ISSN 2239-7035

dicecca.net - cs

Anno I - № 9 – Supplemento al MONITORE NAPOLETANO - № 151 – Settembre 2020

GIOVANNI DI CECCA

Programmi in Linguaggio C per l'esame di: Programmazione Mod. B

dicecca.net
web site

www.monitorenapoletano.it/dicecca.net-cs

Direttore Responsabile: Giovanni Di Cecca

Collana dicecca.net – Computer Science

Anno I - № 9 – Supplemento al Numero 151 – Settembre 2020

Periodico Mensile Registrato presso il Tribunale di Napoli № 45 dell'8 giugno 2011

ISSN: 2239-7035

Premessa

I programmi riportati di seguito, sono stati sviluppati con l'editor di testo Unix vi.

Per poter implementare gli esempi d'uso i sorgenti sono stati compilati con il compilatore Borland C++ 5.02 for Windows 95 (in ambiente Windows ME)

Per una migliore leggibilità gli algoritmi sono stati suddivisi in categorie (come proposto nell'indice alla fine del fascicolo).

I programmi sono stati sviluppati per la stragrande maggioranza con allocazione dinamica e soprattutto negli algoritmi ho implementato (in quasi tutti i programmi) il concetto di modularità insito nel Linguggio C, dividendo in 2 o più file il programma.

Algoritmi di Ordinamento

Insertion Sort – Iterativo

(ordinamento per Inserzione)

- **Scopo:** Il programma permette di ordinare un vettore monodimensionale mediante il metodo dell'inserzione;

- **Specifiche della funzione:**

  File libreria esterna "`inssort.c`"

  Funzione:

  ```
  void inssort(int *arr, int arrln)
  ```

  dove:

  ***arr** è la grandezza dell'array di dimensione dinamica

  arrln è il numero dei valori contenuti nell'array

- **Breve descrizione dell'algoritmo:** l'approccio seguito da questo algoritmo ricorda molto il metodo usato dai giocatori per ordinare le carte. L'idea infatti è quella di costruire la soluzione completa prendendo di volta in volta un elemento dalla parte dell'insieme non ancora ordinata, e inserirlo nella parte ordinata che aumenta così di un elemento. Successivamente, l'elemento che in precedenza occupava la posizione di minimo viene spinto verso l'estrema destra, così da avere alla fine un insieme completamente ordinato.

- **Complessità computazionale:** le operazioni dominanti dell'algoritmo sono confronti e inserimenti. Nel caso peggiore, quando cioè bisogna riordinare successioni ordinate in senso opposto, l'andamento riscontrato della complessità asintotica è $O(n^2)$ [quadratico]. Invece, nel caso migliore, cioè quando l'insieme è già ordinato, la complessità risulta O(n) [lineare]

- **Raccomandazioni d'uso:** Il programma è consigliato solo per ordinare un piccolo insieme di dati. Inoltre la funzione poggia sul file libreria esterna "`scambia.c`"

File libreria "inssort.c"

```
/* Inclusione della libreria utente scambia */
#include "scambia.c"

/* Funzione Insertion Sort */
void inssort (int *arr,int arrln)
{
      int i,j,k,temp,min=0;

      for (i=1;i<arrln;i++)
            if (arr[i]<arr[min]) min=i;

   /* Carica la funzione di Scambia */
   swap (arr,arr+min);

      j=1;

      for (i=2;i<arrln;j=i++)
      {
         if (arr[i]<arr[j])
         {
            temp=arr[i];
            k=i;
            do
            {
                  arr[k]=arr[k-1];
                  k--;
            } while (arr[k-1]>temp);

            arr[k]=temp;
         }
      }
}
```

File libreria generale di scambio "`scambio.c`"

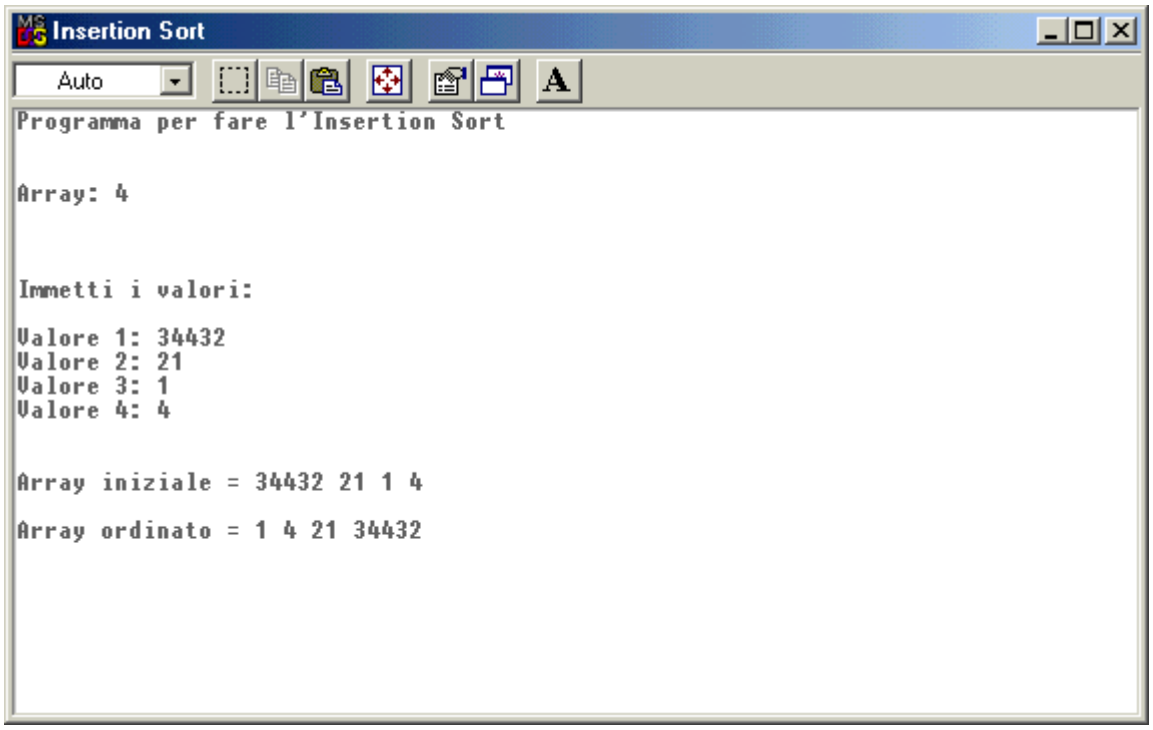
Valido per tutti i programmi che richiedono lo scambio i variabili

```
/* varibile di scambia valori per riferimento*/
void scambiar(int &x, int &y)
{
      int temp;

      temp=x;   /* Variabile d'aiuto */

      x=y;
      y=temp;
}


/* varibile di scambia valori */
void scambia(int x, int y)
{
      int temp;

      temp=x;   /* Variabile d'aiuto */

      x=y;
      y=temp;
}


/* varibile di scambio per puntatori */
void swap (int *a, int *b)
{
      int temp;
      temp=*a;
      *a=*b;
      *b=temp;
}
```

File d'esempio che permette l'uso della libreria "inssort.c"

```
/* Parte preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Liberire utente */
#include "inssort.c"


main()
{
   int *a;
   int n_elem,i;
   printf("Programma per fare l'Insertion Sort");

   /* Inserimento della grandezza dell'Array */
   printf("\n\n Array: ");
   scanf ("%d",&n_elem);

   /* Allocazione dinamica dell'Array */
   a=(int*) malloc(sizeof(int)*n_elem);

   /* Inserimento dei valori nell'Array */
   printf("Immetti i valori: \n\n");
   for (i=0; i<n_elem; i++)
   {
      printf("\nValore ");
      printf("%d",i+1);
      printf(": ");
      scanf("%d",&a[i]);
   }

   /* Visualizzazione dei dati nell'Array */
   printf("\nArray iniziale = ");
   for (i=0; i<n_elem; i++)
   {
      printf("%d",a[i]);
      printf(" ");
   }


   /* Carica  la funzione Insertion Sort*/
   inssort (a, n_elem);


   /* Visualizza Array ORDINATO */
   printf("\n\n\nArray ordinato = ");
   for (i=0; i<n_elem; i++)
   {
      printf("%d",a[i]);
      printf(" ");
   }

   /* Libera la dimensione dell'Array */
   free(a);

}
```

Esempio d'uso



Insertion Sort – Esempio d'uso

Insertion Sort – Ricorsivo

(ordinamento per Inserzione)

- **Scopo:** il presente programma si propone di ordinare in modo non decrescente un insieme di n elementi disposti in modo casuale utilizzando il metodo di inserzione;

- **Specifiche della funzione:**

  File libreria esterna "`inssort-ric.c`"

  Funzione:

  ```
  void INSERTION_SORT_ric (int *M, int k, int n)
  ```

  dove:

  ***M** è la grandezza dell'array di dimensione dinamica

  k è Indice del vettore da ordinare (che all'atto del richiamare la funzione deve assumere valore 0)

  n è il numero dei valori contenuti nell'array

- **Breve descrizione dell'algoritmo:** Il vettore m di n componenti e' ordinato da 0 a k. Se k=n-1, tutto il vettore e' ordinato. Se k<(n-1), sia j (0<=j<=k) la posizione del primo elemento maggiore di v = M[k+1]. Il vettore ordinato si ottiene spostando gli elementi da j a k di una posizione a destra e copiando v in M[j]. Il resto del vettore si ordina richiamando la funzione con (M, k+1, n). La determinazione del valore di j e gli spostamenti si possono fare in una sola scansione della parte di vettore 0,...,k. Per non perdere valori, gli spostamenti si devono fare procedendo da k a j;

- **Complessità computazionale:** le operazioni dominanti dell'algoritmo sono confronti e inserimenti. Nel caso peggiore, quando cioè bisogna riordinare successioni ordinate in senso opposto, l'andamento riscontrato della complessità asintotica è $O(n^2)$ [quadratico]. Invece, nel caso migliore, cioè quando l'insieme è già ordinato, la complessità risulta O(n) [lineare]

- **Raccomandazioni d'uso:** Il programma è consigliato solo per ordinare un piccolo insieme di dati.

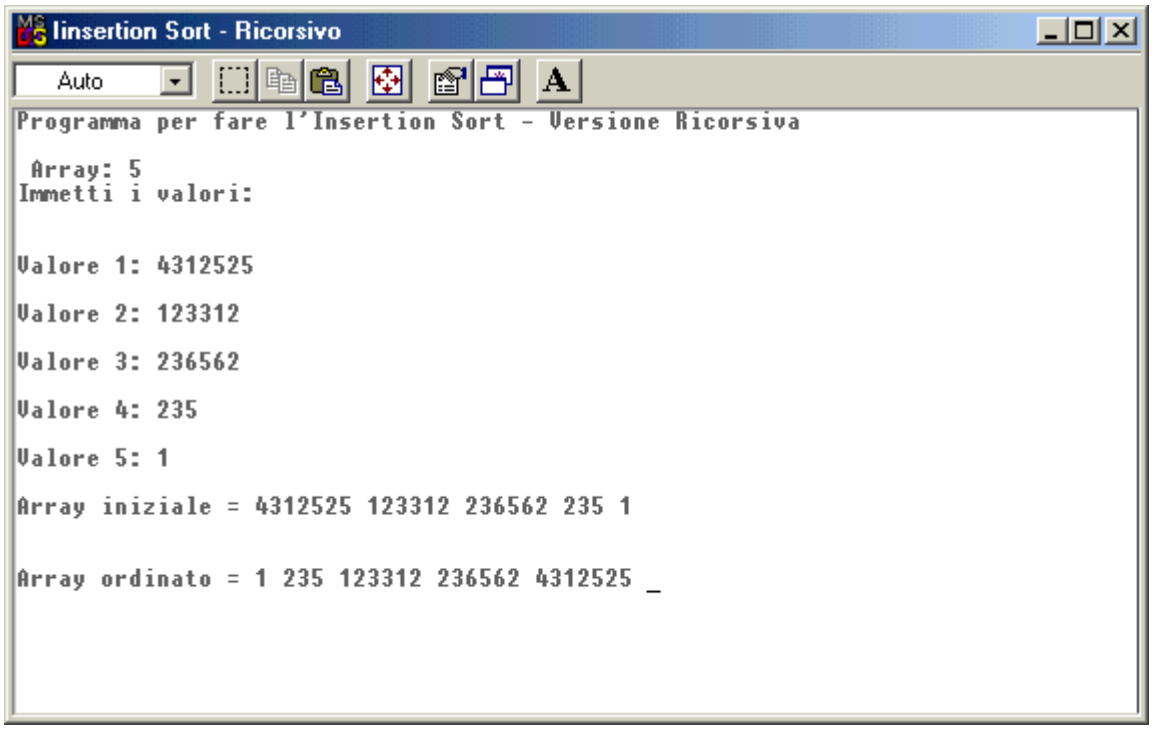
File libreria "inssort-ric.c"

```
/* Funzione Insertion sort Ricorsivo */
void INSERTION_SORT_ric (int *M, int k, int n)
{

      /* dichiarazione variabili locali */
      int j,v,i; /* indici di lavoro */

      if( k < n-1 )
      {
            j = 0;
            v = M [k+1];

            while( j <= k && M[j] <= v )
            j++;
            for(i=k;i>=j;i--)
            {
                  M[i+1] = M[i];    /* crea lo spazio */
                  M[j]=v;           /* inserisce */
            }

            /* Chiamata ricorsiva che ordina la parte restante */
            INSERTION SORT ric (M, k+1, n);
      }

}
```

File d'esempio che permette l'uso della libreria "inssort-r.c"

```
/* Parte preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Liberire utente */
#include "inssort-ric.c"


main()
{
   int *a;
   int n_elem,i;
   printf("Programma per fare l'Insertion Sort - Versione Ricorsiva");

   /* Inserimento della grandezza dell'Array */
   printf("\n\n Array: ");
   scanf ("%d",&n_elem);

   /* Allocazione dinamica dell'Array */
   a=(int*) malloc(sizeof(int)*n_elem);

   /* Inserimento dei valori nell'Array */
   printf("Immetti i valori: \n\n");
   for (i=0; i<n_elem; i++)
   {
      printf("\nValore ");
      printf("%d",i+1);
      printf(": ");
      scanf("%d",&a[i]);
   }

   /* Visualizzazione dei dati nell'Array */
   printf("\nArray iniziale = ");
   for (i=0; i<n_elem; i++)
   {
      printf("%d",a[i]);
      printf(" ");
   }


  /* Carica  la funzione Insertion Sort ricorsiva */
  INSERTION SORT ric (a,0,n elem);


   /* Visualizza Array ORDINATO */
   printf("\n\n\nArray ordinato = ");
   for (i=0; i<n_elem; i++)
   {
      printf("%d",a[i]);
      printf(" ");
   }

   /* Libera la dimensione dell'Array */
   free(a);

}
```

Esempio d'uso



Insertion Sort – Ricorsivo
Esempio d'uso

Bubble Sort

(ordinamento per Interscambi)

- **Scopo:** il programma permette di ordinare un vettore monodimensionale mediante il metodo dello interscambi;

- **Specifiche della funzione:**

  File libreria esterna "`bubblesort.c`"

  Funzione:

  ```
  void bubblesort (int arrln, int *arr)
  ```

  dove:

  ***arr** è la grandezza dell'array di dimensione dinamica

  arrln è il numero dei valori contenuti nell'array

- **Breve descrizione dell'algoritmo:** l'algoritmo procede nell'ordinamento per scambi fin quando tutto l'array non è stato ordinato, inserendo i valori più piccoli a sinistra, verso i più pesanti a destra.

- **Complessità computazionale:** la complessità di tempo di questo algoritmo dipende dal valore dei dati forniti, i quali influenzano il numero di confronti e scambi da effettuare. Infatti, nel caso migliore(elementi già ordinati) si effettueranno solo n-1 confronti e nessuno scambio. Se invece l'array è completamente disordinato, allora bisognerà effettuare n-1 passaggi ed un numero di confronti e scambi pari a n(n-1)/2

- **Raccomandazioni d'uso:** Il programma è consigliato solo per ordinare un piccolo insieme di dati, in quanto il metodo di ordinamento è basato sugli scambi. Ordinare un insieme più ampio fa aumentare il tempo di ordinamento. Inoltre la funzione poggia sul file libreria esterna "`scambia.c`"

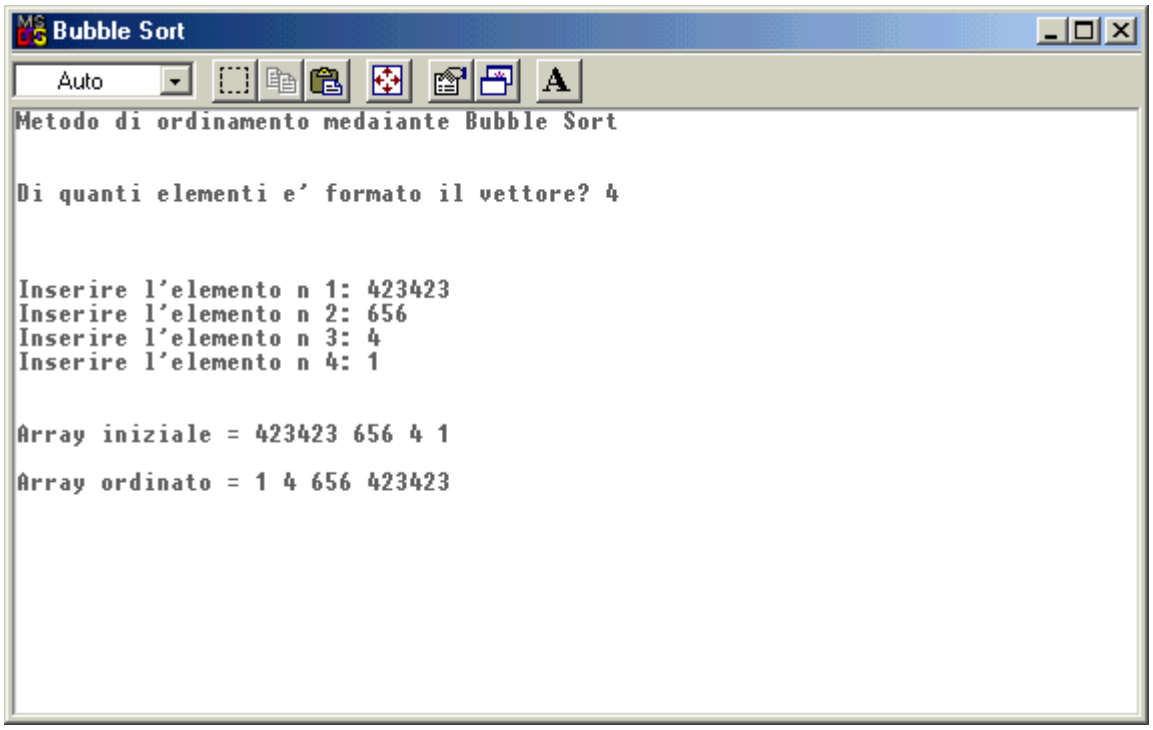
File libreria "bubblesort.c"

```
/* Inclusione della libreria scambia */
#include "scambia.c"

/* Definizione del tipo Booleano */
typedef enum{Vero, Falso}boolean;


void bubblesort (int arrln, int *arr)
{
      int i=0, j, temp;

      /* Uso del tipo booleano precedentemente prototipato */
      boolean scambio=Vero;


      while ((i<arrln-1)&&(scambio==Vero))
      {
            scambio=Falso;
            for (j=0;j<(arrln-i-1);j++)
                  if(arr[j]>arr[j+1])
                  {
                        scambio=Vero;

                        /* Uso del file libreria esterna scambia.c */
                        scambiar (arr[j], arr[j+1]);
                  }

            i++;
      }
}
```

