

**UNIVERSITA' DEGLI STUDI DI NAPOLI**

**“PARTHENOPE”**

*Relazione di PROGRAMMAZIONE 2*



*a cura di*

***Mattia Capasso, Angelo Rea e Pasquale Salma Jr***

# **INDICE**

## **Capitolo 1. Operatori logici e Bitwise**

Esercizio 1[liv.1]

- Conversione da minuscolo a maiuscolo o viceversa.....8

Esercizio 2[liv.1]

- Rotazione bit.....10

Esercizio 3[liv.1]

- Estrazione variabile e calcolo formula.....13

Esercizio 4[liv.1]

- Estrazione bit più significativi o meno significativi.....19

## **Capitolo 2. Tipo numerico intero**

Esercizio 5[liv.1]

- Conversione da base 10 a base 2.....22

Esercizio 6[liv.1]

- Conversione da base 2 a base 10.....25

Esercizio 7[liv.2]

- Conversione da base 2 a base 10(input stringa di bit)...28

Esercizio 8[liv.1]

- Addizione aritmetica binaria.....30

Esercizio 9[liv.1]

- Sottrazione aritmetica binaria.....33

Esercizio 10[liv.1]

- Calcolo per complemento a 2 e per eccesso-B.....36

Esercizio 11[liv.1]

- Addizione algebrica binaria mediante bitwise.....40

## **Capitolo 3. Tipo numerico reale Floating-point**

Esercizio 12[liv.1]

- Rappresentazione binaria(s,e,m) di un numero float.....44

Esercizio 13[liv.3]	
- Calcolo Conversione da base 10 alla rappresentazione floating-point IEEE Std 754.....	45
Esercizio 14[liv.1]	
- Calcolo Epsilon macchina.....	47
Esercizio 18[liv.1]	
- Somma di molti addendi con raddoppiamento ricorsivo..	51
Esercizio 19[liv1]	
- Somma mediante criterio di arresto naturale.....	54
Esercizio 20[liv.1]	
- Somma di addendi ordinati.....	56
<b>Capitolo 4. Approfondimenti sulle Stringhe</b>	
Esercizio 22[liv.1]	
- Quiz.....	59
Esercizio 23a[liv.1]	
- Lettura di n caratteri uno alla volta(statica).....	61
Esercizio 23b[liv.1]	
- Lettura di n caratteri uno alla volta(dinamica).....	63
Esercizio 24a[liv.1]	
- Concatenazione di due stringhe(statica).....	65
Esercizio 24b[liv.1]	
- Concatenazione di due stringhe(dinamica).....	68
Esercizio 25[liv.1]	
- Prima occorrenza di una sottostringa in una stringa.....	71
Esercizio 26[liv.1-2]	
- Numero totale di occorrenze.....	73
Esercizio 27a[liv.2]	
- Eliminazione di tutte le occorrenze(statica).....	76

Esercizio 27b[liv.2]	
- Eliminazione di tutte le occorrenze(dinamica).....	79
<b>Capitolo 5. Allocazione dinamica della memoria</b>	
Esercizio 29a[liv.1]	
- Visualizzazione per colonne A(mxnx)(statica).....	82
Esercizio 29b[liv.1]	
- Visualizzazione per colonne B(mxnx)(dinamica).....	85
Esercizio 30[liv.1]	
- Prodotto righe per colonne di una matrice.....	88
<b>Capitolo 6. Gestione dei file in C</b>	
Esercizio 33[liv1]	
- Lettura testo mediante variabile buffer.....	93
Esercizio 34[liv.2]	
- Lettura,scrittura e aggiornamento file binari.....	96
Esercizio 36[liv.2]	
- Ricerca occorrenze di un pattern in un file.....	102
<b>Capitolo 7. Strutture dati dinamiche lineari</b>	
Esercizio 37[liv.1]	
- Gestione Pila(stack) tramite Push e Pop.....	104
Esercizio 38[liv.1]	
- Gestione Coda(queue) tramite Enqueue e Dequeue....	108
Esercizio 40[liv.1]	
- Algoritmo di visita di una lista lineare.....	112
Esercizio 42[liv.1]	
- Gestione di una lista.....	115
Esercizio 43[liv.2]	
- Costruzione lista versione iterativa.....	122
Esercizio 44[liv.2]	
- Ordinamento lista in ordine alfabetico.....	124

Esercizio 45a[liv.1]	
- Gestione Pila mediante menu'	127
Esercizio 45b[liv.1]	
- Gestione Coda mediante menu'	132
Esercizio 46a[liv.2]	
- Gestione Lista circolare mediante menu'	137
Esercizio 46b[liv.2]	
- Gestione Lista bidirezionale mediante menu'	146
<b>Capitolo 8. Strutture dati dinamiche gerarchiche</b>	
Esercizio 47[liv.2]	
- Costruzione e visita albero qualsiasi mediante array.....	155
Esercizio 48[liv.3]	
- Costruzione e visita albero qualsiasi mediante	
liste multiple.....	155
Esercizio 49[liv.1]	
- Visita preorder,inorder,postorder mediante array.....	162
Esercizio 50[liv.3]	
- Visita preorder,inorder,postorder mediante	
liste multiple.....	166
Esercizio 51[liv.2]	
- Albero binario di ricerca mediante array.....	169
Esercizio 52[liv.3]	
- Albero binario di ricerca mediante liste multiple.....	172
Esercizio 53[liv.3]	
- Decodifica codice morde.....	173
Esercizio 54[liv.2]	
- Heap mediante array.....	175
Esercizio 55[liv.3]	

- Heap di un albero binario mediante array ..... 178

## **Capitolo 9. Strutture dati dinamiche reticolari**

Esercizio 56[liv.1]

- Grafo non orientato mediante matrice di adiacenze ..... 182

Esercizio 57[liv.1]

- Grafo orientato mediante matrice di adiacenze ..... 186

Esercizio 58[liv.2]

- Grafo non orientato mediante liste di adiacenze ..... 191

## **Capitolo 10. Tecniche di programmazione ricorsiva**

Esercizio 63[liv.1]

- Somma iterativa e ricorsiva delle componenti  
di un array ..... 194

Esercizio 64[liv.1]

- Calcolo iterativo e ricorsivo della potenza intera  
 $x^n$  di un numero reale ..... 197

## **Capitolo 11. Algoritmi di ordinamento**

Esercizio 69[liv.1]

- Selection Sort (iterativo e ricorsivo) su array di struttura  
con scambi reali e virtuali ..... 202

Esercizio 71[liv.1]

- Bubble Sort su array di struttura con scambi  
reali e virtuali ..... 204

Esercizio 72[liv.1]

- Insertion Sort su array di struttura ..... 205

Esercizio 73[liv.2]

- Merge Sort iterativo e ricorsivo ..... 209

## Esercizio 1[liv.1]

### - Conversione da minuscolo a maiuscolo o viceversa

```
1.  /**[liv.1] Scrivere una function C
2.  char low_upp(char ch)
3.  che cambia il carattere in input da minuscolo a maiuscolo e viceversa automaticamente.*/
4.
5.
6.          /**LIBRERIE**/
7. #include <stdio.h>
8. #include <stdlib.h>
9.
10.         /**PROTOTIPI**/
11. char low_upp(char ch);
12.
13.         /**MAIN**/
14. int main()
15. {
16.
17.     char ch=0;
18.
19.     printf("\n");
20.
21.     printf("\t Inserisci il carattere da convertire [A,a...z,Z]: ");
22.     fflush(stdin);
23.     scanf("%c",&ch);
24.
25.     printf("\n\n");
26.
27.     printf("\t Carattere convertito: %c", low_upp(ch));
28.     printf("\n");
29.
30.     return 0;
31. }
32.
33.         /**FUNCTION**/
34. /*La function prende in input un carattere e restituisce un dato dello stesso tipo.
35. IDEA: si puÃ² agire sul sesto bit della stringa mediante l'operatore XOR.
36. Utilizzando il valore 32, il sesto bit della seconda stringa avra' valore 1,
37. quindi se la prima
38. stringa avra' valore 0 allora verra' settato ad 1, se avra' valore 1 allora
39. verra' settato a 0.
40. */
41. char low_upp(char ch)
42. {
43.     return ch^32;
44. }
```

## ESECUZIONE 1

```
Inserisci il carattere da convertire [A,a...,Z]: a
```

```
Carattere convertito: A
```

```
Process returned 0 (0x0)    execution time : 1.789 s  
Press ENTER to continue.
```

## ESECUZIONE 2

```
Inserisci il carattere da convertire [A,a...,Z]: A
```

```
Carattere convertito: a
```

```
Process returned 0 (0x0)    execution time : 2.841 s  
Press ENTER to continue.
```

## Esercizio 2[liv.1]

### - Rotazione bit

```
1. /**[LV.1] Scrivere una function C
2.
3. char rotate(char ch, char n_bit)
4.
5. per ruotare di n bit (n_bit), verso sinistra o verso destra
6. (rispettivamente per n_bit<0 e per n_bit>0), il contenuto di
7. una variabile char mediante gli operatori bitwise.*/
8.
9. /**LIBRERIE*/
10. #include <stdio.h>
11. #include <stdlib.h>
12. #include <math.h>
13. #define MAX_LEN 32
14.
15. /**PROTOTIPI*/
16. char rotate(char ch, char n_bit);
17. void bit_show(short len, char ch, short bit[MAX_LEN]);
18.
19. /**MAIN*/
20. int main()
21. {
22.     char ch=0;
23.     short bit[MAX_LEN],n_bit;
24.
25.
26.     printf("\n");
27.
28. //Leggo il carattere da ruotare
29. printf("\t Inserire il carattere da ruotare: ");
30. fflush(stdin);
31. scanf("%c",&ch);
32. //Visualizzo il carattere da ruotare in binario
33. printf("\t Carattere in binario: "); bit_show(sizeof(char), ch, bit);
34.
35. printf("\n");
36. //Inserisco il numero di bit da ruotare
37. printf("\t Inserire il numero di bit da shiftare: ");
38. fflush(stdin);
39. scanf("%hd",&n_bit);
40.
41. printf("\n");
42. //Richiamo la function rotate per visualizzare il carattere shiftato
43. printf("\t Carattere shiftato(char): %c", rotate(ch, n_bit));
44. printf("\n");
45.
46. return 0;
47. }
48.
49. /**FUNCTION*/
50. char rotate(char ch, char n_bit)
51. {
52.     short scelta=0,bit[MAX_LEN];
53.     char temp=ch;
54.
55. //Scelgo se ruotare i bit verso destra o verso sinistra
56. printf("\t Scegliere se shiftare verso destra o sinistra: \n \t 1) Destra \n
   \t 2) Sinistra \n \t Inserire scelta: ");
57. fflush(stdin);
58. scanf("%hd", &scelta);
59.
```

```

60. printf("\n");
61.
62. if(scelta==1)
63. {
64. //Maschera x non perdere i bit shiftati
65. ch=ch>>n_bit; //il carattere ch viene shiftato di n_bit, dove ogni shift a destra rappresenta una divisione per 2 (esempio = ch=ch>>1 bit
66. //equivale a dire ch=ch/2
67. temp=temp<<(8-n_bit);
68. ch=ch^temp;
69.
70. printf("\n");
71. printf("\t Carattere shiftato in binario: "); bit_show(sizeof(char), ch, bit
    );
72. }
73. else if(scelta==2)
74. {
75. //Maschera x non perdere i bit shiftati
76. ch=ch<<n_bit; //il carattere ch viene shiftato di n_bit, dove ogni shift a sinistra rappresenta un prodotto per 2 (esempio = ch=ch<<1 bit
77. //equivale a dire ch=ch*2
78. temp=temp>>(8-n_bit);
79. ch=ch^temp; //in questo caso effettuiamo un operazione booleana di or esclusivo, quindi avremo: ^|0|1|
80. //0|0|1|
81. //1|1|0|
82. printf("\n");
83. printf("\t Carattere shiftato in binario: "); bit_show(sizeof(char), ch, bit
    );
84. }
85.
86. return ch;
87. }
88.
89. /*Function per estrarre i bit*/
90. void bit_show(short len, char ch, short bit[MAX_LEN])
91. {
92. short jc; char c;
93. for (jc=0; jc<MAX_LEN; jc++)
94. bit[jc]=0;//inizializza bit[jc] con tutti 0
95.
96. //estrae i bit
97. for (jc=0; jc<len; jc++) //len, in questo caso, vale 8 ,dato che come parametro gli e' stato passato un sizeof(char) ovvero 1 byte= 8 bit
98. {   c=ch;
99. for (j=0; j<8; j++)
100. {      bit[j+8*jc]=c&1;
101.      c=c>>1;
102.    }
103.  }
104.
105. //stampa i bit
106. for (j=8*len-1; j>=0; j--)
107. printf("%hd", bit[j]);
108.
109. printf("\n");
110. }
```

## ESECUZIONE 1

```
Inserire il carattere da ruotare: z
Carattere in binario: 01111010

Inserire il numero di bit da shiftare: 1

Scegliere se shiftare verso destra o sinistra:
1) Destra
2) Sinistra
Inserire scelta: 1

Carattere shiftato in binario: 00111101
Carattere shiftato(char): =

Process returned 0 (0x0)    execution time : 6.982 s
Press ENTER to continue.
```

## ESECUZIONE 2

```
Inserire il carattere da ruotare: )
Carattere in binario: 00101001

Inserire il numero di bit da shiftare: 1

Scegliere se shiftare verso destra o sinistra:
1) Destra
2) Sinistra
Inserire scelta: 2

Carattere shiftato in binario: 01010010
Carattere shiftato(char): R

Process returned 0 (0x0)    execution time : 9.288 s
Press ENTER to continue.
```

## Esercizio 3[liv.1]

### - Estrazione variabile e calcolo formula

```
1.  /**[liv.1] Scrivere una function C che, dopo aver estratto i bit da una variabile intera X
   (tipo char, short o
2.  int) visualizzi i bit e poi calcoli il valore corrispondente dalla formula:
3.
4.  Val_X = b[n-1]*2^(n-1) + ... + b[2]*2^2 + b[1]*2^1 + b[0]*2^0
5.
6.  dove b e' l'array dei bit di X: b[j], per j=0, 1, ..., n-
   1 dal meno significativo al piu' significativo, (dove n=8 per il
7.  tipo char, n=16 per il tipo short o n=32 per il tipo int). Confrontare il risultato con il
   valore immesso per
8.  la variabile intera X dichiarata una volta signed ed un'altra unsigned.*/
9.
10.             /**LIBRERIE E COSTANTI*/
11. #include <stdio.h>
12. #include <stdlib.h>
13. #include <math.h>
14. #include <cctype.h>
15.
16.
17. #define MAX_LEN    32
18.
19. /* COSTRUTTO UNION
20. Serve una function che visualizzi la rappresentazione binaria di un numero di tipo qualunque
   Per realizzare
21. una function del genere occorre usare il costrutto UNION.
22. La dichiarazione risulta simile a quella delle struct in C.
23. La differenza tra i due costrutti riguarda come sono allocati i byte della memoria
24. per ciascun tipo di dato. In particolare:
25. - il size di una struct e' dato dalla somma dei size dei singoli campo
26. - il costrutto UNION alloca la stessa area di memoria per le variabili comprese tra parentesi
   si
27. Quindi, visto che la UNION usa lo stesso numero di byte per le varie variabili, avremo che:
28.     - la variabile L (e' un long) utilizzerà 32 bit (4 byte)
29.     - la variabile S (array di 2 short) utilizzerà 2x16bit = 32 bit
30.     - la variabile C (array di 4 char) utilizzerà 4x8bit = 32 bit
31. Le 3 variabili condividono la stessa area di memoria e, quindi, fanno riferimento alle stesse locazioni di memoria.
32. La variabile W è dichiarata di tipo union word32bit. In base alle necessità, può essere interpretata come long, 2short o 4char */
33. union word32bit
34. {
35.     long la;
36.     short sa[2];
37.     char ca[4];
38. } W;
39.             /**PROTOTIPI*/
40. void bit_show(short len, char ch[], short bit[MAX_LEN]);
41. void Var_X(short menu, short bit[MAX_LEN]);
42.
43.
44.             /**MAIN*/
45. int main()
46. {
47.     //definisce l'array di grandezza MAX_LEN che conterrà i bit estratti dalla variabile inserita
48.     short bit[MAX_LEN], scegli, menu=0;
49.
50.
51.     /*SELEZIONE DELLA SCELTA*/
52.     printf("\n\t *****");
```

```

53.     printf("\n\t **      MENU'    **");
54.     printf("\n\t *****");
55.
56.     do
57.     {
58.         printf("\n\n\t Scegliere i bit di quale variabile estrarre:");
59.
60.         printf("\n\t [1] long");
61.         printf("\n\t [2] short");
62.         printf("\n\t [3] char");
63.         printf("\n\n\t -----> ");
64.         fflush(stdin);
65.         scanf("%hd", &menu);
66.
67.         Var_X(menu,bit);
68.
69.         printf("\n\t Vuoi eseguire un'altra operazione?\n\t 1) SI\n\t 2) NO\n\t -----");
70.         fflush(stdin);
71.         scanf("%hd", &scegli);
72.     }
73.     while(scegli != 2);
74.
75.     return 0;
76. }
77.
78.
79.         /**FUNCTION*/
80. void Var_X(short menu, short bit[MAX_LEN])
81. {
82.     int i;
83.         /*variabili signed e unsigned che conterranno la stringa di bit convertita in un numero calcolata mediante la formula dalla stringa di bit estratta */
84.     unsigned char var_cu=0;
85.     char var_cs=0;
86.
87.     unsigned short var_su=0;
88.     short var_ss=0;
89.
90.     unsigned long var_lu=0;
91.     long var_ls=0;
92.
93.     printf("\n");
94.
95.
96.     switch(menu)
97.     {
98.
99.         /*TIPO LONG:
100.          Ha a disposizione 4byte (32bit) quindi puo' rappresentare 2^32 (4 294 967 296) valori.
101.          Con i signed long si possono rappresentare numeri nel range [(+2^31)-1,-2^31].
102.          Con gli unsigned short si possono rappresentare (2^32) - 1 valori */
103.         case 1: //IMMISSIONE DI UN LONG
104.             printf("\n\t Inserisci intero long: ");
105.             fflush(stdin);
106.             scanf("%ld", &W.la);
107.             printf("\t Bit estratti: \n\t");
108.             bit_show(sizeof(long), W.ca, bit);
109.                 /*inserisce il risultato in una variabile signed long*/
110.                 for (i[sizeof(long)*8; i>=0; i--)
111.                     var_ls = (var_ls+(bit[i]*pow(2,i)));
112.
113.                 /*inserisce il risultato in una variabile unsigned long*/
114.                 for (i[sizeof(long)*8; i>=0; i--)
115.                     var_lu = (var_lu+(bit[i]*pow(2,i)));
116.

```

```

117.         printf("\n\tValore stringa binaria signed long: %ld",var_ls);
118.         printf("\n\tValore stringa binaria unsigned long: %ld\n\n",var_lu);
119.         break;
120.
121.     /*TIPO SHORT:
122.      Ha a disposizione 2byte (16bit). E' possibile rappresentare 65536 valori differenti
123.
124.      Con i signed short si possono rappresentare numeri compresi nel range [+32767,-
125.      32168]
126.      Con gli unsigned short si possono rappresentare i numeri fra 0 e 65536. */
127.      case 2: //IMMISSIONE DI UNO SHORT
128.          printf("\n\t Inserisci intero short: ");
129.          fflush(stdin);
130.          scanf("%hd",&(W.sa[0]));
131.          printf("\t Bit estratti: \n\t");
132.          bit_show(sizeof(short),W.ca,bit);
133.          /*inserisce il risultato in una variabile signed short*/
134.          for (i[sizeof(short)*8; i>=0; i--)
135.              var_ss = (var_ss+(bit[i]*pow(2,i)));
136.
137.          /*inserisce il risultato in una variabile unsigned short*/
138.          for (i[sizeof(long)*8; i>=0; i--)
139.              var_su = (var_su+(bit[i]*pow(2,i)));
140.
141.          printf("\n\tValore stringa binaria signed long: %ld",var_ss);
142.          printf("\n\tValore stringa binaria unsigned long: %ld\n\n",var_su);
143.          break;
144.
145.      /*TIPO CHAR:
146.      Ha a disposizione 1byte (8bit). E' possibile rappresentare 256 valori differenti.
147.      Con i signed char si possono rappresentare numeri compresi nel range [+127,-127].
148.      Con gli unsigned char si possono rappresentare i numeri nell'intervallo [0,255].
149.      Ad esempio, se si vuole inserire un carattere il cui codice ASCII e' >127:
150.          - nella variabile unsigned sara' memorizzato correttamente
151.          - nella variabile signed sara' memorizzato il complemento a 2 del numero su 8
152.              bit.
153.          Questo perche' quando tentiamo di inserire in una variabile, che puo' memorizzare
154.              fino
155.              ad un certo range, un valore che e' esterno al range, in tale variabile viene memor
156.              izzato
157.              il complemento a 2 del numero su n bit */
158.      case 3: //IMMISSIONE DI UN CHAR
159.          printf("\n\t Inserisci char: ");
160.          fflush(stdin);
161.          scanf("%s", &(W.ca[0]));
162.          printf("\t Bit estratti: \n\t");
163.          bit_show(sizeof(short),W.ca,bit);
164.          /*inserisce il risultato in una variabile signed long*/
165.          for (i[sizeof(long)*8; i>=0; i--)
166.              var_cs = (var_cs+(bit[i]*pow(2,i)));
167.
168.          printf("\n\tValore stringa binaria signed long: %ld",var_cs);
169.          printf("\n\tValore stringa binaria unsigned long: %ld\n\n",var_cu);
170.          break;
171.      }
172.  }
173.
174.
175. /*Function per estrarre i bit*/
176. void bit_show(short len, char ch[], short bit[MAX_LEN])
177. {

```

```
178.     short j,jc; char c;
179.
180.     //estrae i bit
181.     for (j=0; j<MAX_LEN; j++)
182.         bit[j]=0;
183.     for (jc=0; jc<len; jc++)
184.     {
185.         c=ch[jc];
186.         for (j=0; j<8; j++)
187.             {   bit[j+8*jc]=c&1;
188.                 c=c>>1;
189.             }
190.
191.     //stampa i bit
192.     for (j=8*len-1; j>=0; j--)
193.     {
194.         printf("%d", bit[j]);
195.         if(!(j%4))
196.             printf(" "); // stampa a gruppi di 4 bit
197.     }
198.
199.     printf("\n");
200. }
```

## ESECUZIONE 1

```
-----  
*****  
** MENU' **  
*****  
  
Scegliere i bit di quale variabile estrarre:  
[1] long  
[2] short  
[3] char  
  
-----> 1  
  
Inserisci intero long: 1294967  
Bit estratti:  
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0011 1100 0010 0111 0111  
  
Valore stringa binaria signed long: 1294967  
Valore stringa binaria unsigned long: 1294967  
  
Vuoi eseguire un'altra operazione?  
1) SI  
2) NO  
----->
```

## ESECUZIONE 2

```
Scegliere i bit di quale variabile estrarre:  
[1] long  
[2] short  
[3] char  
  
-----> 2  
  
Inserisci intero short: 20000  
Bit estratti:  
0100 1110 0010 0000  
  
Valore stringa binaria signed long: 20000  
Valore stringa binaria unsigned long: 20000
```

```
Vuoi eseguire un'altra operazione?  
1) SI  
2) NO  
----->
```

### ESECUZIONE 3

Scegliere i bit di quale variabile estrarre:

- [1] long
- [2] short
- [3] char

-----> 3

Inserisci char: 122

Bit estratti:

0011 0010 0011 0001

Valore stringa binaria signed long: 49

Valore stringa binaria unsigned long: 49

Vuoi eseguire un'altra operazione?

- 1) SI
- 2) NO

----->2

Process returned 0 (0x0) execution time : 47.197 s  
Press ENTER to continue.

## Esercizio 4[liv.1]

### - Estrazione bit più significativi o meno significativi

```
1.  /**[ES4] [liv.1] Scrivere una function C per estrarre dalla variabile intera X i k bit piu' significativi o meno significativi,
2.  dove X e k sono i parametri di input, usando:
3.  1) Una maschera.
4.  2) L'operatore di shift (>> o <<).
5.  3) Il prodotto o la divisione per potenze di 2.*/
6.
7.          /**LIBRERIE*/
8. #include <stdio.h>
9. #include <stdlib.h>
10. #include <math.h>
11.
12. #define n_len 16
13.
14.          /**PROTOTIPI*/
15. void estrai_bit(short X, short k);
16. void bit_short(short n, unsigned char bit[n_len]);
17.
18.          /**MAIN*/
19. int main()
20. {
21.     short X,k,i=0;
22.     unsigned char bit[n_len];
23.
24.
25.     printf("\n\t Inserire un numero: ");
26.     fflush(stdin);
27.     scanf("%hd", &X);
28.
29.     printf("\n\t I bit della variabile sono: ");
30.     bit_short(X,bit);
31.     for(i=0; i<n_len; i++)
32.         (i%4 == 0) ? printf(" %1u", bit[i]) : printf("%1u", bit[i]);
33.
34.     printf("\n\t Inserire il numero di bit da estrarre: \n\t --> ");
35.     fflush(stdin);
36.     scanf("%hd", &k);
37.
38.     estrai_bit(X,k);
39.
40.     return 0;
41. }
42.
43.
44.          /**FUNCTION*/
45. void estrai_bit(short X, short k)
46. {
47.     short scelta, mask=0,i,temp;
48.     unsigned char bit[n_len];
49.
50.     printf("\n\t Vuoi estrarre i piu' significativi o meno significativi?\n\t 1)Piu' significativi\n\t 2)Meno significativi\n\n\t --> ");
51.     fflush(stdin);
52.     scanf("%hd", &scelta);
53.
54.
55.     if(scelta==1) //Si estraggono i k bit PIU SIGNIFICATIVI
56.     {
57.         mask=1<<k;
58.         mask=mask-1;
```

```

60.         mask=mask<<(16-k);
61.         temp=mask&X;
62.         printf("\n\t");
63.         bit_short(temp,bit);
64.
65.         for(i=0; i<k; i++)
66.             (i%4 == 0) ? printf(" %1u", bit[i]) : printf("%1u", bit[i]);
67.
68.     }
69.     else if(scelta==2) //Si estraggono i k bit MENO SIGNIFICATIVI
70.     {
71.         mask=1<<k;
72.         mask=mask-1;
73.         temp=mask&X;
74.         printf("\n\t");
75.         bit_short(temp,bit);
76.
77.         for(i=16-k; i<=15; i++)
78.             (i%4 == 0) ? printf(" %1u", bit[i]) : printf("%1u", bit[i]);
79.     }
80. }
81.
82.
83. void bit_short(short n, unsigned char bit[n_len])
84. {
85.     short j;
86.     j=n_len-1;
87.
88.     do
89.     {
90.         bit[j--]
91.         ]=n&1; //ricava un bit per volta tramite AND tra i bit della variabile e il bit 1
92.         n=n>>1; //e' come se realizzasse una divisione di n per 2^1 shift per ricavare il bit successivo
93.     }
94.     while(n!=0 && j>=0);
95.
96.     if (j>=0) //nel caso j non sia 0 gli altri bit della variabile vengono posti a 0
97.     {
98.         do
99.         {
100.             bit[j--]=0;
101.         }
102.     }
103. }

```

## ESECUZIONE 1

```
Inserire un numero: 20

I bit della variabile sono: 0000 0000 0001 0100
Inserire il numero di bit da estrarre:
--> 4

Vuoi estrarre i piu' significativi o meno significativi?
1)Piu' significativi
2)Meno significativi

----> 1

0000
Process returned 0 (0x0)    execution time : 27.368 s
Press ENTER to continue.
```

## ESECUZIONE 2

```
Inserire un numero: 7

I bit della variabile sono: 0000 0000 0000 0111
Inserire il numero di bit da estrarre:
--> 2

Vuoi estrarre i piu' significativi o meno significativi?
1)Piu' significativi
2)Meno significativi

----> 2

11
Process returned 0 (0x0)    execution time : 9.998 s
Press ENTER to continue.
```

## ESECUZIONE 3

```
Inserire un numero: -12

I bit della variabile sono: 1111 1111 1111 0100
Inserire il numero di bit da estrarre:
--> 3

Vuoi estrarre i piu' significativi o meno significativi?
1)Piu' significativi
2)Meno significativi

----> 1

111
Process returned 0 (0x0)    execution time : 9.178 s
Press ENTER to continue.
```

## Esercizio 5[liv.1]

### - Conversione da base 10 a base 2

```
1.  /**[liv.1] Scrivere due function C di conversione di un intero positivo (int) da base 10 a
2.  base 2
3.  mediante l'algoritmo delle divisioni successive realizzato rispettivamente:
4.  - Usando gli operatori di quoziente e resto della divisione intera;
5.  - Usando gli operatori bitwise.
6.
7.          /**LIBRERIE E DEFINE**/
8. #include <stdio.h>
9. #include <stdlib.h>
10. #include <math.h>
11.
12.         /**PROTOTIPI**/
13. void Char_bit_divison(unsigned int numero, unsigned char bit[8]);
14. void estrai_bit(unsigned int n, unsigned char bit[8]);
15.
16.         /**MAIN**/
17. int main()
18. {
19.     unsigned int numero,k;
20.     unsigned char bit[8];
21.
22.     printf("\n\t Inserire numero intero positivo in base 10: ");
23.     fflush(stdin);
24.     scanf("%u", &numero);
25.
26.     Char_bit_divison(numero, bit);
27.
28.     estrai_bit(numero,bit);
29.     printf("\n\n\t *Utilizzo l'algoritmo estrai_bit per confrontare \n\t"
30.             " il risultato con quello della funzione char_bit_division.*\n\n\t Conversione
            con char_bit_division= ");
31.     for(k=0; k<8; k++)
32.         (k%4==0) ? printf(" %u",bit[k]) : printf("%u",bit[k]);
33.
34.     printf("\n\n");
35.     return 0;
36. }
37.
38.
39.         /**FUNCTION**/
40. /*Function che permette di convertire un numero intero positivo in binario mediante la cost
        ruzione
41. dell'array dei resti ottenuti della divisione tra il numero intero inserito e 2, cioe' la b
        ase del
42. sistema di numerazione in cui vogliamo convertirlo. */
43. void Char_bit_divison(unsigned int numero, unsigned char bit[8])
44. {
45.     int i,j,k=0;
46.
47.     do
48.     {
49.         bit[k++] = numero%2;
50.         numero=numero/2;
51.     }
52.     while (numero>0);
53.
54.     for(j=k; j<8-1; j++)
55.         bit[j]=0;
56.
57.
```

```
58. //Il ciclo for da 7 a 0 stampa i valori binari in ordine inverso
59. printf("\n\t Numero in Base2: ");
60. for(i=8-1; i>=0; i--)
61.     (i%4==0) ? printf(" %1u",bit[i]) : printf("%1u",bit[i]);
62. }
63.
64.
65. /*Function per estrarre i bit di un intero*/
66. void estrai_bit(unsigned int n, unsigned char bit[8])
67. {
68.     short j;
69.     j=8-1;
70.
71.     do
72.     {
73.         bit[j--]=n&1;
74.         n=n>>1;
75.     }
76.     while(n!=0 && j>=0);
77.
78.     if(j>=0)
79.     {
80.         do
81.         {
82.             bit[j--]=0;
83.         }
84.         while (j>=0);
85.     }
86.
87. }
```

## ESECUZIONE 1

```
Inserire numero intero positivo in base 10: 56
Numero in Base2: 001 1100 0
**Utilizzo l'algoritmo estrai_bit per confrontare
il risultato con quello della funzione char_bit_division.**
Conversione con char_bit_division= 0011 1000

Process returned 0 (0x0) execution time : 3.901 s
Press ENTER to continue.
```

## ESECUZIONE 2

```
Inserire numero intero positivo in base 10: 34
Numero in Base2: 001 0001 0
**Utilizzo l'algoritmo estrai_bit per confrontare
il risultato con quello della funzione char_bit_division.**
Conversione con char_bit_division= 0010 0010

Process returned 0 (0x0) execution time : 5.773 s
Press ENTER to continue.
```

## Esercizio 6[liv.1]

### - Conversione da base 2 a base 10

```
1.  /**[liv.1] Scrivere una function C di conversione di un intero positivo da base 2 a base 10
2.   ,
3.   mediante l'algoritmo delle divisioni successive, che generi un array di caratteri contenenti
4.   i
5.   le cifre decimali.*/
6.
7.           /**LIBRERIE E DEFINE*/
8. #include <stdio.h>
9. #include <stdlib.h>
10. #include <math.h>
11.
12.           /**PROTOTIPI*/
13. void conversion(unsigned char bit[32],short len);
14. void array_char(unsigned int numero);
15.
16.           /**MAIN*/
17. int main()
18. {
19.     unsigned char bit[32];
20.     short len, i , j;
21.
22.     printf ("\n Quanti bit vuoi inserire?\n\t ----> ");
23.     scanf("%hd",&len);
24.
25.     //andiamo ad inizializzare i bit da len a 32, cosÃ¬ facendo i restanti bit non inizializzati
26.     //verranno usati per inserire i bit uno alla volta dal meno significativo al piÃ¹ significativo
27.     //esempio:
28.     //se volessimo inserire 4 bit, l'inizializzazione verrà fatta dalla posizione 4 fino a
29.     //lla 32-1
30.     //così' facendo, nel prossimo ciclo for quando andremo ad inserire i bit uno alla volta
31.
32.     //verranno inseriti nelle posizioni non inizializzate
33.     for (i=len;i<32;i++)
34.         bit[i]=0;
35.
36.     printf("\n Immetti il numero in binario che vuoi convertire in decimale \n (una cifra per volta, dal bit meno significativo\n al piu' significativo):\n");
37.
38.     for (j=0;j<len;j++)
39.         scanf("%d",&bit[j]);
40.
41.     conversion(bit,len);
42.
43.             /**FUNCTION*/
44. void array_char(unsigned int numero)
45. {
46.     unsigned int numero2=numero;
47.     char array_cifre[9];
48.     short cifre=0, i,L=0, s;
49.     int divisore;
50.
51.     while (numero2!=0)
52.     {
53.         numero2=numero2/10;
54.         cifre++;
```

```
55.     }
56.
57.     for (i=cifre-1;i>=0;i--)
58.     {
59.         divisore=pow(10,i);
60.         array_cifre[L++]=(numero/divisore);
61.         numero=(numero%divisore);
62.     }
63.
64.     printf ("\nArray di caratteri: ");
65.     for (s=0; s<cifre; s++)
66.         printf("%d \n\n",array_cifre[s]);
67.
68. }
69.
70. void conversion(unsigned char bit[32], short len)
71. {
72.     unsigned int numero=0;
73.     short i;
74.
75.     for (i=0;i<32;i++)
76.         numero=numero+(bit[i]*pow(2,i));
77.
78.     printf("\nNumero decimale: %d",numero);
79.     array_char(numero);
80. }
```

## ESECUZIONE 1

```
Quanti bit vuoi inserire?  
----> 4
```

```
Immetti il numero in binario che vuoi convertire in decimale  
(una cifra per volta, dal bit meno significativo  
al piu' significativo):
```

```
0  
1  
0  
0
```

```
Numero decimale: 2  
Array di caratteri: 2
```

```
Process returned 0 (0x0)    execution time : 4.962 s  
Press ENTER to continue.
```

## ESECUZIONE 2

```
Quanti bit vuoi inserire?  
----> 8
```

```
Immetti il numero in binario che vuoi convertire in decimale  
(una cifra per volta, dal bit meno significativo  
al piu' significativo):
```

```
1  
0  
1  
0  
1  
1  
0  
0
```

```
Numero decimale: 53  
Array di caratteri: 53
```

```
Process returned 0 (0x0)    execution time : 9.752 s  
Press ENTER to continue.
```

## Esercizio 7 [liv.1]

## - Conversione da base 2 a base 10(input stringa di bit)

58. }

## ESECUZIONE 1

```
*****  
Inserire numero di bit del valore da convertire: 4  
*****
```

```
Inserire stringa binaria [0-1]: 0010
```

```
Sequenza di bit inserita : 0010
```

```
Array di caratteri generato: 2
```

```
Process returned 0 (0x0) execution time : 14.174 s  
Press ENTER to continue.
```

## ESECUZIONE 2

```
*****  
Inserire numero di bit del valore da convertire: 6  
*****
```

```
Inserire stringa binaria [0-1]: 011010
```

```
Sequenza di bit inserita : 011010
```

```
Array di caratteri generato: 26
```

```
Process returned 0 (0x0) execution time : 10.857 s  
Press ENTER to continue.
```

## Esercizio 8[liv.1]

### - Addizione aritmetica binaria

```
1.  /***[liv.1] Scrivere una function C per eseguire l'addizione aritmetica binaria di due numeri naturali p e q (p,qâ^N)
2. mediante gli operatori bitwise (come da algoritmo in P-like in esercizio 11) , traducendo l'algoritmo di seguito riportato:
3. {Algoritmo di "addizione binaria" mediante operatori sui bit}
4.     procedure binary_add(op1,op2)
5.         rip:=1;
6.         while rip>0
7.             sum:=bitXOR(op1,op2);
8.             rip:=bitAND(op1,op2);
9.             rip:=leftSHIFT(rip,1);
10.            op1:=sum; op2:=rip;
11.            endwhile
12.        */
13.
14.                /**LIBRERIE*/
15. #include <stdio.h>
16. #include <stdlib.h>
17. #include <math.h>
18.
19. #define n_len    16
20.
21.                 /**PROTOTIPI*/
22. int add_bin(short p, short q);
23. void bit_short(short n, unsigned char bit[n_len]);
24.
25.                 /**MAIN*/
26. int main()
27. {
28.     int p,q,k;
29.     unsigned char bit[n_len];
30.
31.     printf("\n\t*****\n\t**");
32.     printf("\n\t**");
33.     printf("\n\t** Addizione aritmetica binaria **");
34.     printf("\n\t**");
35.     printf("\n\t*****\n\n");
36.
37.     printf("\n Inserisci primo numero naturale (p): ");
38.     fflush(stdin);
39.     scanf("%d",&p);
40.
41.     printf(" Inserisci secondo numero naturale (q): ");
42.     fflush(stdin);
43.     scanf("%d",&q);
44.
45.     bit_short(p,bit);
46.     printf("\n\n\t p= ");
47.     for(k=0; k<n_len; k++)
48.         (k%4==0)?printf(" %1u",bit[k]):printf("%1u",bit[k]);
49.     printf(" +");
50.
51.     bit_short(q,bit);
52.     printf("\n\t q= ");
53.     for(k=0; k<n_len; k++)
54.         (k%4==0)?printf(" %1u",bit[k]):printf("%1u",bit[k]);
55.     printf(" =");
56.
57.     printf("\n*+* Addizione aritmetica binaria tra i due numeri: %d *+\n\n", add_bin(p,q));
58. }
```

```

59.     return 0;
60. }
61.
62.         /**FUNCTION*/
63. /*Function per addizione aritmetica binaria*/
64. int add_bin(short p, short q)
65. {
66.     short somma=0,riporto=1,k;
67.     unsigned char bit[n_len];
68.
69.     while(riporto>0)//finche il riporto e' maggiore di 0....
70.     {
71.         somma=p^q; //...andreamo ad eseguire un'operazione di bitXOR(1 se i due operandi son
o diversi,altrimenti e' 0)
72.         riporto=p&q; //...andreamo ad eseguire un'operazione di bitAND(1 se i due operandi v
algono 1,altrimenti e' 0)
73.         riporto=riporto<<1;//e andremo a shiftare di 1 posizione verso sinistra(<<1=moltip
icazione per 2)
74.         p=somma;//aggioreremo somma
75.         q=riporto;//e riporto
76.     }
77.
78.     bit_short(somma,bit);
79.     printf("\n\t ----- \n\t ");
80.     for(k=0; k<n_len; k++)
81.         (k%4==0)?printf(" %iu",bit[k]):printf("%iu",bit[k]);
82.     printf("\n");
83.
84.     return somma;
85. }
86.
87. /*Function per estrarre i bit di uno short*/
88. void bit_short(short n, unsigned char bit[n_len])
89. {
90.     short j;
91.     j=n_len-1;
92.
93.     do
94.     {
95.         bit[j--]=n&1;
96.         n=n>>1;
97.     }
98.     while(n!=0 && j>=0);
99.
100.    if(j>=0)
101.    {
102.        do
103.        {
104.            bit[j--]=0;
105.        }
106.        while(j>=0);
107.    }
108.
109. }

```

## ESECUZIONE 1

```
*****
**          **
**  Addizione aritmetica binaria  **
**          **
*****
```

```
Inserisci primo numero naturale (p): 4
Inserisci secondo numero naturale (q): 6
```

```
p= 0000 0000 0000 0100 +
q= 0000 0000 0000 0110 =
-----
0000 0000 0000 1010
```

```
*** Addizione aritmetica binaria tra i due numeri: 10 ***
```

```
Process returned 0 (0x0)    execution time : 6.305 s
Press ENTER to continue.
```

