



Laurea triennale in Informatica

modulo (CFU 6) di

Programmazione II e Lab.

prof. Mariarosaria Rizzardi

Centro Direzionale di Napoli – Isola C4

stanza: n. 423 – IV piano Lato Nord

tel.: 081 547 6545

email: mariarosaria.rizzardi@unipARTHENOPE.it

[1-AC]

Unità didattica: Tipo carattere e stringa in C

Titolo: Richiami sulla gestione di stringhe allocate staticamente

Argomenti trattati:

- ✓ Costante carattere e costante stringa: differenze nella memorizzazione
- ✓ Array di caratteri e array di puntatori a stringhe: differenze
- ✓ Input e output di caratteri e stringhe
- ✓ Input di stringhe facendo uso di puntatori
- ✓ Esempio d'uso delle funzioni C di manipolazione delle stringhe
(in `string.h`)

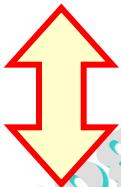
Prerequisiti richiesti: fondamenti del linguaggio C, variabili puntatori

Tipo carattere e tipo stringa

In **C** bisogna far distinzione tra '**a**' e "**a**" perché

- '**a**' (*costante carattere*) = è un intero (1 byte) contenente il valore del codice ASCII corrispondente al carattere.
- "**a**" (*costante stringa*) = è una sequenza di (zero o più) caratteri memorizzata come array di caratteri in cui l'ultima componente contiene il carattere **null** ('**\0**'=0) per indicare la fine della stringa.

'**a**'



ASCII code

97

← 1 byte →

"**a**"



ASCII code terminatore

97

0

← 2 byte →

stringa vuota: " "



terminatore

0

← 1 byte →

Esempio 0a: sbagliato!



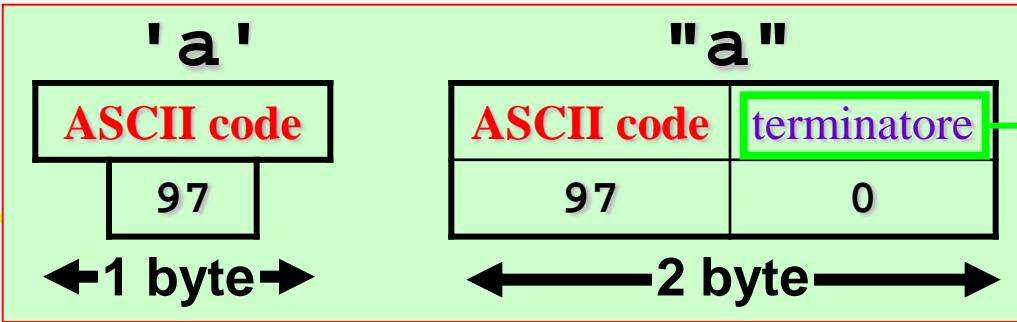
```
#include <stdio.h>
main()
{ char a = 'a'; char A = "a";
printf("sizeof(constant char) = %d\nvalue = %d\n" sizeof(a), a);
```

Search results Build log **Build messages** Debugger Thread search

Line	Message
	In function 'int main()':
3	error: invalid conversion from 'const char*' to 'char'
9	error: invalid types 'char[int]' for array subscript
9	error: invalid types 'char[int]' for array subscript
	==== Build finished: 3 errors, 0 warnings ===

Esempio 0b

```
#include <stdio.h>
void main()
{ char a = 'a'; char A[ ] = "a";
printf("sizeof(constant char) = %d\tvalue = %d\n", sizeof(a),a);
printf("sizeof(constant string) = %d\n", sizeof("a"));
printf("sizeof(string array) = %d\tvalue = %d, %d\n", sizeof(A),A[0],A[1]);
puts("\n");
printf("\tconstant string:\ncontenuto = %d, %d\tindirizzo = %u\n", *"a", *(a+1), "a");
}
```



sizeof(constant char)	= 1	value = 97
sizeof(constant string)	= 2	← strlen(...)+1
sizeof(string array)	= 2	value = 97, 0

constant string:

contenuto = 97, 0

indirizzo = 4206592

Esempio 0c

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

```
void main()
```

```
{ char a = 'a'; char *A = "a";
```



```
printf("sizeof(constant char) = %d\nvalue = %d\n", sizeof(a), a);
```



```
printf("sizeof(constant string) = %d\n", sizeof("a"));
```



```
printf("sizeof(string pointer) = %d\nvalue = %d, %d\n", sizeof(A), *A, *(A+1));
```



