LUCIDI C

Corso di linguaggio C, F. Sacco 2020 – fabriziosacco.it

Sintassi evidenziata – Editor on line – Compiler on Line

LINGUAGGIO C

Dennis Ritchie (anni 70)

Progettato per il S.O. UNIX

BCPL → B (Thompson 1970) → C (Ritchie)

Nato insieme a **Unix**, è supportato dalla **totalità dei sistemi operativi** di largo uso impiegati ed è impiegato principalmente per la realizzazione di **sistemi operativi**, **linguaggi di programmazione**, librerie, giochi e per applicazioni altamente performanti; è **rinomato per la sua efficienza** e si è imposto come linguaggio di riferimento per la realizzazione di software di sistema su gran parte delle piattaforme hardware moderne. La standardizzazione del linguaggio (da parte prima dell'ANSI e poi dell'ISO) garantisce la portabilità dei programmi scritti in C (standard, spesso detto ANSI C) su qualsiasi piattaforma;
(da wikipedia.org)

INTRODUZIONE AL LINGUAGGIO

caratteristiche del linguaggio:

1) Linguaggio relativamente a basso livello

2) Linguaggio di modeste dimensioni

3) Non dispone di operazioni per trattare direttamente oggetti strutturati (stringhe, array, insiemi, etc.)

4) Non possiede propriamente funzioni di input/output

5) Offre costrutti di controllo fra cui:

   if-then-else
   for
   do-while

6) Gestione file NON tipizzata e NON a record, con possibilita' di accesso diretto

7) Notevole indipendenza da elaboratore e da sistema operativo.

VANTAGGI

1) LINGUAGGIO 'GENERAL PURPOSE'

2) POTENZA

3) COMPATTEZZA SORGENTI ED ESEGUIBILI

4) EFFICIENZA ESEGUIBILI

5) PORTABILITA' SW ED HW

6) MODULARITA'

7) POTENTE PRE-PROCESSORE

8) POTENTE GESTIONE DI PUNTATORI

SVANTAGGI

1) SCARSAMENTE TIPIZZATO

2) NECESSITA' DI SIMULARE IL PASSAGGIO PARAMETRI PER VARIABILE CON PUNTATORI

3) IMPOSSIBILITA' DI GESTIRE ARRAY E STRINGHE COME UN TUTT'UNO

4) COMPILATORE: AVARO DI CONTROLLI E SEGNALAZIONI / MESSAGGI DI DIFFICILE INTERPRETAZIONE

5) PORTE APERTE ALLA SCRITTURA DI PROGRAMMI ILLEGGIBILI E/O PERICOLOSI

(Come il seguente … che fa? Come lo fa? Però funziona!)

```c
#include <stdio.h>
int main()
{
   char *t, *s="ciao a tutti";
   for(t=s+11; t+1-s; putc(*t,stdout), --t);
   return 0;
}
```

link consigliati:
http://it.wikipedia.org/wiki/C_(linguaggio)

IL PRIMO PROGRAMMA

OBIETTIVO: realizzare un programma
scritto in C che stampi qualcosa
sul video.

/* Parola.c: Prova di stampa */

```c
#include <stdio.h>
int main()
{
      printf("Funziona !!\n");
      return 0;
}
```

COMPILAZIONE ED ESECUZIONE
(da linea comando)

WINDOWS ---> gcc PAROLA.C → A.EXE

LINUX/UNIX ---> gcc parola.c → a.out

---

link consigliati:

http://www.compileonline.com (compila on line hello.c)

ESERCIZI

1) Spezzare la printf in più printf

2) Togliere o aggiungere \n

3) Eliminare singoli 'pezzi' ed osservare attentamente le segnalazioni del compilatore

4) Inserire \t, \b, \\, \" o altro

(vedi http://it.wikipedia.org/wiki/C_(linguaggio)#Sequenze_di_Escape)

5) Scrivere un programma che stampi una videata a piacere, ad es. un alberello:

```
   *
  ***
 *******
   *
   *
```

TIPI BASE DEL LINGUAGGIO

char ---> 1 byte

short ---> 2 byte

int ---> 2 - 4

long ---> 4 byte

float ---> 4 byte (V.M.)

double --> 8 byte (V.M.)

long double --> 10 - 12 byte (V.M.)

TIPI RICAVATI CON UNSIGNED

unsigned char
unsigned short
unsigned int
unsigned long

(range dei long, con n bit: da 0 a 2^n -1)

IL CICLO while

utilizzo di printf() / commenti / per printf vedi anche pag 12.

/* Ecco un programma che stampa una
tabella di conversione da gradi Fahrenheit a Celsius
Formula: [Cel = 5/9 * (Fah-32)] */

```
#include <stdio.h>
int  main()
{
     int   minimo, massimo, passo;
     float  fahr,   celsius;
     minimo  =   0;
     massimo = 300;
     passo   =  20;
     fahr    =  minimo;
     while ( fahr <= massimo )
     {
        celsius=(5.0/9.0)*(fahr-32.0);
        printf("%4.0f %6.1f\n",
                fahr,celsius);
        fahr = fahr + passo;
     }
     return 0;
}      // domande: perchè 5.0 invece di 5? cos'è %4.0f ?
```

// link:
// http://it.wikibooks.org/wiki/C/Blocchi_e_funzioni/Cicli#Ciclo_while

CICLO while

Osservazioni:

CICLO while, forma 1):

```
while ( condizione )
        istruzione;
```

OPPURE forma 2):

```
while ( condizione )
{
        istruzione1;
        istruzione2;
        ..........
        istruzioneN;
}
```

OPERATORI :

di confronto:

maggiore	>
minore	<
uguale	==
diverso	!=
maggiore uguale	>=
minore uguale	<=

di assegnamento:

assegna	=

logici booleani:

AND	&&
OR	||
NOT	!

valori booleani:

VERO	qualunque, purche' non 0
FALSO	0

(recentemente il C ha aggiunto i tipi **bool** e i valori **true** e **false**)

operatori algebrici:

PIU'	+	
MENO	-	
PER	*	
DIVISO	/	
MODULO	%	(il resto della divisione intera)

LA FUNZIONE STANDARD printf()

%d ---> int, short (char)

%ld ---> long

%o ---> interi in ottale

%x ---> interi in esadec.

%c ---> caratteri

%s ---> stringhe

%f ---> float, double

%Lf ---> long double

```c
/* Il seguente programma (tipibase.c) mostra
l'input/output dei principali tipi base con
scanf/printf:*/

#include <stdio.h>

char          c;
short         s;
int           i;
long          l;
float         f;
double        d;
long double   D;

int main()
{
     printf("Un carattere       ->"); scanf("%c", &c);
     printf("--> %c\n", c);
     printf("Un numero (short ) ->"); scanf("%hd", &s);
     printf("--> %d\n",s);
     printf("Un numero (int   ) ->"); scanf("%d", &i);
     printf("--> %d\n", i);
     printf("Un numero (long  ) ->"); scanf("%ld",&l);
     printf("--> %ld\n",l);
     printf("Un numero (float ) ->"); scanf("%f", &f);
     printf("--> %f\n", f);
     printf("Un numero (double) ->"); scanf("%lf",&d);
     printf("--> %f\n",d);
     printf("Un n.(long double) ->"); scanf("%Lf",&D);
     printf("--> %Lf\n",D);


     printf("\n\n\n");
     printf("Dimensione di char   --> %2d bytes\n",sizeof(c));
     printf("Dimensione di short  --> %2d bytes\n",sizeof(s));
     printf("Dimensione di int    --> %2d bytes\n",sizeof(i));
     printf("Dimensione di long   --> %2d bytes\n",sizeof(l));
     printf("Dimensione di float  --> %2d bytes\n",sizeof(f));
     printf("Dimensione di double --> %2d bytes\n",sizeof(d));
     printf("Dim. di  long double --> %2d bytes\n",sizeof(D));

     return 0;
}
```

CICLO for

Uno dei costrutti piu' eleganti, compatti e versatili in C: vediamo come si puo' riscrivere il programma Fahrenheit-Celsius usando il for anziche' il while.

```
/* tabella fahr. Celsius */


#include <stdio.h>


int main()
{
 int fah;
 for(fah =0; fah <=300; fah +=20)
        printf("%4d %6.1f\n",fah,
                (5.0/9.0)*(fah-32.0));
 return 0;
}
```

Forma generale del ciclo for:

```
for (inizializzazione; condizione; incremento)
        istruzione;
```

Altri esempi di for corretti

```
1)  for ( ;  ; )
              ;  // loop infinito


2)  for  ( ; n >= 0; n-- )
          istruzione;

3)  for (i = 0; i < max; i++)
    {   // loop tipico
        istruzione1;
        istruzione2;
        ...........
        istruzioneN;
    }

// Il seguente è pure corretto...
 4)  for(i=0, j=0, k=1000;
        i<k && j%k != 333;
        i++, k--, j+= k/i)
           istruzione;
```

(Approfondimento) → COSTANTI

```
const int MAX  = 10;
const float PI = 3.14;
```

oppure si puo' utilizzare il preprocessore:

```
#define     MAX  10

#define     MIN   1

#define     MEDIO ((MIN+MAX)/2)
```

Nb: il preprocessore è uno strumento molto potente; la sua utilità non si limita all'utilizzo delle costanti. Ad esempio le seguenti macro sono migliori persino delle funzioni corrispondenti, poiché sono indipendente dai tipi di dati A e B.

```
#define MASSIMO(A,B) A>B?A:B
#define MINIMO(A,B) -(MASSIMO(-A,-B))
#define  MIN3(A,B,C) MINIMO(A,MINIMO(B,C))
```

(Approfondimento)

I/O DA STANDARD INPUT E SU STANDARD OUTPUT BUFFERIZZATO:

getchar() putchar()

Non fanno propriamente parte del linguaggio C, come del resto tutte le funzioni di I/O; sono generalmente delle MACRO:

ESEMPIO:

```
/* Programma cat.c: copia l'input sull'output*/
#include <stdio.h>
int main()
{
        int c;

        while ( (c=getchar() )  != EOF )
                putchar(c);
        return 0;
}
```

// Esercizi consigliati:
1)Usare cat per sostituire il comando type
2)Usare cat per sostituire il comando copy
3)Usare cat per concatenare 3 file di testo
4)Come mai cat.c non copia file binari? (in linux invece fa!)