## ESECUZIONE 2

```
*****
**          **
**  Addizione aritmetica binaria  **
**          **
*****
```

```
Inserisci primo numero naturale (p): 44
Inserisci secondo numero naturale (q): 66
```

```
p= 0000 0000 0010 1100 +
q= 0000 0000 0100 0010 =
-----
0000 0000 0110 1110
```

```
*** Addizione aritmetica binaria tra i due numeri: 110 ***
```

```
Process returned 0 (0x0)    execution time : 9.999 s
Press ENTER to continue.
```

## Esercizio 9[liv.1]

### - Sottrazione aritmetica binaria

```
1.  /**[liv.1] Scrivere una function C per eseguire la sottrazione aritmetica* binaria (*: cioe
   ' primo operando maggiore del secondo)
2.  di due numeri naturali p e q (p, q, p-q^N) mediante gli operatori bitwise.
3.  **/
4.          /**LIBRERIE**/
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.  #include <math.h>
8.
9.  #define n_len    16
10.
11.         /**PROTOTIPI**/
12. int sott_bin(short p, short q);
13. void bit_short(short n, unsigned char bit[n_len]);
14.
15.         /**MAIN**/
16. int main()
17. {
18.     int p,q,k;
19.     unsigned char bit[n_len];
20.
21.     printf("\n\t*****");
22.     printf("\n\t**");
23.     printf("\n\t** Sottrazione aritmetica binaria **");
24.     printf("\n\t**");
25.     printf("\n\t*****\n\n");
26.
27.     printf("\n Inserisci primo numero naturale (p): ");
28.     fflush(stdin);
29.     scanf("%d",&p);
30.
31.     do
32.     {
33.         printf(" Inserisci secondo numero naturale (q): ");
34.         fflush(stdin);
35.         scanf("%d",&q);
36.         if(p<q)
37.             printf("\n ERRORE INSERIMENTO..il secondo numero deve essere minore del primo!!\n\n");
38.     }while(p<q);
39.
40.     bit_short(p,bit);
41.     printf("\n\n\t p= ");
42.     for(k=0; k<n_len; k++)
43.         (k%4==0)?printf(" %1u",bit[k]):printf("%1u",bit[k]);
44.     printf(" -");
45.
46.     bit_short(q,bit);
47.     printf("\n\t q= ");
48.     for(k=0; k<n_len; k++)
49.         (k%4==0)?printf(" %1u",bit[k]):printf("%1u",bit[k]);
50.     printf(" =");
51.
52.     printf("\n*- Sottrazione aritmetica binaria tra i due numeri: %d *-\n\n", sott_bin(p,q));
53.
54.     return 0;
55. }
56.
57.         /**FUNCTION**/
58. /*Function per sottrazione aritmetica binaria*/
```

```

59. int sott_bin(short p, short q)
60. {
61.     short sottrai=0,prestito=1,k;
62.     unsigned char bit[n_len];
63.
64.     while(prestito>0)
65.     {
66.         sottrai=p^q;
67.         prestito=~p&q;
68.         prestito=prestito<<1;
69.         p=sottrai;
70.         q=prestito;
71.     }
72.
73.     bit_short(sottrai,bit);
74.     printf("\n\t ----- \n\t ");
75.     for(k=0; k<n_len; k++)
76.         (k%4==0)?printf(" %1u",bit[k]):printf("%1u",bit[k]);
77.     printf("\n");
78.
79.     return sottrai;
80. }
81.
82. /*Function per estrarre i bit di uno short*/
83. void bit_short(short n, unsigned char bit[n_len])
84. {
85.     short j;
86.     j=n_len-1;
87.
88.     do
89.     {
90.         bit[j--]=n&1;
91.         n=n>>1;
92.     }
93.     while(n!=0 && j>=0);
94.
95.     if(j>=0)
96.     {
97.         do
98.         {
99.             bit[j--]=0;
100.        }
101.        while(j>=0);
102.    }
103.
104. }

```

## ESECUZIONE 1

```
*****  
**  
** Sottrazione aritmetica binaria **  
**  
*****
```

```
Inserisci primo numero naturale (p): 8  
Inserisci secondo numero naturale (q): 3
```

```
p= 0000 0000 0000 1000 -  
q= 0000 0000 0000 0011 =  
-----  
0000 0000 0000 0101
```

```
** Sottrazione aritmetica binaria tra i due numeri: 5 **
```

```
Process returned 0 (0x0) execution time : 15.660 s  
Press ENTER to continue.
```

## ESECUZIONE 2

```
*****  
**  
** Sottrazione aritmetica binaria **  
**  
*****
```

```
Inserisci primo numero naturale (p): 10  
Inserisci secondo numero naturale (q): 11
```

```
ERRORE INSERIMENTO..il secondo numero deve essere minore del primo!!
```

```
Inserisci secondo numero naturale (q): 10
```

```
p= 0000 0000 0000 1010 -  
q= 0000 0000 0000 1010 =  
-----  
0000 0000 0000 0000
```

```
** Sottrazione aritmetica binaria tra i due numeri: 0 **
```

```
Process returned 0 (0x0) execution time : 16.171 s  
Press ENTER to continue.  
■
```

## Esercizio 10[liv.1]

### - Calcolo per complemento a 2 e per eccesso-B

```
1.  /**[liv.1] Scrivere una function C che, fissato il numero n di bit, calcoli la rappresentazione di un intero:  
2.  - per complemento a 2;  
3.  - eccesso B (B-biased).  
4.  **/  
5.  
6.          /**LIBRERIE**/  
7. #include <stdio.h>  
8. #include <stdlib.h>  
9. #include <math.h>  
10.  
11. #define n_len    32  
12.  
13.          /**PROTOTIPI**/  
14. void comp2(int k, int n);  
15. void Biased(int k, int n);  
16. void bit_int(int k, unsigned char bit[n_len]);  
17.  
18.  
19.          /**MAIN**/  
20. int main()  
21. {  
22.     int n, k, range1, range2;  
23.  
24.     printf("\nInserire il numero di bit per la rappresentazione del numero intero(MAX 32 bit):> ");  
25.     fflush(stdin);  
26.     scanf("%d", &n);  
27.  
28.     range1=-(pow(2,n-1));  
29.     range2=pow(2,n-1)-1;  
30.     printf("\nRange complemento a 2 con %d bit\t[ -2^(n-1) , 2^(n-1)-1 ]:\t%d %d", n, range1, range2);  
31.  
32.     range1=-(pow(2,n-1))+1;  
33.     range2=pow(2,n-1);  
34.     printf("\nRange eccesso B con %d bit\t[ -(2^n-1)-1 , 2^n-1 ]:\t\t%d %d", n, range1, range2);  
35.  
36.     printf("\n\nInserire il numero intero decimale: ");  
37.     fflush(stdin);  
38.     scanf("%d", &k);  
39.  
40.     printf("\n\n");  
41.     comp2(k,n);  
42.     Biased(k,n);  
43.  
44.     printf("\n\n");  
45.     return 0;  
46. }  
47.  
48.  
49.          /**FUNCTION**/  
50. /*Function per complemento a 2*/  
51. void comp2(int k, int n)  
52. {  
53.     int r1, r2, i, pow_2;  
54.     unsigned char bit[n_len];  
55.     pow_2=pow(2,n);  
56.
```

```

57.     //La rappresentazione per complemento a 2(rc2) su n_bit di un intero(rappresentabile) k
      che siste in Z, si esprime con la formula: rc2(k)=[2^(n)+k]mod(2^(n))
58.     if(k>=0)
59.     {
60.         //intero decimale positivo = per un intero +k>=0 (k=0,1,...,2^n-1) ----
61.         > (+k)=[2^n+(+k)]%2^n = k
62.         r1=(pow_2+k)%pow_2;
63.         printf("\nRappresentazione per complemento a 2 di un numero intero positivo: %d ",r
64.             1);
65.         bit_int(r1,bit);
66.         printf("\nIn binario= ");
67.         for(i=0; i<n_len; i++)
68.             (i%4==0)?printf(" %1d",bit[i]):printf("%1d",bit[i]);
69.     }
70.     else if(k<0)
71.     {
72.         //intero decimale negativo = per un intero -k<0 (-k=-(2^n-1),..., -1) ----> (-
73.         k)=[2^n+(-k)]%2^n = 2^n-k
74.         r2=(pow_2+(-k))%pow_2;
75.         printf("\nRappresentazione per complemento a 2 di un numero intero negativo: %d ",r
76.             2);
77.         bit_int(r2,bit);
78.         printf("\nIn binario= ");
79.         for(i=0; i<n_len; i++)
80.             (i%4==0)?printf(" %1d",bit[i]):printf("%1d",bit[i]);
81.     }
82. }
83.
84.
85. /*Function eccesso Biased*/
86. void Biased(int k, int n)
87. {
88.     int biased, eccessoB, i;
89.     unsigned char bit[n_len];
90.
91.     //La rappresentazione biased con bias B di un intero(rappresentabile) k su n_bit, rb(k)
92.     , puo' esprimersi con la seguente formula: B=2^(n-1)-1
93.     biased=(pow(2,n-1)-1);
94.     eccessoB=k+biased;
95.
96.     printf("\nRappresentazione per eccesso B-biased di un numero intero: %d ",eccessoB);
97.
98.     bit_int(eccessoB,bit);
99.     printf("\nIn binario= ");
100.    for(i=0; i<n_len; i++)
101.        (i%4==0)?printf(" %1d",bit[i]):printf("%1d",bit[i]);
102.
103. }
104.
105.
106. /*Function per estrarre i bit di uno short*/
107. void bit_int(int k, unsigned char bit[n_len])
108. {
109.     int j;
110.     j=n_len-1;
111.
112.     do
113.     {
114.         bit[j--]=k&1;
115.         k=k>>1;
116.     }

```

```
117.     while(k!=0 && j>=0);
118.
119.     if(j>=0)
120.     {
121.         do
122.         {
123.             bit[j--]=0;
124.         }
125.         while(j>=0);
126.     }
127.
128. }
```

## ESECUZIONE 1

```
*****
**          **
**  Addizione aritmetica binaria  **
**          **
*****
```

```
Inserisci primo numero naturale (p): 4
Inserisci secondo numero naturale (q): 6
```

```
p= 0000 0000 0000 0100 +
q= 0000 0000 0000 0110 =
-----
0000 0000 0000 1010
```

```
*** Addizione aritmetica binaria tra i due numeri: 10 ***
```

```
Process returned 0 (0x0)    execution time : 6.305 s
Press ENTER to continue.
```

## ESECUZIONE 2

```
*****
**          **
**  Addizione aritmetica binaria  **
**          **
*****
```

```
Inserisci primo numero naturale (p): 44
Inserisci secondo numero naturale (q): 66
```

```
p= 0000 0000 0010 1100 +
q= 0000 0000 0100 0010 =
-----
0000 0000 0110 1110
```

```
*** Addizione aritmetica binaria tra i due numeri: 110 ***
```

```
Process returned 0 (0x0)    execution time : 9.999 s
Press ENTER to continue.
```

## Esercizio 11 [liv.1]

### - Addizione algebrica binaria mediante bitwise

```
1.  /**[liv.1] Conoscendo la rappresentazione degli interi in C, riscrivere la function C
2.  per l'addizione aritmetica binaria di due interi mediante gli operatori bitwise (vedi eserc
   izio 8.)
3.  nel caso gli operandi siano interi con segno.
4.
5.  Se l'operazione da implementare deve essere l'addizione algebrica
6.  (cioe' deve valere anche per gli interi negativi rappresentati per complemento a 2),
7.  quale accorgimento va usato nella traduzione in C dell'algoritmo e perche'.*/
8.
9. /*RISPOSTA:
10. In complemento a 2, l'addizione algebrica puo' essere sostituita da una semplice addizione
    in
11. aritmetica modulare. In partica e' possibile usare lo stesso algoritmo di addizione aritmet
    ica
12. per addizionare o sottrarre interi, purche' quest'addizione venga eseguita modulo  $2^n$ .
13.
14. Se sommiamo la rappresentazione per complemento a 2 di un numero K positivo e la
15. rappresentazione del complemento a 2 del suo opposto -
    K, ci accorgiamo che la somma di questi
16. complementi a 2 da  $2^n$  che, rappresentato su n bit e' 0, perche' la divisione di  $2^n$  per se
    stesso
17. da resto 0. Quindi possiamo dire che il complemento a 2 di -
    k si comporta come l'opposto di k
18. nell'aritmetica modulare, questo ci consente di utilizzare lo stesso algoritmo di addizione
19. aritmetica per addizionare o sottrarre interi. Cioe' l'addizione algebrica con la rappresen
    tazione
20. per complemento a 2 si riduce ad un'addizione aritmetica, purche' questa addizione venga
21. eseguita in aritmetica modulo  $2^n$ 
22.
23. ESEMPIO:
24. Vediamo nel caso n=4, vogliamo calcolare 7-5. Andando a scrivere le rappresentazioni per
25. complemento a due su 4 bit (0111 + 1011). Eseguiamo l'addizione binaria, otteniamo un risul
    tato
26. a 5 bit (non 4), ma poiche' il risultato deve essere ricondotto a 4 bit dobbiamo togliere tu
    tto
27. cioe' che esce fuori dai 4 bit e alla fine ci troviamo come risultato 0010 che e' la
28. rappresentazione per complemento a 2 del valore 2 che e' proprio il risultato di 7-5 */
29.
30.          /**LIBRERIE*/
31. #include <stdio.h>
32. #include <stdlib.h>
33. #include <math.h>
34.
35. #define n_len    16
36.
37.
38.          /**PROTOTIPI*/
39. int add_bin(short p, short q);
40. void bit_short(short n, unsigned char bit[n_len]);
41.
42.          /**MAIN*/
43. int main()
44. {
45.     int p,q,k;
46.     unsigned char bit[n_len];
47.
48.     printf("\n\t*****");
49.     printf("\n\t**      **");
50.     printf("\n\t**  Addizione aritmetica binaria  **");
51.     printf("\n\t**      **");
```

```

52.     printf("\n\t*****\n");
53.
54.     printf("\n Inserisci primo numero naturale (p): ");
55.     fflush(stdin);
56.     scanf("%d",&p);
57.
58.     printf(" Inserisci secondo numero naturale (q): ");
59.     fflush(stdin);
60.     scanf("%d",&q);
61.
62.     bit_short(p,bit);
63.     printf("\n\t p= ");
64.     for(k=0; k<n_len; k++)
65.         (k%4==0)?printf(" %1u",bit[k]):printf("%1u",bit[k]);
66.     printf(" +");
67.
68.     bit_short(q,bit);
69.     printf("\n\t q= ");
70.     for(k=0; k<n_len; k++)
71.         (k%4==0)?printf(" %1u",bit[k]):printf("%1u",bit[k]);
72.     printf(" =");
73.
74.     printf("\n*+* Addizione aritmetica binaria tra i due numeri: %d *+\n", add_bin(p,q));
    );
75.
76.     return 0;
77. }
78.
79.
80.         /**FUNCTION*/
81. /*Function per addizione aritmetica binaria*/
82. int add_bin(short p, short q)
83. {
84.     short somma,riporto=1,k;
85.     unsigned char bit[n_len];
86.
87.     while(riporto!=0)
88.     {
89.         somma=p^q;
90.         riporto=p&q;
91.         riporto=riporto<<1;
92.         p=somma;
93.         q=riporto;
94.     }
95.
96.     bit_short(somma,bit);
97.     printf("\n\t ----- \n\t   ");
98.     for(k=0; k<n_len; k++)
99.         (k%4==0)?printf(" %1u",bit[k]):printf("%1u",bit[k]);
100.    printf("\n");
101.
102.    return somma;
103. }
104.
105.
106. /*Function per estrarre i bit di uno short*/
107. void bit_short(short n, unsigned char bit[n_len])
108. {
109.     short j;
110.     j=n_len-1;
111.
112.     do
113.     {
114.         bit[j--]=n&1;
115.         n=n>>1;
116.     }

```

```
117.     while(n!=0 && j>=0);
118.
119.     if(j>=0)
120.     {
121.         do
122.         {
123.             bit[j--]=0;
124.         }
125.         while(j>=0);
126.     }
127.
128. }
```

## ESECUZIONE 1

```
*****  
**  
** Addizione aritmetica binaria **  
**  
*****
```

```
Inserisci primo numero naturale (p): 2  
Inserisci secondo numero naturale (q): -3
```

```
p= 0000 0000 0000 0010 +  
q= 1111 1111 1111 1101 =  
-----  
1111 1111 1111 1111
```

```
*** Addizione aritmetica binaria tra i due numeri: -1 ***
```

```
Process returned 0 (0x0)    execution time : 8.506 s  
Press ENTER to continue.  
■
```

## Esercizio 12[liv.1]

### - Rappresentazione binaria(s,e,m) di un numero float

```
1.  /***[liv.1] Scrivere una function C per visualizzare la rappresentazione binaria (s,e,m) di
   un numero float.
2.  Verificare che il valore del numero ottenuto coincida con il dato iniziale.*/
3.
4.          /**LIBRERIE*/
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.  #include <math.h>
8.
9.          /**STRUCT E UNION*/
10.         typedef struct
11.         {
12.             unsigned int m:23; //bit - significativi
13.             unsigned int e:8;
14.             unsigned int s:1; //bit + significativi
15.         }DEC_SEM;
16.
17.         typedef union
18.         {
19.             DEC_SEM ia;
20.             int Ia;
21.             float fa;
22.         }SEM;
23.         /**PROTOTIPI*/
24. void estrae_bit(int reg, unsigned char B[32]);
25.
26.         /**MAINI*/
27. int main()
28. {
29.     SEM num;
30.     int Bias=(1<<7)-1; //per float n=8: 2^(n-1)-1
31.     unsigned char i,bit[32];
32.
33.     puts("\n\n Un float e' rappresentato da 32 bit cosi' divisi secondo lo standard IEEE 75
   4");
34.     puts(" 1 bit per il segno");
35.     puts(" 8 bit per l'esponente");
36.     puts(" 23 bit per la mantissa\n");
37.
38.     puts(" Il bit segno e' 0 se il numero e' positivo, 1 se e' negativo");
39.     puts(" L'esponente si rappresenta come un BIASED quindi vale da 0 a 255\n");
40.     puts(" Ricordarsi che un bit della mantissa e' implicito \n e che qui i numeri sono nor
   malizzati");
41.
42.     printf("\n Digita il numero float da rappresentare: ");
43.     scanf("%f",&num.fa);
44.
45.     estrae_bit(num.Ia,bit);
46.     printf("\n Float= %e ",num.fa);
47.
48.     puts("\n Bit corrispondenti      Segno      Esponente      Mantissa");
49.     printf("\n\t\t\t\t %d\t\t",bit[0]);
50.
51.     for(i=1; i<=8; i++)
52.         printf("%d",bit[i]);
53.         printf("\t ");
54.     for(i=9; i<32; i++)
55.         printf("%d",bit[i]);
56.
57.     printf("\n");
58.     printf("\n Float x= %g      [float ==> Bias= %d] \n",num.fa,Bias);
```

```

59.     printf(" Valore decimale del campo segno di x= %d\n",num.ia.s);
60.     printf(" Valore decimale del campo esponente di x= %d (biased) = %d (unbiased)\n",num.i
   a.e,num.ia.e-Bias);
61.     printf(" Valore decimale del campo mantissa di x= %d (intero) = %g (frazionario)\n",num
   .ia.m,num.ia.m/((float)(1<<23)));
62.     printf("\n");
63.
64.     return 0;
65. }
66.
67.         /**FUNCTION*/
68. void estrae_bit(int reg, unsigned char B[32])
69. {
70.     /*rappresentazione binaria dell'int reg nell'array B
71.      bit +sign. <----- bit -sign.
72.      B[0] B[1] B[2] .... B[30] B[31]*/
73.     short i;
74.
75.     for(i=31; i>=0; i--)
76.     {
77.         B[i]=(char)(1®); /* estra bit - sign.*/
78.         reg=reg>>1;
79.     }
80. }
```

## ESECUZIONE 1

Un float e' rappresentato da 32 bit cosi' divisi secondo lo standard IEEE 754  
1 bit per il segno  
8 bit per l'esponente  
23 bit per la mantissa

Il bit segno e' 0 se il numero e' positivo, 1 se e' negativo  
L'esponente si rappresenta come un BIASED quindi vale da 0 a 255

Ricordarsi che un bit della mantissa e' implicito  
e che qui i numeri sono normalizzati

Digita il numero float da rappresentare: 6.78

Float= 6.780000e+00  
Bit corrispondenti      Segno      Esponente      Mantissa  
                          0                10000001                1011000111010111000011

Float x= 6.78      [float ==> Bias= 127]  
Valore decimale del campo segno di x= 0  
Valore decimale del campo esponente di x= 129 (biased) = 2 (unbiased)  
Valore decimale del campo mantissa di x= 5830083 (intero) = 0.695 (frazionario)

Process returned 0 (0x0)    execution time : 39.312 s  
Press ENTER to continue.

## ESECUZIONE 2

Un float e' rappresentato da 32 bit cosi' divisi secondo lo standard IEEE 754  
1 bit per il segno  
8 bit per l'esponente  
23 bit per la mantissa

Il bit segno e' 0 se il numero e' positivo, 1 se e' negativo  
L'esponente si rappresenta come un BIASED quindi vale da 0 a 255

Ricordarsi che un bit della mantissa e' implicito  
e che qui i numeri sono normalizzati

Digita il numero float da rappresentare: -8.34

Float= -8.340000e+00  
Bit corrispondenti      Segno      Esponente      Mantissa  
                          1                10000010                00001010111000010100100

Float x= -8.34      [float ==> Bias= 127]  
Valore decimale del campo segno di x= 1  
Valore decimale del campo esponente di x= 130 (biased) = 3 (unbiased)  
Valore decimale del campo mantissa di x= 356516 (intero) = 0.0425 (frazionario)

Process returned 0 (0x0)    execution time : 8.795 s  
Press ENTER to continue.

## Esercizio 13[liv.3]

### - Conversione da base 10 alla rappresentazione floating-point IEEE Std 754

```
1.  /***[liv.3] Scrivere una function C di conversione di un numero reale da base 10 alla rappre-
   sentazione floating-point IEEE Std 754.
2.  L'input e' una stringa di char del tipo [+-
   ]X.Y dove X e Y rappresentano rispettivamente la parte intera e la parte frazionaria del nu-
   mero.*/
3.
4.
5.          /**LIBRERIE*/
6. #include <stdio.h>
7. #include <stdlib.h>
8. #include <string.h>
9. #include <math.h>
10. #include <ctype.h>
11.
12. #define MAX_LEN 32
13.
14. typedef enum{vero,falso} Logico;
15.
16. union s_p{
17.     float fa;
18.     int la;
19.     char C[4];
20. }a;
21.
22.
23.          /**PROTOTIPI*/
24. void converti_float(int len, unsigned char bit[MAX_LEN]);
25. float parte_Interia(char *input);
26. float parte_Reale(char *input);
27.
28.
29.          /**MAIN*/
30. int main()
31. {
32.     char input[1000], T[]={".",},*p;
33.     short j=0,i=0;
34.     Logico esito=falso;
35.     float numero=0.0;
36.     unsigned char bit[MAX_LEN];
37.     while(esito==1)
38.     {
39.         printf("\nInserisci numero reale;\nEsempio \"-12.35\" -> ");
40.         fflush(stdin); scanf("%s",input);
41.         // MI RICAVO IL SEGNO DEL NUMERO
42.         if(input[0]=='-' || input[0]=='+')
43.             j++;
44.
45.         while(isdigit(input[j])!=0 || input[j]=='.')
46.             j++;
47.
48.         if(j==strlen(input))
49.             esito=vero;
50.         else{
51.             printf("INPUT ERROR!\a");
52.         }
53.         j=0;
54.     }
55.     // DIVIDO IN DUE PARTI, LA STRINGA IMMESSA COME INPUT; COSA CHE POSSA CALCOLARE LA PA-
      RTE INTERA E QUELLA REALE.
```

```

56.     p=strtok(input,T);
57.     // CHIAMATA ALLA FUNZIONE PER IL CALCOLO DELLA PARTE INTERA DEL NUMERO. LA PRIMA PARTE
58.     // DELLA STRINGA PRECEDENTEMENTE DIVISA.
59.     numero=parte_Inter(a);
60.     // CHIAMATA ALLA FUNZIONE PER IL CALCOLO DELLA PARTE REALE DEL NUMERO. LA RESTANTE PAR-
61.     // TE DI STRINGA.
62.     while((p=strtok(NULL,T))!=NULL)
63.         numero=numero+parte_Reale(p);
64.     // CONTROLLANDO IL PRIMO BIT DELLA STRINGA IMMESSA IN INPUT CONTROLLO IL SEGNO DEL NUM-
65.     // ERO
66.     if(input[0]=='-')
67.     {
68.         numero=numero*-1;
69.         printf("Segno: (-)\t\tNumero: %.3f",numero);
70.     }
71.     else
72.         printf("Segno: (+)\t\tNumero: %.3f",numero);
73.     // ORA CONVERTO IL NUMERO IN BINARIO E LO VISUALIZZO
74.     a.fa=numero;
75.     converti_float(a.la,bit);
76.     printf("\nBinario:\ts\te\tm\n");
77.     printf("          \t%1d\t",bit[0]);
78.     for(i=1;i<9;i++)
79.         printf("%1d",bit[i]);
80.     printf("\t");
81.     for(i=9;i<32;i++)
82.         printf("%1d",bit[i]);
83.     return 0;
84. }
85.             /**FUNCTION**/
86. // FUNZIONE PER LA CONVERSIONE DI UN NUMERO FLOAT DA BASE 10 A BASE 2.
87. void converti_float(int len, unsigned char bit[MAX_LEN])
88. {
89.     short i;
90.     // SCORRO TUTTE LE POSIZIONI DELL'ARRAY CONTENENTE I BIT, E LO INIZIALIZZO CON I BIT D-
91.     // ELLA VARIABILE "len".
92.     for(i=31;i>=0;i--)
93.     {
94.         bit[i]=(char)(1&len);
95.         len=len>>1;
96.     }
97. float parte_Inter(char *input)
98. {
99.     short j=0,i=strlen(input)-1,cifre_i[strlen(input)],len=0;
100.    int numero=0;
101.    while(i>0)
102.    {
103.        cifre_i[len++]=input[i]-
104.        48; // CONVERTO I CARATTERI IN CIFRE E LI MEMORIZZO NELL'ARRAY.
105.    }
106.    // FORMULA PER IL CALCOLO DELLA PARTE INTERA DEL NUMERO IMMESSO.
107.    for(j=0;j<len;j++)
108.        numero=numero+cifre_i[j]*pow(10,j);
109.        printf("%hd\n", (short)numero);
110.    return (float)numero;
111. }
112. }
113. float parte_Reale(char *input)
114. {
115.     short j=0,cifre_r[strlen(input)],len=0;

```

```
117.     float numero=0.0;
118.     while(j<strlen(input))
119.     {
120.         cifre_r[len++]=input[j]-
48; // CONVERTO I CARATTERI IN CIFRE E LI MEMORIZZO NELL'ARRAY.
121.         j--;
122.     }
123.     // FORMULA PER IL CALCOLO DELLA PARTE REALE DEL NUMERO IMMESSO.
124.     for(j=0;j<len;j++)
125.         numero=numero+cifre_r[j]/pow(10,j+1);
126.
127.     return numero;
128. }
```

## ESECUZIONE 1

```
Inserisci numero reale;  
Esempio "-12.35" -> -12.35  
12  
Segno: (-)           Numero: -12.300  
Binario:      s     e         m  
              1 10000010 10001001100110011001101  
Process returned 0 (0x0)   execution time : 4.799 s  
Press ENTER to continue.
```

## Esercizio 14[liv.1]

### - Calcolo epsilon macchina

```
1.  /**[liv.1] Scrivere delle function C per calcolare rispettivamente l'epsilon macchina del
   tipo float, del tipo double e del tipo long double,
2.    visualizzando ad ogni passo i singoli bit.
3.    Confrontare i risultati ottenuti con i valori delle variabili predefinite FLT_EPSILON, D
   BL_EPSILON e DBL_EPSILON.*/
4.
5.    //Che cos'è l'epsilon macchina:
6.    //l'epsilon di macchina è il più piccolo numero, appartenente a un dato insieme F di
   numeri in virgola mobile,
7.    //diverso in valore assoluto da zero, che sommato all' unità , dà un risultato diverso
   da 1.
8.
9.           /**LIBRERIE*/
10. #include <stdio.h>
11. #include <stdlib.h>
12. #include <float.h>
13.
14.
15. #define MAX_LEN 32
16. #define MAX_LEN_R 64
17. #define bias 127
18.
19. /** - STRUTTURE DATI: -*/
20. union s_p
21. {
22.     float fa;
23.     int la;
24.     char C[4];
25. }a;
26.
27. /** - LISTA PROTOTIPI FUNZIONI UTILIZZATE NEL PROGETTO: -*/
28. float epsilon_float(float eps);
29. double epsilon_double();
30. void converti_float(int len, unsigned char bit[MAX_LEN]);
31. void show_bit_dp(int ch[2]);
32.
33.         /**MAIN*/
34. int main()
35. {
36.     a.fa=1.0f;
37.     /// CALCOLO E VISUALIZZO L'EPSILON MACCHINA PER IL TIPO FLOAT, CONFRONTANDOLA CON L'EPS
   ILON MACCHINA MEMORIZZATA NELLA LIBRERIA "float.h".
38.     printf("Tipo Float\nEpsilon: %e\tFLT_EPSILON: %e\n",epsilon_float(a.fa),FLT_EPSILON);

39.     getchar();
40.
41.     /// CALCOLO E VISUALIZZO L'EPSILON MACCHINA PER IL TIPO DOUBLE, CONFRONTANDOLA CON L'EP
   SILON MACCHINA MEMORIZZATA NELL LIBRERIA "float.h"..
42.     printf("Tipo Double\nEpsilon: %e\tDBL_EPSILON: %e\n",epsilon_double(),DBL_EPSILON);
43.     return 0;
44. }
45.
46.         /**FUNCTION*/
47. /*! - FUNCTION epsilon_float() -*/
48. float epsilon_float(float eps)
49. {
50.     unsigned char bit[MAX_LEN];
51.     float eps1=eps+1;
52.     int i=0,k;
53.     printf("Step\tEsponente\tMantissa\n");
54.     while(eps1>1)
```

```

55.     {
56.         a.fa=eps1;
57.         converti_float(a.la,bit);
58.         printf("i= %d\t",i);
59.         // STAMPA DEI BIT AD OGNI CICLO.
60.         for(k=1;k<9;k++)
61.             printf("%1d",bit[k]);
62.             printf("\t");
63.             for(k=9;k<MAX_LEN;k++)
64.                 printf("%1d",bit[k]);
65.             eps=eps/2;
66.             i++;
67.             eps1=eps+1;
68.             fflush(stdin); getchar();
69.     }
70.     eps=eps*2.0f;
71.     return eps;
72. }
73.
74. /*! - FUNCTION epsilon_double() -/
75. double epsilon_double()
76. {
77.     double eps=1.0;
78.     int i=0;
79.     union
80.     {
81.         int len[2];
82.         double num;
83.     }emach;
84.     emach.num=eps+1.0;
85.     printf("Step\tS Esponente    Mantissa\n");
86.     while(emach.num>1.0)
87.     {
88.         printf("i= %d\t",i++);
89.         // STAMPA DEI BIT AD OGNI CICLO.
90.         show_bit_dp(emach.len);
91.         eps=eps/2;
92.         emach.num=eps+1.0;
93.         fflush(stdin); getchar();
94.     }
95.
96.     return eps*2.0;
97. }
98.
99. /*! - FUNCTION converti_float() -/
100. void converti_float(int len, unsigned char bit[MAX_LEN])
101. {
102.     short i;
103.     // SCORRO TUTTE LE POSIZIONE DELL'ARRAY CONTENENTE I BIT, E LO INIZIALIZZO CON I BIT D
104.     // ELLA VARIABILE "len".
105.     for(i=31;i>=0;i--)
106.     {
107.         bit[i]=(char)(1&len);
108.         len=len>>1;
109.     }
110.
111. /*! - FUNCTION show_bit_dp() -/
112. void show_bit_dp(int ch[2])
113. {
114.     char j,jc, bit[MAX_LEN_R];
115.     int c;
116.
117.     // ESTRAZIONE BIT 2 BYTE PER VOLTA.
118.     for(jc=0;jc<2;jc++)
119.     {

```

```
120.     c=ch[jc];
121.     for(j=0;j<32;j++)
122.     {
123.         bit[j+32*jc]=c&1;
124.         c=c>>1;
125.     }
126. }
127. /// VISUALIZZAZIONE BIT ESTRATTI.
128. for(j=63;j>=0;j--)
129.     (j%63==0 || j%52==0) ? printf("%d ",bit[j]) : printf("%d",bit[j]);
130.
131.
132. }
```

## ESECUZIONE 1

```
Step    Esponente      Mantissa
i= 0    10000000    00000000000000000000000000000000
i= 1    01111111    10000000000000000000000000000000
i= 2    01111111    01000000000000000000000000000000
i= 3    01111111    00100000000000000000000000000000
i= 4    01111111    00010000000000000000000000000000
i= 5    01111111    00001000000000000000000000000000
i= 6    01111111    00000100000000000000000000000000
i= 7    01111111    00000010000000000000000000000000
i= 8    01111111    00000001000000000000000000000000
i= 9    01111111    00000000100000000000000000000000
i= 10   01111111    00000000010000000000000000000000
i= 11   01111111    00000000001000000000000000000000
i= 12   01111111    00000000000100000000000000000000
i= 13   01111111    00000000000010000000000000000000
i= 14   01111111    00000000000001000000000000000000
i= 15   01111111    00000000000000100000000000000000
i= 16   01111111    00000000000000010000000000000000
i= 17   01111111    00000000000000001000000000000000
i= 18   01111111    00000000000000000100000000000000
i= 19   01111111    00000000000000000010000000000000
i= 20   01111111    00000000000000000001000000000000
i= 21   01111111    00000000000000000000100000000000
i= 22   01111111    00000000000000000000010000000000
i= 23   01111111    00000000000000000000001000000000
Tipo Float
Epsilon: 1.192093e-07          FLT_EPSILON: 1.192093e-07

Step    S Esponente      Mantissa
i= 0    0 10000000000 00000000000000000000000000000000
i= 1    0 01111111111 10000000000000000000000000000000
i= 2    0 01111111111 01000000000000000000000000000000
i= 3    0 01111111111 00100000000000000000000000000000
i= 4    0 01111111111 00010000000000000000000000000000
i= 5    0 01111111111 00001000000000000000000000000000
i= 6    0 01111111111 00000100000000000000000000000000
i= 7    0 01111111111 00000010000000000000000000000000
i= 8    0 01111111111 00000001000000000000000000000000
i= 9    0 01111111111 00000000100000000000000000000000
i= 10   0 01111111111 00000000010000000000000000000000
i= 11   0 01111111111 00000000001000000000000000000000
i= 12   0 01111111111 00000000000100000000000000000000
i= 13   0 01111111111 00000000000010000000000000000000
i= 14   0 01111111111 00000000000001000000000000000000
i= 15   0 01111111111 00000000000000100000000000000000
i= 16   0 01111111111 00000000000000010000000000000000
i= 17   0 01111111111 00000000000000001000000000000000
i= 18   0 01111111111 00000000000000000100000000000000
i= 19   0 01111111111 00000000000000000010000000000000
i= 20   0 01111111111 00000000000000000001000000000000
i= 21   0 01111111111 00000000000000000000100000000000
i= 22   0 01111111111 00000000000000000000010000000000
i= 23   0 01111111111 00000000000000000000001000000000
i= 24   0 01111111111 000000000000000000000001000000000
i= 25   0 01111111111 0000000000000000000000001000000000
i= 26   0 01111111111 0000000000000000000000000100000000
i= 27   0 01111111111 00000000000000000000000000100000000
i= 28   0 01111111111 000000000000000000000000000100000000
i= 29   0 01111111111 000000000000000000000000000010000000
i= 30   0 01111111111 0000000000000000000000000000010000000
i= 31   0 01111111111 0000000000000000000000000000001000000000
i= 32   0 01111111111 00000000000000000000000000000001000000000
i= 33   0 01111111111 000000000000000000000000000000001000000000
i= 34   0 01111111111 0000000000000000000000000000000001000000000
i= 35   0 01111111111 00000000000000000000000000000000001000000000
i= 36   0 01111111111 000000000000000000000000000000000001000000000
i= 37   0 01111111111 0000000000000000000000000000000000001000000000
i= 38   0 01111111111 00000000000000000000000000000000000001000000000
i= 39   0 01111111111 000000000000000000000000000000000000001000000000
i= 40   0 01111111111 0000000000000000000000000000000000000001000000000
i= 41   0 01111111111 00000000000000000000000000000000000000001000000000
i= 42   0 01111111111 000000000000000000000000000000000000000001000000000
i= 43   0 01111111111 0000000000000000000000000000000000000000001000000000
i= 44   0 01111111111 00000000000000000000000000000000000000000001000000000
i= 45   0 01111111111 000000000000000000000000000000000000000000001000000000
i= 46   0 01111111111 0000000000000000000000000000000000000000000001000000000
i= 47   0 01111111111 00000000000000000000000000000000000000000000001000000000
i= 48   0 01111111111 000000000000000000000000000000000000000000000001000000000
i= 49   0 01111111111 0000000000000000000000000000000000000000000000001000000000
i= 50   0 01111111111 00000000000000000000000000000000000000000000000001000000000
i= 51   0 01111111111 000000000000000000000000000000000000000000000000001000000000
i= 52   0 01111111111 0000000000000000000000000000000000000000000000000001000000000
Tipo Double
Epsilon: 2.220446e-16          DBL_EPSILON: 2.220446e-16

Process returned 0 (0x0)  execution time : 17.043 s
Press ENTER to continue.
```

## Esercizio 18[liv.1]

### - Somma addendi raddoppiamento ricorsivo

```
1.  /**[liv.1] Scrivere una function C per calcolare la somma di molti addendi ak dello stesso  
ordine di grandezza N  
2.  
3.      S= for(k=1; k<N; k++) di ak  
4.  
5. mediante algoritmo di somma a blocchi (Pairwise summation algorithm) implementato in versio  
ne iterativa o ricorsiva (a scelta).  
6. Applicare l'algoritmo al seguente particolare problema test di cui e' nota la soluzione (10  
0):  
7.  
8.          N=10^8, ak=10^-  
6, per ogni K=1,...,N implica che K=1,...,N di ak= k=1,...,10^8 di 10^-6 = 100.  
9.  
10. **/  
11.         /**LIBRERIE**/  
12. #include <stdio.h>  
13. #include <stdlib.h>  
14. #include <float.h>  
15. #include <math.h>  
16.  
17.         /**PROTOTIPI**/  
18. float RaddoppiamentoRic(float array[],int primo,int ultimo);  
19.  
20.         /**MAIN**/  
21. int main()  
22. {  
23.     float a[200];  
24.     int n,i,p,u;    // N = numero di addendi da sommare  
25.                 // I = indice  
26.                 // P = indice del primo elemento  
27.                 // U = indice dell'ultimo elemento  
28.  
29.     printf("Inserisci il numero di Addendi da sommare: ");  
30.     scanf("%d",&n);  
31.     p=0;  
32.     u=n-1;  
33.     for (i=0; i<n; i++)  
34.     {  
35.         printf("Digita il %d^ valore da sommare: ", i+1);  
36.         scanf("%f",&a[i]);  
37.     }  
38.  
39.     printf("\nSomma= %f\n\n",RaddoppiamentoRic(a,p,u));  
40.  
41.     return 0;  
42. }  
43.  
44.         /**FUNCTION**/  
45. //La function somma tutti gli elementi dell'array tramite raddoppiamento ricorsivo  
46. float RaddoppiamentoRic(float array[], int primo, int ultimo)  
47. {  
48.     int mediano;  
49.     float somma=0;  
50.  
51.     /* Se da input Ã" stato inserito un solo valore per l'array il controllo if prevede la  
52.        visualizzazione di tale valore da output senza entrare nel processo di somma */  
53.     if(ultimo==primo) return array[primo];  
54.  
55.     /* Istanza banale:  
56.        E' il caso in cui la differenza fra l'ultimo valore dell'indice d'array e ed il primo v  
ale
```

```

57.      1, ovvero quando ci sono due soli elementi nell'array da sommare. Si valo
58.      ta prima la
59.      condizione piu' semplice esclusa la quale si passa all'Istanza Generica */
60.      {
61.          somma=array[primo]+array[ultimo];
62.          return somma;
63.      }
64.
65.      /* Istanza Generica:
66.      E' il caso in cui si effettua la somma attraverso Ricorsione.
67.      La function individua l'indice intermedio fra il primo valore d'indice d'array e l'ulti
mo
68.      Questo per:    1) individuare 2 porzioni d'array di pari lunghezza
69.                      2) sommando gli estremi delle due porzioni di array
70.                      3) ripete queste operazioni finchÃ" non giunge all'Istanza Banale */
71.      else
72.      {
73.          mediano = (primo+ultimo)/2;
74.          somma=RaddoppiamentoRic(array,primo,mediano)+RaddoppiamentoRic(array,mediano+1,ulti
mo);
75.          return somma;
76.      }
77.  }

