

Corso di Informatica per la laurea triennale in Matematica
Complementi al libro di Kernighan e Ritchie

Michele Casamassima Nicola Corriero
Vittoria Cozza Eustrat Zhupa

Contents

1	Organizzazione dei dati su un calcolatore	1
2	Programmi e Linguaggi di programmazione	6
3	Primi programmi C	10
4	Digressione: un programma C che si riproduce	23
5	La libreria standard <code>< string.h ></code>	26
6	Teoremi	28
7	Svolgimento degli esercizi del Kernighan e Ritchie, Cap. 1	39
8	Esercizi numerici	70

1 Organizzazione dei dati su un calcolatore

1 Alfabeto, lettera, stringa Un *alfabeto* Γ è un insieme finito non vuoto i cui elementi sono associati ad entità grafiche dette *lettere* o *caratteri*. Una sequenza finita w di $n \geq 0$ lettere di Γ è, per definizione, una *stringa* in Γ (di *lunghezza* n). La stringa di lunghezza 0 è detta *vuota* e viene solitamente denotata col simbolo ε . Γ^* e Γ^+ denotano risp. l'insieme delle stringhe e l'insieme delle stringhe non vuote in Γ .

Se per esempio M è l'alfabeto ordinario si ha $\text{Bill} \in M^+$ nonché $\varepsilon \in M^*$.

Certe volte conviene assumere che un alfabeto includa una lettera con funzioni di spazio. Se ridefiniamo M aggiungendoci lo spazio abbiamo che Bill uguale Peppino è una stringa di lunghezza 19, mentre Bill uguale Peppino è una stringa di lunghezza 21 (perché le parti visibili questa volta sono separate da due spazi).

Quanto sopra è abbastanza pacifico perché siamo andati a capo prima di B e abbiamo continuato a scrivere subito dopo la o. Se però avessimo scritto

Bill uguale Peppino

non sarebbe stato chiaro se c'erano degli spazi a sinistra e a destra, e quanti. Questa è una delle innumerevoli ragioni per cui in molti casi conviene usare *meta*-simboli¹ non appartenenti (per definizione) a Γ per isolare ogni o qualche stringa e/o per esibire gli spazi.

La lunghezza di "Bill" è 4, quella di " Bill " è 7. Cosa che si vede meglio se scriviamo "_Bill_" o anche solo _Bill_ (nel seguito il segno _ sarà usato con altri scopi e non andrà confuso con gli spazi bianchi).

2 Concatenazione La concatenazione della stringa u di lunghezza m con la stringa w di lunghezza n è la stringa z di lunghezza $m + n$ e denotata da $u \circ w$ che comincia con le lettere di u e finisce con quelle di w (nello stesso ordine, per piacere). Sicché ha senso scrivere

$$z = u \circ w$$

Essendo la concatenazione ovviamente associativa ($b \circ (a \circ ri) = (ba) \circ ri$), si omette spesso il simbolo \circ . Osserviamo che si ha

$$\varepsilon w = w\varepsilon = w$$

si può dunque dire che ε è l'elemento neutro della struttura di *monoide* (ossia di gruppo senza inversa) indotta su Γ^* dalla concatenazione.

Le nozioni che precedono hanno carattere matematico, e, più precisamente, logico-algebrico. Lo stesso non può dirsi di quanto segue.

¹La metamatematica è un ramo della matematica che pretende di ridurre gli altri ad oggetto del proprio studio. Di lì viene l'uso del prefisso *meta-* per indicare che si parla in un (superiore) linguaggio degli enti di un altro linguaggio (inferiore, in quel contesto). Parlando degli spazi negli oggetti-stringa usiamo il meta-simbolo _ che è un simbolo del nostro linguaggio *non* appartenente a Γ ma denotante un particolare simbolo di Γ .

3 File È noto che l'informatica studia il comportamento di alcuni dispositivi che consentono di registrare o conservare delle informazioni piú o meno a lungo. L'astrazione principale della *Computer science* si chiama *file*. Un file è una stringa (nell'alfabeto che diciamo subito) registrata in un dispositivo.

In linea di principio è irrilevante se un file è in un CD, in un disco rigido (HD=hard disk) esterno o interno al calcolatore sul quale si sta lavorando, o nella sua *memoria* o addirittura se è *digitato* sulla sua tastiera o se compare sul suo *monitor*. Sempre abbiamo una stringa scritta da qualche parte.

Ogni file ha un *filename* che permette di rintracciarlo nei e tra i diversi dispositivi di un calcolatore. Tale nome ha la forma

$$u.w$$

in cui u è una stringa nell'alfabeto

$$\{a, \dots, z, A, \dots, 0, \dots, 9, ., -\}$$

che non comincia con una cifra decimale, e che, se chi lo crea è una matricola, non deve cominciare con $.$

La parte che segue il punto si chiama *extension* e in genere è costituita da una, due o tre lettere, che informano sulla natura del file. Per esempio la parte a sinistra del filename

Lezione_2007.tex

ricorda a chi lo ha creato che cosa esso contiene (ossia, che cosa significa per lui la stringa registrata in quel file) mentre la parte a destra dice che è *un file di testo* e anche un *file sorgente* (vedremo cosa significa) per il programma noto come L^AT_EX. Questa seconda parte è obbligatoria nei sistemi commerciali; invece in Linux essa è facoltativa per tutti meno le matricole di Matematica. Il primo programma C scritto dallo studente Vito potrà chiamarsi per esempio

Vito_program1.c

4 Bit, byte, ASCII Si sa che tutta la chincaglieria (HW, *hardware*) usata da un calcolatore per registrare informazioni è costituita da circuiti aperti/chiusi, da materiale magnetizzato in positivo/negativo e da altre *cose* a due stati. Tutto è rappresentabile con degli zeri e degli uni, e i file sono dunque stringhe nell'alfabeto $\{0, 1\}$ o, piú semplicemente, stringhe di *bit*. I singoli bit sono poco maneggevoli, e

conviene raggrupparli. È oggi prevalente l'organizzazione dei file in ottetti di bit detti *byte* ².

Siccome una sequenza come 01000001 01000010 01000011 01000100 è illeggibile, conviene dare un nome a ciascun byte. Per fortuna sono finiti i tempi in cui ogni costruttore battezzava i byte a modo suo, e si è imposto uno standard detto ASCII ³. In questo standard i numeri binari da 65 a 90 corrispondono alle maiuscole e la sfilza di bit che abbiamo appena scritto diviene da A B C D. In realtà ASCII codifica solo i 128 byte che cominciano con 0. Sono rappresentate le maiuscole, le minuscole, i segni di interpunzione, vari tipi di parentesi e virgolette, i sacri dollaro e percento, e qualche altra cosa. Alcuni byte molto importanti (lo vedremo) sono *invisibili*, perché corrispondono alle tabulazioni, agli *accapo* etc.

Siccome bisogna fare i conti non solo con i simboli europei ma anche con le lettere cinesi, giapponesi etc, esiste un altro standard detto Unicode che adopera parecchi byte. Mentre ASCII e Unicode sono uguali per tutti, le rappresentazioni di Unicode dipndono dai sistemi. Nel senso che Unicode associa per esempio la alef ebraica all'esadecimale 05D0, ma poi Linux e Windows codificano 05D0 in modi diversi ⁴

5 Directory Ci sono varie categorie di file. Un file *regolare* contiene delle informazioni usate da un utente per suoi scopi che, in linea di principio, riguardano solo lui e i programmi che egli adopera. Esempi di file regolare sono: un testo, un oggetto multimediale (brano musicale, immagine, video), un programma.

In Linux sono file *non regolari* le *directory* di cui infra, ed altri oggetti al di fuori degli argomenti di questo corso.

I file prodotti da un utente possono essere tanti (migliaia o milioni) o anche contarsi sulle dita della mano; i file *di sistema* gestiti dagli amministratori sono tanti ⁵. Bisogna dunque che i file siano organizzati da amministratori e utenti in modo logico.

In tutti i calcolatori i file si dispongono secondo un albero ispirato al metodo *genere*

²forse una corruzione ortografica di bite (= *morso*), boccone: la macchina si *mangia* le stringhe a bocconi di otto bit.

³per American Standard Code for Information Interchange

⁴Una delle ragioni per cui i sistemi commerciali sono più vulnerabili dagli *hacker* è che in Unicode essi usano il primo bit, ed è difficile capire se un lungo testo per esempio in cinese contiene delle insidie. Invece nessuna sotto-stringa dell'Unicode di Linux è una stringa ASCII, ed è facile impedire ai testi Unicode di accedere alle parti critiche del sistema per far danno. .

⁵In questo momento sul calcolatore usato per scrivere queste dispense ci sono 140.000 file di sistema, più quelli dei suoi due utenti (che poi è una stessa persona che di solito fa il *login* come utente semplice, per evitare di fare danni inconsapevolmente; e come amministratore solo quando vuole modificare il sistema).

prossimo - differenza specifica. Le foglie dell'albero sono file regolari. Tutti gli altri nodi si chiamano *directory* o anche *folder* (*cartelle*). Per definizione, una *directory* associata ad un dato nodo N *contiene* gli oggetti associati ai nodi *figli* di N . Essi possono essere file regolari, file di altro tipo e altre *directory*.