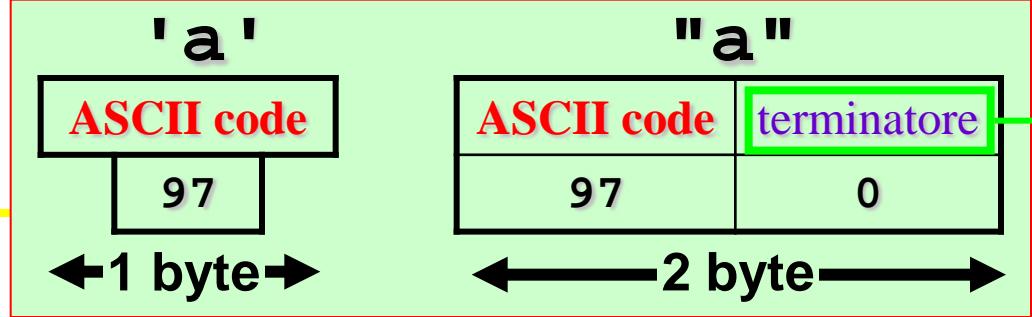
```
puts("\n");
```



```
printf("\tconstant string:\ncontenuto = %d, %d\tindirizzo = %u\n", *"a", *("a"+1), "a");
```



```
}
```

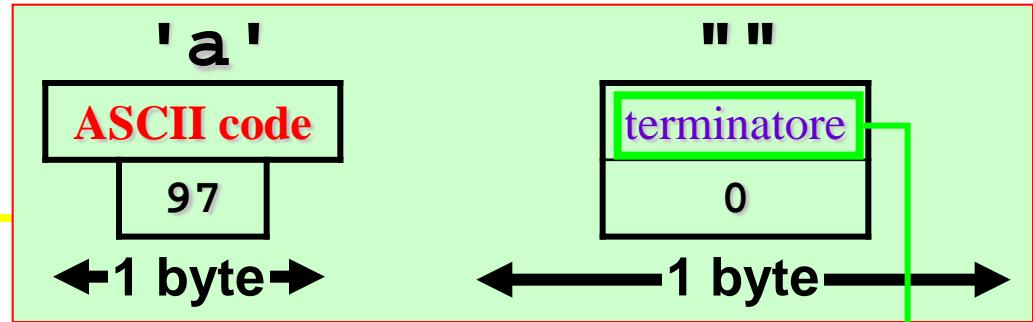


```
sizeof(constant char) = 1      value = 97
sizeof(constant string) = 2      ← strlen(...)+1
sizeof(string pointer) = 8      value = 97, 0
                                ←
                                versione 64 bit
```


constant string:
contenuto = 97, 0 indirizzo = 4206592

Esempio 0d: stringa vuota

```
#include <stdio.h>
void main()
{ char a = 'a'; char A[ ] = "";
printf("sizeof(constant char)    = %d\tvalue = %d\n", sizeof(a),a);
printf("sizeof(constant string)   = %d\n", sizeof(""));
printf("sizeof(string array)      = %d\tvalue = %d\n", sizeof(A),A[0]);
puts("\n");
printf("\tconstant string:\ncontenuto = %d\tindirizzo = %u\n", *", "));
```



sizeof(constant char)	= 1	value = 97
sizeof(constant string)	= 1	← strlen(...)+1
sizeof(string array)	= 1	value = 0

constant string:

contenuto = 0

indirizzo = 4206592

Esempio 1a: main che restituisce il nome del mese

```
#include <stdio.h>
void main()
{int n,riga,j; char mese_di_n[30];
char mesi[][30]={ "Numero di mese non corretto", "Gennaio",
                  "Febbraio", "Marzo", "Aprile",
                  "Maggio", "Giugno", "Luglio", "Agosto", "Settembre",
                  "Ottobre", "Novembre", "Dicembre" };
printf("numero mese = "); scanf("%d",&n);
if (n<1 || n>12) riga=0;
else               riga=n;
for (j=0;j<30;j++)
    mese_di_n[j]=mesi[riga][j];
printf("mese corrispondente = %s\n",mese_di_n);
}
```