File d'esempio che permette l'uso della libreria "bubblesort.c"

```
/* Parte Preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "bubblesort.c"


main()
{
      int *a;
      int i=0, j, n, k;

      printf("Metodo di ordinamento mediante Bubble Sort\n\n");
      printf("Di quanti elementi e' formato il vettore? ");
      scanf("%d",n);             /* La variabile globale n viene aggornata */

      /* definizione di array dinamico */
      a=(int*) malloc(sizeof(int)*n);

      printf("\n\n\n");

      /* inizio cicloc for per inserire i dati */
      for (k=0; k<=n-1; k++)
      {

            printf("\nInserisci l'elemento n ",k+1,": ");
            scanf("%d",a[k]);
      }

       /* Visualizzazione del vettore immesso */
       printf("Il vettore contiene\n\n");
       for (k=0; k<n; k++)
            printf("%d",a[k]," ");

       printf("\n");

       /* Funzione BubbleSort */
       bubblesort (n, a);

       /* Visualizzazione del vettore ordinato */
       printf("\n\n\n");
       printf("Vettore ordinato\n\n");
       for(i=0; i<n;i++)
       printf("%d",a[i]," ");

}
```

Esempio d'uso



Bubble Sort – Esempio d'uso

Selection Sort

(ordinamento per Selezione)

- **Scopo: i**l programma permette di ordinare un vettore monodimensionale di n elementi disposti casualmente utilizzando il metodo di selezione;

- **Specifiche della funzione:**

  File libreria esterna "`selsort.c`"

  Funzione:

  ```
  void selsort(int n, int *a)
  ```

  dove:

  ***a** è la grandezza dell'array di dimensione dinamica

  n è il numero dei valori contenuti nell'array

- **Breve descrizione dell'algoritmo:** l'algoritmo, dopo aver definito l'array, lo considera suddiviso in due parti; una parte disordinata ed una ordinata. Utilizzando due cicli for innestati viene ricercato l'elemento minimo e spostato sempre verso sinistra, in modo da ottenere alla fine un array completamente ordinato;

- **Complessità computazionale:** le operazioni dominanti dell'algoritmo sono confronti e scambi. Gli scambi sono n-1, perché pari al numero di volte che viene eseguito il ciclo for più esterno. I confronti sono invece pari a n(n-1)/2;

- **Raccomandazioni d'uso:** Il programma è consigliato solo per ordinare un piccolo insieme di dati.

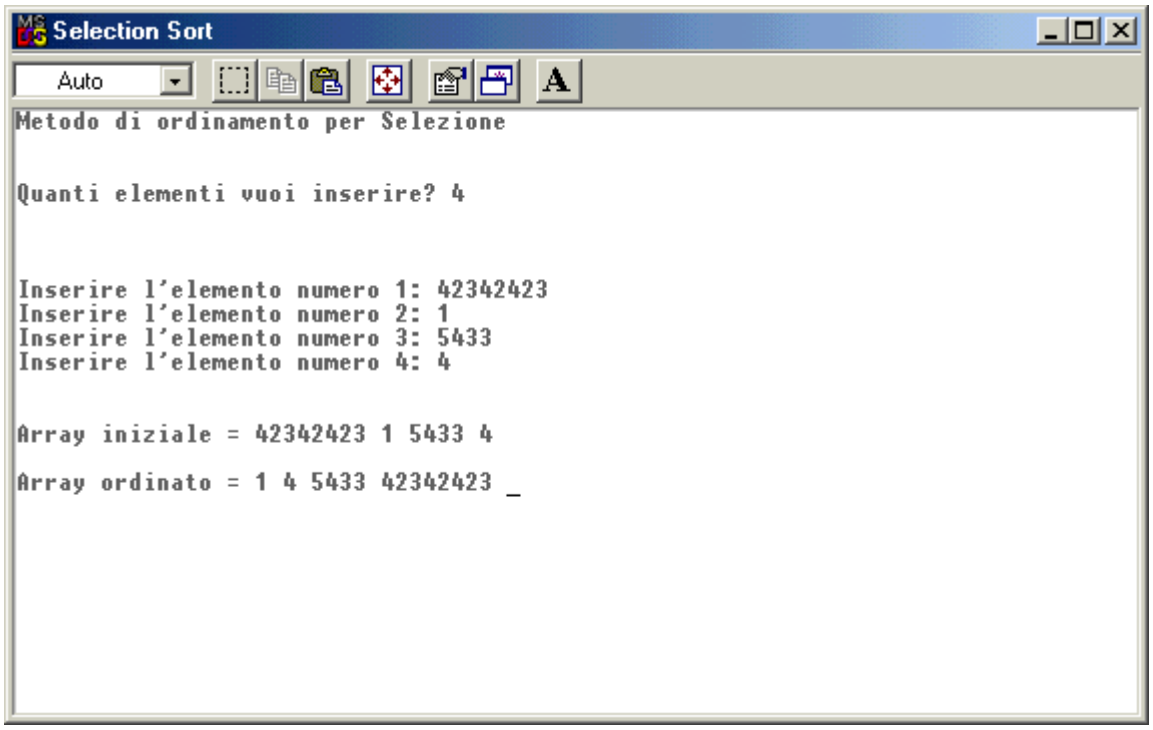
File libreria "`selsort.c`"

```
/* inizio funzione Selection Sort */
void selsort(int n,int *a])
{
      int min,i,j;

      for (i=0; i<n; i++)
      {
            /* Associa alla variabile min il primo elemento dell'array */
            min = a[i];

            /* Ricerca il minimo nell'array */
            for (j=i+1;j<n;j++)
            {
                    /* Associa a min il valore minimo */
                    if (a[j] < min)
                    {
                        min=a[j];
                        a[j]=a[i];
                        a[i]=min;
                    }
            }
      }
}
```

File d'esempio che permette l'uso della libreria "selsort.c"

```
/* Parte Preprocessore */
#include <stdio.h>

/* Libreria utente */
#include "selsort.c"

void main()
{
      /* Dichiarazione variabili */
      int *a;
      int n,i;

      printf ("Metodo di ordinamento per Selezione");

      /* dimensione array */
      printf ("\n\n\nQuanti elementi vuoi inserire? ");
      scanf ("%d",&n);

      /* definizione di array dinamico */
      a=(int*) malloc(sizeof(int)*n);

      printf ("\n\n\n");

      for (i=0;i<n;i++)
      {
            printf("Inserire l'elemento numero %d: ",i+1);
            scanf("%d",&a[i]);
      }


      /* Visualizzazione dei dati nell'Array */
      printf("\n\nArray iniziale = ");
      for (i=0; i<n; i++)
      {
            printf("%d",a[i]);
            printf(" ");
      }

      /* chiamata della funzione Selection Sort */
      selsort(n,a);

      /* Visualizzazione dei dati nell'Array */
      printf("\n\nArray ordinato = ");
      for (i=0; i<n; i++)
      {
            printf("%d",a[i]);
            printf(" ");
      }

      /* Libera la dimensione dell'Array */
      free(a);
}
```

Esempio d'uso



Selection Sort – Esempio d'uso

Shell Sort

(ordinamento mediante il metodo di Shell)

- **Scopo:** ordinare in modo crescente un insieme di n elementi disposti casualmente utilizzando il metodo di Shell di inserzione con diminuzione degli incrementi;

- **Specifiche della funzione:**

  File libreria esterna "`shsort.c`"

  Funzione:

  ```
  void shsort(int *arr, int arrln)
  ```

  dove:

  ***arr** è la grandezza dell'array di dimensione dinamica

  arrln è il numero dei valori contenuti nell'array

- **Breve descrizione dell'algoritmo:** utilizzando tre cicli while ed un ciclo for, la funzione suddivide l'insieme da ordinare in tante catene di due elementi poste a distanza dist; successivamente, ad ogni passo tali catene vengono ordinate utilizzando il metodo di inserzione;

- **Complessità computazionale:** le operazioni eseguite dalla funzione sono confronti ed inserimenti, e il loro numero dipende dal valore dei dati di input;

- **Raccomandazioni d'uso:** adatto per ordinare ampi insiemi di dati, anche se esistono metodi dalle prestazioni migliori. Poiché esso si presenta più complicato degli altri ordinamenti, è adatto per dimostrare l'importanza di un progetto top down corretto e pulito. Inoltre la funzione poggia sui file libreria esterna "`scambia.c`" e "`inssort.c`"

File libreria "shsort.c"

```
/* Carica le librerie utente */
#include "scambia.c"
#include "inssort.c"

/* Funzione ShellSort */
void shsort (int *arr, int arrln)
{
      int i,j,n;
      n=arrln/2;
      i=0;
      j=0;
      while (n>1)
      {
            do
            {
                  if (arr[i]>arr[j])
                  {
                        /* Carica la libreria di scambia.c */
                        swap (&arr[i], &arr[j]);
                  }

                  i++;
                  j=n+1;
            } while (j<arrln);

            i=0;
            n=n/2;
            j=n+i;
      }

      /* Carica la libreria inssort.c */
      inssort (arr,arrln);
}
```

File d'esempio che permette l'uso della libreria "shsort.c"

```
/* Parte Preprocessore */
#include <stdio.h>
#include <stdlib.h>

/*Libreria utente */
#include "shsort.c"

main()
{
   int *a;
   int n_elem,i;

   printf("Programma per fare lo Shell Sort");
   printf("\n\n Array: ");
   scanf ("%d",&n_elem);

   /* Allocazione dimanica della memoria */
   a=(int*) malloc(sizeof(int)*n_elem);

   printf("\n\n");

   /* Immissione dati */
   for (i=0; i<n_elem; i++)
   {
      printf("Immetti i valori separati da uno spazio: ");
      scanf("%d",&a[i]);
   }

   printf("\nArray iniziale = ");
   for (i=0; i<n_elem; i++)
      printf("%d ",a[i]);

   /* Carica la Funzione di Shell Sort */
   shsort(a,n elem);


   /* Array ORDINATO */
   printf("\nArray ordinato = ");
   for (i=0; i<n_elem; i++)
      printf("%d ",a[i]);

   /* Libera la variabile */
      free(a);

}
```

Merge Sort

(ordinamento mediante il metodo di Merge)

- **Scopo:** **Scopo:** ordinare un insieme di n elementi utilizzando il metodo denominato merge;

- **Specifiche della funzione:**

  File libreria esterna "`mergesort.c`"

  Funzione:

```
void mergesort(int *vett, int inizio, int medio, int fine);
```

  dove:

  ***vett** è la grandezza dell'array di dimensione dinamica

  inizio è la posizione di inizio dell'array

  medio variabile locale usata per indicare il punto di confine tra le due sequenze separate da ordinare

  fine ultimo elemento dell'array

- **Breve descrizione dell'algoritmo:** la funzione merge sort suddivide l'insieme in 2 sequenze, ordina separatamente ciascuna di esse con una chiamata ricorsiva, e dopo, chiamando la funzione merge, le due sequenze ordinate vengono fuse in un unico insieme finale ordinato;

- **Complessità computazionale:** l'operazione dominante eseguita è il confronto, ma il numero dipende dal valore dei dati di input;

File libreria "mergesort.c"

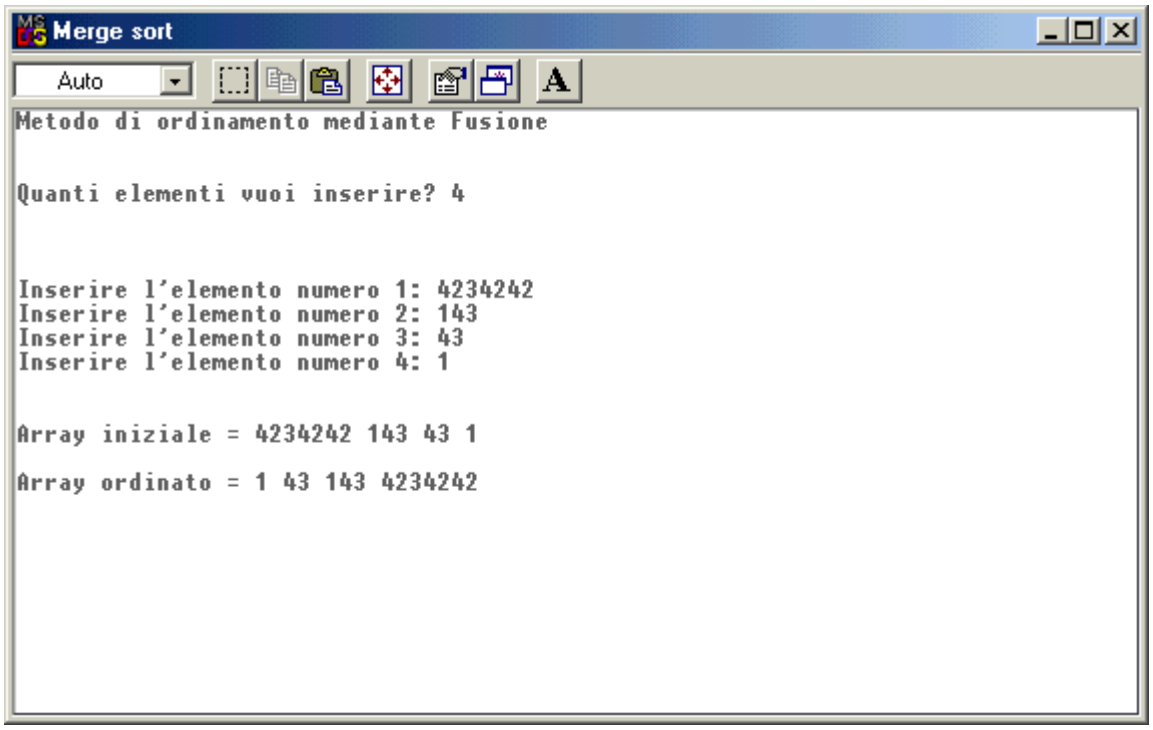
```
/* FUNCTION MERGE ADATTATA ALL'USO DI MERGESORT */

void merge(int *vett, int inizio, int medio, int fine)
{
      int aux_arr[50];
      int i,j,k;

      i=inizio;

      j=medio+1;


       /*
            Il sottovettore di sinistra va da 'i' a 'medio'
            il sottovettore di destra va da 'j=medio+1' a 'fine'.

            Quindi:

            i>medio => il sottovettore di sinistra e' terminato

            j>fine  => il sottovettore di destra e' terminato

            Se si verifica una delle due condizioni vuol dire che tutti gli
            elementi del sottovettore in questione sono già stati opportunamente
            posizionati nel vettore ausiliario e si procederà, a questo punto,
            con la memorizzazione in aux_arr[] dei restanti elementi del
            sottovettore da ordinare(fino al termine delle iterazioni del for)
      */



      for(k=inizio;k<=fine;++k)
      {
            if(i>medio)
                  aux_arr[k]=vett[j++];
            else
                  if(j>fine)
                        aux_arr[k]=vett[i++];
                  else
                        if(vett[i]<=vett[j])
                              aux_arr[k]=vett[i++];
                        else
                              aux_arr[k]=vett[j++];
      }

      /* copia gli elementi appena ordinati nel vettore d'origine */
      for(k=inizio;k<=fine;++k)
            vett[k]=aux_arr[k];
}
```

```
/* Funzione di Merge Sort (ricorsiva) */

void mergesort(int *array, int begin, int end)
{

      int middle;
      if(begin<end)
      {
            middle=(begin+end)/2;

            /* Richiamo della funzione Merge Sort */
            mergesort(array,begin,middle); /* parte sinistra */
            mergesort(array,middle+1,end); /* parte destra */

            merge(array,begin,middle,end);
       }
}
```

File d'esempio che permette l'uso della libreria "mergesort.c"

```
/* Parte preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "mergesort.c"


void main()
{
      /* Dichiarazione variabili */
      int *a;
      int n,i;

      printf ("Metodo di ordinamento mediante Fusione");

      /* dimensione array */
      printf ("\n\n\nQuanti elementi vuoi inserire? ");
      scanf ("%d",&n);

      /* definizione di array dinamico */
      a=(int*) malloc(sizeof(int)*n);

      printf ("\n\n\n");

      for (i=0;i<n;i++)
      {
            printf("Inserire l'elemento numero %d: ",i+1);
            scanf("%d",&a[i]);
      }


      /* Visualizzazione dei dati nell'Array */
      printf("\n\nArray iniziale = ");
      for (i=0; i<n; i++)
      {
            printf("%d",a[i]);
            printf(" ");
      }


      /* chiamata alla funzione di ordinamento */
      mergesort(a,0,n-1);



      /* Visualizzazione dei dati nell'Array */
      printf("\n\nArray ordinato = ");
      for (i=0; i<n; i++)
      {
            printf("%d",a[i]);
            printf(" ");
      }


      /* Libera la dimensione dell'Array */
      free(a);
}
```

Esempio d'uso



Merge Sort – Esempio d'uso

Quick Sort – Iterativo

(ordinamento mediante il metodo di Hoare)

- **Scopo:** ordinare in senso crescente un insieme di elementi disposti a caso, utilizzando il metodo di partizione di Hoare, noto come quick sort;

- **Specifiche della funzione:**

  File libreria esterna "`qsort-it.c`"

  Funzione:

  ```
  qsort_it(a, n);
  ```

  dove:

  a è la grandezza dell'array di dimensione dinamica

  n è il numero dei valori contenuti nell'array

- **Breve descrizione dell'algoritmo:** la funzione effettua una serie di partizioni dell'insieme da ordinare, e ad ogni passaggio, se la partizione minore contiene più di un elemento viene elaborata e ordinata, in modo da ottenere alla fine un insieme completamante ordinato;

- **Complessità computazionale:** l'efficienza di questo algoritmo consiste nel ridurre il numero di operazioni sui dati. L'operazione dominante è il confronto. In media, per un array di n elementi vengono effettuati $\mathbf{nlog_2}$ n confronti; nel caso peggiore i confronti sono invece $\mathbf{O(n^2)}$;

- **Raccomandazioni d'uso**questo programma è altamente consigliato per l'ordinamento interno di insiemi con ampio numero di dati. Esso infatti ha il vantaggio di spostare gli elementi solo quando è strettamente necessario, riducendo in tal modo gli sforzi e aumentando la propria eleganza.

