],int lung);
void stampa(int vet[],int lung);

int main()
{int A[MAX],B[MAX],a,b;
a=leggi(A,'A');
b=leggi(B,'B');
if(media(A,a)<media(B,b))stampa(B,b);
   else if(media(A,a)==media(B,b))printf("I vettori hanno la stessa
media!\n");
      else stampa(A,a);
return 0;
}



int leggi(int vet[],char nome)
{int i=0;
printf("\nScrivi gli elementi del vettore %c\n",nome);
do
   {printf("Elemento %d: ",i+1);
   scanf("%d",&vet[i]);
   i++;
   }
while(vet[i-1]!=0&&i<MAX);
return i-1;
```

```
}

float media(int vet[],int lung)
{int i;
float sum=0;
for(i=0;i<lung;i++)sum+=vet[i];
return sum/lung;
}


void stampa(int vet[],int lung)
{int i;
printf("Il vettore con media maggiore è:\n");
for(i=0;i<lung;i++)printf("%d ",vet[i]);
}
```

**Esercizio**

Si scriva un programma C che legga due serie di dati e le memorizzi in due vettori di strutture.

Nel primo vettore FILM (di dimensione 3) vengono memorizzate strutture (**`struct film`**) del tipo:
- titolo (stringa di lunghezza 20 non contenente spazi bianchi)
- codice film (intero)
- costo (intero)
- numero giorni di riprese

Nel secondo vettore ATTORI (di dimensione 5) vengono memorizzate strutture (**`struct attore`**) del tipo:
- nome (stringa di lunghezza 20)
- codice film (intero)
- costo al giorno (intero)

Si scriva un programma che:
- tramite due procedure **`leggifilm`** e **`leggiattore`** legga a terminale i dati da inserire nei due vettori;

```
void leggifilm(int n, struct film F[]);
void leggiattore(int n, struct attore A[]);
```

- tramite la procedura **`aggiornacosto`** aggiorni il costo del film sommando a questo il costo di ciascun attore che partecipa al film. Il costo dell'attore viene calcolato come costo al giorno per il numero di giorni delle riprese.

```
void aggiornacosto(struct film F[], int n, struct attore A[], int
m);
```

dove **`n`** è la dimensione del vettore **`F[]`** e **`m`** è la dimensione del vettore **`A[]`**

**Soluzione**

```
#include <stdio.h>
#define DIMA 4
#define DIMF 2

struct film{
       char titolo[20];
       int codice;
       int costo;
       int giorni;};

struct attore{
       char nome[20];
       int codicefilm;
       int costogiorno;};

void leggifilm(int n, struct film F[]);
void leggiattore(int n, struct attore A[]);
void aggiornacosto(struct film F[], int n, struct attore A[], int
m);

int main()
{int i;
 struct film FILM[DIMF];
 struct attore ATTORI[DIMA];
 leggifilm(DIMF,FILM);
 leggiattore(DIMA,ATTORI);
 aggiornacosto(FILM,DIMF,ATTORI,DIMA);
 return 0;
 }

void leggifilm(int n, struct  film Vet[])
{int i;
 for(i=0;i<n;i++){
     printf("Inserisci Codice, Titolo, Costo e Numero giorni \n");
     scanf("%d",&Vet[i].codice);
     scanf("%s",Vet[i].titolo);
     scanf("%d",&Vet[i].costo);
     scanf("%d",&Vet[i].giorni);
 }
}

void leggiattore(int n, struct attore Vet[]) {
  int i;
  for(i=0;i<n;i++){
    printf("Inserisci nome, codicefilm, costo al giorno\n");
    scanf("%s",Vet[i].nome);
    scanf("%d",&Vet[i].codicefilm);
```

