IL LINGUAGGIO C

Prof. Fischetti Pietro

Sviluppato tra il 1969 e il 1973 all'interno dei laboratori di ricerca dell'AT&T da Dennis Ritchie e Kevin Thompson

Dennis M. Ritchie (standing) and Ken L. Thompson (seated), inventors of UNIX, at Bell Labs in front of a DEC PDP-11 computer, ca 1970. Courtesy, Computer History Museum

il linguaggio C e' il pilastro dell'informatica moderna: i sistemi operativi Linux e Windows sono scritti in C, i software di sistema,i driver dei dispositivi, i socket di Internet, il kernel dei telefonini, i motori grafici piu' potenti sono scritti in C/C++.Il C e' un linguaggio compilato dove cioe' il codice eseguibile prodotto interagisce direttamente sistema operativo per funzionare, a differenza dei linguaggi interpretati (C#, Java, javascript, Python …) che hanno bisogno di un "interprete" (scritto solitamente in C dall'azienda proprietaria) che si interpone fra il codice e il Sistema Operativo. Generalmente i linguaggi compilati sono piu' difficili da comprendere ma piu' veloci e 'leggeri' mentre quelli interpretati piu' facili da imparare ma piu' costosi (in termine di consumo di risorse) e lenti (i comandi devono essere interpretati prima di arrivare al S.O.)

Costrutti e definizioni.
Il C e' un linguaggio a blocchi, dove per blocco si intende un'insieme, anche vuoto cioe'privo di istruzioni, di definizioni e/o comandi racchiusi da parentesi graffe:

```
{

}
```

Un blocco che ha un nome, che puo' o meno ricevere parametri e che puo' o meno restituire al massimo un solo valore viene chiamato 'funzione', ad es.

```
int somma(int a, int b)
{

        return a+b;

}
```

Nel caso di un programma eseguibile (un .exe sotto Windows per intenderci) e' obbligatoria la presenza di una funzione che deve avere il nome speciale: main.

Deve essere esattamente scritta cosi' in quanto il C e' un linguaggio case-sensitive. Questo perche' quando lanciamo un programma il S.O. ne verifica la presenza all'interno dell'eseguibile e se la trova crea la classica finestra nera del DOS (CUI) e avvia il programma. Esiste un'altra funzione specialesolo in Windows, la WinMain che nel caso il S.O. Windows la trovi al posto della main permette la crezione di finestre grafiche (GUI, esempio notepad.exe, calc.exe). Quindi un eseguibile deve avere obbligatoriamente definito al suo interno la funzione main (o WinMain nel caso di Windows), mentre altre tipologie di applicazioni in C (esempio le librerie .lib, .dll) no. Quindi il + semplice programma in C e':

```
main()
{

}
```

E' un programma perfettamente valido che una volta compilato ed eseguito non fa assolutamente nulla (ma consuma poco o niente), e puo' essere utile per sapere se il compilatore e' stato installato correttamente, per leggere la dimensione del codice prodotto e per farsi un'idea di quanta memoria consumera' rispetto ad un programma interpretato anch'egli vuoto. Il C e' stato creato 'sottile' cioe' "se non uso non consumo". Permette solo semplici operazioni aritmetiche fondamentali o poco piu', tutto il resto deve essere richiesto ('incluso'); anche la semplice scrittura a video e/o lettura da tastiera (infatti esistono programmi in C che non devono leggere/scrivere a video vedi ad. es. i servizi/daemon del S.O.)

Il C non e' un linguaggio ad oggetti (a tale scopo e' stato creato nel 1980 il C++ che estende il C), non e' multipiattaforma, il codice deve essere compilato sul S.O. in cui verra' eseguito. Il C e' un linguaggio tipizzato cioe' ogni variabile utilizzata deve avere un tipo conosciuto dal compilatore: int,short,long per i numeri interi senza virgola, float e double per trattare numeri con la virgola, etc.. I commenti il cui contenuto viene completamente ignorato dal compilatore, sulla singola linea si indicano con //, es:

```
//questo e' un commento
```

mentre i commenti che contengono piu' linee con:

```
/*
Questo
e' un
commento
*/
```

Esempio – Calcolo dell'area del cerchio di raggio r=2

```
A=3.141516*r*r=3.141516*2*2

main()
{
	float A; //l'area di un cerchio puo' contenere numeri con la virgola

	A = 3.141516*2*2;//in C il formato dei numeri segue la cultura Inglese
}
```

In questo esempio se si vuole leggere da tastiera il raggio, il C da solo non permette di farlo occorre includere una libreria (standard in quanto gia' presente con il compilatore e non occorre cercarla) che si occupa dell'input/output da tastiera con nome stdio.h. Il tutto avviene indicando tramite la direttiva al compilatore '#include' e il nome del file stdio.h (che indica l'elenco delle funzioni per l'IO presenti nella libreria).

APPROFONDIMENTO – i File .h

i file .h indicati nella direttiva #include sono dei semplici file di testo che contengono la definizione delle funzioni da utilizzare cioe' il nome, gli eventuali parametri da passare e il valore di ritorno. Il codice vero e proprio che verra' utilizzato si trova nei file di libreria che sono dei file binari. Le librerie sono di due tipi: statiche e dinamiche. Le librerie statiche sotto windows hanno estensione.LIB mentre quelle dinamiche hanno estensione .DLL.Quelle statiche (piu' facili da creare e utilizzare) si distinguono per essere inglobate (linkate) staticamente nel file eseguibile risultante dalla compilazione, questo produrra' un file eseguibile piu' grosso ma piu' sicuro rispetto alle modifiche, mentre le librerie dinamiche forniscono funzioni che possono essere chiamate dinamicamente al volo durante l'esecuzione del programma, avro' cosi' programmi + piccoli ma sensibili alle variazioni del codice e alla presenza o meno della libreria, si puo' fare l'analogia con le pagine Web che possono contenere immagini statiche o dinamiche raggiungibili tramite link. Come esempio in Windows per l'elaborazione dei suoni (midi, wav) esiste la libreria Winmm.lib fornita con il compilatore e Winmm.dll presente in c:\windows\system32, il file include da utilizzare ha nome:Mmsystem.h,queste informazioni si possono trovare nell'helponline di Microsoft.

Si possono utilizzare le parentesi angolari come in <stdio.h> o i doppi apici “stdio.h”, nel primo caso viene cercato il file stdio.h nel percorso INCLUDE specificato nelle variabili d’ambiente (solitamente creata dall’installazione del compilatore), mentre nel secondo caso il file stdio.h viene cercato prima nella directory corrente e se non trovato nelle directory d’ambiente, la relativa libreria viene cercata nei percorsi contenuti nella variabile d’ambiente LIB, mentre la variabile d’ambiente PATH indica il percorso dove il sistema cerchera’ di lanciare il compilatore.

LA FUNZIONE SCANF

La funzione standard del C incaricata di leggere i dati generalmente inseriti da tastiera e dati in pasto al programma e’ la scanf. Deve essere utilizzata con attenzione, dato che e’ facilmente fonte di errori se non utilizzata correttamente cioe’ il programma potrebbe lavorare con dati non corretti. Vediamo quindi la sintassi:

```
#include <stdio.h>

int scanf(const char *format, ...);
```

dato che e’ una funzione dello standard I/O occorre specificare lo <stdio.h>. Quindi occorre specificare il tipo di dati che deve leggere secondo uno specificatore di formato (analoghi a quelli della printf):
ad esempio:
%d (interi)
%f (numeri decimali)
%c (caratteri)
%s (stringhe di caratteri)
Quindi il programmatore deve sapere esattamente che tipo di dato si aspetta. Ad esempio se voglio leggere il raggio di un cerchio (un numero decimale) e scriverlo nella variabile r, avro’:

```
scanf(“%s”,&r);
```