(Approfondimento getchar/putchar)

Osservazioni:

1) `c = getchar() != EOF`
   NON FUNZIONA

2) `int c / non char`

3) `col for :`

```
for (; (c=getchar())!=EOF; putchar(c))
                ;
```

(Approfondimento getchar/putchar)

/* Conteggio caratteri */

```
#include <stdio.h>
int main()
{
 long nc; /* contatore */
 for (nc=0;getchar()!=EOF;nc++)
                    ;
 printf("caratteri %ld\n",nc);
 return 0;
}
```

/* Conteggio linee */

```
#include <stdio.h>

int main()
{
 int c, nl=0;
 while ((c=getchar())!=EOF)
        if ( c == '\n' )
            ++nl;
 printf("linee %d\n",nl);
 return 0;
}
```

(Approfondimento getchar/putchar)

/* Conteggio parole, linee e caratteri */

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
typedef int boolean;

int main()
{
      int c;
      boolean inparola = FALSE;
      long nl, np, nc;
      nl = np = nc = 0;
      while((c=getchar()) != EOF)
      {
        ++nc;
        if (c == '\n')
                ++nl;
        if (c == ' '|| c == '\n' || c == '\t')
                  inparola = FALSE;
        else if (!inparola)
        {
                  inparola = TRUE;
                  ++np;
        }
      } /* fine while */
      printf("%ld %ld %ld\n", nl,np,nc);
      return 0;
}
```

FUNZIONI :

```
/* prova funzione potenza */

#include <stdio.h>

int pot(int x, int n)
{
      int i,p;

      for (p=1,i=1; i<=n; i++)
            p = p*x;
      return p;
}


int main()
{
      int i;
      for ( i=0; i<10; i++)
            printf("%d\n",pot(2,i));
      return 0;
}
```

IL PASSAGGIO A FUNZIONE DEI TIPI BASE (char, int, long, float, double) E' PER VALORE.

```c
#include <stdio.h>

void prova(int a, int b, int c);
/* prototipo */

int main()
{
        int a,b,c;

        a = 3;
        b = 4;
        c = 5;
        prova(a,b,c);
        printf("%d %d %d",a,b,c);
        return 0;
}

void prova(int a, int b, int c)
{
        c = b;
        c--;
        b = a++;
        a = a % 27 + 77;
}
/* che valori stampa sul video ? */
```

(Approfondimento)

L'OPERATORE ternario ? :

```
int max( int a, int b)
{
        if ( a > b )
                return a;
        else
                return b;
}
```

```
int max(int a, int b)
{
        return a > b ? a : b;
}
```

---

esercizi:

1) funzione che ritorna se un numero e' dispari
2) funzione che ritorna il minimo fra tre numeri

(Approfondimento)

OPERATORI LOGICI ORIENTATI AL BIT:

(si possono applicare ai tipi semplici: char, short, int, long)

&	**AND a bit**
|	**OR a bit**
^	**OR esclusivo a bit**
<<	**SHIFT a sx un bit**
>>	**SHIFT a dx un bit**
~	**COMPLEMENTO a 1**

Esercizi:

```
int bit_acc(unsigned char i)
int n_bit  (unsigned short i, int n)
int rot_sx (unsigned short i, int n)
int rot_dx (unsigned short i, int n)

/* contare i bit a 1; ritornare il
   valore dell'ennesimo bit di i,
   ruotare i bit a sx o a dx */
```

COSA SUCCEDE SE ?

```
/* conversioni implicite */

int main()
{
        char            c;
        unsigned char   n;
        short           s;
        int             i;
        long            l;
        float           f;
        double          d;

        d = 0.003;
        c = d;
        d = '0';
        f = 'a' + 'd' -1;
        l = 75000;
        i = l;
        s = l % 25000 + 31254;
        c = 300 -(c+d);
        c += 256;
        d = 0xFF;
        f = 1.0e-5;
        n = 255;
        return 0;
}

/* utilizzare (tipo) per convertire, es: (int) */
```

VETTORI (ARRAY)

```c
/* programma che conta il numero di
   occorrenze di ogni cifra in un testo */

#include <stdio.h>
int main()
{
        int i,c;
        int cifre[10];
        /* cifre e' un vettore di 10 interi */

        for ( i = 0; i < 10; i++ )
                cifre[i] = 0;
        /* azzero il vettore*/

        while((c = getchar())!= EOF)
                if ( c >= '0' && c <= '9')
                        ++cifre[c-'0'];
/*il programma e' tutto in questa ultima
istruzione!*/

