# ESERCIZIO: RICERCA BINARIA
## Soluzione Iterativa

```
int ricerca_bin (int vet[], int el)
{int first=0, last=N-1, med=(first+last)/2;
 int T=0;
 while ((first<=last)&&(T==0))
  { if (el==vet[med])
          T=1;
    else
       if (el < vet[med])  last=med-1;
       else   first=med+1;
    med = (first + last) / 2;
  }
 return  T;
}
```



# Ricerca binaria ricorsiva

- Parametri in ingresso:
  - Array in cui cercare
  - Indice first da cui partire
  - Indice last a cui fermarsi nella ricerca
  - Elemento da cercare

- Valori in uscita:
  - Successo della ricerca
  - Posizione dell'elemento nell'array

  →I due valori sono "condensabili"?

  →La posizione in un array è sempre maggiore o uguale a zero

  →Un numero negativo può essere considerato un insuccesso nella ricerca...

# RICERCA BINARIA RICORSIVA

```
int ricerca_bin (int vet[], int first, int last, int el) {

   int med;

   if (first > last)
     return -1;
   else {
     med = (first + last) / 2;
     if (el == vet[med])
          return med;
     else
          if (el > vet[med])
                 return ricerca_bin(vet, med+1, last, el);
          else
                 return ricerca_bin(vet, first, med-1, el);
   }
}
```

# RICERCA BINARIA RICORSIVA
## Versione “compatta”

```
int binarySearch(int vet[], int dim, int el) {
   int startPos;
   int med = dim / 2;

   if (vet[med] == el)
     return med;
   if (med == 0)
     return -1;
   if (el < vet[med])
   {
     return binarySearch(vet, med, el);
   }
   else {
     startPos = med + 1;
     return startPos +
          binarySearch(&vet[startPos], dim - startPos, el);
     }
}
```

# Ricerca binaria: note

- **`&vet[startPos]`**
  → Indirizzo dell'elemento di posizione **`startPos`**
  → Sotto-array parzialmente sovrapposto all'array di partenza (**`vet`**) i cui elementi sono quelli compresi fra **`startPos`** (compreso) e la fine dell'array
- **`startPos + binarySearch(&vet[startPos],`**
  **`dim - startPos, el);`**
  → La ricerca riparte dal sotto-array che inizia da **`startPos`**
  → occorre sommare la posizione di partenza al risultato della sottoricerca
  → la dimensione del sotto-array è **`dim – startPos`**
  → l'elemento da cercare è sempre lo stesso...



# Esercizio analisi (1)

Il seguente programma C compila correttamente? In caso affermativo, quali sono i valori stampati a tempo di esecuzione? (si motivi opportunamente la risposta data)

```
#include <stdio.h>
void F(int v[], int pos) {
     int N=4;       N--;
     *(v+pos+1) = *(v+pos+1) + *(v+pos);
     return; }

int main() {
     int N=4, v[5], i;
     {   ++N; }

     for (i=0; i<N; i++)
        v[i]=2*i+1;

     for (i=0; i<N-1; i++)
        if (v[i]) F(v, i);

     printf("Adesso N vale: %d\n", N);
     for (i=1; i++ < (i?N:0); )
        printf("%d\n", v[i-1]);
     return 0; }
```

# Esercizio analisi (1)
## Soluzione

Il programma compila correttamente e stampa:

```
Adesso N vale: 5
4
9
16
25
```

- la funzione **F** si limita a modificare il valore dell'array con indice (**pos+1**) sommandogli il valore contenuto all'indice **pos**,
- Nel **main**, il primo ciclo **for** provvede ad inizializzare l'array **v** ai valori {**1,3,5,7,9**}; il secondo ciclo invece invoca ripetutamente la funzione **F**, ed all'uscita dal ciclo il vettore viene a valere {**1,4,9,16,25**}.
- Infine viene stampato il il valore di **N** (che in seguito al preincremento inziale passa da 4 a 5), e poi con un ultimo ciclo viene stampato il contenuto del vettore **v** (però solo le ultime 4 posizioni).