```

## ESECUZIONE 1

```
Inserisci il numero di Addendi da sommare: 5
Digita il 1^ valore da sommare: 6
Digita il 2^ valore da sommare: 89
Digita il 3^ valore da sommare: 45
Digita il 4^ valore da sommare: 3
Digita il 5^ valore da sommare: 23
```

```
Somma= 166.000000
```

```
Process returned 0 (0x0)    execution time : 10.917 s
Press ENTER to continue.
```

## Esercizio 19[liv.1]

### - Somma mediante criterio di arresto naturale

```
1.  /**[liv.1] Scrivere una function C per calcolare iterativamente la somma
2.  k (che va da)1 ad N x^k/k! (circa) e^x
3.  con il criterio di arresto naturale*/
4.
5.  /*
6.  while (a>=ULP(S))      //ULP(S)=S*Emach
7.  {
8.     a=...
9.     S=S+a;
10. }
11. Il criterio di arresto naturale serve per evitare somme inutili di addendi non significativi
12. rispetto a S.
13.
14. ULP(a) e' definito come il minimo epsilon positivo appartenente al sistema aritmetico floating
15. point tale che a+epsilon > a.
16.
17. Viene trovato 23 come numero di divisioni perche considerando la mantissa della singola
18. precisione (23 bit), dato che i numeri floating point vengono memorizzati in notazione
19. scientifica forma normalizzata a bit implicito, quindi e' come se la mantissa fosse di 24 bit
20. ci vogliono 23 divisioni per 2, ovvero 23 shift a destra per posizionare il bit implicito
21. nell'ultima posizione tale che il numero A e' = all'epsilon macchina quindi da ancora
22. contributo alla somma, il che significa che l'epsilon macchina in realta si puo scrivere
23. come 2^-
24.      23 ma 23 e' proprio t, il numero di bit della mantissa realmente rappresentati. */
24.
25.
26.         /**LIBRERIE*/
27. #include <stdio.h>
28. #include <float.h>
29.
30.         /**MAIN*/
31. int main()
32. {
33.     float a,somma=0.0;
34.     int n=0; //contatore dei cicli
35.
36.     printf("Inserire valore A: "); scanf("%f",&a);
37.
38.     /*FLT_EPSILON lavorando in singola precisione (float) e' il piu piccolo numero float che
39.     sommato ad 1 da contributo alla somma */
40.     while (a>=somma*FLT_EPSILON)
41.     {
42.         n++;
43.         somma = somma + a;
44.         a = a/2;
45.     }
46.
47.     a = a*2; //calcola l'ultimo valore di a che ha dato contributo alla somma
48.
49.     printf("\nNumero di divisioni per 2: %d",n);
50.     printf("\nSomma con criterio di arresto naturale: %10f",somma);
51.     printf("\nUltimo valore di a che ha dato contributo alla somma: %10f\n\n",a);
52.
53.     return 0;
54. }
```

## ESECUZIONE 1

```
Inserisci il numero di Addendi da sommare: 5
```

```
Digita il 1^ valore da sommare: 6
```

```
Digita il 2^ valore da sommare: 89
```

```
Digita il 3^ valore da sommare: 45
```

```
Digita il 4^ valore da sommare: 3
```

```
Digita il 5^ valore da sommare: 23
```

```
Somma= 166.000000
```

```
Process returned 0 (0x0)    execution time : 10.917 s
```

```
Press ENTER to continue.
```

## Esercizio 20[liv.1]

### - Somma di addendi ordinati

```
1.  /**[liv.1] Scrivere una function C per calcolare la somma di addendi ordinati; sommare gli
   addendi una volta in
2.  ordine crescente ed un'altra in ordine decrescente. Mostrare qual Ã" il modo migliore di so-
   mmare in questo caso*/
3.          /**LIBRERIE**/
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <math.h>
7. #include <float.h>
8.
9.          /**PROTOTIPI**/
10. float errore_assoluto(float, float); // calcolo l'errore assoluto
11. float errore_relativo(float, float); // calcolo l'errore relativo
12. float valore_assoluto(float); // restituisco il valore assoluto del parametro formale
13.
14. void somma_valori(int);
15. /* in questa funzione calcolo contemporaneamente la somma in ordine crescente e decrescente
16. calcolando anche il relativo errore relativo. Come si potra' notare dalla stampa dei risultati
17. i valori ottenuti denotano che sommando i termini in ordine decrescente si ottiene un errore
18. relativo costante a differenza della somma in ordine crescente dove il risultato dell'errore
19. relativo e' di massima accuratezza*/
20. /*Si vede che quando sommiamo i termini in ordine decrescente l'errore e' sempre piu grande
21. di
22. quello ottenuto sommando i termini in ordine crescente, addirittura risulta costante.
23. Il problema e' dovuto al fatto che, da una certa iterazione in poi, l'ennesimo termine della
24. successione avra avuto valore inferiore all'ULP dell'attuale somma parziale, non portando
25. dunque alcun contributo al risultato che invece avrebbe dovuto essere diverso. */
26.          /**MAIN**/
27. int main()
28. {
29.     somma_valori(5000);
30.     somma_valori(10000);
31.     somma_valori(20000);
32.     somma_valori(50000);
33.     return 0;
34. }
35.
36.          /**FUNCTION**/
37. float errore_assoluto(float calcolato, float esatto)
38. {
39.     return valore_assoluto(calcolato-esatto);
40. }
41.
42.
43. float errore_relativo(float calcolato, float esatto)
44. {
45.     return valore_assoluto(calcolato-esatto)/valore_assoluto(calcolato);
46. }
47.
48.
49. void somma_valori(int n)
50. {
51.     int i, indice_decrescente=n;
52.     float val_cre=0.0, val_dec=0.0, risultato_esatto=(float)(pow(3.1415926535,2)/6), Er;
```

```
54.     for (i=1; i<=n; i++)
55.     {
56.         val_cre = val_cre + (1/((float)(pow(i,2))));
57.         val_dec = val_dec + (1/((float)(pow(indice_decrescente,2))));
58.         indice_decrescente--;
59.     }
60.
61.     Er=errore_relativo(val_cre, risultato_esatto);
62.     printf("Per N=%d\n",n);
63.     printf("L'errore relativo per la somma in ordine crescente e':\t\t%8.4e\n",Er);
64.
65.     Er=errore_relativo(val_dec, risultato_esatto);
66.     printf("L'errore relativo per la somma in ordine decrescente e':\t%8.4e\n\n",Er);
67. }
68.
69.
70. float valore_assoluto(float n)
71. {
72.     if (n<0) return -n;
73.     else return n;
74. }
```

## ESECUZIONE 1

Per N=5000

L'errore relativo per la somma in ordine crescente e':	1.2691e-04
L'errore relativo per la somma in ordine decrescente e':	1.2162e-04

Per N=10000

L'errore relativo per la somma in ordine crescente e':	1.2691e-04
L'errore relativo per la somma in ordine decrescente e':	6.0806e-05

Per N=20000

L'errore relativo per la somma in ordine crescente e':	1.2691e-04
L'errore relativo per la somma in ordine decrescente e':	3.0366e-05

Per N=50000

L'errore relativo per la somma in ordine crescente e':	1.2691e-04
L'errore relativo per la somma in ordine decrescente e':	1.2175e-05

Process returned 0 (0x0) execution time : 0.013 s  
Press ENTER to continue.

## Esercizio 22[liv.1]

### - Quiz

```
1. /**[Quiz]: Che differenza c'Ã tra i due codici C che seguono*/
2.
3. #include <stdio.h>
4. #include <string.h>
5.
6. /*1*/
7. int main()
8. {
9.     char stringa[] = "ciao";
10.    puts(stringa);
11.    *stringa='m';
12.    puts(stringa);
13.
14.    return 0;
15. }
16.
17. /*2*/
18. int main()
19. {
20.     char *stringa="ciao";
21.     puts(stringa);
22.     *stringa='m';
23.     puts(stringa);
24.
25.     return 0;
26. }
27.
28.
29. /**RISPOSTA
30. La differenza sostanziale tra i due codici, viene vista subito
31. nell'inizializzazione della variabile stringa, in quanto nel
32. primo codice stringa Ã dichiarato come un vettore e sappiamo
33. che i caratteri presenti possono essere modificati come gli
34. elementi di un vettore, infatti nel primo codice ciao diventa miao.
35. Nel secondo codice, dove troviamo la versione puntatore anzichÃ
36. vettore, la variabile stringa Ã di tipo puntatore e quindi punta
37. a una stringa letterale, quindi non modificabile, per questo ciao
38. non diventa miao come nel primo codice C.**/
```

## ESECUZIONE 1

(codice 1)

```
ciao  
miao
```

```
Process returned 0 (0x0)    execution time : 0.006 s  
Press ENTER to continue.
```

## ESECUZIONE 2

(codice 2)

```
ciao
```

```
Process returned -1 (0xFFFFFFFF)    execution time : 0.044 s  
Press ENTER to continue.
```

## Esercizio 23a[liv.1]

### - Lettura di n caratteri uno alla volta(statica)

```
1. /***[liv.1] Confrontando i risultati con quelli delle relative funzioni del C
   ed utilizzando per le stringhe
2. o l'allocazione statica
3. o l'allocazione dinamica
4. scrivere una function C che accetti in input il numero n e legga da tastiera
   n caratteri (uno alla volta)
5. costruendo la stringa che li contiene (parametro di output), senza usare str
   cat(...).*/
6.
7.
8. ****ALLOCAZIONE STATICA*****
9.
10.
11.          /**LIBRERIE**/
12. #include <stdio.h>
13. #include <stdlib.h> //libreria che comprende le funzioni malloc,calloc,realloc,free
14. #include <string.h>
15.
16. #define len      50 //lunghezza dell'array
17.
18.          /**PROTOTIPI**/
19. void static_string(int n, char stringa_a[]);
20.
21.          /**MAIN**/
22. int main()
23. {
24.     int n=len;
25.     char stringa_a[n];
26.     printf("\n");
27.
28.     static_string(n,stringa_a);
29.     printf("\n\n");
30.
31.     return 0;
32. }
33.
34.          /**FUNCTION**/
35. void static_string(int n, char stringa_a[])
36. {
37.     int c,j=0;
38.
39.     printf("\n Immetti stringa (max %d): ", n);
40.     do
41.     {
42.         c=getchar(); // legge un carattere (non richiede Invio)
43.         stringa_a[j]=(char)c; //salva i caratteri in stringa_a
44.         j++;
45.     } while (c != '\n'); //fino a non premiamo invio, ovvero '\n', inseris
   ci caratteri nell'array
46.
47.     stringa_a[j-1]='\0'; /* toglie '\n'*/
48.
49.     printf("\n La stringa %s e' lunga "
50.           "%ld caratteri", stringa_a, strlen(stringa_a));
51.     printf("\n Stringa imposta= %s", stringa_a);
52. }
```

## ESECUZIONE 1

```
Immetti stringa (max 50): Ci son due coccodrilli e un orangotango
```

```
La stringa Ci son due coccodrilli e un orangotango e' lunga 39 caratteri  
Stringa immessa= Ci son due coccodrilli e un orangotango
```

```
Process returned 0 (0x0)    execution time : 25.347 s  
Press ENTER to continue.
```

## Esercizio 23b[liv.1]

### - Lettura di n caratteri uno alla volta(dinamica)

```
1.  /***[liv.1] Confrontando i risultati con quelli delle relative funzioni del C ed utilizzando
2.  per le stringhe
3.  o l'allocazione statica
4.  o l'allocazione dinamica
5.  scrivere una function C che accetti in input il numero n e legga da tastiera n caratteri (u
6.  no alla volta)
7.  costruendo la stringa che li contiene (parametro di output), senza usare strcat(...).*/
8.
9.
10.
11.          /**LIBRERIE**/
12. #include <stdio.h>
13. #include <stdlib.h> //libreria che comprende le funzioni malloc,calloc,realloc,free
14. #include <string.h>
15.
16.          /**PROTOTIPI**/
17. void dynamic_string(int n, char *stringa_a);
18.
19.          /**MAIN**/
20. int main()
21. {
22.     int n=50;
23.     char *stringa_a;
24.
25.     printf("\n");
26.     stringa_a=(char*)malloc(n+1); //alloca dinamicamente con malloc
27.
28.     if(stringa_a != NULL) //controlla se la stringa Ã" stata allocata correttamente
29.     {
30.         dynamic_string(n,stringa_a);
31.         printf("\n\n");
32.     }
33.     free(stringa_a); //dealloca
34.
35.     return 0;
36. }
37.
38.
39.          /**FUNCTION**/
40. void dynamic_string(int n, char *stringa_a)
41. {
42.     int c,j=0;
43.
44.     printf("\n Immetti stringa (max %d): ", n);
45.     do
46.     {
47.         c=getchar(); // legge un carattere (non richiede Invio)
48.         *(stringa_a+j)=(char)c;
49.         j++;
50.     } while (c != '\n');
51.
52.     *(stringa_a+j-1]='\0'; /* toglie '\n'*/
53.
54.     printf("\n La stringa %s e' lunga "
55.             "%ld caratteri", stringa_a, strlen(stringa_a));
56.     printf("\n Stringa immessa= %s", stringa_a);
57. }
```

## ESECUZIONE 1

```
Immetti stringa (max 50): Ci son due coccodrilli e un orangotango
```

```
La stringa Ci son due coccodrilli e un orangotango e' lunga 39 caratteri  
Stringa immessa= Ci son due coccodrilli e un orangotango
```

```
Process returned 0 (0x0)    execution time : 13.790 s  
Press ENTER to continue.  
=
```

## Esercizio 24a[liv.1]

### - Concatenazione di due stringhe(statica)

```
1.  /***[liv.1] Confrontando i risultati con quelli delle relative funzioni del C ed utilizzando
   per le stringhe
2.  o l'allocazione statica
3.  o l'allocazione dinamica
4.  scrivere una function C che restituisca la concatenazione di due stringhe (parametri di input)
   ut) senza usare strcat(...).
5.  E' a scelta restituire la concatenazione delle due stringhe in una terza variabile (parametro di output o function stessa)
6.  oppure nella prima delle due variabili di input.*/
7.
8.
9.  ****ALLOCAZIONE STATICAGA*****
10.
11.
12.          **LIBRERIE**
13. #include <stdio.h>
14. #include <stdlib.h> //libreria che comprende le funzioni malloc,calloc,realloc,free
15. #include <string.h>
16.
17. #define len1 103
18.
19.          **PROTOTIPI**
20. void stringa_concatenata(char str1[], char str2[], char strconc[len1]);
21.
22.          **MAIN**
23. int main()
24. {
25.     char str1[51],str2[51];
26.     char strconc[len1];
27.
28.     printf("\n Inserire prima stringa (max 50): ");
29.     fflush(stdin);
30.     gets(str1);
31.
32.     printf(" Inserire seconda stringa (max 50): ");
33.     fflush(stdin);
34.     gets(str2);
35.     printf("\n");
36.
37.     stringa_concatenata(str1,str2,strconc);
38.     printf("\n\n");
39.
40.     return 0;
41. }
42.
43.
44.          **FUNCTION**
45. void stringa_concatenata(char str1[], char str2[], char strconc[])
46. {
47.     int i,j=0;//j contatore
48.     int l1,l2;
49.
50.     l1=strlen(str1);
51.     l2=strlen(str2);
52.
53.     for(i=0; i<l1; i++)
54.     {
55.         strconc[i]=str1[i]; // inserisc
56.     }
57.
58.     for(i=l1; i<l1+l2; i++)
```

```
59.     {
60.         strconc[i]=str2[j];
61.         j++;
62.     }
63.
64.     printf("\n Stringhe concatenate (senza strcat): %s", strconc);
65.
66. }
```

## ESECUZIONE 1

```
Inserire prima stringa (max 50): Nel mezzo del cammin di nostra vita  
Inserire seconda stringa (max 50): mi ritrovai per una selva oscura
```

```
Stringhe concatenate (senza strcat): Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura
```

```
Process returned 0 (0x0) execution time : 17.617 s  
Press ENTER to continue.
```

## Esercizio 24b[liv.1]

### - Concatenazione di due stringhe(dinamica)

```
1.  /***[liv.1] Confrontando i risultati con quelli delle relative funzioni del C ed utilizzando
2.  per le stringhe
3.  o l'allocazione statica
4.  o l'allocazione dinamica
5.  scrivere una function C che restituisca la concatenazione di due stringhe (parametri di input)
6.  ut) senza usare strcat(...).
7.  E' a scelta restituire la concatenazione delle due stringhe in una terza variabile (parametro di output o function stessa)
8.  oppure nella prima delle due variabili di input.*/
9.
10.
11.
12.          /**LIBRERIE**/
13. #include <stdio.h>
14. #include <stdlib.h> //libreria che comprende le funzioni malloc,calloc,realloc,free
15. #include <string.h>
16.
17. #define len 103
18.
19.          /**PROTOTIPI**/
20. void stringa_concatenata(char *str1, char *str2, char *strconc);
21.
22.          /**MAIN**/
23. int main()
24. {
25.     char *str1,*str2;
26.     char *strconc;
27.
28.     str1=(char*)malloc(50+1); //allocazione dinamica con maggiore dell'array
29.     str2=(char*)malloc(50+1); //
30.     strconc=(char*)malloc(len+1); //
31.
32.     if(str1 == NULL)
33.         printf("\n Error ");
34.
35.     if(str2 == NULL)
36.         printf("\n Error ");
37.
38.     if(strconc == NULL)
39.         printf("\n Error ");
40.
41.     printf("\n Inserire prima stringa (max 50): ");
42.     fflush(stdin);
43.     gets(str1);
44.
45.     printf(" Inserire seconda stringa (max 50): ");
46.     fflush(stdin);
47.     gets(str2);
48.     printf("\n");
49.
50.     stringa_concatenata(str1,str2,strconc);
51.     printf("\n\n");
52.
53.     return 0;
54. }
55.
56.
57.          /**FUNCTION**/
58. void stringa_concatenata(char *str1, char *str2, char *strconc)
```

```
59. {
60.     int i,j=0;//j contatore
61.     int l1,l2;
62.
63.     l1=strlen(str1);
64.     l2=strlen(str2);
65.
66.     for(i=0; i<l1; i++)
67.     {
68.         *(strconc+i)=*(str1+i); //inserisco la prima stringa nell'array strconc, str1+i equivale a str1[i]
69.     }
70.
71.     for(i=l1; i<l1+l2; i++)
72.     {
73.         *(strconc+i)=*(str2+j); //inserisco la seconda stringa nell'array strconc, str2+i equivale a str2[i]
74.         j++;
75.     }
76.
77.     printf("\n Stringhe concatenate (senza strcat): %s", strconc);
78.
79. }
```

## ESECUZIONE 1

```
Inserire prima stringa (max 50): Nel mezzo del cammin di nostra vita  
Inserire seconda stringa (max 50): mi ritrovai per una selva oscura
```

```
Stringhe concatenate (senza strcat): Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura
```

```
Process returned 0 (0x0) execution time : 17.617 s  
Press ENTER to continue.
```

## Esercizio 25[liv.1]

### - Primo occorrenza di una sottostringa in una stringa

```
1.  /**[liv.1] Confrontando i risultati con quelli delle relative funzioni del C
   , scrivere una function C che restituisca la
2. prima occorrenza di una sottostringa in una stringa senza usare strstr(...).
 */
3.
4.          /**LIBRERIE**/
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <string.h>
8.
9.
10.         /**PROTOTIPI**/
11. void prima_occorrnza(char *pattern, char *testo);
12.
13.         /**MAIN**/
14. int main()
15. {
16.     char testo[100], pattern[20];
17.
18.     printf("\n Inserisci testo (max 100): ");
19.     fflush(stdin);
20.     gets(testo);
21.
22.     printf(" Inserisci sottostringa (max 20): ");
23.     fflush(stdin);
24.     gets(pattern);
25.     printf("\n");
26.
27.     prima_occorrnza(pattern,testo);
28.
29.     return 0;
30. }
31.
32.         /**FUNCTION**/
33. void prima_occorrnza(char *pattern, char *testo)
34. {
35.     int m,n,i=0, prima_occ=0;
36.     short conta_chiave=0;
37.
38.     n=strlen(pattern);
39.     m=strlen(testo);
40.
41.     while(testo[i]!='\0' && conta_chiave==0)
42.     {
43.         if(strncmp(pattern,&testo[i],n)==0)
44.         {
45.             conta_chiave=1;
46.             prima_occ=i;
47.         }
48.         else i++;
49.     }
50.     printf("\n La prima occorrenza della sottostringa '%s' si trova nella po
sizione %d \n\n",pattern, prima_occ);
51. }
```

## ESECUZIONE 1

Inserisci testo (max 100): Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura, che la dritta via era smarrita  
Inserisci sottostringa (max 20): ra

La prima occorrenza della sottostringa 'ra' si trova nella posizione 28

```
Process returned 0 (0x0) execution time : 38.267 s
Press ENTER to continue.
```

## Esercizio 26[liv.1-2]

### - Numero totale di occorrenze

```
1.  /**[liv.2]Usando l'allocazione dinamica e le funzioni C per manipolare le stringhe, scrivere  
e una function C che  
2.  restituisca il numero totale delle occorrenze di una sottocorrenza in una stringa e  
3.  [liv.1] ne visualizzi la posizione di tutte le occorrenze trovate.  
4.  [liv.2] restituisca in un array la posizione di tutte le occorrenze trovate.**/  
5.  
6.          /**LIBRERIE**/  
7.  #include <stdio.h>  
8.  #include <stdlib.h>  
9.  #include <string.h>  
10.  
11.  
12.         /**PROTOTIPI**/  
13. void string_matching(char *pattern, char *testo, char occorrenze[]);  
14.  
15.         /**MAIN**/  
16. int main()  
17. {  
18.     int l1,l2;  
19.     char *testo, *pattern, *occorrenze;  
20.  
21.     testo=(char*)malloc(150);  
22.     pattern=(char*)malloc(50);  
23.  
24.     if(testo && pattern == NULL)  
25.         printf("\n Error ");  
26.  
27.     printf("\n Inserisci testo: ");  
28.     fflush(stdin);  
29.     gets(testo);  
30.  
31.     printf(" Inserisci sottocorrenza: ");  
32.     fflush(stdin);  
33.     gets(pattern);  
34.     printf("\n");  
35.  
36.     l1=strlen(testo);  
37.     l2=strlen(pattern);  
38.  
39.     testo=(char*)realloc(testo,l1+1);  
40.     pattern=(char*)realloc(pattern,l2+1);  
41.     occorrenze=(char*)calloc(l1,sizeof(char));  
42.  
43.     if(occorrenze == NULL)  
44.         printf("\n Error ");  
45.  
46.     string_matching(pattern,testo,occorrenze);  
47.  
48.     return 0;  
49. }  
50.  
51.         /**FUNCTION**/  
52. void string_matching(char *pattern, char *testo, char occorrenze[])  
53. {  
54.     int conta_chiave=0,l1,l2,int_pos=-1,i;  
55.     char *n,*offset;  
56.  
57.     l1=strlen(testo);  
58.     l2=strlen(pattern);  
59.  
60.     offset=(char*)malloc(l1+1);
```

```
61.     strcpy(offset,testo);
62.
63.     while((n=strstr(offset,pattern)) != NULL)
64.     {
65.         int_pos+=strcspn(offset,pattern)+12;
66.         conta_chiave++;
67.         printf(" Occorrenza: %d \t Pos: %d\n",conta_chiave,int_pos);
68.         offset=&n[12];
69.     }
70.     printf("\n L'Occorrenza '%s' appare '%d' volte nel testo '%s'\n\n",pattern,conta_chiave
,testo);
71. }
```

## ESECUZIONE 1

```
Inserisci testo: Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura che la dritta via era smarrita  
Inserisci sottostringa: ra
```

```
Occorrenza: 1 Pos: 16  
Occorrenza: 2 Pos: 22  
Occorrenza: 3 Pos: 30
```

```
L'Occorrenza 'ra' appare '3' volte nel testo 'Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura che la dritta via era smarrita'
```

```
Process returned 0 (0x0) execution time : 28.194 s  
Press ENTER to continue.  
■
```

## Esercizio 27a[liv.2]

### - Eliminazione di tutte le occorrenze(statica)

```
1.  /***[liv.2] Utilizzando per le stringhe
2.  o l'allocazione statica
3.  o l'allocazione dinamica
4.  scrivere una function C che elimini tutte le occorrenze di una data sottostringa in una str
   inga col minimo
5.  numero di spostamenti di blocchi di memoria.*/
6.
7.
8.  ****ALLOCAZIONE STATICA*****
9.
10.  **LIBRERIE*/
11. #include <stdio.h>
12. #include <stdlib.h>
13. #include <string.h>
14.
15.  **PROTOTIPI*/
16. void statica();
17. void eliminazione(char *, char *, short , short );
18.
19.  **MAIN*/
20. int main()
21. {
22.     printf("\n ALLOCAZIONE STATICA\n ELiminazione di tutte le occorrenze di un data sottost
   ringa in un stringa col"
23.                         "minimo numero di spostamenti di blocchi di
   memoria.\n -----> ");
24.
25.     statica();
26.
27. }
28.
29.  **FUNCTION*/
30. void statica()
31. {
32.     char text[100], patt[20]; //dichiaro testo e pattern, cioÃ la stringa e la sottostri
   nga
33.     short len_t, len_p; // dichiaro queste due variabili, che avranno come valore,
   le lunghezze del testo e del patt, date dalla funzione strlen()
34.     //Inserimento da tastiera di testo e pattern da eliminare nel testo col minimo spostame
   nto di blocchi in memoria
35.     printf("\n Inserisci il testo: ");
36.     fflush(stdin);
37.     gets(text);
38.     printf("\n Inserisci il patt: ");
39.     fflush(stdin);
40.     gets(patt);
41.     //ricavo con la funzione strlen() la lunghezza delle due stringhe
42.     len_t=strlen(text);
43.     len_p=strlen(patt);
44.     //chiamo la function per eliminare la sottostringa(patt) nella stringa(text), come rich
   iesto dalla traccia
45.     eliminazione(text, patt, len_t, len_p);
46. }
47.
48. void eliminazione(char *text, char *patt, short len_t, short len_p)
49. {
50.     short i;
51.     for(i=len_t-len_p;i>=0;i--)
52.     {
53.         if(strncmp(text+i, patt, len_p)==0)
54.         {
```

```
55.         memmove(text+i, text+i+len_p, len_t-len_p-i+1);
56.         len_t-=len_p;
57.     }
58. }
59. printf("\n %s\n\n",text);
60. }
```

## ESECUZIONE 1

```
ALLOCAZIONE STATICÀ
Eliminazione di tutte le occorrenze di un data sottostringa in un stringa colminimo numero di spostamenti di blocchi di memoria.
-----
warning: this program uses gets(), which is unsafe.
Inserisci il testo: Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura che la dritta via era smarrita
Inserisci il patt: ra
Nel mezzo del cammin di nost vita mi ritrovai per una selva oscu che la dritta via e smarrita

Process returned 0 (0x0)   execution time : 32.393 s
Press ENTER to continue.
```

## Esercizio 27b[liv.2]

### - Eliminazione di tutte le occorrenze(dinamica)

```
1.  /**[liv.2] Utilizzando per le stringhe
2.  o l'allocazione statica
3.  o l'allocazione dinamica
4.  scrivere una function C che elimini tutte le occorrenze di una data sottostringa in una str
   inga col minimo
5.  numero di spostamenti di blocchi di memoria.*/
6.
7.  ****ALLOCAZIONE DINAMICA*****
8.
9.          **LIBRERIE**
10. #include <stdio.h>
11. #include <stdlib.h>
12. #include <string.h>
13.
14.          **PROTOTIPI**
15. void dinamica();
16. void eliminazione(char *, char *, short , short );
17.
18.          **MAIN**
19. int main()
20. {
21.     printf("\n *ALLOCAZIONE DINAMICA*\n ELiminazione di tutte le occorrenze di un data sott
   ostringa in un stringa\n"
22.             " col minimo numero di spostamenti di blocchi di me
   moria.\n\n");
23.
24.     dinamica();
25.
26.     return 0;
27. }
28.          **FUNCTION**
29. void dinamica()
30. {
31.     char *text, *patt;           //dichiaro testo e pattern, cioÃ la stringa e la sottostri
   nga
32.     short len_t, len_p;         // dichiaro queste due variabili, che avranno come valore,
   le lunghezze del testo e del patt, date dalla funzione strlen()
33.     //Inserimento da tastiera di testo e pattern da eliminare nel testo col minimo spostame
   nto di blocchi in memoria
34.     text=(char *)malloc(100);
35.     patt=(char *)malloc(20);
36.     printf("\n Inserisci il testo: ");
37.     fflush(stdin);
38.     gets(text);
39.     printf("\n Inserisci il patt: ");
40.     fflush(stdin);
41.     gets(patt);
42.     //ricavo con la funzione strlen() la lunghezza delle due stringhe
43.     len_t=strlen(text);
44.     len_p=strlen(patt);
45.     //realloco in memoria per evitare sprechi
46.     text=(char *)realloc(text, len_t+1);
47.     patt=(char *)realloc(patt, len_p+1);
48.     //chiamo la function per eliminare la sottostringa(patt) nella stringa(text), come rich
   iesto dalla traccia
49.     eliminazione(text, patt, len_t, len_p);
50. }
51.
52. void eliminazione(char *text, char *patt, short len_t, short len_p)
53. {
54.     short i;
```

```
55.     for(i=len_t-len_p;i>=0;i--)
56.     {
57.         if(strncmp(text+i, patt, len_p)==0)
58.         {
59.             //La funzione "memmove(dest,src,num_byte)" , sposta il blocco di "num_byte" dal
59.             //La funzione "memmove(dest,src,num_byte)" , sposta il blocco di "num_byte" dal
60.             memmove(text+i, text+i+len_p, len_t-len_p-i+1);
61.             len_t-=len_p;
62.         }
63.     }
64.     printf("\n Testo dopo l'eliminazione del pattern: %s\n\n",text);
65. }
```

## ESECUZIONE 1

```
*ALLOCAZIONE DINAMICA*
Eliminazione di tutte le occorrenze di un data sottostringa in un stringa
col minimo numero di spostamenti di blocchi di memoria.
```

```
warning: this program uses gets(), which is unsafe.
Inserisci il testo: Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura che la dritta via era smarrita
```

```
Inserisci il patt: ra
```

```
Testo dopo l'eliminazione del pattern: Nel mezzo del cammin di nost vita mi ritrovai per una selva oscu che la dritta via e smarrita
```

```
Process returned 0 (0x0)   execution time : 19.290 s
Press ENTER to continue.
...
```

## Esercizio 29a[liv.1]

### - Visualizzazione per colonne A(mxN)(statica)

```
1.  /***[liv.1] A partire dalla matrice A(mxN), del tipo sotto indicato, allocata per righe
2.  o staticamente
3.  o dinamicamente
4.  visualizzarne gli elementi per colonne:
5.
6.          | 11  12  13  14  15  16 |
7.          |                   |
8.          | 21  22  23  24  25  26 |
9.  A(4x6)= |                   |
10.         | 31  32  33  34  35  36 |
11.         |                   |
12.         | 41  42  43  44  45  46 |
13.
14.
15. Gli elementi a[ij] della matrice sono tali che le unita' indicate la colonna e le decine indicate la riga cui l'elemento appartiene.*/
16.
17.
18. ****ALLOCAZIONE STATICAG*****/
19.
20.           **LIBRERIE**/
21. #include <stdio.h>
22. #include <stdlib.h>
23.
24. #define      RIGA       4
25. #define      COLONNA    6
26.
27.           **PROTOTIPI**/
28. void stampamatrice(int A[][COLONNA]);
29.
30.           **MAIN**/
31. int main()
32. {
33.     int A[RIGA][COLONNA]={{11,12,13,14,15,16},
34.                           {21,22,23,24,25,26},
35.                           {31,32,33,34,35,36},
36.                           {41,42,43,44,45,46}};
37.
38.     printf("\n \t\t Matrice statica:\n");
39.
40.     stampamatrice(A);
41.
42.     return 0;
43. }
44.
45.           **FUNCTION**/
46. void stampamatrice(int A[][COLONNA])
47. {
48.     int i,j;
49.
50.     for(j=1;j<COLONNA;j++)
51.     {
52.         printf("\t\t |");
53.         for(i=0; i<RIGA; i++)//---|
54.             printf("----|");
55.             printf("\n");
56.
57.         printf("\t\t |");
58.         for(i=0; i<RIGA; i++)
59.             printf(" %d |",A[i][j]);//stampa la matrice
60.             printf("\n");
```

```
61.         {
62.             printf("\t\t |");
63.             for(i=0; i<RIGA; i++)//---|
64.                 printf("----|");
65.                 printf("\n");
66.         }
67.     }
68.     printf("\n");
69. }
```

## ESECUZIONE 1

Matrice statica:

12	22	32	42
13	23	33	43
14	24	34	44
15	25	35	45
16	26	36	46

```
Process returned 0 (0x0)    execution time : 0.018 s
Press ENTER to continue.
```

## Esercizio 29b[liv.1]

### - Visualizzazione per colonne A(mxnx)(dinamica)

```
1.  /**[liv.1] A partire dalla matrice A(mxnx), del tipo sotto indicato, allocata per righe
2.  o staticamente
3.  o dinamicamente
4.  visualizzarne gli elementi per colonne:
5.
6.          | 11  12  13  14  15  16 |
7.          |                   |
8.          | 21  22  23  24  25  26 |
9.  A(4x6)= |                   |
10.         | 31  32  33  34  35  36 |
11.         |                   |
12.         | 41  42  43  44  45  46 |
13.
14.
15. Gli elementi a[ij] della matrice sono tali che le unita' indicate la colonna e le decine indicate la riga cui l'elemento appartiene.*/
16.
17.
18. ****ALLOCAZIONE DINAMICA*****
19.
20.      **LIBRERIE**
21. #include <stdio.h>
22. #include <stdlib.h>
23.
24. #define    RIGA     4
25. #define    COLONNA   6
26.
27.      **PROTOTIPI**
28. void stampamatrice(int i, int j, int *pa);
29.
30.      **MAIN**
31. int main()
32. {
33.     int *pa,i,j;
34.     int A[RIGA][COLONNA]={ {11,12,13,14,15,16},
35.                           {21,22,23,24,25,26},
36.                           {31,32,33,34,35,36},
37.                           {41,42,43,44,45,46}};
38.
39.     pa=malloc(RIGA*COLONNA*sizeof(int));
40.     if(pa==NULL)
41.     {
42.         printf("\n Error ");
43.         exit(EXIT_FAILURE);
44.     }
45.
46.     for(i=0; i<RIGA; i++)
47.     {
48.         for(j=0; j<COLONNA; j++)
49.         {
50.             *(pa+i*COLONNA+j)=A[i][j];
51.         }
52.     }
53.
54.     printf("\n \t\t Matrice dinamica:\n");
55.
56.     stampamatrice(i,j,pa);
57.
58.     return 0;
59. }
```

```
61.
62.          /**FUNCTION*/
63. void stampamatrice(int i, int j, int *pa)
64. {
65.
66.     for(j=0;j<COLONNA;j++)
67.     {
68.         printf("\t\t |");
69.         for(i=1; i<=RIGA; i++)//---|
70.             printf("----|");
71.             printf("\n");
72.
73.         printf("\t\t |");
74.         for(i=0;i<RIGA;i++)
75.             printf(" %d |",*(pa+i*COLONNA+j));//stampa la matrice
76.             printf("\n");
77.         {
78.             printf("\t\t |");
79.             for(i=1; i<=RIGA; i++)//---|
80.                 printf("----|");
81.                 printf("\n");
82.             }
83.         }
84.         printf("\n");
85.
86. }
```

## ESECUZIONE 1

Matrice dinamica:

11	21	31	41
12	22	32	42
13	23	33	43
14	24	34	44
15	25	35	45
16	26	36	46

```
Process returned 0 (0x0)    execution time : 0.010 s
Press ENTER to continue.
```

## Esercizio 30[liv.1]

### - Prodotto righe per colonne di una matrice

```
1.  /**[liv.1] Scrivere una function C che restituisca la matrice C prodotto righecolonne [ved  
i pdf delle dispense]  
2.  di due matrici rettangolari A e B le cui dimensioni sono stabilite in input  
3.  (usare per tutte le matrici l'allocazione dinamica e generarle come numeri reali random).  
  
4.  C'è qualche preferenza nell'usare malloc() o calloc() rispettivamente per A, B o C?  
5.  Verificare se i tempi di esecuzione, per la sola allocazione e totali, sono gli stessi.**/  
  
6.  
7.  
8.          /**LIBRERIE**/  
9. #include <stdio.h>  
10. #include <stdlib.h>  
11. #include <time.h>  
12.  
13.          /**PROTOTIPI**/  
14. void prodottoAxB(int *A, int *B, int m, int p, int n);  
15.  
16.          /**MAIN**/  
17. int main()  
18. {  
19.     time_t Time_start, Time_finish;  
20.     int elapsed_time;  
21.     int *A,*B;  
22.     int m,p,n; /*m=righe di A - p=colonne di A e righe di B - n=colonne di B*/  
23.     short i,j;  
24.  
25.     printf("\n\t*****\n");  
26.     printf("\n\t* m=righe di A - p=colonne di A e righe di B - n=colonne di B *\n");  
27.     printf("\n\t*****\n\n");  
28.  
29.     printf("\n Inserire numero di righe(m) per A(mxp): ");  
30.     fflush(stdin);  
31.     scanf("%d", &m);  
32.     printf(" Inserire numero di colonne(p) per A(mxp) e righe(p) per B(pxn): ");  
33.     fflush(stdin);  
34.     scanf("%d", &p);  
35.     printf(" Inserire numero di colonne(n) per B(pxn): ");  
36.     fflush(stdin);  
37.     scanf("%d", &n);  
38.  
39.     printf("\n Matrice A(%dx%d) - Matrice B(%dx%d) \n\n",m,p,p,n);  
40.  
41. /**ALLOCAZIONE DINAMICA DELLE MATRICI CON CALLOC E MALLOC E TEMPO DI ESECUZIONE DI UN BLOCC  
O DI ISTRUZIONI**/  
42.  
43.     /**VERIFICO TEMPI DI ESECUZIONE PER LA MATRICE A(mxp) CON CALLOC E MALLOC**/  
44. ****  
45.     /**CALLOC**/  
46.     time(&Time_start); //Tempo iniziale  
47.     printf("\n ora e data:%s", ctime(&Time_start));  
48.     A=(int*)calloc(m*p,sizeof(int));  
49.     time(&Time_finish); //Tempo finale  
50.     if(A==NULL)  
51.     {  
52.         fprintf(stderr, "\n ***Allocazione Fallita***\n");  
53.         exit(EXIT_FAILURE);  
54.     }  
55.     else  
56.     {
```

```

57.         printf("\n ***Allocazione di int A(%dx%d): OK!***\n ", m,p);
58.         elapsed_time=difftime(Time_finish,Time_start);
59.     }
60.     printf("allocazione con calloc() richiede %10d secondi\n",elapsed_time);
61.     free(A);
62.
63.     /**MALLOC*/
64.     time(&Time_start); //Tempo iniziale
65.     printf("\n ora e data:%s", ctime(&Time_start));
66.     A=(int*)malloc(m*p*sizeof(int));
67.     time(&Time_finish); //Tempo finale
68.     if(A==NULL)
69.     {
70.         fprintf(stderr,"\n ***Allocazione Fallita***\n");
71.         exit(EXIT_FAILURE);
72.     }
73.     else
74.     {
75.         printf(" ***Allocazione di int A(%dx%d): OK!***\n ", m,p);
76.         elapsed_time=difftime(Time_finish,Time_start);
77.     }
78.     printf("allocazione con malloc() richiede %10d secondi\n",elapsed_time);
79. /*****MALLOC*****/
*****CALLOC*****/
80.
81.     /**VERIFICO TEMPI DI ESECUZIONE PER LA MATRICE B(pxn) CON CALLOC E MALLOC*/
82. /*****CALLOC*****/
83.     /**CALLOC*/
84.     time(&Time_start); //Tempo iniziale
85.     printf("\n ora e data:%s", ctime(&Time_start));
86.     B=(int*)calloc(p*n,sizeof(int));
87.     time(&Time_finish); //Tempo finale
88.     if(B==NULL)
89.     {
90.         fprintf(stderr,"\n ***Allocazione Fallita***\n");
91.         exit(EXIT_FAILURE);
92.     }
93.     else
94.     {
95.         printf("\n ***Allocazione di int B(%dx%d): OK!***\n ", p,n);
96.         elapsed_time=difftime(Time_finish,Time_start);
97.     }
98.     printf("allocazione con calloc() richiede %10d secondi\n",elapsed_time);
99.     free(B);
100.
101.    /**MALLOC*/
102.    time(& Time_start); //Tempo iniziale
103.    printf("\n ora e data:%s", ctime(&Time_start));
104.    B=(int*)malloc(p*n*sizeof(int));
105.    time(&Time_finish); //Tempo finale
106.    if(B==NULL)
107.    {
108.        fprintf(stderr,"\n ***Allocazione Fallita***\n");
109.        exit(EXIT_FAILURE);
110.    }
111.    else
112.    {
113.        printf(" ***Allocazione di int B(%dx%d): OK!***\n ", p,n);
114.        elapsed_time=difftime(Time_finish,Time_start);
115.    }
116.    printf("allocazione con malloc() richiede %10d secondi\n",elapsed_time);
117. /*****MALLOC*****/
*****Generamento numeri interi random per le matrici**/
118.
119.    /**Generamento numeri interi random per le matrici**/

```

```

120.     srand((unsigned)time(NULL));
121.     /*inizializzo A(mxp)*/
122.     printf("\nMatrice A(%dx%d) generata con numeri random\n",m,p);
123.     for(i=0; i<m; i++)
124.     {
125.         for(j=0; j<p; j++)
126.         {
127.             *(A+i*p+j)=(int)(rand()%99); //genera numeri random da 0 a 99
128.             printf("%d\t",*(A+i*p+j));
129.         }
130.         printf("\n");
131.     }
132.
133.     /*inizializzo B(pxn)*/
134.     printf("\nMatrice B(%dx%d) generata con numeri random\n",p,n);
135.     for(i=0; i<p; i++)
136.     {
137.         for(j=0; j<n; j++)
138.         {
139.             *(B+i*n+j)=(int)(rand()%99); //genera numeri random da 0 a 99
140.             printf("%d\t",*(B+i*n+j));
141.         }
142.         printf("\n");
143.     }
144.
145.     /*Richiamo function prodotto righexcolonne(AxB)*/
146.     prodottoAxB(A,B,m,p,n);
147.
148.     printf("\n\n");
149.     return 0;
150. }
151.
152.         /**FUNCTION*/
153. void prodottoAxB(int *A, int *B, int m, int p, int n)
154. {
155.     short i,j,k;
156.     int *C;
157.
158.     C=(int*)malloc(m*n*sizeof(int));
159.     if(C==NULL)
160.     {
161.         printf("\n Error ");
162.         exit(EXIT_FAILURE);
163.     }
164.
165.     printf("\nMatrice C(%dx%d) prodotto righe per colonne \n",m,n);
166.     for(i=0; i<m; i++)//righe
167.     {
168.         for(j=0; j<n; j++)//colonne
169.         {
170.             for(k=0; k<p; k++)//sommatoria prodotti moltiplicando ogni elemnto delle righe
di A con ogni elemento delle colonne di B.
171.                 *(C+i*n+j)=*(C+i*n+j)+*(A+i*p+k)*(*(B+k*n+j));
172.                 printf("%d\t",*(C+i*n+j));
173.             }
174.             printf("\n");
175.         }
176.
177. }

```

## ESECUZIONE 1

### (tempi di esecuzione con malloc())

```
*****  
* m=righe di A - p=colonne di A e righe di B - n=colonne di B *  
*****
```

Inserire numero di righe(m) per A(mxpx): 4  
Inserire numero di colonne(p) per A(mxpx) e righe(p) per B(pxn): 3  
Inserire numero di colonne(n) per B(pxn): 3

Matrice A(4x3) – Matrice B(3x3)

ora e data:Tue Jan 26 15:36:33 2016  
\*\*\*Allocazione di int A(4x3): OK!\*\*\*  
allocazione con malloc() richiede 0 secondi

ora e data:Tue Jan 26 15:36:33 2016  
\*\*\*Allocazione di int B(3x3): OK!\*\*\*  
allocazione con malloc() richiede 0 secondi

Matrice A(4x3) generata con numeri random

40	61	4
98	96	19
91	91	55
46	52	30

Matrice B(3x3) generata con numeri random

38	83	73
96	56	16
4	38	63

Matrice C(4x3) prodotto righe per colonne

7392	7544	7644
7468	8190	1197
7612	9702	-1035989591
7512	8652	1890

Process returned 0 (0x0) execution time : 16.573 s  
Press ENTER to continue.

