ESERCIZIO

Si scriva un programma C che:

a) legga da terminale una sequenza di interi terminati dal valore 0 (uno su ogni linea) e li inserisca in un vettore DIGIT di 20 componenti (si suppone che la sequenza sia di lunghezza minore o uguale a 20);

b) Stampi la media degli interi positivi pari.

Soluzione:

```
#include <stdio.h>

main()
{
    int DIGIT[20];
    int i,j,n_pos_pari;
    float media;

    i=0;
    do
    {
     printf("Inserisci il numero %d: ",i);
     scanf("%d", &DIGIT[i]);
     i=i+1;
    }
    while (DIGIT[i-1]!=0);
/* i-2 e' la posizione dell'ultimo numero
inserito */

media=0;
n_pos_pari=0;
    for(j=0;j<i-1;j++)
    {
     if (DIGIT[j]>0)
        if (DIGIT[j]%2==0)
        {
           media=media+DIGIT[j];
           n_pos_pari++;
        }
    }
media=media/n_pos_pari;
    printf("La media vale: %f\n",media);
}
```

Esercizio

Dato il seguente programma C:

```
#include <stdio.h>
#define D 4
float A(float *V, int k)
{int i;
 float s=0.0;
 for(i=0;i<D; i+=k)
    {
       V[i]=i;
       s=s+V[i];
    }
 return s;
}

main()
{ float V[]={1.5, 2.5, 3.5, 4.5};
 int i;
 for (i=1; i<D; i=i+1)
    V[i]=V[i-1];
 printf("%f\n", A(V,2));
 for(i=0; i<D; i=i+1)
    printf("%f", V[i]);
}
```

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

Soluzione

Dopo la prima iterazione il vettore V vale

`V[D]={1.5,1.5,1.5,1.5}`

perché il ciclo copia ciascun elemento nella posizione successiva, quindi il primo elemento viene replicato per tutto il vettore.
La chiamata della funzione **`A`** restituisce 2 che viene stampato dalla **`printf`**.
Dopo la chiamata il vettore **`V`** vale

`V[D]={0,1.5,2,1.5}`

Il secondo ciclo stampa quindi

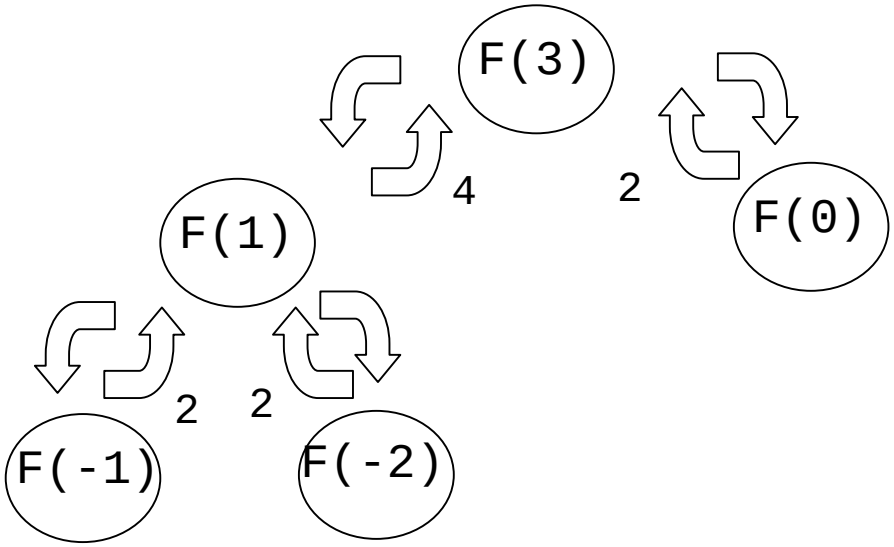
`0,1.5,2,1.5`

Esercizio

Si consideri la seguente funzione F la cui specifica è data in modo ricorsivo (si supponga N intero):
F(N) = restituisce 2 se N <= 0,
F(N-2) * F(N-3), altrimenti
a) Si scriva il risultato della funzione quando chiamata con N= 3 e si mostrino i valori intermedi assunti da N;
b) Si scriva la funzione C che realizzerebbe tale specifica
c) Si mostrino i record di attivazione nello stack