File libreria "qsort-it.c"

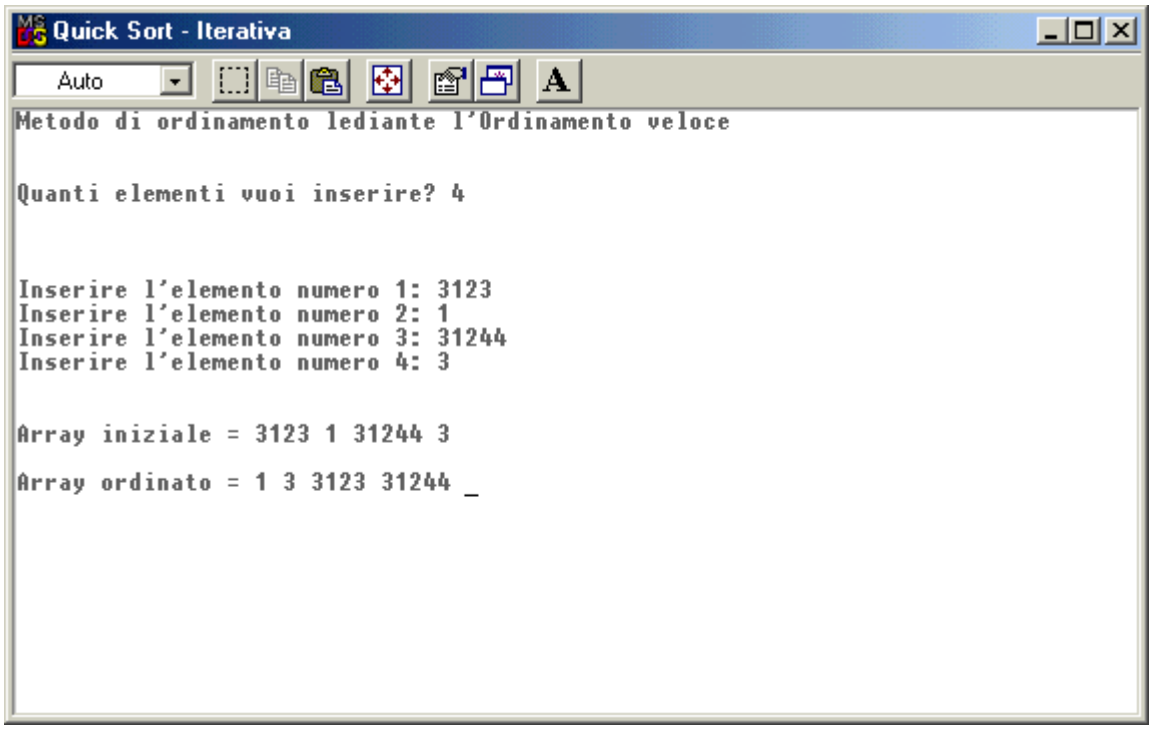
```
void qsort_it(int *a, int n)
{
   int inizio, fine, leftupper, rightlower, medio, medium, temp, head_pila;
   int *stack;

   stack=(int *)calloc(n, sizeof(int));
   head_pila=1;
   stack[0]=0;
   stack[1]=n-1;

   while (head_pila > 0)
   {
      fine=stack[head_pila];
      inizio=stack[head_pila-1];
      head_pila=head_pila-2;

      while (inizio < fine)
      {
            leftupper=inizio;
            rightlower=fine;
            medio=(inizio+fine)/2;
            medium=a[medio];

            while (a[leftupper] < medium)
                  leftupper=leftupper+1;

            while (medium < a[rightlower])
                  rightlower=rightlower-1;

            while (leftupper < rightlower-1)
            {
                  temp=a[leftupper];

                  a[leftupper]=a[rightlower];

                  a[rightlower]=temp;

                  leftupper++;

                  rightlower--;

                  while (a[leftupper] < medium)
                        leftupper=leftupper+1;

                  while (medium < a[rightlower])
                        rightlower=rightlower-1;
            }

            if (leftupper <= rightlower)
            {
                  if(leftupper < rightlower)
                  {
                        temp=a[leftupper];
                        a[leftupper]=a[rightlower];
                        a[rightlower]=temp;
                  }

                  leftupper++;
                  rightlower--;
            }
```

```
            if (rightlower < medio)
            {
                  stack[head_pila+1]=leftupper;
                  head_pila=head_pila+2;
                  stack[head_pila]=fine;
                  fine=rightlower;
            }
            else
            {
                  stack[head_pila+1]=inizio;
                  head_pila=head_pila+2;
                  stack[head_pila]=rightlower;
                  inizio=leftupper;
            }
      }
   }

}
```

File d'esempio che permette l'uso della libreria "qsort-it.c"

```
/* Parte Preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "qsort-it.c"

void main()
{
   /* Dichiarazione variabili */
   int *a;
   int n,i;

   printf ("Metodo di ordinamento lediante l'Ordinamento veloce");

   /* dimensione array */
   printf ("\n\n\nQuanti elementi vuoi inserire? ");
   scanf ("%d",&n);

   /* definizione di array dinamico */
   a=(int*) malloc(sizeof(int)*n);

   printf ("\n\n\n");

   for (i=0;i<n;i++)
   {
            printf("Inserire l'elemento numero %d: ",i+1);
            scanf("%d",&a[i]);
   }


   /* Visualizzazione dei dati nell'Array */
   printf("\n\nArray iniziale = ");
   for (i=0; i<n; i++)
   {
      printf("%d",a[i]);
      printf(" ");
   }

   /* Chiamata della funzione Quick Sort Iterativa*/
   qsort it(a, n);

   /* Visualizzazione dei dati nell'Array */
   printf("\n\nArray ordinato = ");
   for (i=0; i<n; i++)
   {
      printf("%d",a[i]);
      printf(" ");
   }

   /* Libera la dimensione dell'Array */
   free(a);
}
```

Esempio d'uso



Quick Sort - Iterativa – Esempio d'uso

Quick Sort – Ricorsivo

(ordinamento mediante il metodo di Hoare)

- **Scopo:** progettare ed implementare una versione ricorsiva dell'algoritmo di ordinamento rapido;

- **Specifiche della funzione:**

  File libreria esterna "`qsort-ric.c`"

  Funzione:

  ```
  qsort_ric(a,0,n-1);
  ```

  dove:

  a è la grandezza dell'array di dimesione dinamica

  0 è la prima posizione dell'array

  n-1 è il numero dei valori contenuti nell'array

- **Breve descrizione dell'algoritmo:** utilizzando un approccio di tipo ricorsivo, la funzione, dopo aver verificato che l'estremo inferiore è minore di quello superiore, procede con la suddivisione dell'insieme in due partizioni: sinistra e destra. Fatto ciò lavora su entrambe le stesse, effettuando delle chiamate ricorsive alla funzione;

- **Complessità computazionale:** Nel caso peggiore, cioè quando la dimensione dell'insieme è molto grande, la profondità della ricorsione può arrivare fino ad **n-1**;

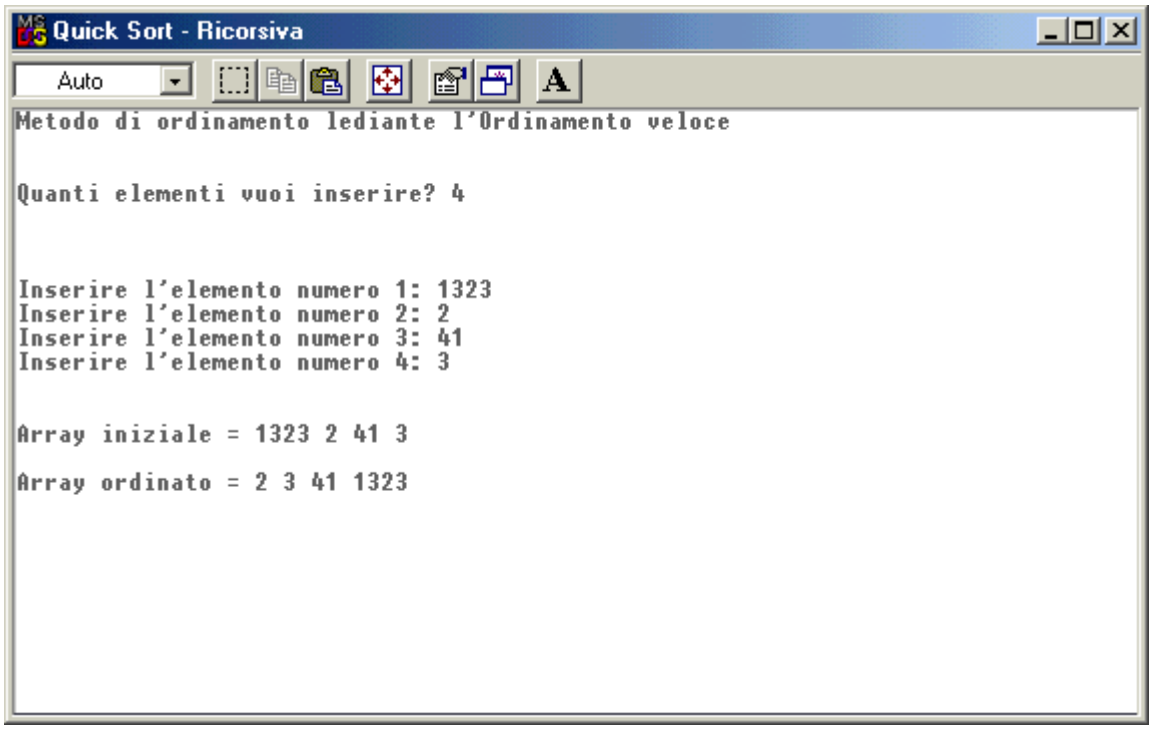
File libreria "qsort-ric.c"

```
/* Carica la libreria utente */
#include "scambia.c"

void qsort_ric(int *a, int m, int n)
{
   int i,j,media;
   media=(a[m]+a[n])/2;
   i=m; j=n;

   do
   {
         while(a[i]<media)   ++i;

         while(a[j]>media)   --j;

         if(i<j)
         {
            swap(&a[i],&a[j]);
            ++i; --j;
         }
    } while(i<j);

    /* Richiamo della funzione qsort-ric */

    if(m<j) qsort_ric(a,m,j);

    if(n>i) qsort_ric(a,i,n);
}
```

File d'esempio che permette l'uso della libreria "qsort-ric.c"

```
/* Parte Preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "qsort-ric.c"

void main()
{
      /* Dichiarazione variabili */
      int *a;
      int n,i;

      printf ("Metodo di ordinamento mediante l'Ordinamento veloce");

      /* dimensione array */
      printf ("\n\n\nQuanti elementi vuoi inserire? ");
      scanf ("%d",&n);

      /* definizione di array dinamico */
      a=(int*) malloc(sizeof(int)*n);

      printf ("\n\n\n");

      for (i=0;i<n;i++)
      {
            printf("Inserire l'elemento numero %d: ",i+1);
            scanf("%d",&a[i]);
      }


      /* Visualizzazione dei dati nell'Array */
      printf("\n\nArray iniziale = ");
      for (i=0; i<n; i++)
      {
            printf("%d",a[i]);
            printf(" ");
      }

      /* chiamata della funzione Quick Sort Ricorsiva*/
      qsort ric(a,0,n-1);

      /* Visualizzazione dei dati nell'Array */
      printf("\n\nArray ordinato = ");
      for (i=0; i<n; i++)
      {
            printf("%d",a[i]);
            printf(" ");
      }

      /* Libera la dimensione dell'Array */
      free(a);
}
```

Esempio d'uso



Quick Sort - Ricorsiva – Esempio d'uso

Matrici
ed operazioni su matrici

Gaxpy

- **Scopo:** il programma esegue un operazione dell'algebra lineare numerica, denominata gaxpy, che somma il prodotto tra una matrice ed un vettore ad un altro vettore

- **Specifiche della funzione:**

  Funzione:

  ```
  int prod_scal(int x[], int y[], int n)
  ```

  dove:

  x[] array del vettore

  y[]array del vettore

  n numero di elementi del vettore risultante, che corrisponde anche al numero di elementi del vettore da addizionare

- **Breve descrizione dell'algoritmo:** la funzione esegue direttamente la somma tra il vettore risultante del prodotto mat x vet (ricevuto in input dal programma chiamante) ed un altro vettore, restituendo poi il risultato al programma chiamante;

- **Complessità computazionale:** il programma prevede l'esecuzione di moltiplicazioni ed addizioni

Funzione "**int prod_scal(int x[], int y[], int n)**"

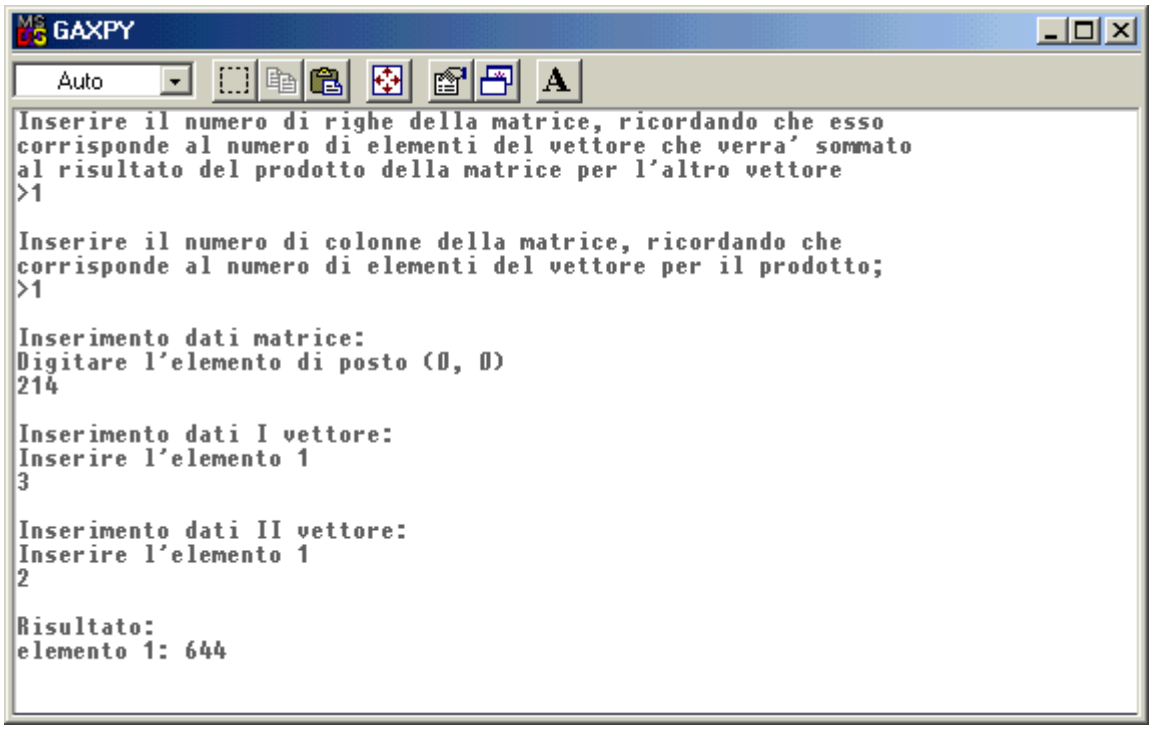
```
/* Routine ausiliare per il calcolo del prodotto scalare; */
int prod_scal(int x[], int y[], int n)
{

      int scal, i;

      scal=0;

      for(i=0; i<n; i++)
            scal=scal+x[i]*y[i];

      return(scal);
}
```

File “gaxpy.c”

```
/* Parte preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Programma che esegue un'operazione del calcolo matriciale, detta gaxpy, */
/* del tipo 'b=b+Aa' con 'a' e 'b' vettori ed 'A' matrice di dimensioni    */
/* stabilite dall'utente.                                                  */


/* Routine ausiliare per il calcolo del prodotto scalare; */

int prod_scal(int x[], int y[], int n)
{

      int scal, i;

      scal=0;

      for(i=0; i<n; i++)
            scal=scal+x[i]*y[i];

      return(scal);
}



main()

{

      /* Dichiarazione delle variabili e della function per il */
      /* calcolo del prodotto scalare fra due vettori */

      int n_rig, n_col, i, j;
      int *Mat, *vet_a, *vet_b, *vet_c;
      int prod_scal(int x[], int y[], int n);

      /* Inserimento dimensioni ed allocazione dinamica di memoria; */

      printf("\nInserire il numero di righe della matrice, ricordando che
esso\n");

      printf("corrisponde al numero di elementi del vettore che verra'
sommato\n");

      printf("al risultato del prodotto della matrice per l'altro vettore\n");
      scanf("%d", &n_rig);

      printf("\nInserire il numero di colonne della matrice, ricordando che\n");

      printf("corrisponde al numero di elementi del vettore per il
prodotto;\n");

      scanf("%d", &n_col);

      Mat=(int *)calloc((n_rig*n_col), sizeof(int));
      vet_a=(int *)calloc(n_col, sizeof(int));
      vet_c=(int *)calloc(n_col, sizeof(int));
      vet_b=(int *)calloc(n_rig, sizeof(int));
```

```
	/* Inserimento dati; */

	printf("\nInserimento dati matrice:\n");
	for(i=0; i<n_rig; i++)
	{
		for(j=0; j<n_col; j++)
		{
		  printf("Digitare l'elemento di posto (%d, %d)\n", i, j);
		  scanf("%d", Mat+(n_col*i+j));
		}
	}

	printf("\nInserimento dati I vettore:\n");
	for(i=0; i<n_col; i++)
	{
		printf("Inserire l'elemento %d\n", i+1);
		scanf("%d", vet_a+i);
	}

	printf("\nInserimento dati II vettore:\n");
	for(i=0; i<n_rig; i++)
	{
		printf("Inserire l'elemento %d\n", i+1);
		scanf("%d", vet_b+i);
	}

	for(i=0; i<n_rig; i++)
	{
		for(j=0; j<n_col; j++)
			*(vet_c+j) = *(Mat+n_col*i+j);

		*(vet_b+i) = (*(vet_b+i) + (prod_scal(vet_a, vet_c, n_col)));
	}

	printf("\nRisultato:\n");

	for(i=0; i<n_rig; i++)
	{
		printf("elemento %d: %d\n", i+1, *(vet_b+i));
	}

}
```