Sia dato un utente XYZ che non ha *privilegi di amministratore*, nel senso che non ha né il diritto di modificare l'assetto del sistema né quello di ficcare il naso nei file altrui. Assumiamo che egli abbia il livello minimo di conoscenze sul sistema che si acquisisce in una mattinata spesa seriamente. La prima volta egli può interagire col sistema perché un amministratore gli ha dato una *login*, una *password* e probabilmente ha creato per lui una *home directory* con un nome particolare, diciamo XY. È lì che egli colloca i suoi file, e li organizza in un albero di *directory*, mettendoci quel tanto di buon senso e metodo che vuole e può .

Passiamo ora ad illustrare quello che succede in Linux, e non necessariamente nei sistemi commerciali di tipo Windows.

6 Il file system Linux (1) In rigorosa applicazione del principio secondo cui ogni cosa è un file, le *directory* sono file.

(2) I file regolari contengono solo le informazioni che servono al loro utente. Tutte le meta-informazioni relative ad un file regolare (tipo, dimensione, data di creazione e modifica, privilegi di lettura scrittura ed esecuzione, etc.) sono contenute nella (stringa registrata nella) *directory* a cui esso appartiene. Siccome una *directory* è un file, le metainformazioni che la riguardano sono nella *directory* a cui essa appartiene ⁶.

(3) La radice dell'albero dell'intero computer si chiama *root*. Essa contiene le metainformazioni relative alle sue sotto-*directory*, mentre le meta-info su di lei sono costanti note ai programmi di gestione del file system. Se in un dato momento XYZ sta usando il calcolatore, la sua *home directory* XY è denotata da \sim .

(4) Si individua un file (regolare, *directory* o altro) F nell'albero mediante il suo *path* (cammino). Esso ha la forma

$$/F_1/ \dots /F_n/F \quad (n \geq 0)$$

dove gli F_i sono tutte e sole le *directory* che stanno tra la *root* e F. Siccome ho chiamato *home* la mia *home directory*, e l'ho messa nella *directory* /, il *path* di

⁶Per esempio un file video comincia con informazioni sul fatto che contiene, appunto, un video, poi su regista, attori, etc; infine viene una (codifica) del video stesso; invece il numero dei byte del file è associato al filename nella *directory* immediatamente superiore. Questa è la ragione per cui in Linux le extension nei filename hanno solo funzioni mnemoniche.

queste dispense e quello di tutta la musica nel mio computer sono risp.

`/home/X/dispense/matematica/anno_2007` `/home/X/musica`

e possono anche essere scritti nella forma

`~ /dispense/matematica/anno_2007` `~ /musica`

2 Programmi e Linguaggi di programmazione

7 Memorie Ogni calcolatore moderno dispone di almeno due tipi di *memorie*, dette RAM (*random access memory*) e HD (*hard disk*). La prima è costosa, veloce e *volatile*, nel senso che il suo contenuto è cancellato quando la macchina viene spenta. Lo HD è circa cento volte più capace della RAM, migliaia di volte più lento, ma il suo contenuto è permanente ⁷

Una differenza essenziale tra RAM e HD è che quella è *indirizzabile* (onde l'aggettivo *random*). Questo significa che (quasi) ad ogni byte è associato un indirizzo numerico compreso tra 0 e un milione.

Denotiamo con B l'alfabeto costituito dai 256 byte. GB denoti B^{10^6} (prodotto cartesiano per un milione di volte) e $100GB$ denoti GB^{100} . Una i.d. (*descrizione istantanea*) è un elemento di $GB \times 100GB$, può essere vista come una stringa lunga 101 milioni di byte, e rappresenta il contenuto di RAM e HD in un dato istante. Sia ID l'insieme di tutte le i.d. Esse possono essere classificate in *iniziali*, *attive* e *halting*, con il significato che il loro nome suggerisce.

8 Computazione Un *passo* da parte di una data macchina M è una funzione

$$\text{step}_M : ID \mapsto ID$$

Una *computazione* da parte di M è una sequenza

$$d_1, \dots, d_k \quad (d_{j+1} = \text{step}_M(d_j) \quad \text{per ogni } j < k)$$

in cui per ogni j si ha $d_{j+1} = \text{step}_M(d_j)$.

M per entrata (input) la id d dà in uscita (output) la d^* se esiste una computazione $d = d_1, \dots, d_k = d^*$, e se d e d^* sono risp. iniziale e halting.

⁷In realtà le cose sono più complicate di così, e va detto almeno che oltre alla RAM ci sono i *registri*, che sono delle memorie volatili molto più veloci e più care. Siccome il contenuto dei registri non è modificabile dai comuni mortali, possiamo liquidarli come fatto tecnologico teoricamente irrilevante, e supporre che avvenga nella RAM quello che in realtà avviene in essi.

9 Ciclo di macchina La funzione step_M rappresenta il comportamento della CPU (central processing unit) che è la parte più importante di un calcolatore, e ne caratterizza il comportamento. La CPU e quindi la funzione step dipendono dal costruttore, e si pone ancora una volta un problema di standard.

Tutte hanno però in comune un *ciclo di macchina* la cui astrazione nonché prima realizzazione è dovuta al matematico von Neumann (pronuncia: *fon nòimann*, per piacere). L'*invariante* di tale ciclo consiste nell'eseguire l'*istruzione I* contenuta in un dato indirizzo m . Tale I determina

- (a) l'operazione Y da eseguire (in genere un'operazione *aritmetica* o un *trasferimento*);
- (b) l'indirizzo n della prossima istruzione da eseguire.

La funzione step consiste dunque di una serie di clausole della forma

if in m c'è I do Y ; go to n

Questo ciclo comincia con una i.d. iniziale e viene ripetuto finché non si raggiunge una i.d. finale.

10 Programmi e linguaggi Il programma di un concerto o di un corso è una lista di brani o argomenti da suonare o illustrare. Un programma per un calcolatore è una lista di istruzioni da eseguire una dopo l'altra. Si conviene che, per *default*, le istruzioni debbono essere eseguite nell'ordine in cui sono scritte; la parte *go to n* è inclusa solo quando non si voglia passare all'istruzione successiva. Per esempio un programma può avere la forma

13000427743
14000593419
12000274027

e significare che un certo dato deve essere trasferito in un registro, che gli si deve sommare un altro dato e che il risultato deve essere registrato in un qualche altro indirizzo. Si dice che questo è un programma in *linguaggio macchina*

È previsto un mezzo tecnico per ottenere che quelle istruzioni siano inserite nella macchina ed essere eseguite. Dalla fine degli anni '60 i linguaggi macchina sono capiti solo dai produttori di hardware. Un pò meno oscuro è un programma in linguaggio *assembler* come

LOAD b, %r0

```
LOAD c, %r1
ADD %r0, %r1
SUB &2, %r1
STORE %r1, a
```

Ogni riga di assembler è solo un'istruzione di linguaggio macchina scritta in modo un pò più leggibile, e c'è un programma in linguaggio macchina detto *interprete* che lo trasforma in un programma in linguaggio macchina. (Per chi ha fame di dettagli: `%ri` significa registro `i`, e `&2` è la costante 2; il ruolo delle 4 parole in maiuscolo è trasparente). Quello che fa è essere descritto dall'espressione aritmetica

$$a=b+c-2$$

Questa espressione appartiene anche al linguaggio C che ci accingiamo a studiare.

11 Scrittura, compilazione, esecuzione Il C è uno dei più diffusi linguaggi di programmazione ad *alto livello* (l'altezza di un linguaggio misura la sua distanza dal linguaggio macchina, e la sua potenza). Un programma in C deve essere sottoposto alle fasi principali seguenti.

(1) Deve essere scritto con l'aiuto di un programma con funzioni di *editor*. Se l'editor che si chiama Kate è disponibile in laboratorio lo adotteremo, se no si vedrà.

Una volta scritto, il programma deve essere *salvato* in un file nella directory del suo autore, con un filename nella forma

```
xxx.c
```

dove `xxx` è un nome scelto dall'utente (con educazione, e senza troppo humour).

(2) Il programma `xxx.c` è trasformato in un programma in linguaggio macchina da un programma *compilatore*. Il compilatore Linux che useremo si chiama `gcc`. Se al *prompt* si digita il *comando*

```
gcc -o xxx.bin xxx.c
```

il compilatore trasforma il programma *sorgente* `xxx.c` nel programma in linguaggio macchina (in *binario*) `xxx.bin`.

(3) A questo punto il comando

```
$ xxx.bin
```

provoca l'esecuzione di `xxx.bin` e, se non sono stati fatti sbagli, la funzione descritta da `xxx.c` viene calcolata.

(5) I comandi vengono immessi da tastiera (*stdin*=standard input) quando il sistema si dichiara pronto a riceverli, facendo apparire sul monitor dei segni prestabiliti, detti appunto *prompt*. Il prompt è personalizzabile, e dipende quindi dalla macchina e anche dall'utente. Spesso è ridotto al solo segno # o al solo segno \$.

12 Il corso, le esercitazioni, l'esame Scopo del corso è insegnare a scrivere semplici programmi C. Un pò come con i linguaggio naturali è facile masticarne un pò, ed è difficilissimo dominarlo.

Tutti gli studenti saranno ripartiti in gruppi di studio permanente di 5 unità ciascuno. Subito dopo la trattazione in aula delle nozioni che precedono, è previsto (se le necessarie risorse saranno fornite dal dipartimento di matematica) che venga illustrato a ciascun gruppo, direttamente su PC con *sistema operativo* Linux, come

- (1) ci si auto-presenta al PC mediante il processo di **login**;
- (2) si usa un editor, e si salva quanto scritto;
- (3) si effettuano alcune semplici operazioni con i comandi **echo**, **ls**, **cat**, **makedir**, **cd**, **rm**, **rmdir**.