Soluzione:

Sequenza di attivazioni:

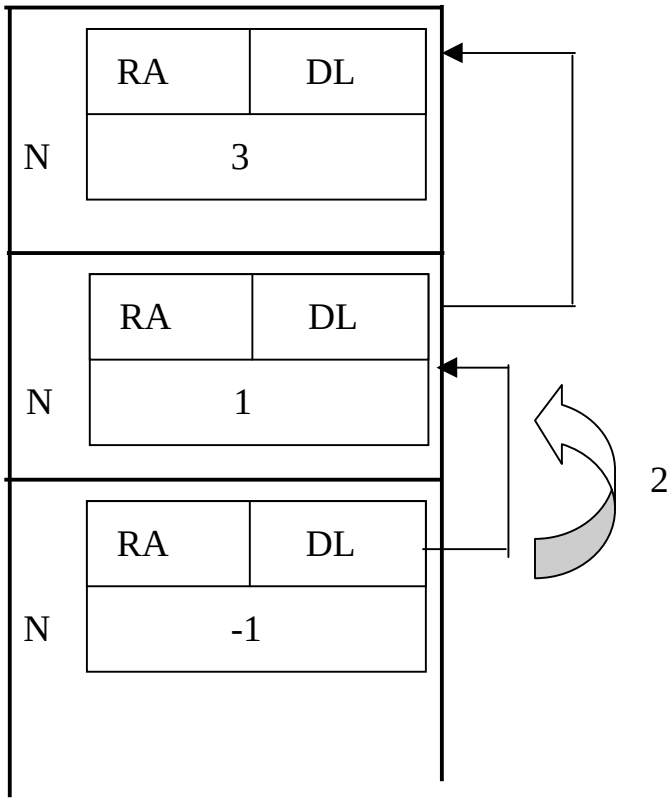


F(3) restituisce il valore **8**

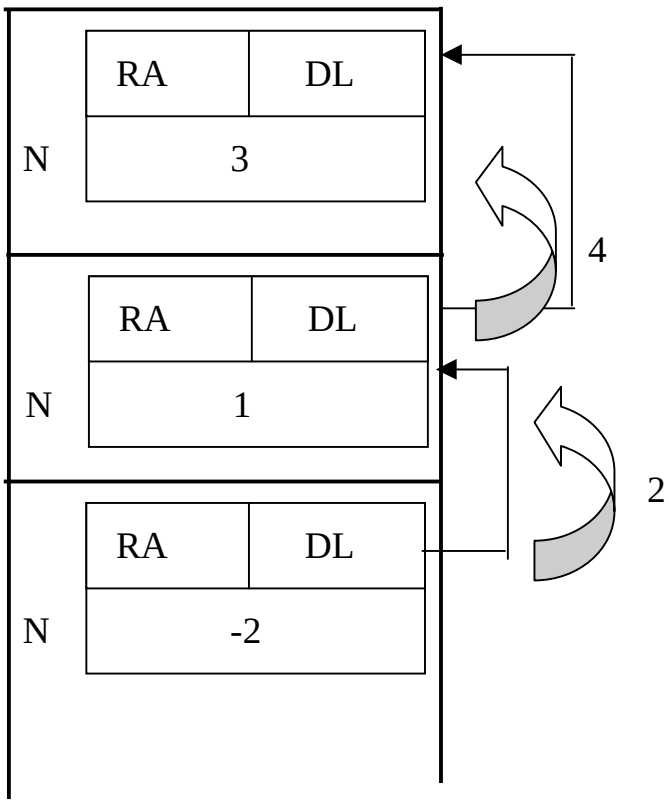
Valori assunti da N: 3 1 -1 -2 0

```
int  F(int N)
{if (N<=0) return 2;
 else  return F(N-2)*F(N-3);
}
```