## ESECUZIONE 2

### (tempi di esecuzione con `calloc()`)

```
*****  
* m=righe di A - p=colonne di A e righe di B - n=colonne di B *  
*****
```

```
Inserire numero di righe(m) per A(mxP): 4  
Inserire numero di colonne(p) per A(mxP) e righe(p) per B(pxn): 3  
Inserire numero di colonne(n) per B(pxn): 5
```

```
Matrice A(4x3) - Matrice B(3x5)
```

```
ora e data:Tue Jan 26 15:43:43 2016
```

```
***Allocazione di int A(4x3): OK!***  
allocazione con calloc() richiede 0 secondi
```

```
ora e data:Tue Jan 26 15:43:43 2016
```

```
***Allocazione di int B(3x5): OK!***  
allocazione con calloc() richiede 0 secondi
```

```
Matrice A(4x3) generata con numeri random
```

50	71	20
49	84	71
95	9	80
11	58	87

```
Matrice B(3x5) generata con numeri random
```

64	7	14	62	32
62	68	79	87	44
14	41	12	78	26

```
Matrice C(4x5) prodotto righe per colonne
```

7882	8702	8122	9442	8402
8876	11787	852	5538	1846
9002	12282	960	6240	1835629646
9100	12667	1044	39533	2262

```
Process returned 0 (0x0) execution time : 3.822 s  
Press ENTER to continue.
```

## Esercizio 33[liv.1]

### - Lettura testo mediante variabile buffer

```
1.  /***[liv.1] Scrivere una function C che legga, mediante una variabile buffer di 200char,
2.  un file testo e lo visualizzi sullo schermo 40 char per riga e 25 righe per ogni schermata,
3.  fermandosi finche' non viene premuto un tasto per continuare.*/
4.
5.
6.          /**LIBRERIE**/
7. #include <stdio.h>
8. #include <stdlib.h>
9.
10. #define BUFSIZE 200
11.
12.          /**PROTOTIPI**/
13. void buf_write(char buffer[], FILE *fp);
14.
15.          /**MAIN**/
16. int main()
17. {
18.     char buffer[BUFSIZE];
19.     char *nomefile="/Users/MattiaCapasso/file.txt";
20.     /*Dichiariamo *fp puntatore a file, dove *fp Ã" un puntatore alla struttura contenente
   informazioni
21.     sul file. Serve per indirizzare tutte le operazioni sul file */
22.     FILE *fp;
23.
24.     printf("\n %s e' il path del file testo. \n\n",nomefile);
25.     /*Apertura del file di testo in lettura*/
26.     fp=fopen(nomefile,"r");
27.
28.     /*Controllo per vedere se il file e' stato aperto correttamente, se cosi non fosse,
   restituira' un puntatore nullo(NULL)*/
29.     if(fp==NULL)
30.     {
31.         printf("\n Errore apertura file. ");
32.         exit(EXIT_FAILURE);
33.     }
34.
35.
36.     buf_write(buffer,fp);
37.
38.     return 0;
39. }
40.          /**FUNCTION**/
41. void buf_write(char buffer[], FILE *fp)
42. {
43.     int riga=0,schermo=0,ind_buf,i,j=0;
44.     char ch;//variabile usata per stampare i caratteri singolarmente
45.
46.     while(!feof(fp))
47.     {
48.         ind_buf=0;
49.
50.         while(ind_buf<BUFSIZE && !feof(fp))
51.         {
52.             ch=(char)getc(fp);
53.             buffer[ind_buf]=ch;
54.             ind_buf++;
55.         }
56.
57.         schermo=schermo+ind_buf;
58.
59.         i=0;
```

```

60.         while (i<ind_buf) /*Mentre ci sono ancora file nel buffer da stampare*/
61.         {
62.             while (riga<40 && i<ind_buf)
63.                 /*Mentre non si e' stampati 40 caratteri sulla stessa riga e non sono finiti gli
64.                  elementi del buffer */
65.                 {
66.                     printf("%c",buffer[i]); //Stampa il carattere i-esimo del buffer
67.                     riga++; //Incrementa il contatore per il controllo dei caratteri per riga
68.                     i++; //Incrementa l'indice del buffer
69.                 }
70.             riga=0; /*Azzera il contatore di caratteri di riga */
71.             j++; //Incrementa il contatore di righe
72.             printf(" %d\n",j); //Va alla riga a capo per memorizzare i successivi 40 caratt
73.         eri
73.     }
74.
75.     /*Dopo la stampa del buffer, se sei arrivato alla fine della schermata (hai raggiun
76.      to
76.      cioe 25x40 = 1000 caratteri, posizionati sulla prossima schermata (lascia lo spazio
76.      ) */
77.     if (schermo == 1000)
78.     {
79.         printf("\n\n");
80.         schermo=0; /*azzerà il contatore dei caratteri per schermata*/
81.         j=0; /*azzerà il contatore */
82.     }
83. }
84. fclose(fp);
85. printf("");
86. }

```

## ESECUZIONE 1

/Users/MattiaCapasso/file.txt e' il path del file testo.

Nel mezzo del cammin di nostra vita mi r 1  
itrovai per una selva oscura, che la dir 2  
itta via era smarrita. Ahi quanto a dir 3  
qual era Ė cosa dura esta selva selvagg 4  
ia e aspra e forte che nel pensier rinnov 5  
a la paura! Tant' e amara che poco e piu 6  
morte; ma per trattar del ben ch'i' vi 7  
trovai, diro de l'altre cose ch'i' v'ho 8  
scorte. Io non so ben ridir com' i' v'in 9  
trai, tant' era pien di sonno a quel pun 10  
to che la verace via abbandonai. Ma poi 11  
ch'i' fui al piĒ d'un colle giunto, la 12  
dove terminava quella valle che m'avea d 13  
i paura il cor compunto, guardai in alto 14  
e vidi le sue spalle vestite gi‡ de' 15  
raggi del pianeta che mena dritto altrui 16  
per ogne calle. Allor fu la paura un po 17  
co queta, che nel lago del cor m'era dur 18  
ata la notte ch'i' passai con tanta piet 19  
a. E come quei che con lena affannata, u 20  
scito fuor del pelago a la riva, si volg 21  
e a l'acqua perigiosa e guata, cosi l'a 22  
nimo mio, ch'ancor fuggiva, si volse a r 23  
etro a rimirar lo passo che non lascio g 24  
ia mai persona viva. Poi ch'ei posato un 25

poco il corpo lasso, ripresi via per la 1  
piaggia diserta, si che 'l pie fermo se 2  
mpre era 'l piu basso. Ed ecco, quasi al 3  
cominciar de l'erta, una lonza leggiera 4  
e presta molto, che di pel macolato era 5  
coverta; e non mi si partia dinanzi al 6  
volto, anzi 'mpediva tanto il mio cammin 7  
o, ch'i' fui per ritornar piu volte volt 8  
o. Temp' era dal principio del mattino, 9  
e 'l sol montava 'n su con quelle stelle 10  
ch'eran con lui quando l'amor divino mo 11  
sse di prima quelle cose belle; si ch'a 12  
bene sperar m'era cagione di quella fier 13  
a a la gaetta pelle l'ora del tempo e la 14  
dolce stagione; ma non si che paura non 15  
mi desse la vista che m'apparve d'un le 16  
one. Questi parea che contra me venisse 17  
con la test' alta e con rabbiosa fame, s 18  
i che parea che l'aere ne tremesse. Ed u 19  
na lupa, che di tutte brame sembiava car 20  
ca ne la sua magrezza, e molte genti fe 21  
gia viver grame, questa mi porse tanto d 22  
i gravezza con la paura ch'uscia di sua 23  
vista, ch'io perdei la speranza de l'alt 24  
ezza.? 25

Process returned 0 (0x0) execution time : 0.008 s  
Press ENTER to continue.  
■

## Esercizio 34 [liv.2]

### Lettura, scrittura e aggiornamento file binari

```
1.  /***[liv.2] Scrivere un programma C che crei un file binario studenti1.dat contenente le seguenti informazioni:  
2.  
3.  - cognome e nome (30c) c = char  
4.  - matricola (cccc/ccccc) [es. 0124/002345]  
5.  - numero degli esami superati (short)  
6.  - media pesata degli esami (float)  
7.  - crediti acquisiti (short).  
8.  
9.  Il file contiene le informazini gia' ordinate per matricola. Scrivere una function C che, a partire da un file  
10. di aggiornamento relativo ad un certo esame (per esempio, "esameProg2.dat") contenente gli studenti che l'hanno  
11. superato ed i relativi voti, crei il file "studenti2.dat" aggiornato.**/  
12.  
13.          /**LIBRERIE**/  
14. #include <stdio.h>  
15. #include <stdlib.h>  
16. #include <string.h>  
17.  
18. #define LEN_N_C 30  
19. #define LEN_M 11  
20. #define CLASSE 3  
21. #define PROMOSSI 2  
22.  
23. typedef struct  
24. {  
25.     char nome_cognome[LEN_N_C];  
26.     char matricola[LEN_M];  
27.     short esami;  
28.     float media;  
29.     short cfu;  
30.     short voto;  
31. }studente;  
32.  
33. studente studenti[CLASSE], studenti2[PROMOSSI];  
34.  
35.          /**PROTOTIPI**/  
36. void inserimento_dati(FILE *fp);  
37. void lettura_dati(FILE *fp);  
38. void new_esame(FILE *fp);  
39. float calcola_media(short *j, short *i);  
40. void aggiornamento();  
41. void stampa_aggiornamento(FILE *fp);  
42.  
43.          /**MAIN**/  
44. int main()  
45. {  
46.     FILE *fp;  
47.     //ad ogni funzione passiamo come parametro un puntatore al file che verra' aperto  
48.     //grazie alla funzione fopen(), i cui parametri sono 1)nome del file - 2)la modalita' d  
i apertura del file  
49.     inserimento_dati(fp);  
50.     lettura_dati(fp);  
51.     getchar();  
52.     printf("\n\n");  
53.     new_esame(fp);  
54.     getchar();  
55.     printf("\n\n");  
56.     fp=fopen("studenti1.dat","r+b");  
57.     if((fread(&studenti,sizeof(studente),CLASSE,fp))!=CLASSE)
```

```

58.    {
59.        puts("READ ERROR!\a");
60.        exit(1);
61.    }
62.    fclose(fp);
63.
64.    fp=fopen("esameProg2.dat","r+b");
65.    if((fread(&studenti2,sizeof(studente),PROMOSSI,fp))!=PROMOSSI)
66.    {
67.        puts("READ ERROR!\a");
68.        exit(1);
69.    }
70.    //utilizziamo la funzione fclose, perche' non dobbiamo dimenticare che il file alla fi
ne
71.    //dovra' essere sempre chiuso.
72.    fclose(fp);
73.    aggiornamento();
74.    fp=fopen("Studenti2.dat","w+b");
75.    if((fwrite(&studenti,sizeof(studente),CLASSE,fp))!=CLASSE)
76.    {
77.        puts("WRITE ERROR!\a");
78.        exit(1);
79.    }
80.    fclose(fp);
81.    lettura_dati(fp);
82.    stampa_aggiornamento(fp);
83.    return 0;
84. }
85.
86.         /**FUNCTION**/
87. //Nelle function utilizzeremo molto spesso le operazioni di lettura e scrittura, tramite le
funzioni fread() e fwrite()
88. //tali operazioni avvengono su interi blocchi di dati in formato intero
89. //INPUT: fread(void *buffer, type_i size, type_i count, FILE *fp)
90. //OUTPUT: fwrite(void *buffer, type_i size, type_i count, FILE *fp)
91. //la variabile "buffer" e' un puntatore all'area di memoria interessata dal trasferimento d
i informazioni.
92. //stessa cosa vale per "fp" che e' il puntatore al file interessato dalle operazioni di I/O
.
93. //Le variabili "size" e "count" indicano rispettivamente il numero massimo di voci trasferi
to e il numero di byte di ciascuna voce.
94. /*Le chiamate fread ed fwrite producono il seguente risultato:
95. Sono trasferite fino a "count" voci, ciascuno di "size" bytes, tra l'unita' specificata da
"fp" e l'area di memoria puntata da "buffer".
96. Le due funzioni restituiscono il numero di voci completamente trasferite.*/
97.
98. void inserimento_dati(FILE *fp)
99. {
100.    strcpy(studenti[0].nome_cognome,"Angelo Rea");
101.    strcpy(studenti[0].matricola,"0124/001124");
102.    studenti[0].esami=5;
103.    studenti[0].media=28.5;
104.    studenti[0].cfu=40;
105.    studenti[0].voto=0;
106.
107.    strcpy(studenti[1].nome_cognome,"Claudio Esposito");
108.    strcpy(studenti[1].matricola,"0124/001125");
109.    studenti[1].esami=3;
110.    studenti[1].media=29.3;
111.    studenti[1].cfu=30;
112.    studenti[1].voto=0;
113.
114.    strcpy(studenti[2].nome_cognome,"Mattia Capasso");
115.    strcpy(studenti[2].matricola,"0124/001145");
116.    studenti[2].esami=5;
117.    studenti[2].media=25.6;

```

```

118.     studenti[2].cfu=46;
119.     studenti[2].voto=0;
120.     fp=fopen("studenti1.dat","w+b");
121.
122.     if((fwrite(studenti,sizeof(studente),3,fp))!=3)
123.     {
124.         puts("WRITE ERROR!");
125.         exit(1);
126.     }
127.     fclose(fp);
128. }
129.
130. void lettura_dati(FILE *fp)
131. {
132.     short i;
133.     fp=fopen("studenti1.dat","r+b");
134.     printf("\t\nLISTA STUDENTI:");
135.     for(i=0;i<CLASSE;i++)
136.     {
137.         if((fread(studenti,sizeof(studente),1,fp))!=1)
138.         {
139.             puts("READ ERROR!\a");
140.             exit(1);
141.         }
142.         printf("\nStudente No. %d\n",i+1);
143.         printf("Nome e Cognome: %s\n",studenti[i].nome_cognome);
144.         printf("Matricola: %s\n",studenti[i].matricola);
145.         printf("Esami: %hd\n",studenti[i].esami);
146.         printf("Media Ponderata: %.3f\n",studenti[i].media);
147.         printf("CFU: %hd\n",studenti[i].cfu);
148.
149.     }
150.     fclose(fp);
151. }
152.
153. void new_esame(FILE *fp)
154. {
155.     short i;
156.     fp=fopen("EsameProg2.dat","w+b");
157.     strcpy(studenti[0].nome_cognome,"Angelo Rea");
158.     strcpy(studenti[0].matricola,"0124/001124");
159.     studenti[0].voto=30;
160.     strcpy(studenti[1].nome_cognome,"Esposito Claudio");
161.     strcpy(studenti[1].matricola,"0124/001125");
162.     studenti[1].voto=28;
163.     if((fwrite(studenti,sizeof(studente),3,fp))!=3)
164.     {
165.         puts("WRITE ERROR!");
166.         exit(1);
167.     }
168.     fclose(fp);
169.     printf("\t\nLISTA STUDENTI CHE HANNO SUPERATO L'ESAME:");
170.     fp=fopen("esameProg2.dat","r+b");
171.     for(i=0;i<PROMOSSI;i++)
172.     {
173.         if((fread(studenti,sizeof(studente),1,fp))!=1)
174.         {
175.             puts("READ ERROR!\a");
176.             exit(1);
177.         }
178.         printf("\nStudente No. %d\n",i+1);
179.         printf("Nome e Cognome: %s\n",studenti[i].nome_cognome);
180.         printf("Matricola: %s\n",studenti[i].matricola);
181.         printf("Voto: %hd\n",studenti[i].voto);
182.     }
183.     fclose(fp);

```

```

184. }
185.
186.
187. float calcola_media(short *j, short *i)
188. {
189.     float OLD_MEDIA=studenti[*j].media, NEW_MEDIA=0.0;
190.     short OLD_CFU=studenti[*j].cfu,NEW_CFU=OLD_CFU+9;
191.     NEW_MEDIA=(OLD_MEDIA*OLD_CFU)+(9*studenti2[*i].voto);
192.     return NEW_MEDIA/NEW_CFU;
193. }
194.
195.
196. void aggiornamento()
197. {
198.     short i,j;
199.     for(i=0;i<PROMOSSI;i++)
200.     {
201.         for(j=0;j<CLASSE;j++)
202.         {
203.             if(strcmp(studenti[j].matricola,studenti2[i].matricola)==0)
204.             {
205.                 studenti[j].media=calcola_media(&j,&i);
206.                 studenti[j].cfu+=9;
207.                 studenti[j].esami+=1;
208.                 studenti[j].voto=studenti2[i].voto;
209.                 break;
210.             }
211.         }
212.     }
213. }
214.
215.
216. void stampa_aggiornamento(FILE *fp)
217. {
218.     short i;
219.     fp=fopen("studenti2.dat","r+b");
220.     printf("\t\nLISTA STUDENTI AGGIORNATA:");
221.     if((fread(&studenti,sizeof(studente),1,fp))!=1)
222.     {
223.         puts("READ ERROR!\a");
224.         exit(1);
225.     }
226.     for(i=0;i<CLASSE;i++)
227.     {
228.         printf("\nStudente No. %d\n",i+1);
229.         printf("Nome e Cognome: %s\n",studenti[i].nome_cognome);
230.         printf("Matricola: %s\n",studenti[i].matricola);
231.         printf("Esami tot: %hd\n",studenti[i].esami);
232.         printf("Media Ponderata Aggiornata: %.3f\n",studenti[i].media);
233.         printf("CFU tot: %hd\n",studenti[i].cfu);
234.         printf("Ultimo voto registrato: %hd\n",studenti[i].voto);
235.     }
236.     fclose(fp);
237. }

```

## ESECUZIONE 1

LISTA STUDENTI:

Studente No. 1

Nome e Cognome: Angelo Rea

Matricola: 0124/001124

Esami: 5

Media Ponderata: 28.500

CFU: 40

Studente No. 2

Nome e Cognome: Claudio Esposito

Matricola: 0124/001125

Esami: 3

Media Ponderata: 29.300

CFU: 30

Studente No. 3

Nome e Cognome: Mattia Capasso

Matricola: 0124/001145

Esami: 5

Media Ponderata: 25.600

CFU: 46

LISTA STUDENTI CHE HANNO SUPERATO L'ESAME:

Studente No. 1

Nome e Cognome: Angelo Rea

Matricola: 0124/001124

Voto: 30

Studente No. 2

Nome e Cognome: Esposito Claudio

Matricola: 0124/001125

Voto: 28

[Continuo esecuzione 1] -----→

**LISTA STUDENTI:**  
Studente No. 1  
Nome e Cognome: Angelo Rea  
Matricola: 0124/001124  
Esami: 5  
Media Ponderata: 28.500  
CFU: 40

Studente No. 2  
Nome e Cognome: Claudio Esposito  
Matricola: 0124/001125  
Esami: 4  
Media Ponderata: 29.000  
CFU: 39

Studente No. 3  
Nome e Cognome: Mattia Capasso  
Matricola: 0124/001145  
Esami: 5  
Media Ponderata: 25.600  
CFU: 46

**LISTA STUDENTI AGGIORNATA:**  
Studente No. 1  
Nome e Cognome: Angelo Rea  
Matricola: 0124/001124  
Esami tot: 6  
Media Ponderata Aggiornata: 28.776  
CFU tot: 49  
Ultimo voto registrato: 30

Studente No. 2  
Nome e Cognome: Claudio Esposito  
Matricola: 0124/001125  
Esami tot: 4  
Media Ponderata Aggiornata: 29.000  
CFU tot: 39  
Ultimo voto registrato: 28

Studente No. 3  
Nome e Cognome: Mattia Capasso  
Matricola: 0124/001145  
Esami tot: 5  
Media Ponderata Aggiornata: 25.600  
CFU tot: 46  
Ultimo voto registrato: 0

Process returned 0 (0x0) execution time : 4.404 s  
Press ENTER to continue.

## Esercizio 36[liv.2]

### Ricerca occorrenze di un pattern in un file

```
1.  /***[liv.2] Scrivere una function C per la ricerca diretta di tutte le occorrenze di un pattern in un testo dove:  
2.  - il testo e' memorizzato in un file.  
3.  - la lettura del file avviene a pezzi mediante un array-buffer.  
4.  - il pattern e' definito in input (e costruito dinamicamente).**/  
5.  
6.  
7.          /**LIBRERIE**/  
8. #include <stdio.h>  
9. #include <stdlib.h>  
10. #include <string.h>  
11.  
12. #define N 255 /*numero massimo di caratteri in una riga di un file*/  
13.  
14. struct occorrenza  
15. {  
16.     int riga;  
17.     int posizione;  
18. };  
19.  
20.          /**PROTOTIPI**/  
21. void stampa_occorrenze(struct occorrenza a[], int n, char *patt);  
22.  
23.          /**MAIN**/  
24. int main()  
25. {  
26.     int i, j; //indici  
27.     int riga=0, trovato=-1;  
28.     int dpatt, lpatt, lbuff;  
29.     //lpatt: lunghezza pattern - dpatt: dimensione pattern - lbuff: lunghezza buffer  
30.     char *pattern, buffer[N];  
31.     FILE *fp;  
32.     struct occorrenza a[100];  
33.  
34.     char percorso[100]="/Users/MattiaCapasso/Desktop/UNI/PROG2/ES36/Testo.txt";  
35.  
36.     //Apertura del file e controllo di eventuali errori  
37.     if ((fp=fopen(percorso,"r")) == NULL) {  
38.         printf("\n\n***ERRORE APERTURA FILE***\n\n");  
39.         exit(1);  
40.     }  
41.  
42.     //Inserimento del pattern da ricercare nel file  
43.     printf("\n Inserire dimensione pattern: "); scanf("%d",&dpatt); //inserimento dimensione  
44.     e  
45.     pattern=(char *)malloc(dpatt); //allocazione dinamica memoria per il pattern  
46.     printf("\n Inserire il pattern: "); fflush(stdin); //inserimento dei caratteri  
47.     gets(pattern);  
48.     gets(pattern);  
49.     lpatt=strlen(pattern); //lpatt: lunghezza pattern  
50.  
51.  
52.     while (!feof(fp)) //si itera finche non finisce il file  
53.     {  
54.         fgets(buffer, N, fp); //si leggono al piu' N-  
55.         1 caratteri dal file e si mettono in buffer  
56.         riga++;  
57.         i=0;  
58.         lbuff=strlen(buffer); //si imposta la lunghezza del buffer
```

```

59.         while (i <= lbuff-lpatt) //serve per trovare pattern a cavallo tra due buffer
60.         {
61.             j=0;
62.             /*aumenta j per tutte le corrispondenze che ci sono tra i caratteri del pattern
63. e i caratteri dei buffer */
64.             while (j<lpatt && (*(buffer+i+j) == *(pattern+j)))
65.                 j++;
66.             /*se il valore di j uguaglia il valore della lunghezza del pattern, allora e' s
tata
67.             trovata un'occorrenza */
68.             if (j==lpatt)
69.             {
70.                 trovato++;
71.                 a[trovato].riga=riga; //memorizza riga e posizione dell'occorrenza
72.                 a[trovato].posizione=i+1; //+1 dovuto all'indice del buffer che parte da 0
73.             }
74.             i++;
75.         }
76.     }
77.     fclose(fp); //chiudi il file
78.
79.     if (trovato==-1) //se non sono state trovate occorrenze, stampa un messaggio
80.         printf("\n Pattern '%s' NON TROVATO\n\n",pattern);
81.     else //altrimenti stampa le occorrenze e le loro 'informazioni' (riga e posizione)
82.         stampa_occorrenze(a,trovato+1,pattern);
83.     }
84.
85.         /**FUNCTION*/
86.     /*Function per la stampa delle occorrenze trovate nel file */
87.     void stampa_occorrenze(struct occorrenza a[], int n, char *patt)
88.     {
89.         int i;
90.         printf("\n Pattern '%s'\n",patt);
91.         printf("\n OCCORRENZE TROVATE: %d\n\n",n);
92.         for (i=0; i<n; i++)
93.             printf(" Occorrenza %2d --
> Riga %2d - Posizione %3d\n",i+1,a[i].riga,a[i].posizione);
94.     }

```

## Esercizio 37[liv.1]

### Gestione Pila(stack) tramite Push e Pop

```
1.  /***[liv.1] Simulare in C la gestione di una pila (stack) tramite array statico (puo' essere  
    anche un array di struct)  
2.  creando le funzioni di manipolazione push() [inserimento] e pop() [eliminazione].  
3.  Il programma deve prevedere un menu' che consenta di scegliere l'operazione da eseguire.**/  
4.  
5.  
6.          /**LIBRERIE**/  
7.  #include <stdio.h>  
8.  #include <stdlib.h>  
9.  
10. #define MAX_STACK_SIZE 6  
11.  
12.          /**PROTOTIPI**/  
13. void menu(char pila[], short head);  
14. void push_s(char elem, char pila[], short *head);  
15. void pop_s(char pila[], short *head);  
16.  
17.          /**MAIN**/  
18. int main()  
19. {  
20.     char pila[MAX_STACK_SIZE]={'A','B','C','D','E','F'};  
21.     short head=MAX_STACK_SIZE-1;  
22.  
23.     menu(pila,head);  
24.  
25.     printf("\n");  
26.     return 0;  
27. }  
28.          /**FUNCTION**/  
29. void menu(char pila[], short head)  
30. {  
31.     char elem=0;  
32.     short i, scegli=0,sel;  
33.  
34.     printf("\n\n");  
35.  
36.     printf("\n\t PILA INIZIALE");  
37.     for(i=head; i>=0; i--)  
38.     {  
39.         printf("\n\t Posizione[%hd]= '%c'",i,pila[i]);  
40.     }  
41.  
42.     do  
43.     {  
44.         printf("\n\n\t MENU'\n\t [1] push(inserimento)\n\t [2]pop(eliminazione)\n\t-----  
----> ");  
45.         fflush(stdin);  
46.         scanf("%hd", &scegli);  
47.  
48.  
49.         switch(scegli)  
50.         {  
51.             case 1: printf("\n\n\t");  
52.                         printf(" Inserisci carattere per aggiungerlo alla pila(A-Z): ");  
53.                         fflush(stdin);  
54.                         do{  
55.                             scanf("%c",&elem);  
56.                         }while(elem<'A' || elem>'Z');  
57.  
58.                         if(head==MAX_STACK_SIZE-2)
```

```

59.             {
60.                 printf("\n IMPOSSIBILE AGGIUNGERE (head==MAX_STACK_SIZE-1) ");
61.             }
62.             push_s(elem, pila,&head);
63.             printf("\n\t PILA DOPO L'OPERAZIONE DI PUSH");
64.             for(i=head; i>=0; i--)
65.             {
66.                 printf("\n\t Posizione[%hd]= '%c'",i,pila[i]);
67.             }
68.             fflush(NULL);break;
69.         }
70.     case 2: if(head== -1)
71.     {
72.         printf("\n IMPOSSIBILE ELIMINARE (head== -1)");break;
73.     }
74.     pop_s(pila,&head);
75.     printf("\n\t PILA DOPO L'OPERAZIONE DI POP");
76.     for(i=head; i>=0; i--)
77.     {
78.         printf("\n\t Posizione[%hd]= '%c'",i,pila[i]);
79.     }
80.     fflush(NULL);break;
81. }
82. }
83. }
84. printf("\n\n\n\t Vuoi eseguire un'altra operazione?\n\t\t [1] SI\n\t\t [2] NO\n\t-----");
85. ---> );
86. fflush(stdin);
87. scanf("%hd",&sel);
88. }while(sel!=2);
89. }
90.
91. /*FUNCTION PER INSERIRE ELEMENTI NELLA PILA*/
92. void push_s(char elem, char pila[], short *head)
93. {
94.     /* (*head)++ <-> *head=*head+1 */
95.     *(pila+ ++*head)=elem;
96. }
97.
98. /*FUNCTION PER ELIMINARE ELEMENTI NELLA PILA*/
99. void pop_s(char pila[], short *head)
100.{}
101.    /* (*head)-- <-> *head=*head-1 */
102.    --*head;
103. }

```

## ESECUZIONE 1

```
PILA INIZIALE
Posizione[5]= 'F'
Posizione[4]= 'E'
Posizione[3]= 'D'
Posizione[2]= 'C'
Posizione[1]= 'B'
Posizione[0]= 'A'

MENU'
[1] push(inserimento)
[2]pop(eliminazione)
-----> 2

PILA DOPO L'OPERAZIONE DI POP
Posizione[4]= 'E'
Posizione[3]= 'D'
Posizione[2]= 'C'
Posizione[1]= 'B'
Posizione[0]= 'A'

Vuoi eseguire un'altra operazione?
[1] SI
[2] NO
-----> 1

MENU'
[1] push(inserimento)
[2]pop(eliminazione)
-----> 2

PILA DOPO L'OPERAZIONE DI POP
Posizione[3]= 'D'
Posizione[2]= 'C'
Posizione[1]= 'B'
Posizione[0]= 'A'
```

```
Vuoi eseguire un'altra operazione?  
[1] SI  
[2] NO  
-----> 1
```

```
MENU'  
[1] push(inserimento)  
[2]pop(eliminazione)  
-----> 1
```

```
Inserisci carattere per aggiungerlo alla pila(A-Z): 0
```

```
PILA DOPO L'OPERAZIONE DI PUSH  
Posizione[4]= '0'  
Posizione[3]= 'D'  
Posizione[2]= 'C'  
Posizione[1]= 'B'  
Posizione[0]= 'A'
```

```
Vuoi eseguire un'altra operazione?  
[1] SI  
[2] NO  
-----> 2
```

```
Process returned 0 (0x0)    execution time : 18.734 s  
Press ENTER to continue.
```

## Esercizio 38[liv.1]

### Gestione Coda(queue) tramite Enqueue e Dequeue

```
1.  /**[liv.1] Simulare in C la gestione di una coda (queue) tramite array statico (puo' essere  
    anche un array di struct)  
2.  creando le funzioni di manipolazione enqueue() [inserimento] e dequeue() [eliminazione].  
3.  Il programma deve prevedere un menu' che consenta di scegliere l'operazione da eseguire.  
4.  Le informazioni NON vanno spostate!**/  
5.  
6.          /**LIBRERIE**/  
7.  #include <stdio.h>  
8.  #include <stdlib.h>  
9.  
10. #define MAX_SIZE_QUEUE  6  
11.  
12.         /**PROTOTIPI**/  
13. void menu(char coda[], short back, short head);  
14. void enqueue(char elem, char coda[], short *back);  
15. void dequeue(char coda[], short *head);  
16.  
17.         /**MAIN**/  
18. int main()  
19. {  
20.     char coda[MAX_SIZE_QUEUE]={ 'A', 'B', 'C', 'D', 'E', 'F' };  
21.     short back,head,len=6-1;  
22.     head=-1;  
23.     back=len;  
24.  
25.     menu(coda,back,head);  
26.  
27.     printf("\n");  
28.     return 0;  
29. }  
30.         /**FUNCTION**/  
31. /*MENU'*/  
32. void menu(char coda[], short back, short head)  
33. {  
34.     char elem=0;  
35.     short i, scegli=0,sel;  
36.  
37.     printf("\n\n");  
38.  
39.     printf("\n\t CODA INIZIALE");  
40.     for(i=back; i>head; i--)  
41.     {  
42.         printf("\n\t Posizione[%hd]= '%c'",i,coda[i]);  
43.     }  
44.  
45.     do  
46.     {  
47.         printf("\n\n\t MENU'\n\t [1] Enqueue(inserimento)\n\t [2] Dequeue(eliminazione)\n\t  
\t -----> ");  
48.         fflush(stdin);  
49.         scanf("%hd", &scegli);  
50.  
51.  
52.         switch(scegli)  
53.         {  
54.             case 1: printf("\n\n\t");  
55.                         printf(" Inserisci carattere per aggiungerlo alla pila(A-Z): ");  
56.                         fflush(stdin);  
57.                         do{  
58.                             scanf("%c",&elem);  
59.                         }while(elem<'A' || elem>'Z');
```

```

60.
61.           if(back>MAX_SIZE_QUEUE-1)
62.           {
63.               printf("\n IMPOSSIBILE AGGIUNGERE (back==MAX_STACK_SIZE-1) ");
64.           }
65.           enqueue(elem,coda,&back);
66.           printf("\n\t CODA DOPO L'OPERAZIONE DI ENQUEUE(inserimento)");
67.
68.           for(i=back; i>head; i--)
69.           {
70.               printf("\n\t Posizione[%hd]= '%c'",i,coda[i]);
71.           }
72.           fflush(NULL);break;
73.
74.       case 2: if(head==back)
75.       {
76.           printf("\n\t LA CODA E' VUOTA");
77.       }
78.       dequeue(coda,&head);
79.       printf("\n\t CODA DOPO L'OPERAZIONE DI DEQUEUE(eliminazione)");
80.
81.           for(i=back; i>head; i--)
82.           {
83.               printf("\n\t Posizione[%hd]= '%c'",i,coda[i]);
84.           }
85.           fflush(NULL);break;
86.       }
87.
88.   printf("\n\n\n\t Vuoi eseguire un'altra operazione?\n\t\t [1] SI\n\t\t [2] NO\n\t\t----\n---> ");
89.   fflush(stdin);
90.   scanf("%hd",&sel);
91.   }while(sel!=2);
92. }
93.
94. /*FUNCTION PER INSERIRE ELEMENTI NELLA CODA*/
95. void enqueue(char elem, char coda[], short *back)
96. {
97.     *(coda+ ++*back)=elem;
98. }
99.
100. /*FUNCTION PER ELIMINARE ELEMENTI DALLA CODA*/
101. void dequeue(char coda[], short *head)
102. {
103.     ++*head;
104. }

```

## ESECUZIONE 1

```
CODA INIZIALE
Posizione[5]= 'F'
Posizione[4]= 'E'
Posizione[3]= 'D'
Posizione[2]= 'C'
Posizione[1]= 'B'
Posizione[0]= 'A'
```

```
MENU'
[1] Enqueue(inserimento)
[2] Dequeue(eliminazione)
-----> 2
```

```
CODA DOPO L'OPERAZIONE DI DEQUEUE(eliminazione)
Posizione[5]= 'F'
Posizione[4]= 'E'
Posizione[3]= 'D'
Posizione[2]= 'C'
Posizione[1]= 'B'
```

```
Vuoi eseguire un'altra operazione?
[1] SI
[2] NO
-----> 1
```

```
MENU'
[1] Enqueue(inserimento)
[2] Dequeue(eliminazione)
-----> 2
```

```
CODA DOPO L'OPERAZIONE DI DEQUEUE(eliminazione)
Posizione[5]= 'F'
Posizione[4]= 'E'
Posizione[3]= 'D'
Posizione[2]= 'C'
```

```
Vuoi eseguire un'altra operazione?
[1] SI
[2] NO
-----> 1
```

```
MENU'
[1] Enqueue(inserimento)
[2] Dequeue(eliminazione)
-----> 1

Inserisci carattere per aggiungerlo alla pila(A-Z): T

CODA DOPO L'OPERAZIONE DI ENQUEUE(inserimento)
Posizione[6]= 'T'
Posizione[5]= 'F'
Posizione[4]= 'E'
Posizione[3]= 'D'
Posizione[2]= 'C'

Vuoi eseguire un'altra operazione?
[1] SI
[2] NO
----->
2

Process returned 0 (0x0)    execution time : 20.561 s
Press ENTER to continue.
```

## Esercizio 40[liv.1]

### Algoritmo di visita di una lista lineare

```
1. 
2.  /**[liv.1] Simulare in C l'algoritmo di visita di una lista lineare gia' memorizzata mediane
3.  te un array statico di struct
4.  (come nella tabella in basso) in cui il primo campo contiene l'informazione ed il secondo con
5.  tiene il link al nodo successivo
6.  (in questo caso il link e' l'indice di una componente dell'array).
7.  Memorizzando nell'array i dati come mostrato nella figura che segue, l'output del programma
8.  consiste nell'elenco di nomi ordinato alfabeticamente.*/
9. 
10. 
11. 
12. 
13. #define LEN 20
14. 
15. struct node{
16.     char Lista_nomi[LEN]; //dato contenuto nel nodo per inserire le informazioni
17.     struct node *next; //puntatore al nodo successivo
18. };
19. 
20. struct node new_node[11]; //array che conterra le informazioni
21. 
22. 
23. 
24. 
25. int main()
26. {
27.     int i,p_Testa=3;
28.     struct node *punt;
29. 
30.     strcpy(new_node[0].Lista_nomi,"Anna\t      5");
31.     new_node[0].next=&new_node[5];
32. 
33.     strcpy(new_node[1].Lista_nomi,"Mario\t      8");
34.     new_node[1].next=&new_node[8];
35. 
36.     strcpy(new_node[2].Lista_nomi,"Giuseppe\t      6");
37.     new_node[2].next=&new_node[6];
38. 
39.     strcpy(new_node[3].Lista_nomi,"Angela\t      0");
40.     new_node[3].next=&new_node[0];
41. 
42.     strcpy(new_node[4].Lista_nomi,"Valeria\t      -1");
43.     new_node[4].next=NULL;
44. 
45.     strcpy(new_node[5].Lista_nomi,"Fabrizio\t      7");
46.     new_node[5].next=&new_node[7];
47. 
48.     strcpy(new_node[6].Lista_nomi,"Marianna\t      1");
49.     new_node[6].next=&new_node[1];
50. 
51.     strcpy(new_node[7].Lista_nomi,"Giovanni\t      2");
52.     new_node[7].next=&new_node[2];
53. 
54.     strcpy(new_node[8].Lista_nomi,"Patrizia\t      10");
55.     new_node[8].next=&new_node[10];
56. 
57.     strcpy(new_node[9].Lista_nomi,"Valentina\t      4");
```

```
58.     new_node[9].next=&new_node[4];
59.
60.     strcpy(new_node[10].Lista_nomi,"Sara\t         9");
61.     new_node[10].next=&new_node[9];
62.
63.     printf("\n\n\t LISTA NON ORDINATA \n");
64.     printf("\n\t INFO\t\t PT ");
65.     for(i=0; i<11; i++)
66.     {
67.         printf("\n\t %s ",new_node[i].Lista_nomi);
68.     }
69.     printf("\n\n");
70.
71.     punt=&new_node[p_Testa];
72.
73. //VISITA DI UNA LISTA LINEARE
74.     printf("\n\n\t LISTA ORDINATA \n");
75.     while(punt != NULL)
76.     {
77.         printf("\n\t %s ", punt->Lista_nomi);
78.         punt=punt->next;
79.     }
80.     printf("\n");
81.     return 0;
82. }
```