```
    scanf("%d",&Vet[i].costogiorno);
 }
}


void aggiornacosto(struct film F[], int n, struct attore A[], int
m){
  int i,j;

  for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      if (A[j].codicefilm == F[i].codice){
           F[i].costo = F[i].costo + A[j].costogiorno*F[i].giorni;
  printf("\nCosto aggiornato del film %s: %d", F[i].titolo,
F[i].costo);
      }
}
```

**Esercizio**

Una compagnia di autobus che effettua servizio su lunghe distanze vuole realizzare un programma di controllo delle prenotazioni dei posti. A tal scopo rappresenta ogni prenotazione tramite una struttura **booking** contenente nome del cliente (al massimo 1023 caratteri, senza spazi) e numero del posto prenotato (un intero). Le prenotazioni effettuate vengono registrate tramite un array (di dimensione prefissata **DIM**) di strutture **booking**, di dimensione logica iniziale pari a 0.

a) Si realizzi una funzione:

```
int assegna(   booking list[],
               int * lengthList,
               char * name,
               int pref)
```

La funzione riceve in ingresso l'array di prenotazioni e la sua dimensione logica, e poi il nome del cliente ed il posto da lui indicato. La funzione deve controllare che il posto indicato non sia già stato assegnato, ed in caso contrario deve restituire il valore 1. Qualora invece il posto sia ancora libero, la funzione deve assegnare tale posto al cliente copiando i dati della prenotazione nell'ultima posizione libera nell'array, e deve provvedere ad aggiornare correttamente la dimensione logica dell'array. In questo secondo caso la funzione deve invece restituire come valore uno 0, indicante il successo nella prenotazione. Al fine di copiare il nome del cliente, si utilizzi la funzine di libreria

```
char * strcpy(char * s, char * ct)
```

che copia ct in s (terminatore compreso).

b) Si realizzi un programma main che chieda all'operatore il nome di un utente, e di seguito il posto prescelto. Il programma deve cercare di registrare la prenotazione tramite la funzione **assegna**; qualora l'operazione di prenotazione fallisca (perché il posto risulta essere già

assegnato), il programma provveda a chiedere all'operatore un nuovo posto, finché non si riesca ad effettuare la prenotazione. Qualora l'operatore inserisca il nome "fine", il programma deve terminare; qualora invece venga inserita la stringa "stampa", il programma deve stampare a video le prenotazioni già effettuate. A tal scopo si usi la funzione di libreria:

```
int strcmp(char * ct, char * cs)
```

che restituisce 0 se e solo se le due stringhe sono identiche (lessicograficamente).

**Soluzione**

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

#define MAX 1024
#define DIM 10


typedef struct {
     char name[MAX];
     int seat;
} booking;

int assegna(   booking list[], int * lengthList,
               char * name, int pref) {
     int i=0;
     int trovato = 0;

     while (( i < *lengthList) && !trovato) {
          if (list[i].seat == pref)
               trovato = 1;
          i++;
     }
     if (!trovato) {
          list[*lengthList].seat = pref;
          strcpy(list[*lengthList].name, name);
          (*lengthList)++;
          return 0;
     }
     else
          return 1;
}

int stampaBooking(booking list[], int lengthList) {
     int i=0;

     for (i=0; i<lengthList; i++)
          printf("%s: %d\n", list[i].name, list[i].seat);
```

```
     return 0;
}

int main() {
     booking list[DIM];
     int alengthList = 0;
     char nome[MAX];
     int seat = 0;

     printf("Inserire Nome Passeggero: ");
     scanf("%s", nome);

while (strcmp(nome, "fine")) {
          if (strcmp(nome, "stampa")) {
             printf("Posto preferito: ");
             scanf("%d%*c", &seat);

             while(assegna(list, &alengthList, nome, seat)) {
                printf("Spiacenti ma il posto scelto e' gia'
                       occupato.\n");
                printf("Per favore specificare un altro posto: ");
                scanf("%d%*c", &seat);
             }
          }
          else
               stampaBooking(list, alengthList);

          printf("Inserire Nome Passeggero: ");
          scanf("%s", nome);
     }

     return 0;
}
```

**Esercizio sulle matrici 1**

Una compagnia aerea tiene traccia, ogni giorno, delle prenotazioni dei posti sui suoi voli. In particolare, in una matrice di interi, di dimensione fissata MAX_FLIGHTS x MAX_SEATS, tiene traccia di quante prenotazioni sono state fatte sull'i-esimo aereo (0<=i<=MAX_FLIGHTS), per il posto j-esimo (0<=j<=MAX_SEATS).