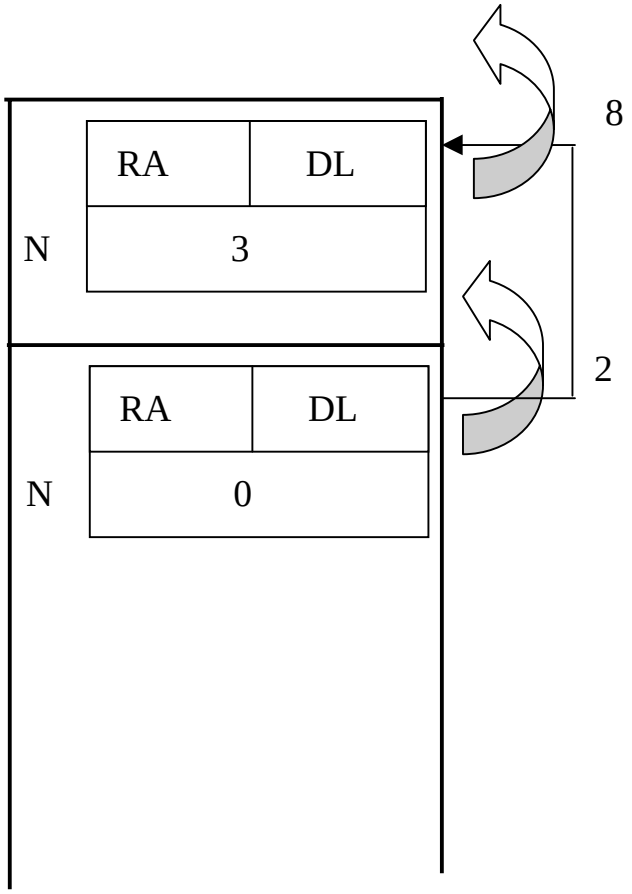
Record di attivazione della chiamata con N = 3



La F(3) chiama la F(1) che a sua volta chiama F(-1). Quest'ultima finisce restituendo il valore 2



Dopo di che la funzione F(1) chiama F(-2). Quest'ultima finisce restituendo il valore 2. Quindi finisce anche F(1) restituendo il valore 4 (risultato della moltiplicazione tra 2 e 2).



Dopo di che la funzione F(3) chiama F(0). Quest'ultima finisce restituendo il valore 2. Quindi finisce anche F(3) restituendo il valore 8 (risultato della moltiplicazione tra 4 e 2).

Esercizio

Indicare che cosa stampa il seguente programma. La risposta deve essere opportunamente motivata.

```
#include <stdio.h>

main()
{     int A[6]={0,0,0,0,0,0};
      int i;

      for (i=0; i<6; i+= 3)
          A[i]= A[i] + 2*i;

      for(i=0; i<6; i++)
          if (A[i])
                printf("%d\n",A[i]);
             else
                printf("%d\n",i-A[i]);
}
```

Soluzione

Dopo il primo ciclo for il vettore A vale:
A=

0	0	0	6	0	0

Il secondo ciclo for stampa quindi:

0	1	2	6	4	5

Esercizio

Si realizzi un programma C che legga da standard input i dati relativi ai corsi tenuti presso una università. In particolare, per ogni corso vengono dati:

- **denominazione** del corso: una stringa che riporta il nome del corso;
- **nome** del docente: una stringa che rappresenta il nome del docente del corso;
- **cognome** del docente: una stringa che rappresenta il cognome del docente del corso;
- **iscritti**: un intero che indica il numero di studenti che frequentano il corso.

Il programma deve stampare la denominazione ed il cognome relativi a tutti i corsi che hanno il numero di iscritti maggiore o uguale alla media aritmetica degli iscritti (calcolata su tutti i corsi).

```
#include <stdio.h>
#define N 300
struct stud {
     char denominazione[20];
     char nome_docente[10];
     char cognome_docente[15];
     int studenti;
 } corsi[N];


main()
{int i, somma, nc;
 float media;

printf("Inserisci il numero dei corsi ");
scanf("%d", &nc);

/* inserimento dati */
for (i=0; i<nc; i++)
{
 printf("Inserisci il nome del corso ");
 scanf("%s",corsi[i].denominazione);
 printf("Inserisci il nome del docente ");
 scanf("%s",corsi[i].nome_docente);
 printf("Inserisci il cognome del docente ");
 scanf("%s",corsi[i].cognome_docente);
 printf("Inserisci il numero degli iscritti");
 scanf("%d",&corsi[i].studenti);
}
somma=0;
for (i=0; i< nc; i++)
 somma=somma + corsi[i].studenti;
media=((float)somma)/nc;
for (i=0; i< nc; i++)
 if (corsi[i].studenti>=media)
     printf("%s %s\n",corsi[i].denominazione,
        corsi[i].cognome_docente);
}
```

Esercizio:

Leggere da input alcuni caratteri alfabetici maiuscoli (per ipotesi al massimo 10) e riscriverli in uscita evitando di ripetere caratteri già stampati.