N.B. Notare il carattere speciale & che indica a scanf di leggere da tastiera e scrivere nella variabile di nome r. (per scrivere a video il valore di r utilizzero’ la funzione printf:

```
printf(“%f”,r);
```

in questo caso la funzione printf legge (notare l’assenza del carattere &) il valore presente nella variabile r e a scrivera’ a video,
E’ importante sapere come ha ‘lavorato’ la funzione scanf analizzandone il valore di ritorno, che indica il numero di variabili che e’ stato assegnato, oppure 0 se non ha potuto effettuarne l’asse per dati non previsti, oppure EOF (che indica il valore -1) se si e’ incontrata la fine dell’input, comunque ritorna un valore intero. Vediamo subito un esempio: Abbiamo visto precedentemente come leggere un numero decimale ed assegnarlo ad una variabile, ma cosa succede se l’utente del nostro programma inserisce per sbaglio un carattere alfabetico al posto di un numero, il programma deve fare calcoli (ad esempio l’area di un cerchio di raggio r dato) su dati che non sono numeri? Ovviamente no. Esempio:

```
if(scanf(“%f”,&r)==1){//oppure si puo’ scrivere if(scanf(“%f”,&r)>0){
    //OK calcola
}else{
    printf(“dato non valido”);
}
```

In questo caso (se il valore di ritorno di scanf e’ maggiore di zero) calcolo solo se ho un numero decimale altrimenti scrivo ‘dato non valido’ nel caso in cui scanf ritorni un valore minore o uguale a zero. Vediamo un altro esempio. Supponiamo che debba sviluppare un programma ‘calcolatrice’ che svolga dei semplici calcoli aritmetici. Cioe’ vogliamo che se l’utente scrive 3+2 si ottiene 5, oppure 7.8-0.3 si ottenga 7.5. Il programma deve quindi leggere 3 dati: Numero1OperatoreNumero2, quindi avro’:

```
char op;
float n1,n2;
scanf(“%f%c%f”,&n1,&op,&n2);
```

Quindi devo avere esattamente 3 dati inseriti correttamente: un numero decimale seguito da un carattere a sua volta seguito da un numero decimale. Analizziamo il valore ritornato da scanf, dopo averlo salvato in una variabile,in questo esempio:

```
char op;
float n1,n2;
int rs=0;
rs=scanf(“%f%c%f”,&n1,&op,&n2);
printf(“%d”,rs);
```

I:4.5+2.1
O:3

I:52+A
O:2

Ho indicato con I l'input da tastiera, con O l'output su schermo. Quindi se voglio in questo esempio essere sicuro dei dati inseriti scrivero':

```
char op;
float n1,n2;
int rs=0;
if(scanf("%f%c%f",&n1,&op,&n2)==3){//NON si puo' scrivere if(scanf("%f%c%f",&n1,&op,&n2)>0){ Perche'?
    //OK calcola
}else{
    printf("dato non valido");
}
```

Il caso del valore EOF si vedra' nei cicli While

APPROFONDIMENTO – scanf e le Stringhe

Attenzione ad usare scanf con le stringhe perche' ad esempio se ho il seguente frammento::

```
…
char buff[1024];
scanf/("%s",buff);
….
```

se da tastiera scrivo

```
uno due
```

nell'array buff ci vanno solo i caratteri fino al primo spazio, cioe' buff conterra':uno
come riportato in: https://cplusplus.com/reference/cstdio/scanf/:

specifier	**Description**	**Characters extracted**
`s`	String of characters	Any number of non-whitespace characters, stopping at the first whitespace character found. A terminating null character is automatically added at the end of the stored sequence.

Per leggere tutta la linea spazi compresi fino al ritorno a capo si utilizza la funzione gets(),

```
#include <stdio.h>

int main(int argc, char **argv){
char buff[1024]={0};
while(gets(buff)){
        printf("%s\n",buff);
}
return 0;
```

Oppure dato che la funzione e' considerata pericolosa (causa attacchi buffer overflow) si utilizza la fgets facendo attenzione che il '\n' va a finire nel buffer, come si trova in
https://cplusplus.com/reference/cstdio/fgets/ :

Get string from stream

Reads characters from *stream* and stores them as a C string into *str* until (*num*-1) characters have been read or either a newline or the *end-of-file* is reached, whichever happens first.

A newline character makes `fgets` stop reading, but it is considered a valid character by the function and included in the string copied to *str*.

A terminating null character is automatically appended after the characters copied to *str*.

Notice that `fgets` is quite different from gets: not only `fgets` accepts a *stream* argument, but also allows to specify the maximum size of *str* and includes in the string any ending newline character.

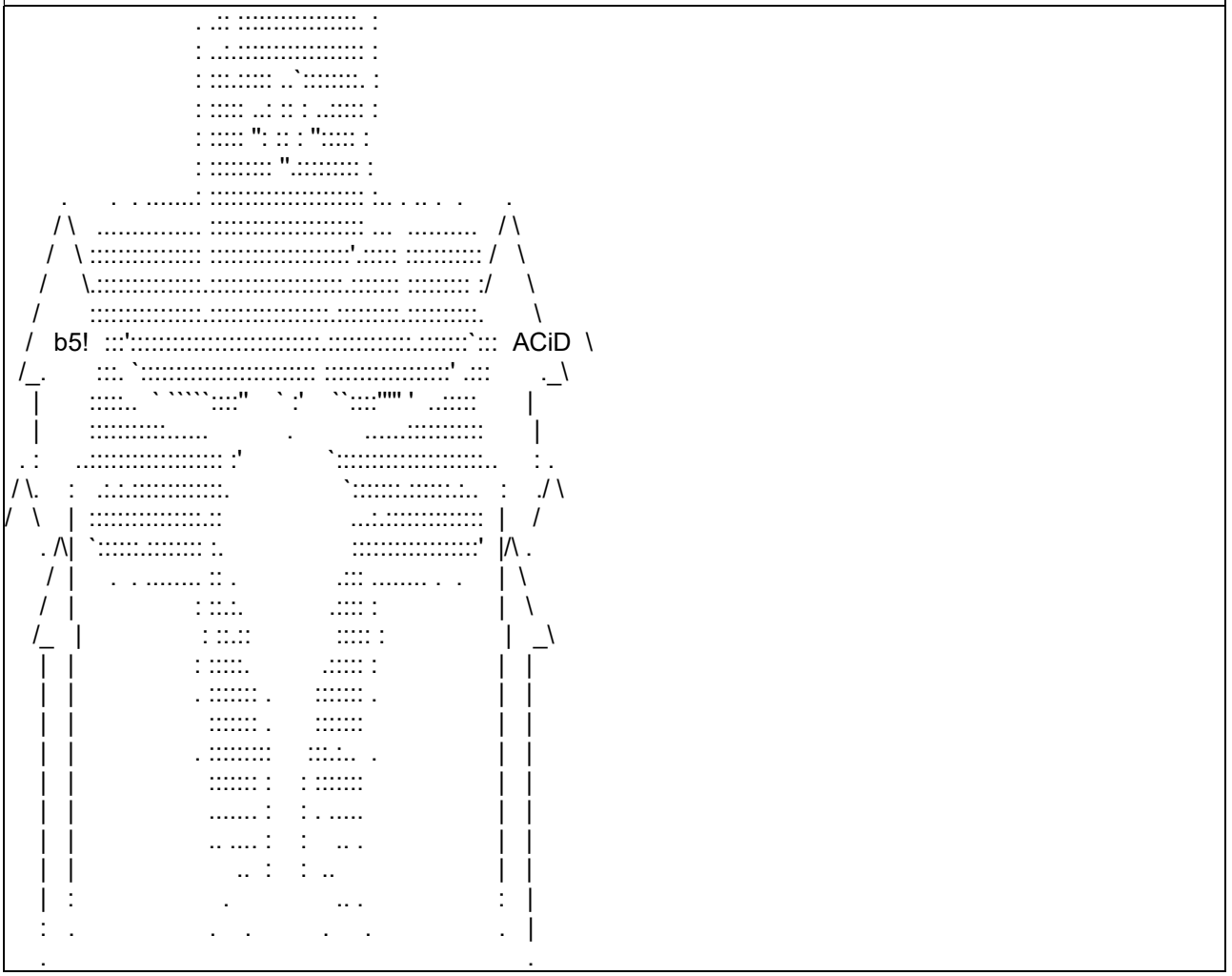
```
#include <stdio.h>
```

```
int main(int argc, char **argv){
char buff[1024]={0};
while(fgets (buff, sizeof(buff),stdin) != NULL){
        buff[strlen(buff)-1]=0; //rimuovo l'eventuale ritorno a capo
        printf("%s\n",buff);
}
return 0;
}
```

oppure la meno conosciuta tecnica *scanset*
https://www.geeksforgeeks.org/scansets-in-c/
https://www.delftstack.com/howto/c/c-scanf-string-with-spaces/
In pratica se voglio leggere l'intera linea con la tecnica scanset::

```
#include <stdio.h>
int main(int argc, char **argv){
        char buff[1024]={0};
        int r;
        while((r=scanf("%[^\n]%*c",buff))>=0){
                if(r==0){
                        getchar();
                        putchar('\n');
                }
                else
                         printf("%s\n",buff);
        }
        return 0;
}
```

```
C:\ >art.exe < cdc.txt
```