Esempio d'uso



Gaxpy – Esempio d'uso

Saxpy

- **Scopo:** il programma esegue un operazione dell'algebra lineare numerica, denominata saxpy, che somma il prodotto tra uno scalare ed un vettore ad un altro vettore

- **Specifiche della funzione:**

  Funzione:

  ```
  void SAXPY (int i, int n, int b, int x[], int y[])
  ```

  dove:

  i è l'indice

  n è il numero di elementi dei due vettori

  b è lo scalare considerato

  x[ ] è il vettore per il prodotto

  y[ ] è il vettore per il somma

- **Breve descrizione dell'algoritmo:** utilizzando un ciclo for, la funzione esegue il prodotto tra lo scalare ed il primo vettore, e poi somma il risultato al secondo vettore;

- **Complessità computazionale:** il programma prevede l'esecuzione di n moltiplicazioni ed n addizioni;

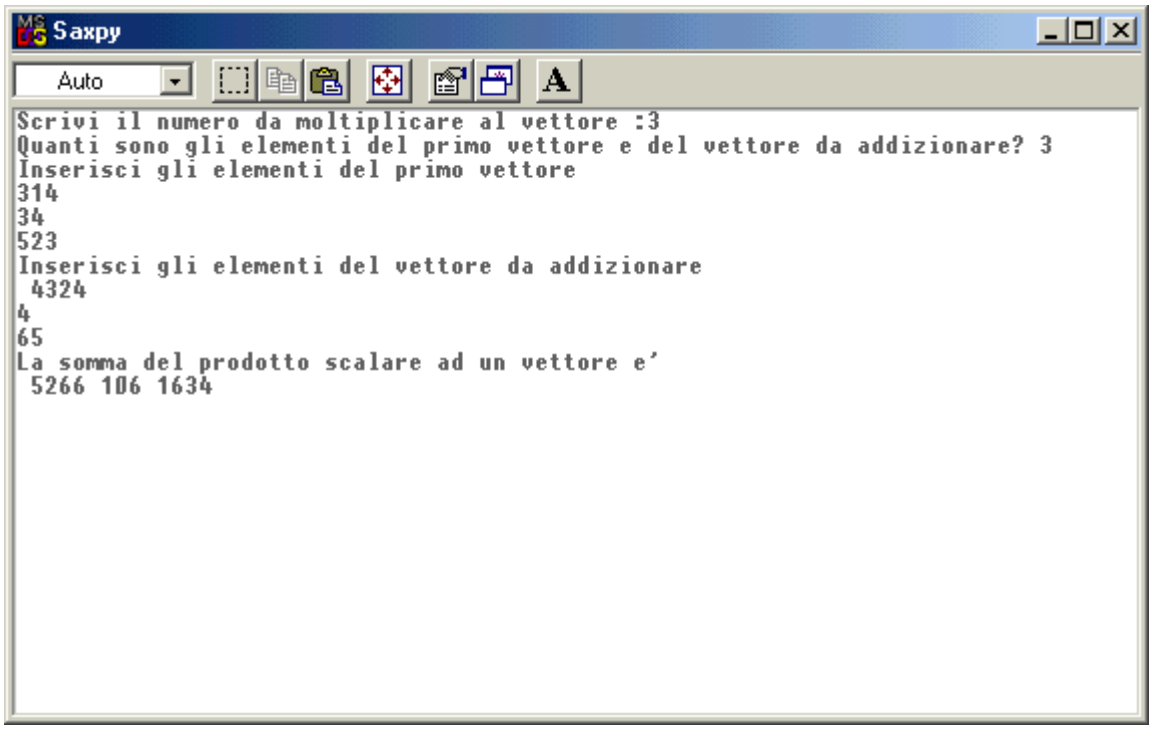
File d'esempio che permette l'uso della libreria "saxpy.c"

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

/* Funzione di calcolo del SAXPY */
void SAXPY (int i, int n, int b, int x[], int y[])
{

      /*corpo della funzione */
      for(i=0;i<n;i++)
            y[i]=(x[i]*b)+y[i];

      return;
}



main( )
{
      /*dichiarazione di variabili */
      int i, n, b, *x, *y;

      /*inserimento dati */
      printf("Scrivi il numero da moltiplicare al vettore :");
      scanf("%d", &b);

      printf("Quanti sono gli elementi del primo vettore e del vettore da
addizionare? ");
      scanf("%d", &n);

      /*allocazione di memoria */
      x = malloc(n* sizeof(int));
      y = malloc(n* sizeof(int));

      printf("Inserisci gli elementi del primo vettore \n");
      for(i=0;i<n;++i)
            scanf("%d", &x[i]);

      printf("Inserisci gli elementi del vettore da addizionare \n ");
      for(i=0;i<n;++i)
            scanf("%d", &y[i]);


      /*chiamata della funzione SAXPY */
      SAXPY (i, n, b, x, y);


      /*stampa risultato */
      printf("La somma del prodotto scalare ad un vettore e' \n ");
      for(i=0;i<n;++i)
            printf("%d ",y[i]);

      /*Libera lo spazio delle variabili allocate dinamicamente*/
      free(x);
      free(y);

      return;
}
```

File libreria "saxpy.c"

```
void SAXPY (int i, int n, int b, int x[], int y[])
{

      /*corpo della funzione */
      for(i=0;i<n;i++)
            y[i]=(x[i]*b)+y[i];

      return;
}
```

Esempio d’uso



Saxpy – Esempio d’uso

Prodotto Matrice – Matrice

- **Scopo:** il presente programma si propone di effettuare il calcolo del prodotto tra due matrici;

- **Specifiche della funzione:**

  Funzione:

```
void mat mat(int a[], int b[], int rig 1, int col 1, int col 2)
```

  dove:

  a[] contiene i dati della prima matrice

  b[] contiene i dati della seconda matrice

  rig_1: indica il numero di righe della prima matrice

  col_1: indica il numero di righe della prima matrice, che equivale anche al numero di righe della seconda matrice.

  col_2: intero. indica il numero di colonne della seconda matrice.

- **Breve descrizione dell'algoritmo:** la funzione, dopo aver allocato dinamicamente lo spazio necessario per memorizzare la matrice risultato, utilizza tre cicli for innestati per effettuare il calcolo del prodotto matriciale ed altri due cicli for per posizionare gli elementi nella matrice risultato.

- **Complessità computazionale:** il programma prevede l'esecuzione di moltiplicazioni e addizioni in numero pari a **rig_1*col_1*col_2.**

File del programma "`matrix.c`"

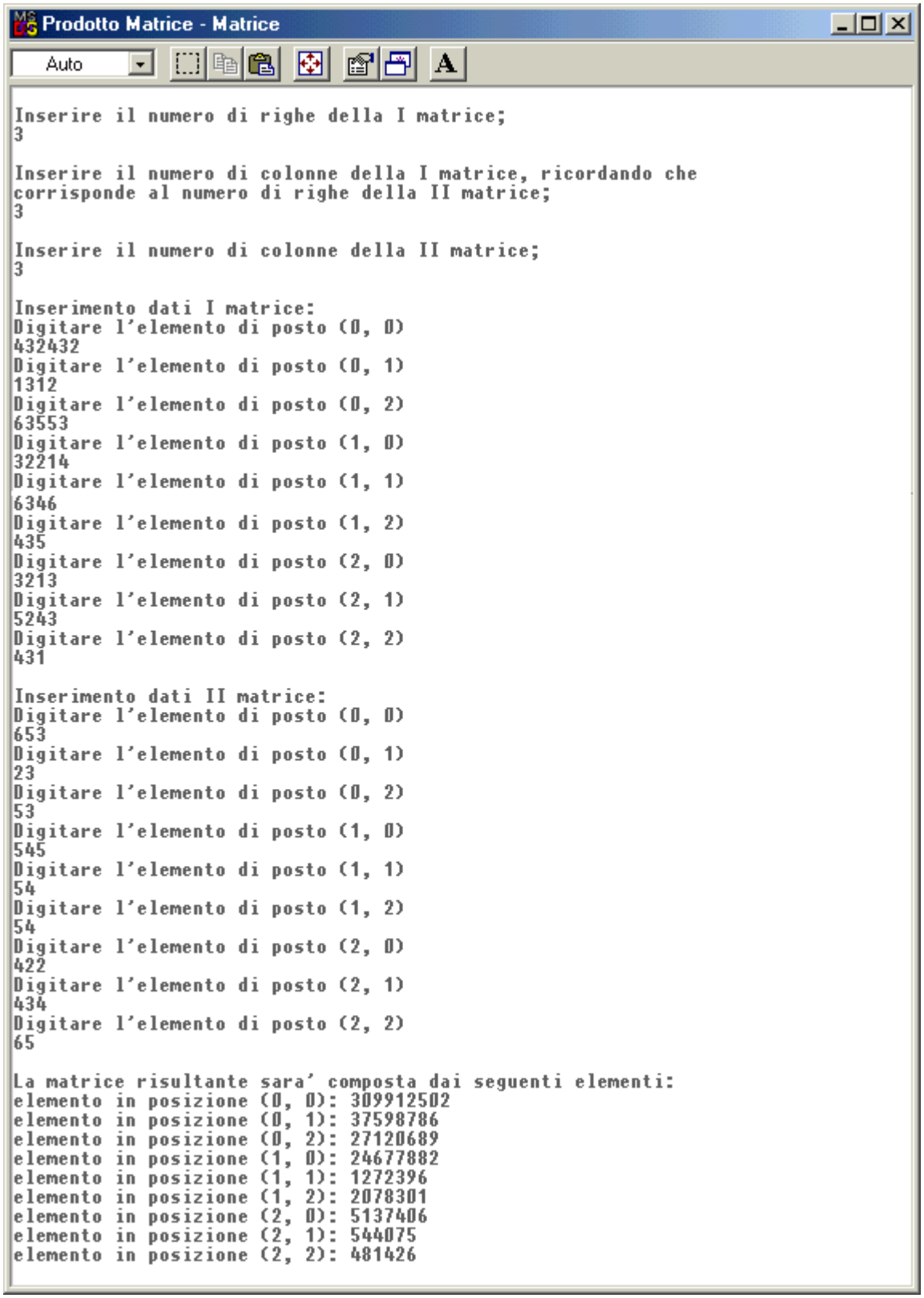
```
/* Parte preprocessore */
#include <stdio.h>
#include <stdlib.h>


/* Funzione per il calcolo del prodotto Matriciale */

void mat_mat(int a[], int b[], int rig_1, int col_1, int col_2)

{

int i, j, k;
int *matrix;

/* Allocazione dinamica di memoria della matrice risultante */

matrix=(int *)calloc((rig_1*col_2), sizeof(int));

for(i=0; i<rig_1; i++)
{
 for(j=0; j<col_2; j++)
 {
  *(matrix+col_2*i+j)=0;
  for(k=0; k<col_1; k++)
  {
   *(matrix+col_2*i+j) = *(matrix+col_2*i+j) + a[col_1*i+k] * b[col_2*k+j];
  }
 }
}

for(i=0; i<rig_1; i++)
{
 for(j=0; j<col_2; j++)
 {
  printf("elemento in posizione (%d, %d): %d\n", i, j, *(matrix+col_2*i+j));
 }
}

}


/* Programma per il calcolo del prodotto fra due matrici. */

main()

{

/* Dichiarazione delle variabili e della routine ausiliare; */

int n_rig, n_col_1, n_col_2, i, j;
int *p, *q;
void mat_mat(int a[], int b[], int rig_1, int col_1, int col_2);

/* L'utente potra' stabilire, in modo incondizionato, le dimensioni delle due */
/* matrici, in quanto la memoria richiesta verrà allocata dinamicamente;      */

printf("\nInserire il numero di righe della I matrice;\n");
scanf("%d", &n_rig);
printf("\nInserire il numero di colonne della I matrice, ricordando che\n");
```

```
printf("corrisponde al numero di righe della II matrice;\n");
scanf("%d", &n_col_1);
printf("\nInserire il numero di colonne della II matrice;\n");
scanf("%d", &n_col_2);
p=(int *)calloc((n_rig*n_col_1), sizeof(int));
q=(int *)calloc((n_col_1*n_col_2), sizeof(int));

/* Si ricorda che in "C", la rappresentazione interna di un array 2D è per    */
/* righe, quindi è possibile determinare l'indirizzo di un generico elemento  */
/* (i,j), mediante l'indirizzo della componente (0,0) sommato al numero di    */
/* colonne dell'array moltiplicato per "i", + "j";                            */

printf("\nInserimento dati I matrice:\n");
for(i=0; i<n_rig; i++)
{
 for(j=0; j<n_col_1; j++)
 {
  printf("Digitare l'elemento di posto (%d, %d)\n", i, j);
  scanf("%d", p+(n_col_1*i+j));
 }
}

printf("\nInserimento dati II matrice:\n");
for(i=0; i<n_col_1; i++)
{
 for(j=0; j<n_col_2; j++)
 {
  printf("Digitare l'elemento di posto (%d, %d)\n", i, j);
  scanf("%d", q+(n_col_2*i+j));
 }
}

printf("\nLa matrice risultante sara' composta dai seguenti elementi:\n");
mat_mat(p, q, n_rig, n_col_1, n_col_2);

}
```

Function “void mat_mat(int a[], int b[], int rig_1, int col_1, int col_2)”

```
/* Function per il calcolo del prodotto */

void mat_mat(int a[], int b[], int rig_1, int col_1, int col_2)

{

int i, j, k;
int *matrix;

/* Allocazione dinamica di memoria della matrice risultante */

matrix=(int *)calloc((rig_1*col_2), sizeof(int));

for(i=0; i<rig_1; i++)
{
 for(j=0; j<col_2; j++)
 {
  *(matrix+col_2*i+j)=0;
  for(k=0; k<col_1; k++)
  {
   *(matrix+col_2*i+j) = *(matrix+col_2*i+j) + a[col_1*i+k] * b[col_2*k+j];
  }
 }
}

for(i=0; i<rig_1; i++)
{
 for(j=0; j<col_2; j++)
 {
  printf("elemento in posizione (%d, %d): %d\n", i, j, *(matrix+col_2*i+j));
 }
}

}
```

Esempio d'uso



Prodotto Matrice - Matrice – Esempio d'uso

Matrice sparsa

- **Scopo:** il programma effettua la memorizzazione di una matrice sparsa utilizzando il metodo de tre vettori;

- **Specifiche della funzione:**

  Funzione:

```
int MatSpar(int Dim, int Dim2, int *PMat, int *PV1, int *PV2, int *PV3)
```

  dove:

  Dim1, Dim2 indicano le dimensione delle matrici

  ***PMat** è il puntatore alla matrice

  ***PV1** è il puntatore al 1° vettore contenete i valori non nulli della matrice

  ***PV2** è il puntatore al vettore contenete l'indice di colonna degli elementi non nulli della matrice

  ***PV3** è il puntatore al vettore contenente le posizioni nei primi due vettori del primo elemento non nullo di ciascuna riga della matrice

- **Breve descrizione dell'algoritmo:** utilizzando una combinazione tra due cicli for e due costrutti if, la funzione scompone la matrice data in tre vettori, restituendo poi al programma chiamante il valore del terzo vettore, in cui è specificata la posizione esatta del primo elemento non nullo di ciascuna riga;

- **Complessità computazionale:** le operazioni eseguite dal programma sono solo addizioni;

Funzione "`MatSpar(int Dim, int Dim2, int *PMat, int *PV1, int *PV2, int *PV3)`"

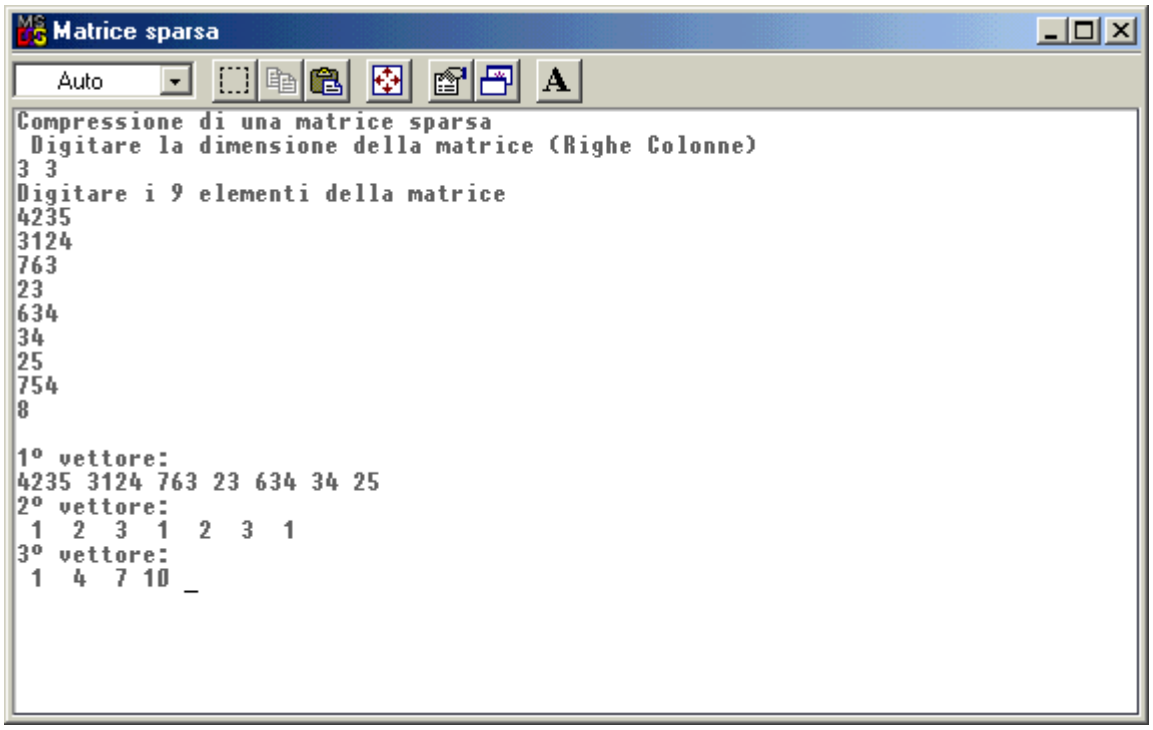
```
/* Funzione di una Matrice Sparsa */
int MatSpar(int Dim, int Dim2, int *PMat, int *PV1, int *PV2, int *PV3)
{
      int Ind, Ind2, Trovato, K, J;