Soluzione:

```
while <ci sono caratteri da leggere>
    {
    <leggi carattere>;
    if <non già memorizzato>
       <memorizzalo in una struttura dati>;
    };
while <ci sono elementi della struttura
dati>
    <stampa elemento>;
```

Occorre una struttura dati in cui memorizzare (senza ripetizioni) gli elementi letti in ingresso.

```
char A[10];
```

Codifica:

```
#include <stdio.h>

main()
{ char A[10], c;
  int i, j, inseriti, trovato;

inseriti=0;
printf("\n Dammi 10 caratteri: ");
for (i=0; (i<10); i++)
 {
 leggi_carattere(&c);
 printf("letto %c\n", c);
 /* verifica unicita' */
 trovato=0;
 for(j=0;(j<inseriti)&&!trovato; j++)
    { if (c==A[j])
      trovato=1;
    }
 if (!trovato)
    { A[j]=c;
      inseriti++;
      printf("inserito %c\n", c);
     }
 trovato=0;
 }
 printf("Inseriti %d caratteri \n",
              inseriti);
}

void leggi_carattere(char *c)
{ char s[1];
  scanf("%1s",s);
  *c=s[0];}
```

ESERCIZIO:

Programma che calcola la lunghezza di una stringa.

```
#include <stdio.h>

/* lunghezza di una stringa */

main()
{
    char   str[81]; /* stringa di al max.
                       80 caratteri */
    int    i;

printf("\nImmettere una stringa: \t");
scanf("%s",&str); /* %s formato
                     stringa */

for (i=0; str[i]!='\0'; i++);

printf("\nLunghezza: \t %d\n",i);

}
```

Vengono acquisiti i caratteri in ingresso fino al primo carattere di spaziatura (bianco, newline)

ESERCIZIO

Scrivere una funzione con la seguente interfaccia: `void concat(char s1[], char s2[], char s3[])`, che ponga nella stringa s3 la concatenazione delle stringhe s1 e s2. Per esempio: se s1=”eta”, s2=”beta”, al ritorno dalla chiamata a *concat()* si deve avere s3=”etabeta”.
Ovviamente, non si deve fare uso di alcuna funzione di libreria relativa alla gestione delle stringhe.

```
#include <stdio.h>

void concat(char s1[],char s2[], char s3[])
{
   int i=0,j=0;

   while (s1[i] != '\0') s3[j++] = s1[i++];
   i = 0;
   while (s2[i] != '\0') s3[j++] = s2[i++];
   s3[j] = '\0';
}
```

Oppure:

```
void concat(char s1[],char s2[], char s3[])
{
   int i,j;

   for (i=0; s1[i] != '\0'; i++)
      s3[i] = s1[i];
   for (j=i,i=0; s2[i] != '\0'; i++,j++)
      s3[j] = s2[i];
   s3[j] = '\0';
}
```

Esercizio

Scrivere una funzione con la seguente interfaccia `void Scambia(char s[])`, che scambi il primo carattere della stringa *s* con l'ultimo. E' possibile, ma non necessario, fare uso di funzioni di libreria per la gestione delle stringhe.

```
void Scambia(char s[])
{
   int i;
   char c;

   for (i=0; s[i] != '\0'; i++);
/* all'uscita dal ciclo i vale strlen(s) */

   c = s[0];
   s[0] = s[i-1];
   s[i-1] = c;
}
```

ESERCIZIO

Scrivere una funzione che copi in una stringa destinazione una parte di una stringa sorgente, dal carattere **n1** al carattere **n2**. Scrivere un main cliente di prova.

```
#include <stdio.h>

#define N 81

void sottostr(char sorg[],int n1,int n2,char
                                        dest[]);

main() {
    char s[N],d[N];
    int n1,n2;
    printf("Inserire una stringa:");
    scanf(“%s”,s);
    printf("\ninizio sottostringa: ");
    scanf("%d",&n1);
    printf("fine sottostringa: ");
    scanf("%d",&n2);
    sottostr(s,n1,n2,d);
    printf("\nLa sottostringa e': ");
    printf(“%s”,d);
}


void sottostr(char sorg[],int n1, int n2,
                              char dest[])
{
    int i=0;

    while ((sorg[n1] != '\0') && (n1<=n2))
                        dest[i++] = sorg[n1++];
    dest[i] = '\0';
}
```