# Esercizio analisi (2)

Il seguente programma C compila correttamente? In caso affermativo, quali sono i valori stampati a tempo di esecuzione? (si motivi opportunamente la risposta data)

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 9

int Funz(char y[], int *N, int *M)
{
     int i;
     (*N)++;
     for (i=(*N)-1; i<*N; i++)
         y[i] = y[*M];
     return *N;
}
```

```
int main () {
     char s[] = "Paperone";
     char ss[DIM]; int i=3, N = 2;
     N = (++N)-2;
     N = Funz(s, &N, &i);
     printf("N vale adesso: %d\n",N);
     {
         for (i=0; i<DIM; ++i) {
             *(ss+i) = s[i];
             printf("%c", *(ss+i));
         }
     }
     return 0;
}
```

# Esercizio analisi (2)
## Soluzione

Il programma compila correttamente e stampa:

```
N vale adesso: 2
Peperone
```

- La funzione **`Funz`** incrementa il valore riferito da **`N`**, e poi assegna all'elemento in posizione **`*N-1`** del vettore **`y`** il valore in posizione **`*M`**; restituisce infine il nuovo valore riferito da **`N`**
- Il **`main`** inizia dichiarando alcune vairabili, instanziando **`N`** a **`2`** e poi modificandolo con un assegnamento al valore **`1`**
- Viene poi invocata la funzione **`Funz`**, con parametri (**`"Paperone",1,3`**); per quanto detto, viene modificato l'elemento in posizione **`1`** (**`'a'`**), a cui viene assegnato il valore in posizione 3 (**`'e'`**)
- Inifine l'array **`s`** viene copiato (elemento per elemento) nell'array **`ss`**, che viene poi stampato a video



# Esercizio Record di Attivazione (3)

Si consideri la seguente funzione:

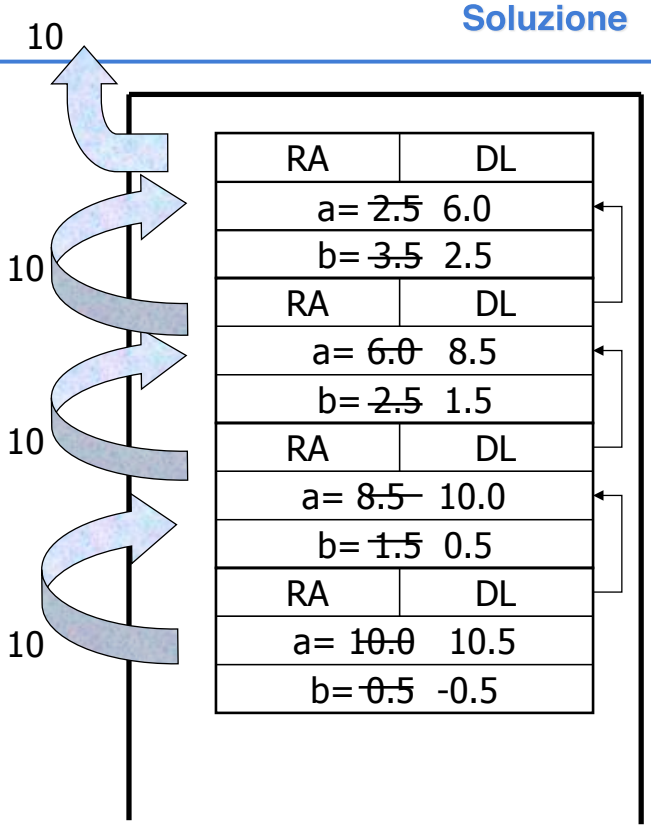
```
int funzione(float a, float b) {
    a = a+b;
    b--;
    if (b > 0)
       return funzione(a, b);
    else
       return a;
}
```

Si scriva il risultato della funzione quando invocata come:

```
funzione(2.5, 3.5)
```

e si disegnino i corrispondenti record di attivazione (si faccia particolare attenzione al fatto che la funzione restituisce un valore intero).

## Esercizio Record di Attivazione (3)
### Soluzione






## Esercizio Record di Attivazione (4)

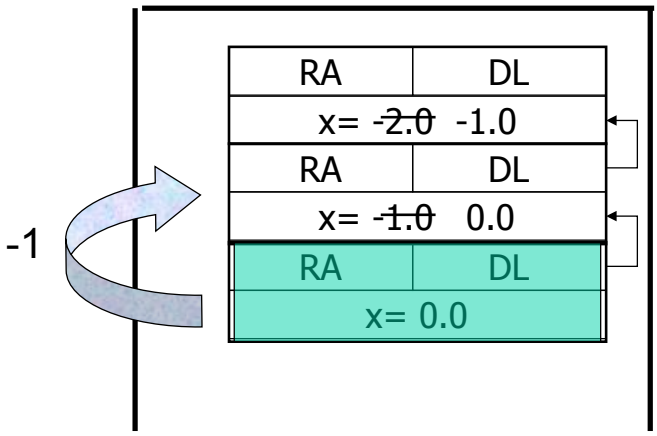
Si consideri la seguente funzione **W**:

```
double W(int x){
   if (x<0) {
      x++;
      return W(x)+W(x/2);
      x++;
   }
   else
      return -1;
}
```

Si scriva il risultato della funzione quando invocata come **W(-2)** e si disegnino i corrispondenti record di attivazione.

## Esercizio Record di Attivazione (4)
### Soluzione

La funzione restituisce il valore –3.00. Supponendo che la valutazione degli addendi nella somma venga fatta a partire da sinistra, si ottiene prima:

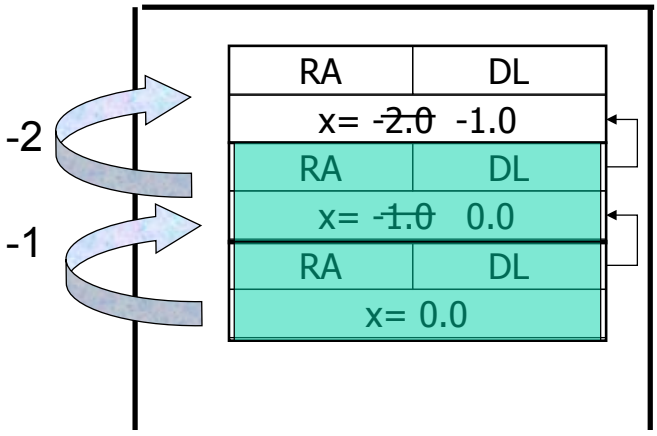



poi l'eliminazione dell'ultimo record, e …



## Esercizio Record di Attivazione (4)
### Soluzione

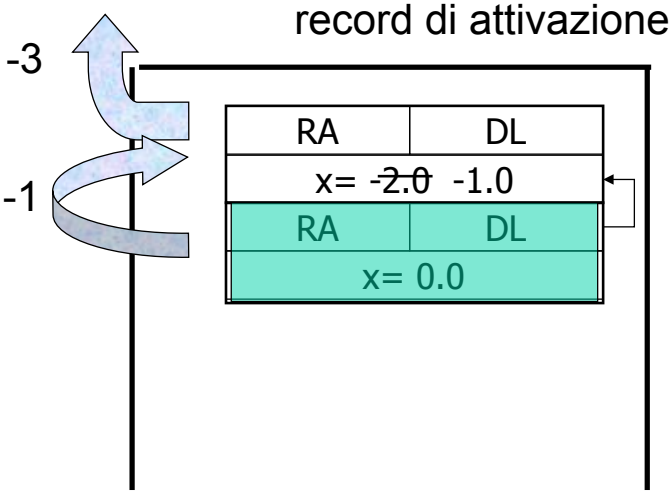
… dove viene fatta l'ultima invocazione di W(0), con somma finale restituita alla prima invocazione di W.




Quindi viene fatta la prima somma con risultato -2.00, restituita indietro

## Esercizio Record di Attivazione (4)
### Soluzione

…poi l'eliminazione dell'ultimo record, e un nuovo ulteriore record di attivazione per W(0):




La somma finale è restituita alla prima invocazione di W.



## Esercizio sintesi (5)

Uno dei più antichi sistemi di codificazione di messaggi segreti si basa sulla sostituzione, secondo un certo ordine, dei caratteri componenti il messaggio.

Ad esempio, dato un messaggio composto dalle lettere:

{a, b, c}

E data una chiave di sostituzione che, per ogni lettera ne associa un'altra:

'a' → 'x'

'b' → 'y'

'c' → 'z'

Il messaggio originale può essere così riscritto:

{x, y, z}

# Esercizio sintesi (5)

Si vuole costruire un sistema di codifica/decodifica di questo tipo, facendo le seguenti assunzioni:

1. Le lettere componenti il messaggio sono tutte minuscole, ed i messaggi non possono contenere altri caratteri che lettere (no spazi, no numeri)
2. Il codice di sostituzione è dato da un array di 26 caratteri, che viene interpretato nel seguente modo: nella posizione ad indice 0 vi è il carattere che deve sostituire la lettera ‘a’, in posizione con indice 1 vi è il carattere che deve sostituire la lettera ‘b’, etc.



# Esercizio sintesi (5)

Si strutturi la soluzione implementando due funzioni:

```
void crypt( char source[],
            int length,
            char code[DIM_ALPHA],
            char dest[]);


void decrypt( char source[],
              int length,
              char code[DIM_ALPHA],
              char dest[]);
```

Ed infine si scriva un semplice main di prova.

# Esercizio sintesi (5)
## Soluzione

```
void crypt(char source[], int length, char code
    [DIM_ALPHA], char dest[]) {
      int i;

      for (i=0; i<length; i++) {
         dest[i] = code[source[i] - 'a'];
      }
}
```



# Esercizio sintesi (5)
## Soluzione

```
void decrypt(char source[], int length, char code
    [DIM_ALPHA], char dest[])
{
    int i;
    int j;
    int pos= -1;

    for (i=0; i<length; i++) {
       for (j=0; j<DIM_ALPHA && pos<0; j++) {
          if(source[i] == code[j])
                pos = j;
       }
       dest[i] = 'a' + pos;
       pos = -1;
    }
}
```

# Esercizio sintesi (5)
## Soluzione

```
#define DIM 256
#define DIM_ALPHA 26

int main()
{
    char source[DIM] = "abc";
    char dest1[DIM] = {'\0',...};
    char dest2[DIM] = {'\0',...};
    char codice[DIM_ALPHA] = "cab";

    printf("ORIGINALE: %s\n", source);
    crypt(source, 3, codice, dest1);
    printf("CRIPTATO: %s\n", dest1);
    decrypt(dest1, 3, codice, dest2);
    printf("DE-CRIPTATO: %s\n", dest2);

    return 0; }
```



# Esercizio sintesi (6)

Si vuole implementare un programma per il calcolo dell'inflazione su determinati prodotti commerciali.

A tal scopo ogni prodotto è rappresentato tramite una struttura **`item`**, definita da una stringa **`name`** con il nome del prodotto, e da due float **`old_price`** e **`new_price`** rappresentanti i prezzi.

# Esercizio sintesi (6)

a) Si scriva una funzione **`lettura()`** che riceva come parametri di ingresso un vettore **`prezzi`** di strutture **`item`**, la dimensione fisica **`max`** del vettore **`prezzi`**, e un puntatore a intero **`num`** che rappresenta la dimensione logica del vettore. La funzione deve leggere da standard input il nome del prodotto ed i due prezzi, e deve copiare tale informazione nella prima posizione libera nel vettore **`prezzi`**.



# Esercizio sintesi (6)

La funzione deve terminare se l'utente inserisce come nome del prodotto il termine "fine", oppure se viene raggiunta la dimensione fisica del vettore.

La dimensione logica del vettore **`prezzi`** così riempito deve essere restituita tramite il parametro **`num`** (passato appunto per riferimento). Al termine della lettura dei dati la funzione deve restituire il valore 0.

# Esercizio sintesi (6)

b) Si scriva un programma **`main`** che, dopo aver definito un vettore di strutture **`item`** (di dimensione massima **`MAX_ITEM`**), invochi la funzione **`lettura()`** per riempire tale vettore.

Il programma stampi poi a video nome e tasso d'inflazione per ogni prodotto, utilizzando la formula:

$$infl_i = \left( \frac{new\_price_i}{old\_price_i} - 1 \right) * 100$$



# Esercizio sintesi (6)
## soluzione

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

#define DIM 21
#define MAX_ITEM 100

typedef struct {
    char name[DIM];
    float old_price;
    float new_price;
} item;
...
```

# Esercizio sintesi (6)
## soluzione

```
int lettura (item prezzi[], int max, int * num) {
     char name[DIM];
     *num = 0;

     printf("Inserire nome prodotto: ");       scanf("%s", name);

     while ((strcmp(name, "fine")) && (*num < max)) {
         strcpy(prezzi[*num].name, name);
         printf("Inserire old price: ");
         scanf("%f", &prezzi[*num].old_price);
         printf("Inserire new price: ");
         scanf("%f%*c", &prezzi[*num].new_price);

         (*num)++;

         printf("Inserire nome prodotto: ");
         scanf("%s", name);
     }   return 0;
}
...
```



# Esercizio sintesi (6)
## soluzione

```
int main() {
     item V[MAX_ITEM];
     int num, i, result;
     float infl;

     result = lettura(V, MAX_ITEM, &num);

     if (result!=0) {
         printf("Problemi durante la lettura...\n");
         exit(-1);
     }

     for (i=0; i < num; i++) {
         infl = (V[i].new_price/V[i].old_price -1)*100;
         printf("Inflazione del prodotto %s: %6.2f%%\n", V[i].name, infl);
     }
     return 0;
}
...
```