        printf("cifre:\n")
                for (i = 0; i < 10; i++)
                   printf(" %d -- %d\n",
                   i,cifre[i]);
        return 0;
}
```

Suggerimento: compila il programma nel file cifre.c e poi lancialo dal prompt di comandi su un file di testo, come segue:

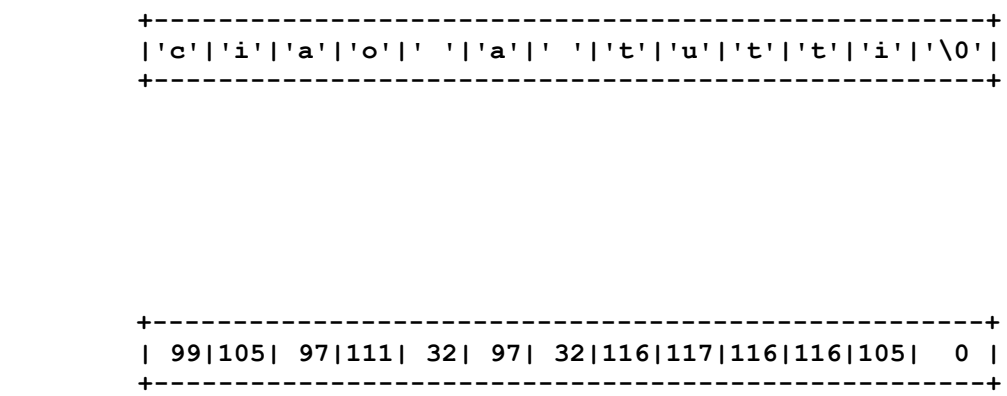
```
cifre.exe < testo.txt
```

STRINGHE IN C

(ARRAY DI CARATTERI TERMINANTI CON IL CARATTERE '\0')

ES:

"ciao a tutti"



IL PASSAGGIO DI ARRAY ALLE FUNZIONI E' SEMPRE PER INDIRIZZO!

Infatti in C il nome del vettore e' sempre l'indirizzo di memoria del suo primo elemento!!!

ALCUNE FUNZIONI DISPONIBILI SULLE STRINGHE:

```
/* copia di stringhe */
void strcpy(char s[], char t[])
{
        int i;
        for (i=0; (s[i]=t[i]) != '\0'; i++)
                        ;
}


/* lunghezza di una stringa */
int strlen(char s[])
{
        int i;
        for (i=0; s[i] != '\0'; i++)
                                ;
        return i;
}


/* confronto fra stringhe */
int strcmp(char s[], char t[])
{
        int i;
        for (i=0; (s[i] == t[i])&&s[i];i++)
                                ;
        return (s[i] - t[i]);
}
```

```c
/* concatenamento di stringhe */
void    strcat(char s[], char t[])
{
        int i,j;
        for (i = 0, j = 0; s[i]; i++)
                ;
        while(s[i++] = t[j++])
                ;
}
```

ESERCIZI:

1) `void squeeze(char s[], char c)`
   `/* toglie le occorrenze di c in s */`

2) `void reverse(char s[])`
   `/* rovescia la stringa */`

3) `void str_up(char s[])`
   `/* converte in maiuscolo s */`

4) `void str_down(char s[])`
   `/* converte in minuscolo s */`

5) `void left(char s[], char t[],int n)`
   `/* in s i primi n caratteri di t */`

6) `void right(char s[],char t[],int n)`
   `/*in s gli ultimi n caratteri di t*/`

7) `int idx(char s[], char t[])`
   `/* ritorna la posizione di s in t */`

(Approfondimento) VISIBILITA' VARIABILI:

1) VARIABILI GLOBALI
2) VARIABILI GLOBALI STATICHE
3) VARIABILI LOCALI AUTOMATICHE
4) VARIABILI LOCALI STATICHE

```c
#include <stdio.h> /* main.c */
int gl;          /*  globale pubblica */
static int gls; /*  globale statica: privata*/

void prova2();      /*  prototipo */

void prova()
{
        static int ls = 0;    /* statica locale */
        int a;                /* volatile locale */
        a++; ls++;   /* cosa c'e' in a?  */
}

int main()
{
        int c;                /* volatile locale */
        prova(); prova(); prova();
        prova2();
        return 0;
}
```

```c
/* file prova2.c */
extern int gl;
/* e'la variabile globale gl di main.c */
void prova2()
{
        gl++;
}
```

if - else

```
A)   if ( condizione )
          istruzione;

B)   if ( condizione )
          istruzione1;
     else
          istruzione2;


C)  if       (condizione1)
           istruzione1;
    else if (condizione2)
           istruzione2;
    else if (condizione3)
           istruzione3;
    ....................
```

switch

_____ : VERIFICA SE UN'ESPRESSIONE COLLIMA CON UNA SERIE DI VALORI COSTANTI.

```
/* Verifica bilanciamento parentesi */
#include <stdio.h>

int main()
{
   int c; /*carattere dall'input*/
   int pg, pt, pq; /* Parentesi Graffe, Tonde, Quadre */

   pg = pt = pq = 0; /* assegnazione multipla */
   while((c = getchar()) != EOF)
// Lettura carattere per carattere, fino a fine file
         switch(c)
         {
             case '{' : pg++;
             break;
             case '}' : pg--;
             break;
             case '(' : pt++;
             break;
             case ')' : pt--;
             break;
             case '[' : pq++;
             break;
             case ']' : pq--;
             break;
         }
         pg||pt||pq ? printf("NO"): printf("SI");
         return 0;
}
```

Suggerimento: compila il programma nel file parentesi.c e poi lancialo dal prompt di comandi su un file di testo, come segue:

```
parentesi.exe < testo.txt
```

(Approfondimento)

ESEMPIO if-else:

```
/* Ricerca elemento in un array
   ordinato: ricerca dicotomica */

int binaria (int x, int v[], int n)
{
    int alto, basso, mezzo;
    basso = 0;     /* estremo inferiore */
    alto  = n-1;   /* estremo superiore */
    while ( basso <= alto )
    {
       mezzo = (basso+alto)/2;
       if ( x < v[mezzo] )
           alto = mezzo-1;
       else if ( x > v[mezzo] )
           basso= mezzo+1;
       else      /* trovato */
           return mezzo;
    }
    return -1; /* non c'e' l'elemento cercato */
}
```

do - while

A)
```
do
      istruzione;
 while(condizione);
```

B)
```
do
{
   istruzione1;
   istruzione2;
   ..........
   istruzioneN;
} while(condizione);
```

ESEMPIO do-while:

Costruiamo una funzione che, dato un numero intero, lo converta in stringa