N.B !!!!!!!! Attenzione a mescolare scanf con getchar:
nel seguente frammento di codice:

…
Int a,c;
scanf(“%d”,&a);
c=getchar();
printf(“bye:%d”,c);
….

dopo aver inserito un numero il programma scrive subito ‘bye’, non aspetta il secondo input da tastiera con getchar().
Questo e’ spiegato dal comportamento di scanf:
ad esempio sul sito: https://www.ibm.com/docs/en/i/7.4?topic=functions-scanf-read-data
si dice che:
…Characters outside of format specifications are expected to match the sequence of characters in `stdin`*; the matched characters in* `stdin` *are scanned but not stored…*
Cioe’ cio’ che non e’ previsto dal formato nella scanf in questo caso int, viene lasciato nel buffer dello stdin.
Quindi se do 5 e invio da tastiera nel buffer dello stdin rimane il ritorno a capo, inoltre se rieseguo l’esempio dando da tastiera 5+ nel buffer resta + e viene restituito da getchar().

APPROFONDIMENTO – I/O BUFFERIZZATO E NON.
scanf utilizza un buffer interno (cioe’ un array di caratteri) per attendere che vanga premuto il tasto Invio prima di consegnare i dati al programma, questo permette di tornare indietro con la frecce se si vuol modificare cio’ che si e’ scritto. Ma esiste anche l’input da tastiera non bufferizzato utilizzando la funzione getc() presente in <conio.h> che non attende l’invio come nella scanf; e’ utilizzata ad esempio nei programmi che chiedono ‘Premere un tasto per continuare . . .’, oppure nei videogiochi per far muovere un personaggio senza che si debba premere invio ogni volta che premo un tasto di movimento.
Vediamo un esempio:

T,C
```
#include <stdio.h>
int main(){
   char c;
   setbuf(stdin,NULL);
   setbuf(stdout,NULL);
   while(scanf("%c",&c)>0){
        printf("%c",c);
   }
return 0;
}
```
R,C
```
#include <stdio.h>
int main(){
   char c;
   while(scanf("%c",&c)>0){
     if(c!='\n')
          printf("R:%c\n",c);
    }
return 0;
}
```

Colleghiamo i due programmi tramite pipe della shell d evrifichiamo cosa succede commentando o meno le due righe che contengono detbuf in T.C:
C:\>T | R

ESEMPIO – Calcolo dell’area di un cerchio di raggio r.
```
#include <stdio.h>   //devo leggere/scrivere
main()


{


          float r;
```

```
        float A;

        scanf("%f",&r);//leggo un numero da tastiera e lo 'memorizzo' nella variabile r
         A = 3.141516*r*r;
}
```

Si vuole stampare il valore dell'area calcolata:

```
#include <stdio.h>

main()

{

        float r;

        float A;

        scanf("%f",&r);

        A = 3.141516*r*r;

        printf("Area=%f",A);//scrivo a video il risultato

}
```

Il segnaposto %f indica la formattazione utilizzata nell'input/output in questo caso sto trattando numeri con la virgola a precisione singola (6 cifre dopo la virgola), nel caso di numeri interi:%d, nel caso di caratteri:%c.

Tornando all'esempio non si vuole calcolare l'area di un cerchio con raggio negativo, quindi devo dire: calcola l'area solo se il raggio che ho inserito da tastiera e' un numero positivo, occorre usare il costrutto if:

```
if (condizione_vera)
{
        //blocco eseguito se la condizione e' vera
}
else
{
        //blocco eventuale eseguito se la condizione e' false
}
```

Nel caso in cui un blocco vero/falso contenga una sola istruzione le parentesi possono essere omesse.

```
#include <stdio.h>
main()
{
        float r;
        float A; //si puo' anche definire tutto su una sola riga: float r,A;
        scanf("%f",&r); //la & indica che il numero letto da tastiera deve essere SCRITTO nella variabile r
        if (r>=0) //condizione
        {
                A = 3.141516*r*r;
                printf("Area=%f",A);//scrivo a video il risultato LETTO dalla variabile A (non ci va il &)
        }
        else
        printf("Il raggio non puo essere negativo\n");//\n indica ritorno a capo
}
```

E' molto importante sottolineare che il C non e' tenuto (sempre nell'ottica delle performance) a differenza di altri linguaggi ad inizializzare una variabile quando viene definita, ad esempio:

int a; // a puo' assumere un valore iniziale casuale.

Vediamo un semplice esempio:

```
#include <stdio.h>
int main(){
 int a;
 float f;
 char s[128];
 printf("%d          %f  %s",a,f,s);
 return 0;
}
```

Compilato con MS Visual C produce:

```
0      0.000000       á☼
```

Mentre con Dev-C++:

```
1       0.000000  p @
```

Si nota chiaramente la presenza di dati casuali non definiti nel codice sorgente.
Quindi al fine di evitare errori difficili da scovare, in C inzializzare sempre una variabile prima di utilizzarla, in definitiva avremo:
Esempio areaCerchio.c

```
#include <stdio.h>
int main()
{
float r=0.0;
        float A=0.0;        //si puo' anche definire tutto su una sola riga: float r,A;
        scanf("%f",&r); //la & indica che il numero letto da tastiera deve essere SCRITTO nella variabile r
        if (r>=0) //condizione di validazione
        {
                A = 3.141516*r*r;
                printf("Area=%f",A);//scrivo a video il risultato LETTO dalla variabile A (non ci va il &)
        }
        else
        {
                printf("Il raggio non puo essere negativo\n");//\n e' il ritorno a capo
                return 1;
        }
        return 0;
}
```

N.B. In C si utilizza il termine stream (flusso) di input o di output che e' piu' generico invece di dati letti da tastiera o inviati allo schermo in quanto i dati possono essere letti e/o scritti su dispositivi quali ad esempio HD, stampante, rete, etc.

APPROFONDIMENTO – il return 0; nel main

Ho cambiato leggermente il main per indicare che in alcune situazioni e' importante che il sistema conosca come e' terminato il programma. La convenzione consolidata da decenni dice che un programma nel caso in cui termini correttamente debba restituire il valore intero 0, altrimenti un qualsiasi intero diverso da zero, questo nel caso in cui si voglia automatizzare tramite script l'esecuzione di piu' programmi, ad es. si voglia eseguire un certo programma solo nel caso in cui un programma precedente sia andato a buon fine, es un programma acquisisce una sequenza di numeri e un secondo programma ne calcola la media, si vuole ovviamente che il calcolo della mediacioe' l'avvio del secondo programma avvenga solo se il primo programma di raccolta dati non sia andato in errore .