/*
   Alla variabile "Trovato", visto che le variabili logiche non
   esistono, facciamo assumere valore 1 al posto di Vero, e 0 per Falso;
   "Ind" e "Ind2" sono due contatori che ci servono per riempire i 3 array:
   "Ind" viene utilizzato per i primi 2 e ci restituire il numero di elementi
   non nulli della matrice, "Ind2" viene utilizzato per il terzo, la cui
   dimensione (generalmente minore di quella degli altri 2) deve essere
   restituito come valore della funzione.
*/

      Ind = Ind2 = Trovato = 0;

      for (K = 0; K < Dim; ++K)
      {
          for (J = 0; J < Dim2; ++J)
          {
            if (*(PMat + K * 10 + J) != 0)
            {
                  *(PV1 + Ind) = *(PMat + K * 10 + J);
                  *(PV2 + Ind) = J + 1;

                  if (Trovato == 0)
                  {
                        *(PV3 + Ind2) = Ind + 1;
                        Trovato = 1;
                        ++Ind2;
                  }

                  ++Ind;
            }
          }

          Trovato = 0;
      }

      *(PV3 + Ind2) = Ind + 1;

      return Ind2;
}
```

File “Matrix-sprs.c”

```
/* Parte preprocessore */
#include <stdio.h>

/* Funzione di una Matrice Sparsa */
int MatSpar(int Dim, int Dim2, int *PMat, int *PV1, int *PV2, int *PV3)
{
      int Ind, Ind2, Trovato, K, J;

/*
   Alla variabile "Trovato", visto che le variabili logiche non
   esistono, facciamo assumere valore 1 al posto di Vero, e 0 per Falso;
   "Ind" e "Ind2" sono due contatori che ci servono per riempire i 3 array:
   "Ind" viene utilizzato per i primi 2 e ci restituire il numero di elementi
   non nulli della matrice, "Ind2" viene utilizzato per il terzo, la cui
   dimensione (generalmente minore di quella degli altri 2) deve essere
   restituito come valore della funzione.
*/

      Ind = Ind2 = Trovato = 0;

      for (K = 0; K < Dim; ++K)
      {
          for (J = 0; J < Dim2; ++J)
          {
            if (*(PMat + K * 10 + J) != 0)
            {
                  *(PV1 + Ind) = *(PMat + K * 10 + J);
                  *(PV2 + Ind) = J + 1;

                  if (Trovato == 0)
                  {
                        *(PV3 + Ind2) = Ind + 1;
                        Trovato = 1;
                        ++Ind2;
                  }

                  ++Ind;
            }
          }

          Trovato = 0;
      }

      *(PV3 + Ind2) = Ind + 1;

      return Ind2;
}

/* Funzione Main */
main()
{
      int Dim, Dim2, K, J, Num;
      int M[10][10], V1[15], V2[15], V3[16];

      printf("Compressione di una matrice sparsa\n Digitare la dimensione della
matrice (Righe Colonne)\n");

      scanf("%d %d", &Dim, &Dim2);

      printf("Digitare i %d elementi della matrice\n", (Dim * Dim2));
```

```
        for (K = 0; K < Dim; ++K)
              for (J = 0; J < Dim2; ++J)
                    scanf("%d", &M[K][J]);

        /* Carica la funzione MatSpar */
        Num = MatSpar(Dim, Dim2, &M[0][0], &V1[0], &V2[0], &V3[0]);

        if (Num == 0)
              printf("La matrice nulla!\n");
        else
        {
              printf("\n1ø vettore:\n");
              for (K = 0; K < V3[Num -1]; ++K)
                    printf("%2d ", V1[K]);

              printf("\n2ø vettore:\n");
              for (K = 0; K < V3[Num -1]; ++K)
                    printf("%2d ", V2[K]);

              printf("\n3ø vettore:\n");
              for (K = 0; K <= Num; ++K)
                    printf("%2d ", V3[K]);
        }
}
```

Esempio d'uso



Matrice Sparsa – Esempio d'uso

Prodotto Matrice Vettore

- **Scopo:** il programma esegue il prodotto tra una matrice ed un vettore;

- **Specifiche della funzione:**

  File libreria esterna "**`matxvet.c`**"

  Funzione:

```
void PRODOTTO_MAT_VET (float* A, float* B, int righe, int colonne, float* R)
```

  dove:

  ***A** è il puntatore all'array 1D contenente gli elementi non nulli della matrice;

  ***B** è il puntatore al vettore da moltiplicare;

  righe è il numero di righe della matrice e il numero dicomponenti del vettore risultato;

  colonne è il numero di colonne della matrice e il numero di componenti del vettore da moltiplicare;

  ***R** Puntatore al vettore risultato;

- **Breve descrizione dell'algoritmo:** utilizzando due cicli for innestati, la funzione esegue il prodotto richiesto, tenendo conto che la matrice è memorizzata sotto forma di un array monodimensionale;

- **Complessità computazionale:** il programma prevede l'esecuzione di moltiplicazioni e addizioni in numero pari a **righe * colonne;**

File libreria "matxvet.c"

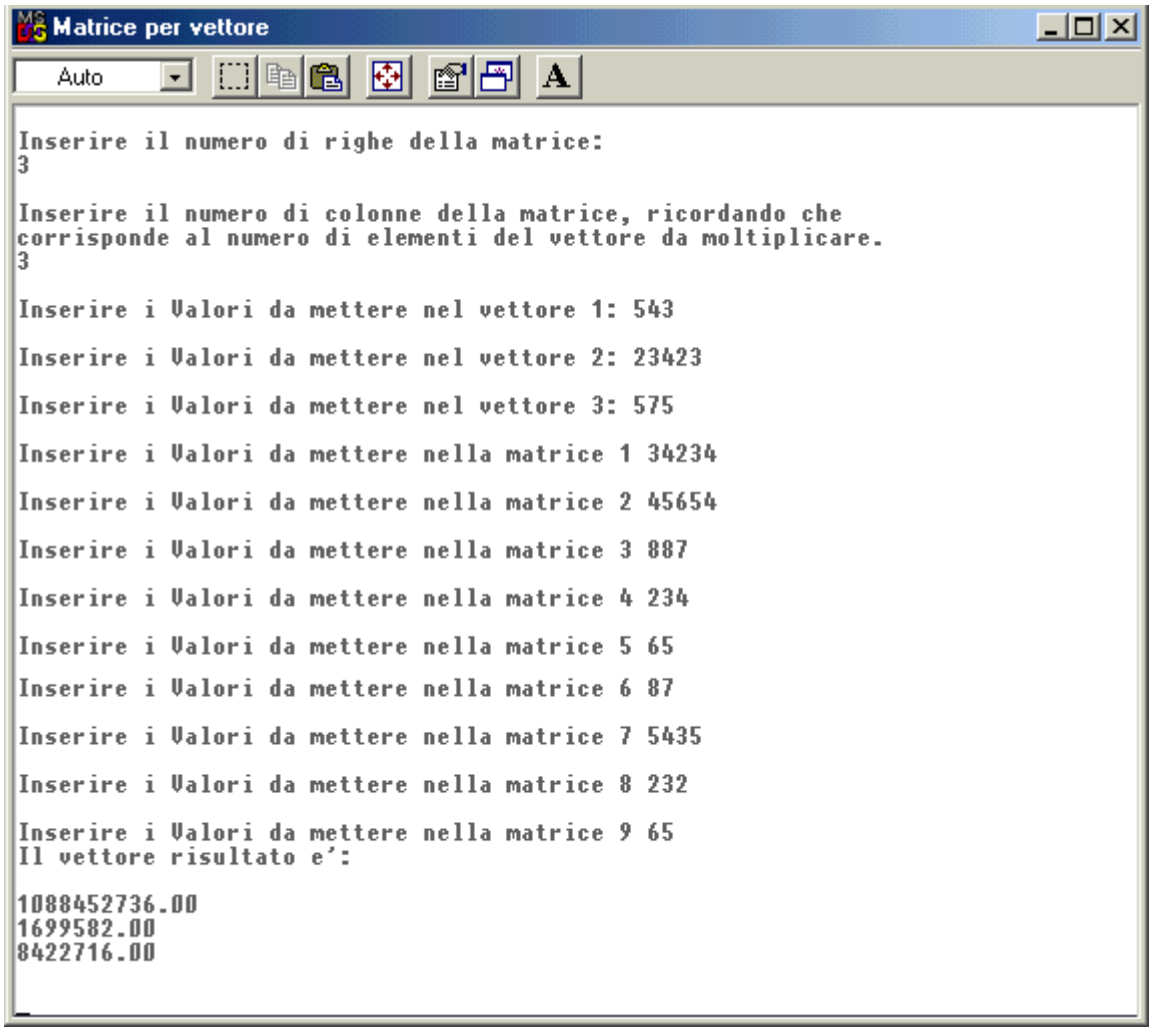
```
void PRODOTTO_MAT_VET (float* A, float* B, int righe, int colonne, float* R)
{
      /*variabili locali */
      int i, j;

      for (i=0; i<righe; i++)
      {
            R[i]= 0;
            for (j=0; j<colonne; j++)
                  *(R+ i) =  *(A+ (i * colonne) +j) * *(B + j)+*(R + i);
      }

}
```

File d'esempio che permette l'uso della libreria "matxvet.c"

```
/* Parte preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "matxvet.c"

/* Funzione principale Main */
main ( )
{
      /*dichiarazione di variabili */
      int i, j, righe, colonne;

      float* matrice; /* puntatore all'array 1D con gli elementi non nulli della
matrice */
      float* vettore;  /*puntatore al vettore da moltiplicare */
      float* risultato; /* puntatore al vettore risultato */

      printf("\nInserire il numero di righe della matrice:\n");
      scanf("%d", &righe);

      printf ("\nInserire il numero di colonne della matrice, ricordando
che\n");
      printf ("corrisponde al numero di elementi del vettore da
moltiplicare.\n");
      scanf ("%d", &colonne);

      /*Allocazione dinamica della matrice  e del vettore da moltiplicare */
      matrice = (float*) calloc ((righe*colonne), sizeof (float));
      vettore = (float*) calloc (colonne, sizeof (float));

      /* inserimento dati*/
      for (j=0; j<colonne; j++)
      {
            printf("\nInserire i Valori da mettere nel vettore %d: ",j+1);
            scanf ("%f", &vettore[j]);
      }


      for (i=0; i<(righe * colonne); i++)
      {
            printf("\nInserire i Valori da mettere nella matrice %d ",i+1);
            scanf ("%f", &matrice[i]);
      }

      /*allocazione dinamica del vettore risultato */
      risultato = (float*) calloc (righe, sizeof (float));

      /*chiamata della funzione PRODOTTO MAT VET */
      PRODOTTO MAT VET (matrice, vettore, righe, colonne, risultato);

      /*stampa risultato */
      printf ("Il vettore risultato e':\n");

      for (i=0; i<righe; i++)
            printf ("\n%.2f", risultato[i]);

      printf("\n\n");
```

```
	/* Libera lo spazio in memoria allocato dinamicamente */
	free (matrice);
	free (vettore);
	free (risultato);
}
```

Esempio d’uso



Prodotto Matrice Vettore – Esempio d’uso

Prodotto Scalare tra due vettori

- **Scopo: :** il programma effettua il prodotto scalare tra due vettori;

- **Specifiche della funzione:**

  File libreria esterna "**`prodscal.c`**"

  Funzione:

```
int prodotto_scalare (int n, int *a, int *b)
```

  dove:

  n è la dimensione dei due vettori;

  ***a** è il puntatore all'array al primo vettore;

  ***b** è il puntatore all'array al secondo vettore;

- **Breve descrizione dell'algoritmo:** utilizzando un ciclo for, la funzione esegue il prodotto tra i due vettori, e poi restituisce il valore ottenuto al programma chiamante;

- **Complessità computazionale:** il programma prevede l'esecuzione di **n** moltiplicazioni ed **n-1** addizioni**;**

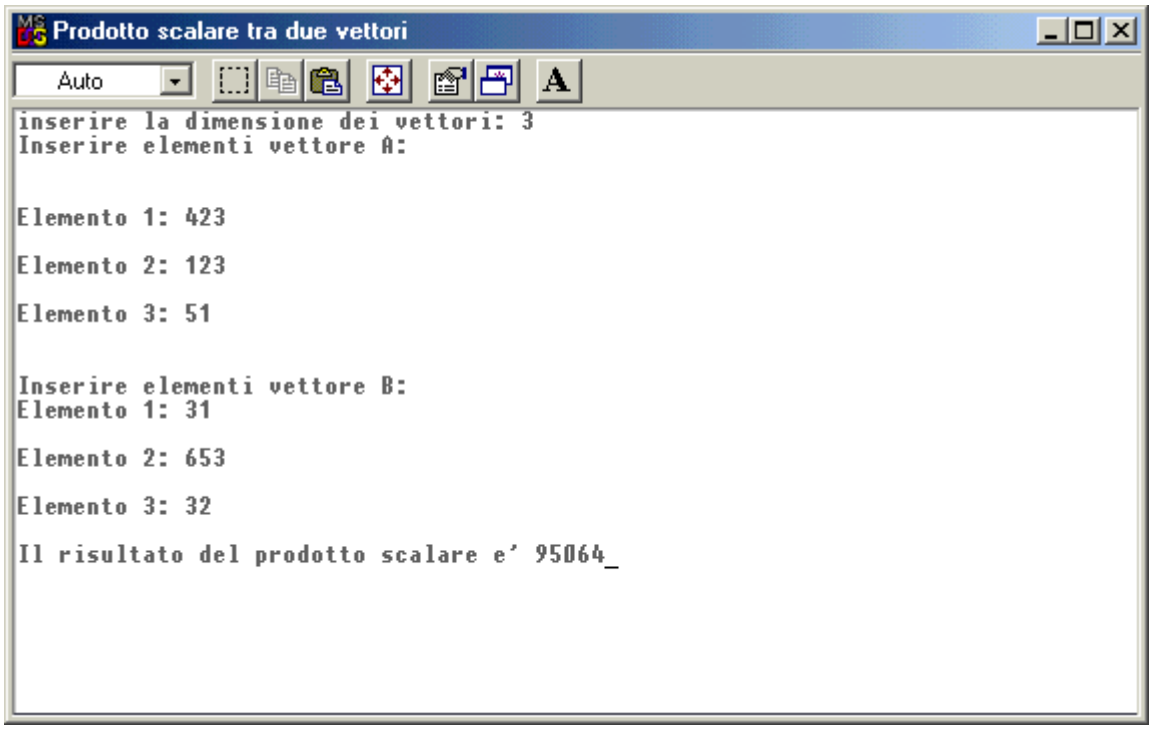
File libreria "`prodscal.c`"

```
int prodotto_scalare (int n, int *a, int *b)
{
      /*variabili locali */
      int i, prod;

      prod=0;

      for (i=0; i<n; i++)
            prod = prod + a[i]*b[i];

      return prod;
}
```

File d'esempio che permette l'uso della libreria "prodscal.c"

```
/* Parte preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "prodscal.c"

main ( )
{
      /*dichiarazione di variabili */
      int i, n, C;
      int *A, *B;

      printf ("inserire la dimensione dei vettori: ");
      scanf ("%d", &n);

      /*Allocazione dinamica dei vettori A e B */
      A=(int*) calloc (n, sizeof(int));
      B=(int*) calloc (n, sizeof(int));

      /*Inserimento elementi */
      printf ("Inserire elementi vettore A:");
      printf("\n\n");
      for(i=0; i<n; i++)
      {
            printf("\nElemento %d: ",i+1);
            scanf ("%d", &A[i]);
      }

      printf ("\n\nInserire elementi vettore B:");

      for(i=0; i<n; i++)
      {
            printf("\nElemento %d: ",i+1);
            scanf ("%d", &B[i]);
      }

      /*chiamata della funzione prodotto scalare */
      C = prodotto_scalare (n, A, B);

      /*stampa risultato */
      printf ("\nIl risultato del prodotto scalare e' %d", C);

}
```

Esempio d'uso



Prodotto Scalare tra due vettori – Esempio d'uso

Algoritmi di ricerca

Ricerca Hash

- **Scopo:** ricercare un elemento in un insieme utilizzando il metodo della ricerca calcolata Hash;

- **Specifiche della funzione:**

  File libreria esterna "`hash.c`"

  Funzione:

  ```
  void hash(array, table, dim, tablesize);
  ```

  dove:

  array è la grandezza dell'array di dimesione dinamica

  table è la prima posizione dell'array

  dim è il numero dei valori contenuti nell'array

  tablesize è puntatore all'array degli elementi

- **Breve descrizione dell'algoritmo:** l'algoritmo dopo aver definito la tabella calcolata da esaminare, la chiave cercata, la dimensione della tabella e il modo per segnalare una casella vuota, l'algoritmo (suddiviso in due funzioni) esegue la ricerca fino a che la chiave viene trovata, oppure finché incontra una casella vuota o si accorge che la tabella è piena ma la chiave non c'è. Tali condizioni vengono rilevate con l'utilizzo di due variabili logiche: la prima (active) segnala quando la ricerca deve terminare; la seconda (found) indica se l'elemento è stato trovato;

- **Raccomandazioni d'uso:** Il programma è consigliato solo per ordinare un piccolo insieme di dati.

File libreria "hash.c"

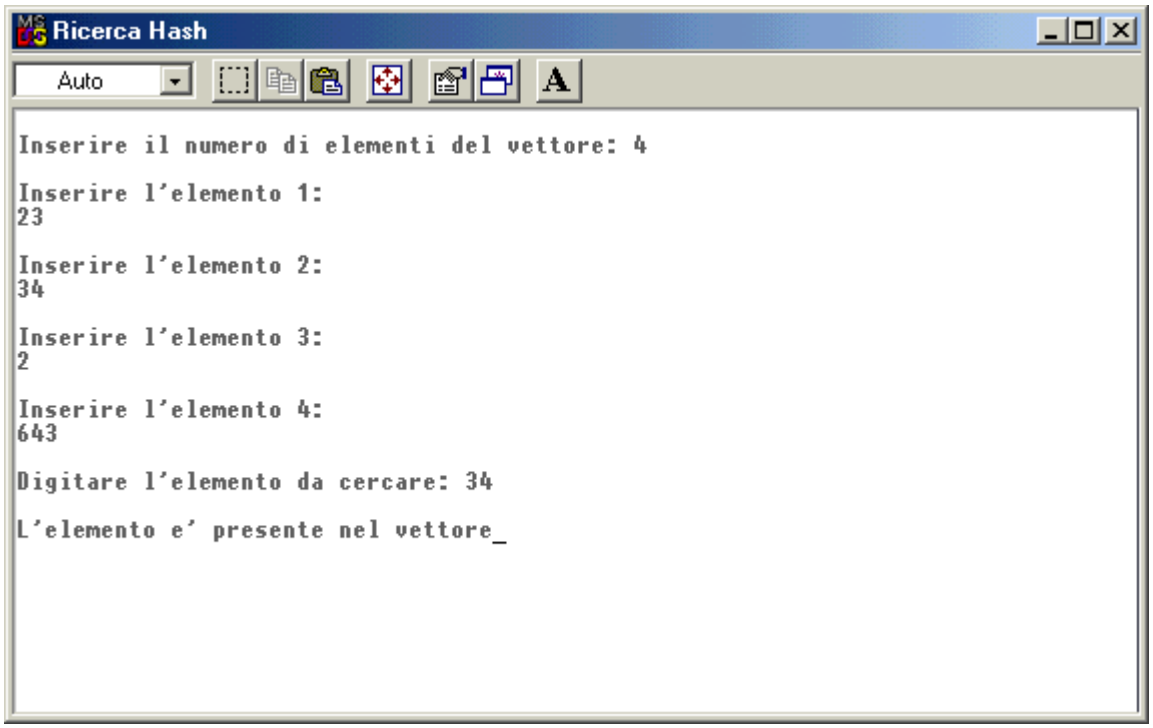
```
/* inclusione della libreria Matematica */
#include <math.h>


/* Definizione deò tipo logical */
typedef enum {falso, vero} logical;


/* Funzione hash */
void hash(int *array, int *table, int dim, int tablesize)
{
 int *vett_app, n, j, k, i;
 vett_app=(int *)calloc(dim, sizeof(int));
 n=0;

 for(i=0; i<dim; i++)
 {
  j=array[i] % tablesize;
  if(table[j]==NULL)
   table[j]=array[i];
  else
  {
    vett_app[n]=array[i];
    n++;
  }
 }

 for(i=0; i<n; i++)
 {
  k=vett_app[i] % tablesize;
  while(table[k] != NULL)
    k++;
  table[k]=vett_app[i];
 }

 /* Libera il vettore dall'allocazione dinamica */
 free(vett_app);
}


/* Funzione local, defita precedentemente */

logical hash_search(int *table, int tablesize, int key)
{
 logical active, found;
 int start, position, temp;

 active=vero;
 found=falso;
 start=key % tablesize;
 position=start;
 if(table[position]==key)
 {
  active=falso;
  found=vero;
  temp=table[position];
 }
 else
 {
  temp=table[position];
```

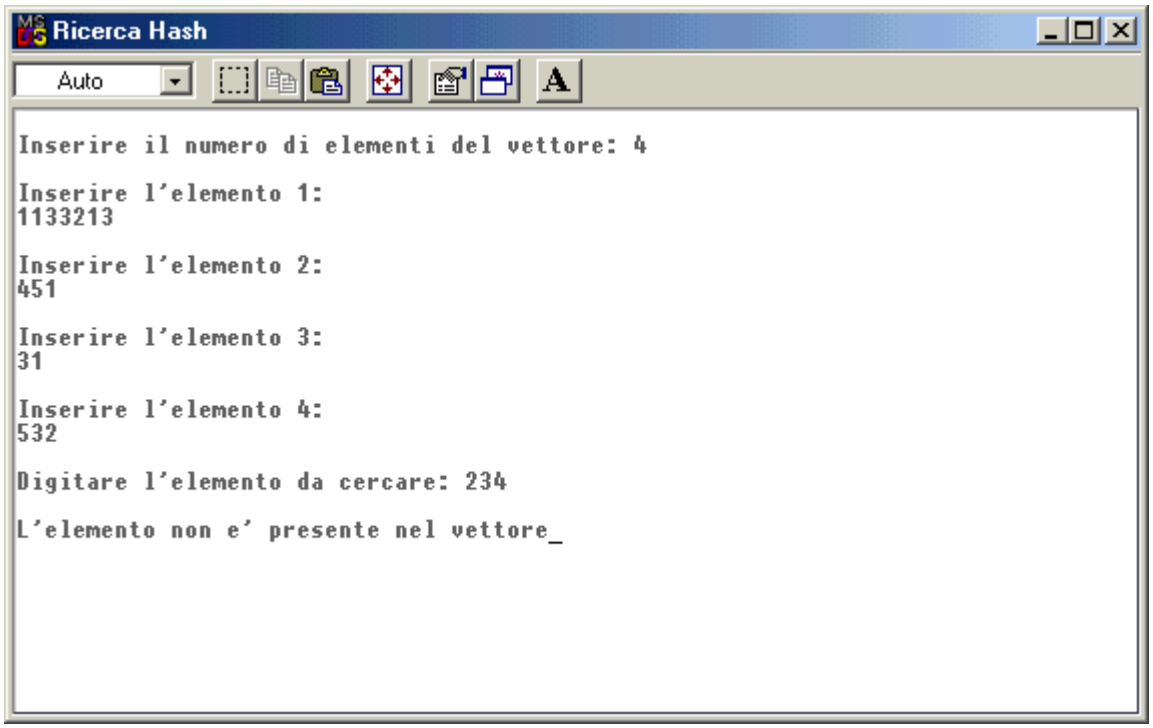
```
  table[position]=key;
 }

 while(active)
 {
  position=(position+1) % tablesize;
  if(table[position]==key)
  {
    active=falso;
    if(position != start)
      found=vero;
  }
  else
   if(table[position]==NULL)
    active=falso;

  table[start]=temp;
 }
 return found;
}
```

File d'esempio che permette l'uso della libreria "hash.c"

```
/* Parte preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "hash.c"

main()
{
 int *array, *table, i, tablesize, dim, key;

 printf("Metodo di ricerca Hash\n\n\n");

 printf("\nInserire il numero di elementi del vettore: ");
 scanf("%d", &dim);

 /* Allocazione dinamica dell'Array */
 array=(int *)calloc(dim, sizeof(int));

 for(i=0; i<dim; i++)
 {
  printf("\nInserire l'elemento %d: \n", i+1);
  scanf("%d", &array[i]);
 }

 tablesize=dim+(dim*25/100)+1;

 /*allocazione dinamica della tabella */
 table=(int *)calloc(tablesize, sizeof(int));

 for(i=0; i<tablesize; i++)
  table[i]=NULL;

 /* Carica la funzione di ricerca Hash */
 hash(array, table, dim, tablesize);

 printf("\nDigitare l'elemento da cercare: ");
 scanf("%d", &key);

 /* Verifica se l'elemento è nell'Array */
 if(hash_search(table, tablesize, key))
   printf("\nL'elemento e' presente nel vettore");
 else
   printf("\nL'elemento non e' presente nel vettore");
}
```

Esempio d'uso





Ricerca Hash – Esempi d'uso

Ricerca Binaria

- **Scopo:** dato un elemento x ed un insieme di dati ordinati in senso crescente, stabilire se x appartiene all'insieme;

- **Specifiche della funzione:**

  File libreria esterna "`ric-bin.c`"

  Funzione:

  ```
  logical RICERCA_BINARIA (int *vett, int n, int k)
  ```

  dove:

  ***vett** è la variabile puntatore all'array ordinato

  n è la dimensione effettiva dell'array ordinato

  k è l'elementoda cercare

- **Breve descrizione dell'algoritmo:** dopo aver definito l'array ordinato e l'elemento da cercare, i due estremi assumono il ruolo di primo ed ultimo. Successivamente, finché il primo è maggiore dell'ultimo, si calcola l'elemento medio e lo si confronta con x, fino a che quest'ultimo non viene trovato;

- **Complessità computazionale:** l'operazione dominante eseguita dalla function è il confronto. Nel caso migliore (x trovato immediatamente), viene eseguito un solo confronto. Nel caso peggiore la complessità computazionale equivale a $\log_2 (n)$

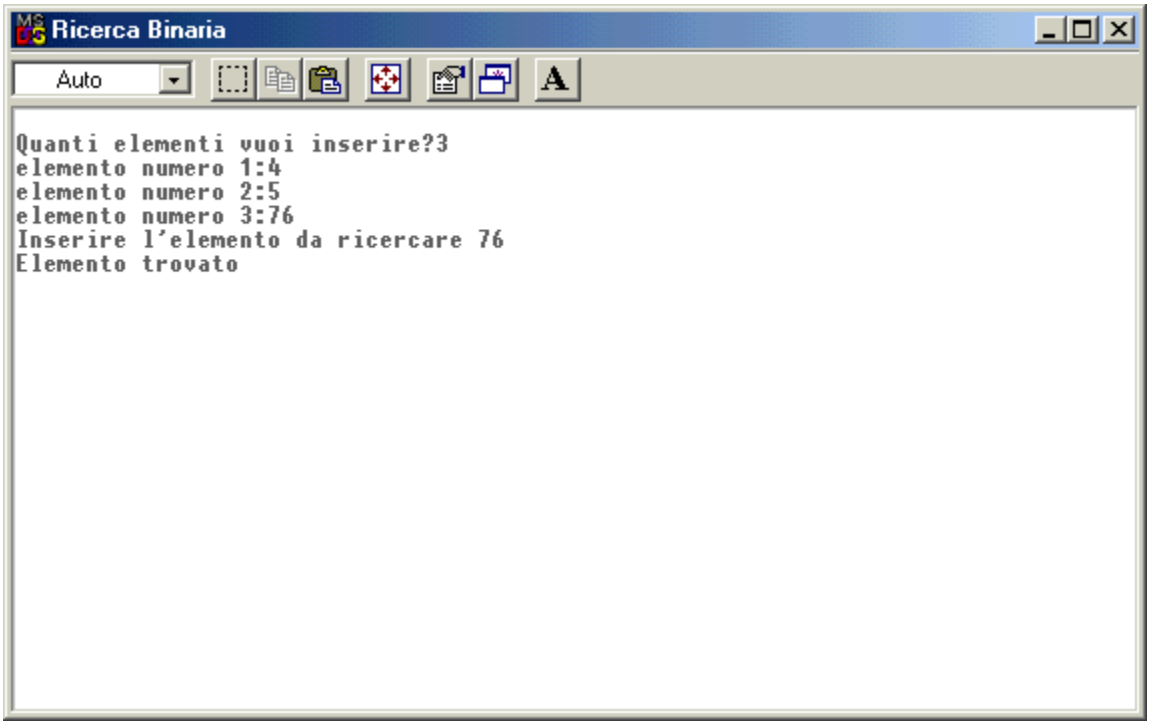
File libreria "ric-bin.c"

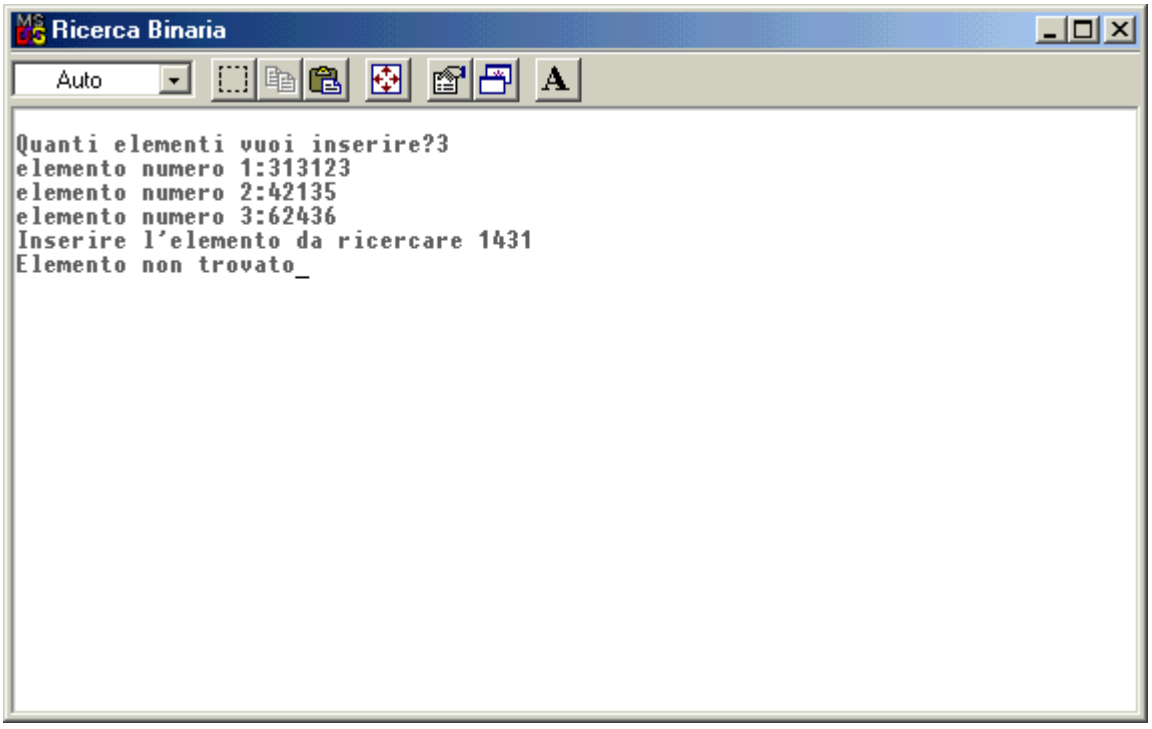
```
/* Dichiarazione di un nuovo tipo booleano */
typedef enum {false, true} logical;


logical RICERCA_BINARIA (int *vett, int n, int k)
{
      /*dichiarazione di variabili locali */
      int m, inizio, fine;
      logical trovato;

      /* Ricerca dell'elemento scelto */
      inizio=0;
      n--;
      fine =n;
      trovato =false;
      do
      {
            m=(inizio + fine)/2;
            if ( vett[m]==k)
                  trovato = true;
      else
            if (k>vett[m])
                  inizio=m+1;

      else
            if (k<vett[m])
                  fine=m-1;

      }while ((!trovato)&&(inizio <= fine));

      return trovato;

}
```

File d'esempio che permette l'uso della libreria "ric-bin.c"

```
/* Parte Preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "ric-bin.c"

main ( )
{
      /* dichiarazione di variabili */
      int *a, n, chiave, i;

      /* Inserimento dell'array ordinato */
      printf ("\nQuanti elementi vuoi inserire?");
      scanf ("%d", &n);

      /* alloca spazio dinamicamente */
      a=(int *) calloc (n, sizeof(int));

      for(i=0;i<n;i++)
      {
            printf ("elemento numero %d:",i+1);
            scanf ("%d", &a[i]);
      }

      printf ("Inserire l'elemento da ricercare ");
      scanf ("%d", &chiave);

      /* Visualizzazione del risultato richiamando la function RIC. BIN. */
      if (RICERCA BINARIA (a, n, chiave))

            printf ("Elemento trovato");

      else

            printf ("Elemento non trovato");

}
```

Esempio d'uso





Ricerca binaria – Esempio d'uso

Ricerca sequenziale

- **Scopo:** stabilire se un dato assegnato appartiene ad un insieme, usando il metodo della ricerca sequenziale;

- **Specifiche della funzione:**

  File libreria esterna "`ric-seq.c`"

  Funzione:

```
int Ricerca_seq (int n, int i, int CHIAVE, int *a)
```

  dove:

  n è la dimensione dell'insieme in cui effettuare la ricerca;

  i è l'indice;

  CHIAVE è l'elemento da ricercare.

  ***a** è il puntatore all'array degli elementi.

- **Breve descrizione dell'algoritmo:** utilizzando un ciclo di tipo while, la funzione esegue una serie di confronti tra l'elemento da ricercare ed i vari componenti dell'insieme, fino a che l'elemento in questione non viene trovato;

- **Complessità computazionale:** l'operazione dominante del programma è il confronto. Nel caso peggiore verranno effettuati n-1 confronti; nel caso migliore 1 confronto.

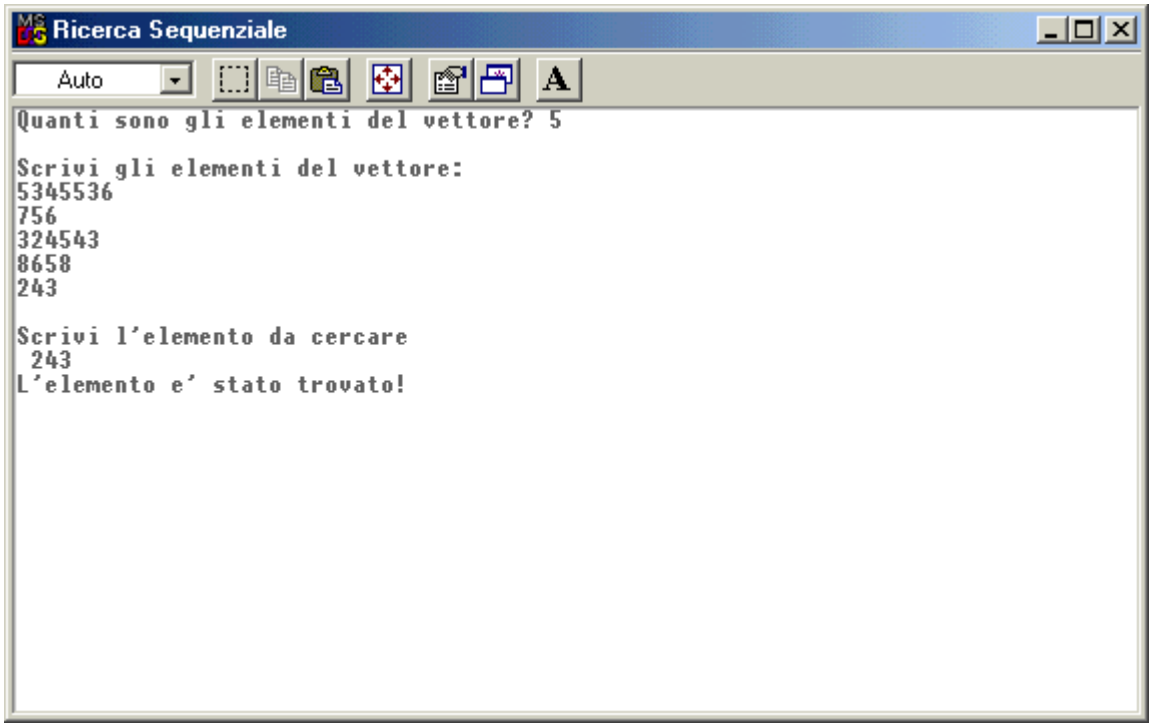
File libreria "`ric-seq.c`"

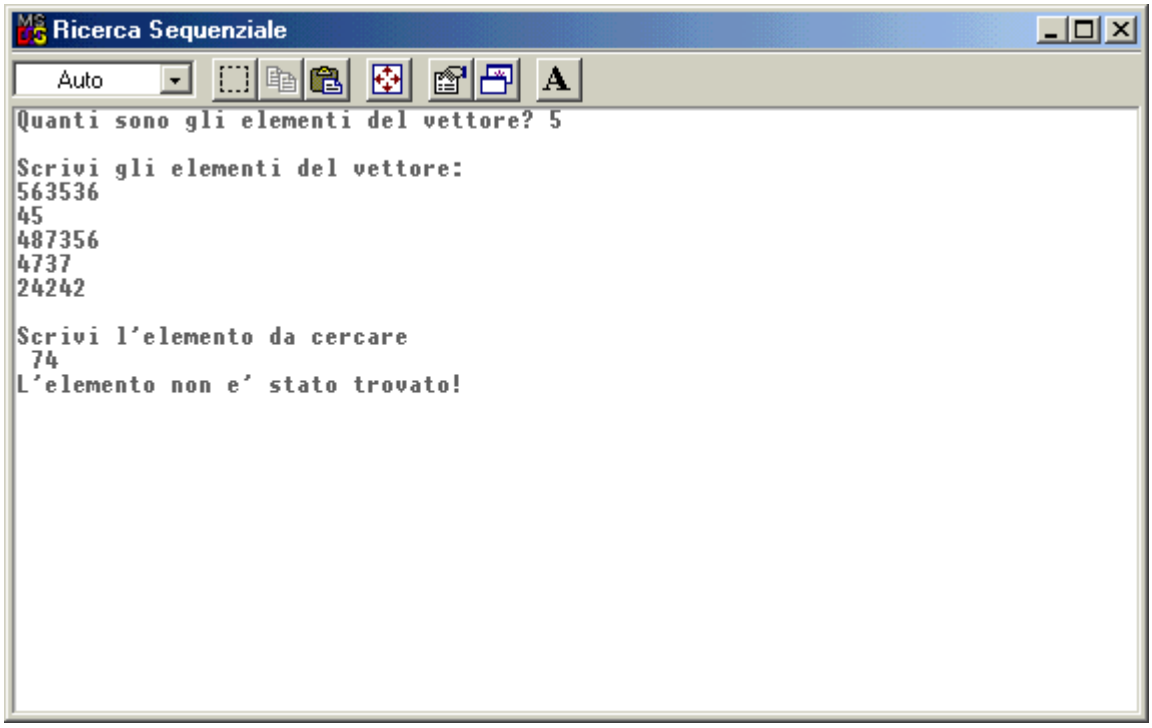
```
/* Funzione di ricerca sequenziale */
int Ricerca_seq (int n, int i, int CHIAVE, int *a)
{

      int e;

      e=0;
      i=0;

      while(i<n)
      {
            if(CHIAVE == a[i])
                  e=1;
      i++;
      }

      return;
}
```

File d'esempio che permette l'uso della libreria "ric-seq.c"

```
/* Parte Preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "ric-seq.c"

main( )
{
      int E, CHIAVE;
      int n, i, r, *a;
      printf("Quanti sono gli elementi del vettore? ");
      scanf("%d", &n);

      /* Alloca spazio dinamicamente */
      a=(int*) calloc (n, sizeof (int));

      printf("\nScrivi gli elementi del vettore: \n");

      for(i=0;i<n;i++)
            scanf("%d", &a[i]);

      printf("\nScrivi l'elemento da cercare\n ");
      scanf("%d", &CHIAVE );

      /* Ricarica la funzione da file */
      Ricerca seq (n, i, CHIAVE, a);

      if(E==1)
            printf("L'elemento e' stato trovato!");
      else
            printf("L'elemento non e' stato trovato!");

}
```

Esempio d'uso





Ricerca sequenziale – Esempio d'uso

Algoritmi combinatori

Permutazioni caratteri

- **Scopo:** lo scopo del programma è quello di combinare i caratteri che vengono immessi, senza creare ripetizioni;

- **Specifiche della funzione:**

  Funzione:

  ```
  void perm_ric(char *arr_perm, int pri, int ult)
  ```

  dove:

  arr_perm è la grandezza dell'array di dimesione dinamica

  pri è l'inzio dell'array, all'inizio posto uguale a 0

  ult è l'ultima posizione dell'array, cioè l'**n-1** elemento

- **Breve descrizione dell'algoritmo:** una volta immessi i dati, la funzione analizza se il primo elemento è uguale all'ultimo, e stampa la permutazione, mediante un ciclo for. In caso contrario (else), la funzione attiva un altro ciclo for, e carica la libreria di scambio e incominca a modificare la posizione degli elementi, e poi, ricorsivamente, carica la fuzione che ha il parametro **pri** incrementato di 1 e continua ricaricando la funzione di scambia per continuare la permutazione.

- **Complessità computazionale:** la funzione esegue permutazioni pari a **n!**

- **Raccomandazioni d'uso:** la funzione poggia sul file libreria esterna "`scambia.c`"

File “permcar.c”

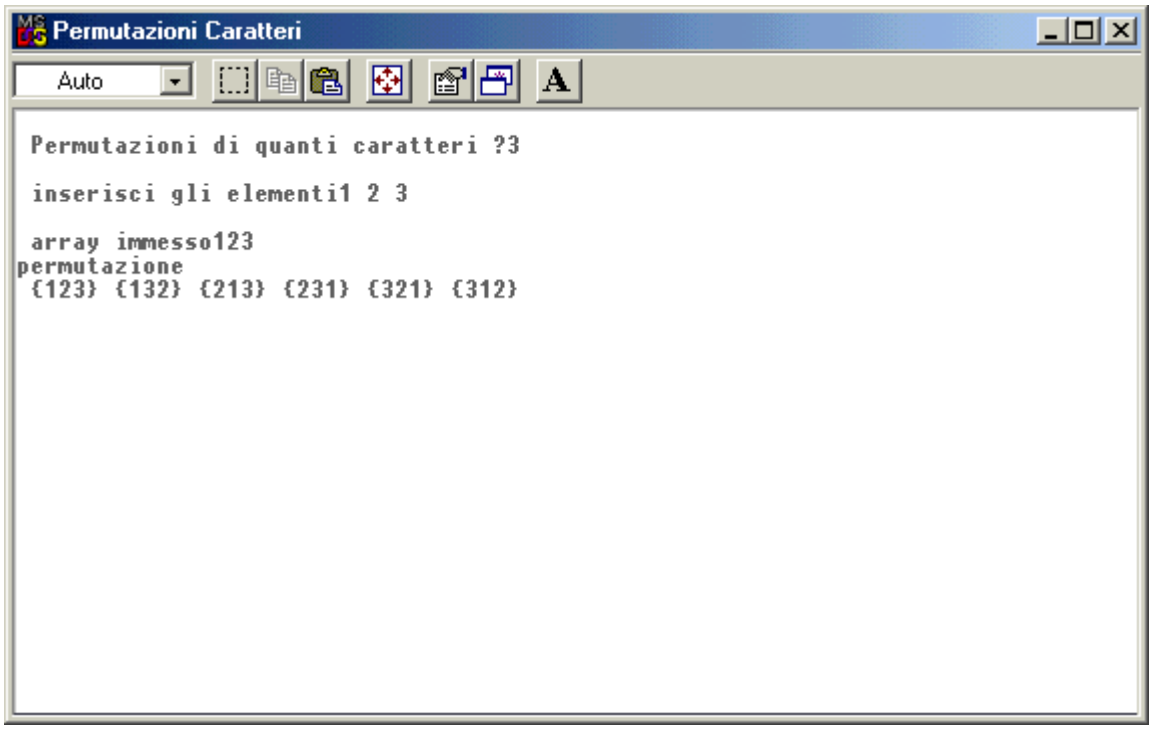
```
/* Parte Preprocessore */
#include <stdio.h>
#include <stdlib.h>

/* Libreria utente */
#include "scambia.c"

void perm_ric(char *arr_perm,int pri,int ult)
{
      int i;
      if (pri == ult)
   {
      printf(" {");
         for(i = 0;i<=ult;i++)
            printf("%c", arr_perm[i]);

      printf("}");
   }

   else
   {

     for(i = pri; i <= ult; i++)
     {
                  /* File della libreria utente scambia.c */
                  swap (&arr perm[pri], &arr perm[i]);


                  /* Richiamo della funzione perm ric */
                  perm ric(arr perm, pri+1, ult);


                  /* File della libreria utente scambia.c */
                  swap (&arr perm[pri], &arr perm[i]);
      }
   }
}

/* Funzione main */

int main()
{
      int n_elem,i;
      int primo;
      int ultimo;
      char *arr;

       printf("\n Permutazioni di quanti caratteri ?");
      scanf("%d",&n_elem);

      /* Allocazione di memoria dinamica */
      arr=(char*)malloc(sizeof(char)*(n_elem+1));

      printf("\n inserisci gli elementi separati da uno spazio: ");

      for(i=0 ; i < n_elem; i++)
      scanf("%s",&arr[i]);

      printf("\n array immesso");
```

```
    for(i=0 ; i < n_elem; i++)
    printf("%c",arr[i]);

    primo=0;
    ultimo=n_elem-1;
    printf("\npermutazione\n");

    /* Carica la funzione della permutazione caratteri ricorsiva */
    perm ric(arr,primo,ultimo);

    /* Libera la memoria allocata dinamicamente */
    free (arr);

    return 0 ;
}
```

Funzione "void perm ric(char *arr perm,int pri,int ult)"

```
void perm_ric(char *arr_perm,int pri,int ult)
{
      int i;
      if (pri == ult)
   {
      printf(" {");
         for(i = 0;i<=ult;i++)
            printf("%c", arr_perm[i]);

      printf("}");
   }

   else
   {

     for(i = pri; i <= ult; i++)
     {
                  /* File della libreria utente scambia.c */
                  swap (&arr perm[pri], &arr perm[i]);


                  /* Richiamo della funzione perm_ric */
                  perm ric(arr perm, pri+1, ult);


                  /* File della libreria utente scambia.c */
                  swap (&arr perm[pri], &arr perm[i]);
      }
   }
}
```

Esempio d'uso



Permutazione caratteri – Esempio d'uso

Generatore di combinazioni

- **Scopo:** il presente programma si propone di generare tutte le possibili combinazioni dei primi n numeri naturali presi r alla volta;

- **Specifiche della funzione:**

  Funzione:

  ```
  void gen_comb(int* arr_gen,int n,int r,int k)
  ```

  dove:

  ***arr_gen** è il puntatore all'array dove stampare le combinazioni

  n è il numero degli elementi da combinare

  r è lentità della combinazione

  k è l'indice dell'array dove stampare le combinazioni

- **Breve descrizione dell'algoritmo:** da fà.

- **Complessità computazionale:** la ricorsione applicata dalla funzione presenta una profondità pari al valore di r.

Funzione "void gen comb(int* arr gen,int n,int r,int k)"

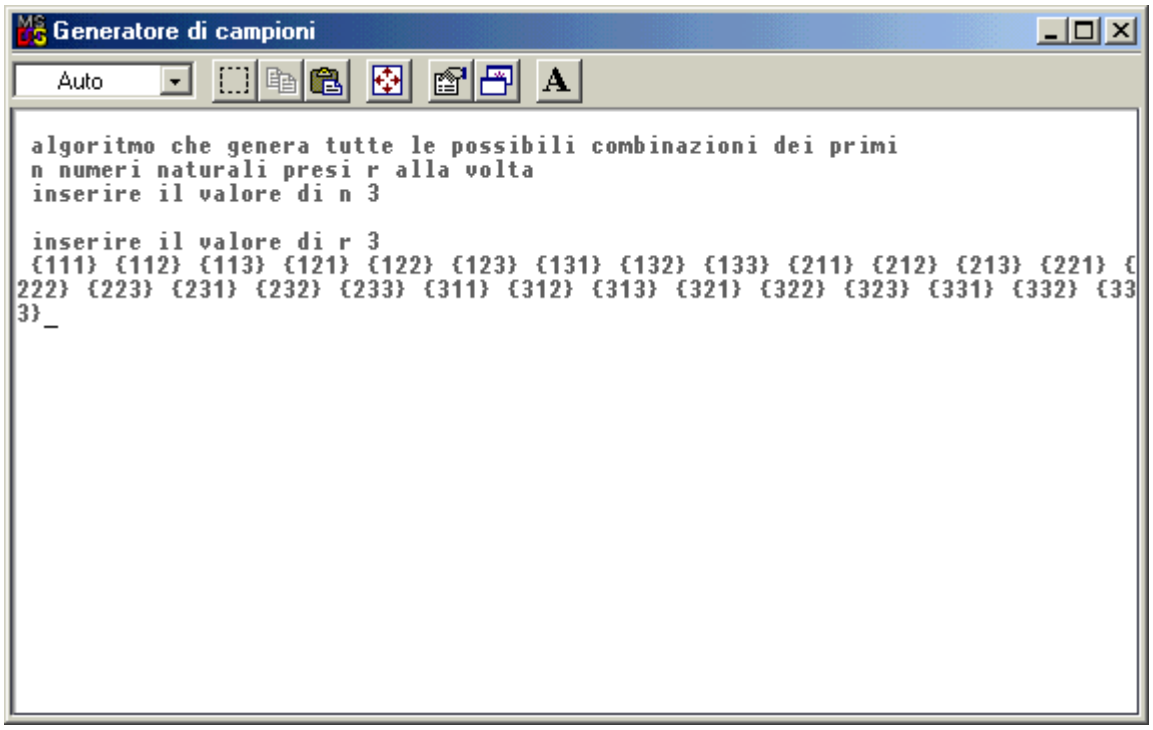
```
/*generazione di combinazioni dei primi n numeri presi r alla volta*/
void gen_comb(int* arr_gen,int n,int r,int k)
{
      int i;
      arr_gen[k]=0;

      for (i=arr_gen[k];i<n;i++)
      {

         arr_gen[k]++;/*scrive il numero nella colonna*/

         if (k < r)  /*condizione di chiamata ricorsiva*/
                  gen_comb(arr_gen,n,r,k+1);
         else
         {
                  printf(" {");
                  for (i=0;i<n;i++)
                  {
                     printf("%d",arr_gen[i]);
                     printf("}");
                  }
         }
}
```

File “gen-camp.c”

```
/* Parte preprocessore */

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

/*generazione di combinazioni dei primi n numeri presi r alla volta*/
void gen_camp(int* arr_gen,int n,int r,int k)
{
      int i;
      arr_gen[k]=0;

      for(i=arr_gen[k];i<n;i++ )
      {

         arr_gen[k]++;/*scrive il numero nella colonna*/

         if (k < r)  /*condizione di chiamata ricorsiva*/
                  gen_camp(arr_gen,n,r,k+1);
         else
         {
            printf(" {");
            for (i=0;i<n;i++)
            {
               printf("%d",arr_gen[i]);
               printf("}");
            }
      }
}


main()
{
      int* arr;
      int k;
      int num_el,r_volte;
      printf("\n  algoritmo  che  genera  tutte  le  possibili  combinazioni  dei
primi");
      printf("\n n numeri naturali presi r alla volta");
      printf("\n inserire il valore di n ");
      scanf(" %d",&num_el);

      num_el=fabs(num_el);

      printf("\n inserire il valore di r ");
      scanf(" %d",&r_volte);

      r_volte=fabs(r_volte);

      /*allocazione dell'array dove stampare le permutazioni*/
      arr=(int*)malloc(sizeof(int)*r_volte);

      /*all'inizio cnt = 0 poi si deve incrementare all'interno della function*/
      k=0;

      /* Carica la funzione di generazione campioni */
      gen_camp(arr,num_el,r_volte,k);

      /* Libera la memoria allocata dinamicamente */
      free(arr);
```

```
    return 0;
}
```

Esempio d’uso



Generatore di campioni – Esempio d’uso

Algoritmi floating point

Epsilon macchina
(Singola precisione)

- **Scopo:** calcolare l'epsilon macchina di un calcolatore in singola precisione (max 23 cifre decimali);

- **Specifiche della funzione:**

  Funzione:

  ```
  void EPSILON_MACCHINA (int b, float E)
  ```

  dove:

  b è la grandezza dell'array di dimesione dinamica

  E è il calcolo del valore di Epsilon macchina in singola precisione

- **Breve descrizione dell'algoritmo:** utilizzando un ciclo di tipo while, la funzione calcola il valore di epsilon, dividendo una variabile temporanea per la base, fino a quando, trovato tale valore, lo restituisce al programma chiamante;

- **Complessità computazionale:** il programma implica un'esecuzione di divisioni, addizioni e una conversione temporanea di tipo della variabile "base";

Funzione "**float EPSILON MACCHINA (int b, float E)**"

```
/* Funzione Epsilon Macchina */
float EPSILON_MACCHINA (int b, float E)
{

      /*dichiarazione di variabili */
      float eps, E2;

      while (E2!=1)
      {
            eps =E;
            E =E/(float)b;
            E2 =E+1;
      }