variabile del main
matrice di caratteri

input: numero $n \in \{1,2,\dots,12\}$

output: nome del mese

Esempio 1b: function che restituisce il nome del mese

p2

```
#include <stdio.h>
void nome_mese(int n, char *);
void main()
{
    int n; char mese_di_n[30];
    printf("numero mese = "); scanf("%d",&n);
    nome_mese(n,mese_di_n);
    printf("mese corrispondente = %s\n",mese_di_n);
}
void nome_mese(int n,char mese[])
{char mesi[][30]={ "Numero di mese non corretto", "Gennaio",
                  "Febbraio", "Marzo", "Aprile", "Maggio",
                  "Giugno", "Luglio", "Agosto", "Settembre",
                  "Ottobre", "Novembre", "Dicembre"};
short j,riga;
    if (n<1 || n>12)    riga=0;
    else                  riga=n;
    for (j=0;j<30;j++)
        mese[j]=mesi[riga][j];
}
```

variabile della function

Negli **eempi 1a** e **1b** la variabile **mesi** è un array bidimensionale (matrice) di caratteri di size 13×30 .



spreco di memoria

La matrice **mesi**(13×30) deve avere tutte le righe eguali anche se una riga contiene solo un carattere!

Esempio 1c: main che restituisce il nome del mese (con puntatori)

eSEMPIO DI array frastagliato

```
#include <stdio.h>
void main()
{short n,riga; char *mese_di_n;
char *mesi[ ]={"Numero di mese non corretto","Gennaio",
    "Febbraio","Marzo","Aprile","Maggio",
    "Giugno","Luglio","Agosto","Settembre",
    "Ottobre","Novembre","Dicembre"};  

printf("numero mese = "); scanf("%d",&n);
if (n<1 || n>12) riga=0;
else
    riga=n;
mese_di_n=mesi[riga];
printf("mese corrispondente = %s\n",mese_di_n);
}
```