Il corso comincia a questo punto la trattazione del C, alternando la presentazione del linguaggio con l'analisi di esempi di programma di complessità crescente, e con la soluzione di esercizi.

I gruppi che parteciperanno attivamente all'analisi degli esempi e svolgeranno gli esercizi, dimostrando così di aver capito i rudimenti del linguaggio e di saperlo usare, lucreeranno degli *esoneri* di peso decisivo sull'esame. Gli altri dimostreranno la stessa cosa in sede di esame risolvendo un semplice esercizio e discutendo uno degli esempi forniti.

13 Il materiale didattico (1) Almeno le prime 34 pagine del libro

Brian Kernighan and Dennis Ritchie *The ANSI C programming Language*
seconda edizione, editore Prentice-Hall

ne esiste una traduzione italiana, abbastanza disponibile nelle librerie universitarie.

(2) Queste pagine.

(3) Gli esercizi del libro svolti ed illustrati dai docenti e ricercatori che collaborano a questo corso.

Quanto sub (2) e (3) sarà in rete sulla pagina web del docente. Sarebbe da biasimare il comportamento di chi usasse un software p2p per *scaricare* dalla rete il libro (in inglese), o ne chiedesse una copia ad un docente che avesse eventualmente già compiuto per i propri scopi un'operazione così disdicevole. Si ricorda anche che è proibito usare una copia del libro prestata dalla biblioteca del dip. di Informatica o da un docente per fotocopiarne le pagine in programma.

3 Primi programmi C

14 Notazioni (1) Scriviamo

```
/* xxx.c */  
L1  
...  
Lk
```

per dire che un'opportuna directory contiene il file `xxx.c` e che questo file è costituito da quelle $k \geq 1$ linee.

(2) `n --->` è il *prompt* adottato per questi appunti; in esso `n` è un numero identificativo e non progressivo.

(2) Scriviamo invece

```
R1  
...  
Rh
```

per dire che ad un certo punto il monitor contiene quelle $h \geq 1$ righe. Se sono nella forma

```
519 ~/EsC ---> ...
```

esse dicono che al prompt è stata digitata l'espressione ... (per lo più un comando). Le righe non precedute dal prompt possono essere

- (a) uno *stdout* (standard output) provocato dall'esecuzione di un nostro programma;
- (b) un messaggio di errore mandato dal sistema, perché il monitor è anche il file di *stderr* (standard error);
- (c) l'eco sul monitor dello *stdin*.

Solo il contesto permette di distinguere i tre casi.

Per esempio nelle esecuzioni del programma `div_2arg.c`, sono *stdout*:

```
''dividendo'' ''divisore'' ''22 diviso 7 fa 3 con resto 1''
```

sono invece *stdin*: `''22'' ''7'' ''0''`;

infine abbiamo ricevuto il messaggio *stderr* `''Floating point exception''` perché abbiamo provato a calcolare l'infinito.

15 La funzione main Ogni *programma* calcola una o più *funzioni*. La principale di esse si deve chiamare `main`. Le parentesi tonde racchiudono gli argomenti. Le graffe descrivono il da farsi.

Il calcolo di una funzione può dare sotto prodotti interessanti, detti *side effect*. Spesso si calcolano funzioni senza argomenti perché interessano solo i side effect.

```
/* ciao.c */
main(){
putchar('c');
putchar('i');
putchar(97);
putchar('o');
putchar('!');
putchar('\n');
}
```

```
519 ~/EsC ---> gcc -o ciao.bin ciao.c
520 ~/EsC ---> ciao.bin
ciao!
521 ~/EsC --->
```

Perché 97 è l'ASCII per *a* minuscola nella maggioranza dei sistemi, ma non in tutti: scrivendo 97 invece di 'a' abbiamo risparmiato un simbolo, ma abbiamo reso il programma `ciao.c` non portabile.

Si noti che gli apici sono in C l'equivalente del corsivo. Servono a passare da una lettera al nome della lettera: Bari comincia con Santo Spirito e finisce con Torre a Mare, mentre *Bari* comincia con *B* e finisce con *i*. Senza questa distinzione non potremmo fargli capire se vogliamo scrivere la cifra decimale 7 oppure suonare un beep (7 in ASCII è l'allarme \a).

16 Librerie È stupido rifare una cosa già fatta da qualcun altro. Un programma può includere funzioni già scritte. Per esempio funzioni che facilitano l'I/O. Per includere la *libreria* di funzioni già scritte `lib` si usa la *direttiva* `#include <lib >`.

```
/* buongiorno.c */
#include <stdio.h>
main(){
printf("buongiorno, padrone\n");
}
```

```
517 ~/EsC ---> gcc -o buongiorno.bin buongiorno.c
518 ~/EsC ---> buongiorno.bin
buongiorno, padrone
```

17 Protezioni e simboli invisibili Il *newline* è ottenuto da un particolare simbolo ASCII che è denotato in C dall'espressione `\n` che conta come un solo carattere. Si dice che ** protegge la lettera che lo segue: la lettera `n` è protetta in

```
    putchar('\n');
```

In genere una lettera protetta denota un qualche simbolo non scrivibile o particolare (per esempio, per togliere alle virgolette semplici e doppie certi loro ruoli si scrive `\'` e `\''`; il *backslash* viene protetto scrivendo `\\`).

```
/* buongiorno2.c */
#include <stdio.h>
main(){
printf("buongiorno ");
printf(",");
printf("padrone\n");
}
```

```
519 ~/EsC ---> gcc -o buongiorno2.bin buongiorno2.c
520 ~/EsC ---> buongiorno2.bin
buongiorno ,padrone
```

abbiamo messo male lo spazio, correggiamo

```
/* buongiorno2.c */
#include <stdio.h>
main(){
printf("buongiorno");
printf(",");
printf(" padrone\n");
}
```

```
521 ~/EsC ---> buongiorno2.bin
buongiorno ,padrone
522 ~/EsC ---> gcc -o buongiorno2.bin buongiorno2.c
523 ~/EsC ---> buongiorno2.bin
buongiorno, padrone
```

se non ci ricordiamo di ri-compilare la macchina non lo sa e fa come prima.

18 Caratteri, interi e aritmetica Le variabili di tipo carattere sono numeri.

```
/* dpl.c */
char c;
main(){
c=getchar();
putchar(c);
c=c+1;
putchar(c);
putchar('\n');
}
```

```
523 ~/EsC ---> gcc -o dpl.bin dpl.c
524 ~/EsC ---> dpl.bin
a
ab
525 ~/EsC ---> dpl.bin
E
EF
526 ~/EsC ---> dpl.bin
z
z{
527 ~/EsC ---> 523 ~/EsC ---> gcc -o dpl.bin dpl.c
524 ~/EsC ---> dpl.bin
a
ab
525 ~/EsC ---> dpl.bin
```

```
E
EF
526 ~/EsC ----> dpl.bin
z
z{
527 ~/EsC ----> dpl.bin
6
67
528 ~/EsC ----> dpl.bin
0
01
529 ~/EsC ----> dpl.bin
9
9:
530 ~/EsC ---->
```

```
/* div_int.c */
#include <stdio.h>
int a,b,q,r;
main(){
a=22;
b=7;
q=a/b;
r=a%b;
printf("%d diviso %d fa %d con resto %d\n",a,b,q,r);
}
```

```
534 ~/EsC ----> gcc -o div_int.bin div_int.c
535 ~/EsC ----> div_int.bin
22 diviso 7 fa 3 con resto 1
```

19 Programmi che leggono i loro argomenti

```
/* div_2arg.c */
#include <stdio.h>
int a,b,q,r;
main(){
printf("dividendo:\n");
scanf("%d",&a);
printf("divisore:\n");
scanf("%d",&b);
q=a/b;
r=a%b;
printf("%d diviso %d fa %d con resto %d\n",a,b,q,r);
}
```

```
570 ~/EsC ---> gcc -o div_2arg.bin div_2arg.c
571 ~/EsC ---> div_2arg.bin
dividendo:
22
divisore:
7
22 diviso 7 fa 3 con resto 1
572 ~/EsC ---> div_2arg.bin
dividendo:
22000
divisore:
7
22000 diviso 7 fa 3142 con resto 6
573 ~/EsC ---> div_2arg.bin
dividendo:
22
divisore:
0
Floating point exception
574 ~/EsC --->
```

20 Virgola mobile Un modo un pò artigianale di ottenere virgole e decimali

```
/* div_3d.c */
#include <stdio.h>
int a,a1,b,q,q1,r1;
main(){
a=22;
a1=a*1000;
b=7;
q=a1/b;
q1=q/1000;
r1=q%1000;
printf("%d diviso %d fa %d.%d\n",a,b,q1,r1);
}
```

```
561 ~/EsC ---> gcc -o div_3d.bin div_3d.c
562 ~/EsC ---> div_3d.bin
22 diviso 7 fa 3.142
```