## ESECUZIONE 1

### LISTA NON ORDINATA

	PT
INFO	
Anna	5
Mario	8
Giuseppe	6
Angela	0
Valeria	-1
Fabrizio	7
Marianna	1
Giovanni	2
Patrizia	10
Valentina	4
Sara	9

### LISTA ORDINATA

Angela	0
Anna	5
Fabrizio	7
Giovanni	2
Giuseppe	6
Marianna	1
Mario	8
Patrizia	10
Sara	9
Valentina	4
Valeria	-1

```
Process returned 0 (0x0)    execution time : 0.011 s
Press ENTER to continue.
```

## Esercizio 42[liv.1]

### Gestione di una lista

```
1.  /***[liv.1] Realizzare la gestione di una lista lineare mediante menu' (visualizzazione mediante visita,
2.  inserimento in testa, inserimento in mezzo, eliminazione in testa, eliminazione in mezzo) implementando
3.  la lista lineare con una struttura autoriferente dinamica.*/
4.
5.
6.          /**LIBRERIE*/
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10.
11. //Definiamo un nuovo tipo di struct INFO_FIELD, ovvero il campo informazione del nodo
12. typedef struct{
13.     char nome[20]; //dato di tipo char contenuto nella struttura
14.     short eta; //dato di tipo short contenuto nella struttura
15. } INFO_FIELD; //nome campo informazione della struttura
16.
17. //Definiamo una struct PERSONA, cioÃ un singolo nodo
18. struct PERSONA
19. {
20.     INFO_FIELD info; //dato di tipo INFO_FIELD(quindi dato che puÃ² accedere alla struttura precedente), contenuto nella struttura
21.     struct PERSONA *p_next; //variabile puntatore(32 bit) autoriferente dinamica, cioÃ che punta alla struttura stessa
22. };
23.
24.
25.          /**PROTOTIPI*/
26. struct PERSONA *crea_lista(); //crea la lista vuota
27. void insl_testa(INFO_FIELD nuovodato,struct PERSONA **p_head); //equivale a: struct PERSONA *insl_testa(INFO_FIELD dato,struct PERSONA *p_head);
28. void insl_nodo(INFO_FIELD nuovodato,struct PERSONA **p_punt); //equivale a: struct PERSONA *insl_nodo(INFO_FIELD dato,struct PERSONA *p_head);
29. INFO_FIELD legge_info_nodo(struct PERSONA **p_punt); //restituisce il campo informazione del nodo corrente
30. void elim_testa(struct PERSONA**p_head); //elimina nodo in testa
31. void elim_nodo(struct PERSONA **p_punt); //elimina nodo successore
32. void visualizza_list(struct PERSONA**);
33.
34.
35.          /**MAIN*/
36. int main()
37. {
38.     struct PERSONA *head, *punt; //Definiamo due puntatori alla struttura PERSONA
39.     short scegli;
40.     INFO_FIELD nuovodato;
41.
42.     head=crea_lista();
43.
44.     while(1)
45.     {
46.         printf("\n\t\t MENU \n [1] Visualizzazione mediante visita\n [2] Inserimento in testa\n [3] Inserimento in mezzo\n"
47.             " [4] Eliminazione in testa\n [5] Eliminazione in mezzo\n [6] Esci\n -----");
48.         fflush(stdin);
49.         scanf("%hd",&scegli);
50.
51.
```

```

52.         switch(scegli)
53.         {
54.             case 1: visualizza_lista(&head);break;
55.
56.             case 2: punt=head; /*aggiorna il puntatore al nodo corrente */
57.                         printf("\n INSERIMENTO NODO IN TESTA");
58.                         printf("\n\n Inserisci nome: ");
59.                         fflush(stdin);
60.                         scanf("%s", nuovodato.nome);
61.                         printf(" Inserisci eta': ");
62.                         fflush(stdin);
63.                         scanf("%hd", &nuovodato.eta);
64.                         insl_testa(nuovodato,&head);break;
65.
66.             case 3: punt=head; /*aggiorna il puntatore al nodo corrente */
67.                         printf("\n INSERIMENTO NODO IN MEZZO (dopo nodo corrente)");
68.                         printf("\n\n Inserisci nome: ");
69.                         fflush(stdin);
70.                         scanf("%s", nuovodato.nome);
71.                         printf(" Inserisci eta': ");
72.                         fflush(stdin);
73.                         scanf("%hd", &nuovodato.eta);
74.                         insl_nodo(nuovodato,&punt);break;
75.
76.             case 4: elim_testa(&head);break;
77.             case 5: elim_nodo(&punt);break;
78.             case 6: exit(EXIT_FAILURE);break;
79.         }
80.     }
81.
82.     return 0;
83. }
84.
85.
86. /**FUNCTION*/
87.
88. /*FUNCTION PER CREARE UNA LISTA VUOTA*/
89. struct PERSONA *crea_lista() //chiamata alla function head=crea_lista();
90. {
91.     struct PERSONA *testa; //testa della lista
92.     testa=NULL;
93.     return testa;
94. };
95.
96. /**
97. /*FUNCTION PER INSERIRE DATO IN TESTA ALLA LISTA
98. Si ricorda che per l'inserimento in testa occorre:
99. - prima far puntare il nuovo nodo al nodo di testa
100. - poi aggiornare il puntatore "head" al nuovo nodo */
101. void insl_testa(INFO_FIELD nuovodato, struct PERSONA **p_head)/**p_head punta a head cioÃ"
102. alla testa della lista
103. {
104.     struct PERSONA *ptr; //puntatore alla struttura PERSONA
105.     ptr=calloc(1, sizeof(struct PERSONA)); //alloca 1 spazio in memoria di sizeof(struct PE
106.     RSONA) byte ovvero 32 byte
107.     if(ptr==NULL)
108.     {
109.         printf("\n Allocazione fallita ");
110.         ptr->info=nuovodato;
111.         ptr->p_next=*p_head;
112.         *p_head=ptr; //aggiorna head al nuovo nodo
113.     }
114. }
115.

```

```

116. /**
117. /*FUNCTION PER INSERIRE DATO DOPO IL NODO CORRENTE
118. Si ricorda che per inserire un nodo nel mezzo occorre:
119. - prima modificare il puntatore del nuovo nodo in modo da farlo puntare al nodo successivo
120. - poi modificare il puntatore del nodo precedente in modo da farlo puntare al nuovo nodo */
121. void insl_nodo(INFO_FIELD nuovodato, struct PERSONA **p_punt)// p_punt punta a punt
122. {
123.     struct PERSONA *ptr;
124.
125.     ptr=calloc(1, sizeof(struct PERSONA));
126.     if(ptr==NULL)
127.     {
128.         printf("\n Allocazione fallita ");
129.     }
130.     ptr->info=nuovodato;
131.     ptr->p_next=(*p_punt)->p_next;
132.     (*p_punt)->p_next=ptr; //aggiorna punt al nodo corrente
133. }
134.
135. /**
136. /*FUNCTION PER ELIMININARE NODO IN TESTA
137. Si ricorda che per eliminare il nodo di testa non occorre far altro che modificare il nodo
138. puntato da "head" facendolo puntare direttamente al nodo successivo al primo */
139. void elim_testa(struct PERSONA **p_head) //chiamata: elim_testa(&p_head)
140. {
141.     struct PERSONA *ptr;
142.
143.     if((*p_head)->p_next == NULL) /*se Ã" presente un solo nodo nella lista */
144.     {
145.         free(*p_head);
146.         *p_head = NULL; /*la lista ora e' vuota*/
147.     }
148.     else /*altrimenti*/
149.     {
150.         ptr=(*p_head)->p_next; /*salva il link del nodo puntato dal nodo di testa */
151.         free(*p_head); /*libera l'area puntata da p_head */
152.         *p_head = ptr; /*aggiorna il puntatore alla testa della lista mediante il link salvato*/
153.     }
154. }
155.
156. /**
157. /*FUNCTION PER ELIMINARE NODO SUCCESSORE
158. Si ricorda che per eliminare un nodo nel mezzo non occorre far altro che modificare il campo
159. o
160. puntatore del nodo precedente facendolo puntare direttamente al nodo successivo */
161. void elim_nodo(struct PERSONA **p_punt) //chiamata: elim_nodo(&punt)
162. {
163.     struct PERSONA *ptr;
164.
165.     ptr=(*p_punt)->p_next; /*si fa in modo che ptr sia il nodo successivo di quello corrente */
166.     /*ptr punta al successore del nodo successivo a quello corrente:
167.      corrente -> successivo -> successore */
168.     (*p_punt)->p_next = ptr->p_next; /*si assegna il link puntato da ptr (successore) al nodo
169.                                         corrente in modo che il successivo venga "saltato" */
170.     /*libera l'area occupata dalla variabile ptr */
171.     free(ptr);
172.
173. /**
174. /*FUNCTION CHE RESITUTISCE IL CAMPO INFORMAZIONI DEL NODO CORRENTE*/

```

```

175. INFO_FIELD legge_info_nodo(struct PERSONA **p_punt) //chiamata: info=legge_info_nodo(pt_nod
o);
176. {
177.     /*definidce una struttura lista generica che condivide con il main solo il tipo
178.     INFO_FIELD (globale) */
179.     INFO_FIELD nuovodato; //definisce una nuova variabile di tipo INFO_FIELD (dichiarata gl
obalmente)
180.
181.     nuovodato = (*p_punt)->info; /*gli si assegna il campo info del nodo puntato da p_punt */
182.     return nuovodato;
183. }
184.
185. /**
186. /*FUNCTIONE PER VISUALIZZARE LA LISTA MEDIANTE VISITA*/
187. void visualizza_listा(struct PERSONA **p_head)
188. {
189.     int i=0;
190.     struct PERSONA *ptr;
191.
192.     printf("\n LISTA ATTUALE");
193.     if(*p_head != NULL)
194.     {
195.         /*stampa il nodo di testa*/
196.         printf("\n Nodo[%d] Nome= %s - Eta'= %hd\n",i++, (*p_head)->info.nome, (*p_head)->info.eta);
197.
198.         /*inizializza il puntatore che scorre la lista*/
199.         ptr =(*p_head)->p_next;
200.
201.         while (ptr != NULL) /*mentre la lista non e' finita */
202.         {
203.             /*stampa il nodo corrente */
204.             printf(" Nodo[%d] Nome= %s - Eta'= %hd\n",i++, ptr->info.nome, ptr->info.eta);
205.             ptr=(ptr)->p_next; /*aggiorna il puntatore */
206.         }
207.     }
208.     else /*altrimenti stampa */
209.         printf("****LISTA VUOTA****\n\n");
210. }

```

## ESECUZIONE 1

```
MENU
[1] Visualizzazione mediante visita
[2] Inserimento in testa
[3] Inserimento in mezzo
[4] Eliminazione in testa
[5] Eliminazione in mezzo
[6] Esci
-----> 2
```

### INSERIMENTO NODO IN TESTA

```
Inserisci nome: mattia
Inserisci eta': 21
```

```
MENU
[1] Visualizzazione mediante visita
[2] Inserimento in testa
[3] Inserimento in mezzo
[4] Eliminazione in testa
[5] Eliminazione in mezzo
[6] Esci
-----> 2
```

### INSERIMENTO NODO IN TESTA

```
Inserisci nome: angelo
Inserisci eta': 20
```

```
MENU
[1] Visualizzazione mediante visita
[2] Inserimento in testa
[3] Inserimento in mezzo
[4] Eliminazione in testa
[5] Eliminazione in mezzo
[6] Esci
-----> 3
```

### INSERIMENTO NODO IN MEZZO (dopo nodo corrente)

```
Inserisci nome: claudio
Inserisci eta': 21
```

MENU

- [1] Visualizzazione mediante visita
  - [2] Inserimento in testa
  - [3] Inserimento in mezzo
  - [4] Eliminazione in testa
  - [5] Eliminazione in mezzo
  - [6] Esci
- > 1

**LISTA ATTUALE**

Nodo[0] Nome= angelo - Eta'= 20  
Nodo[1] Nome= claudio - Eta'= 21  
Nodo[2] Nome= mattia - Eta'= 21

MENU

- [1] Visualizzazione mediante visita
  - [2] Inserimento in testa
  - [3] Inserimento in mezzo
  - [4] Eliminazione in testa
  - [5] Eliminazione in mezzo
  - [6] Esci
- > 5

MENU

- [1] Visualizzazione mediante visita
  - [2] Inserimento in testa
  - [3] Inserimento in mezzo
  - [4] Eliminazione in testa
  - [5] Eliminazione in mezzo
  - [6] Esci
- > 1

**LISTA ATTUALE**

Nodo[0] Nome= angelo - Eta'= 20  
Nodo[1] Nome= mattia - Eta'= 21

```
        MENU
[1] Visualizzazione mediante visita
[2] Inserimento in testa
[3] Inserimento in mezzo
[4] Eliminazione in testa
[5] Eliminazione in mezzo
[6] Esci
-----> 4
```

```
        MENU
[1] Visualizzazione mediante visita
[2] Inserimento in testa
[3] Inserimento in mezzo
[4] Eliminazione in testa
[5] Eliminazione in mezzo
[6] Esci
-----> 1
```

```
LISTA ATTUALE
Nodo[0] Nome= mattia - Eta'= 21
```

```
        MENU
[1] Visualizzazione mediante visita
[2] Inserimento in testa
[3] Inserimento in mezzo
[4] Eliminazione in testa
[5] Eliminazione in mezzo
[6] Esci
-----> 6
```

```
Process returned 1 (0x1)    execution time : 61.465 s
Press ENTER to continue.
```

## Esercizio 43[liv.2]

### Costruzione lista versione iterativa

```
1.  /**[liv.2] A partire dalla versione ricorsiva di costruzione di una lista in C, scriverne 1
   a versione iterativa.*/
2.
3.          /**LIBRERIE*/
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. typedef char DATA;
8.
9. struct linked_list{
10.     DATA d;
11.     struct linked_list *next;
12. };
13.
14. typedef struct linked_list ELEMENT;
15. typedef ELEMENT *LINK;
16.
17.          /**PROTOTIPI*/
18. LINK array_to_list(DATA []);
19.
20.          /**MAIN*/
21. int main()
22. {
23.     DATA c_arr[]="\n Lista costruita con successo!";
24.     LINK head_list, p_list;
25.
26.     head_list=array_to_list(c_arr);
27.     p_list=head_list;
28.
29.     while(p_list != NULL)
30.     {
31.         putchar(p_list->d);
32.         p_list=p_list->next;
33.     }
34.     puts("");
35. }
36.          /**FUNCTION*/
37. LINK array_to_list(DATA s[])
38. {
39.     LINK head=NULL, coda;
40.     int i;
41.
42.     if(s[0] != '\0')
43.     {
44.         printf("\n");
45.         head=malloc(sizeof(ELEMENT));
46.         head->d=s[0];
47.
48.         coda=head;
49.         for(i=1; s[i]!='\0'; i++) //aggiunge in coda
50.         {
51.             coda->next=malloc(sizeof(ELEMENT));
52.             coda=coda->next;
53.             coda->d=s[i];
54.         }
55.         coda->next=NULL;
56.     }
57.     return head; //fine lista
58. }
```

## ESECUZIONE 1

Lista costruita con successo!

```
Process returned 0 (0x0)    execution time : 0.008 s
Press ENTER to continue.
```

## Esercizio 44[liv.2]

### Ordinamento lista in ordine alfabetico

```
1.  /***[liv.2] Scrivere una function C per costruire una lista ordinata in ordine alfabetico a
2.  partire da un elenco di
3.
4.           **LIBRERIE**
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.  #include <string.h>
8.
9.           **STRUCT**
10.
11. typedef struct lista
12. {
13.     char name[20];
14.     struct lista *p_next;
15. }LISTA;
16.
17. LISTA Lista_nome[11];
18.
19.           **PROTOTIPI**
20. void Bubble_sort(LISTA Lista_nome[], int n);
21.
22.           **MAIN**
23. int main()
24. {
25.     int i,size;
26.     LISTA *head,*punt;
27.     head=malloc(sizeof(LISTA));
28.     punt=malloc(sizeof(LISTA));
29.
30.     printf("\n Inserisci il numero di nomi da inserire: ");
31.     scanf("%d",&size);
32.
33.     printf("\n");
34.     for(i=0; i<size; i++)
35.     {
36.         printf(" Inserisci nome %d: ",i);
37.         fflush(stdin);
38.         scanf("%s",Lista_nome[i].name);
39.     }
40.
41.     printf("\n\n LISTA NON ORDINATA \n");
42.     for(i=0; i<size; i++)
43.     {
44.         printf("\n %d: %s",i,Lista_nome[i].name);
45.     }
46.
47.     Bubble_sort(Lista_nome,size);
48.
49. //STAMPA ELEMENTI ORDINATI
50. printf("\n\n LISTA ORDINATA \n");
51. for(i=0; i<size; i++)
52. {
53.     printf("\n %d: %s", i, Lista_nome[i].name);
54. }
55. printf("\n\n");
56.
57. return 0;
58. }
59.           **FUNCTION**/
```

```
60. void Bubble_sort(LISTA Lista_nome[], int n)
61. {
62.     int i;
63.     LISTA c;//variabile d'appoggio
64.
65.     while(n>1)
66.     {
67.         for(i=0; i<n-1; i++)
68.         {
69.             if(strcmp(Lista_nome[i].name,Lista_nome[i+1].name)>0)//se l'elemento i e' piÃ¹
grande dell'elemento i+1 allora....
70.             {
71.                 c=Lista_nome[i];//c sara' uguale all'elemento i-esimo
72.                 Lista_nome[i]=Lista_nome[i+1];//l'elemento i-
esimo sara' uguale all'elemento i-esimo+1
73.                 Lista_nome[i+1]=c;//e l'elemento i-
esimo+1 sara' uguale a c che e' l'lemento i-esimo della lista
74.             }
75.         }
76.         n--;
77.     }
78. }
```

## ESECUZIONE 1

```
Inserisci il numero di nomi da inserire: 4
```

```
Inserisci nome 0: Mario  
Inserisci nome 1: Mattia  
Inserisci nome 2: Claudio  
Inserisci nome 3: Angelo
```

```
LISTA NON ORDINATA
```

```
0: Mario  
1: Mattia  
2: Claudio  
3: Angelo
```

```
LISTA ORDINATA
```

```
0: Angelo  
1: Claudio  
2: Mario  
3: Mattia
```

```
Process returned 0 (0x0)    execution time : 21.406 s  
Press ENTER to continue.
```

## Esercizio 45a[liv.1] Gestione Pila mediante menu'

```
1.  /**[liv.1] Realizzare in C le funzioni per la gestione, mediante menu', delle strutture dat
   i pila e coda mediante lista
2.  lineare dinamica e generica con [rispettivamente senza] nodo sentinella.*/
3.
4.
5.  ****[PILA]*****
6.
7.      **LIBRERIE**
8.  #include <stdio.h>
9.  #include <stdlib.h>
10. #include <string.h>
11.
12.
13. /*Il nodo sentinella serve per evitare di avere due function per l'inserimento e due functi
   on
14. per l'eliminazione. Infatti Ã" possibile avere un'unica sequenza di operazioni sia per gli
15. inserimenti/eliminazioni sulla testa che per quelli fatti sul nodo corrente.
16. Occorre semplicemente aggiugere un nodo fittizio che funga da testa ma che punti alla testa
17. reale della lista: un nodo sentinella.
18. Ogni modifica sulla testa comporterÃ in questo modo la sola alterazione del campo puntator
   e del
19. nodo sentinella. */
20.
21. typedef struct
22. {
23.     char nome[20]; //tipo del campo informazione del nodo della lista globale
24. }INFO_FIELD;
25.
26. short scegli=0,len_info; //variabile dichiarata globalmente per i vari menÃ¹
27.
28.         **PROTOTIPI**
29. void *crea_lista();//crea lista vuota
30. void Inserimento_testa(void **head); //inserimento in testa alla lista
31. void Eliminazione_testa(void **p_head); //eliminazione in testa alla lista
32.
33.         **MAIN**
34. int main()
35. {
36.     struct PERSONA *head;
37.
38.     head=(struct PERSONA *)crea_lista(); //crea lista inizialmente vuota
39.
40.     do
41.     {
42.         printf("*MENU*\n\n");
43.         printf(" [1] Inserire dato nella pila\n");
44.         printf(" [2] Eliminare dato dalla pila\n");
45.         printf(" [3] Esci\n");
46.         printf(" Inserire scelta: ");
47.         fflush(stdin);
48.         scanf("%hd",&scegli);
49.
50.         switch(scegli)
51.         {
52.             case 1: Inserimento_testa((void **)&head);break;
53.             case 2: Eliminazione_testa((void **)&head);break;
54.             case 3: exit(EXIT_FAILURE);break;
55.         }
56.     }while(scegli!=4);
```

```

57.     return 0;
58. }
59.
60.         /**FUNCTION*/
61. /**FUNCTION PER CREARE PILA*/
62. void *crea_lista()
63. {
64.     struct PERSONA //lista lineare generica
65.     {
66.         INFO_FIELD info;
67.         struct PERSONA *p_next;
68.     }*testa;
69.     testa=NULL;
70.     return testa;
71. }
72.
73. /**FUNCTION PER INSERIRE NODO IN TESTA*/
74. void Inserimento_testa(void **p_head)
75. {
76.     struct LISTA //lista lineare generica
77.     {
78.         INFO_FIELD info;
79.         struct LISTA *p_next;
80.     }*ptr,*pts,*p_dato;
81.
82.     p_dato=malloc(sizeof(struct LISTA));
83.     len_info(sizeof(struct LISTA));
84.
85.     printf("\n Inserisci nome: ");
86.     scanf(" %s",p_dato->info.nome);
87.
88.     /*ptr ora punterà a una struttura i cui membri info e p_next sono imposti a zero*/
89.     ptr=calloc(1,sizeof(struct LISTA)); //alloco 1 blocco di memoria e azzera tutti bit
90.     memcpy(ptr,p_dato,len_info);
91.
92.     ptr->p_next=(struct LISTA*)*p_head;
93.     *p_head=ptr;
94.     pts=*p_head;
95.
96.     printf("\n *PILA DOPO L'INSERIMENTO*\n");
97.     while (pts != NULL) /*mentre la lista non e' finita */
98.     {
99.         /*stampa il nodo corrente */
100.        printf(" Nome= %s\n", pts->info.nome);
101.        pts=pts->p_next; /*aggiorna il puntatore */
102.    }
103.    printf("\n\n");
104. }
105.
106. /**FUNCTION PER ELIMINARE IN TESTA*/
107. void Eliminazione_testa(void **p_head)
108. {
109.     struct LISTA //lista lineare generica
110.     {
111.         INFO_FIELD info;
112.         struct LISTA *p_next;
113.     }*ptr,*pts;
114.
115.     ptr=((struct LISTA *)*p_head)->p_next;
116.     free((struct LISTA *)*p_head); //dealloca la memoria puntata da *p_head
117.
118.     *p_head=ptr;
119.     pts=*p_head;
120.
121.     printf("\n *PILA DOPO L'ELIMINAZIONE*\n");
122.     while (pts != NULL) /*mentre la lista non e' finita */

```

```
123. {
124.     /*stampa il nodo corrente */
125.     printf(" Nome= %s\n",pts->info.nome);
126.     pts=pts->p_next; /*aggiorna il puntatore */
127. }
128. printf("\n\n");
129. }
```

## ESECUZIONE 1

\*MENU'\*

- [1] Inserire dato nella pila
  - [2] Eliminare dato dalla pila
  - [3] Esci
- Inserire scelta: 1

Inserisci nome: mattia

\*PILA DOPO L'INSERIMENTO\*

Nome= mattia

\*MENU'\*

- [1] Inserire dato nella pila
  - [2] Eliminare dato dalla pila
  - [3] Esci
- Inserire scelta: 1

Inserisci nome: angelo

\*PILA DOPO L'INSERIMENTO\*

Nome= angelo

Nome= mattia

\*MENU'\*

- [1] Inserire dato nella pila
  - [2] Eliminare dato dalla pila
  - [3] Esci
- Inserire scelta: 1

Inserisci nome: claudio

\*PILA DOPO L'INSERIMENTO\*

Nome= claudio

Nome= angelo

Nome= mattia

\*MENU'\*

- [1] Inserire dato nella pila
  - [2] Eliminare dato dalla pila
  - [3] Esci
- Inserire scelta: 1

Inserisci nome: mario

\*PILA DOPO L'INSERIMENTO\*

Nome= mario

Nome= claudio

Nome= angelo

Nome= mattia

```
*MENU'*  
  
[1] Inserire dato nella pila  
[2] Eliminare dato dalla pila  
[3] Esci  
Inserire scelta: 2  
  
*PILA DOPO L'ELIMINAZIONE*  
Nome= claudio  
Nome= angelo  
Nome= mattia  
  
*MENU'*  
  
[1] Inserire dato nella pila  
[2] Eliminare dato dalla pila  
[3] Esci  
Inserire scelta: 2  
  
*PILA DOPO L'ELIMINAZIONE*  
Nome= angelo  
Nome= mattia  
  
*MENU'*  
  
[1] Inserire dato nella pila  
[2] Eliminare dato dalla pila  
[3] Esci  
Inserire scelta: 2  
  
*PILA DOPO L'ELIMINAZIONE*  
Nome= mattia  
  
*MENU'*  
  
[1] Inserire dato nella pila  
[2] Eliminare dato dalla pila  
[3] Esci  
Inserire scelta: 2  
  
*PILA DOPO L'ELIMINAZIONE*  
  
*MENU'*  
  
[1] Inserire dato nella pila  
[2] Eliminare dato dalla pila  
[3] Esci  
Inserire scelta: 3  
  
Process returned 1 (0x1)  execution time : 19.234 s  
Press ENTER to continue.  
█
```

## Esercizio 45b[liv.1]

### Gesione Coda mediante menu'

```
1.  /***[liv.1] Realizzare in C le funzioni per la gestione, mediante menu', delle strutture dat
   i pila e coda mediante lista
2.  lineare dinamica e generica con [rispettivamente senza] nodo sentinella.*/
3.
4.
5.  ****[CODA]*****
6.
7.      **LIBRERIE**
8.  #include <stdio.h>
9.  #include <stdlib.h>
10. #include <string.h>
11.
12.
13. /*Il nodo sentinella serve per evitare di avere due function per l'inserimento e due functi
   on
14. per l'eliminazione. Infatti Ã" possibile avere un'unica sequenza di operazioni sia per gli
15. inserimenti/eliminazioni sulla testa che per quelli fatti sul nodo corrente.
16. Occorre semplicemente aggiugere un nodo fittizio che funga da testa ma che punti alla testa
17. reale della lista: un nodo sentinella.
18. Ogni modifica sulla testa comporterÃ in questo modo la sola alterazione del campo puntator
   e del
19. nodo sentinella. */
20.
21. typedef struct
22. {
23.     char nome[20]; //tipo del campo informazione del nodo della lista globale
24. }INFO_FIELD;
25.
26. short scegli=0,len_info; //variabile dichiarata globalmente per i vari menÃ¹
27.
28.         **PROTOTIPI**
29. void *crea_lista();//crea lista vuota
30. void Inserimento(void **head, void **p_punt); //inserimento in testa alla lista
31. void Eliminazione(void **p_head); //eliminazione nodo successore
32.
33.         **MAIN**
34. int main()
35. {
36.     struct PERSONA *head,*punt;
37.
38.     head=(struct PERSONA *)crea_lista();
39.     punt=(struct PERSONA *)crea_lista();
40.
41.     do
42.     {
43.         printf("*MENU*\n\n");
44.         printf(" [1] Inserire dato nella coda\n");
45.         printf(" [2] Eliminare dato dalla coda\n");
46.         printf(" [3] Esci\n");
47.         printf(" Inserire scelta: ");
48.         fflush(stdin);
49.         scanf("%hd",&scegli);
50.
51.         switch(scegli)
52.         {
53.             case 1: Inserimento((void **)&head,(void **)&punt);break;
54.             case 2: Eliminazione((void **)&head);break;
55.             case 3: exit(EXIT_FAILURE);break;
56.         }
57.     }
```

```

57.     }while(scegli!=4);
58.     return 0;
59. }
60.
61.         /**FUNCTION*/
62. /**FUNCTION PER CREARE LA CODA*/
63. void *crea_lista()
64. {
65.     struct PERSONA //lista lineare generica
66.     {
67.         INFO_FIELD info;
68.         struct PERSONA *p_next;
69.     }*testa;
70.     testa=NULL;
71.     return testa;
72. }
73.
74. /**FUNCTION PER INSERIRE NODO IN TESTA*/
75. void Inserimento(void **p_head, void **p_punt)
76. {
77.     struct LISTA //lista lineare generica
78.     {
79.         INFO_FIELD info;
80.         struct LISTA *p_next;
81.     }*ptr,*pts,*p_data;
82.
83.     p_data=malloc(sizeof(struct LISTA));
84.     len_info(sizeof(struct LISTA));
85.
86.     printf("\n Inserisci nome: ");
87.     scanf(" %s",p_data->info.nome);
88.
89.     /*ptr ora punterà a una struttura i cui membri info e p_next sono imposti a zero*/
90.     ptr=calloc(1,sizeof(struct LISTA)); //alloco 1 blocco di memoria e azzera tutti bit
91.     memcpy(ptr,p_data,len_info);
92.
93.     if((struct LISTA *)*p_head==NULL)
94.     {
95.         ptr->p_next=(struct LISTA*)*p_head;
96.         *p_head=ptr;
97.         *p_punt=ptr;
98.     }
99.     else
100.    {
101.        ptr->p_next=NULL;
102.        ((struct LISTA *)*p_punt)->p_next=ptr;
103.        *p_punt=ptr;
104.    }
105.
106.    pts=*p_head;
107.
108.    printf("\n *CODA DOPO L'INSERIMENTO*\n");
109.    while (pts != NULL) /*mentre la lista non e' finita */
110.    {
111.        /*stampa il nodo corrente */
112.        printf(" Nome= %s\n", pts->info.nome);
113.        pts=pts->p_next; /*aggiorna il puntatore */
114.    }
115.    printf("\n\n");
116. }
117.
118.
119. /**ELIMINA NODO SUCCESSORE*/
120. void Eliminazione(void **p_head)
121. {
122.     struct LISTA //lista lineare generica

```

```

123. {
124.     INFO_FIELD info;
125.     struct LISTA *p_next;
126. }*ptr;
127.
128. if(*p_head==NULL)
129. {
130.     printf("\n LA CODA E' VUOTA ");
131. }
132. else
133. {
134.     ptr=((struct LISTA *)*p_head)->p_next;
135.     ((struct LISTA*)*p_head)->p_next=ptr->p_next;
136.     free(ptr);
137.     *p_head=ptr;
138. }
139.
140. ptr=*p_head;
141.
142. printf("\n *CODA DOPO L'ELIMINAZIONE*\n");
143. while (ptr != NULL) /*mentre la lista non e' finita */
144. {
145.     /*stampa il nodo corrente */
146.     printf(" Nome= %s\n",ptr->info.nome);
147.     ptr=ptr->p_next; /*aggiorna il puntatore */
148. }
149. printf("\n\n");
150. }
```

## ESECUZIONE 1

\*MENU'\*

```
[1] Inserire dato nella coda  
[2] Eliminare dato dalla coda  
[3] Esci  
Inserire scelta: 1
```

Inserisci nome: mattia

\*CODA DOPO L'INSERIMENTO\*  
Nome= mattia

\*MENU'\*

```
[1] Inserire dato nella coda  
[2] Eliminare dato dalla coda  
[3] Esci  
Inserire scelta: 1
```

Inserisci nome: angelo

\*CODA DOPO L'INSERIMENTO\*  
Nome= mattia  
Nome= angelo

\*MENU'\*

```
[1] Inserire dato nella coda  
[2] Eliminare dato dalla coda  
[3] Esci  
Inserire scelta: 1
```

Inserisci nome: claudio

\*CODA DOPO L'INSERIMENTO\*  
Nome= mattia  
Nome= angelo  
Nome= claudio

\*MENU'\*

```
[1] Inserire dato nella coda  
[2] Eliminare dato dalla coda  
[3] Esci  
Inserire scelta: 1
```

Inserisci nome: mario

\*CODA DOPO L'INSERIMENTO\*  
Nome= mattia  
Nome= angelo  
Nome= claudio  
Nome= mario

```
*MENU'*  
[1] Inserire dato nella coda  
[2] Eliminare dato dalla coda  
[3] Esci  
Inserire scelta: 2
```

```
*CODA DOPO L'ELIMINAZIONE*  
Nome= angelo  
Nome= claudio  
Nome= mario
```

```
*MENU'*  
[1] Inserire dato nella coda  
[2] Eliminare dato dalla coda  
[3] Esci  
Inserire scelta: 2
```

```
*CODA DOPO L'ELIMINAZIONE*  
Nome= claudio  
Nome= mario
```

```
*MENU'*  
[1] Inserire dato nella coda  
[2] Eliminare dato dalla coda  
[3] Esci  
Inserire scelta: 2
```

```
*CODA DOPO L'ELIMINAZIONE*  
Nome= mario
```

```
*MENU'*  
[1] Inserire dato nella coda  
[2] Eliminare dato dalla coda  
[3] Esci  
Inserire scelta: 3
```

```
Process returned 1 (0x1)    execution time : 15.346 s  
Press ENTER to continue.  
■
```

## Esercizio 46a[liv.2]

### Gesione Lista circolare mediante menu'

```
1.  /**[liv.2] Realizzare in C le funzioni per la gestione, mediante menu', delle strutture dat
   i catena e lista
2.  bidirezionale mediante lista lineare dinamica e generica con [rispettivamente senza] nodo s
   entinella.*/
3.
4.
5.  ****LISTA CIRCOLARE*****
6.
7.          **LIBRERIE**
8.  #include <stdio.h>
9.  #include <stdlib.h>
10. #include <string.h>
11.
12.
13. /*Il nodo sentinella serve per evitare di avere due function per l'inserimento e due functi
   on
14. per l'eliminazione. Infatti Ã" possibile avere un'unica sequenza di operazioni sia per gli
15. inserimenti/eliminazioni sulla testa che per quelli fatti sul nodo corrente.
16. Occorre semplicemente aggiugere un nodo fittizio che funga da testa ma che punti alla testa

17. reale della lista: un nodo sentinella.
18. Ogni modifica sulla testa comporterÃ in questo modo la sola alterazione del campo puntator
   e del
19. nodo sentinella. */
20.
21. struct info
22. {
23.     int cnt;
24.     char nome[20];
25. };
26.
27. typedef struct info INFO_FIELD;
28.
29. struct nodo
30. {
31.     INFO_FIELD info;
32.     struct LISTA *p_next;
33. };
34.
35. typedef struct nodo NODO;
36.
37.
38.          **PROTOTIPI**
39. /**FUNZIONI PER LISTA CATENA CON SENTINELLA*/
40. void crea_lista_sent(NODO **p_head, INFO_FIELD **list);
41. void Inserimento(NODO **p_head,INFO_FIELD **list);
42. void Eliminazione(NODO **p_head,INFO_FIELD **list);
43. /**FUNZIONI PER LISTA CATENA SENZA SENTINELLA*/
44. void crea_lista(NODO **p_head,INFO_FIELD **list);
45. void Inserimento_testa(NODO **p_head,INFO_FIELD **list);
46. void Inserimento_mezzo(NODO **p_head,INFO_FIELD **list);
47. void Eliminazione_testa(NODO **p_head,INFO_FIELD **list);
48. void Eliminazione_mezzo(NODO **p_head, INFO_FIELD **list);
49. /**VISITA DELLA LISTA*/
50. void Visualizza(NODO **p_head,INFO_FIELD **list);
51.
52.
53.          **MAIN**
54. int main()
55. {
```

```

56.     short scegli=0;
57.     NODO *head;
58.     INFO_FIELD *list;
59.     head=malloc(sizeof(NODO));
60.     list=malloc(sizeof(INFO_FIELD));
61.
62.     printf("\n **LISTA CON STRUTTURE DATI CATENA**\n [1] Con sentinella\n [2] Senza sentine
63. lla\n -----> ");
64.     scanf("%hd",&scegli);
65.
66.     if(scegli == 1)
67.     {
68.         crea_lista_sent(&head,&list);
69.
70.         while(1)
71.         {
72.             printf("\n *****[MENU']*****\n [1] Inserimento\n [2] Eliminazione\n [
73. 3] Visualizza lista circolare\n [4] Exit\n *****\n -----> ");
74.             fflush(stdin);
75.             scanf("%hd",&scegli);
76.
77.             switch(scegli)
78.             {
79.                 case 1: Inserimento(&head,&list);break;
80.                 case 2: Eliminazione(&head,&list);break;
81.                 case 3: Visualizza(&head,&list);break;
82.                 case 4: exit(EXIT_FAILURE);
83.             }
84.         }
85.     }
86.     else if(scegli == 2)
87.     {
88.         crea_lista(&head,&list);
89.
90.         while(1)
91.         {
92.             printf("\n *****[MENU']*****\n [1] Inserimento in testa \n [2] Inseri
93. mento in mezzo\n [3] Eliminazione in testa\n [4] Eliminazione in mezzo\n [5] Visualizza lis
94. ta circolare\n [6] Exit\n *****\n -----> ");
95.             fflush(stdin);
96.             scanf("%hd",&scegli);
97.
98.             switch(scegli)
99.             {
100.                 case 1: Inserimento_testa(&head,&list);break;
101.                 case 2: Inserimento_mezzo(&head,&list);break;
102.                 case 3: Eliminazione_testa(&head,&list);break;
103.                 case 4: Eliminazione_mezzo(&head,&list);break;
104.                 case 5: Visualizza(&head,&list);break;
105.                 case 6: exit(EXIT_FAILURE);
106.             }
107.         }
108.     }
109.     return 0;
110. }
111. */
112. void crea_lista_sent(NODO **p_head, INFO_FIELD **list)
113. {
114.     strcpy((*p_head)->info.nome,"Sentinella");
115.     (*p_head)->p_next=NULL;
116.     (*list)->cnt=0;
117. }

```

```

118.
119. /*Function che inserisce nodo*/
120. void Inserimento(NODO **p_head, INFO_FIELD **list)
121. {
122.     char dato[20];
123.     NODO *punt;
124.     punt=malloc(sizeof(NODO));
125.
126.     printf("\n Inserisci nome: ");
127.     fflush(stdin);
128.     scanf("%s",dato);
129.
130.     strcpy(punt->info.nome,dato);
131.     punt->p_next=*p_head;
132.     *p_head=punt;
133.
134.     (*list)->cnt++;
135. }
136.
137. void Eliminazione(NODO **p_head,INFO_FIELD **list)
138. {
139.     NODO *ptr;
140.
141.     if((*p_head)->p_next == NULL) /*se Ã" presente un solo nodo nella lista */
142.     {
143.         free(*p_head);
144.         *p_head = NULL; /*la lista ora e' vuota*/
145.     }
146.     else
147.     {
148.         ptr=(*p_head)->p_next; /*salva il link del nodo puntato dal nodo di testa */
149.         free(*p_head); /*libera l'area puntata da p_head */
150.         *p_head = ptr; /*aggiorna il puntatore alla testa della lista mediante il link salvato*/
151.     }
152.
153.     (*list)->cnt--;
154. }
155. /**FINE FUNZIONI CON NODO SENTINELLA*/
156.
157. ****
158.
159. /**INIZIO FUNZIONE SENZA NODO SENTINELLA*/
160. /*Function che crea lista con nodo sentinella*/
161. void crea_lista(NODO **p_head,INFO_FIELD **list)
162. {
163.     *p_head=NULL;
164.     (*list)->cnt=0;
165. }
166.
167. void Inserimento_testa(NODO **p_head,INFO_FIELD **list)
168. {
169.     NODO *new_nodo;
170.     new_nodo=malloc(sizeof(NODO));
171.
172.     printf("\n Inserisci nome del nuovo nodo da inserire in testa: ");
173.     scanf("%s",new_nodo->info.nome);
174.
175.     new_nodo->p_next=*p_head;
176.     *p_head=new_nodo;
177.
178.     (*list)->cnt++;
179. }
180.
181. void Inserimento_mezzo(NODO **p_head,INFO_FIELD **list)
182. {