```
/* Integer TO Ascii */
void itoa(int n, char s[])
{
        int i, segno;
        if((segno=n) < 0)
                n = -n;
        i = 0;

        do
                s[i++] = n % 10 + '0';
        while((n/=10) > 0);

        if ( segno < 0)
                s[i++] = '-';
        s[i] = '\0';
        reverse(s);
/* funzione reverse: da costruire ! */
}
```

Suggerimento: ottieni lo stesso risultato con la funzione di sistema:

sprintf(char s[],"%d",int n)

(Approfondimento)

RICORSIONE:

```
/* una funzione che emuli printf("%d",n) */

#include <stdio.h>

void printd(int n)
{
        int i;
        if ( n < 0 )
        {
                putchar('-');
                n = -n;
        }
        if ((i=n/10) != 0)
                printd(i);
        putchar(n%10 + '0');
}
```

Esercizi:

1) **SCRIVERE printd NON RICORSIVA**
2) **SCRIVERE long fatt(long f) SIA RICORSIVA CHE NON RICORSIVA - (fattoriale).**
3) **SCRIVERE LA FUNZIONE:**
   double pow(double n, int p) - (potenza)

FUNZIONI CHE RITONANO VALORI DIVERSI DA INTERI:

```
/* Funzione radice quadrata */


double sqrt(double num)
{
  double epsi = 0.0001; /*precisione*/
  double rad  = 1;      /*radice    */
  extern double abs(double); /* da codificare*/
  /* prototipo per abs */

  if (num <= 0)
          return (double) 0;
  do
          rad = (num/rad +rad) / 2;
  while(abs(num/(rad*rad)-1) > epsi);
  return rad;
}
```

ESERCIZIO: SCRIVERE FUNZIONE double abs(double n)
PROVARE LA FUNZIONE sqrt CON VALORI DI epsi DIVERSI E CONFRONTARE LA PRECISIONE CON LA VOSTRA CALCOLATRICE

(Approfondimento)
PUNTATORI

#include <stdio.h>

```
int a, b, c;
int *p1, *p2;

int main()
{
  a =  4;
  b = 10;
  p1 = &a;
  printf("Puntato p1  %d\n",*p1);
  printf("Indirizzo di p1 %ld\n",(long)p1);
  p2  =  p1;
  *p2 =   5;
  c   = *p2;
  printf("C contiene  %d\n",c);
  p2++;    /*incremento puntatore */
  (*p1)++; /*incremento contenuto */
  printf("A contiene %d\n",a);
  return 0;
}
```

Cosa viene stampato sullo schermo?

(Approfondimento)

CHE COS'E' IL NOME DI UN ARRAY IN C?

```
#include <stdio.h>

int a, b, c, d, *pi, v[4];

int main()
{
    int i;
    for (i = 0; i < 4; i++)
        v[i] = 2*i;
    pi = &v[0];
    printf(" %d ",*pi);
    pi = v;
    printf(" %d ",*pi);

    a = *(pi+3);
    b = v[3];
    c = *(v+3);
    d = pi[3];

    printf("\n %d %d %d %d",a,b,c,d);
        return 0;
}
```

DIFFERENZA FRA ARRAY E PUNTATORE?
L'ARRAY E' UNA COSTANTE!

(Approfondimento)

PASSAGGIO A FUNZIONI PER INDIRIZZO

```c
#include <stdio.h>

void non_scambia(int a, int b)
{
    int temp = a;
        a    = b;
        b    = temp;
} /* funzione completamente inutile! */



void scambia(int *x, int *y)
{
    int temp = *x;
        *x   = *y;
        *y   = temp;
}


int main()
{
        int a = 3, b = 4;
        non_scambia(a,b);
        printf("%d %d", a,b);
        scambia(&a,&b);
        printf("%d %d", a,b);
        return 0;
}
```

(Approfondimento)

FUNZIONI CON PARAMETRI VISTI COME POINTER

/* confrontale con quelle di pag. 27*/

```
void str_cpy(char *a, char *b)
{
        while ( *a++ = *b++)
            ;
       /* il compilatore potrebbe dare
          warning... ma il codice e' corretto!!
          … nel caso aggiungi parentesi:
          ((*a++=*b++))
       */
}

int str_len(char *s)
{
        char *p;
        for ( p = s; *p; p++)
            ;
        return (int) (p - s);
}

void str_cat( char *a, char *b)
{
        str_cpy(a+strlen(a),b);
}
```

Esercizio:
Riscrivere le funzioni di Pag. 28
vedendo i parametri come pointer.

STRUTTURE (record)

```
#include <stdio.h>
struct cliente {
                    long  codice;
                    char  nome [40+1];
                    char  via  [40+1];
                    char  citta[25+1];
                    short anno;
                  };
struct cliente a,b,c; /*Variabili di tipo cliente*/

void prova_cliente()
{
        a.codice = 11225;
        strcpy(a.nome,"Giorgio Bruni");
        b = a; /* copia di strutture */
        if (strcmp(a.nome,b.nome) == 0)
            printf("Copia Ok!");
        c.anno = 10;
        a.anno = b.anno = c.anno;
}
```

NB: per trovare la dimensione in byte di una struttura si usa l'operatore **sizeof()**

Passaggio a funzione / puntatore a struttura:

```
void azzera_cliente(struct cliente *c)
{
      c->codice  = 0;
      c->anno    = 1980; // ho aperto ditta nel 1980
      c->nome[0] = c->via[0] = c->citta[0] = '\0;
}
```

Richiamo funzione (da main() o altra funzione):

```
azzera_cliente(&a);
```

Accesso diretto a file: funzione fseek()
lettura/scrittura insieme: parametro "+"

```
#include <stdio.h>
#include <stdlib.h>
char nomefile[] = "numeri.dat";

void main()
{
        FILE *fd;
        int elementi,n;

        fd = fopen(nomefile,"rb+");
 // apertura lettura/scrittura/binario - se esiste
        if ( fd == 0)  // file NON esiste
             fd = fopen("numeri.dat","wb+");
 // apertura scrittura/lettura/binario
        else // il file esisteva
             fseek(fd,0L,2); // posiziono in fondo

 printf("Quanti numeri scrivo nel file binario?");
        scanf("%d",&elementi);
        if (elementi > 0)
                printf("Digita i valori\n");
        while(elementi--)
        {
                scanf("%d",&n);
                fwrite(&n,sizeof(int),1,fd);
        } // Nota che NON chiudo il file!!!!

 printf("Quanti elementi vuoi ricercare ?");
        scanf("%d",&elementi);
// Continua ...
```

```
        while(elementi--)
        {
         printf("num. da ricercare in “);
         printf(posizione(0,1,2,...)?");
         scanf("%d",&n);

         if (fseek(fd,(long)n*sizeof(int),0) == 0)
            if(fread(&n,sizeof(int),1,fd))
                printf(" Numero = %d\n",n);
        }
        fclose(fd);
}
```

NB: il terzo parametro di fseek puo' essere:

0 (da inizio file)
1 (da pos. corrente)
2 (da fine file)

VETTORI DI STRUTTURE

```
#include <stdio.h>
#include "cliente.rec"

/* struct cliente definita nel file
   "cliente.rec". Identica a quella di alcune
   pagine indietro */

struct ditta {
                long codice;
                char ragsoc[64];
                struct cliente cli;
                /*struttura dentro struttura */
                /* sarebbe meglio un
                   vettore struct cli[50];*/
             };

#define MAX 100

struct ditta ad[MAX];
/* vettore di record o tabella */

void prova_ditta()
{
        int i;
        for (i = 0; i < MAX; i++)
        {
                ad[i].codice     = 0;
                ad[i].cli.codice = 0;
        }
}
```

Nb: per un vettore dinamico, definire un vettore di puntatori a struttura e allocare spazio con la malloc();

PUNTATORE void *(Approfondimento)*

```c
#include <stdio.h>
#include <string.h>    /* per strlen */

void cpybuf(void *a, void *b, unsigned size)
{   /* questa funzione copia qualunque cosa */
    register char *d, *s;
    for ( d = a, s = b; size--; *d++=*s++)
        ;
}

int main()  // collaudo della funzione cpybuf
{
    float f1,f2;
    int  vi1[10], vi2[10];
    long vl1[10], vl2[10];
    char st1[11], st2[11];
    int i;
    f1 = 27.45;
    cpybuf(&f2,&f1,sizeof(float));
    printf(" %lf\n",f2);
    strcpy(st1,"Arcobaleno");
    cpybuf(st2,st1,strlen(st1)+1);
    printf("%s\n",st2);
    for ( i = 0; i < 10; i++)
    {
        vi1[i] = i;
        vl1[i] = 10 *i;
    }
    cpybuf(vi2,vi1,sizeof(vi2));
    cpybuf(vl2,vl1, sizeof(vl2));
    for ( i = 0; i < 10; i++)
        printf(" %d %ld\n",vi2[i],vl2[i]);
    return 0;
}
```

Allocazione dinamica della memoria. Le funzioni malloc() e free()

```c
#include <stdio.h>   /* per printf */
#include <string.h>  /* per strlen e strcpy */
#include <stdlib.h>  /*per malloc/free/exit */

Problema: eseguire una copia di qualunque stringa
creando nuova memoria...

char *strsave(char *s)
{
   char *p;

   if ((p = malloc(strlen(s)+1)) == 0)
    {
                printf("Spiacente, esaurita");
                printf(" memoria nello heap!");
                exit(1);
    }
    return strcpy(p,s);
}
```