Capita spesso che uno stesso posto venga richiesto da più clienti: in tal caso la compagnia all'atto della prenotazione concede lo stesso posto a tutti i richiedenti, salvo poi comunicare all'ultimo uno spostamento obbligatorio di posto. Quindi può succedere che:

- Se il valore in posizione [i][ j] è 0, significa che nessuno ha prenotato il posto j-esimo per il volo i-esimo
- Se il valore in posizione [i][ j] è 1, significa che una persona ha prenotato il posto j-esimo per il volo i-esimo
- Se il valore in posizione [i][ j] è 3, significa che tre persone ha prenotato lo stesso posto j-esimo per il volo i-esimo

Si realizzi un programma che:

1. chieda all'utente il codice di un volo (si supponga per semplicità che tale codice coincida con l'indice della matrice), e calcoli se tale volo è effettivamente in overbooking (cioè se la somma di tutte le prenotazioni per quel volo è maggiore della costante MAX_SEATS), o se invece vi sono posti disponibili per tutti coloro che hanno prenotato
2. calcoli (e stampi a video) qual è il posto più richiesto, rispetto a tutti i voli della giornata

Al fine di realizzare i punti 1 e 2, il candidato implementi delle opportune funzioni, e poi mostri un programma main di esempio che ne fa uso.

**Soluzione**

```
#include <stdio.h>


#define TRUE    1
#define FALSE   0

#define MAX_SEATS 5
#define MAX_FLIGHTS 3

typedef int BOOL;



BOOL canFly(int flightIndex, int bookingMatrix[][MAX_SEATS]) {
      int totalBooking = 0;
      int j;

      for (j=0; j<MAX_SEATS; j++)
            totalBooking = bookingMatrix[flightIndex][j] + totalBooking;
      if (totalBooking <= MAX_SEATS)
            return TRUE;
      else
            return FALSE;
}
```

```
int mostWantedSeat(int bookingMatrix[MAX_FLIGHTS][MAX_SEATS]) {
      int mostWanted = 0; // il posto piu' prenotato
      int bookingRecord = 0; // il record di prenotazioni per quel posto
      int howManyBooking = 0; // variabile accumulatore in cui tengo traccia
                              // delle prenotazioni per un certo posto
      int i, j;

      for (j=0; j<MAX_SEATS; j++) { // per ogni posto sugli aerei,
                                    //ne calcolo le prenotazioni
            for (i=0; i<MAX_FLIGHTS; i++) { // per ogni aereo,
                                            //conto quante prentoazioni
                                            // sono state fatte per quel posto
                  howManyBooking = howManyBooking + bookingMatrix[i][j];
            }
            if (howManyBooking > bookingRecord) { // se batto il record
                                                  // precedente...
                  mostWanted = j;                 // aggiorno il nuovo record!
                  bookingRecord = howManyBooking;
            }
            howManyBooking = 0;
      }
      return mostWanted;
}



int main(void)
{
      int flight;
      int bookingMatrix[MAX_FLIGHTS][MAX_SEATS] = {
            {2,3,1,0,0},
            {1,0,0,1,1},
            {4,0,0,0,0}
      };

      printf("Insert index of the desired plane: ");
      scanf ("%d", &flight);

      if (canFly(flight, bookingMatrix)) {
            printf("\nThe plane is overbooked, but enough space is available");
      }
      else {
            printf("\nThe plane is too much overbooked!!!\n");
      }

      printf("The most booked place is the number %d\n",
                              mostWantedSeat(bookingMatrix));
      return (0);
}
```

**Esercizio sulle matrici 2**

Un centro di meteorologia tiene traccia di tutte le temperature registrate nell'anno in un vettore bidimensionale (matrice) di interi. Tale matrice è costituita di tante righe quanti sono i mesi dell'anno ed ha per ogni riga spazio sufficiente a contenere il massimo numero di valori possibile, corrispondente alla durata massima di un mese in giorni.