Architettura

Puo' essere utile sapere il tipo di Endian che si ha sulla propria macchina

```
#include <stdio.h>
int main(){
```

```
int num = 1;
if(*(char *)&num == 1)
   printf("\nLittle-Endian\n");
else
   printf("Big-Endian\n");
return 0;
```

DIFFERENZA IMPORTANTE WINDOWS-LINUX

E'importante sapere che Windows gestisce i ritorni a capo in un file di testo con due caratteri (0x0D 0x0A) mentre Linux ne usa solo uno (0x0A), questo puo' essere considerato con attenziona ad esempio se si trasferiscono file di dati Tra Windows e Linux. Vediamo di seguito un esempio

```
********** NEW-LINE IN WINDOWS E LINUX
Creiamo un semplice file di testo con due parole separate da ritorno a capo:
C:\>copy con b.txt
uno
due
^Z
       1 file copiati.

C:\>type b.txt
uno
due

con powershell vediamo all'interno dei file:
C:\>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.

PS C:\> format-hex b.txt


         Percorso: C:\b.txt

         00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   75 6E 6F 0D 0A 64 75 65 0D 0A                uno..due..



Proviamo la stessa cosa in Linux:
studente@debian:~$ cat > /tmp/b.txt
uno
due
studente@debian:~$ cat /tmp/b.txt
uno
due
studente@debian:~$ hexdump -C /tmp/b.txt
00000000  75 6e 6f 0a 64 75 65 0a                     |uno.due.|
00000008
studente@debian:~$
```

APPROFONDIMENTO - Differenze con il C++

in C++ la definizione delle variabili puo' non essere obbligatoriamente fatta all'inizio della funzione a differenza del C, inoltre il C++ oltre alle printf/scanf introduce la cin e cout piu' semplici da utilizzare (soprattutto non si utilizza il carattere & per leggere da tastiera)

```
#include <iostream>

using namespace std;

int main()

{

          float r=0.0;
```

```
        cin >>r;

        float A=0.0;//Solo in C++ puo' essere messa in questa posizione
        if (r>=0) //condizione di validazione {


                A = 3.141516*r*r;

                cout << A;//scrivo a video il risultato LETTO dalla variabile A (non ci va il &)

        }

        else

        {


        cout << "Il raggio non puo' essere negativo" << endl;

                return 1;

        }


return 0;//

}
```

Esempio – Calcolatrice Aritmetica

Scrivere un programma in C che legge un'espressione aritmetica semplice e ne stampi a video il risultato:es:3.7+2 =5.7

```
#include <stdio.h>

int main()
{
 float a,b,r;

 char ch;

 while(scanf("%f%c%f",&a,&ch,&b)!=EOF)

 {
  switch(ch)

  {
    case '+':
        r=a+b;

        break;
    case '-':

        r=a-b;
        break;
    case '*':

        r=a*b;
        break;
    case '/':

        if (b != 0)
                r=a/b;
```

```
        else

        {
                printf("opd.2 equal 0!\n");

                continue;
        }
        break;

  default:
        printf("Invalid Operator!\n");
       continue;

 }
 printf("=%f\n",r);
}

return 0;
}
```

ESEMPIO

Scrivere un programma in C che generi un certo numero (10) di numeri casuali compresi in un certo intervallo (50, 70).

La funzione del C che genera numeri casuali e' la rand() definita in stdlib.h, che genera numeri casuali nell'intervallo 0, RAND_MAX (che vale 32767).

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
        int i;
        int y;
        for( i = 0 ; i < 10 ; i++ ) {
                y=rand();
     printf("%d\n", y);
    }
  return 0;
}
```

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
```

Il problema che se rieseguo il programma si ottiene esattamente la stessa sequenza in output.Questo perche' occorre impostare oppurtunamente il 'seme' dell'algoritmo di generazione dei numeri casuali, come riportato dall'help online (https://www.cplusplus.com/reference/cstdlib/rand/).

"This algorithm uses a seed to generate the series, which should be initialized to some distinctive value using function srand". In pratica all'inizio del programma si scrive:

```
srand(time(NULL));
```

che indica di far partire il generatore di numeri casuali in corrispondenza dell'istante di tempo di avvio del programma. (la funzione time richiede time.h). Quindi di seguito il programma con l'output di 2 esecuzioni dello stesso:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char *argv[]) {
        int i;
        int y;
```

```
        srand(time(NULL));
        for( i = 0 ; i < 10 ; i++ ) {
                y=rand();
     printf("%d\n", y);
    }
  return 0;
}
```

output:

```
11961
20998
15998
8106
19292
20953
31766
29938
20886
7219
```

Output:

```
12030
17339
30698
21915
6522
5110
31591
1233
7181
9181
```

Ora si vuole la possibilita' di generare numeri casuali in un intervallo impostato da noi. Basta utilizzare la proporzione:

$$\frac{x - x1}{x2 - x1} = \frac{y - y1}{y2 - y1}$$

Che risolta in x:

$$x = x1 + \frac{y - y1}{y2 - y1} \cdot (x2 - x1)$$

Ponendo:
Y1=0
Y2=RAND_MAX
X1=50
X2=70
Si ottiene:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char *argv[]) {
        int i;
        int y,y1,y2,x,x1,x2;
        y1=0;
        y2=RAND_MAX;
        x1=50;
        x2=70;
        srand(time(NULL));
        for( i = 0 ; i < 10 ; i++ ) {
                y=rand();
                x=x1+(y-y1)*(x2-x1)/(y2-y1);
                printf("%d\n", x);
    }
  return 0;
}
```

```
59
57
69
59
50
69
54
68
65
68
```

E se volessimo generare dei numeri casuali reali?

Programma genRnd.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char *argv[]) {
        int i;
        float y,y1,y2,x,x1,x2;
        y1=0;
        y2=RAND_MAX;
        x1=50;
        x2=70;
        srand(time(NULL));
        for( i = 0 ; i < 10 ; i++ ) {
                y=rand();
                x=x1+y-y1)*(x2-x1)/(y2-y1);
                printf("%d\n", x);
    }
  return 0;
}
```

```
60.462357
67.463303
52.596516
68.658409
66.464737
63.932922
62.295906
54.619282
66.257820
62.968536
```

Ora vogliamo visualizzare i dati generati su un grafico Excel.

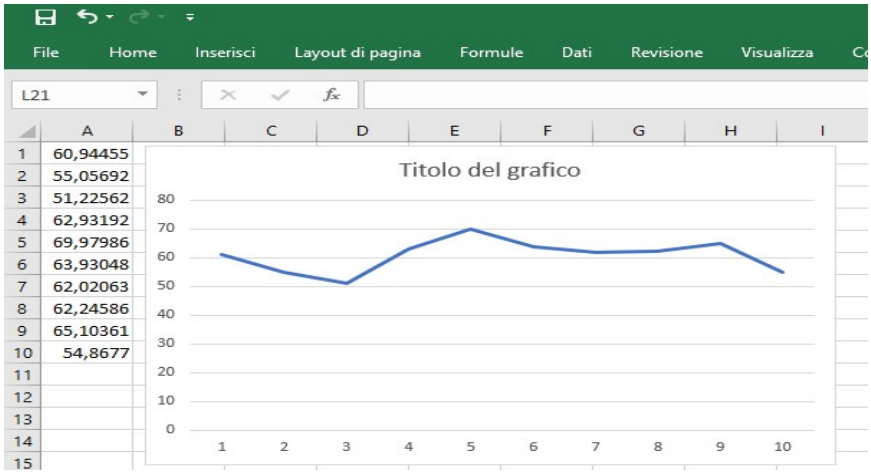
Allora compiliamo il programma precedente chiamato ad esempio myRnd.c. Il risultato della compilazione sara un eseguibile sotto Windows con nome myRnd.exe. Eseguiamolo redirezionando il risultato su un file (che chiamiamo ad esempio dati.txt).