```c
/* Esempio: */

     char *p = strsave("ciao");
     .......  /* utilizzo p */
     free(p); /* libero mem.*/
```

ALLOCARE E LIBERARE SPAZIO PER CONTENERE QUALUNQUE OGGETTO:

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

int main()
{
        int    *pi;  /* per array di interi */
        double *pd;  /* per array di double */
        char   *s;   /* per stringa */

        int  n_int, n_double, n_char;
        printf(" Quanti interi ? ");
        scanf("%d",&n_int);
        printf(" Quanti double ? ");
        scanf("%d",&n_double);
        printf(" Quanti char ? ");
        scanf("%d",&n_char);

        pi = malloc(n_int    * sizeof(int));
        pd = malloc(n_double * sizeof(double));
        s  = malloc(n_char +1); // 1 char == 1 byte
                                // +1 per terminatore

/* da qui in poi ho 3 vettori: pi[], pd[], s[]*/

        /* ......... utilizzo i vettori ....*/
        /* ......... e poi ................*/

        free(pi);    /* libero memoria      */
        free(pd);    /* quando i vettori    */
        free( s);    /* non mi servono piu' */

}
```

parametri in ingresso al main(): variabili d'ambiente *(Approfondimento)*

Costruiamo un programma che simuli
il comando Unix/linux "echo" :

```
#include <stdio.h>

void main(int ac, char *av[], char *ev[] )
/* ac e av : parametri in ingresso al main
   QUESITO: perchè va bene anche "char **av"  ?*/
{
    int i;
    for ( i = 1; i < ac; i++)
      printf("%s%c",av[i], i < ac-1 ? ' ' : '\n');
}
```

Esercizi:

1 Compattare il suddetto programma
2 Provare a stampare **char** *ev[] (ambiente)
3 Copiare av in un altro array di
  puntatori a **char**, allocando memoria
4 Costruire una funzione che ordina
  un vettore di puntatori a **char**
5 Costruire la funzione:
  void free_a(**char** *v[])
  che libera la memoria allocata per
  il vettore di puntatori v

Puntatori a struttura e vettori dinamici *(Approfondimento)*

Rivediamo prima due semplici funzioni per l'input di stringhe e di numeri:

```c
#include <stdio.h>
#include <stdlib.h>   /* per atof() */

int getline(char s[], int lim)
{
    int c, i=0;
    while(--lim > 0 &&
          (c=getchar())!= EOF &&
           c != '\n')
               s[i++] = c;
    s[i] = '\0';
    return(i); /* lunghezza stringa */
}

/* getnum() va bene per qualunque numero */

double getnum()
{
    char w[80+1];
    getline(w,80);
    return atof(w);
}
```

NB: per un numero diverso da **double** usare un cast;

```c
es: int n = (int) getnum();
```

Ed ora vettori semidinamici di struct!
(Approfondimento)

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

int getline(char *, int);
double getnum(void);  // non codificate ...
// vedi pagina precedente per getline() e getnum()

#define MAX 1024

struct app {
                char            codice[5+1];
                int             metri;
                unsigned long   prezzo;
            };
struct app *pa[MAX];
/* vettore di pointer a struttura */


void zero(void *a, unsigned size)
/* azzera qualunque cosa */
{
        register char *d = a;
        while(size--)
                *d++ = 0;
}
/* continua...*/
```

```c
void alloca_app(struct app *pa[], unsigned elem)
{
        int i;
        for ( i = 0; i < elem; i++)
        {
              pa[i] = malloc(sizeof(struct app));
              if (!pa[i])
              {
                        fprintf(stderr,
                        "Finita memoria");
                        exit(1);
              }
        }
        pa[elem] = NULL;
        // idea simile al fine stringa
        // cioe' metto un NULL invece che un '\0'
}

void azzera_app(struct app *ap[])
{
        int i;
        for (i = 0; ap[i]; i++)
                zero(ap[i],sizeof(struct app));
}

void get_app(struct app *p)
{
    printf("Immetti cod., mt. e pr.:\n");
    getline(p->codice,5+1);
    p->metri  = getnum();
    p->prezzo = getnum();
}

void print_app(struct app *p)
{
    printf("%s %d %d\n",p->codice,
                p->metri,p->prezzo);
}
```

```c
void libera_app(struct app **ap)
{
        for ( ; *ap ; *ap++= 0)
                        // l'ultimo pointer e' a NULL
                free(*ap);
}


int main()
{
  unsigned n,i;
  printf("Quante strutture [MAX %d]",MAX-1);
  n = getnum();
  alloca_app(pa,n);
  azzera_app(pa);
  for ( i = 0; i < n; i++)
        get_app(pa[i]);
  for ( i = 0; pa[i]; i++)
        print_app(pa[i]);
  libera_app(pa);
  return 0;
}
```

NB: la funzione azzera_app e' superflua in questo programma;

Esercizi interessanti:

1 Inserire una funzione di ordinamento del vettore; - che vantaggio c'e' rispetto ad un normale vettore di **struct**?

2 Ordinare secondo criteri diversi usando la qsort() di sistema

```c
/* I/O di strutture: file di record */
/* ancora utilizzo del doppio puntatore */
(Approfondimento)
#include <stdio.h>
#include <stdlib.h>

struct libro {
                char titolo [30];
                long prezzo;
                char autore [20];
                char casa_ed[15];
            };

void    getline(char *, int);
double  getnum(void);

void fatalerror(char msg[])
{
    fprintf(stderr,"ERRORE: %s\n",msg);
    exit(1);
}

void get_libro(struct libro *l)
// legge da tastiera un libro
{

    printf(" Titolo ---->");
    getline(l->titolo,sizeof(l->titolo));
    printf(" Prezzo ---->");
    l->prezzo = getnum();
    printf(" Autore ---->");
    getline(l->autore,sizeof(l->autore));
    printf(" Casa ed. -->");
    getline(l->casa_ed,sizeof(l->casa_ed));
}
```