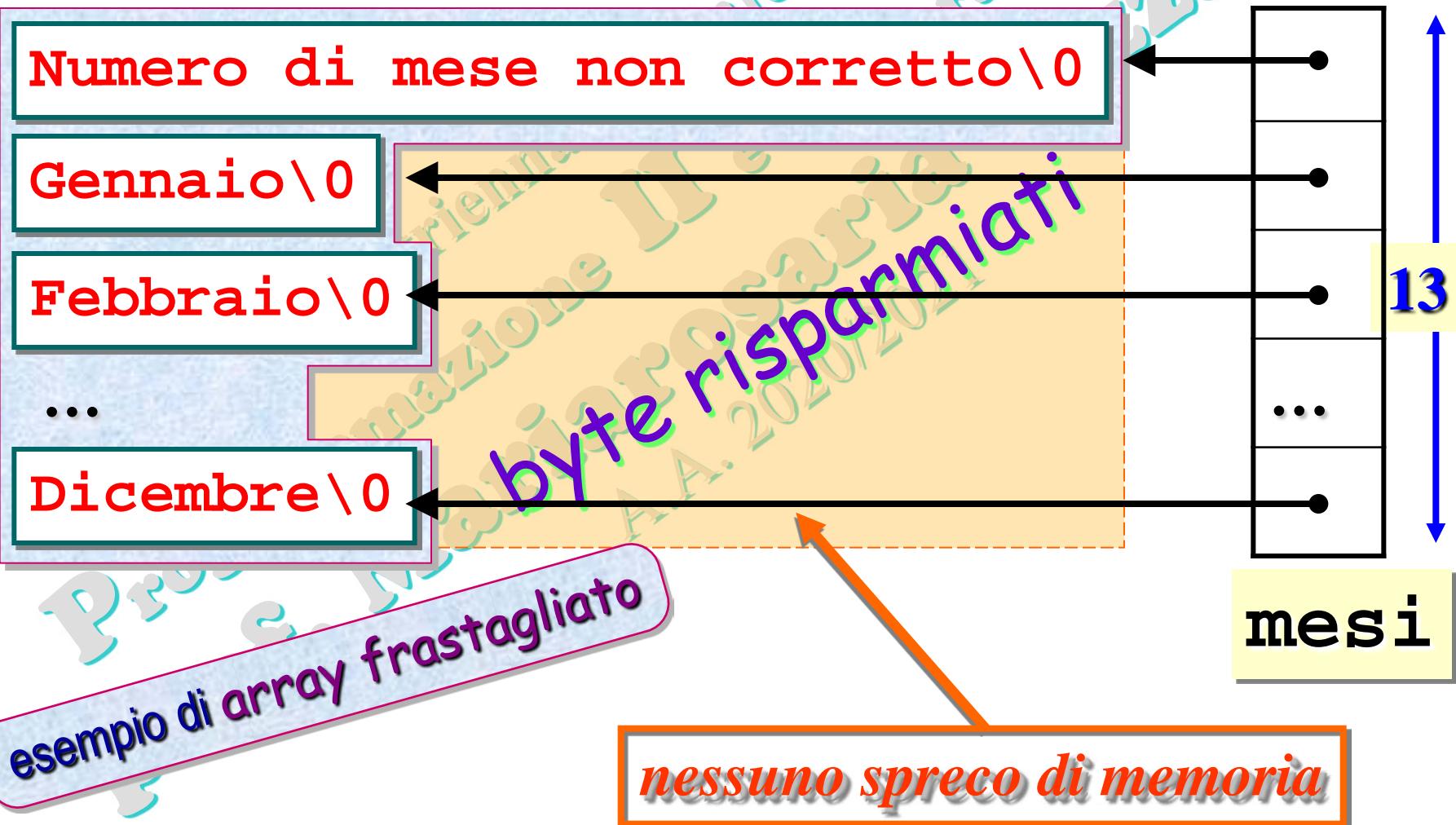
array di puntatori a stringhe

Esempio 1d: function che restituisce il nome del mese (con puntatori)

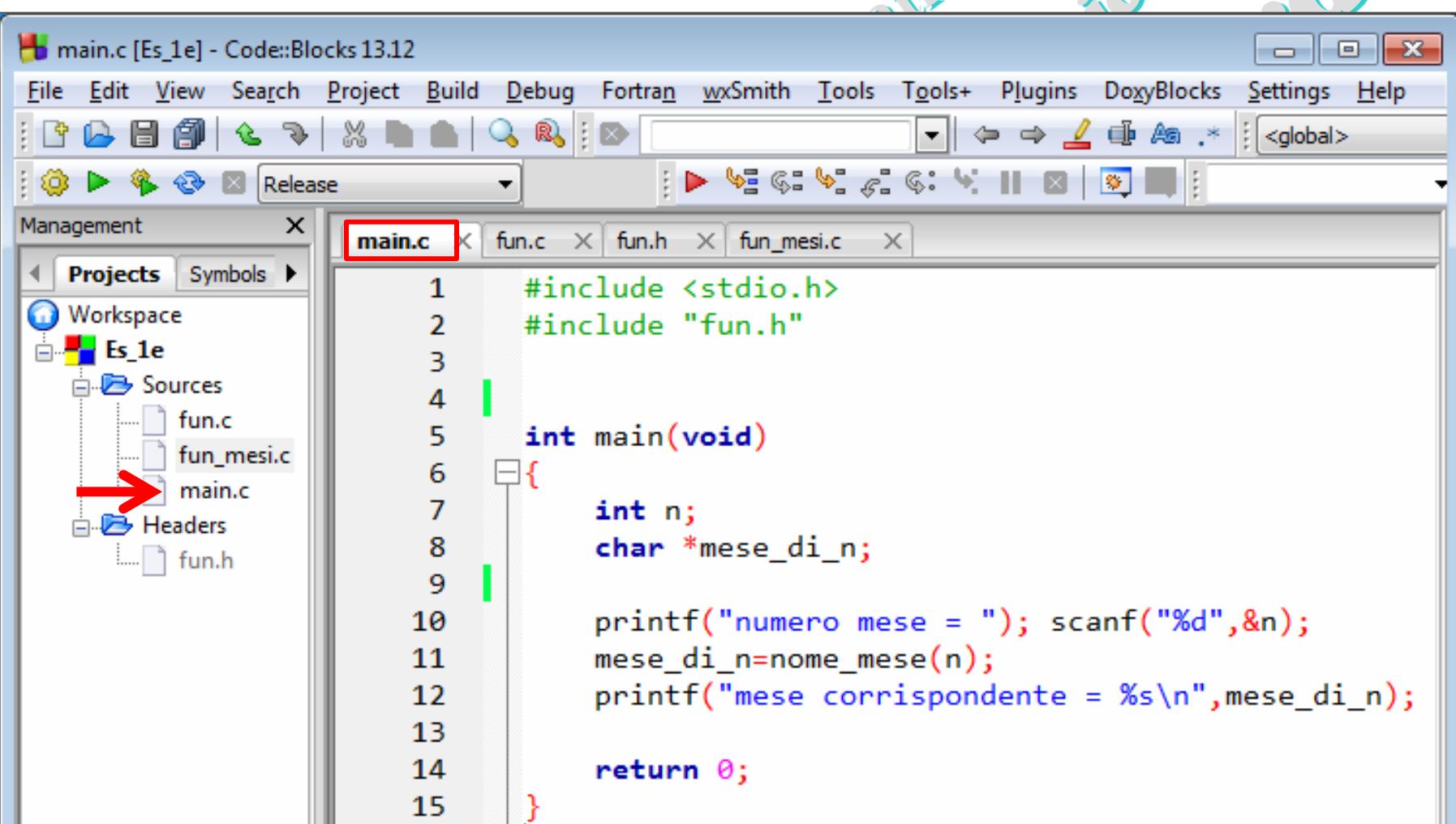
```
#include <stdio.h>
char *mesi[]={"Numero di mese non corretto","Gennaio",
               "Febbraio","Marzo","Aprile","Maggio",
               "Giugno","Luglio","Agosto","Settembre",
               "Ottobre","Novembre","Dicembre"};
char *nome_mese(int n);
void main()
{int n; char *mese_di_n;
 printf("numero mese = "); scanf("%d",&n);
 mese_di_n=nome_mese(n);
 printf("mese corrispondente = %s\n",mese_di_n);
}
/*-----*/
char *nome_mese(int n)
{short riga;
 if (n<1 || n>12) riga=0;
 else
    riga=n;
return mesi[riga];
}
```