Dopo aver definito ed inizializzato correttamente tale matrice, si realizzi un programma che:

a) effettui il calcolo della temperatura media di un mese specificato dall'utente;
b) effettui il calcolo della temperatura media di uno giorno (specificato dall'utente) di tutti i mesi (ad esempio la media delle temperature rilevate il giorno 5 di tutti i mesi);
c) effettui il calcolo della temperatura media dei giorni che risiedono all'interno di un range specificato dall'utente (ad esempio dal 15 Febbraio al 3 Aprile).

```
#include <stdio.h>

int main(){

	// Definizione delle variabili
	int i,j,giorno,mese,sommatoria,somGiorni,giornoIn,meseIn,giornoFin,meseFin;
	float media;

	// Definizione ed inizializzazione della variabile
// contenente le temperature di tutti i giorni dell'anno:
	// matrice bidimensionale di interi con 12 righe (i mesi) e 31 colonne (i
giorni).
	// La matrice nel suo complesso è composta da 372 (=12*31) interi,
	// in quanto tutti i mesi vengono considerati di 31 giorni.
	// Alcuni giorni di alcuni mesi però non esistono (ad esempio il 30
Febbraio);
	// per specificare che un certo valore non ha senso
// (poiché corrisponde ad un giorno che non esiste)
	// viene inserito un valore di controllo (-100) che è al di fuori del range
	// dei valori validi (ad esempio [-30, +50])
	int temp[12][31]={
		// Gennaio (31 giorni)
		{5,7,8,6,2,3,4,3,2,2,4,5,3,5,7,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,3},
		// Febbraio (28 giorni)
		{5,7,8,6,2,3,4,3,2,2,4,5,3,5,7,1,3,4,7,8,1,2,3,4,2,6,8,3,-100,-100,-
100},
		{5,7,8,6,2,3,4,13,2,2,14,5,3,15,7,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,3},
		{5,7,8,6,2,3,14,3,2,2,4,15,3,5,17,1,13,4,17,8,1,12,3,4,2,6,8,3,1,2,-
100},
		{5,7,8,16,21,3,14,3,2,2,4,5,3,15,7,1,3,4,7,8,1,2,13,14,2,6,8,3,1,2,3},
		{15,27,8,6,2,3,4,3,2,2,4,5,3,5,7,11,3,4,7,8,1,2,3,4,2,6,8,3,1,2,-100},
		{15,27,8,6,2,3,4,3,2,2,4,5,3,5,7,1,3,4,7,8,11,2,3,4,12,6,8,3,1,2,3},

	{5,7,8,6,22,23,24,23,22,22,24,25,23,25,72,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,3},
		{25,27,8,6,2,3,4,3,2,2,4,5,3,5,7,11,3,4,7,8,1,2,3,4,2,6,8,3,1,2,-100},
		{5,7,8,6,2,3,4,3,2,2,4,5,3,5,7,1,3,14,7,8,1,2,3,4,2,6,8,3,1,2,3},
		{5,7,8,6,2,3,4,3,2,2,4,5,3,15,7,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,-100},
		{5,7,8,6,2,3,4,3,12,2,4,5,3,5,7,1,3,4,7,8,1,2,3,4,2,6,8,3,1,2,3}
	};

	// In alternativa alla precedente definizione ed inizializzazione
```

```
      // era possibile richiedere all'utente le temperature.
      /*printf("Inserire -100 come temperatura dei giorni che non esistono");
      for(i=0;i<12;i++){
            for(j=0;j<31;j++){
                  printf("Inserire la temperatura del giorno %d del mese
%d\n",(i+1),(j+1));
                  scanf("%d",temp[i][j]);
            }
      }*/

      // Calcolo della temperatura media di un particolare mese
      printf("\nInserire il mese di cui si vuole conoscere la temperatura
media\n");
      printf("(1=Gennaio, 2=Febbraio, ... , 12=Dicembre)\n");
      // Occhio agli indici!
      // temp[0][0] è la temperatura del primo giorno di Gennaio
      // temp[0][30] è la temperatura del trentunesimo giorno di Gennaio
      // temp[1][0] è la temperatura del primo giorno di Febbraio