```
C:\>myRnd.exe > dati.txt
```

Controlliamo di aver ottenuto i dati

```
C:\>type dati.txt
60.944550
55.056915
51.225624
62.931915
69.979858
63.930481
62.020630
62.245857
65.103607
54.867702
```

Apriamo con Excel il file ottenuto (ricordando che i numeri nel file sono in formato Americano (US), e che quindi occorre trasformarli se l'Excel e' installato in Italiano)



ESEMPIO
Un'azienda produttrice di coperchi vuole vedere su Excel l'andamento dell'area dei coperchi prodotti che devono avere un raggio di 30 ± 2 cm. Il reparto di produzione genera un file txt (prod.txt) contenente i raggi dei coperchi prodotti che deve essere letto dal reparto analisi che deve generare un altro file di testo (aree.txt) contenente le aree calcolate. Il programma deve in caso di errori nei dati del file prod.txt stampare il valore -1 altrimenti solo il valore dell'area calcolata
Per creare il file prod.txt utilizzeremo un programma in C che genera numeri casuali compresi tra 28 e 32. Cioe' utilizzeremo il programma dell'esempio precedente (gen.c) con limiti tra 28 e 32, e manderemo l'output su prod.txt da shell del DOS:

```
C:\>GenRnd.exe > prod.txt
```

Quindi il programma che calcola l'area del cerchio leggera' questo file e il risultato del calcolo lo inviera' al file aree.txt:

```
C:\>areaCerchio.exe < prod.txt > aree.txt
```

Ora il file aree.txt conterra' i dati numerici delle aree che potra' essere letto da Excel.

FUNZIONI

Una funzione e' un blocco di istruzioni con un nome, che puo' o meno restituire un valore e che puo' o meno accettare un certo numero di parametri (racchiusi tra parentesi tonde) passati per valore e/o riferimento.

Esempio:

funzione che calcola e restituisce la somma di 2 numeri interi:

```
int somma(int a , int b)
{
        int c=0;
        c=a+b;

        return c;
}
```

Il blocco e' tutto cio' che e' contenuto tra le parentesi graffe, il nome del blocco in questo caso e' 'somma', i parametri sono a e b passati per valore (vuol dire che viene passata una copia, un po' come dare la fotocopia della propria carta d'identita'), il tipo di ritorno e' un int in questo caso (nel caso in cui la funzione non ritorni nulla si mette return void).

Il compilatore C deve conoscere preventivamente la dichiarazione della funzione una volta che viene incontrata la chiamata. Ci sono 2 modi per farlo: 1)mettere la funzione prima del main

```
#include <stdio.h>
int somma(int a , int b)
{
```

```
	int c=0;
	c=a+b;

	return c;
}
int main()
{
	int r;
	r=somma(3,2);
}
```

2) mettere prima del main (o in un file esterno con estensione .h richiamato con la #include) la sola dichiarazione:

```
#include <stdio.h>
int somma(int a , int b);

int main()
{
	int r;
	r=somma(3,2);
}

int somma(int a , int b)

{
	int c=0;
	c=a+b;
	return c;
}
```

Notare che anche main e' una funzione, e' speciale perche' il nome deve essere esattamente scritto cosi' (in minuscolo) in quanto viene chiamata dal sistema operativo quando lancio il programma.

APPROFONDIMENTO – Significato di: int main(int argc, char *argv[])

In alcuni situazioni la main si presenta in modo diverso, cioe':

```
int main(int argc, char *argv[])
```

invece di:

```
int main()
```

perche' essendo main una funzione chiamata dal sistema operativo all'avvio del programma che la contiene, e' possibile passargli dei parametri che vengono interpretati come array di stringhe (argv[]) il cui numero e' presente nel parametro argc. Entrambi preparati opportunamente dal sistema operativo in fase di avvio del programma. Vediamo un esempio:

```
//esargc.c
#include <stdio.h>

int main(int argc, char *argv[]){
int i;
for (i=0; i < argc; i++){
printf("%s\n",argv[i]);
}
return 0;
}
```

Se lancio il programma dalla shell:

```
C:\MY\MYDEV\C>esargv.exe
argv[0]: esargv.exe
```

l'unico parametro (argc=1) e' il nome del programma. Ma se lo lanciamo con dei parametri inventati come nell'esempio seguente.

```
C:\MY\MYDEV\C>esargv ciao a tutti
```

argv[0]: esargv
argv[1]: ciao
argv[2]: a
argv[3]: tutti

attenzione agli spazi!:
C:\MY\MYDEV\C>esargv ciao a tu tti
argv[0]: esargv
argv[1]: ciao
argv[2]: a
argv[3]: tu
argv[4]: tti
se ci sono degli spazi che devono essere considerati come parte dell'argomento, si racchiudono tra virgolette:
C:\MY\MYDEV\C>esargv ciao a "tu tti"
argv[0]: esargv
argv[1]: ciao
argv[2]: a
argv[3]: tu tti
Questa tecnica si utilizza tantissimo nei programmi in C. Ad esempio esiste in DOS un programma (scritto in C da Microsoft) che permette di sapere se un certo sito e' operativo e di restituirne l'indirizzo IP e il tempo di chiamata. Questo programma si chiama ping:
Ad esempio voglio testare il sito google:
C:\ >ping www.google.it

Esecuzione di Ping www.google.it [216.58.205.67] con 32 byte di dati:
Risposta da 216.58.205.67: byte=32 durata=77ms TTL=112
Risposta da 216.58.205.67: byte=32 durata=60ms TTL=112
Risposta da 216.58.205.67: byte=32 durata=79ms TTL=112
Risposta da 216.58.205.67: byte=32 durata=71ms TTL=112

Statistiche Ping per 216.58.205.67:
  Pacchetti: Trasmessi = 4, Ricevuti = 4,
  Persi = 0 (0% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
  Minimo = 60ms, Massimo = 79ms, Medio = 71ms
E Facebook?
C:\>ping www.facebook.com

Esecuzione di Ping star-mini.c10r.facebook.com [69.171.250.35] con 32 byte di dati:
Risposta da 69.171.250.35: byte=32 durata=57ms TTL=53
Risposta da 69.171.250.35: byte=32 durata=57ms TTL=53
Risposta da 69.171.250.35: byte=32 durata=68ms TTL=53
Risposta da 69.171.250.35: byte=32 durata=54ms TTL=53

Statistiche Ping per 69.171.250.35:
  Pacchetti: Trasmessi = 4, Ricevuti = 4,
  Persi = 0 (0% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
  Minimo = 54ms, Massimo = 68ms, Medio = 59ms

Si vede chiaramente che il programma e' lo stesso (ping.exe) ma il sito da testare lo sceglie l'utente, e in base alla scelta il programma contattera' il sito richiesto..
ARRAY

Se si vogliono memorizzare dati sappiamo che dobbiamo definire delle variabili,il C ci costringe anche a specificarne il tipo a differenza di altri linguaggi come ad esempio il Python. Quindi ad esempio se voglio memorizzare la temperatura odierna definisco una variabile ad esempio intera. Pero' se voglio memorizzare le temperature settimanali avro' bisogno di definire 7 interi, per un anno avremo bisogno di definire 365 interi: int t1,t2, t3,..,t365. Piuttosto scomodo. A questo ci viene in aiuto il concetto di array.

Partiamo dalla definizione:

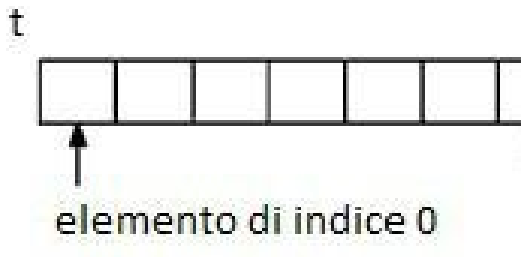
“Un array è un insieme contiguo di elementi tutti dello stesso tipo identificato da un nome e da un indice intero (che parte da 0) che ne individua l’elemento di interesse. La dimensione di questo insieme e’ fissata in fase di compilazione e non puo’ piu’ essere modificata”.
Vediamo l’esempio delle temperature settimanali senza l’uso degli array:

```
int main(){
        int t1,t2,t3,t4,t5,t6,t7;
        printf(“Temperatura Lunedi:”);
        scanf(“%d”,&t1);
        printf(“Temperatura Martedi:”);
        scanf(“%d”,&t2);
        printf(“Temperatura Mercoledi:”);
        scanf(“%d”,&t3);
        ……..
}
```

Piuttosto scomodo, ma con gli array si semplifica (per non parlare del caso annuale):

```
int main(){
int t[7];
for(i=0;i < 7; i++)
        scanf(“%d”,&t[i]);
}
```

Dichiarando int t[7] si crea un insieme statico di 7 interi come in figura:



Notare che l’indice parte da 0 ed arriva al valore massimo meno uno. E’ importantissimo sapereche il C a differenza di altri linguaggi piu’ recenti non controlla se l’indice oltrepassa i limiti. Cioe’ nell’esempio si puo’ scrivere t[9]=23 il C non protesta ma si va a scrivere in un’area di memoria sconosciuta, che puo’ provocare disastri nell’esecuzione del programma. E’ importante anche in considerazione della programmazione su microcontrollori dove la memoria e’ limitata a pochi Kb, conoscere il consumo della memoria. In C la funzione sizeof() restituisce la dimensione in bytes di un tipo o di un nome.

Il programma seguente mi dice che l’occupazione di ‘int t[7]’ e’ di 28 Bytes.