Un modo un pò piú raffinato è con variabili di tipo *float(-ing point)*

```
/* div_float.c */
#include <stdio.h>
float a,b,q,r;
main(){
printf("dividendo:\n");
scanf("%f",&a);
printf("divisore:\n");
scanf("%f",&b);
q=a/b;
printf("%f diviso %f fa %f\n",a,b,q);
printf("%5.f diviso %5.f fa %f\n",a,b,q);
}
```

```
677 ~/EsC ---> gcc -o div_float.bin div_float.c
678 ~/EsC ---> div_float.bin
dividendo:
22
divisore:
7
22.000000 diviso 7.000000 fa 3.142857
    22 diviso    7 fa 3.142857
```

Finora abbiamo tre tipi di dati: char int float
Variabili di tipo intero e caratteri

```
/* char_int.c */
#include <stdio.h>
main(){
char a;
int b,c,d;
a=getchar();
b=a;
c=a+1;
printf("%d = valore di %c\n",a,b);
printf("dopo %d viene %c essendo %d il suo valore\n",a,c,c);
}
```

```
535 ~/EsC ---> gcc -o char_int.bin char_int.c
536 ~/EsC ---> char_int.bin
E
69 = valore di E
dopo 69 viene F essendo 70 il suo valore
537 ~/EsC ---> char_int.bin
z
122 = valore di z
dopo 122 viene { essendo 123 il suo valore
538 ~/EsC ---> char_int.bin
```

```
0
48 = valore di 0
dopo 48 viene 1 essendo 49 il suo valore
539 ~/EsC ---> char_int.bin
9
57 = valore di 9
dopo 57 viene : essendo 58 il suo valore
```

21 Strutture di controllo Nel corso di una computazione accade
(1) di scegliere tra azioni diverse;
(2) di ripetere per un numero fisso di volte una certa sotto-computazione;
(3) di ripetere una sotto-computazione per un numero di volte dipendente dall'andamento dei risultati parziali.

Nel primo caso, il modo piú semplice è con un *costrutto* di tipo `if`

```
/* min.c */
#include <stdio.h>
int a,b;
main(){
printf("primo argomento:\n");
scanf("%d",&a);
printf("secondo argomento:\n");
scanf("%d",&b);
if (a <= b) printf("%d minore o uguale di %d\n",a,b);
else printf("%d maggiore di %d\n",a,b);
}
```

```
592 ~/EsC ---> gcc -o min.bin min.c
593 ~/EsC ---> min.bin
primo argomento:
22
secondo argomento:
7
22 maggiore di 7
594 ~/EsC ---> min.bin
primo argomento:
```

0
secondo argomento:
7
0 minore o uguale di 7

Una sub-computazione può essere ripetuta con un costrutto di tipo `for`

```
/* rad_q.c */
#include<stdio.h>
int a,i;
main(){
scanf("%d",&a);
for(i=0;i*i<=a;i++);
printf("%d\n",i-1);
}
```

```
544 ~/EsC ---> gcc -o rad_q.bin rad_q.c
545 ~/EsC ---> rad_q.bin
100
10
546 ~/EsC ---> rad_q.bin
0
0
547 ~/EsC ---> rad_q.bin
1
1
548 ~/EsC ---> rad_q.bin
5
2
```

```
/* num_primo.c (funziona solo per a>4) */
#include<stdio.h>
int a,i,b;
main(){
```

```

scanf("%d",&a);
b=0;
for(i=3;i<=a/2;i+=2)
if(a%i==0)b=1;
(b==0)?printf("primo\n"):printf("non primo\n");
}

```

```

562 ~/EsC ---> gcc -o num_primo.bin num_primo.c
563 ~/EsC ---> num_primo.bin
7
primo
564 ~/EsC ---> num_primo.bin
2007
non primo
565 ~/EsC ---> num_primo.bin
2009
non primo
566 ~/EsC ---> num_primo.bin
2011
primo
567 ~/EsC ---> num_primo.bin
5
primo
568 ~/EsC ---> num_primo.bin
6
non primo

```

```

/* prime.c
* tavola di tutti gli i < 300 che sono primi
* calcola la radice j di i
* poi prova a dividere per tutti i k < j
* l=7 valori per riga
* per avere le colonne ordinate pone %3d */
#include <stdio.h>
int i, j, k;

```

```

int l = 7;
main(){
    for(i = 3;i <= 299;i += 2){
        for(j=2;j*j<=i;j++);
        for(k=3;k<j && i % k;k += 2 );
        if (k >= j){
            printf("%3d  ",i);
            if (l == 0){
                l = 7;
                printf("\n",i);
            }
            else --l;
        }
    }
    printf("\n",i);
}

```

```

570 ~/EsC ---> prime.bin
   3   5   7  11  13  17  19  23
  29  31  37  41  43  47  53  59
  61  67  71  73  79  83  89  97
101 103 107 109 113 127 131 137
139 149 151 157 163 167 173 179
181 191 193 197 199 211 223 227
229 233 239 241 251 257 263 269
271 277 281 283 293

```

22 Funzioni; annidamento di for diverse; indenting

```

/* prime.c
* tavola di tutti gli i < 100 che sono primi
* usa una funzione per calcolare la radice k di i
* e una funzione per dividere per alcuni j < k */
#include <stdio.h>

```

```

int sqr(int);
int prm(int);
main(){
    int i;
    printf("2 ");
    for(i = 3;i < 100;i+=2)
        if(prm(i)==1) printf("%d ",i);
    printf("\n",i);
}
int sqr(int a){
    int k;
    for(k=1;k*k<=a;k++);
    return k--;
}
int prm(int b){
    int j,c;
    for(j=3;j<=sqr(b) && (c=b%j) != 0;j += 2 );
        if (c==0) return 0; else return 1;
}

```

```

605 ~/EsC ---> gcc -o prime.bin prime.c
606 ~/EsC ---> prime.bin
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

607 ~/EsC --->

```

La riga bianca in piú prima di 607 perché per combinazione la riga in uscita 2...97 è larga come il mio monitor.

4 Digressione: un programma C che si riproduce

23 Programma che stampa la sua dichiarazione iniziale

```
# include<stdio.h>
char s[]={
'0',
'8',
'0',
0,
};
main(){
int i;
printf("char s[]={\n");
for(i=0;s[i];i++)
printf("\t%c,\n",s[i]);
printf("}\n");
}
```

```
~ --> a.out
char s[]={
    0,
    8,
    0,
}
```

24 Programma che stampa se stesso

```
char s[ ]={
'\t',
'0',
'\n',
'}',
';',
'\n',
'\n',
'/',
'*',
```

```

    '\n',

    (200 righe ca. omesse)

    0,
};

/*
 * la stringa s rappresenta tutto
 * il programma da 0 in poi
 */
main( )
{
    int i;
    printf("char\ts[ ]={\n");
    for(i=0;s[i];i++)
        printf("\t%d,\n",s[i]);
    printf("%s",s);
}

```

Chiamiamo P questo programma e Q la sua uscita. Abbiamo $Q = Q_1Q_2$, dove Q_1 è l'output delle righe di P fino allo scope della `for` incluso e Q_2 è prodotto dall'ultima `printf`. Si verifica per simulazione che $Q_1 =$

```

char s[ ]={
    9,                \t in ASCII
    0,
    10,              \n
    125,            }
    73,
    10,
    10,
    47,
    42,
    10,

```

(200 righe ca. omesse)

a questo punto il test della `for` diviene falso (perché si ha $s[i] = 0$ — si osservi la differenza tra `'0'` = 48 e 0) e si passa all'ultima `printf`, ottenendo $Q_2 =$

```

    0
};

/*
 * la stringa s rappresenta tutto
 * il programma da 0 in poi
 */
main( ){
    int i;
    printf("char\t s[ ]={\n");
    for(i=0;s[i];i++)
        printf("\t%d,\n",s[i]);
    printf("%s",s);
}

```

Q differisce da P solo perché l'enumerazione delle componenti dell'array di caratteri s è fatta con i numeri ASCII, invece che con caratteri espliciti. Siccome la seconda `printf` usa `%d`, l'output di Q è Q stesso.

Im/morale Così come Q scrive perfino il suo commento C , si può scrivere un qualunque programma che, non solo si riproduce, ma porta con sé, invece di un innocente commento, tutto il più o meno esplosivo bagaglio B che vuole.

5 La libreria standard < string.h >

Questa libreria contiene alcune funzioni C che consentono di lavorare con facilità su stringhe. In questa sezione **r**, **s** e **t** sono stringhe. Inoltre si assume che tutti i programmi comincino con le direttive e la dichiarazione

```
#include <stdio.h>
#include <string.h>
#define M 100
char s[M],t[M];
```

25 Copiare una stringa La funzione `strcpy(s,t)` sostituisce (il valore di) **s** con (quello di) **t**, e restituisce (il nuovo valore di) **s**.

```
main(){
scanf("%s",t);
printf("%s\n",strcpy(t,s));
}
74 ~/EsC ---> a.out (d'ora in poi il comando di compilazione viene omissso)
roma
roma
```

26 Concatenare due stringhe La funzione `strcat(s,t)` sostituisce **s** con la concatenazione di **s** (prima) con **t** (dopo). Per esempio

```
main(){
scanf("%s",s);
printf("%s\n",strcat(s,"bari"));
}
81 ~/EsC ---> a.out
roma
romabari
```

Per default, le stringhe dichiarate ma non assegnate sono vuote. Quindi se **t** è vuota `strcpy(t,s)` e `strcat(t,s)` si equivalgono.