      // temp[11][1] è la temperatura del secondo giorno di Dicembre
      // ...
      scanf("%d", &mese);
      sommatoria=0;
      // Il seguente ciclo for somma il valore delle temperatura di tutti i giorni
del
// mese dato; il ciclo for itera finché non si verifica una delle due seguenti
// condizioni di fine mese:
      // a) la temperatura è -100
      // b) è stato raggiunto il numero massimo di giorni (31)
      for(j=0; temp[mese-1][j]!=-100 && j<31; j++){
            // è costante il primo indice (il mese),
// si itera sul secondo indice (il giorno)
            sommatoria+=temp[mese-1][j];
      }
      media=sommatoria/j;
      printf("numero totale di giorni %d\n",j);
      printf("sommatoria %d\n",sommatoria);
      printf("temperatura media del mese %d: %f\n\n",mese,media);

      printf("Inserire il giorno di cui si vuole conoscere la temperatura
media\n");
      // il giorno inserito deve essere valido
// per tutti i mesi dell'anno (no 29, 30, 31)
      scanf("%d", &giorno);
      sommatoria=0;
      for(i=0;i<12;i++){
            // si itera sul primo indice (il mese),
// è costante il secondo indice (il giorno)
            sommatoria+=temp[i][giorno-1];
      }
      media=sommatoria/12; // i mesi dell'anno sono sempre 12
      printf("temperatura media del giorno %d: %f\n\n",mese,media);

      printf("Inserire il range di date di cui si vuole conoscere la temperatura
media\n");
      printf("(giorno e mese iniziale     giorno e mese finale)\n");
      scanf("%d %d %d %d", &giornoIn, &meseIn, &giornoFin, &meseFin);
      // per ipotesi le date inserite sono entrambe valide (no 30 Febbraio)
      // ed la data iniziale non è successiva alla data finale
```

```
	sommatoria=0;
	somGiorni=0;
	i=meseIn-1;
	j=giornoIn-1;
	// Il seguente ciclo for somma il valore delle temperatura
	// di tutti i giorni nel range dato, estremi compresi.
	// Il ciclo itera finché la data in esame è precedente o uguale alla data
finale, ovvero:
	// a) il mese corrente è precedente al mese finale
	// b) il mese corrente è il mese finale e
	//          il giorno corrente è precedente o uguale al giorno finale
	while(i<meseFin-1 || (i==meseFin-1 && j<=giornoFin-1) ){
		printf("%d:%d  ",j,i);
		sommatoria+=temp[i][j];
		somGiorni++;
		j++;
		if(j==31 || temp[i][j]==-100){
			// Il mese corrente è finito;
			// riparto dal primo giorno del mese successivo
			j=0; // (primo giorno)
			i++; // (mese successivo)
		}
	}
	media=sommatoria/somGiorni;
	printf("temperatura media effettuata %d giorni: %f\n\n",somGiorni,media);

	return 0; }
```

## ESERCIZI SULLE GRAMMATICHE

**Esercizio**
Data la grammatica **G** con scopo **S** e simboli terminali **{a,c,0,1}**

**S ::= a F c**
**F ::= a S c | E**
**E ::= 0 | 1**

si mostri (mediante derivazione left-most) che la stringa **aaa1ccc**
appartiene alla grammatica

**Soluzione**

S -> aFc -> aaScc -> aaaFccc -> aaaEccc -> aaa1ccc

**ESERCIZIO**
Si consideri la grammatica **G** con scopo **S** e simboli terminali **{a, z, 0, 1}**.

**S ::= C B C | C**
**C ::= B A | A**
**B ::= a | z**
**A ::= 0 | 1**

Si dica se la stringa **a1z0** è sintatticamente corretta secondo tale grammatica, e se ne mostri dla derivazione left-most.