```

```

183.     NODO *new_nodo;
184.
185.     new_nodo=malloc(sizeof(NODO));
186.
187.     printf("\n Inserisci nome del nuovo nodo da inserire: ");
188.     scanf("%s",new_nodo->info.nome);
189.
190.     new_nodo->p_next=(*p_head)->p_next;
191.     (*p_head)->p_next=new_nodo;
192.
193.     (*list)->cnt++;
194. }
195.
196. void Eliminazione_testa(NODO **p_head,INFO_FIELD **list)
197. {
198.     NODO *ptr;
199.     ptr=malloc(sizeof(NODO));
200.
201.     if((*p_head)->p_next == NULL) /*se Ã¨ presente un solo nodo nella lista */
202.     {
203.         free(*p_head);
204.         *p_head = NULL; /*la lista ora e' vuota*/
205.     }
206.     else
207.     {
208.         ptr=(*p_head)->p_next; /*salva il link del nodo puntato dal nodo di testa */
209.         free(*p_head); /*libera l'area puntata da p_head */
210.         *p_head = ptr; /*aggiorna il puntatore alla testa della lista mediante il link salvato*/
211.     }
212.
213.     (*list)->cnt--;
214. }
215.
216. void Eliminazione_mezzo(NODO **p_head, INFO_FIELD **list)
217. {
218.     NODO *ptr;
219.     ptr=malloc(sizeof(NODO));
220.
221.     ptr=(*p_head)->p_next;
222.     (*p_head)->p_next = ptr->p_next;
223.     free(ptr);
224.
225.     (*list)->cnt--;
226. }
227. /**FINE FUNZIONI SENZA NODO SENTINELLA*/
228.
229. /*Function che visualizza lista mediante visita*/
230. void Visualizza(NODO **p_head,INFO_FIELD **list)
231. {
232.     int i;
233.     NODO *punt;
234.     punt=malloc(sizeof(NODO));
235.
236.     if(*p_head==NULL)
237.     {
238.         printf("\n *LISTA VUOTA*\n\n");
239.     }
240.     else
241.     {
242.         punt=*p_head;
243.         printf("\n *LISTA CIRCOLARE*\n");
244.         for(i=0; i<(*list)->cnt; i++) /*mentre la lista non e' finita */
245.         {
246.             /*stampa il nodo corrente */
247.             printf("Nodo[%d] -> Nome= %s\n", i,punt->info.nome);

```

```
248.         punt=punt->p_next; /*aggiorna il puntatore */
249.     }
250.     printf("\n\n");
251. }
252. }
```

## ESECUZIONE 1 (con sentinella)

\*\*LISTA CON STRUTTURE DATI CATENA\*\*

- [1] Con sentinella
- [2] Senza sentinella

-----> 1

\*\*\*\*\*[MENU']\*\*\*\*\*

- [1] Inserimento
- [2] Eliminazione
- [3] Visualizza lista circolare
- [4] Exit

\*\*\*\*\*

-----> 1

Inserisci nome: mattia

\*\*\*\*\*[MENU']\*\*\*\*\*

- [1] Inserimento
- [2] Eliminazione
- [3] Visualizza lista circolare
- [4] Exit

\*\*\*\*\*

-----> 1

Inserisci nome: angelo

\*\*\*\*\*[MENU']\*\*\*\*\*

- [1] Inserimento
- [2] Eliminazione
- [3] Visualizza lista circolare
- [4] Exit

\*\*\*\*\*

-----> 3

\*LISTA CIRCOLARE\*

Nodo[0] -> Nome= angelo

Nodo[1] -> Nome= mattia

```
*****[MENU']*****
[1] Inserimento
[2] Eliminazione
[3] Visualizza lista circolare
[4] Exit
*****  
-----> 2
```

```
*****[MENU']*****
[1] Inserimento
[2] Eliminazione
[3] Visualizza lista circolare
[4] Exit
*****  
-----> 3
```

```
*LISTA CIRCOLARE*
Nodo [0] -> Nome= mattia
```

```
*****[MENU']*****
[1] Inserimento
[2] Eliminazione
[3] Visualizza lista circolare
[4] Exit
*****  
-----> 2
```

```
*****[MENU']*****
[1] Inserimento
[2] Eliminazione
[3] Visualizza lista circolare
[4] Exit
*****  
-----> 3
```

```
*LISTA CIRCOLARE*
```

```
*****[MENU']*****
[1] Inserimento
[2] Eliminazione
[3] Visualizza lista circolare
[4] Exit
*****  
-----> 4
```

```
Process returned 1 (0x1)    execution time : 32.016 s
Press ENTER to continue.
```

## **ESECUZIONE 2 (senza sentinella)**

```
**LISTA CON STRUTTURE DATI CATENA**
[1] Con sentinella
[2] Senza sentinella
-----> 2
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Eliminazione in testa
[4] ELiminazione in mezzo
[5] Visualizza lista circolare
[6] Exit
*****-> 1
```

Inserisci nome del nuovo nodo da inserire in testa: mattia

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Eliminazione in testa
[4] ELiminazione in mezzo
[5] Visualizza lista circolare
[6] Exit
*****-> 1
```

Inserisci nome del nuovo nodo da inserire in testa: angelo

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Eliminazione in testa
[4] ELiminazione in mezzo
[5] Visualizza lista circolare
[6] Exit
*****-> 2
```

Inserisci nome del nuovo nodo da inserire: claudio

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Eliminazione in testa
[4] ELiminazione in mezzo
[5] Visualizza lista circolare
[6] Exit
*****-> 5
```

```
*LISTA CIRCOLARE*
Nodo[0] -> Nome= angelo
Nodo[1] -> Nome= claudio
Nodo[2] -> Nome= mattia
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Eliminazione in testa
[4] ELiminazione in mezzo
[5] Visualizza lista circolare
[6] Exit
*****  
-----> 4
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Eliminazione in testa
[4] ELiminazione in mezzo
[5] Visualizza lista circolare
[6] Exit
*****  
-----> 5
```

```
*LISTA CIRCOLARE*
Nodo[0] -> Nome= angelo
Nodo[1] -> Nome= mattia
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Eliminazione in testa
[4] ELiminazione in mezzo
[5] Visualizza lista circolare
[6] Exit
*****  
-----> 3
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Eliminazione in testa
[4] ELiminazione in mezzo
[5] Visualizza lista circolare
[6] Exit
*****  
-----> 5
```

```
*LISTA CIRCOLARE*
Nodo[0] -> Nome= mattia
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Eliminazione in testa
[4] ELiminazione in mezzo
[5] Visualizza lista circolare
[6] Exit
*****  
-----> 6
```

```
Process returned 1 (0x1)  execution time : 47.012 s
Press ENTER to continue.
```

## Esercizio 46b[liv.2]

### Gesione Lista bidirezionale mediante menù

```
1.  /**[liv.2] Realizzare in C le funzioni per la gestione, mediante menu', delle strutture dat
   i catena e lista
2.  bidirezionale mediante lista lineare dinamica e generica con [rispettivamente senza] nodo s
   entinella.*/
3.
4.
5.  ****LISTA BIDIREZIONALE*****
6.
7.          **LIBRERIE**
8.  #include <stdio.h>
9.  #include <stdlib.h>
10. #include <string.h>
11.
12.
13. /*Il nodo sentinella serve per evitare di avere due function per l'inserimento e due functi
   on
14. per l'eliminazione. Infatti Ã" possibile avere un'unica sequenza di operazioni sia per gli
15. inserimenti/eliminazioni sulla testa che per quelli fatti sul nodo corrente.
16. Occorre semplicemente aggiugere un nodo fittizio che funga da testa ma che punti alla testa
   reale della lista: un nodo sentinella.
17. Ogni modifica sulla testa comporterÃ in questo modo la sola alterazione del campo puntator
   e del
18. nodo sentinella. */
19.
20.
21.
22. struct info
23. {
24.     int cnt;
25.     char nome[50];
26. };
27.
28. typedef struct info INFO_FIELD;
29.
30. struct nodo
31. {
32.     struct nodo *p_prev; //puntatore al primo nodo della lista
33.     INFO_FIELD info;
34.     struct nodo *p_next; //puntatore all'ultimo nodo della lista
35. };
36.
37. typedef struct nodo NODO;
38.
39.
40.          **PROTOTIPI**
41. /**FUNZIONI CON NODI SENTINELLA*/
42. void Crea_lista(NODO **p_head, NODO **p_tail, INFO_FIELD **list);
43. void Inserimento(NODO **p_head, NODO **p_tail, INFO_FIELD **list);
44. void Eliminazione(NODO **p_head, NODO **p_tail, INFO_FIELD **list);
45. /**FUNZIONI SENZA NODO SENTINELLA*/
46. void crea_lista(NODO **p_tail, NODO **p_head, INFO_FIELD **list);
47. void Inserimento_testa(NODO **p_head, NODO **p_tail, INFO_FIELD **list);
48. void Inserimento_mezzo(NODO **p_tail, INFO_FIELD **list);
49. void Eliminazione_testa(NODO **p_head, INFO_FIELD **list);
50. void Eliminazione_mezzo(NODO **p_head, INFO_FIELD **list);
51. /**VISITA DELLA LISTA*/
52. void Visualizza(NODO **p_head, INFO_FIELD **list);
53.
54.          **MAIN**
55. int main()
```

```

56. {
57.     short scegli=0;
58.     NODO *head,*tail;
59.     INFO_FIELD *list;
60.
61.     head=malloc(sizeof(NODO));
62.     tail=malloc(sizeof(NODO));
63.     list=malloc(sizeof(INFO_FIELD));
64.
65.     printf("\n **LISTA BIDIREZIONALE **\n [1] Con sentinella\n [2] Senza sentinella\n -----
66.           -----> ");
67.     scanf("%hd",&scegli);
68.
69.     if(scegli == 1)
70.     {
71.         Crea_lista(&head,&tail,&list);
72.         while(1)
73.         {
74.             printf("\n *****[MENU']*****\n [1] Inserimento\n [2] Eliminazione\n [
75.                 3] Visualizza lista bidirezionale\n [4] Exit\n *****\n -----
76.             > ");
77.             fflush(stdin);
78.             switch(scegli)
79.             {
80.                 case 1: Inserimento(&head,&tail,&list);break;
81.                 case 2: Eliminazione(&head,&tail,&list);break;
82.                 case 3: Visualizza(&head,&list);break;
83.                 case 4: exit(EXIT_FAILURE);
84.             }
85.         }
86.     }
87.     else if(scegli == 2)
88.     {
89.         crea_lista(&head,&tail,&list);
90.
91.         while(1)
92.         {
93.             printf("\n *****[MENU']*****\n [1] Inserimento in testa \n [2] Inseri-
94.               mento in mezzo\n [3] Elimina nodo in testa\n [4] Elimina nodo in mezzo\n [5] Visualizza lis-
95.               ta bidirezionale\n [6] Exit\n *****\n -----
96.             > ");
97.             fflush(stdin);
98.             switch(scegli)
99.             {
100.                 case 1: Inserimento_testa(&head,&tail,&list);break;
101.                 case 2: Inserimento_mezzo(&tail,&list);break;
102.                 case 3: Eliminazione_testa(&head,&list);break;
103.                 case 4: Eliminazione_mezzo(&head,&list);break;
104.                 case 5: Visualizza(&head,&list);break;
105.             }
106.         }
107.     }
108.
109.     return 0;
110. }
111.
112.         /**FUNCTION*/
113. /*INIZIO FUNZIONI CON NODI SENTINELLA*/
114. /*Function che crea la lista iniziale con i due nodi sentinella che puntano inizialmente he-
115. ad->tail e tail->head*/
115. void Crea_lista(NODO **p_head, NODO **p_tail, INFO_FIELD **list)

```

```

116. {
117.     (*list)->cnt=0;
118.     strcpy((*p_head)->info.nome,"Sentinella Head");
119.     (*p_head)->p_next=*p_tail;
120.     strcpy((*p_tail)->info.nome,"Sentinella Tail");
121.     (*p_tail)->p_prev=*p_head;
122. }
123.
124. /*Function che ci permette di inserire nodi nella lista*/
125. void Inserimento(NODO **p_head, NODO **p_tail, INFO_FIELD **list)
126. {
127.     char dato[50];
128.     NODO *punt;
129.     punt=malloc(sizeof(NODO));
130.
131.     printf("\n Inserisci nome: ");
132.     fflush(stdin);
133.     scanf("%s",dato);
134.
135.     strcpy(punt->info.nome,dato);
136.
137.     punt->p_prev=*p_head;
138.     (*p_head)->p_next=punt;
139.
140.     punt->p_next=*p_tail;
141.     *p_tail=punt;
142.
143.     (*list)->cnt++;
144. }
145.
146. /*Function che elimina nodo all'interno della lista*/
147. void Eliminazione(NODO **p_head, NODO **p_tail, INFO_FIELD **list)
148. {
149.     NODO *punt;
150.     punt=malloc(sizeof(NODO));
151.
152.     if(*p_head==NULL)
153.     {
154.         printf("\n *LA LISTA E' VUOTA*\n\n");
155.     }
156.     else
157.     {
158.         punt=(*p_tail);
159.         punt->p_prev->p_next=punt->p_next;
160.         punt->p_next->p_prev=punt->p_prev;
161.         *p_tail=punt->p_next;
162.         free(punt);
163.     }
164.     (*list)->cnt--;
165. }
166. /**FINE FUNZIONI CON NODI SENTINELLA*/
167.
168. ****
169.
170. /**INIZIO FUNZIONI SENZA NODI SENTINELLA*/
171. /**
172. void crea_lista(NODO **p_tail, NODO **p_head, INFO_FIELD **list)
173. {
174.     (*list)->cnt=0;
175.     *p_head=NULL;
176.     *p_tail=NULL;
177. }
178.
179. /**
180. void Inserimento_testa(NODO **p_head, NODO **p_tail, INFO_FIELD **list)
181. {

```

```

182.     NODO *new_nodo;
183.
184.     new_nodo=malloc(sizeof(NODO));
185.
186.     printf("\n Inserisci nome del nuovo nodo da inserire: ");
187.     scanf("%s",new_nodo->info.nome);
188.
189.     new_nodo->p_prev=*p_head;
190.     new_nodo->p_next=*p_tail;
191.     *p_head=new_nodo;
192.     *p_tail=new_nodo;
193.
194.     (*list)->cnt++;
195. }
196.
197. /**
198. void Inserimento_mezzo(NODO **p_tail, INFO_FIELD **list)
199. {
200.     NODO *new_nodo;
201.
202.     new_nodo=malloc(sizeof(NODO));
203.
204.     printf("\n Inserisci nome del nuovo nodo da inserire: ");
205.     scanf("%s",new_nodo->info.nome);
206.
207.     new_nodo->p_next=(*p_tail)->p_next;
208.     (*p_tail)->p_next=new_nodo;
209.     *p_tail=new_nodo;
210.
211.     (*list)->cnt++;
212. }
213.
214. /**
215. /*Function che elimina nodo in testa*/
216. void Eliminazione_testa(NODO **p_head, INFO_FIELD **list)
217. {
218.     NODO *punt;
219.     punt=malloc(sizeof(NODO));
220.
221.     if((*p_head)->p_next == NULL)
222.     {
223.         free(*p_head);
224.         *p_head=NULL;
225.     }
226.     else
227.     {
228.         punt=(*p_head)->p_next;
229.         free(*p_head);
230.         *p_head=punt;
231.     }
232.     (*list)->cnt--;
233. }
234.
235. /*Function che elimina nodo in mezzo*/
236. void Eliminazione_mezzo(NODO **p_head, INFO_FIELD **list)
237. {
238.     NODO *punt;
239.     punt=malloc(sizeof(NODO));
240.
241.     if((*p_head)->p_next == NULL)
242.     {
243.         free(*p_head);
244.         *p_head=NULL;
245.     }
246.     else
247.     {

```

```
248.     punt=(*p_head)->p_next;
249.     (*p_head)->p_next=punt->p_next;
250.     free(punt);
251. }
252. (*list)->cnt--;
253. }
254.
255. /**FINE FUNZIONI SENZA NODI SENTINELLA*/
256.
257. /*Function che visualizza lista mediante visita*/
258. void Visualizza(NODO **p_head, INFO_FIELD **list)
259. {
260.     NODO *punt;
261.     punt=malloc(sizeof(NODO));
262.
263.     if((*list)->cnt == 0)
264.     {
265.         printf("\n *LISTA VUOTA*\n\n");
266.     }
267.     else
268.     {
269.         punt=*p_head;
270.         printf("\n *LISTA BIDIREZIONALE*\n");
271.         while (punt != NULL) /*mentre la lista non e' finita */
272.         {
273.             /*stampa il nodo corrente */
274.             printf(" Nome= %s\n", punt->info.nome);
275.             punt=punt->p_next; /*aggiorna il puntatore */
276.         }
277.         printf("\n\n");
278.     }
279. }
```

## ESECUZIONE 1 (con sentinella)

```
**LISTA BIDIREZIONALE **
[1] Con sentinella
[2] Senza sentinella
-----> 1

*****[MENU']*****
[1] Inserimento
[2] Eliminazione
[3] Visualizza lista bidirezionale
[4] Exit
***** 
-----> 1

Inserisci nome: mattia

*****[MENU']*****
[1] Inserimento
[2] Eliminazione
[3] Visualizza lista bidirezionale
[4] Exit
***** 
-----> 1

Inserisci nome: angelo

*****[MENU']*****
[1] Inserimento
[2] Eliminazione
[3] Visualizza lista bidirezionale
[4] Exit
***** 
-----> 1

Inserisci nome: claudio

*****[MENU']*****
[1] Inserimento
[2] Eliminazione
[3] Visualizza lista bidirezionale
[4] Exit
***** 
-----> 3

*LISTA BIDIREZIONALE*
Nome= Sentinella Head
Nome= claudio
Nome= angelo
Nome= mattia
Nome= Sentinella Tail
```

```
***** [MENU'] *****  
[1] Inserimento  
[2] Eliminazione  
[3] Visualizza lista bidirezionale  
[4] Exit  
*****  
-----> 2
```

```
***** [MENU'] *****  
[1] Inserimento  
[2] Eliminazione  
[3] Visualizza lista bidirezionale  
[4] Exit  
*****  
-----> 2
```

```
***** [MENU'] *****  
[1] Inserimento  
[2] Eliminazione  
[3] Visualizza lista bidirezionale  
[4] Exit  
*****  
-----> 3
```

```
*LISTA BIDIREZIONALE*  
Nome= Sentinella Head  
Nome= mattia  
Nome= Sentinella Tail
```

```
***** [MENU'] *****  
[1] Inserimento  
[2] Eliminazione  
[3] Visualizza lista bidirezionale  
[4] Exit  
*****  
-----> 4
```

```
Process returned 1 (0x1)    execution time : 27.619 s  
Press ENTER to continue.
```

## ESECUZIONE 2 (senza sentinella)

```
**LISTA BIDIREZIONALE **
[1] Con sentinella
[2] Senza sentinella
-----> 2

*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Elimina nodo in testa
[4] Elimina nodo in mezzo
[5] Visualizza lista bidirezionale
[6] Exit
*****> 1

Inserisci nome del nuovo nodo da inserire: mattia

*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Elimina nodo in testa
[4] Elimina nodo in mezzo
[5] Visualizza lista bidirezionale
[6] Exit
*****> 1

Inserisci nome del nuovo nodo da inserire: angelo

*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Elimina nodo in testa
[4] Elimina nodo in mezzo
[5] Visualizza lista bidirezionale
[6] Exit
*****> 2

Inserisci nome del nuovo nodo da inserire: claudio

*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Elimina nodo in testa
[4] Elimina nodo in mezzo
[5] Visualizza lista bidirezionale
[6] Exit
*****> 5

*LISTA BIDIREZIONALE*
Nome= angelo
Nome= claudio
Nome= mattia
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Elimina nodo in testa
[4] Elimina nodo in mezzo
[5] Visualizza lista bidirezionale
[6] Exit
*****
```

```
-----> 4
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Elimina nodo in testa
[4] Elimina nodo in mezzo
[5] Visualizza lista bidirezionale
[6] Exit
*****
```

```
-----> 5
```

```
*LISTA BIDIREZIONALE*
```

```
Nome= angelo
```

```
Nome= mattia
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Elimina nodo in testa
[4] Elimina nodo in mezzo
[5] Visualizza lista bidirezionale
[6] Exit
*****
```

```
-----> 3
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Elimina nodo in testa
[4] Elimina nodo in mezzo
[5] Visualizza lista bidirezionale
[6] Exit
*****
```

```
-----> 5
```

```
*LISTA BIDIREZIONALE*
```

```
Nome= mattia
```

```
*****[MENU']*****
[1] Inserimento in testa
[2] Inserimento in mezzo
[3] Elimina nodo in testa
[4] Elimina nodo in mezzo
[5] Visualizza lista bidirezionale
[6] Exit
*****
```

```
-----> 6
```

```
Process returned 1 (0x1)  execution time : 30.338 s
Press ENTER to continue.
■
```

## Esercizio 47[liv.2]

### Coustruzione e visita albero qualsiasi mediante array

```
1.  /***[liv.2] Scrivere function C per la costruzione e visita per livelli di un albero qualsiasi rappresentato mediante array.
2.  [Suggerimento: la struct che definisce il generico nodo dell'albero, come nella figura sotto, deve contenere i seguenti campi:
3.  l'informazione, il grado del nodo ed un array di puntatori (ai nodi figli) di dimensione pari al massimo grado dei nodi, che si suppone noto]*/
4.
5.
6.          /**LIBRERIE*/
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10.
11. #define SIZE 20
12.
13. typedef struct
14. {
15.     char nome;
16. }INFO_FIELD;
17.
18. struct nodo
19. {
20.     INFO_FIELD info;
21.     int grado;
22. };
23.
24. typedef struct nodo NODO;
25.
26.          /**PROTOTIPI*/
27. void Costruisci_albero(NODO albero[]);
28. void Visita_albero();
29. void enqueue(NODO coda[], char info, int grado, int *tail);
30. void dequeue(NODO coda[], char *info, int *grado, int *head);
31.
32.          /**MAIN*/
33. int main()
34. {
35.     NODO albero[SIZE];
36.
37.     Costruisci_albero(albero);
38.     Visita_albero();
39.
40.     return 0;
41. }
42.
43.          /**FUNCTION*/
44. void Costruisci_albero(NODO albero[])
45. {
46.     int i=0,k,grado_nodo,head=0,teal=0,n;
47.     char info_nome;
48.
49.     NODO coda[SIZE];
50.
51.     printf("\n Inserire il numero di nodi da inserire: ");
52.     scanf("%d",&n);
53.
54.     printf("\n Inserire nome della radice [%d]: ",i);
55.     scanf("%s",&albero[i].info.nome);
```

```

56.     printf(" Inserire grado della radice: ");
57.     scanf("%d",&albero[i].grado);
58.
59.     //visita la radice
60.     printf("\n *VISITA LA RADICE*");
61.     printf("\n Radice: %c \n",albero[i].info.nome);
62.
63.     //Inserisce radice nella coda
64.     enqueue(coda,albero[i].info.nome,albero[i].grado,&teal);
65.
66.     //finche' la coda non e' vuota
67.     while (teal != head)
68.     {
69.         //grazie a dequeue ottengo il grado_nodo del nodo precedentemente inserito
70.         dequeue(coda,&info_nome,&grado_nodo,&head);
71.
72.         for(k=0; k<grado_nodo; k++)
73.         {
74.             printf("\n Inserire Figlio [%d] di [%c]: ",k+1,info_nome);
75.
76.             printf("\n Inserire nome Figlio: ");
77.             scanf("%s",&albero[i].info.nome);
78.             printf(" Inserire grado Figlio: ");
79.             scanf("%d",&albero[i].grado);
80.             printf("\n");
81.             enqueue(coda,albero[i].info.nome,albero[i].grado,&teal);
82.         }
83.     }
84.     printf("\n\n");
85. }
86.
87. void Visita_albero()
88. {
89.     int i=0,k,grado_nodo,head=0,teal=0;
90.     char info_nome;
91.
92.     NODO albero[SIZE];
93.     NODO coda[SIZE];
94.
95.     //visita la radice
96.     printf("\n *VISITA LA RADICE*");
97.     printf("\n Radice: %c ",albero[i].info.nome);
98.
99.     //Inserisce radice nella coda
100.    enqueue(coda,albero[i].info.nome,albero[i].grado,&teal);
101.
102.    //finche' la coda non e' vuota
103.    while (teal != head)
104.    {
105.        dequeue(coda,&info_nome,&grado_nodo,&head);
106.
107.        for(k=0; k<grado_nodo; k++)
108.        {
109.            i++;
110.            printf("\n %d: %c",i,albero[i].info.nome);
111.            enqueue(coda,albero[i].info.nome,albero[i].grado,&teal);
112.        }
113.    }
114.    printf("\n\n");
115. }
116.
117. void enqueue(NODO coda[], char info, int grado, int *tail)
118. {
119.     coda[*tail].info.nome=info;
120.     coda[*tail].grado=grado;
121.     *tail+=1;

```

```
122. }
123.
124. void dequeue(NODO coda[], char *info, int *grado, int *head)
125. {
126.     *info=coda[*head].info.nome;
127.     *grado=coda[*head].grado;
128.     *head+=1;
129. }
```

## ESECUZIONE 1

Inserire il numero di nodi da inserire: 10

Inserire nome della radice [0]: G  
Inserire grado della radice: 3

\*VISITA LA RADICE\*  
Radice: G

Inserire Figlio [1] di [G]:  
Inserire nome Figlio: H  
Inserire grado Figlio: 1

Inserire Figlio [2] di [G]:  
Inserire nome Figlio: F  
Inserire grado Figlio: 2

Inserire Figlio [3] di [G]:  
Inserire nome Figlio: N  
Inserire grado Figlio: 3

Inserire Figlio [1] di [H]:  
Inserire nome Figlio: P  
Inserire grado Figlio: 0

Inserire Figlio [1] di [F]:  
Inserire nome Figlio: O  
Inserire grado Figlio: 0

Inserire Figlio [2] di [F]:  
Inserire nome Figlio: L  
Inserire grado Figlio: 0

Inserire Figlio [1] di [N]:  
Inserire nome Figlio: A  
Inserire grado Figlio: 0

Inserire Figlio [2] di [N]:  
Inserire nome Figlio: K  
Inserire grado Figlio: 0

Inserire Figlio [3] di [N]:  
Inserire nome Figlio: Z  
Inserire grado Figlio: 0

\*VISITA LA RADICE\*  
Radice: G  
1: H  
2: F  
3: N  
4: P  
5: O  
6: L  
7: A  
8: K  
9: Z

Process returned 0 (0x0) execution time : 64.325 s  
Press ENTER to continue.

## Esercizio 48[liv.3]

# Costruzione e visita albero qualsiasi mediante liste multiple

```
1.  /***[liv.3] Scrivere function C per la costruzione e visita per livelli di un albero qualsiasi rappresentato mediante liste multiple.
2.  [Suggerimento: i puntatori ai figli risiedono in un array dinamicamente allocato e indirizzato dall'unico campo puntatore del nodo
3.  dell'albero (pt_figli) come nella figura che segue]*/
4.
5.          /**LIBRERIE*/
6. #include <stdio.h>
7. #include <stdlib.h>
8.
9. //DICHIAZIONE DEL NUOVO TIPO DI DATO boolean.
10. typedef enum{falso,vero} boolean;
11.
12.
13. //STRUTTURA UTILIZZATA PER MEMORIZZARE LE INFORMAZIONI DEI NODI DELL'ALBERO.
14. struct info
15. {
16.     char d;
17.     short grado;
18. };
19. typedef struct info INFO;
20.
21. //STRUTTURA ALBERO GENERICO MEDIANTE LISTE MULTIPLE.
22. struct node
23. {
24.     INFO dato;
25.     struct node *son;
26.     struct node *bro;
27. };
28. typedef struct node node;
29. typedef node *NODE;
30.
31. //STRUTTURA CODA MEDIANTE LISTE MULTIPLE.
32. struct coda
33. {
34.     short cnt;
35.     NODE front;
36.     NODE rear;
37. };
38. typedef struct coda coda;
39. typedef coda *CODA;
40.
41.
42.
43.          /**PROTOTIPI*/
44. void initialize(CODA);
45. boolean vuoto(const CODA);
46. void enqueue(INFO,CODA);
47. INFO dequeue(CODA);
48. void init_node(NODE,INFO);
49. INFO set_info(INFO,short);
50. void add_nodo(INFO,NODE *);
51. NODE create_tree(NODE,CODA);
52. void visita_tree(NODE);
53.
54.
55.          /**MAIN*/
56. int main()
```

```

57. {
58.     NODE tree,punt;
59.     CODA queue;
60.     INFO info;
61.     /** ALLOCAZIONE DI MEMORIA UTILIZZATA PER LE STRUTTURE DATI DINAMICHE. ***/
62.     punt=(NODE)malloc(sizeof(node));
63.     queue=(CODA)malloc(sizeof(coda));
64.     initialize(queue);
65.
66.     /** INSERIMENTO RADICE DELL'ALBERO. ***/
67.     printf("\t\t** INSERISCI LA RADICE DELL'ALBERO **\n\t\t->");
68.     fflush(stdin); scanf("%c",&info.d);
69.     printf("\t\t** INSERISCI IL GRADO DELLA RADICE **\n\t\t->");
70.     fflush(stdin); scanf("%hd",&info.grado);
71.     enqueue(info,queue);
72.     init_node(tree,info);
73.
74.     /** COSTRUZIONE ALBERO GENERICO MEDIANTE LISTE MULTIPLE. ***/
75.     punt=create_tree(tree,queue);
76.
77.     /** VISITA ALBERO GENERICO MEDIANTE LISTE MULTIPLE. ***/
78.     printf("\n\n");
79.     visita_tree(punt);
80.
81.     /** DEALLOCO LA MEMORIA HEAP OCCUPATA. ***/
82.     free(punt);
83.     free(tree);
84.     free(queue);
85.     return 0;
86. }
87.
88.
89.         /**FUNCTION*/
90. /** FUNZIONE PER L'INIZIALIZZAZIONE DEI NODI DELL'ALBERO. */
91. void init_node(NODE tree,INFO info)
92. {
93.     tree->dato.d=info.d;
94.     tree->dato.grado=info.grado;
95.     tree->son=NULL;
96.     tree->bro=NULL;
97. }
98.
99. /** FUNZIONE PER L'ACQUISIZIONE DELLE INFORMAZIONI CONTENUTE NEI NODI. */
100. INFO set_info(INFO nodo,short i)
101. {
102.     INFO info;
103.     printf("\t\t** INSERISCI %d FIGLIO DI %c **\n\t\t->",i+1,nodo.d);
104.     fflush(stdin);scanf("%c",&info.d);
105.     printf("\t\t** INSERISCI IL GRADO DI %c **\n\t\t->",info.d);
106.     fflush(stdin); scanf("%hd",&info.grado);
107.     return info;
108. }
109.
110. /** FUNZIONE PER L'INSERIMENTO DI UN NUOVO NODO IN UNA LISTA. */
111. void add_nodo(INFO info, NODE *head)
112. {
113.     NODE pnt;
114.     pnt=(NODE)calloc(1,sizeof(node));
115.     init_node(pnt,info);
116.     (*head)->bro=pnt;
117.     pnt->bro=NULL;
118.     *head=pnt;
119. }
120.
121. /** FUNZIONE PER LA CREAZIONE DI UN ALBERO GENERICO RAPPRESENTATO MEDIANTE LISTE MULTIPLE.
122. */

```

```

122. NODE create_tree(NODE tree,CODA queue)
123. {
124.     NODE tmp,pnt,punt;
125.     INFO info,temp;
126.     short i,grado=0;
127.     punt=(NODE)malloc(sizeof(node));
128.     pnt=(NODE)malloc(sizeof(node));
129.     punt=tree;
130.     while(!vuoto(queue))
131.     {
132.         tmp=(NODE)malloc(sizeof(node));
133.         info=dequeue(queue);
134.         init_node(tmp,info);
135.         grado=info.grado;
136.         pnt=tmp;
137.         for(i=0;i<grado;i++)
138.         {
139.             temp=set_info(info,i);
140.             add_nodo(temp,&pnt);
141.             enqueue(temp,queue);
142.         }
143.
144.         if(grado>0)
145.         {
146.             tree->son=pnt;
147.             tree=pnt;
148.         }
149.     }
150.     free(tmp);
151.     free(pnt);
152.     return punt;
153. }
154.
155. /** FUNZIONE PER LA VISITA PER LIVELLI DI UN ALBERO GENERICO RAPPRESENTATO MEDIANTE LISTE MULTIPLE. */
156. void visita_tree(NODE tree)
157. {
158.     NODE tmp;
159.     tmp=(NODE)malloc(sizeof(node));
160.     tmp=tree;
161.     while(tree->son!=NULL)
162.     {
163.         while(tmp!=NULL)
164.         {
165.             printf("\t%c ",tmp->dato.d);
166.             tmp=tmp->bro;
167.         }
168.         tree=tree->son;
169.         tmp=tree->bro;
170.     }
171.     while(tmp!=NULL)
172.     {
173.         printf("\t%c ",tmp->dato.d);
174.         tmp=tmp->bro;
175.     }
176. }
177.
178. /** FUNZIONE PER L'INIZIALIZZAZIONE DI UNA CODA RAPPRESENTATA MEDIANTE LISTE MULTIPLE. */
179. void initialize(CODA c)
180. {
181.     c->cnt=0;
182.     c->front=NULL;
183.     c->rear=NULL;
184. }
185.

```

```

186. /** FUNZIONE CHE CONTROLLA SE LA CODA E' VUOTA O MENO. ***/
187. boolean vuoto(const CODA c)
188. {
189.     return ((boolean)(c->cnt==0));
190. }
191.
192. /** FUNZIONE PER L'INSERIMENTO DI UN NUOVO ELEMENTO NELLA CODA. ***/
193. void enqueue(INFO info ,CODA c)
194. {
195.     NODE p;
196.     p=(NODE)malloc(sizeof(node));
197.     init_node(p,info);
198.     if(!vuoto(c))
199.     {
200.         c->rear->bro=p;
201.         c->rear=p;
202.     }
203.     else
204.         c->front=c->rear=p;
205.     c->cnt++;
206. }
207.
208. /** FUNZIONE PER IL PRELIEVO DELLE INFORMAZIONI, CCONTENUTE NEGLI ELEMENTI PRESENTI NELLA C
ODA. ***/
209. INFO dequeue(CODA c)
210. {
211.     NODE p;
212.     INFO info;
213.     p=c->front;
214.     info=p->dato;
215.     c->front=c->front->bro;
216.     c->cnt--;
217.     free(p);
218.     return info;
219. }

```

## ESECUZIONE 1

```
"C:\Users\Revers\Documents\Università\Programmazione 2\... - □ X
** INSERISCI LA RADICE DELL'ALBERO **
->A
** INSERISCI IL GRADO DELLA RADICE **
->3
** INSERISCI 1 FIGLIO DI A **
->B
** INSERISCI IL GRADO DI B **
->0
** INSERISCI 2 FIGLIO DI A **
->C
** INSERISCI IL GRADO DI C **
->0
** INSERISCI 3 FIGLIO DI A **
->D
** INSERISCI IL GRADO DI D **
->1
** INSERISCI 1 FIGLIO DI D **
->E
** INSERISCI IL GRADO DI E **
->1
** INSERISCI 1 FIGLIO DI E **
->F
** INSERISCI IL GRADO DI F **
->0

A      B      C      D      E      F
Process returned 0 (0x0)   execution time : 38.860 s
Press any key to continue.
```

# Esercizio 49[liv.1]

## Visita inorder,preorder,postorder mediante array

```
1.  /**[liv.1] Scrivere le function C per la visita (preorder, inorder e postorder) di un alber  
o binario rappresentato  
2. mediante array.**/  
3.  
4. /**LIBRERIE**/  
5. #include <stdio.h>  
6. #include <stdlib.h>  
7. #include <string.h>  
8.  
9.  
10. /**PROTOTIPI**/  
11. void inorder(char *array,short indice,short size_A);  
12. void preorder(char *array,short indice,short size_A);  
13. void postorder(char *array,short indice,short size_A);  
14.  
15.  
16. /**MAIN**/  
17. int main()  
18. {  
19. short i=0,size_A;  
20. char array[]={ "GDIBFHJACE" }; //array in input  
21. size_A=strlen(array);  
22.  
23. printf("\n *ARRAY IN INPUT*");  
24. for(i=0; i<size_A; i++)  
25. {  
26. printf("\n [%hd]: %c",i,array[i]);  
27. }  
28. printf("\n");  
29.  
30. i=0;  
31.  
32. printf("\n\n *** VISITA PREORDER ***\n ");  
33. inorder(array,0,size_A);  
34. printf("\n\n *** VISITA PREORDER ***\n ");  
35. preorder(array,0,size_A);  
36. printf("\n\n *** VISITA POSTORDER ***\n ");  
37. postorder(array,0,size_A);  
38. printf("\n\n");  
39. }  
40.  
41. /**FUNCTION**/  
42. /**GLI ALBERI BINARI VENGONO VISITATI MOLTO SPESSO UTILIZZANDO UNA DELLE TRE STRATEGIE FOND  
AMENTALI:  
43. ORDINE ANTICIPATO,ORDINE SIMMETRICO E ORDINE POSTICIPATO.  
44. CIASCUN ORDINE E' DETERMINATO DAL PUNTO IN CUI VIENE VISITATA LA RADICE: NELL'ORDINE ANTICI  
ATO  
45. LA RADICE E' VISITATA PER PRIMA, NELL'ORDINE SIMMETRICO DOPO IL SOTTOALBERO SINISTRO E NELL  
'ORDINE POSTICIPATO PER ULTIMA.**/  
46.  
47. /*Visita di un albero binario in ordine simmetrico*/  
48. void inorder(char *array,short indice,short size_A)  
49. {  
50. if(indice >= size_A)  
51. {  
52. return;  
53. }  
54. inorder(array,2*indice+1,size_A); //Ricorsione a sinistra-sottoalbero sinistro
```

```
55.     printf("%c ", array[indice]); //radice
56.     inorder(array,2*indice+2,size_A); //Ricorsione a destra-sottoalbero destro
57. }
58.
59. /*Visita in ordine anticipato*/
60. void preorder(char *array,short indice,short size_A)
61. {
62.     if(indice >= size_A)
63.     {
64.         return;
65.     }
66.     printf("%c ", array[indice]); //radice
67.     preorder(array,2*indice+1,size_A); //Ricorsione a sinistra-sottoalbero sinistro
68.     preorder(array,2*indice+2,size_A); //Ricorsione a destra-sottoalbero destro
69. }
70.
71. /*Visita in ordine posticipato*/
72. void postorder(char *array,short indice,short size_A)
73. {
74.     if(indice >= size_A)
75.     {
76.         return;
77.     }
78.     postorder(array,2*indice+1,size_A); //Ricorsione a sinistra-sottoalbero sinistro
79.     postorder(array,2*indice+2,size_A); //Ricorsione a destra-sottoalbero destro
80.     printf("%c ", array[indice]); //radice
81. }
```

## ESECUZIONE 1

```
*ARRAY IN INPUT*
```

```
[0]: G  
[1]: D  
[2]: I  
[3]: B  
[4]: F  
[5]: H  
[6]: J  
[7]: A  
[8]: C  
[9]: E
```

```
*** VISITA PREORDER ***
```

```
A B C D E F G H I J
```

```
*** VISITA PREORDER ***
```

```
G D B A C F E I H J
```

```
*** VISITA POSTORDER ***
```

```
A C B E F D H J I G
```

```
Process returned 0 (0x0)    execution time : 0.006 s  
Press ENTER to continue.
```

## Esercizio 50[liv.3]

# Visita inorder,preorder,postorder mediante liste multiple

```
1.  /***[liv.3] Scrivere function C per la costruzione e visita (preorder, inorder e postorder)
   di un albero binario
2.  rappresentato mediante liste multiple.*/
3.
4.          /**LIBRERIE*/
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <string.h>
8.
9. typedef char DATA;
10.
11. struct node
12. {
13.     DATA d;
14.     struct node *son_sx;
15.     struct node *son_dx;
16. };
17.
18. typedef struct node NODE;
19. typedef NODE *BTREE;
20.
21.
22.          /**PROTOTIPI*/
23. BTREE init_node(char d1, BTREE p1, BTREE p2);
24. BTREE create_tree(char a[], int i, int size);
25. void inorder(BTREE root);
26. void preorder(BTREE root);
27. void postorder(BTREE root);
28.
29.          /**MAIN*/
30. int main()
31. {
32.     BTREE Root;
33.     short i=0,size_A;
34.     char array[]={“GDIBFHJACE”};//ARRAY IN INPUT
35.     size_A=strlen(array);
36.
37.     printf(“\n *ARRAY IN INPUT*”);
38.     for(i=0; i<size_A; i++)
39.     {
40.         printf(“\n [%hd]: %c”,i,array[i]);
41.     }
42.     printf(“\n”);
43.
44.     i=0;
45.     Root=create_tree(array,i,size_A);
46.
47.     printf(“\n\n *** VISITA PREORDER ***\n ”);
48.     inorder(Root);
49.     printf(“\n\n *** VISITA PREORDER ***\n ”);
50.     preorder(Root);
51.     printf(“\n\n *** VISITA POSTORDER ***\n ”);
52.     postorder(Root);
53.     printf(“\n\n”);
54. }
55.
56.          /**FUNCTION*/
```