      /*ritorna il valore dell'epsilon */
      return eps;
}
```

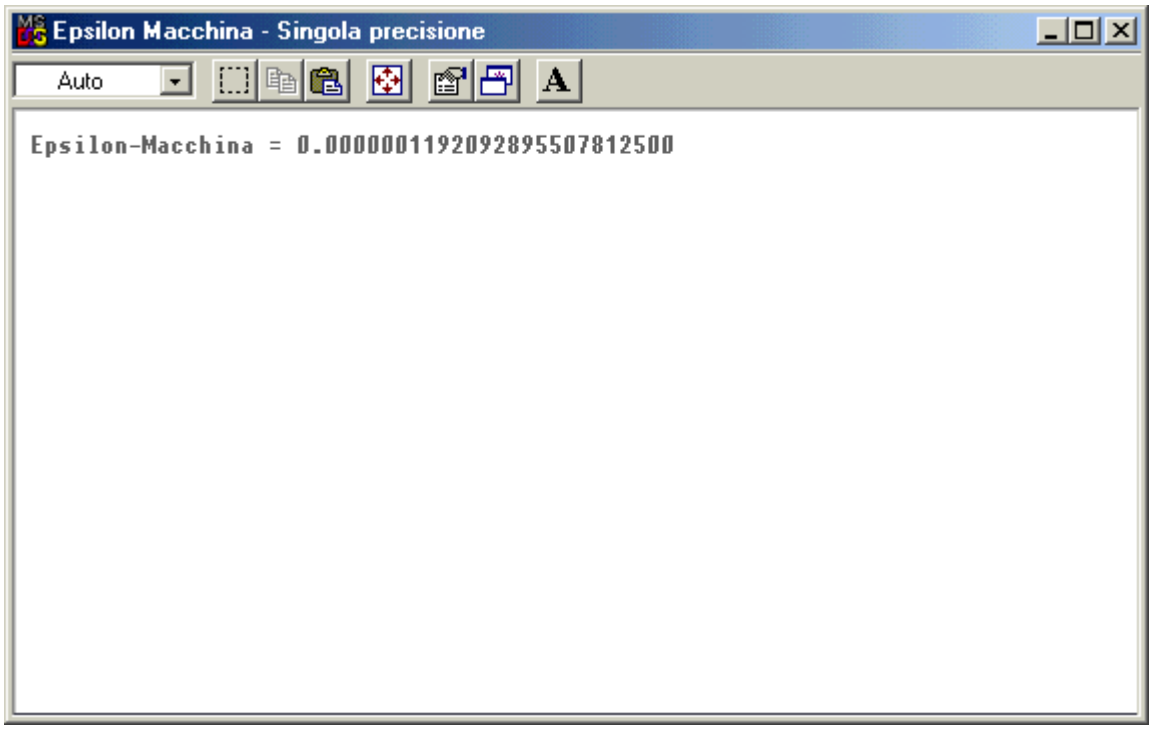
File "**epsilon.c**"

```
/* Parte Preprocessore */
#include <stdio.h>


/* Funzione Epsilon Macchina */
float EPSILON_MACCHINA (int b, float E)
{

      /*dichiarazione di variabili */
      float eps, E2;

      while (E2!=1)
      {
            eps =E;
            E =E/(float)b;
            E2 =E+1;
      }

      /*ritorna il valore dell'epsilon */
      return eps;
}


/* Funzione Main */

main ( )
{
      /*dichiarazione di variabili */
      float E, epsilon;
      int base;

      E =1.0;
      base =2;

      /*richiamo della funzione EPSILON MACCHINA */
      epsilon =EPSILON MACCHINA (base, E);

      /*stampa risultato */
      printf ("\nEpsilon macchina=%.16f\n", epsilon);
}
```

Esempio d'uso

Di seguito è riportato un esempio d'uso.

Il metodo è quello dello screen shot di Windows



Epsilon macchina singola precisione
Esempio d'uso

Epsilon macchina
(Doppia precisione)

- **Scopo:** calcolare l'epsilon macchina di un calcolatore in doppia precisione (max 52 cifre decimali);

- **Specifiche della funzione:**

  Funzione:

  ```
  void EPSILON_MACCHINA (int b, double E)
  ```

  dove:

  b è la grandezza dell'array di dimesione dinamica

  E è il calcolo del valore di Epsilon macchina in doppia precisione

- **Breve descrizione dell'algoritmo:** utilizzando un ciclo di tipo while, la funzione calcola il valore di epsilon, dividendo una variabile temporanea per la base, fino a quando, trovato tale valore, lo restituisce al programma chiamante;

- **Complessità computazionale:** il programma implica un'esecuzione di divisioni, addizioni e una conversione temporanea di tipo della variabile "base";

Funzione "**double EPSILON MACCHINA (int b, float E)**"

```
/* Funzione Epsilon Macchina in doppia precisione */
double EPSILON_MACCHINA (int b, double E)
{

/*dichiarazione di variabili */
      double eps, E2;

      while (E2!=1)
      {
            eps =E;
            E =E/(double)b;
            E2 =E+1;
      }
      /*ritorna il valore dell'epsilon */
      return eps;
}
```

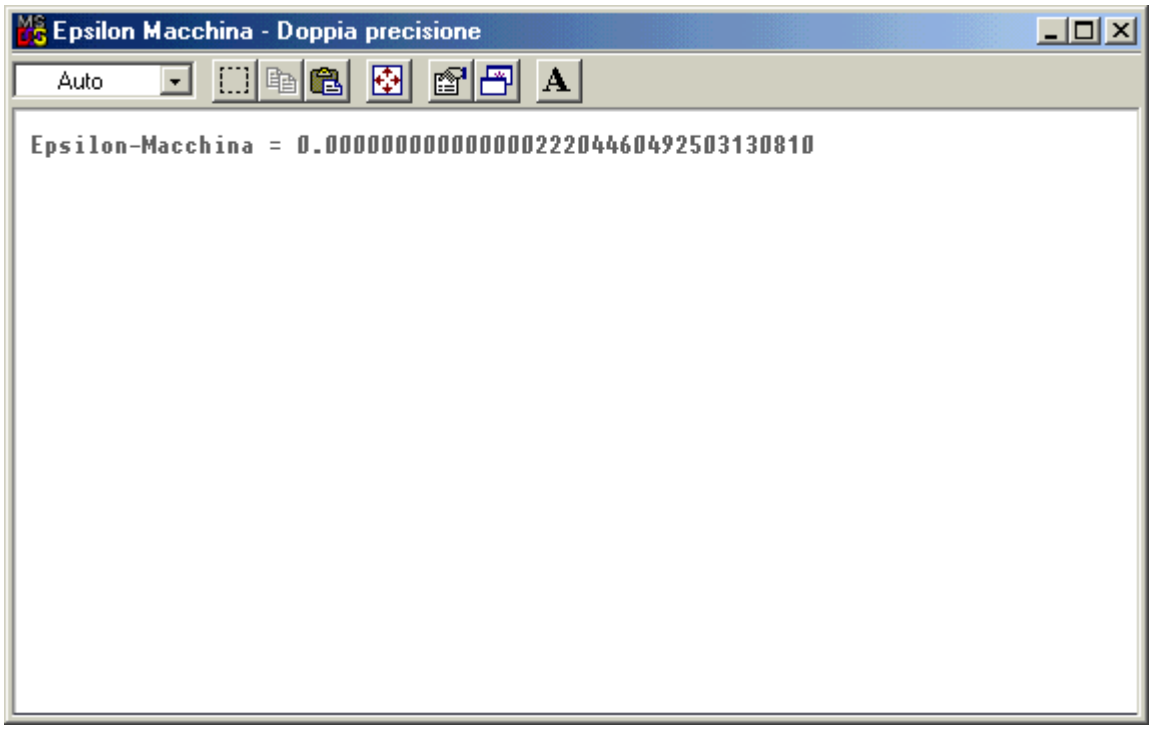
File “**epsilon2.c**”

```
/* Parte preprocessore */
#include <stdio.h>


/* Funzione Epsilon Macchina in doppia precisione */
double EPSILON_MACCHINA (int b, double E)
{

/*dichiarazione di variabili */
      double eps, E2;

      while (E2!=1)
      {
            eps =E;
            E =E/(double)b;
            E2 =E+1;
      }
      /*ritorna il valore dell’epsilon */
      return eps;
}



/* Funzione Main */

main ( )
{
      /*dichiarazione di variabili */
      double E, epsilon;
      int base;

      E =1.0;
      base =2;

      /*richiamo della funzione EPSILON MACCHINA */
      epsilon =EPSILON MACCHINA (base, E);

      /*stampa risultato */
      printf (“\nEpsilon macchina=%.32f\n”, epsilon);
}
```

Esempio d'uso

Di seguito è riportato un esempio d'uso.

Il metodo è quello dello screen shot di Windows



Epsilon macchina singola doppia
Esempio d'uso

Ricerca minimo rappresentabile

- **Scopo:** la funzione individua il più piccolo numero rappresentabile del sistema aritmetico, in singola precisione;

- **Specifiche della funzione:**

  Funzione:

  ```
  float minimo_rappresentabile_sp (float x,int b)
  ```

  dove:

  x

  b base (binaria) nella quale calcolare il minimo

- **Breve descrizione dell'algoritmo:** utilizzando un ciclo while, la funzione effettua ripetute divisioni del numero x per la base (che in questo caso è 2), allo scopo di determinare il più piccolo numero rappresentabile;

- **Complessità computazionale:** dipende dalla precisione del sistema aritmetico;

Funzione "**float minimo rappresentabile sp (float x,int b)**"

```
/* Funzione del minimo rappresentabile */
float minimo_rappresentabile_sp (float x,int b)
{
 float rmin;
 while (x != 0.)
 {
      rmin=x;
      x=rmin/(float)b;
 }

 return rmin;
}
```

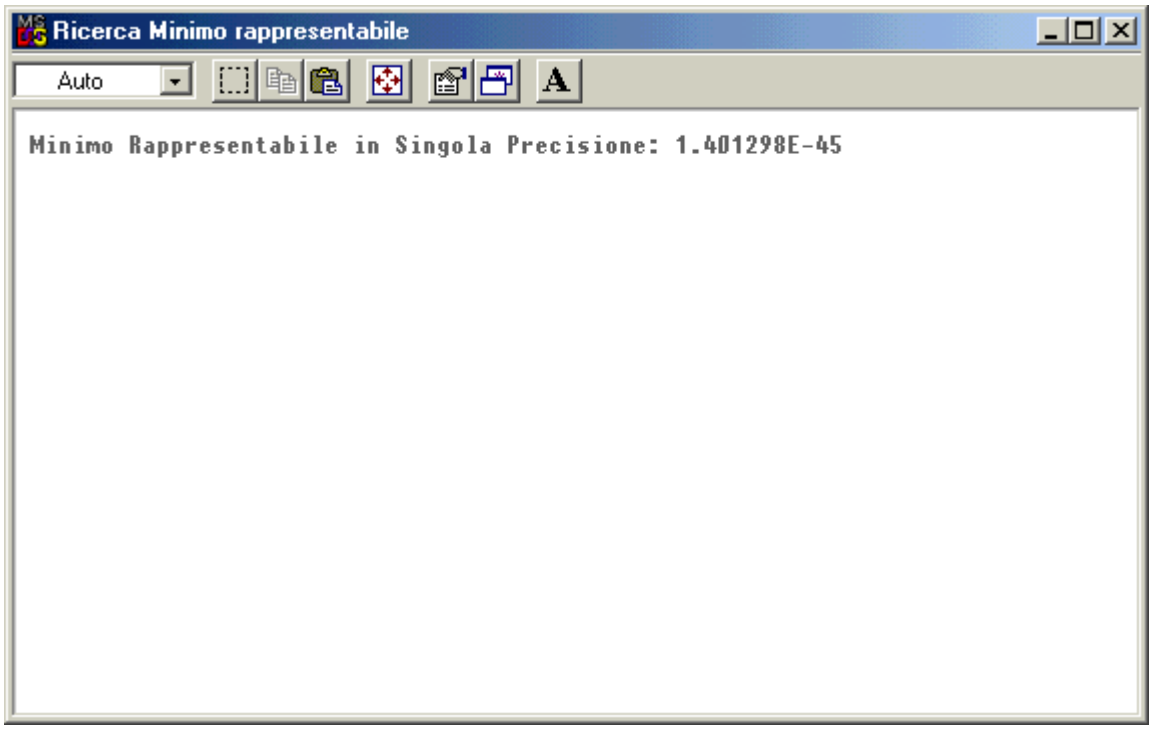
File "**ric-min.c**"

```
/* Parte preprocessore */
#include <stdio.h>

/* Funzione del minimo rappresentabile */
float minimo_rappresentabile_sp (float x,int b)
{
 float rmin;
 while (x != 0.)
 {
      rmin=x;
      x=rmin/(float)b;
 }

 return rmin;
}

/* Funzione Main */
main ()
{
 float e,rmins;

 int base;

 e=1.;

 base=2;

 /* Carica la funzione del Minimo Rappresentabile */
 rmins=minimo rappresentabile sp (e,base);

 printf ("\n Minimo Rappresentabile in Singola Precisione: %E",rmins);

}
```

Esempio d’uso

Di seguito è riportato un esempio d’uso.

Il metodo è quello dello screen shot di Windows



Ricerca minimo rappresentabile
Esempio d’uso

Algoritmi di
Complessità Computazionale

Complessità computazionale
Algoritmo di Horner

- **Scopo:** effettuare la valutazione di un polinomio in un punto prefissato, utilizzando il metodo sperimentato da Hostrowski nel 1954, e noto come algoritmo di Horner.;

- **Specifiche della funzione:**

  Funzione:

  ```
  float horner (int dim, float c[], float p)
  ```

  dove:

  dim è il grado del polinomio

  c[] è l'array (statico) dei coeffiienti

  p incognita del polinomio

- **Breve descrizione dell'algoritmo:** dopo l'inserimento dei dati riferiti al grado del polinomio, coefficienti e valutazione del punto in cui deve essere valutato, il programma carica la funzione del calcolo sperimentato da Hostrowski;

- **Complessità computazionale:** la complessità computazionale dell'algoritmo consiste in **n moltiplicazioni** ed **n addizioni**;

File “**float horner (int dim,float c[],float p)**”

```
/* Funzione del calcolo del polinomio di Horner */
float horner (int dim,float c[],float p)
{
 int indice;
 float ris;
 ris=c[dim];

 for(indice=dim-1;indice>=0;indice--)
    ris=ris*p+c[indice];
 return ris;
}
```

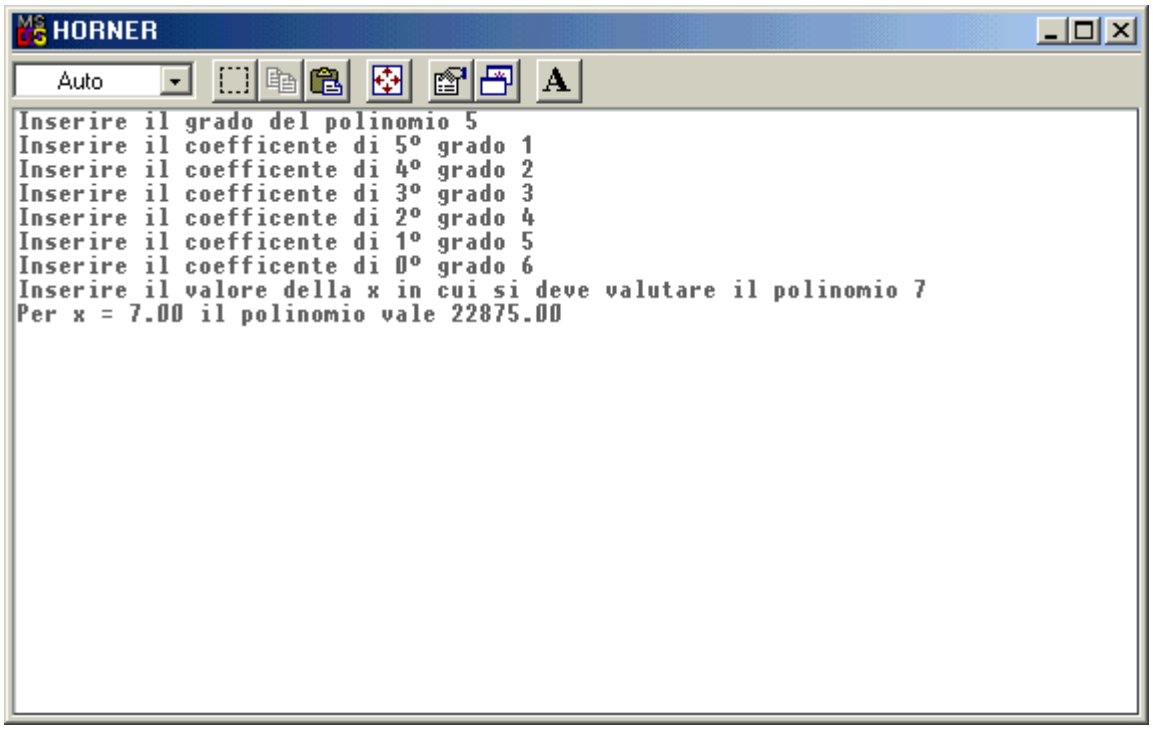
File "**horner.c**"

```
/* Parte Preprocessore */
#include <stdio.h>

/* Funzione del calcolo del polinomio di Horner */
float horner (int dim,float c[],float p)
{
 int indice;
 float ris;
 ris=c[dim];

 for(indice=dim-1;indice>=0;indice--)
    ris=ris*p+c[indice];
 return ris;
}

/* Funzione Main */
main ()
{
 int i,n;
 float a[50],x,v;

 printf ("Inserire il grado del polinomio ");
 scanf ("%d",&n);

 /* Lettura dei coefficenti del polinomio */
 for(i=n;i>=0;i--)
 {
    printf ("Inserire il coefficente di %iø grado ",i);
    scanf ("%f",&a[i]);
 }

 /* lettura del punto in cui si deve valutare il polinomio */
 printf ("Inserire il valore della x in cui si deve valutare il polinomio ");
 scanf ("%f",&x);

 /* Richiamo della Formula risolutiva dell'Algoritmo di Horner */
 v=horner(n,a,x);

 /* Visualizzazione del risultato */
 printf ("Per x = %3.2f il polinomio vale %3.2f",x,v);

}
```

Esempio d’uso

Di seguito è riportato un esempio d’uso.

Il metodo è quello dello screen shot di Windows



Complessità computazionale – HORNER
Esempio d’uso

Indice

Premessa

LIBERTÀ EGUAGLIANZA

MONITORE NAPOLETANO

Fondato nel 1799 da
Carlo Lauberg ed Eleonora de Fonseca Pimentel

Rifondato nel 2010
Direttore: Giovanni Di Cecca

Anno CCXXI

Contatti

C.Ph.: +39 392 842 76 67

www.monitorenapoletano.it

info@monitorenapoletano.it