27 Trovare una lettera La funzione `strcspn(s,t)` restituisce l'intero

$\min i[(i + 1)\text{-ma lettera di } s \text{ è anche una lettera di } t \text{ o non esiste}]$

```

char t[]="() []";
int i;
main(){
    scanf("%s",s);
    i=strcspn(s,t);
    printf("%d\n",i);
}
131 ~/EsC ---> a.out
ro(ma
2
132 ~/EsC ---> a.out
roma
4

```

perché [è la terza lettera di ro[ma e perché la quinta lettera di roma non esiste.

28 Testa di n caratteri La funzione `strncpy(t,s,i)` si comporta come `strcpy(t,s)` ma tronca dopo i lettere, senza mettere il fine stringa `'\0'`. Se vogliamo lavorare su t (per esempio stampandolo) ce lo dobbiamo mettere noi.

```

int i;
main(){
    scanf("%s",t);
    scanf("%d",&i);
    strncpy(s,t,i);
    s[i]='\0';
    printf("%s\n",s);
}
144 ~/EsC ---> a.out
roma
2
ro

```

29 Confronto tra stringhe `strcmp(s,t)` restituisce -1 , 0 , $+1$ a seconda che s preceda t in ordine alfabetico ASCII, le sia uguale, o la segua. Il suo dominio è a tre valori per facilitare la programmazione degli algoritmi di ordinamento.

30 La testa fino alla prima parentesi Definiamo ora una funzione nostra (ossia non appartenente alla libreria `< string.h >`) che restituisce la parte della stringa `s` che precede la prima parentesi tonda aperta.

```
hd_lt_par(){
    char r[]="(";
    int i;
    i=strcspn(s,r);
    strncpy(t,s,i);
}
```

Essa assume che `char s[]` e `t[]` siano state dichiarate come stringhe, e assegna ad `s` le lettere di `t` che precedono la prima '(' . Abbiamo per esempio

```
main(){
    scanf("%s",s);
    hd_lt_par();
    printf("%s\n",t);
}
hd_lt_par(){
    char r[]="(";
    int i;
    i=strcspn(s,r);
    strncpy(t,s,i);
    t[i]='\0';
}
141 ~/EsC ---> a.out
ro(ma
ro
```

6 Teoremi

31 Notazione Σ denoterà l'insieme delle stringhe nell'alfabeto nel quale si scrivono i programmi C . $\varphi^{(n)}$ denoterà una generica funzione *parziale* (ossia non necessariamente definita dappertutto) ad n posti a valori in Σ . In simboli:

$$\varphi^{(n)} : \Sigma^n \mapsto \Sigma \quad (n = 1, 2)$$

l'indicazione del numero dei posti sarà omessa quando esso è noto o privo d'interesse.

Intendiamoci: $\varphi^{(n)}$ è un ente matematico (sottoinsieme di $\Sigma^{(n+1)}$), e non un'espressione del C: non è detto che ci sia un programma C che la calcola.

32 Definizione Sia data una definizione x di una funzione C nella forma $x=$

$$x = \text{fn}()\{\dots\}$$

la quale non ha parametri in entrata, ed ha dunque solo dei *side effect*. Assumiamo che essi consistano nel modificare una stringa c , a partire da una stringa a e forse anche da una seconda stringa b .

Diciamo che x calcola in modo standard (*s-calcola*) la funzione $\varphi^{(1)}$ se quando si esce da x dopo una chiamata sintatticamente corretta si ha

$$c = \varphi(a)$$

Sicché per esempio il programma

```
main(){
    scanf("%s",s);
    fn();
    printf("%s\n",c);
}
x
```

stampa la stringa c sse si ha $c = \varphi(a)$.

In modo del tutto analogo, x *s-calcola* $\varphi^{(2)}$ se dopo una chiamata corretta si ha $c = \varphi(a, b)$.

Notazione $\varphi_x^{(n)}$ ($n = 1, 2$, spesso omessa) è la funzione *s-calcolata* da x .

33 Nota Occorre distinguere tra

(1) s e t che sono usate (per esempio in sez.5) come variabili C generiche, che possono essere sostituite da identificatori qualsiasi.

(2) a , b e c che sono identificatori fissati una volta per tutte come parte dello standard di calcolo (se no la dimostrazione del lemma 38 cade).

(3) u, \dots, z che sono meta-variabili definite su Σ , i cui valori possono venire assegnati ad a, b, c .

Si noti inoltre che una funzione che calcola in modo standard può chiamare funzioni che invece non rispettino lo standard.

34 Esempio La stringa $x=$

```
id(){
    strcpy (c,a);
}
```

s-calcola la funzione-identità $\varphi(z) = z$. Poniamo ora $y=$

```
due_a_diverse(){
    char d[]="a";
    strcpy(c,d);
    strcat(c,a);
    strcat(c,b);
    strcat(c,d);
}
```

Abbiamo

```
main(){
    scanf("%s%s",a,b);
    char d[]="a";
    due_a_diverse();
    printf("%s\n",c);
}
due_a_diverse(){
    char d[]="a";
    strcpy(c,d);
    strcat(c,a);
    strcat(c,b);
    strcat(c,d);
}
```

```
474 ~/EsC ---> a.out
```

```
a a
```

```
aaaa
```

```
475 ~/EsC ---> a.out
```

```
a b
```

```
aaba
```

```
476 ~/EsC ---> a.out
```

```
roma bari
```

```
aromabaria
```

Possiamo concludere che, per i valori di \mathbf{x} e di \mathbf{y} ora indicati, e per ogni valore delle variabili \mathbf{w} e \mathbf{z} si ha

$$\varphi_{\mathbf{x}}(\mathbf{z}) = \mathbf{z} \qquad \varphi_{\mathbf{y}}(\mathbf{w}, \mathbf{z}) = \mathbf{awza}$$

nel secondo caso perché qualcuno si è divertito ad usare \mathbf{a} sia come nome di variabile, che come stringa costituita da un solo carattere assegnata alla costante \mathbf{d} . Un limite a questo genere di scherzi, che può anche dar luogo a conseguenze sulle quali c'è da ridere molto di meno (o di più, secondo gusti ed etiche) può venire da quanto segue.

35 λ -notazione In logica ed informatica teorica si distingue tra

- (a) funzioni e loro descrizioni;
- (b) costanti e variabili.

Se z è una variabile definita su un dominio D e se

... z ...

è un'espressione di un qualche linguaggio che associa ai valori di z elementi di un codominio R , allora (per definizione)

$$\lambda z.[\dots z \dots]$$

è la funzione a dominio D e codominio R che è descritta da quell'espressione (ma che avrebbe potuto essere descritta in millanta altri modi). Questo permette per esempio di sistemare una questione (meno oziosa di quanto la giudicherebbe un matematico ingenuo) come quella se $\sin(x)^2$ e $1 - \cos(y)^2$ siano o no la stessa cosa. Sono due descrizioni diverse, mentre le funzioni $\lambda x[\sin(x)^2]$ e $\lambda x[1 - \cos(y)^2]$ sono uguali. λx si chiama *operatore di astrazione* perché permette di ottenere un *concetto* (una funzione) astraendo dal particolare veicolo linguistico (italiano, inglese, programma C, etc.) adottato per descriverlo.

Una volta stabilito che $\lambda x.[\dots x \dots]$ è una funzione in x definita in D è naturale denotare il suo valore in $m \in D$ con $\lambda x.[\dots x \dots]m$, e ha senso allora scrivere

$$\lambda x[1 - \cos(x)^2]\pi = 0$$

Il meccanismo viene esteso facilmente al caso di funzioni ad n posti. Possiamo allora ridefinire la notazione 32 ponendo

$$\varphi_{\mathbf{x}} =_{\text{df}} \lambda \mathbf{ab}[\mathbf{x}]$$

36 Virgolette Poniamo $\mathbf{u} =$

```

vrg(){
    char r1[]="printf(\"%s\", \"a\"); stampa a; per ";
    char r2[]=" devi scrivere printf(\"%s\",a);";
    strcat(c,r1);
    strcat(c,a);
    strcat(c,r2);
}

```

Abbiamo

```

main(){
scanf("%s",a);
vrg();
printf("%s\n",c);
}

```

u

```
481 ~/EsC ---> a.out
```

a

```
printf("%s","a"); stampa a; per a devi scrivere printf("%s",a);
```

```
482 ~/EsC ---> a.out
```

ciao

```
printf("%s","a"); stampa a; per ciao devi scrivere printf("%s",a);
```

e possiamo concludere che la funzione

$$\varphi_u() = \lambda z[\text{printf}("%s", "a") \text{ stampa a; per z ci vuole printf}("%s", a)];$$

assume i seguenti valori:

$$\varphi_u(a) = \text{printf}("%s", "a") \text{ stampa a; per a ci vuole printf}("%s", a);$$

$$\varphi_u(\text{ciao}) = \text{printf}("%s", "a") \text{ stampa a; per ciao ci vuole printf}("%s", a);$$

37 Premessa Con un lavoro un pò noioso abbiamo predisposto gli strumenti per la dimostrazione di tre teoremi importanti, sia perché dicono alcune cose che non si possono fare con un computer, sia perché mostrano il modo di farne certe altre, le quali, a loro volta, possono essere sia utili che no.