variabile globale nel file corrente

Negli **esempi 1c e 1d** la variabile **mesi** è un **array** di **puntatori** a stringhe di caratteri: in tal modo non si copiano caratteri bensì solo un intero (il *puntatore*) ed in più si occupa solo la memoria che serve (*efficienza di tempo e di spazio*).



Esempio 1e: function che restituisce il nome del mese (con puntatori) e variabile **extern** [1]



The screenshot shows the Code::Blocks IDE interface. The title bar reads "main.c [Es_1e] - Code::Blocks 13.12". The menu bar includes File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The status bar shows "<global>". The left sidebar has a "Management" section with "Projects" selected, showing a workspace named "Es_1e" containing "Sources" (with files fun.c, fun_mesi.c, and main.c) and "Headers" (with file fun.h). A red arrow points to the "main.c" file in the "Sources" folder. The main editor window displays the following C code:

```
1 #include <stdio.h>
2 #include "fun.h"
3
4
5 int main(void)
6 {
7     int n;
8     char *mese_di_n;
9
10    printf("numero mese = "); scanf("%d",&n);
11    mese_di_n=nome_mese(n);
12    printf("mese corrispondente = %s\n",mese_di_n);
13
14    return 0;
15 }
```

Esempio 1e: function che restituisce il nome del mese (con puntatori) e variabile **extern** [2]

The screenshot shows the Code::Blocks IDE interface. On the left, the Project Manager window displays a workspace named 'Es_1e' containing files: 'Sources' (with 'fun.c') and 'Headers' (with 'fun.h'). A red arrow points to the 'Sources' folder. The main editor window shows the 'fun.c' file with the following code:

```
1 char *nome_mese(int n)
2 {
3     short riga;
4     extern char *mesi[];
5
6     if (n<1 || n>12)
7         riga=0;
8     else
9         riga=n;
10
11     return mesi[riga];
12 }
```

A green callout box with a red arrow points to the declaration of the variable 'mesi' in line 4. The text inside the callout box reads: "Le variabili extern sono variabili globali condivise da function memorizzate in file diversi di uno stesso progetto".

Questo file usa
l'array frastagliato
mesi

The screenshot shows the Code::Blocks IDE interface. The main editor window shows the 'fun.h' file with the following code:

```
1 char *nome_mese(int n);
```

A red arrow points to the header file 'fun.h' in the Project Manager window.