```
void put_libro(struct libro l)
{
    //scrive sullo schermo un libro
   printf(" Titolo ---->%s\n",   l.titolo);
   printf(" Prezzo ---->%10ld\n",l.prezzo);
   printf(" Autore ---->%s\n",   l.autore);
   printf(" Casa ed. -->%s\n",   l.casa_ed);
}

void crea_file(FILE **f, struct libro *l)
{
        int elementi;

        if (!(int)(*f =fopen("libro.dat","wb")) )
                fatalerror("scrittura");
        printf("Quanti libri ? ");
        elementi = getnum();
        while(elementi--)
        {
          get_libro(l);
          fwrite(l,sizeof(struct libro),1,*f);
        }
        fclose(*f);
}


// da completare, scrivete voi il main() !!!
```

Puntatori a funzione:

(Approfondimento)

```
#include <stdio.h>

int  (*pf)     (int);
/* puntatore a funz. che riceve un int e ritorna un int */

int  (*vpf[5])(void);
/* vettore di 5 punt. a funz. che ritornano un int*/

int zero(int a) // funzione di prova...
{  printf("\n[%d]\n",a); return a; }

int uno()   // funzione di prova...
{ printf("Uno\n"); return 1; }

int due()
{ printf("Due\n"); return 2; }

void main()
{
        int s;
        pf      = zero;
        // pf punta alla funz. zero
        vpf[0] = 0;      // vpf[0] = NULL
        vpf[1] = uno;
        vpf[2] = vpf[3] = vpf[4] = due;
        printf("Scelta [0..4] -->");
        // menu' rapido
        scanf("%d",&s);
        if ( s > 0 && s < 5)
                (*vpf[s])(); // richiamo funzione
        else
                 (*pf)(s);
}
```

Uso di typedef (definizione di nuovi tipi)

```c
#include <stdio.h>

struct prova {
                int a;
                char b[20];
                  };

typedef  struct prova PROVA;
typedef int (*PFI) ();

int messaggio(PROVA *p)
{
        printf("%d %s\n",p->a,p->b);
        return(0);
}

void main()
{
        PROVA p;
        PFI   pf = messaggio;
        p.a = 33;
        strcpy(p.b,"uno due tre");
        (*pf)(&p);
}
```

Unioni:
strutture ove i membri sono allocati al primo byte

```c
#include <stdio.h>

struct uno { char *p;    long q; };

struct due { int a[10]; char c; };

typedef struct uno UNO;

typedef struct due DUE;

union tre    {
                UNO u;
                DUE d;
             };

typedef union tre TRE;

void main()
{
     TRE t;
     int i;

     t.u.p = "CIAO";
     t.u.q = 100000L;
     printf("%s %ld\n",t.u.p,t.u.q);
     for (i = 0; i < 10; i++)
                          t.d.a[i] = 0;
     t.d.c = '\0';
     printf("%s %ld %d\n",
     t.u.p,t.u.q, sizeof(TRE));
}
```

Strutture dati avanzate

pila gestita con lista linkata:

(Approfondimento)

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

struct lista {
                int numero;
                struct lista *prossimo;
                };

struct lista *testa = 0;
int num;

void push(int num)
{
        struct lista *appoggio =
        malloc(sizeof(struct lista));
        if (appoggio != 0)
        {
                appoggio->numero   = num;
                appoggio->prossimo = testa;
                testa              = appoggio;
        }
        else
                printf("Finita memoria\n");
}

/* Continua...*/
```

```c
int pop()
{
        int ritorno = 0;
        struct lista *appoggio = testa;

        if (testa == 0)
        {
                printf("Pila vuota!\n");
                exit(0);
        }
        else
        {
                ritorno = testa->numero;
                testa   = testa->prossimo;
                free(appoggio);
        }
        return ritorno;
}
```

Esercizi:

1 Completare il programma con il main()

2 Scrivere un programma che gestisca
una coda con lista linkata

3 Aggiungere ai programmi 1 e 2:

-memorizzazione della lista su file
-caricamento di un file in una lista
-costruzione di una lista ordinata
-caricamento da file in lista e modifica della lista in un array di puntatori.

Strutture dati avanzate - Albero binario di ricerca *(Approfondimento)*

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

struct albero {
                char *parola;
                struct albero *sinistra, *destra;
                int num;
                };
typedef struct albero ALBERO;

ALBERO *radice = 0;
char parola[80+2];

ALBERO *crea(ALBERO *rad,char *parola)
{
   if (rad == 0 )
   {
     rad = malloc(sizeof(ALBERO));
     rad->parola = malloc(strlen(parola)+1);
     // aggiungi qui controllo memoria heap esaurita
     strcpy(rad->parola,parola);
     rad->sinistra = rad->destra = 0;
     rad->num = 1;
   }
   else
   {
     int n;
     if((n = strcmp(rad->parola,parola))> 0)
        rad->sinistra = crea(rad->sinistra,parola);
     else if ( n < 0)
        rad->destra   =crea(rad->destra,  parola);
     else /* parole uguali */
        rad->num++;
   }
   return rad;
}
```

```
void visita(struct albero *radice)
{
        if ( radice )
        {
                visita(radice->sinistra);

                printf("[%-26s]--> %3d volta(e)\n",
                radice->parola,radice->num);

                visita(radice->destra);
        }
}

int main()
{
        while(scanf("%s",parola) != EOF)
                radice = crea(radice,parola);
        visita(radice);
        return 0;
}
```

Buona programmazione C a tutti!!!