```

57. /*GLI ALBERI BINARI VENGONO VISITATI MOLTO SPESO UTILIZZANDO UNA DELLE TRE STRATEGIE FONDAMENTALI:
58. ORDINE ANTICIPATO,ORDINE SIMMETRICO E ORDINE POSTICIPATO.
59. CIASCUN ORDINE E' DETERMINATO DAL PIANO IN CUI VIENE VISITATA LA RADICE: NELL'ORDINE ANTICIPATO
60. LA RADICE E' VISITATA PER PRIMA, NELL'ORDINE SIMMETRICO DOPO IL SOTTOALBERO SINISTRO E NELL'ORDINE POSTICIPATO PER ULTIMA.*/
61.
62. /*Costruzione di un albero binario*/
63. BTREE init_node(char d1, BTREE p1, BTREE p2)
64. {
65.     BTREE t;
66.
67.     t=malloc(sizeof(NODE));
68.     t->d=d1;
69.     t->son_sx=p1;
70.     t->son_dx=p2;
71.     return t;
72. }
73.
74. /*Costruzione di un albero binario da un array*/
75. /*GLI INDICI DI UN ARRAY LINEARE POSSONO ESSERE FATTI CORRISPONDERE A NODI DI UN ALBERO BINARIO.
76. CIO' VIENE EFFETTUATO PRENDENDO IL VALORE a[i] E CONSIDERANDO COME SUOI FIGLI I VALORI:
77. 1) a[2*i+1] - figlio sinistro
78. 2) a[2*i+2] - figlio destro
79. LA FUNZIONE create_tree INCORPORA APPUNTO QUESTA MAPPA.
80. IL PARAMETRO FORMALE SIZE E' IL NUMERO DI NODI DELL'ALBERO BINARIO.*/
81.
82. BTREE create_tree(char a[], int i, int size)
83. {
84.     if(i >= size)
85.     {
86.         return NULL;
87.     }
88.     else
89.     {
90.         return (init_node(a[i],
91.                             create_tree(a, 2*i+1, size),
92.                             create_tree(a, 2*i+2, size)));
93.     }
94. }
95.
96. /*Visita di un albero binario in ordine simmetrico*/
97. void inorder(BTREE root)
98. {
99.     if(root != NULL)
100.    {
101.        inorder(root->son_sx); //Ricorsione a sinistra
102.        printf("%c ", root->d);
103.        inorder(root->son_dx); //Ricorsione a destra
104.    }
105. }
106.
107. /*Visita in ordine anticipato*/
108. void preorder(BTREE root)
109. {
110.     if(root != NULL)
111.    {
112.        printf("%c ", root->d);
113.        preorder(root->son_sx); //Ricorsione a sinistra
114.        preorder(root->son_dx); //Ricorsione a destra
115.    }
116. }
117.
118. /*Visita in ordine posticipato*/

```

```
119. void postorder(BTREE root)
120. {
121.     if(root != NULL)
122.     {
123.         postorder(root->son_sx); //Ricorsione a sinistra
124.         postorder(root->son_dx); //Ricorsione a destra
125.         printf("%c ", root->d);
126.     }
127. }
```

## ESECUZIONE 1

```
*ARRAY IN INPUT*
```

```
[0]: G  
[1]: D  
[2]: I  
[3]: B  
[4]: F  
[5]: H  
[6]: J  
[7]: A  
[8]: C  
[9]: E
```

```
*** VISITA PREORDER ***
```

```
A B C D E F G H I J
```

```
*** VISITA PREORDER ***
```

```
G D B A C F E I H J
```

```
*** VISITA POSTORDER ***
```

```
A C B E F D H J I G
```

```
Process returned 0 (0x0)    execution time : 0.006 s
Press ENTER to continue.
```

## Esercizio 51 [liv.2]

### Albero binario di ricerca mediante array

```
1.  /** [liv.2] Scrivere function C iterativa per la costruzione di un albero binario di ricerca rappresentato mediante array.*/
2.
3.
4.          /**LIBRERIE**/
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <string.h>
8.
9. struct node
10. {
11.     short dato;
12.     short key;
13. };
14.
15. typedef struct node NODE;
16.
17.          /**PROTOTIPI**/
18. void new_tree(NODE tree[], short array[]);
19.
20.          /**MAIN**/
21. int main()
22. {
23.     NODE tree[21];
24.     short array[]={5,7,8,1,6,0,3,4,9,2};
25.     short i;
26.
27. //inizializzo a 0 gli elemnti dell'albero
28. for(i=1; i<=21; i++)
29. {
30.     tree[i].key=0;
31. }
32.
33. printf("\n **ARRAY IN INPUT**");
34. for(i=0; i<10; i++)
35. {
36.     printf("\n [%hd]: %hd",i,array[i]);
37. }
38.
39. new_tree(tree,array);
40. return 0;
41. }
42.
43.          /**FUNCTION**/
44. void new_tree(NODE tree[], short array[])
45. {
46.     short nodo_corrente=0,i=0,new_node=0;
47.     //inserisco la radice
48.     tree[1].dato=array[i];
49.     tree[1].key=1;
50.
51.     while(1)
52.     {
53.         i++;
54.         new_node=array[i]; //input di un nodo n
55.         nodo_corrente=tree[0].dato; //nodo_corrente e' uguale alla radice
56.
57.         //Se il nodo inserito e' minore del corrente &&(e) ci sono nodi a sinistra
58.         while(new_node <= tree[nodo_corrente].dato && tree[2*nodo_corrente].key == 1)
59.             nodo_corrente=2*nodo_corrente;//il nodo corrente e' a destra
```

```

60.         //Se il nodo inserito e' maggiore del corrente &&(e) ci sono nodi a destra
61.         while(new_node >= tree[nodo_corrente].dato && tree[(2*nodo_corrente)+1].key == 1)
62.             nodo_corrente=(2*nodo_corrente)+1;//il nodo corrente e' a sinistra
63.
64.         //Se non ci sono figli inserisce a destra o a sinistra il nodo acquisito
65.         if(new_node >= tree[nodo_corrente].dato) //destra
66.         {
67.             tree[(2*nodo_corrente)+1].dato=new_node;
68.             tree[(2*nodo_corrente)+1].key=1;
69.
70.         }
71.         else //sinistra
72.         {
73.             tree[2*nodo_corrente].dato=new_node;
74.             tree[2*nodo_corrente].key=1;
75.         }
76.         if(i==9)
77.             break;
78.     }
79.
80.     printf("\n\n    ***ALBERO BINARIO DI RICERCA MEDIANTE ARRAY***\n");
81.     for(i=1;i<=21;i++)
82.     {
83.         if(tree[i].key==1)
84.             printf("    %hd ",tree[i].dato);
85.     }
86.     printf("\n\n");
87. }
```

## ESECUZIONE 1

\*\*ARRAY IN INPUT\*\*

```
[0]: 5  
[1]: 7  
[2]: 8  
[3]: 1  
[4]: 6  
[5]: 0  
[6]: 3  
[7]: 4  
[8]: 9  
[9]: 2
```

\*\*\*ALBERO BINARIO DI RICERCA MEDIANTE ARRAY\*\*\*

```
5      1      7      0      3      6      8      2      4      9
```

```
Process returned 0 (0x0)    execution time : 0.005 s  
Press ENTER to continue.
```

## Esercizio 52[liv.3]

# Albero binario di ricerca mediante liste multiple

```
1.  /** [liv.3] Scrivere function C iterativa per la costruzione di un albero binario di ricerca rappresentato mediante liste multiple.*/
2.  liste multiple.*/
3.
4.          /**LIBRERIE*/
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.  #include <string.h>
8.
9.  struct nodo
10. {
11.     short dato;
12.     struct nodo *right;
13.     struct nodo *left;
14. };
15.
16. typedef struct nodo NODO;
17.
18.          /**PROTOTIPI*/
19. NODO *init_node(short d1, NODO *p1, NODO *p2);
20. NODO *b_search_ric(NODO *tree, short array[]);
21. void inorder(NODO *tree);
22.
23.          /**MAIN*/
24. int main()
25. {
26.     NODO *tree;
27.     short array[]={5,7,8,1,6,0,3,4,9,2},i=0;
28.
29.     tree=malloc(sizeof(NODO));
30.
31.     printf("\n **ARRAY IN INPUT**");
32.     for(i=0; i<10; i++)
33.     {
34.         printf("\n [%hd]: %hd",i,array[i]);
35.     }
36.
37.     tree=b_search_ric(tree,array);
38.     printf("\n\n ***ALBERO BINARIO DI RICERCA MEDIANTE LISTE MULTIPLE-VISITA INORDER***\n");
39.     inorder(tree);
40.     return 0;
41. }
42.
43.          /**FUNCTION*/
44. /*Costruzione di un albero binario*/
45. NODO *init_node(short d1, NODO *p1, NODO *p2)
46. {
47.     NODO *t;
48.
49.     t=malloc(sizeof(NODO));
50.     t->dato=d1;
51.     t->left=p1;
52.     t->right=p2;
53.     return t;
54. }
55.
56. NODO *b_search_ric(NODO *tree, short array[])
```

```

57. {
58.     short i=0,new_node;
59.     tree=init_node(array[0],NULL,NULL);
60.     NODO *temp=tree;
61.
62.     while(1)
63.     {
64.         i++;
65.         new_node=array[i];
66.
67.         //Se il nodo inserito e' minore del corrente &&(e) ci sono nodi a sinistra
68.         while(new_node <= temp->dato && temp->left != NULL)
69.             temp=temp->left;//il nodo corrente e' a destra
70.         //Se il nodo inserito e' minore del corrente &&(e) ci sono nodi a destra
71.         while(new_node >= temp->dato && temp->right != NULL)
72.             temp=temp->right;//il nodo corrente e' a sinistra
73.
74.         if(new_node <= temp->dato)
75.             temp->left=init_node(new_node,NULL,NULL);
76.         else
77.             temp->right=init_node(new_node,NULL,NULL);
78.
79.         if(i==9)
80.             break;
81.     }
82.     return tree;
83. }
84.
85. /*Visita in ordine simmetrico*/
86. void inorder(NODO *tree)
87. {
88.     if(tree == NULL)
89.     {
90.         return;
91.     }
92.     inorder(tree->left); //Ricorsione a sinistra
93.     printf(" %hd ",tree->dato);
94.     inorder(tree->right); //Ricorsione a destra
95. }

```

## ESECUZIONE 1

```
**ARRAY IN INPUT**
[0]: 5
[1]: 7
[2]: 8
[3]: 1
[4]: 6
[5]: 0
[6]: 3
[7]: 4
[8]: 9
[9]: 2

***ALBERO BINARIO DI RICERCA MEDIANTE LISTE MULTIPLE-VISITA INORDER***
5   0   1   3   2   4   9   6   7   8
Process returned 0 (0x0)  execution time : 0.005 s
Press ENTER to continue.
```

## Esercizio 53[liv.3] Decodifica codice morse

```
1.  /**[liv.2] A quale messaggio corrisponde la sequenza in codice morse:  
2.  --- --. --- .. .--- - - .- .- . - .- .- .- .-  
3.  dove 1 spazio separa i singoli caratteri e 3 spazi le parole.**/  
4.  
5.          /**LIBRERIE**/  
6.  #include <stdio.h>  
7.  #include <stdlib.h>  
8.  #include <string.h>  
9.  
10. /** STRUTTURA NODO ALBERO BINARIO MEDIANTE LISTA MULTIPLA**/  
11. struct node  
12. {  
13.     char d;  
14.     struct node *left;  
15.     struct node *right;  
16. };  
17. typedef struct node node;  
18. typedef node *BTREE;  
19.  
20.  
21.          /**PROTOTIPI**/  
22. BTREE init_node(char , BTREE , BTREE );  
23. BTREE create_tree(char [], int , int );  
24. char *decodifica_morse(BTREE,char[]);  
25.  
26.  
27.          /**MAIN**/  
28. int main()  
29. {  
30.     BTREE root;  
31.     short i=0;  
32.     char array[]="aETIANMSURWDKGHOHVF|L|PJBXCYZQ||",  
33.     input[]="--- --. --- .. .--- - - .- .- . - .- .- .-",*message;  
34.     message=malloc(sizeof(char)*strlen(input));  
35.     root=create_tree(array,i,strlen(array));  
36.     printf("** MESSAGGIO MORSE **\n");  
37.     puts(input);  
38.     message=decodifica_morse(root,input);  
39.     printf("** MESSAGGIO **\n%s",message);  
40.     return 0;  
41. }  
42.  
43.  
44.          /**FUNCTION**/  
45. /** INIZIALIZZAZIONE DEL NUOVO NODO. **/  
46. BTREE init_node(char d1, BTREE p1, BTREE p2)  
47. {  
48.     BTREE t;  
49.     t=malloc(sizeof(node));  
50.     t->d=d1;  
51.     t->left=p1;  
52.     t->right=p2;  
53.     return t;  
54. }  
55.  
56. /** COSTRUZIONE DI UN ALBERO BINARIO DA UN ARRAY. **/  
57. BTREE create_tree(char a[], int i, int size)  
58. {  
59.     if(i>=size)  
60.         return NULL;
```

```

61.     else
62.         return (init_node(a[i],
63.                             create_tree(a,2*i+1,size),
64.                             create_tree(a,2*i+2,size)));
65.     }
66.
67. char *decodifica_morse(BTREE root,char input[])
68. {
69.     BTREE tmp;
70.     char *message,nodo;
71.     short i=0,j=0,k=0,word=1;
72.     tmp=malloc(sizeof(node));
73.     message=malloc(sizeof(char)*strlen(input));
74.     tmp=root;
75.     while(input[i]!='\0')
76.     {
77.         nodo=input[i];
78.         switch (nodo)
79.         {
80.             case '-':
81.                 root=root->right;
82.                 break;
83.             case '.':
84.                 root=root->left;
85.                 break;
86.             case ' ':
87.                 j=i;
88.                 j++;
89.                 while(input[j]==' ')
90.                 {
91.                     word++;
92.                     j++;
93.                 }
94.                 break;
95.             default:
96.                 break;
97.         }
98.
99.         if(word==3)
100.        {
101.             message[k++]=root->d;
102.             message[k++]=' ';
103.             root=tmp;
104.             i=j-1;
105.         }
106.         else if(nodo==' ')
107.         {
108.             message[k++]=root->d;
109.             root=tmp;
110.         }else if(i==strlen(input)-1)
111.         {
112.             message[k++]=root->d;
113.             message[k]='\0';
114.         }
115.         word=1;
116.         i++;
117.     }
118.     return message;
119. }
```

## ESECUZIONE 1

```
-----  
** MESSAGGIO MORSE **  
----- . - . . . . - - - - . - - - - -  
** MESSAGGIO **  
OGGI IL MARE BLU  
Process returned 0 (0x0) execution time : 0.006 s  
Press ENTER to continue.  
_
```

## Esercizio 54[liv.2]

### Heap mediante array

```
1.  /**[liv.2] Scrivere function C iterativa per la costruzione di un heap rappresentato mediante array*/
2.
3.          /**LIBRERIE**/
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7.
8. #define SIZE    10
9.
10.         /**PROTOTIPI**/
11. void Heapify(short array[],short i);
12. void swap(short *a,short *b);
13.
14.         /**MAIN**/
15. int main()
16. {
17.     short array[]={2,6,18,3,42,12,55,44,94,79},i;
18.
19.     printf("\n **ARRAY IN INPUT**");
20.     for(i=0; i<SIZE; i++)
21.     {
22.         printf("\n [%d]: %d ",i,array[i]);
23.     }
24.     i=0;
25.     Heapify(array,i);
26.     return 0;
27. }
28.
29.         /**FUNCTION**/
30. /*Proprieta' fondamentale degli Heap e' che il valore
31. associato al nodo padre e' sempre maggiore o uguale a
32. quello associato ai nodi figli.
33. PROPRIETA' HEAP: se x e' un qualsiasi nodo dell'heap(ad esclusione della radice) si ha:
34.
35.             key(x) <= key(padre(x))
36. Ne consegue che in un heap l'elemento massimo e' memorizzato nella radice.*/
37. void Heapify(short array[],short i)
38. {
39.     i=-1;
40.     while(1)
41.     {
42.         i++;
43.         /*confronta l'elemento corrente col padre. Mentre l'elemento corrente e' piu grande
44.         del
45.             padre e non siamo arrivati alla radice (i=0)*/
46.             while(array[i]>array[i/2] && i!=0)
47.             {
48.                 swap(&array[i],&array[i/2]);//scambia i due nodi.
49.                 i=i/2; /*aggiorna l'indice del nodo corrente */
50.             }
51.             if(i==SIZE-1)
52.                 break;
53.     }
54.     printf("\n\n **VISITA HEAP**");
55.     for(i=0; i<SIZE; i++)
56.         printf("\n [%d]: %d ",i,array[i]);
57. }
58. void swap(short *a,short *b)
```

```
59. {
60.     short temp;
61.     temp=*a;
62.     *a=*b;
63.     *b=temp;
64. }
```

## ESECUZIONE 1

```
**ARRAY IN INPUT**
[0]: 5
[1]: 7
[2]: 8
[3]: 1
[4]: 6
[5]: 0
[6]: 3
[7]: 4
[8]: 9
[9]: 2

***ALBERO BINARIO DI RICERCA MEDIANTE LISTE MULTIPLE-VISITA INORDER***
5    0    1    3    2    4    9    6    7    8
Process returned 0 (0x0)  execution time : 0.005 s
Press ENTER to continue.
=
```

## Esercizio 55[liv.3]

### Heap di un albero binario mediante array

```
1.  /**[liv.3] Scrivere function C iterativa per la trasformazione in un heap di un albero binario rappresentato mediante array.*/
2.
3.          /**LIBRERIE*/
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7.
8. #define SIZE    10
9.
10.         /**PROTOTIPI*/
11. short Left(short i);
12. short Right(short i);
13. void Swap(short *a,short *b);
14. void Heapify(short array[],short i);
15. void Build_max_heap(short array[]);
16.
17.         /**MAIN*/
18. int main()
19. {
20.     short array[]={2,6,18,3,42,12,55,44,94,79},i;
21.
22.     printf("\n **ARRAY INPUT**\n");
23.     for(i=0;i<SIZE;i++)
24.         printf(" %hd",array[i]);
25.
26.     Build_max_heap(array);
27.
28.     printf("\n **ARRAY HEAP**\n");
29.     for(i=0;i<SIZE;i++)
30.         printf(" %hd",array[i]);
31.     return 0;
32. }
33.
34.         /**FUNCTION*/
35. /*FUNZIONE CHE CALCOLA IL FIGLIO SINISTRO*/
36. short Left(short i)
37. {
38.     return (2*i)+1;
39. }
40.
41. /*FUNZIONE CHE CALCOLA IL FIGLIO DESTRO*/
42. short Right(short i)
43. {
44.     return (2*i)+2;
45. }
46.
47. void Swap(short *a,short *b)
48. {
49.     short tmp;
50.     tmp=*a;
51.     *a=*b;
52.     *b=tmp;
53. }
54.
55. void Heapify(short array[],short i)
56. {
57.     short l=0,r=0,massimo=0;
58.     l=Left(i);
59.     r=Right(i);
```

```

60.    //SE E' VERO CHE:
61.    //IL FIGLIO SINISTRO DELL'INDICE CORRENTE E' MINORE DEL HEAPSIZE
62.    //ED E' MAGGIOR DEL PADRE
63.    if(l<SIZE && array[l]>array[i])
64.        massimo=l;//IL MASSIMO DIVENTA IL FIGLIO SINISTRO
65.    else
66.        massimo=i;//ALTRIMENTI IL MASSIMO RESTA L'INDICE CORRENTE
67.    //SE E' VERO CHE:
68.    //IL FIGLIO DESTRO DELL'INDICE CORRENTE E' MINORE DEL HEAPSIZE
69.    //ED E' MAGGIOR DEL PADRE
70.    if(r<SIZE && array[r]>array[massimo])
71.        massimo=r;//IL MASSIMO DIVENTA IL FIGLIO DESTRO
72.
73.    //SE IL MASSIMO E' DIVERSO DA i, OVVERO, NON E' L'INDICE CORRENTE:
74.    if(massimo!=i)
75.    {
76.        //SCAMBIA L'INDICE CORRENTE CON IL MASSIMO
77.        Swap(&array[i],&array[massimo]);
78.        Heapify(array,massimo);
79.    }
80. }
81.
82. void Build_max_heap(short array[])
83. {
84.     short i;
85.     /*CON QUESTO CICLO FOR ANDIAMO AD ORDINARE
86.      L'ALBERO BINARIO CON LA MODALITA' BOTTOM UP,
87.      OVVERO PARTENDO DAL BASSO VERSO L'ALTO.*/
88.     for(i=(SIZE/2)-1;i>=0;i--)
89.         Heapify(array,i);
90. }

```

## ESECUZIONE 1

```
**ARRAY INPUT**
2 6 18 3 42 12 55 44 94 79
**ARRAY HEAP**
94 79 55 44 42 12 18 6 3 2
Process returned 0 (0x0)    execution time : 0.006 s
Press ENTER to continue.
```

## Esercizio 56[liv.1]

### Grafo non orientato mediante matrice di adiacenze

```
1.  /***[liv.1] Scrivere function C per la costruzione di un grafo non orientato mediante matric
e di adiacenze:
2.      in input per ogni nodo sono specificati quelli adiacenti.
3.      Scegliendo in input un nodo, scrivere una function C che restituisca il suo grado.*/
4.
5.          /**LIBRERIE*/
6. #include <stdio.h>
7. #include <stdlib.h>
8.
9. #define SIZE_A 20
10.
11. struct nodo
12. {
13.     char dato;
14.     char adiacenza[20];
15. };
16.
17. typedef struct nodo INPUT;
18.
19.         /**PROTOTIPI*/
20. void Set_info(INPUT *set, short N, short i, short j);
21. void Insertion_ad(short *Matrice, INPUT *set, short N, short i, short j);
22. short search_degree(char dato , short *Matrice, short N);
23. void Stampa(short *Matrice, short N);
24.
25.         /**MAIN*/
26. int main()
27. {
28.     INPUT *set;//variabile che ci permette di accedere al campo informazione della strutt
ura in input
29.     char dato;
30.     short i=0,j=0,N=0,*Matrice,grado=0;
31.
32.     printf("\n **COSTRUZIONE DI UN GRAFO NON ORIENTATO MEDIANTE MATRICE DI ADIACENZE**");
33.     printf("\n\n Inserisci numero di nodi da inserire: ");
34.     scanf("%d",&N);
35.
36.     set=malloc(sizeof(INPUT)*N);
37.     Matrice=calloc(N*N,sizeof(short));
38.     Set_info(set,N,i,j);
39.     Insertion_ad(Matrice,set,N,i,j);
40.     Stampa(Matrice,N);
41.
42.     getchar();
43.     printf("\n\n Inserire il nodo di cui si vuol sapere il grado: ");
44.     scanf("%c",&dato);
45.     grado=search_degree(dato,Matrice,N);
46.     printf("\n Il grado del nodo [%c] e' [%d].",dato,grado);
47.
48.     return 0;
49. }
50.
51.         /**FUNCTION*/
52. /*In questa funzione andremo a settare tutte le informazioni e ad inserire le adiacenze*/
53. void Set_info(INPUT *set, short N, short i, short j)
54. {
55.     short size_adiac=0;
```

```

57.    /**INIZIALIZZAZIONE DELLA STRUTTURA IN INPUT*/
58.    for(i=0; i<N; i++)
59.    {
60.        set[i].dato=65+i; //setto automaticamente, a seconda dei nodi inseriti, aggiungendo 1
61.        //etere da 65 a 65+i (A,B,C,D,...).
62.        for(j=0; j<SIZE_A; j++)
63.            set[i].adiacenza[j]=0; //inizializzo l'array adiacenze.
64.
65.    /**STAMPA NODI IN INPUT*/
66.    printf("\n **NODI IN INPUT**\n");
67.    for(i=0; i<N; i++)
68.    {
69.        printf(" [%c]",set[i].dato);
70.    }
71.    printf("\n");
72.
73.    /**INSERIMENTO ADIACENZE*/
74.    for(i=0; i<N; i++)
75.    {
76.        printf("\n Inserisci numero di adiacenze per il nodo [%c]: ",set[i].dato);
77.        scanf("%hd",&size_adiac);
78.        j=0;
79.        for(j=0; j<size_adiac; j++)
80.        {
81.            getchar();
82.            fflush(NULL);
83.            printf(" Inserisci adiacenza [%d] del nodo [%c]: ",j+1,set[i].dato);
84.            scanf("%c",&set[i].adiacenza[j]);
85.        }
86.    }
87.
88.    /**STAMPA NODI E ADIACENZE*/
89.    printf("\n\n**NODI---ADIACENZE**");
90.    for(i=0; i<N; i++)
91.    {
92.        printf("\n [%c]-->",set[i].dato);
93.        j=0;
94.        while(set[i].adiacenza[j]!=0)
95.        {
96.            printf(" [%c]",set[i].adiacenza[j]);
97.            j++;
98.        }
99.        printf("\n");
100.    }
101. }
102.
103. /*In questa funzione andremo ad inserire nella nostra matrice 1 o 0 a seconda delle adiacenze*/
104. void Insertion_ad(short *Matrice, INPUT *set, short N, short i, short j)
105. {
106.     short indice=0;
107.
108.     for(i=0; i<N; i++)
109.     {
110.         j=0;
111.         while(set[i].adiacenza[j]!=0)
112.         {
113.             indice=set[i].adiacenza[j]-65;
114.             *(Matrice+i*N+indice)=1;
115.             j++;
116.         }
117.     }
118. }
119.
120. /*Questa funzione ci restituira' il grado del nodo che noi vogliamo sapere*/

```

```
121. short search_degree(char dato , short *Matrice, short N)
122. {
123.     short i,grado=0;
124.     dato-=65;
125.     for(i=0; i<N; i++)
126.     {
127.         if(*(Matrice+dato*N+i)==1)
128.             grado++;
129.     }
130.     return grado;
131. }
132.
133. /*Stampa matrice tramite mappa di memorizzazione*/
134. void Stampa(short *Matrice, short N)
135. {
136.     short i,j;
137.     printf("\n");
138.     printf(" | ");
139.     for(i=0;i<N;i++)
140.         printf("%c ",65+i);
141.     printf("|");
142.     puts("");
143.     for(i=0; i<N; i++)
144.     {
145.         printf(" |%c ",65+i);
146.         for(j=0; j<N; j++)
147.         {
148.             printf(" %hd ",*(Matrice+i*N+j));
149.         }
150.         printf("|");
151.         puts("");
152.     }
153. }
```

## ESECUZIONE 1

\*\*COSTRUZIONE DI UN GRAFO NON ORIENTATO MEDIANTE MATRICE DI ADIACENZE\*\*

Inserisci numero di nodi da inserire: 7

\*\*NODI IN INPUT\*\*

[A] [B] [C] [D] [E] [F] [G]

Inserisci numero di adiacenze per il nodo [A]: 4

Inserisci adiacenza [1] del nodo [A]: F

Inserisci adiacenza [2] del nodo [A]: B

Inserisci adiacenza [3] del nodo [A]: C

Inserisci adiacenza [4] del nodo [A]: G

Inserisci numero di adiacenze per il nodo [B]: 1

Inserisci adiacenza [1] del nodo [B]: A

Inserisci numero di adiacenze per il nodo [C]: 1

Inserisci adiacenza [1] del nodo [C]: A

Inserisci numero di adiacenze per il nodo [D]: 2

Inserisci adiacenza [1] del nodo [D]: E

Inserisci adiacenza [2] del nodo [D]: F

Inserisci numero di adiacenze per il nodo [E]: 3

Inserisci adiacenza [1] del nodo [E]: F

Inserisci adiacenza [2] del nodo [E]: D

Inserisci adiacenza [3] del nodo [E]: G

Inserisci numero di adiacenze per il nodo [F]: 3

Inserisci adiacenza [1] del nodo [F]: A

Inserisci adiacenza [2] del nodo [F]: D

Inserisci adiacenza [3] del nodo [F]: E

Inserisci numero di adiacenze per il nodo [G]: 2

Inserisci adiacenza [1] del nodo [G]: A

Inserisci adiacenza [2] del nodo [G]: E

\*\*NODI----ADIACENZE\*\*

[A]---> [F] [B] [C] [G]

[B]---> [A]

[C]---> [A]

[D]---> [E] [F]

[E]---> [F] [D] [G]

[F]---> [A] [D] [E]

[G]---> [A] [E]

	A	B	C	D	E	F	G
A	0	1	1	0	0	1	1
B	1	0	0	0	0	0	0
C	1	0	0	0	0	0	0
D	0	0	0	1	1	0	
E	0	0	0	1	0	1	1
F	1	0	0	1	1	0	0
G	1	0	0	0	1	0	0

Inserire il nodo di cui si vuol sapere il grado: A

Il grado del nodo [A] e' [4].

Process returned 0 (0x0) execution time : 134.613 s

Press ENTER to continue.

## Esercizio 57[liv.1]

### Grafo orientato mediante matrice di adiacenze

```
1.  /***[liv.1] Scrivere function C per la costruzione di un grafo orientato mediante matrice di adiacenze: in input per
2.  ogni nodo sono specificati quelli raggiungibili. Scegliendo in input un nodo, scrivere una
3.  function C che
4.  restituisca il numero degli archi uscenti e quello degli archi entranti.*/
5.          /**LIBRERIE*/
6. #include <stdio.h>
7. #include <stdlib.h>
8.
9. #define SIZE_A 20
10.
11. struct nodo
12. {
13.     char dato;
14.     char adiacenza[20];
15. };
16.
17. typedef struct nodo INPUT;
18.
19.          /**PROTOTIPI*/
20. void Set_info(INPUT *set, short N, short i, short j);
21. void Insertion_ad(short *Matrice, INPUT *set, short N, short i, short j);
22. void search_arches(char dato , short *Matrice, short N, short *archi_uscenti, short *archi_entranti);
23. void Stampa(short *Matrice, short N);
24.
25.          /**MAIN*/
26. int main()
27. {
28.     INPUT *set;//variabile che ci permette di accedere al campo informazione della struttura in input
29.     char dato;
30.     short i=0,j=0,N=0,*Matrice,archi_uscenti=0, archi_entranti=0;
31.
32.     printf("\n **COSTRUZIONE DI UN GRAFO ORIENTATO MEDIANTE MATRICE DI ADIACENZE**");
33.     printf("\n\n Inserisci numero di nodi da inserire: ");
34.     scanf("%d",&N);
35.
36.     set=malloc(sizeof(INPUT)*N);
37.     Matrice=calloc(N*N,sizeof(short));
38.     Set_info(set,N,i,j);
39.     Insertion_ad(Matrice,set,N,i,j);
40.     Stampa(Matrice,N);
41.
42.     getchar();
43.     printf("\n\n Inserire il nodo di cui si vuol sapere gli archi entranti e uscenti: ");
44.     scanf("%c",&dato);
45.     search_arches(dato,Matrice,N,&archi_uscenti,&archi_entranti);
46.     printf("\n Nodo: %c\n Archi uscenti: %hd\n Archi entranti: %hd\n\n",dato,archi_uscenti,
47.             archi_entranti);
48.     return 0;
49. }
50.
51.          /**FUNCTION*/
52. /*In questa funzione andremo a settare tutte le informazioni e ad inserire le adiacenze*/
53. void Set_info(INPUT *set, short N, short i, short j)
```

```

54. {
55.     short size_adiac=0;
56.
57.     /**INIZIALIZZAZIONE DELLA STRUTTURA IN INPUT**/
58.     for(i=0; i<N; i++)
59.     {
60.         set[i].dato=65+i; //setto automaticamente, a seconda dei nodi inseriti, aggiungendo 1
61.         //terre da 65 a 65+i (A,B,C,D,...).
62.         for(j=0; j<SIZE_A; j++)
63.             set[i].adiacenza[j]=0; //inizializzo l'array adiacenze.
64.     }
65.
66.     /**STAMPA NODI IN INPUT**/
67.     printf("\n **NODI IN INPUT**\n");
68.     for(i=0; i<N; i++)
69.     {
70.         printf(" [%c]",set[i].dato);
71.     }
72.     printf("\n");
73.
74.     /**INSERIMENTO ADIACENZE**/
75.     for(i=0; i<N; i++)
76.     {
77.         printf("\n Inserisci il numero di nodi raggiungibili da [%c]: ",set[i].dato);
78.         scanf("%hd",&size_adiac);
79.         j=0;
80.         for(j=0; j<size_adiac; j++)
81.         {
82.             getchar();
83.             fflush(NULL);
84.             printf(" Inserisci nodo [%d] raggiungibile dal nodo [%c]: ",j+1,set[i].dato);
85.             scanf("%c",&set[i].adiacenza[j]);
86.         }
87.
88.         /**STAMPA NODI E ADIACENZE**/
89.         printf("\n\n**NODI---RAGGIUNGIBILI**");
90.         for(i=0; i<N; i++)
91.         {
92.             printf("\n [%c]-->",set[i].dato);
93.             j=0;
94.             while(set[i].adiacenza[j]!=0)
95.             {
96.                 printf(" [%c]",set[i].adiacenza[j]);
97.                 j++;
98.             }
99.             printf("\n");
100.        }
101.    }
102.
103. /*In questa funzione andremo ad inserire nella nostra matrice 1 o 0 a seconda delle adiacenze*/
104. void Insertion_ad(short *Matrice, INPUT *set, short N, short i, short j)
105. {
106.     short indice=0;
107.
108.     for(i=0; i<N; i++)
109.     {
110.         j=0;
111.         while(set[i].adiacenza[j]!=0)
112.         {
113.             indice=set[i].adiacenza[j]-65;
114.             *(Matrice+i*N+indice)=1;
115.             j++;
116.         }
117.     }

```

```

118. }
119.
120. /*Questa funzione ci restituira' gli archi uscenti ed entranti del nodo che abbiamo inserito
   o in precedenza*/
121. void search_arches(char dato , short *Matrice, short N, short *archi_uscenti, short *archi_
   entranti)
122. {
123.     short i;
124.     dato-=65;
125.     for(i=0; i<N; i++)
126.     {
127.         if(*(Matrice+dato*N+i)==1)
128.             (*archi_uscenti)++;
129.     }
130.     for(i=0; i<N; i++)
131.     {
132.         if(*(Matrice+i*N+dato)==1)
133.             (*archi_entranti)++;
134.     }
135. }
136.
137. /*Stampa matrice tramite mappa di memorizzazione*/
138. void Stampa(short *Matrice, short N)
139. {
140.     short i,j;
141.     printf("\n");
142.     printf(" | ");
143.     for(i=0;i<N;i++)
144.         printf("%c ",65+i);
145.     printf("|\n");
146.     puts("");
147.     for(i=0; i<N; i++)
148.     {
149.         printf(" |%c ",65+i);
150.         for(j=0; j<N; j++)
151.         {
152.             printf("%hd ",*(Matrice+i*N+j));
153.         }
154.         printf("|\n");
155.         puts("");
156.     }
157. }

```

## ESECUZIONE 1

\*\*COSTRUZIONE DI UN GRAFO ORIENTATO MEDIANTE MATRICE DI ADIACENZE\*\*

Inserisci numero di nodi da inserire: 7

\*\*NODI IN INPUT\*\*

[A] [B] [C] [D] [E] [F] [G]

Inserisci il numero di nodi raggiungibili da [A]: 2

Inserisci nodo [1] raggiungibile dal nodo [A]: F

Inserisci nodo [2] raggiungibile dal nodo [A]: C

Inserisci il numero di nodi raggiungibili da [B]: 1

Inserisci nodo [1] raggiungibile dal nodo [B]: A

Inserisci il numero di nodi raggiungibili da [C]: 0

Inserisci il numero di nodi raggiungibili da [D]: 0

Inserisci il numero di nodi raggiungibili da [E]: 3

Inserisci nodo [1] raggiungibile dal nodo [E]: F

Inserisci nodo [2] raggiungibile dal nodo [E]: D

Inserisci nodo [3] raggiungibile dal nodo [E]: G

Inserisci il numero di nodi raggiungibili da [F]: 2

Inserisci nodo [1] raggiungibile dal nodo [F]: A

Inserisci nodo [2] raggiungibile dal nodo [F]: D

Inserisci il numero di nodi raggiungibili da [G]: 1

Inserisci nodo [1] raggiungibile dal nodo [G]: A

\*\*NODI----RAGGIUNGIBILI\*\*

[A]---> [F] [C]

[B]---> [A]

[C]---

[D]---

[E]---> [F] [D] [G]

[F]---> [A] [D]

[G]---> [A]

	A	B	C	D	E	F	G
A	0	0	1	0	0	1	0
B	1	0	0	0	0	0	0
C	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0
E	0	0	0	1	0	1	1
F	1	0	0	1	0	0	0
G	1	0	0	0	0	0	0

Inserire il nodo di cui si vuol sapere gli archi entranti e uscenti: A

Nodo: A

Archi uscenti: 2

Archi entranti: 3

Process returned 0 (0x0) execution time : 39.807 s  
Press ENTER to continue.

## Esercizio 58[liv.2]

### Grafo non orientato mediante liste di adiacenze

```
1.  /**[liv.2] Scrivere function C per la costruzione di un grafo non orientato mediante liste
2.   di adiacenze:
3.   in input per ogni nodo sono specificati quelli adiacenti.*/
4.   */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. #define N 13
9.
10.
11. struct adiacenze
12. {
13.     char dato;
14.     struct adiacenze *next;
15. };
16. typedef struct adiacenze adiacenze;
17. typedef adiacenze *ADIACENZE;
18.
19. struct grafo
20. {
21.     char dato;
22.     ADIACENZE vertice;
23. };
24. typedef struct grafo grafo;
25. typedef grafo *GRAFO;
26.
27. /**
28. void add_nodo(char, ADIACENZE *);
29.
30. */
31. int main()
32. {
33.     grafo vertici[N]; ADIACENZE tmp,pnt;
34.     short i,j,MAX_A=0;
35.     char dato;
36.     pnt=(ADIACENZE)malloc(sizeof(adiacenze));
37.
38.     for(i=0;i<N;i++)
39.     {
40.         vertici[i].dato=65+i;//inizializzo inserendo i caratteri da 65 a 65+i---
41.         (A,B,C,D,...)
42.         vertici[i].vertice=NULL;
43.     }
44.     /**
45.     **NODI IN INPUT**
46.     printf("\n\n** NODI GRAFO: **\n");
47.     for(i=0;i<N;i++)
48.         printf("%c ",vertici[i].dato);
49.     /**
50.     **INSERIMENTO ADIACENZE PER OGNI NODO**
51.     for(i=0;i<N;i++)
52.     {
53.         printf("\n\nInserisci il numero di adiacenze del nodo %c\n-> ",vertici[i].dato);
54.         fflush(stdin); scanf("%hd",&MAX_A);
55.         j=0;
56.         //Se il size adiacenze del nodo corrente e' maggiore di 0 inseriamo la prima
57.         //adiacenza di quel nodo in modo da inserirla in testa.
58.         //se avessimo inserito le adiacenze in tutto il ciclo while
59.         //avremmo avuto in output valori sporchi in testa alla lista.
```