```
int main(){
        int t[7];
        printf("%d",sizeof(t));
        return 0;
}
```

Per capire perche’ dobbiamo conoscere l’occupazione in bytes di un intero:
printf("%d",sizeof(int));
Questo programma restituisce 4. Quindi ‘int t[7]’ occupa 28=sizeof(int)*7=4*7 Bytes.
APPROFONDIMENTO – sizeof dipende dal compilatore!

Riportiamo di seguito un programma che stampa la dimensione dei tipi interi:

szTypes.c

```
#include <stdio.h>
#include <math.h>
void printType(char *sType, size_t sz){
        printf("%12s: %zdByte(s)\t",sType,sz);
        printf("unsigned = 0,%22lld\t",(long long)pow(2.,sz*8)-1);
        printf("signed = -%lld,%lld\n",(long long)pow(2.,sz*(8-1)),(long long)pow(2.,sz*(8-1))-1);
}
int main(){
printType("char",sizeof(char));
printType("wchar_t",sizeof(wchar_t));
printType("short",sizeof(short));
printType("int",sizeof(int));
printType("long",sizeof(long));
printType("long long",sizeof(long long));
return 0;
}
```

N.B. Il compilatore C garantisce i seguenti vincoli:

1. sizeof(char) == 1
2. sizeof(char) <= sizeof(short)
3. sizeof(short) <= sizeof(int)
4. sizeof(int) <= sizeof(long)
5. sizeof(long) <= sizeof(long long)

Proviamo il codice precedente compilando prima con Visual Studio C++ X64 Native otteniamo:

```
C:\>sztypes
      char: 1Byte(s)  unsigned = 0,                255     signed = -128,127
    wchar_t: 2Byte(s)  unsigned = 0,              65535     signed = -16384,16383
     short: 2Byte(s)  unsigned = 0,              65535     signed = -16384,16383
       int: 4Byte(s)  unsigned = 0,         4294967295     signed = -268435456,268435455
      long: 4Byte(s)  unsigned = 0,         4294967295     signed = -268435456,268435455
 long long: 8Byte(s)  unsigned = 0,   9223372036854775807     signed = -72057594037927936,72057594037927935
```

E poi con un altro compilatore ad esempio mingw64, oppure Kali-Linux x64:

```
FP@DESKTOP-LK51Q96 ~ ./a.exe
      char: 1Byte(s)  unsigned = 0,                255     signed = -128,127
    wchar_t: 2Byte(s)  unsigned = 0,              65535     signed = -16384,16383
     short: 2Byte(s)  unsigned = 0,              65535     signed = -16384,16383
       int: 4Byte(s)  unsigned = 0,         4294967295     signed = -268435456,268435455
      long: 8Byte(s)  unsigned = 0,   9223372036854775807     signed = -72057594037927936,72057594037927935
 long long: 8Byte(s)  unsigned = 0,   9223372036854775807     signed = -72057594037927936,72057594037927935
```

Quindi e' importare sottolineare che il tipo long in Visual Studio C++ il long e ampio 4Byte mentre con cygwin64 e ampio il doppio, cioe' 8Byte!!!!

E' importante rimarcare il fatto che il C a differenza di altri linguaggi non azzera automaticamente una variabile in fase di definizione della stessa, come nell'esempio seguente: //array.c

```
#include <stdio.h>
int main(){
        int v[5];
        int i;
        for(i=0;i<5;i++)
                printf("%d\n",v[i]);
        return 0;
```

```
}
```

```
C:\ MYDEV\C>array

-392554243

32758
7
0
0
```

Per azzerare velocemente un'array in fase di definizione utilizzare:
int v[5]={0};
oppure se si vuole azzerare in piu' punti di un programma:

```
for(i=0;i<5;i++)
        v[i]=0;
```

Come detto il C, a differenza di altri linguaggi in cui gli array sono oggetti, non memorizza la lunghezza massima di un array, questo rende il C da un lato piu' veloce ma ne complica non poco l'utilizzo. In pratica nelle applicazioni scritte in C in cui occorre trattare con array, si dimensiona inizialmente l'array (ripeto con valore costante) ad un valore che si presume sufficiente a contenere tutti i dati (questo puo' comportare uno spreco di memoria) e si memorizza in una variabile quanti dati sono stati effettivamente inseriti. Nell'esempio seguente si crea un vettore che puo'contenere al massimo 100 interi, poi si passa in un while in cui l'utente inserisce i dati fino a che decide di terminare inserendo la combinazione di tasti CONTROL+D (scanf ritorna un valore >0 se ha letto almeno un dato del tipo specificato), il numero di dati effettivamente letti vengono memorizzati nella variabile n, (inserire un controllo nel caso in cui n superi NMAX per evitare spiacevoli sorprese):

```
#define NMAX 100
int v[NMAX];
int n=0,i=0;
//legge
while (scanf("%d",&v[n])>0)
        n++;

//stampa il vettore
for(i=0;i<n;i++)
        printf ("%d ",v[i]);
```

ARRAY DI CARATTERI (STRINGHE)
Una stringa e' un vettore di caratteri terminato con il carattere speciale '\0' (NULL)
Esempio:

"salve"

viene memorizzata internamente come (6*8=48 bits nel caso di codifica ASCII, N.B. 6 Bytes!!!):

s	a	l	v	e	'\0'

Quindi attenzione:

```
char s[]="salve";
```

e' profondamente diverso da:

```
char v[]={'s','a','l','v','e'};
```

per renderli equivalenti devo aggiungere il carattere di fine stringa:

```
char v[]={'s','a','l','v','e','\0'};
```

Precedentemente si e' detto che gli array sono allocati staticamente, immutabili, ma in C esiste anche la possibilita' di allocare dinamicamente delle strutture dati come gli array che non vedremo qui. E' importante approfondire il concetto con delle analogie. L'allocazione o memorizzazione statica viene fissata una volta per tutte e non puo' essere variata, come ad esempio i posti a sedere in un aeroplano. I vantaggi sono facilita' e velocita' di gestione (assegno/tolgo passaggeri ai posti che vedo liberi/occupati), svantaggi: spreco di posti se ho pochi passeggeri e numero massimo di passeggeri fisso anche in caso di forte richiesta. L'allocazione o memorizzazione dinamica riserva/riduce spazio su richiesta, e' il caso del treno. Se ho forte richiesta aggiungo vagoni altrimenti li tolgo, questo comporta una migliore gestione delle risorse a scapito della facilita' di gestione (devo comunque impiegare tempo e attenzione nell'aggiungere/togliere vagoni al convoglio corretto).

ESEMPIO:

Un'agenzia di viaggi genera un file di testo le cui righe indicano N° Fila(Max 25): posti occupati (max 5), a partire dalla fila 1, ad esempio:

3:2
7:4
5:3
….

N.B.Nel caso di righe ripetute viene considerata l'ultima. Cioe' nel caso:
3:2
3:4

Nella fila 3 ci saranno 4 posti occupati

Questo file viene inviato al ckin di un aereoporto che provvede a visualizzare a video l'occupazione posti per ogni fila (max 25 file) di un volo:
1]
2]
3]**
5]***
6]
7]****
….

SOLUZIONE
Scriviamo due programmi in C, il primo che scrive su file di testo una serie di coppie di numeri casuali separati dai due punti:

//genposti.c

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

//Funzione che genera un numero casuale tra lowest e highest
 int genRnd( int lowest, int highest) {
   int random_integer;
   int range=max(highest,lowest)-min(highest,lowest)+1*1;
   random_integer = lowest+(int)(range*(rand()-0)/(RAND_MAX -0 + 1.0*1)); return
   random_integer;
}

main(){
        int i;
        int N=25;// Max Num. File
        int P=5; //Max num. Posti per fila
        srand(time(NULL));
```

```
	for(i=0;i<N;i++){
		printf("%d:%d\n",genRnd(1,N),genRnd(1,P));
	}
	return 0;
}
```