Osservazione

Non viene fatto alcun controllo sugli estremi della sottostringa, né sulla dimensione della stringa destinazione. Esercizio: modificare il programma inserendo tali controlli.

ESERCIZIO

Progettare un programma che legga una stringa da terminale e riconosca se la stringa è palindroma (per esempio: “acca”,”abcXcba” sono stringhe palindrome).

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

main() {
  char parola[81];
  int i=0,n;

  printf("\nInserire parola: ");
  scanf("%s",parola);
  n = strlen(parola);

  while (i<n/2 && parola[i]==parola[n-i-1])
                                    i++;
  if (i==n/2) printf("\npalindroma");
  else printf("\nnon palindroma");
}
```

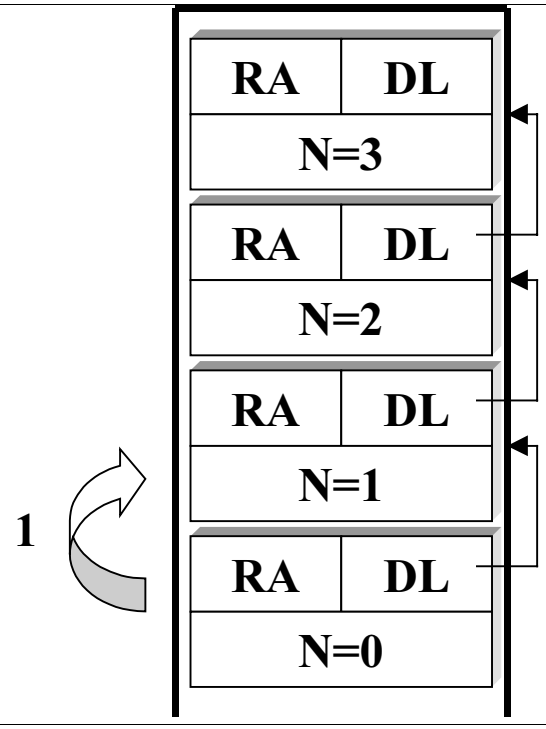
ESERCIZIO:
Si consideri la seguente funzione F la cui specifica è data in modo ricorsivo (si supponga N intero):
F(N) = restituisce 1 se N <= 0,
-1+4*F(N-1) + F(N-2), altrimenti
1. Si scriva la funzione C che realizzerebbe tale specifica
2. Si scriva il risultato della funzione quando chiamata con N= 3 e si mostri la sequenza dei record di attivazione;

SOLUZIONE:

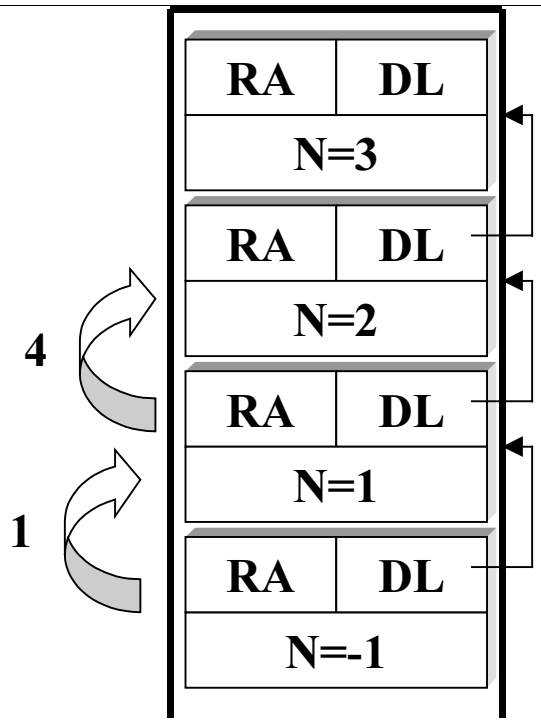