**Soluzione**

La stringa è sintatticamente corretta.

Derivazione left most:

**S -> C B C -> B A B C**
**-> a A B C -> a 1 B C**
**-> a 1 z C -> a 1 z A**
**-> a 1 z 0**

# ESERCIZI SU RAPPRESENTAZIONE DEI NUMERI

**Esercizio**
Un elaboratore rappresenta numeri interi su ***8 bit***. Scrivere tramite questa rappresentazione interna i numeri 17 e 23 (espressi in base 10), e eseguirne la somma (sempre tramite la rappresentazione interna).

**Soluzione**

```
+17 → 00010001
+23 → 00010111
-------------------------
       00101000 →  40 (in base 10)
```

**Esercizio**
Un elaboratore rappresenta numeri interi su ***8 bit*** dei quali ***7*** sono dedicati alla rappresentazione del ***modulo*** del numero e ***uno*** al suo ***segno***.
Indicare qual è l'intervallo di interi rappresentabile. Come varia tale intervallo se si dedicano invece 9 bit alla rappresentazione? (8 dedicati al modulo e uno al segno)

**Soluzione**

L'intervallo di interi va da –127 a +127. Aggiungendo un bit, l'intervallo raddoppia, da –255 a +255

**Esercizio**
Un elaboratore rappresenta numeri interi su ***8 bit*** dei quali ***7*** sono dedicati alla rappresentazione del ***modulo*** del numero e ***uno*** al suo ***segno***. Indicare come viene svolta la seguente operazione aritmetica:

37 + 51

in codifica binaria.

**Soluzione**

```
+37 → 0  0100101
+51 → 0  0110011
```

Tra i (moduli dei) due numeri si esegue una somma:

```
          0100101
```

```
        0110011
        -------
        1011000
```
che vale 88 in base 10

**Esercizio**
Un elaboratore rappresenta numeri interi su ***8 bit*** dei quali ***7*** sono dedicati alla rappresentazione del modulo del numero e ***uno*** al suo ***segno***. Indicare come viene svolta la seguente operazione aritmetica:

59 – 27

in codifica binaria

**Soluzione**

```
59   →0  0111011
-27  →1  0011011
```

Tra i (moduli dei) due numeri si esegue una sottrazione:

```
        0111011
    -   0011011
        -------
        0100000
```

che vale 32 in base 10

**Esercizio**
Un elaboratore rappresenta i numeri interi su 8 bit dei quali 7 sono dedicati alla rappresentazione del modulo del numero e uno al suo segno. Indicare come viene svolta la seguente operazione aritmetica e determinarne il risultato traslandolo poi in decimale per la verifica:

118 - 37

**Soluzione**

```
118    -> 0 1110110
-37    -> 1 0100101
```

Tra i moduli dei numeri si esegue una sottrazione:

```
  1110110
- 0100101
----------
  1010001
```

che vale 81 in base dieci.