I risultati negativi si basano sul far vedere che certe contraddizioni del linguaggio naturale si ritrovano in ambito informatico. Tra queste contraddizioni, quella storicamente piú antica viene da una frase di una Lettera di San Paolo

un loro profeta disse che i cretesi mentono sempre

Il problema viene dal fatto che secondo quell'Autore i profeti dicono il vero. L'esempio della prossima riga è piú cogente

questa frase è falsa

Il meccanismo del paradosso consiste nell'usare un auto-riferimento per negare quello che si sta dicendo. Per creare un auto-riferimento abbiamo bisogno di un meccanismo per sostituire una variabile che figura in un'espressione con l'espressione stessa.

Una seconda premessa. Non si può dire che il numero 9^{9^9} non esiste solo perché non basterebbe tutta la materia dell'universo per scriverlo. Del pari, non si può dire che una funzione non è calcolabile da un computer solo perché non c'è abbastanza memoria. Per questo, molti risultati negativi nella forma *non esiste un programma C che ...* che vedremo vanno intesi come riferiti a un C astratto in cui c'è almeno una dichiarazione `int a[M]` non accompagnata dalla direttiva `# define M ...`

38 Lemma della Sostituzione Diagonale (Variante del risultato noto in Letteratura come teorema *s-m-n*) Esiste una funzione di stringa s-calcolabile $\sigma^{(1)}$ tale che per ogni $\varphi_x^{(2)}(y, z)$ e per ogni x, u si ha

$$\sigma(x) = u \quad \text{implica} \quad \varphi_u(z) = \varphi_x(x, z)$$

Dimostrazione. Cominciamo col definire ed eseguire il programma

```
main(){
    scanf("%s%s", a, b);
    subst();
    printf("%s", c);
}
subst(){
    strcpy(c, b);
    strcat(c, "_(){\n    strcpy(b, a);\n    strcpy(a, \"\");
    printf("%s\n", c);
    strcat(c, a);
    printf("%s\n", c);
    strcat(c, "\");\n    ");
    printf("%s\n", c);
    strcat(c, b);
    strcat(c, "();\n}\n");
    strcat(c, a);
```

```

    strcat(c, "\n");
}
423 ~/EsC ---> a.out
fn(){...} fn                a=fn(){...}      b=fn
fn_(){                      c dopo la prima strcat
    strcpy(b,a);
    strcpy(a,"
fn_(){                      c dopo la seconda
    strcpy(b,a);
    strcpy(a,"fn(){...}
fn_(){                      c dopo la terza (non si vedono
    strcpy(b,a);          i quattro spazi di indent
    strcpy(a,"fn(){...}"); dopo l'accapo)

fn_(){                      c quando si esce dalla funzione
    strcpy(b,a);
    strcpy(a,"fn(){...}");
    fn();
}
fn(){...}

```

Riscrivendo il programma senza le stampe di *debugging* otteniamo

```

char a[M], b[M], c[M];
main(){
    scanf("%s%s", a, b);
    subst();
    printf("%s", c);
}
subst(){
    strcpy(c, b);
    strcat(c, "_(){\n    strcpy(b, a);\n    strcpy(a, \"\");
    strcat(c, a);
    strcat(c, \"\");\n    ");
    strcat(c, b);
    strcat(c, "();\n}\n");
    strcat(c, a);
    strcat(c, "\n");
}

```

```

426 ~/EsC ---> a.out
fn(){...} fn          input
fn_(){                output
    strcpy(b,a);
    strcpy(a,"fn(){...}");
    fn();
}
fn(){...}

```

Possiamo concludere che la funzione `subst` s-calcola la funzione τ

$$\tau(\mathbf{x}, \mathbf{fn}) = \mathbf{u} \quad \text{nonché} \quad \mathbf{x} = \mathbf{fn}\{\dots\} \quad \text{implica} \quad \varphi_{\mathbf{u}}(\mathbf{a}) = \text{varphi}_{\mathbf{x}}(\mathbf{x}, \mathbf{a})$$

Infatti `subst` assegna `c` con la concatenazione di una nuova funzione

```
fn_(){...}
```

con la funzione data `fn(){...}`. L'identificatore della nuova funzione è quello della vecchia seguito da `_`; nel corpo della nuova funzione

- (1) si trasferisce `a` in `b`;
- (2) si trasferisce l'intera definizione della vecchia funzione in `a`;
- (3) si chiama la vecchia funzione per input i nuovi valori di `a` e di `b`.

Per concludere la dimostrazione rimane da sostituire la lettura dell'identificatore da `stdin` con la funzione `hd_lt_par`, ponendo

```

dgsb(){
    hd_lt_par();
    strcpy(b,c);
    strcat(c,"_(){\n    strcpy(b,a);\n    strcpy(a,\"\");
    strcat(c,a);
    strcat(c,\"\");\n    ");
    strcat(c,b);
    strcat(c,"();\n}\n");
    strcat(c,a);
    strcat(c,\"\n");
}

```

La funzione `dgsb` s-calcola la funzione σ perché se $\mathbf{x} = \mathbf{fn}\{\dots\}$ è assegnata ad `a` abbiamo che `dgsb` per input `a` si comporta come `subst` per input `a` e `fn`.

39 Teorema della ricorsione (Anche: Teorema di Kleene) Per ogni $\varphi_x^{(2)}(y, z)$ esiste un *punto fisso* u tale che

$$\varphi_u(z) = \varphi_x(u, z).$$

Dimostrazione. Dato $x = \text{fn}()\{\dots\}$ definiamo una nuova funzione con la stringa $x0=$

```
fn0(){
    dgsb();
    strcpy(a, c);
fn();
}
dgsb(){...}
x
```

Abbiamo

$$\varphi_{x0}(y, z) = \varphi_x(\sigma(y), z) \tag{A}$$

perché $x0$, se a è assegnata con y e b con z

(1) chiama `dgsb` la quale assegna c con $\sigma(y)$;

(2) avendo assegnato a con $\sigma(y)$ (via `strcpy`) ed avendo lasciato $b=z$ chiama `fn` col risultato che c viene assegnato con

$$\varphi_x(\sigma(y), z)$$

A questo punto possiamo definire la stringa u menzionata nell'asserto

$$u = \sigma(x0) \tag{B}$$

e verificare che si comporta nel modo promesso.

$$\varphi_u(z) = \varphi_{\sigma(x0)}(z) = \varphi_{x0}(x0, z) = \varphi_x(\sigma(x0), z) = \varphi_x(u, z)$$

dove dobbiamo la prima uguaglianza alla definizione (B) e la seconda al Lemma della sostituzione diagonale; e dove le ultime due uguaglianze vengono rispettivamente dall'equazione (A) e ancora dalla definizione (B).

40 Corollario Esistono funzioni C che danno in uscita la loro propria definizione.

Dimostrazione. Si applichi il teorema alla funzione

```

mm(){
strcpy(b,a);
strcpy(c,a);
printf("%s",c);
}

```

41 Commento Questo corollario dà luogo a considerazioni di carattere speculativo sulle macchine che *riproducono sé stesse*. Inoltre è il fondamento teorico sia di fenomeni negativi (*virus* informatici) sia positivi (*booting* nei sistemi operativi). Da questo punto di vista è utile osservare che una funzione che riproduce se stesso può essere ben più complicata di quella usata nella dimo precedente: essa può dapprima compiere varie azioni sul contenuto di registri, dischi, etc., per poi riprodursi solo in un secondo momento.

42 Funzione universale È un fatto che esiste una funzione C univ tale che per ogni definizione x di una funzione unaria ed ogni y si ha

$$\varphi_{univ}(x, y) = \varphi_x(y).$$

Essa rispetta il nostro standard di calcolo.

43 Teorema (Forma forte del teorema di Kleene, anche Teorema di Rogers) Per ogni definizione x di una funzione unaria $\varphi_x(y)$ esiste un valore v tale che per ogni z si ha

$$\varphi_{\varphi_x(v)} = \varphi_v$$

Dimostrazione. Consideriamo la funzione definita dalla stringa $w=$

```

fn(){
x();
strcpy(c,a);
univ();
}

```

abbiamo

$$\varphi_w(y, z) = \varphi_{univ}(\varphi_x(y), z) \tag{C}$$

Infatti w , se y , z sono assegnate risp. ad a, b

(1) chiama x con $a=y$ la quale assegna $\varphi_x(y)$ a c ;

(2) assegna questo valore a \mathbf{a} e lo passa con \mathbf{z} ad \mathbf{univ} .

L'asserto segue scegliendo come \mathbf{v} il punto fisso che si ottiene applicando il teorema di Kleene con $\mathbf{x} = \mathbf{w}$. In questo modo si ha infatti

$$\varphi_{\mathbf{v}}(\mathbf{z}) = \varphi_{\mathbf{w}}(\mathbf{v}, \mathbf{z}) = \varphi_{\mathbf{univ}}(\varphi_{\mathbf{x}}(\mathbf{v}), \mathbf{z}) = \varphi_{\varphi_{\mathbf{x}}(\mathbf{v})}(\mathbf{z})$$

dove la prima uguaglianza segue dal teorema di Kleene, la seconda dall'equazione (C) e l'ultima dalla definizione di \mathbf{univ} .

Un gran numero di risultati negativi sull'informatica è riassunto dal prossimo teorema.

44 Definizione Una classe \mathcal{A} è (*in-*)*decidibile* se (non) esiste una funzione $\varphi_{\mathbf{x}}$ s-calcolabile tale che

$$\varphi_{\mathbf{x}}(\mathbf{z}) = \begin{cases} \text{"0"} & \text{qualora } \mathbf{z} \in \mathcal{A} \\ \text{"1"} & \text{qualora } \mathbf{z} \notin \mathcal{A}. \end{cases}$$

Diciamo che una classe di funzioni s-calcolabili è *non banale* se non è vuota e se non coincide con l'intera classe delle funzioni s-calcolabili.