1.

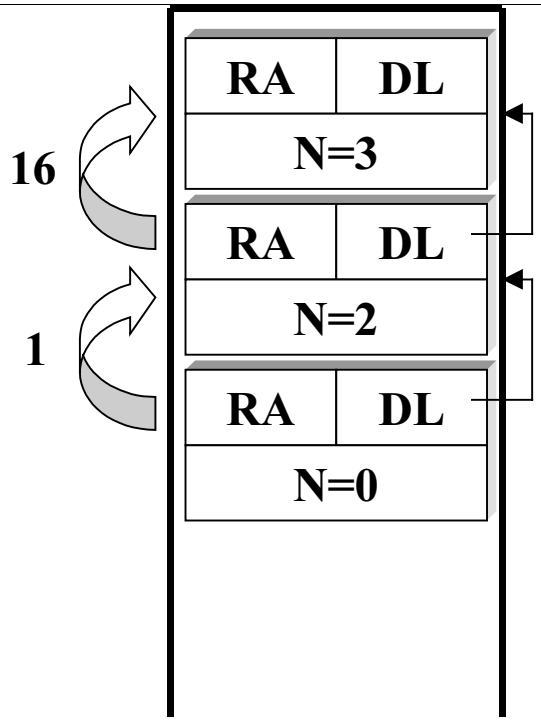
```
int F(int N)
{
    if (N<=0) return 1
     else return -1+4*F(N-1) + F(N-2);
}
```

2.



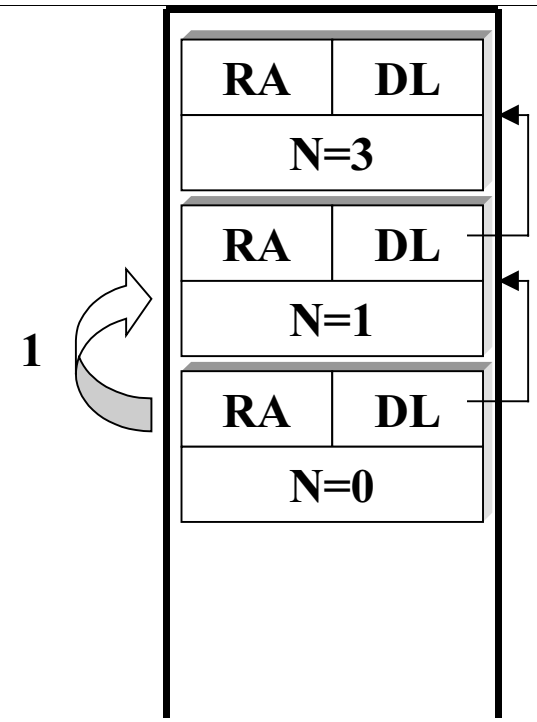
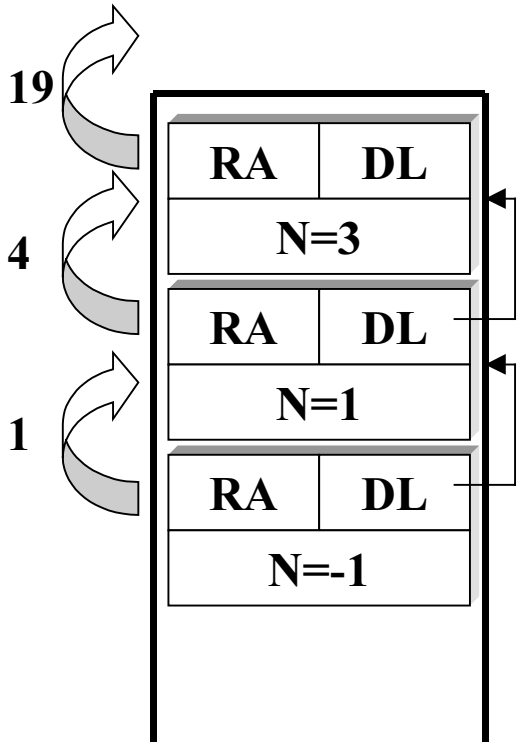
Viene invocata F con N=3, che richiama F(2), che richiama F(1), infine F(0). F(0) restituisce 1 al chiamante

<table>
<tr>
<td> 
</td>
<td>F(1), terminata l’esecuzione di F(0), invoca F(-1), poi termina e fornisce il risultato 4 al chiamante.

F(1)=4</td>
</tr>
<tr>
<td> 
</td>
<td>F(2) invoca F(0), che restituisce 1.

F(2)=-1+4*F(1)+F(0)
= -1 + 4*4 +1
=16

F(2) restituisce il valore 16 al chiamante</td>
</tr>
</table>

	F(3) invoca F(1), che invoca F(0). Come prima, F(0) restituisce 1.
	F(1) invoca F(-1), che restituisce 1. F(1) termina restituendo 4 al chiamante F(3)). F(3)= -1+4*F(2)+F(1) =-1+16+4 =19 F(3) termina e restituisce 19

ESERCIZIO: Numeri Primi

Si scriva una funzione C che calcola se un numero è primo, utilizzando la seguente specifica definita ricorsivamente:

- 2 è primo.
- un numero *N* è primo se non è divisibile per alcun numero primo strettamente compreso fra 2 ed *N*/2.

SOLUZIONE:

```
int div(N,M)
{ return (N % M) == 0;
}

int primo(int N)
{   int i, T = 1;
    if (N == 2)
     return 1
    else
     {for (i=2; i<N/2; i++)
        if (primo(i))
           Trovato = Trovato && !div(N,i);
      return Trovato;
     }
}
```

Esercizio:

Si consideri la seguente procedura P la cui specifica è data in modo ricorsivo (si supponga N intero):

P(*N*) = stampa *N* se *N* <= 10,
stampa *N* ed invoca P(*N*-10), altrimenti

a) Scrivere il codice C di tale procedura.
b) La procedura è tail ricorsiva?
c) Si scriva la sequenza di valori stampati quando la procedura è chiamata con *N*= 30. Si mostri anche la sequenza dei record di attivazione, sia nel caso il compilatore effettui l'ottimizzazione tail, sia se non la effettua.

Soluzione:

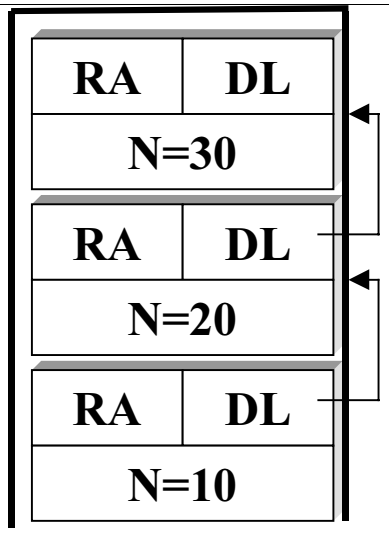
a) Codice della procedura

```
void P(int N)
{ printf("%d ",N);
  if (N>10)
    { P(N-10); }
}
```

b) La procedura è tail ricorsiva, in quanto la chiamata ricorsiva è l'ultima operazione effettuata.

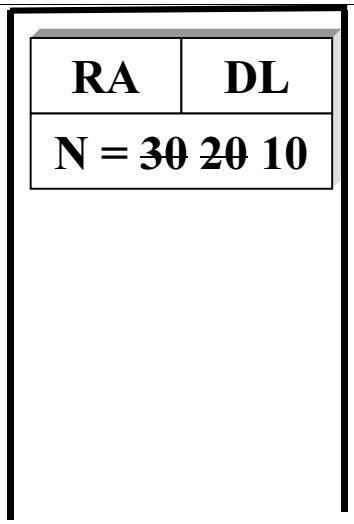
c) La procedura stampa la sequenza

30 20 10



La procedura stampa 30, poi invoca P(20), che stampa il valore 20 ed invoca P(10). P(10) stampa 10 e termina. Il controllo torna a P(20) che termina, poi a P(30) che termina.

Se il compilatore effettua l'ottimizzazione tail, viene riutilizzato sempre lo stesso record di attivazione:



La procedura stampa 30, poi invoca P(20), senza allocare un nuovo record di attivazione. Stampa il valore 20 ed invoca P(10). P(10) stampa 10 e termina. Il controllo torna direttamente al programma chiamante.