## ESERCIZI SU RECORD DI ATTIVAZIONE

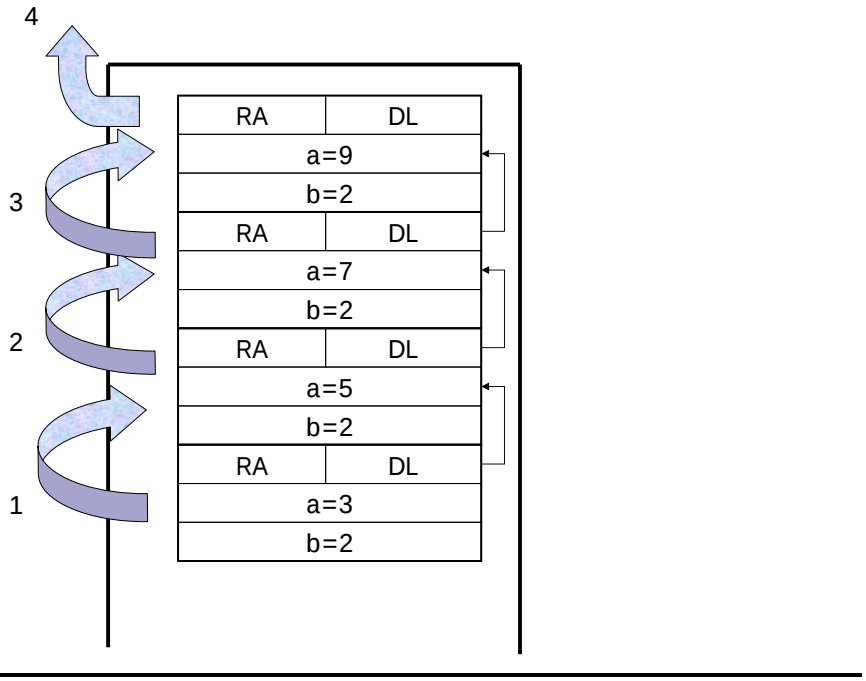
**Esercizio**

Esercizio 3
Si consideri la seguente funzione g():

```
int g(int a, int b) {
     if ((a/b) <2 )
          return (b-1);
     else
          return 1 + g(a-2, b);
}
```

Si scriva il risultato della funzione quando invocata come **`g(9, 2)`** e si mostrino i record di attivazione.

**Soluzione**



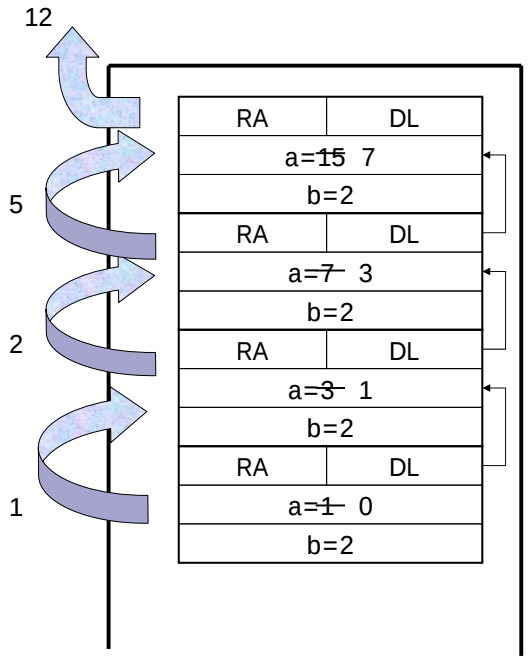
**Esercizio**

Si consideri la seguente funzione FUN():

```
int FUN(int a, int b){
     a=a/b;
     if ((a%b)>0)
          return FUN(a, b) + a;
     else
          return 1;
}
```

Si scriva il risultato della funzione quando invocata come **`FUN(15, 2)`** e si disegnino i corrispondenti record di attivazione.

**Soluzione**



# FUNZIONI RICORSIVE

**Esercizio**
Scrivere una funzione ricorsiva:

int ric(int x)

che calcoli, ricorsivamente, la somma di tutti i numeri compresi tra 0 ed x.

**Soluzione**

```
int ric(int x) {
     if (x == 0)
          return 0;
     else
          return x + ric(x-1);
}
```

**Esercizio**
Scrivere una procedura ricorsiva:

**void print(int list[], int length)**

che stampi, *ricorsivamente*, tutti i numeri contenuti nell'array **list**.