Esempio 1e: function che restituisce il nome del mese (con puntatori) e variabile **extern** [3]

The screenshot shows the Code::Blocks IDE interface. The title bar says "fun_mesi.c [Es_1e] - Code::Blocks 13.12". The menu bar includes File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The status bar at the bottom right shows "ricevuto" and "adattato". The left sidebar has a "Management" tab with "Projects" selected, showing a workspace named "Es_1e" containing files "fun.c", "fun_mesi.c" (highlighted with a red arrow), "main.c", and headers "fun.h". The main editor window shows the code for "fun_mesi.c":

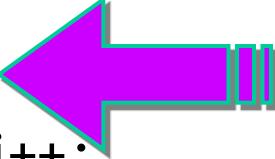
```
1 char *mesi[] = {"Numero di mese non corretto",
2                 "Gennaio",
3                 "Febbraio",
4                 "Marzo",
5                 "Aprile",
6                 "Maggio",
7                 "Giugno",
8                 "Luglio",
9                 "Agosto",
10                "Settembre",
11                "Ottobre",
12                "Novembre",
13                "Dicembre"};
```

Questo file definisce
l'array frastagliato mesi

Il modo più semplice per leggere un carattere da tastiera è tramite **getchar()** in **<stdio.h>** (**int getchar(void)**)

Il seguente programma legge da tastiera una stringa, carattere per carattere (fino a **\n = new line**), e ne calcola il numero di caratteri

```
#include <stdio.h>
void main()
{char buffer[61]; int c,j;
j=0;
do{c=getchar();
  buffer[j]=(char)c; j++;
} while (c != '\n');
buffer[j-1]='\0'; /* toglie '\n'*/
printf("la stringa %s e' lunga "
      "%d caratteri\n", buffer, strlen(buffer));
}
```



equivalente a:

c = getc(stdin);
c = fgetc(stdin);

Il modo più semplice per visualizzare un carattere è tramite **putchar(c)** in **<stdio.h>** (**int putchar(int)**)

\$ man getchar
\$ man putchar

In Linux per la documentazione

equivalente a:

putc(c, stdout);
fputc(c, stdout);

... e per leggere una stringa?

[...tramite array]

```
#include <stdio.h>
#include <string.h>
void main()
{char s[10], c32=32;
puts("immetti stringa ...");
fflush(stdin); gets(s); /* scanf("%s",s); */
puts("strlen = "); printf("%d\n", (int)strlen(s));
puts("la stringa e' ...");
printf("%s\n",s);
s[0] = s[0]^c32;
puts("la stringa e' ...");
printf("%s\n",s);
}
```

deprecata

È meglio usare:
fgets(s,10,stdin);

fflush(stdin);

ripulisce il buffer di input

```
immetti stringa ...
ciao
strlen =
4
la stringa e' ...
ciao
la stringa e' ...
Ciao
```

Invece di gets(s) è meglio usare fgets(s,4,stdin): esempio 1

```
#include <stdio.h>
#include <string.h>
void main()
{
    char s[4], s1[10]={'A','B','C','\0'};
    printf("\t1) stringa s1 = \"%s\"\n", s1);
    puts("nimmetti stringa s:");
    fflush(stdin); gets(s);
    printf("strlen(s) = %d\n", (int)strlen(s));
    printf("la stringa e': \"%s\"\n\n",s);
    printf("\t2) stringa s1 = \"%s\"\n", s1);
}
```

in memoria

...	n
1013	i
1012	r
1011	t
1010	s
...	
1003	'\0'
1002	'C'
1001	'B'
1000	'A'

s

s1

Più caratteri dello spazio
disponibile nell'array s

1) stringa s1 = "ABC"

immetti stringa s:
stringa lunga!

strlen(s) = 14

la stringa e': "stringa lunga!"

Tutto OK!

Invece di gets(s) è meglio usare fgets(s,4,stdin): esempio 2

```
#include <stdio.h>
#include <string.h>
void main()
{
    char s1[10]={'A','B','C','\0'}, s[4];
    printf("\t1) stringa s1 = \"%s\"\n", s1);
    puts("nimmetti stringa s:");
    fflush(stdin); gets(s);
    printf("strlen(s) = %d\n", (int)strlen(s));
    printf("la stringa e': \"%s\"\n\n",s);
    printf("\t2) stringa s1 = \"%s\"\n", s1);
}
```

in memoria		
...		u
1007	'\0'	l
1006	'C'	a
1005	'B'	g
1004	'A'	n
1003	i	
1002	r	
1001	t	
1000	s	



Più caratteri dello spazio disponibile nell'array s

Modificata!