Un volta compilato si lancia il programma e si scrive su file, ad es:

C:\>genposti.exe > posti.txt

C:\MYDEV\C>type posti.txt

```
11:5
12:2
5:1
9:2
25:4
20:2
17:1
20:3
20:4
8:4
10:4
7:3
24:3
24:2
24:5
1:5
22:1
18:1
23:5
18:3
24:2
10:5
22:3
8:5
5:4
```

Il secondo programma (volo.c) legge il file precedentemente prodotto (posti.txt) e visualizza la situazione volo:

```
//volo.c

#include <stdio.h>
#define N 25
#define P 5

int main(){
int f=0;
int v[N],i,p=0;
//Azzero i posti
for(i=0;i<N;i++)
	v[i]=0;
//Lettura assegnazione posti (con verifica)
while(scanf("%d:%d",&f,&p)>0){
 if(f>25 || f <=0){
	fprintf(stderr,"Numero di fila:%d errato",f);
	continue;
 }
 if(p>5 || p <0){
```

```
            fprintf(stderr,"Numero di posti occupati %d errato",p); continue;
 }
 v[f]=p;
}

//Stampa situazione
for(i=0;i<N;i++){
            printf("%3d] ",i+1);
            if(v[i]>0){
                        for(int j=0;j<v[i];j++)
                                    printf("*");
            }
            printf("\n");
}
return 0;
}
```

Eseguo il programma:

C:\MYDEV\C>volo < posti.txt

```
  1]
  2] *****
  3]
  4]
  5]
  6] ****
  7]
  8] ***
  9] *****
 10] **
 11] *****
 12] *****
 13] **
 14]
 15]
 16]
 17]
 18] *
 19] ***
 20]
 21] ****
 22]
 23] ***
 24] *****
 25] **
```

Provare a inserire qualche dato non valido nel file posti.txt es:28:2 e notare se il programma lo rileva.

MATRICI

Le matrici sono vettori in piu' dimensioni e vengono definite in C come segue, ad esempio nel caso di una

matrice 3x4 cioe' 3 righe e 4 colonne:

int M[3][4];
La rappresentazione in C e' la seguente dove occorre notare che gli indici di riga e colonna partono da 0. Come per gli array monodimensionali il C non controlla eventuali accessi fuori limite quindi occorre prestare la massima attenzione quando si accede ad un array in lettura e/o scrittura.

M[0][0]	M[0][1]	M[0][2]	M[0][3]
M[1][0]	M[1][1]	M[1][2]	M[1][3]
M[2][0]	M[2][1]	M[2][2]	M[2][3]

Esempio:

Si dispone di un file di testo che contiene la rappresentazione di una scena (labirinto, scacchiera,…) costituita da una sequenza di 1 (ostacoli) e 0 (spazio libero) in una scena di max di 10 colonne e 8 righe.
Esempio file:maze.txt:

```
0100011011
0001111001
0000000000
1100001000
0101000001
1100000001
1000001000
0100001011
```

Si vuole realizzare un programma in C che visualizzi il file precedente con un rettangolo pieno nel caso di 1 e di uno spazio nel caso di 0.

matrici.c

```
#include <stdio.h>

#pragma warning(disable:4996)

//Numero di colonne
#define NC 10
//numero di righe
#define NR 8
int main() {
	int M[NR][NC];
	int i, j;
	char v;
	//N.B.!!! Azzeramento matrice
	for (i = 0; i<NR; i++)
		for (j = 0; j<NC; j++)
		M[i][j] = 0;

	//Lettura dati
	for (i = 0; i<NR; i++) {
		for (j = 0; j < NC; j++) {
			scanf("%c", &v);
while (v == '\n' && v != EOF)
				scanf("%c", &v);
			//scanf(" %c", &ch);
			M[i][j] = v-'0';
		}
	}

	//Stampa matrice codificata
	for (i = 0; i<NR; i++) {
		for (j = 0; j<NC; j++)
			printf("%c",M[i][j]==1?char(219):char(0));
		printf("\n");
	}
	return 0;
}
```

Occorre precisare meglio la lettura dei dati. Non posso dichiarare v come intero perche' con:
int v;

scanf("%d",&v);

verrebbe letta l'intera riga in quanto 0100011011 e' un numero. Occorrera' leggere carattere per carattere su ogni singola riga, quindi la variabile v utilizzata per la lettura dovra' essere dichiarata come char: char v;
scanf("%c",&v);

e memorizzare nella matrice il valore come intero (M[i][j] = v-'0'), altrimenti verrebbe memorizzato il codice ASCII (48 per lo '0' e 49 per '1'). Ma ora si presenta un altro problema, il fatto che leggendo caratteri con la scanf devo considerare il carattere ritorno a capo ('\n') presente in ogni riga del file. Ci sono due modi per fare cio':

```
scanf("%c", &v);
while (v == '\n' && v != EOF)
        scanf("%c", &v);
```

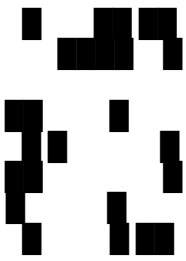
che in pratica significa:*finche' ci sono ritorni a capo ignorali*.

oppure:

scanf(" %c", &ch); (NB. Lo spazio prima di %c).

Eseguendo il programma si ottiene:

C:\>matrici< maze.txt



FUNZIONI e ARRAY

Per quanto riguarda i parametri di funzioni che sono array occorre obbligatoriamente specificare le dimensioni, a meno che che non siano stringhe, in questo caso si puo' anche omettere la dimensione facendo grande attenzione al fatto che deve essere terminato con il carattere NULL.
Esempio:

```
#include <stdio.h>
#define NMAX 100

//riempie un vettore di interi v di lunghezza n con il valore passato in a
int v1d(int v[],int n,int a)
{
 int i=0;
 for (i=0; i < n; i++)
          v[i]=a;
 return i;
}

//riempie una matrice m [nr x nc] con il valore passato in a,
//Il C richiede di specificare il numero massimo di colonne
int v2d(int m[][NMAX],int nr,int nc,int a) {

 int i=0,j=0;
 for (i=0; i < nr; i++)
```

```
        for (j=0; j < nc; j++)

                m[i][j]=a;
 return i+j;
}
//funzione che conta quanti caratteri presenti nella stringa s sono uguali al parametro a
int s1d(char s[], char a)

{
 int i=0,k=0;
 while(s[i++]!='\0')
        if (s[i]==a)
                k++;
 return k;
}

int main()
{
        #define NR 100
        #define NC 100

        int V[NMAX] ={0},M[NMAX][NMAX]={0},i=0,j=0; char
        S[NMAX]="salve a tutti";

        v1d(V,NR,1);
        for (i=0; i< NR;i++)
                printf("%d ",V[i]);

        printf ("\n\n");

        v2d(M,NR,NC,1);
        for (i=0; i< NR;i++)
        {
                for (j=0; j< NC;j++)
                        printf("%d ",M[i][j]);
                printf("\n");
        }
        printf ("\n\n");

        printf("%d",s1d(S,'a'));

        return 0;
}
```

FUNZIONI RICORSIVE

Le funzioni possono essere ricorsive cioe' una funzione puo' chiamare se stessa. Ma dato che una funzione occupa memoria, se non altro perche' puo' dichiarare delle variabili al suo interno, se puo' richiamare se stessa questa puo' esaurire rapidamente la memoria se non si provvede a definire un criterio di terminazione della stessa. Vediamo il classico caso della funzione che calcola il fattoriale di un intero.