45 Teorema Tutte le classi non banali di funzioni s-calcolabili sono indecidibili.

Dimostrazione. Ammettiamo (ad absurdum) che per la classe di funzioni \mathcal{A} si abbia

$$\varphi_{\mathbf{x}}(\mathbf{z}) = \text{"0"} \text{ se } \varphi_{\mathbf{z}} \in \mathcal{A}; \quad \varphi_{\mathbf{x}}(\mathbf{z}) = \text{"0"} \text{ se } \varphi_{\mathbf{y}} \notin \mathcal{A}.$$

Poiché \mathcal{A} non è banale, esistono le stringhe \mathbf{h}, \mathbf{k} tali che

$$\varphi_{\mathbf{h}} \in \mathcal{A}; \quad \varphi_{\mathbf{k}} \notin \mathcal{A}. \tag{D}$$

La forma di \mathbf{x} sia $\mathbf{fn}()\{\dots\}$. Ponendo $\mathbf{y} =$

```
fn_(){
    fn();
    if strcmp(c,"0")
        then strcpy(c,"k");
    else strcpy(c,"m");
}
x
```

si ha

$$\varphi_y(\mathbf{z}) = k \text{ se } \varphi_z \in \mathcal{A}; \quad \varphi_y(\mathbf{z}) = m \text{ se } \varphi_z \notin \mathcal{A}. \quad (\text{E})$$

Sia u la stringa fornita dal teorema di Rogers tale che

$$\varphi_u = \varphi_{\varphi_y(u)} \quad (\text{F})$$

Otteniamo la contraddizione seguente

$\varphi_u \in \mathcal{A}$	\Rightarrow	$\varphi_y(u) = k$	equazione (E)
	\Rightarrow	$\varphi_u = \varphi_k$	equazione (F)
	\Rightarrow	$\varphi_u \notin \mathcal{A}$	equazione (D)
$\varphi_u \notin \mathcal{A}$	\Rightarrow	$\varphi_y(u) = m$	equazione (E)
	\Rightarrow	$\varphi_u = \varphi_m$	equazione (F)
	\Rightarrow	$\varphi_u \in \mathcal{A}$	equazione (D).

46 Corollario Sono ad esempio indecidibili le classi:

delle funzioni totali (ossia definite in tutto $\Sigma^{(n)}$);

delle funzioni definite per un dato input (halting problem: programmi che danno un output per un dato input)

delle funzioni che hanno un dato valore nel loro codominio (per esempio: programmi che stampano almeno una volta zero);

delle funzioni costanti.

sono perfino indecidibili le classi dei programmi che danno un output identicamente uguale a zero, a uno, etc.

7 Svolgimento degli esercizi del Kernighan e Ritchie, Cap. 1

INDICE

CelsFahr.c	1.3	1.4	1.5
ciao.c	1.1	1.2	
ClEnd.c	1.18		
cntblnk.c	1.8		
cp1wpl.c	1.12		
FahrFnct.c	1.15		
getcharEOF.c	1.6		

```

invisibili.c      1.10
IstOriz.c        1.12
IstVert.c        1.13
lineelunghe.c    1.17
longest20.c      1.16
longestline.c    sect. 1.9 del libro
prime.c          tavola numeri primi
pwrstr.c         power rozzo
reverse.c        1.19
shrinkblanks.c  1.9
sqrt.c           square root
valorediEOF.c   1.7
wco.c           1.11

```

```

for i in *; do
cat $i >> esercizi
done for i in *; do
cat $i
done

```

```

/* Es. 1.3 1.4 1.5
*  Intestazione
*  -----
*  inserire
*  printf("Fahr Celsius\n");
*  e aggiungere 2 spazi nell'altra printf
*  printf("%3.0f  %6.1f\n");
*  Tabella inversa
*  -----
*  ...
*  celsius=lower;
*  while (celsius <= upper){
*      fahr = (9.0*celsius) / 5.0 + 32.0;
*      printf ...
*      celsius = celsius + step;

```

```

*   }
*
*   A scendere
*   -----
*   for(fahr = 300; fahr >= 0; fahr = fahr - 20)

*/
/* ciao.c con prove ASCII-int-char es. 1.1 1.2 */

#include <stdio.h>

main()
{
char a;
a=10;
printf("ciao");
printf("/*ciao.c*/");
printf("%c",a);
putchar(65);
putchar(a);
}

/*
540 ~/Esc ---->  ciao.bin
ciao                ASCII 10 = \n
A                   ASCII 65
541 ~/Esc ---->

*/
/* CleanEnd.c  es. 1.18
*  elimina le righe vuote
*  e quelle che lo divengono dopo eliminazione
*  di blank e \t  a destra
*/

#include <stdio.h>
#define MAXLINE 1000

```

```

int getline(char line[], int maxline);

main()
{
    char line[MAXLINE];
    while (getline(line, MAXLINE) > 0)
        if (sopprimi(line) > 0)
            printf("%s",line);
}

/* sopprime blank e tab in fine riga */

int sopprimi(char s[]){
    int i = 0;

    while(s[i] != '\n')
        ++i;          /* cerca \n e misura lunghezza riga */
    --i;
    while ( i >= 0 && (s[i] == '\t' || s[i] == ' '))
        --i;
    if (i >= 0){
        ++i;
        s[i]='\n';
        ++i;
        s[i]='\0';
    }
    return i;
}

int getline(char s[], int lim)
{
    int c, i, j;

    j = 0;
    for (i=0; (c = getchar()) != EOF && c != '\n'; ++i)
        if (i < lim - 2){

```

```

        s[j] = c;
        ++j;
    }
    if (c == '\n'){
        s[j] = c;
        ++j;
        ++i;
    }
    s[j] = '\0';
    return i;
}

```

```

/*
561 ~/EsC ----> ClEnd.bin
abc

```

de

f

```

<return><^D>
abc
de
f
562 ~/EsC ---->

```

che dopo abc e dopo de ci fossero dei blank e \t
bisogna crederlo sulla parola
dopo f non ho messo niente
per verificare che nella if di sopprimi
ci vuole il >= e non basta il >
*/
/* cntblnk.c es. 1.8
* conta gli spazi (ns) i tab (nt) e i fine riga (nf)
*/

```

#include <stdio.h>

main()
{
    int c,ns,nt,nf;
    ns = 0;
    nt = 0;
    nf = 0;
    while ((c = getchar()) != EOF){
        if (c==' ') ++ns;
        if (c=='\t') ++nt;
        if (c=='\n') ++nf;
    }
    printf("%d spazi %d tab %d endl\n",ns,nt,nf);
}

```

```

/*
528 ~/EsC ---> gcc -o cntblnk.bin cntblnk.c
529 ~/EsC ---> 755 cntblnk.bin
530 ~/EsC ---> cntblnk
-su: cntblnk: command not found
531 ~/EsC ---> cntblnk.bin
a b c
d e f g h
k <return> <^D>
13 spazi 1 tab 3 endl
532 ~/EsC --->
*/
/* cp1wpl.c es. 1.12
* copy 1 word per line
*/

```

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

```
# define IN 1
```

```

# define OUT 0

main()
{
    int c,state;
    state = OUT;
    while (( c= getchar()) != EOF){
        if (c == ' ' || c == '\t' || c == '\n'){
            if (state == IN){
                putchar('\n');
                state = OUT;
            }
        }
        else if (state == OUT){
            state = IN;
            putchar(c);
        }
        else
            putchar(c);
    }
}

/* si comincia con state=OUT perche' il testo
* potrebbe iniziarsi con degli spazi
* la I if chiede se c e' un separatore
* la II se c e' alla fine di una parola
* se si' a capo e aggiorna state
* se no c segue un altro separatore
* si esce dalla I if non si entra nella I else
* si ripete la while con un nuovo c e questo
* c non ha effetti
* se c non e' un separatore siamo all'inizio o dentro
* una parola ; se l'ultima if dice che inizio
* si aggiorna state; in ogni caso si scrive c

```

```

514 ~/EsC ---> cp1wpl.bin
    silvia ricordi ancora<^D>silvia
ricordi
ancora<^D>515 ~/EsC ---> cp1wpl.bin
silvia ricordi ancora <^D> silvia
ricordi
ancora <return><^D>

516 ~/EsC ---> cp1wpl.bin
    silvia ricordi ancora <return>
silvia
ricordi
ancora
<^D>517 ~/EsC --->

*/
/* FahrFnct.c es. 1.15
* Fahrenheit usando una funzione
*/
#include <stdio.h>

float celsius(float fahr);

main()
{
    float fahr;
    int min, max, step;
    min = 0;
    max = 300;
    step = 20;
    fahr = min;
    while (fahr <= max) {
        printf ("%3.0f %6.1f\n",fahr,celsius(fahr));
        fahr = fahr + step;
    }
}

float celsius(float fahr)

```

```

{
    return (5.0/9.0) * (fahr - 32.0);
}

/*

534 ~/EsC ---> FahrFnct.bin
    0  -17.8
   20  -6.7
   40   4.4
   60  15.6
   80  26.7
  100  37.8
  120  48.9
  140  60.0
  160  71.1
  180  82.2
  200  93.3
  220 104.4
  240 115.6
  260 126.7
  280 137.8
  300 148.9

*/
/* getcharEOF.c  es 1.6
* legge da stdin e scrive 1 se soddisfatta
* la condizione
* c = getchar() != EOF
* equivalente a
* c = (getchar() != EOF)
* EOF via <^D> dopo <return>
*/
# include <stdio.h>

main()