```

57.         if(MAX_A>0)
58.         {
59.             getchar();
60.             printf("Inserisci adiacenza %d del nodo %c\n-> ",j+1,vertici[i].dato);
61.             fflush(stdin); scanf("%c",&dato);
62.             if(dato>=65 && dato<65+N)//controlliamo che il valore inserito non sia diverso
63.                 da 65 a 65+i
64.                 {
65.                     tmp=(ADIACENZE)malloc(sizeof(adiacenze));
66.                     add_nodo(dato,&tmp);
67.                     pnt=tmp;
68.                     j++;
69.                 }else
70.                 {
71.                     printf("INPUT ERROR!\n Inserire nodo esistente.\a");
72.                     fflush(stdin); getchar();
73.                     j=MAX_A;
74.                     i--;
75.                 }
76.             //finche' j non e' uguale al size delle adiacenze restanti continueremo ad inserire
77.             while(j!=MAX_A)
78.             {
79.                 getchar();
80.                 printf("Inserisci adiacenza %d del nodo %c\n-> ",j+1,vertici[i].dato);
81.                 fflush(stdin); scanf("%c",&dato);
82.                 if(dato>=65 && dato<65+N)
83.                 {
84.                     add_nodo(dato,&tmp);
85.                     j++;
86.                 }else
87.                 {
88.                     printf("INPUT ERROR!\n Inserire nodo esistente.\a");
89.                     fflush(stdin); getchar();
90.                 }
91.             vertici[i].vertice=pnt;
92.         }
93.     }
94.
95.     /**VISITA DEL GRAFO NON ORIENTATO MEDIANTE LISTE DI ADIACENZA*/
96.     printf("\n\n** VISUALIZZAZIONE GRAFO **\n");
97.     for(i=0;i<N;i++)
98.     {
99.         tmp=vertici[i].vertice;
100.        printf("\nNODO: %c\tADIACENZE: ",vertici[i].dato);
101.        //finche' temp non arriva alla fine della lista
102.        //visualizziamo le adiacenze del nodo i-esimo
103.        while(tmp!=NULL)
104.        {
105.            printf(" %c ",tmp->dato);
106.            tmp=tmp->next;
107.        }
108.    }
109.    free(tmp);
110.    free(pnt);
111.    printf("\n");
112.    return 0;
113. }
114.
115.         /**FUNCTION*/
116. /** FUNZIONE PER L'INSERIMENTO DI UN NUOVO NODO IN UNA LISTA.*/
117. void add_nodo(char info, ADIACENZE *head)
118. {
119.     ADIACENZE pnt;
120.     pnt=(ADIACENZE)calloc(1,sizeof(adiacenze));

```

```
121.     pnt->dato=info;
122.     pnt->next=NULL;
123.     (*head)->next=pnt;
124.     pnt->next=NULL;
125.     *head=pnt;
126. }
```

## ESECUZIONE 1

\*\* NODI GRAFO: \*\*

A B C D E F G H I J K L M

Inserisci il numero di adiacenze del nodo A

-> 4

Inserisci adiacenza 1 del nodo A

-> B

Inserisci adiacenza 2 del nodo A

-> C

Inserisci adiacenza 3 del nodo A

-> F

Inserisci adiacenza 4 del nodo A

-> G

Inserisci il numero di adiacenze del nodo B

-> 1

Inserisci adiacenza 1 del nodo B

-> A

Inserisci il numero di adiacenze del nodo C

-> 1

Inserisci adiacenza 1 del nodo C

-> A

Inserisci il numero di adiacenze del nodo D

-> 2

Inserisci adiacenza 1 del nodo D

-> E

Inserisci adiacenza 2 del nodo D

-> F

Inserisci il numero di adiacenze del nodo E

-> 3

Inserisci adiacenza 1 del nodo E

-> D

Inserisci adiacenza 2 del nodo E

-> F

Inserisci adiacenza 3 del nodo E

-> G

Inserisci il numero di adiacenze del nodo F

-> 3

Inserisci adiacenza 1 del nodo F

-> A

Inserisci adiacenza 2 del nodo F

-> D

Inserisci adiacenza 3 del nodo F

-> E

```
Inserisci il numero di adiacenze del nodo G  
-> 2  
Inserisci adiacenza 1 del nodo G  
-> A  
Inserisci adiacenza 2 del nodo G  
-> E  
  
Inserisci il numero di adiacenze del nodo H  
-> 1  
Inserisci adiacenza 1 del nodo H  
-> I  
  
Inserisci il numero di adiacenze del nodo I  
-> 1  
Inserisci adiacenza 1 del nodo I  
-> H  
  
Inserisci il numero di adiacenze del nodo J  
-> 3  
Inserisci adiacenza 1 del nodo J  
-> K  
Inserisci adiacenza 2 del nodo J  
-> L  
Inserisci adiacenza 3 del nodo J  
-> M  
  
Inserisci il numero di adiacenze del nodo K  
-> 1  
Inserisci adiacenza 1 del nodo K  
-> J  
  
Inserisci il numero di adiacenze del nodo L  
-> 2  
Inserisci adiacenza 1 del nodo L  
-> J  
Inserisci adiacenza 2 del nodo L  
-> M  
  
Inserisci il numero di adiacenze del nodo M  
-> 2  
Inserisci adiacenza 1 del nodo M  
-> J  
Inserisci adiacenza 2 del nodo M  
-> L  
  
** VISUALIZZAZIONE GRAFO **  
  
NODO: A ADIACENZE: B C F G  
NODO: B ADIACENZE: A  
NODO: C ADIACENZE: A  
NODO: D ADIACENZE: E F  
NODO: E ADIACENZE: D F G  
NODO: F ADIACENZE: A D E  
NODO: G ADIACENZE: A E  
NODO: H ADIACENZE: I  
NODO: I ADIACENZE: H  
NODO: J ADIACENZE: K L M  
NODO: K ADIACENZE: J  
NODO: L ADIACENZE: J M  
NODO: M ADIACENZE: J L
```

```
Process returned 0 (0x0) execution time : 109.856 s  
Press ENTER to continue. █
```

# Esercizio 63[liv.1]

## Somma iterativa e ricorsiva delle componenti di un array

```
1.  /**[liv.1] Scrivere delle function C (rispettivamente iterativa e ricorsiva) per calcolare
2.  (con ricorsione sia lineare
3.  sia binaria) la somma delle componenti di un array.*/
4.          /**LIBRERIE*/
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. #define SIZE      5
9.
10.         /**PROTOTIPI*/
11. short Somma_iterativa(short A[], short n);
12. short Somma_ric_lin(short A[], short n);
13. short Somma_ric_bin(short A[], short n);
14.
15.         /**MAIN*/
16. int main()
17. {
18.     short A[SIZE],n=0,i=0,scelta=0,somma_it=0,somma_ric=0,somma_bin=0;
19.     n=SIZE;
20.     srand((unsigned)time(NULL));
21.
22.     printf("\n **SOMMA ITERATIVA E RICORSIVA DI %d COMPONENTI DI UN ARRAY**\n",n);
23.     for(i=0; i<n; i++)
24.     {
25.         fflush(stdin);
26.         A[i]=(unsigned short)(rand()%10);
27.     }
28.     printf("\n **ARRAY IN INPUT**\n");
29.     for(i=0; i<n; i++)
30.         printf(" [%d]",A[i]);
31.     printf("\n");
32.     do
33.     {
34.         printf("\n **MENU**\n [1] Somma iterativa\n [2] Somma ricorsiva lineare\n [3] Somma
35. ricorsiva binaria\n [4] Exit\n Inserire scelta: ");
36.         fflush(stdin);
37.         scanf("%d",&scelta);
38.         switch(scelta)
39.         {
40.             case 1: printf("\n **SOMMA ITERATIVA**\n");
41.                     for(i=0; i<n; i++)
42.                         printf(" [%d]",A[i]);
43.                     printf(" = %d",somma_it=Somma_iterativa(A,n));
44.                     break;
45.             case 2: printf("\n **SOMMA RICORSIVA LINEARE**\n");
46.                     for(i=0; i<n; i++)
47.                         printf(" [%d]",A[i]);
48.                     printf(" = %d",somma_ric=Somma_ric_lin(A,n));
49.                     break;
50.             case 3: printf("\n **SOMMA RICORSIVA BINARIA**\n");
51.                     for(i=0; i<n; i++)
52.                         printf(" [%d]",A[i]);
53.                     printf(" = %d",somma_bin=Somma_ric_bin(A,n));
54.                     break;
55.             case 4:exit(EXIT_FAILURE);
56.         }
57.         printf("\n\n");
```

```

57.         printf("\n **NUOVO ARRAY PER UNA NUOVA OPERAZIONE**\n");
58.         for(i=0; i<n; i++)
59.         {
60.             fflush(stdin);
61.             A[i]=(unsigned short)(rand()%10);
62.         }
63.         for(i=0; i<n; i++)
64.             printf(" [%hd]",A[i]);
65.             printf("\n\n");
66.     }while(scelta!=4);
67.     return 0;
68. }
69.
70.
71.             /**FUNCTION*/
72. /*SOMMA ITERATIVA DELLE COMPONENTI DI UN ARRAY*/
73. short Somma_iterativa(short A[], short n)
74. {
75.     short somma=0,i=0;
76.     for(i=0; i<n; i++)
77.         somma+=A[i];
78.     return somma;
79. }
80.
81. /*SOMMA RICORSIVA LINEARE, OVVERO SINGOLA AUTOATTIVAZIONE, DELLE COMPONENTI DI UN ARRAY*/
82. short Somma_ric_lin(short A[], short n)
83. {
84.     /**CASO BANALE*/
85.     if(n==1)//se c'e' un solo elemento nell'array
86.         return A[0];
87.     /**AUTOATTIVAZIONE*/
88.     else
89.         //sommiamo ricorsivamente partendo
90.         //dall'ultimo elemento(n-1)
91.         //fino a quando n non sara' 1.
92.         return A[n-1]+Somma_ric_lin(A,n-1);
93. }
94.
95. /*SOMMA RICORSIVA BINARIA(APPROCCIO DIVIDE ET IMPERA), OVVERO CON LA SUDDIVISIONE DEI PROBLEMI
96. IN SOTTOPROBLEMI PIU' SEMPLICI DA RISOLVERE, DELLE COMPONENTI DI UN ARRAY*/
97. short Somma_ric_bin(short A[], short n)
98. {
99.     short mediano=0;
100.    /**CASO BANALE*/
101.    if(n==1)//se c'e' un solo elemento nell'array
102.        return A[0];
103.    /**AUTOATTIVAZIONE*/
104.    else
105.    {
106.        //troviamo ogni volta l'elemento mediano
107.        mediano=(n-1)/2;
108.        return Somma_ric_bin(A,mediano+1)+Somma_ric_bin(A+mediano+1,n-mediano-1);
109.    }
110. }

```

## ESECUZIONE 1

```
**SOMMA ITERATIVA E RICORSIVA DI 5 COMPONENTI DI UN ARRAY**
```

```
**ARRAY IN INPUT**  
[3] [6] [7] [0] [6]
```

```
**MENU**  
[1] Somma iterativa  
[2] Somma ricorsiva lineare  
[3] Somma ricorsiva binaria  
[4] Exit  
Inserire scelta: 1
```

```
**SOMMA ITERATIVA**  
[3] [6] [7] [0] [6] = 22
```

```
**NUOVO ARRAY PER UNA NUOVA OPERAZIONE**  
[3] [1] [8] [4] [8]
```

```
**MENU**  
[1] Somma iterativa  
[2] Somma ricorsiva lineare  
[3] Somma ricorsiva binaria  
[4] Exit  
Inserire scelta: 2
```

```
**SOMMA RICORSIVA LINEARE**  
[3] [1] [8] [4] [8] = 24
```

```
**NUOVO ARRAY PER UNA NUOVA OPERAZIONE**  
[6] [5] [6] [6] [5]
```

```
**MENU**  
[1] Somma iterativa  
[2] Somma ricorsiva lineare  
[3] Somma ricorsiva binaria  
[4] Exit  
Inserire scelta: 3
```

```
**SOMMA RICORSIVA BINARIA**  
[6] [5] [6] [6] [5] = 28
```

```
**NUOVO ARRAY PER UNA NUOVA OPERAZIONE**  
[7] [2] [4] [1] [4]
```

```
**MENU**  
[1] Somma iterativa  
[2] Somma ricorsiva lineare  
[3] Somma ricorsiva binaria  
[4] Exit  
Inserire scelta: 4
```

```
Process returned 1 (0x1)    execution time : 15.187 s  
Press ENTER to continue.
```

# Esercizio 64[liv.1]

## Calcolo iterativo e ricorsivo della potenza intera $X^n$ di un numero reale

```
1.  /**[liv.1] Scrivere delle function C (rispettivamente iterativa e ricorsiva) per calcolare
   (con ricorsione sia lineare
2.  sia binaria) la potenza intera  $x^n$  di un numero reale.*/
3.
4.          /**LIBRERIE*/
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8.          /**PROTOTIPI*/
9. short Somma_iterativa(short X, short n);
10. short Somma_ric_lin(short X, short n);
11. short Somma_ric_bin(short X, short n);
12.
13.          /**MAIN*/
14. int main()
15. {
16.     short X=0,n=0,scelta=0,somma_it=0,somma_lin=0,somma_bin=0;
17.
18.     srand((unsigned)time(NULL));
19.     X=(unsigned short)(rand()%15); //GENERA NUMERI INTERI CASUALI DA 0 A 15
20.     n=(unsigned short)(rand()%10); //GENERA NUMERI INTERI CASUALI DA 0 A 10
21.
22.     printf("\n**SOMMA ITERATIVA E RICORSIVA DELLA POTENZA INTERA DI [%hd^%hd] DI UN NUMERO
      REALE**",X,n);
23.     printf("\n**DATI IN INPUT**");
24.     printf("\n X= %hd\n n= %hd",X,n);
25.
26.     do
27.     {
28.         printf("\n **MENU**\n [1] Somma iterativa\n [2] Somma ricorsiva lineare\n [3] Somma
      ricorsiva binaria\n [4] Exit\n Inserire scelta: ");
29.         fflush(stdin);
30.         scanf("%hd",&scelta);
31.         switch(scelta)
32.         {
33.             case 1: printf("\n **SOMMA ITERATIVA**\n");
34.                     printf(" X^n= %hd",somma_it=Somma_iterativa(X,n));
35.                     break;
36.             case 2: printf("\n **SOMMA RICORSIVA LINEARE**\n");
37.                     printf(" X^n= %hd",somma_lin=Somma_ric_lin(X,n));
38.                     break;
39.             case 3: printf("\n **SOMMA RICORSIVA BINARIA**\n");
40.                     printf(" X^n= %hd",somma_bin=Somma_ric_bin(X,n));
41.                     break;
42.             case 4: exit(EXIT_FAILURE);
43.         }
44.         printf("\n\n");
45.         X=(unsigned short)(rand()%15);
46.         n=(unsigned short)(rand()%10);
47.         printf("\n **NUOVI DATI PER UNA NUOVA OPERAZIONE**");
48.         printf("\n X= %hd\n n= %hd\n",X,n);
49.     }while(scelta!=4);
50.     return 0;
51. }
52.
53.
54.          /**FUNCTION*/
55. /**CALCOLO DELLA POTENZA INTERA DI X ITERATIVO**/
```

```

56. short Somma_iterativa(short X, short n)
57. {
58.     short somma=1;
59.
60.     while(n!=0)
61.     {
62.         somma=X*somma;
63.         n--;
64.     }
65.     return somma;
66. }
67.
68. /**CALCOLO DELLA POTENZA INTERA DI X RICORSIVO LINEARE*/
69. short Somma_ric_lin(short X, short n)
70. {
71.     /**CASO BANALE**/
72.     if(n==0)
73.         return 1;
74.     else
75.         return X*Somma_ric_lin(X,n-1);
76. }
77.
78. /**CALCOLO DELLA POTENZA INTERA DI X RICORSIVO BINARIO*/
79. short Somma_ric_bin(short X, short n)
80. {
81.     /**CASO BANALE**/
82.     if(n==0)
83.         return 1;
84.     /*verifica che la potenza sia pari*/
85.     else if (n%2==1) /*se non lo e' */
86.         /*autoattivazione*/
87.         return X*Somma_ric_bin(X,n/2)*Somma_ric_bin(X,n/2);
88.     else
89.         return Somma_ric_bin(X,n/2)*Somma_ric_bin(X,n/2);
90. }

```

## ESECUZIONE 1

```
**SOMMA ITERATIVA E RICORSIVA DELLA POTENZA INTERA DI [2^4] DI UN NUMERO REALE**
**DATI IN INPUT**
X= 2
n= 4
**MENU**
[1] Somma iterativa
[2] Somma ricorsiva lineare
[3] Somma ricorsiva binaria
[4] Exit
Inserire scelta: 1

**SOMMA ITERATIVA**
X^n= 16

**NUOVI DATI PER UNA NUOVA OPERAZIONE**
X= 2
n= 6

**MENU**
[1] Somma iterativa
[2] Somma ricorsiva lineare
[3] Somma ricorsiva binaria
[4] Exit
Inserire scelta: 2

**SOMMA RICORSIVA LINEARE**
X^n= 64

**NUOVI DATI PER UNA NUOVA OPERAZIONE**
X= 2
n= 9

**MENU**
[1] Somma iterativa
[2] Somma ricorsiva lineare
[3] Somma ricorsiva binaria
[4] Exit
Inserire scelta: 3

**SOMMA RICORSIVA BINARIA**
X^n= 512

**NUOVI DATI PER UNA NUOVA OPERAZIONE**
X= 12
n= 9

**MENU**
[1] Somma iterativa
[2] Somma ricorsiva lineare
[3] Somma ricorsiva binaria
[4] Exit
Inserire scelta: 4

Process returned 1 (0x1)  execution time : 17.807 s
Press ENTER to continue.
```

# Esercizio 69[liv.1]

## Selection Sort(iterativo e ricorsivo) su un array di struttura con scambi reali e virtuali

```
1.  /**[liv.1] Scrivere due function C (rispettivamente iterativa e ricorsiva) per implementare
2.  l'algoritmo Selection
3.
4.           /**LIBRERIE**/
5.  #include <stdio.h>
6.  #include <stdlib.h>
7.
8.  #define SIZE 5
9.
10. typedef struct
11. {
12.     short dato;
13. }SELECT;
14.           /**PROTOTIPI**/
15. void Selection_it_real(SELECT A[], short n);
16. void Selection_it_virt(SELECT *A[], short n);
17. void Selection_ric_real(SELECT A[], short n, short inizio);
18. void Selection_ric_virt(SELECT *A[], short n, short inizio);
19. void Swap(short *a, short *b);
20.
21.           /**MAIN**/
22. int main()
23. {
24.     SELECT *A_punt[SIZE],A[SIZE];
25.     short i=0,k=0,n=0,scelta;
26.     n=SIZE;
27.     /**INIZIALIZZAZIONE DELL'ARRAY**/
28.     do{
29.         printf("\n **INSERIMENTO ELEMENTI NELL'ARRAY**\n");
30.         for(i=0; i<n; i++)
31.         {
32.             printf(" Inserisci elemento %d: ",i+1);
33.             fflush(stdin);
34.             scanf("%hd",&A[i].dato);
35.         }
36.         /**INIZIALIZZO L'ARREY DI PUNTATORI CON GLI INDIRIZZI DI MEMORIA DELL'ARAY STATICO*
37. */
38.         for(i=0; i<n; i++)
39.             A_punt[i]=&A[i];
40.
41.         printf("\n **SELECTION SORT CON SCAMBI REALI E VIRTUALI**\n ARRAY IN INPUT: ");
42.         for(i=0; i<n; i++)
43.             printf(" [%hd]",A[i].dato);
44.
45.             printf("\n\n **MENU**\n [1] Iterativo con scambi reali\n [2] Iterativo con scambi
46. virtuali\n"
47.             " [3] Ricorsivo con scambi reali\n [4] Ricorsivo con scambi virtuali\n [5] E
48. xit\n Inserire scelta: ");
49.             scanf("%hd",&scelta);
50.             switch(scelta)
51.             {
52.                 case 1: printf("\n **ITERATIVO CON SCAMBI REALI**\n");
53.                         Selection_it_real(A,n);
54.                         break;
55.                 case 2: printf("\n **ITERATIVO CON SCAMBI VIRTUALI**\n");
```

```

53.             Selection_it_virt(A_punt,n);
54.             break;
55.         case 3: printf("\n **RICORSIVO CON SCAMBI REALI**\n");
56.             Selection_ric_real(A,n,0);
57.             for(k=0; k<n; k++)
58.                 printf(" [%hd]",A[k].dato);
59.             break;
60.         case 4: printf("\n **RICORSIVO CON SCAMBI VIRTUALI**\n");
61.             Selection_ric_virt(A_punt,n,0);
62.             for(k=0; k<n; k++)
63.                 printf(" [%hd]",A[k].dato);
64.             break;
65.         case 5: exit(EXIT_FAILURE);
66.     }
67.     printf("\n\n");
68. }while(scelta!=5);
69. printf("\n\n");
70. return 0;
71. }

73.         /**FUNCTION**/
74. /*PER QUANTO RIGUARDA LA SELEZIONE DI ELEMENTO MASSIMO O DI ELEMENTO MINIMO,
75. E' STATO SCELTO DI ORDINARE L'ARRAY PER SELEZIONE DELL'ELEMENTO MINIMO IN QUESTO CASO*/
76.
77. /**SELECTION SORT ITERATIVO CON SCAMBI REALI*/
78. void Selection_it_real(SELECT A[], short n)
79. {
80.     short i=0,j=0,k=0,min=0;
81.
82.     //ciclo for che va da 0 ad n-1 per via dello shape dell'array
83.     for(i=0; i<n-1; i++)
84.     {
85.         //inizialmente il minimo e' il primo elemento dell'array
86.         min=A[i].dato;
87.         /*il secondo ciclo for innestato va da j=i+1 perche' ad ogni iterazione,
88.         andremo a confrontare l'elemento i-esimo con quello i-esimo+1*/
89.         for(j=i+1; j<n; j++)
90.         {
91.             //Se l'elemento i+1 e' minore dell'elemento i..swap
92.             if(A[j].dato<min)
93.                 min=i;
94.             Swap(&A[min].dato,&A[j].dato);
95.         }
96.     }
97.     for(k=0; k<n; k++)
98.         printf(" [%hd]",A[k].dato);
99. }

100. /**SELECTION SORT ITERATIVO CON SCAMBI VIRTUALI*/
101. void Selection_it_virt(SELECT *A[], short n)
102. {
103.     short i=0,j=0,k=0,min=0;
104.
105.     for(i=0; i<n-1; i++)
106.     {
107.         min=A[i]->dato;
108.         for(j=i+1; j<n; j++)
109.         {
110.             if(A[j]->dato<min)
111.                 min=i;
112.             Swap(&A[min]->dato,&A[j]->dato);
113.         }
114.     }
115.     for(k=0; k<n; k++)
116.         printf(" [%hd]",A[k]->dato);
117. }

118.

```

```

119.
120. /**SELECTION SORT RICORSIVO CON SCAMBI REALI*/
121. void Selection_ric_real(SELECT A[], short n, short inizio)
122. {
123.     short i=0,min;
124.     /**CASO BASE: se l'inizio che in questo caso e' A[0]
125.     e' maggiore o uguale al size dell'array usciremo dalla ricorsione*/
126.     if(inizio>=n-1)
127.         return;
128.     //il minimo ovviamente e' sempre l'inizio dell'array ovvero A[0]
129.     min=inizio;
130.     /*in questo ciclo for inizio verra' incrementato ad ogni chiamata ricorsiva
131.     quindi sara' un ciclo for da inizio(A[0] inizialmente),inizio+1,+2+3+4...
132.     finche' non sara' maggiore o uguale del size dell'array*/
133.     for(i=inizio; i<n; i++)
134.     {
135.         if(A[i].dato<A[min].dato)
136.         {
137.             min=i;
138.             Swap(&A[min].dato,&A[inizio].dato);
139.         }
140.     }
141.     Selection_ric_real(A,n,inizio+1);
142. }
143.
144. /**SELECTION SORT RICORSIVO CON SCAMBI VIRTUALI*/
145. void Selection_ric_virt(SELECT *A[], short n, short inizio)
146. {
147.     short i=0,min;
148.     if(inizio>=n-1)
149.         return;
150.     min=inizio;
151.     for(i=inizio; i<n; i++)
152.     {
153.         if(A[i]->dato<A[min]->dato)
154.         {
155.             min=i;
156.             Swap(&A[min]->dato,&A[inizio]->dato);
157.         }
158.     }
159.     Selection_ric_virt(A,n,inizio+1);
160. }
161.
162. /**SCAMBI*/
163. void Swap(short *a, short *b)
164. {
165.     short temp;
166.
167.     temp=*a;
168.     *a=*b;
169.     *b=temp;
170. }

```

## ESECUZIONE 1

```
**INSERIMENTO ELEMENTI NELL'ARRAY**
Inserisci elemento 1: 5
Inserisci elemento 2: 4
Inserisci elemento 3: 3
Inserisci elemento 4: 2
Inserisci elemento 5: 1

**SELECTION SORT CON SCAMBI REALI E VIRTUALI**
ARRAY IN INPUT: [5] [4] [3] [2] [1]

**MENU'**
[1] Iterativo con scambi reali
[2] Iterativo con scambi virtuali
[3] Ricorsivo con scambi reali
[4] Ricorsivo con scambi virtuali
[5] Exit
Inserire scelta: 1

**ITERATIVO CON SCAMBI REALI**
[1] [2] [3] [4] [5]

**INSERIMENTO ELEMENTI NELL'ARRAY**
Inserisci elemento 1: 10
Inserisci elemento 2: 9
Inserisci elemento 3: 8
Inserisci elemento 4: 7
Inserisci elemento 5: 6

**SELECTION SORT CON SCAMBI REALI E VIRTUALI**
ARRAY IN INPUT: [10] [9] [8] [7] [6]

**MENU'**
[1] Iterativo con scambi reali
[2] Iterativo con scambi virtuali
[3] Ricorsivo con scambi reali
[4] Ricorsivo con scambi virtuali
[5] Exit
Inserire scelta: 2

**ITERATIVO CON SCAMBI VIRTUALI**
[6] [7] [8] [9] [10]
```

```

**INSERIMENTO ELEMENTI NELL'ARRAY**
Inserisci elemento 1: 15
Inserisci elemento 2: 14
Inserisci elemento 3: 13
Inserisci elemento 4: 12
Inserisci elemento 5: 11

**SELECTION SORT CON SCAMBI REALI E VIRTUALI**
ARRAY IN INPUT: [15] [14] [13] [12] [11]

**MENU'**
[1] Iterativo con scambi reali
[2] Iterativo con scambi virtuali
[3] Ricorsivo con scambi reali
[4] Ricorsivo con scambi virtuali
[5] Exit
Inserire scelta: 3

**RICORSIVO CON SCAMBI REALI**
[11] [12] [13] [14] [15]

**INSERIMENTO ELEMENTI NELL'ARRAY**
Inserisci elemento 1: 100
Inserisci elemento 2: 80
Inserisci elemento 3: 60
Inserisci elemento 4: 40
Inserisci elemento 5: 20

**SELECTION SORT CON SCAMBI REALI E VIRTUALI**
ARRAY IN INPUT: [100] [80] [60] [40] [20]

**MENU'**
[1] Iterativo con scambi reali
[2] Iterativo con scambi virtuali
[3] Ricorsivo con scambi reali
[4] Ricorsivo con scambi virtuali
[5] Exit
Inserire scelta: 4

**RICORSIVO CON SCAMBI VIRTUALI**
[20] [40] [60] [80] [100]

**INSERIMENTO ELEMENTI NELL'ARRAY**
Inserisci elemento 1: 0
Inserisci elemento 2: 0
Inserisci elemento 3: 0
Inserisci elemento 4: 0
Inserisci elemento 5: 0

**SELECTION SORT CON SCAMBI REALI E VIRTUALI**
ARRAY IN INPUT: [0] [0] [0] [0] [0]

**MENU'**
[1] Iterativo con scambi reali
[2] Iterativo con scambi virtuali
[3] Ricorsivo con scambi reali
[4] Ricorsivo con scambi virtuali
[5] Exit
Inserire scelta: 5

Process returned 1 (0x1)  execution time : 53.633 s
Press ENTER to continue.

```

## Esercizio 71 [liv.1]

### Bubble Sort su array di struttura mediante scambi reali e virtuali

```
1.  /***[liv.1] Scrivere una function C per implementare l'algoritmo Bubble Sort su un array di struttura,  
2.    sia mediante scambi reali sia mediante scambi virtuali.**/  
3.  
4.  
5.          /**LIBRERIE**/  
6. #include <stdio.h>  
7. #include <stdlib.h>  
8.  
9. #define SIZE 5  
10.  
11. typedef struct  
12. {  
13.     short dato;  
14. }SELECT;  
15.          /**PROTOTIPI**/  
16. void Bubble_sort_real(SELECT A[], short n);  
17. void Bubble_sort_virt(SELECT *A_punt[], short n);  
18.  
19.          /**MAIN**/  
20. int main()  
21. {  
22.     SELECT A[SIZE],*A_punt[SIZE];  
23.     short i,k,n,scelta=0;  
24.     n=SIZE;  
25.  
26.     do{  
27.         printf("\n **INSERIMENTO ELEMENTI NELL'ARRAY**\n");  
28.         for(i=0; i<n; i++)  
29.         {  
30.             printf(" Inserisci elemento %d: ",i+1);  
31.             fflush(stdin);  
32.             scanf("%hd",&A[i].dato);  
33.         }  
34.  
35.         /**INIZIALIZZO L'ARREY DI PUNTATORI CON GLI INDIRIZZI DI MEMORIA DELL'ARAY STATICO*  
36.     */  
37.     for(i=0; i<n; i++)  
38.         A_punt[i]=&A[i];  
39.  
40.     printf("\n **BUBBLE SORT CON SCAMBI REALI E VIRTUALI**\n ARRAY IN INPUT: ");  
41.     for(i=0; i<n; i++)  
42.         printf(" [%hd]",A[i].dato);  
43.  
44.     printf("\n\n **MENU**\n [1] Con scambi reali\n [2] Con scambi virtuali\n"  
45.             " [3] Exit\n Inserire scelta: ");  
46.     scanf("%hd",&scelta);  
47.     switch(scelta)  
48.     {  
49.         case 1: printf("\n **BUBBLE SORT CON SCAMBI REALI**\n");  
50.                 Bubble_sort_real(A,n);  
51.                 for(k=0; k<n; k++)  
52.                     printf(" [%hd]",A[k].dato);  
53.                     break;  
54.         case 2: printf("\n **BUBBLE SORT CON SCAMBI VIRTUALI**\n");  
55.                 Bubble_sort_virt(A_punt,n);  
56.                 for(k=0; k<n; k++)  
57.                     printf(" [%hd]",A_punt[k]->dato);
```

```

57.         break;
58.     case 3: exit(EXIT_FAILURE);
59.     }
60.     printf("\n\n");
61. }while(scelta!=3);
62. printf("\n\n");
63. return 0;
64.
65. printf("\n\n");
66. return 0;
67. }
68.
69.         /**FUNCTION*/
70. /**CON SCAMBI REALI*/
71. void Bubble_sort_real(SELECT A[], short n)
72. {
73.     short i=0,max;
74.
75.     while(n>0)
76.     {
77.         for(i=0; i<n-1; i++)
78.         {
79.             if(A[i].dato>A[i+1].dato)
80.             {
81.                 max=A[i].dato;
82.                 A[i].dato=A[i+1].dato;
83.                 A[i+1].dato=max;
84.             }
85.         }
86.         n--;
87.     }
88. }
89.
90. /**CON SCAMBI VIRTUALI*/
91. void Bubble_sort_virt(SELECT *A_punt[], short n)
92. {
93.     short i=0,max;
94.
95.     while(n>0)
96.     {
97.         for(i=0; i<n-1; i++)
98.         {
99.             if(A_punt[i]->dato>A_punt[i+1]->dato)
100.             {
101.                 max=A_punt[i]->dato;
102.                 A_punt[i]->dato=A_punt[i+1]->dato;
103.                 A_punt[i+1]->dato=max;
104.             }
105.         }
106.         n--;
107.     }
108. }

```

## ESECUZIONE 1

```
**INSERIMENTO ELEMENTI NELL'ARRAY**
Inserisci elemento 1: 5
Inserisci elemento 2: 4
Inserisci elemento 3: 3
Inserisci elemento 4: 2
Inserisci elemento 5: 1

**BUBBLE SORT CON SCAMBI REALI E VIRTUALI**
ARRAY IN INPUT: [5] [4] [3] [2] [1]

**MENU'**
[1] Con scambi reali
[2] Con scambi virtuali
[3] Exit
Inserire scelta: 1

**BUBBLE SORT CON SCAMBI REALI**
[1] [2] [3] [4] [5]

**INSERIMENTO ELEMENTI NELL'ARRAY**
Inserisci elemento 1: 10
Inserisci elemento 2: 9
Inserisci elemento 3: 8
Inserisci elemento 4: 7
Inserisci elemento 5: 6

**BUBBLE SORT CON SCAMBI REALI E VIRTUALI**
ARRAY IN INPUT: [10] [9] [8] [7] [6]

**MENU'**
[1] Con scambi reali
[2] Con scambi virtuali
[3] Exit
Inserire scelta: 2

**BUBBLE SORT CON SCAMBI VIRTUALI**
[6] [7] [8] [9] [10]

**INSERIMENTO ELEMENTI NELL'ARRAY**
Inserisci elemento 1: 0
Inserisci elemento 2: 0
Inserisci elemento 3: 0
Inserisci elemento 4: 0
Inserisci elemento 5: 0

**BUBBLE SORT CON SCAMBI REALI E VIRTUALI**
ARRAY IN INPUT: [0] [0] [0] [0] [0]

**MENU'**
[1] Con scambi reali
[2] Con scambi virtuali
[3] Exit
Inserire scelta: 3

Process returned 1 (0x1)    execution time : 19.388 s
Press ENTER to continue.
```

## Esercizio 72[liv.1]

### Insertion Sort su array di struttura

```
1.  /**[liv.1] Scrivere una function C per implementare l'algoritmo Insertion Sort su un array
   di struttura.*/
2.
3.          /**LIBRERIE*/
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. #define SIZE 5
8.
9. typedef struct
10. {
11.     short dato;
12. }SELECT;
13.           /**PROTOTIPI*/
14. void Insertion_Sort(SELECT A[], short n);
15.
16.           /**MAIN*/
17. int main()
18. {
19.     SELECT A[SIZE];
20.     short i,n;
21.     n=SIZE;
22.
23.     printf("\n **INSERTION SORT SU UN ARRAY DI STRUCT**\n");
24.     for(i=0; i<n; i++)
25.     {
26.         printf(" Inserire elemento %d: ",i+1);
27.         scanf("%hd",&A[i].dato);
28.     }
29.     printf("\n **ARRAY IN INPUT**\n");
30.     for(i=0; i<n; i++)
31.         printf(" [%hd]",A[i].dato);
32.     Insertion_Sort(A,n);
33.     printf("\n\n");
34.     return 0;
35. }
36.
37.           /**FUNCTION*/
38. /**IMPLEMENTAZIONE INSERTION SORT*/
39. void Insertion_Sort(SELECT A[], short n)
40. {
41.     short key=0, j=0, i=0;
42.
43.     /*partiamo da i=1, perche' nel successivo ciclo while,
44.      andremo a scambiare l'elemento i-esimo con quello i-esimo-1,
45.      ovviamente de quello i-esimo-1 sara' minore di quello i-esimo
46.      e fino a quando j, che verra' decrementato ad ogni iterazione,
47.      sara' maggiore di 0.*/
48.     for(i=1; i<n; i++)
49.     {
50.         j=i;
51.         while(j>0 && A[j-1].dato>A[j].dato)
52.         {
53.             key=A[j].dato;
54.             A[j].dato=A[j-1].dato;
55.             A[j-1].dato=key;
56.             j--;
57.         }
58.     }
59.     printf("\n **ARRAY ORDINATO**\n");
```

```
60.     for(int k=0; k<n; k++)
61.     {
62.         printf(" [%hd]",A[k].dato);
63.     }
64. }
```

## ESECUZIONE 1

```
**INSERTION SORT SU UN ARRAY DI STRUCT**
Inserire elemento 1: 5
Inserire elemento 2: 4
Inserire elemento 3: 3
Inserire elemento 4: 2
Inserire elemento 5: 1

**ARRAy IN INPUT**
[5] [4] [3] [2] [1]
**ARRAY ORDINATO**
[1] [2] [3] [4] [5]
```

```
Process returned 0 (0x0)    execution time : 4.072 s
Press ENTER to continue.
```

## Esercizio 73[liv.2]

### Merge Sort iterativo e ricorsivo

```
1.  /**[liv.2] Scrivere function C per implementare l'algoritmo Merge Sort su un array in versi  
one iterativa e ricorsiva.**/  
2.  
3.          /**LIBRERIE**/  
4. #include <stdio.h>  
5. #include <stdlib.h>  
6.  
7.          /**PROTOTIPI**/  
8. int min(int, int);  
9. void Merge_Sort_it(int*, int);  
10. void Merge(int *, int , int , int);  
11. void MergeSort(int *, int , int);  
12.  
13.          /**MAIN**/  
14. int main()  
15. {  
16.     int n,*A,i,scelta=0;  
17.     printf("** ALGORITMO MERGE SORT **\n");  
18.     //INSERIMENTO DATI  
19.     printf("Inserisci dimensione dell'array\n->");  
20.     scanf("%d",&n);  
21.     A=malloc(sizeof(int)*n);  
22.     for(i=0; i<n; i++)  
23.     {  
24.         printf("Inserisci elemnto %d: ",i);  
25.         scanf("%d",A+i);  
26.     }  
27.     if(A==NULL)  
28.     {  
29.         printf("Allocazione fallita!\n");  
30.         return 0;  
31.     }  
32.     printf("\n[1] Iterativo\n[2] Ricorsivo\nInserire scelta: ");scanf("%d",&scelta);  
33.     switch(scelta)  
34.     {  
35.         case 1: Merge_Sort_it(A,n);  
36.             break;  
37.         case 2: MergeSort(A,0,n-1);  
38.             break;  
39.     }  
40.     //STAMPA DELL'ARRAY  
41.     printf("\nArray Ordinato:\n");  
42.     for(i=0;i<n;i++)  
43.     {  
44.         printf("[%d] ",A[i]);  
45.     }  
46.     return 0;  
47. }  
48.  
49.          /**FUNCTION**/  
50. int min(int a, int b)  
51. {  
52.     if(a<b)  
53.         return a;  
54.     else  
55.         return b;  
56. }  
57.  
58. /**IMPLEMENTAZIONE MERGE SORT ITERATIVO**/  
59. void Merge_Sort_it(int *A, int n)
```

```

60. {
61.     int left,mid=0,right=0,curr;
62.     for(curr=1;curr<=n-1;curr=curr*2)
63.     {
64.         for(left=0;left<n-1; left+=curr*2)
65.         {
66.             mid=left+curr-1;
67.             right=min(left+2*curr-1,n-1);
68.             Merge(A,left,mid,right);
69.         }
70.     }
71. }
72.
73. /**La seguente procedura, chiamata ricorsivamente, ordina ad ogni passo una porzione dell'array in input(A).
74.     PARAMETRI DI INPUT:
75.         - int *A: array allocato dinamicamente.
76.         - int p: indice iniziale della porzione di array da ordinare.
77.         - int mediano: usato per stabilire la lunghezza dei due array L ed R ad ogni passo,
e stabilisce l*/
78. void Merge(int *A, int p, int mediano, int n)
79. {
80.
81.     int i,j,k,n1=0,n2=0, *L,*R;
82.     n1=mediano-p+1;
83.     n2=n-medano;
84.
85.     L=malloc(sizeof(int)*n1+1);
86.     if(L==NULL)
87.     {
88.         exit(1);
89.     }
90.
91.     R=malloc(sizeof(int)*n2+1);
92.     if(R==NULL)
93.     {
94.         exit(1);
95.     }
96.
97.     for(i=0; i<n1; i++)
98.         L[i]=A[p+i];
99.
100.    for(j=0; j<n2; j++)
101.        R[j]=A[mediano+j+1];
102.
103.    /** NODO SENTINELLA CON VALORE MASSIMO **/
104.    L[n1]=200;
105.    R[n2]=200;
106.    i=0;
107.    j=0;
108.
109.    for(k=p; k<=n; k++)
110.    {
111.        if(L[i]<=R[j])
112.        {
113.            A[k]=L[i];
114.            i++;
115.        }else
116.        {
117.            A[k]=R[j];
118.            j++;
119.        }
120.
121.    }
122. }
123.

```

```
124. void MergeSort(int *A, int p, int n)
125. {
126.     int mediano;
127.     if(p < n)
128.     {
129.         mediano=(p+n)/2;
130.         MergeSort(A,p,mediano);
131.         MergeSort(A,mediano+1,n);
132.         Merge(A,p,mediano,n);
133.     }
134. }
```

## ESECUZIONE 1(iterativo)

```
** ALGORITMO MERGE SORT **  
Inserisci dimensione dell'array  
->5  
Inserisci elemnto 0: 5  
Inserisci elemnto 1: 4  
Inserisci elemnto 2: 3  
Inserisci elemnto 3: 2  
Inserisci elemnto 4: 1  
  
[1] Iterativo  
[2] Ricorsivo  
Inserire scelta: 1  
  
Array Ordinato:  
[1] [2] [3] [4] [5]  
Process returned 0 (0x0)    execution time : 4.547 s  
Press ENTER to continue.  
■
```

## ESECUZIONE 2(Ricorsivo)

```
** ALGORITMO MERGE SORT **  
Inserisci dimensione dell'array  
->5  
Inserisci elemnto 0: 5  
Inserisci elemnto 1: 4  
Inserisci elemnto 2: 3  
Inserisci elemnto 3: 2  
Inserisci elemnto 4: 1  
  
[1] Iterativo  
[2] Ricorsivo  
Inserire scelta: 2  
  
Array Ordinato:  
[1] [2] [3] [4] [5]  
Process returned 0 (0x0)    execution time : 3.354 s  
Press ENTER to continue.  
■
```