1) stringa s1 = "ABC"

immetti stringa s:
stringa lunga!

strlen(s) = 14

la stringa e': "stringa lunga!"

2) stringa s1 = nga lunga!

Invece di gets(s) è meglio usare fgets(s,4,stdin): esempio 3

```
#include <stdio.h>
#include <string.h>
void main()
{
    char *s1="ABC", s[4];
    printf("\t1) stringa s1 = \"%s\"\n", s1);
    puts("nimmetti stringa s:");
    fflush(stdin); gets(s);
    printf("strlen(s) = %d\n", (int)strlen(s));
    printf("la stringa e': \"%s\"\n\n",s);
    printf("\t2) stringa s1 = \"%s\"\n", s1);
}
```

in memoria

		u
1007	'\0'	l
1006	'C'	a
1005	'B'	g
1004	'A'	n
1003	i	
1002	r	
1001	t	
1000	s	

s1

s

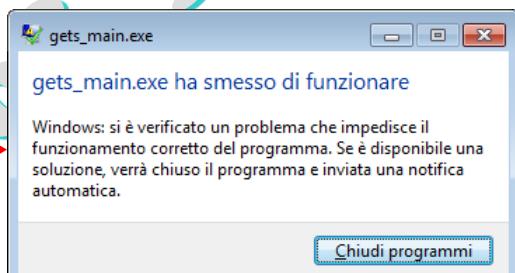
i caratteri immessi cercano di modificare la stringa costante

1) stringa s1 = "ABC"

immetti stringa s:
stringa lunga!

strlen(s) = 14

la stringa e': "stringa lunga!"



Invece di gets(s) è meglio usare fgets(s,4,stdin): esempio 4

```
#include <stdio.h>
#include <string.h>
void main()
{
    char *s1="ABC", s[4];
    printf("\t1) stringa s1 = \"%s\"\n", s1);
    puts("nimmetti stringa s:");
    fflush(stdin); fgets(s,4,stdin);
    printf("strlen(s) = %d\n", (int)strlen(s));
    printf("la stringa e': \"%s\"\n\n",s);
    printf("\t2) stringa s1 = \"%s\"\n", s1);
}
```

Più caratteri dello spazio disponibile nell'array s

Tutto OK!

1) stringa s1 = "ABC"

immetti stringa s:

stringa lunga!

strlen(s) = 3

la stringa e': "str"

2) stringa s1 = "ABC"

... e per leggere una stringa tramite puntatore?

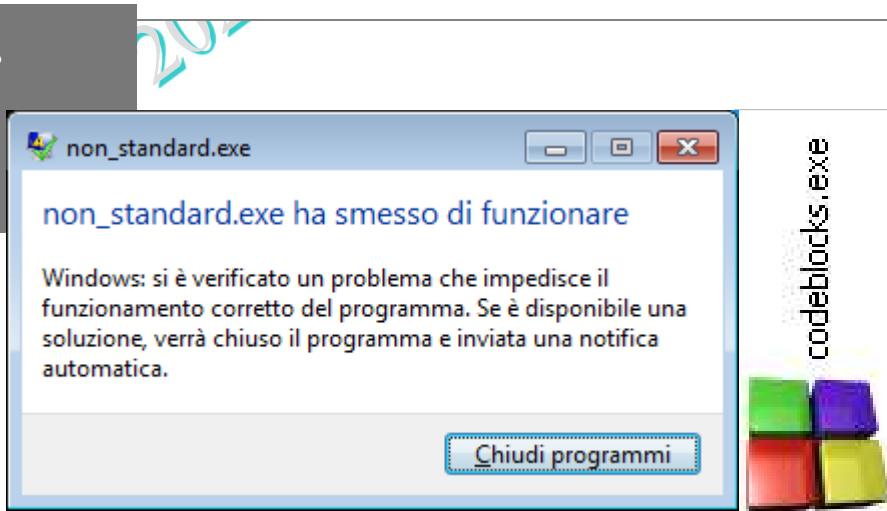
```

1 | #include <stdio.h>
2 | #include <string.h>
3 | int main(void)
4 |
5 |     char *p_string, c32 = 32; // c32=' ' spazio bianco
6 |     puts("immetti stringa ... ");
7 |     fflush(stdin); gets(p_string); //scanf("%s",p_string);
8 |
9 |     puts("strlen = "); printf("%d\n", strlen(p_string));
10 |    puts("la stringa e' ... "); printf("%s\n", p_string);
11 |    (*p_string) = (*p_string)^c32;
12 |    puts("la stringa e' "); printf("%s\n", p_string);
13 |    return 0;
14 |

```

NON FUNZIONA!!!

immetti stringa ...
ciao



... ma allora ...

come si gestiscono le stringhe
tramite puntatori?