**Soluzione**

```
void print (int list[], int length) {
     if (length == 0)
          return;
     else {
          printf("%d\n", * list);
           print(++list, --length);
     }
}
```

**Soluzione - bis**

```
void print (int list[], int length) {
     if (length == 0)
          return;
     else {
          print(list, length-1);
          printf("%d\n", list[length-1]);
     }
```

```
}
```

**Esercizio**
Scrivere una procedura ricorsiva:

```
void printchar(char stringa[])
```

che stampi, *ricorsivamente*, tutti i caratteri contenuti in **stringa**, un carattere per linea, assumendo che **stringa** sia *ben formata*.

**Soluzione**

```
void printchar (char stringa[]) {
     if (*stringa == '\0')
          return;
     else {
          printf("%c\n", * stringa);
          printchar(stringa+1);
     }
}
```

**Esercizio**
Scrivere una procedura ricorsiva che, ricevuto in ingresso un array di interi, esegua la somma degli interi in posizione con indice dispari.

**Soluzione**

```
int sumOdd2(int list[], int length, int pos) {
     if (pos >=length)
          return 0;
     else
          return list[pos] + sumOdd2(list, length, pos+2);
}

int sumOdd(int list[], int length) {
     return sumOdd2(list, length, 1);
}
```

**Esercizio**
Si scrivano le versioni ricorsiva ed iterativa (utilizzo di while) di una funzione:

```
double  f(double a, int n);
```

che calcoli il seguente valore:

$$\sum_{i=1}^{n}\left(a-\frac{i}{a}\right)$$

**Soluzione**

```
double f(double a, int n)
{ if (n==1) return a – 1/a;
  else return a – n/a + f(a, n-1);
}

double f(double a, int n)
{ int i=1;
  double sum=0;
  while(i<=n)
      {sum = sum + a – i/a;
       i++;}
  return  sum;
}
```

# DOMANDE VARIE DI TEORIA

**Esercizio (domanda)**
Supponiamo di dover assegnare due strutture identiche contenenti un intero e un float:

```
struct prova{
     int A;
     float B;
};
struct prova S1;
struct prova S2;
```

Quali delle seguenti istruzioni non è consentita per effettuare l'assegnamento ?

```
1.   S1 = S2;
2.   S1 == S2;
3.   S1.A = S2.A; S1.B = S2.B;
```

**Soluzione**
2. Non è un operatore di assegnamento ma di confronto.

---

**Esercizio (domanda)**
Qual è la relazione tra vettori e puntatori nel linguaggio C? È lecito scrivere:

```
int V[10], *punt;
punt=V;
```

Motivare la risposta.

**Soluzione:**
Un puntatore è una variabile che ha come valore l'indirizzo di un altro dato. Un vettore rappresenta un indirizzo costante (quello del primo elemento del vettore). La differenza è che tale indirizzo è costante e non se ne può variare il valore.
L'istruzione punt=V è lecita e assegna al puntatore l'indirizzo del primo elemento del vettore (rappresentato dal nome V).

---

**Esercizio (domanda)**
Qual è la differenza tra un parametro formale passato per indirizzo e uno passato per valore ad una procedura:

A. Se modificato all'interno della procedura, il parametro passato per indirizzo non comporta modifica sul parametro attuale. Viceversa avviene per il parametro passato per valore;
B. Se modificato all'interno della procedura, il parametro passato per valore non comporta modifica sul parametro attuale. Viceversa avviene per il parametro passato per indirizzo;

C. I parametri passati per indirizzo sono solo vettori, mentre tutti gli altri non possono essere passati che per valore.

**Soluzione**
La risposta corretta è la B.

---

**Esercizio (domanda)**
Siano date due stringhe char s1[]="Pippo", s2[20];
Se si scrive s1=s2; che cosa succede?

A. Tutto il contenuto di s2 viene copiato in s1
B. Si ottiene un errore di compilazione
C. Il primo elemento di s2 viene ricopiato nel primo elemento di s1

**Soluzione**
B. Si ottiene un errore di compilazione.