Il fattoriale del numero n intero positivo (cioè n!) è definito:

uguale a n*(n-1) (n-2)…..1 se n!= 0
uguale a 1 se n = 0

In altri termini è vera la seguente uguaglianza: n! = n*(n-1)*(n-2)*...2*1. La prima defi-nizione data è di tipo ricorsivo: il fattoriale di un numero viene calcolato come prodotto del numero stesso per il fattoriale del numero che lo precede. Affinché il procedimento sia finito si assume: 0! = 1.

```
        /* Fattoriale di un numero */

        #include <stdio.h>
```

```
int calcFattoriale(int numero);

main(){
     int n,fat;
     printf("\nCalcola il fattoriale di un numero");
     printf("\n\nIntrodurre il numero ");
     scanf("%d",&n);
     fat=calcFattoriale(n);
     printf("\n Fattoriale di %d = %d",n,fat);
}

/* Calcola Fattoriale utilizzando la Ricorsione*/

int calcFattoriale(int numero){ int f;

     if (0==numero)
          f=1;
     else
          f=numero*calcFatt(numero-1);
     return f;
}
```

Per capire la ricorsione di una funzione bisogna sapere che il linguaggio garantisce la rientranza, cioe' se esco da una funzione perche' ad esempio ne chiamo un'altra, quando rientro mi trovo tutte le 'cose' nello stato in cui le avevo lasciate. Cioe':

Esempio di chiamata di funzioni

```
#include <stdio.h>

void g(){

          int a=15;

          printf("a=%d\n",a);

}

void f(){

          int a=3;
          printf("a=%d\n",a);
          g();
          printf("a=%d\n",a);
}

int main(){
    f();
    return 0;
}
```

Output:

```
a=3
a=15
```

L'esempio fa vedere anche che la variabile a definita in g() non ha niente a che fare con la stessa definita in f(), ma la cosa importante e' che la funzione f() dopo aver chiamato la funzione g() si ritrova nella variabile a lo stesso valore (3) che aveva prima di avr chiamato g(). Le funzioni ricorsive sono molto utilizzate ad esempio in robotica come nelle'esempio seguente in cui un robot deve trovare l'uscita in un labirinto rappresentato come una matrice con ostacoli. L'algoritmo per trovare l'uscita e' il piu' semplice possibile nel senso che prova ricorsivamente in tutte e 4 le direzioni. La cella di partenza viene indicata dal numero:2

mentre la cella di arrivo con il numero 3. Il robot qundo si trova in una cella chiama La funzione Solve prova a spostarsi ricorsivamente nella direzione up finche' non ha trovato l'uscita o un ostacolo in questo caso prova a muoversi a sinistra ripetendo le stesse condizioni di prima, seguita dalla ricerca down e right. Il programma seguente contiene anche dei semplici accorgimenti per visualizzare con animazione la ricerca dell'uscita dal labirinto, che comporta la chiamata alla funzione sleep specifica dell'ambiente windows.

roboex.c

```
 /************ROBOEX.C******************
 Semplice Labirinto con Animazione
                                        (Fischetti P.)
 ************************************/
 //////https://www.cs.bu.edu/teaching/alg/maze/
 #include <stdio.h>
 #include <io.h>
 #include <windows.h>
 #include <stdio.h>
 #include <conio.h>

 int Delay = 100;//Animation (ms)

 int MazeRows = 0;
 #define MazeCols            9

 //Il labirinto:
 //            2 la cella di partenza
 //            3 la cella di arrivo
 //            # muri
 //N.B. il numero di righe non e' specificato viene calcolato nel main
 char Maze[][MazeCols+1] =
 {
      "#2#######",
      "#    #   #",
      "# ### # #",
      "# #   # #",
      "# # # ###",
      "#   # # #",
      "# ### # #",
      "#   #   #",
      "#######3#",
 };

 const char chStart = '2'; //Cella di Partenza const char chTarget =
 '3'; // Cella obiettivo const char chWall = '#'; //Cella Muro const
 char chFree = ' '; //Cella Libera
 const char chRobot = '*'; //Il Robot
//Visualizza il Labirinto
void PrintMaze(void )
{
       int r,c;
       system("CLS");
       for (r = 0; r < MazeRows; r++)
       {
              for (c = 0; c < MazeCols; c++)
              {
                            switch(Maze[r][c])
                            {
                              case '#'://Disegno il muro...

                                          printf("%c",(char)219); //... con un quadrato pieno break;

                              default:
```

```
                                        printf("%c",Maze[r][c]);
                                }
                        }
                        printf("\n");
                }
                printf("\n");
}

//Algoritmo di risoluzione ricorsivo
int Solve(int r, int c, int rTarget, int cTarget)
{
        Maze[r][c] = chRobot;

        if(Delay > 0)
        {
          PrintMaze();
          Sleep(Delay);
        }

        // Target Raggiunto?.
        if (r == rTarget && c == cTarget)
        {
                return 1;
        }

        //    Ricerca Ricorsiva...
                //UP

        if (r > 0 && Maze[r-1][c] == chFree && Solve(r - 1, c, rTarget, cTarget)) {
                return 1;
        }
                //LEFT
        if (c > 0 && Maze[r][c-1] == chFree && Solve(r, c - 1, rTarget, cTarget)){
                return 1;
        }

                //DOWN

        if (r < MazeRows-1 && Maze[r+1][c] == chFree && Solve(r+1, c, rTarget, cTarget)){
                return 1;
        }
                //RIGHT

        if (c < MazeCols-1 && Maze[r][c+1] == chFree && Solve(r, c + 1, rTarget, cTarget)){
                return 1;
        }

        //    Cerco un'altra strada. Maze[r][c] =
        chFree;

        if(Delay>0){
                                //Anim
                PrintMaze();
                Sleep(Delay);
        }
      return 0;
}

//Cerco le coordinate della cella di partenza e di arrivo
```

```
int FindStartTarget(int *rs,int *cs,int *rt, int *ct) {
  int sr=-1;
  int sc=-1;
  int er=-1;
  int ec=-1;
  int r,c;
  for(r=0; r < MazeRows; r++)
                for(c=0; c < MazeCols; c++)
                {
                                if(chStart == Maze[r][c])
                                {
                                                sr=r;
                                                sc=c;
                                                Maze[r][c] = chFree;
                                }
                                else if(chTarget == Maze[r][c])
                                {
                                                er=r;
                                                ec=c;
                                                Maze[r][c] = chFree;
                                }
                }
                if(sr<0 || sc <0 || er<0 || ec <0)
                {
                                return 0;
                }
                *rs = sr;*cs=sc;*rt=er,*ct=ec;
                return 1;
}
int main(int argc, char * argv[])
{

        int rStart,cStart,rTarget,cTarget; //Calcolo Numero di righe del
        labirinto MazeRows = sizeof(Maze) / sizeof(Maze[0]);
        if (!FindStartTarget(&rStart,&cStart,&rTarget,&cTarget))
        {

                printf("\nMissing Start/Target item\n"); return -1;
        }

        //Posizionamento iniziale del robot
        Maze[rStart][cStart]=chRobot;
        PrintMaze();
        printf("Premere un tasto per iniziare..."); getch();

        if (Solve(rStart, cStart,rTarget,cTarget))
        {
                 PrintMaze();
                printf("OK");
        }
        else
        {
                 printf("\nFail!!!\n");
        }

        return 0;
}
```