... mediante

allocazione dinamica!!!



Scuola triennale in Ingegneria
dell'Informazione e delle Tecnologie
dell'Informazione
Ricerca II e Didattica
di prof. M. Rizzardi
2021

per leggere una stringa tramite puntatore con allocazione dinamica

OK !!!

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

void main()

{char *p_string; int stringlen;

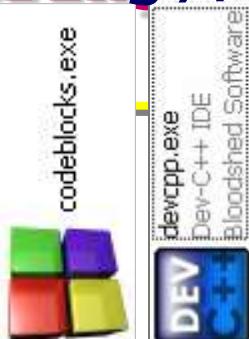
printf("\n lunghezza stringa = ");scanf("%d",&stringlen);

p_string = (char *)malloc(stringlen + 1);

printf("\n immetti stringa = ");
fflush(stdin); fgets(p_string, stringlen + 1, stdin);
printf("\n strlen(p_string) = %d\n",strlen(p_string));
printf("\n stringa immessa = %s\n",p_string);

...
free(p_string);
}
```

Pro

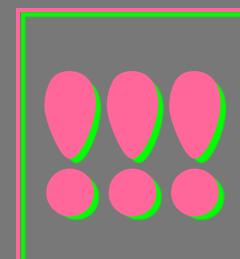


```
lunghezza stringa = 4

immetti stringa = CIAO

strlen(p_string) = 4

stringa immessa = CIAO
```



MSDEV.EXE
Microsoft (R) Developer Studio
Microsoft Corporation

Alcune funzioni sulle stringhe (in `<string.h>`)

`char *pc, *ps, *pt, *s1, *s2`

strlen(ps)	Restituisce la lunghezza (senza contare il carattere \0) di <code>*ps</code>
strcpy(pt,ps)	Copia <code>*ps</code> in <code>*pt</code> compreso il carattere \0
strcat(s1,s2)	Concatena ad <code>*s1</code> la stringa <code>*s2</code>
strcmp(s1,s2)	Confronta <code>*s1</code> e <code>*s2</code> : restituisce un valore < 0 se <code>*s1 < *s2</code>
strchr(pt,pc)	Restituisce un puntatore alla prima occorrenza del carattere <code>*pc</code> in <code>*pt</code>
strstr(pt,ps)	Restituisce un puntatore alla prima occorrenza della stringa <code>*ps</code> in <code>*pt</code>

```

#include <stdio.h>
#include <string.h>
void main()
{char stringa[]="AP: Algoritmi e Programmazione";
char newString[30];
char *ps1="Algoritmi", *ps2="Programmazione";
printf("\n%d=lunghezza di stringa[%s]\n",strlen(stringa),
stringa);
printf("\nps1->stringa = %s\n",ps1);
strcpy(newString, ps1);
printf("\nnewString->stringa = %s\n",newString);
strcat(newString, " e ");
printf("\nnewString->stringa...");
strcat(newString, ps2);
printf("\nnewString->stringa...");

```

30 = lunghezza di stringa[AP: Algoritmi e Programmazione]

ps1->stringa = Algoritmi

newString->stringa = Algoritmi

newString->stringa = Algoritmi e

newString->stringa = Algoritmi e Programmazione

Quiz: che differenza c'è nei due programmi C che seguono

```
...
void main()
{char stringa[ ] = "ciao";
puts(stringa);
*stringa = 'm';
puts(stringa);
}
```

```
...
void main()
{char *stringa = "ciao";
puts(stringa);
*stringa = 'm';
puts(stringa);
}
```

Esercizi: confrontando i risultati con le relative funzioni del C ed utilizzando per le stringhe

- l'allocazione statica
- l'allocazione dinamica

...

1

... scrivere una *function C* che, dopo aver stabilito il numero totale dei caratteri, legga da tastiera i singoli caratteri costruendo la stringa che li contiene senza usare **strcat(...)**.

2

... scrivere una *function C* che restituisca la concatenazione di due stringhe date come parametri di input senza usare **strcat(...)**.