```

```

{

int c;

while (c = getchar() != EOF)
    printf("%d\n",c);
printf("%d ecco end of file \n",c);
}

/*
536 ~/EsC ---> gcc -o getcharEOF.bin getcharEOF.c
537 ~/EsC ---> 755 getcharEOF.bin
538 ~/EsC --->  getcharEOF.bin
a          un car + <return>
1
1          due uni
a b c      3 car + 2 bl + <return>
1
1
1
1
1
1
1          sei uni <^D>
0 ecco end of file
539 ~/EsC --->
*/

/* invisibili.c es. 1.10
* esibisce i \t \n \\
* esercizio mal posto rispetto gli accapo
* senza un accapo in piu' nella printf
* endless loop
*/

# include <stdio.h>

main()

```

```

{
    int c;
    while ((c = getchar()) != EOF){
        if (c == '\t')
            printf("\\t");
        if (c == '\n')
            printf("\\n\\n");
        if (c == '\\')
            printf("\\\\");
        if (c != '\t')
            if (c != '\n')
                if (c != '\\')
                    putchar(c);
    }
}

/*
551 ~/EsC ---> gcc -o invisibili.bin invisibili.c
*          senza il secondo \n
*          non va a capo e non accetta <^D>
552 ~/EsC ---> invisibili.bi
a b      c
a b\tc\ndef
def\n553 ~/EsC --->
553 ~/EsC ---> invisibili.bin
a        b
a\tb\n
\n
\n
\t\n554 ~/EsC --->
*          arrestato con <^C>
554 ~/EsC ---> gcc -o invisibili.bin invisibili.c
*          corretto col secondo \n
555 ~/EsC ---> invisibili.bin
a        t \
a\tt \\n
a        c \

```

```

a\tc \\ \n <^D>
556 ~/EsC --->
*/

/* Ist0riz.c es. 1.12
* produce un istogramma delle frequenze
* delle parole di lunghezza <= WORDMAX
* con righe di lunghezza < ISTMAX
* un asterisco per numero_parole_lunghezza/ISTMAX
* almeno uno
*/

#include <stdio.h>
#define IN 1
#define OUT 0
#define ISTMAX 15
#define WORDMAX 11

main(){
int c, nc, i, stato;
int lung;          /* lunghezza della barra */
int valmax;       /* massimo valore di lw[] */
int sballano;     /* quante con piu' di WORDMAX lettere */
int wl[WORDMAX]; /* quante di lunghezza wl[nc] */

stato = OUT;
nc = 0;          /* lunghezza parole scandita */
sballano = 0;
for (i = 0; i < WORDMAX; i++)
    wl[i] = 0;
while ((c = getchar()) != EOF) {
    if (c == ' ' || c == '\t' || c == '\n') {
        stato = OUT;
        if (nc > 0)
            if (nc < WORDMAX)
                ++wl[nc];
            else ++sballano;
        nc = 0;
    }
}
}

```

```

    } else if (stato == OUT) {
        stato = IN;
        nc = 1;          /* comincia nuova parola */
    } else
        ++nc;          /* stessa parola */
}
valmax = 0;
for (i = 1; i < WORDMAX; ++i)
    if (wl[i] > valmax)
        valmax = wl[i];
for (i = 1; i < WORDMAX; ++i) {
    printf("%5d - %5d", i, wl[i]);
    if (wl[i] > 0){
        if ((lung = (wl[i]*ISTMAX/valmax)) <= 1)
            lung = 1;
    } else lung = 0;
    while (lung > 0) {
        putchar('*');
        --lung;
    }
    putchar('\n');
}
if (sballano > 0)
    printf("ci sono %d parole con piu\' di %d lettere\n", sballano,WORDMAX);
}

```

```

/*
515 ~ ---> Ist0riz.bin

```

Silvia, rimembri ancora
 quel tempo della tua vita mortale,
 quando belt splendea
 negli occhi tuoi ridenti e fuggitivi,
 e tu, lieta e pensosa, il limitare
 di giovent salivi?

Sonavan le quiete

stanze, e le vie dintorno,
al tuo perpetuo canto,
allor che all'opre femminili intenta
sedevi, assai contenta
di quel vago avvenir che in mente avevi.
Era il maggio odoroso: e tu solevi
cos menare il giorno.

Io gli studi leggiadri
talor lasciando e le sudate carte,
ove il tempo mio primo
e di me si spendea la miglior parte,
d'in su i veroni del paterno ostello
porgea gli orecchi al suon della tua voce,
ed alla man veloce
che percorrea la faticosa tela.
Mirava il ciel sereno,
le vie dorate e gli orti,
e quinci il mar da lungi, e quindi il monte.
Lingua mortal non dice
quel ch'io sentiva in seno.

Che pensieri soavi,
che speranze, che cori, o Silvia mia!
Quale allor ci apparia
la vita umana e il fato!
Quando sovviemmi di cotanta speme,
un affetto mi preme
acerbo e sconcolato,
e tornami a doler di mia sventura.
O natura, o natura,
perch non rendi poi
quel che prometti allor? perch di tanto
inganni i figli tuoi?

Tu pria che l'erbe inaridisse il verno,
da chiuso morbo combattuta e vinta,
perivi, o tenerella. E non vedevi

il fior degli anni tuoi;
non ti molceva il core
la dolce lode or delle negre chiome,
or degli sguardi innamorati e schivi;
n teco le compagne ai d festivi
ragionavan d'amore.

Anche peria tra poco
la speranza mia dolce: agli anni miei
anche negaro i fati
la giovanezza. Ahi come,
come passata sei,
cara compagna dell'et mia nova,
mia lacrimata speme!
Questo quel mondo? questi
i dilette, l'amor, l'opre, gli eventi
onde cotanto ragionammo insieme?
questa la sorte dell'umane genti?
All'apparir del vero
tu, misera, cadesti: e con la mano
la fredda morte ed una tomba ignuda
mostravi di lontano.

```
1 - 26*****
2 - 54*****
3 - 41*****
4 - 32*****
5 - 45*****
6 - 45*****
7 - 36*****
8 - 21*****
9 - 11***
10 - 8**
```

ci sono 3 parole con piu' di 11 lettere

516 ~ --->

*/

/* Ist0riz.c es. 1.13

* produce un istogramma verticale delle frequenze

* delle parole di lunghezza <= WORDMAX

```

* con righe di lunghezza < ISTMAX
* un asterisco per numero_parole_lunghezza/ISTMAX
* almeno uno
*/

#include <stdio.h>
#define IN 1
#define OUT 0
#define ISTMAX 15
#define WORDMAX 11

main(){
int c, nc, i, j, stato;
int lung;          /* lunghezza della barra */
int valmax;       /* massimo valore di lw[] */
int sballano;     /* quante con piu' di WORDMAX lettere */
int wl[WORDMAX]; /* quante di lunghezza wl[nc] */

stato = OUT;
nc = 0;          /* lunghezza parole scandita */
sballano = 0;
for (i = 0; i < WORDMAX; i++)
    wl[i] = 0;
while ((c = getchar()) != EOF) {
    if (c == ' ' || c == '\t' || c == '\n') {
        stato = OUT;
        if (nc > 0)
            if (nc < WORDMAX)
                ++wl[nc];
            else ++sballano;
        nc = 0;
    } else if (stato == OUT) {
        stato = IN;
        nc = 1;      /* comincia nuova parola */
    } else
        ++nc;      /* stessa parola */
}
valmax = 0;

```

```

for (i = 1; i < WORDMAX; ++i)
    if (wl[i] > valmax)
        valmax = wl[i];
for (i = ISTMAX; i > 0; --i) {
    for (j = 1; j < WORDMAX; ++j)
        if ((wl[j] * ISTMAX / valmax) >= i)
            printf(" * ");
        else
            printf("    ");
        putchar('\n');
        putchar('\b');
}
for (i = 1; i < WORDMAX; ++i)
    printf("%4d ",i);
putchar('\n');
putchar('\b');
for (i = 1; i < WORDMAX; ++i)
    printf("%4d ",wl[i]);
putchar('\n');
if (sballano > 0)
    printf("ci sono %d parole con piu\' di %d lettere\n", sballano,WORDMAX);
}

```

```

/*
541 ~/EsC ---> /root/EsC/IstOriz.bin
Sempre caro mi fu quest'ermo colle,
e questa siepe, che da tanta parte
dell'ultimo orizzonte il guardo esclude.
Ma sedendo e mirando, interminati
spazi di l da quella, e sovrumani
silenzi, e profondissima quiete
io nel pensier mi fingo; ove per poco
il cor non si spaura. E come il vento
odo stormir tra queste piante, io quello
infinito silenzio a questa voce
vo comparando: e mi sovvien l'eterno,
e le morte stagioni, e la presente

```

e viva, e il suon di lei. Cos tra questa
immensit s'annega il pensier mio:
e il naufragar m' dolce in questo mare.

- 1 - 12*****
- 2 - 22*****
- 3 - 10*****
- 4 - 8*****
- 5 - 9*****
- 6 - 12*****
- 7 - 8*****
- 8 - 7****
- 9 - 5***
- 10 - 2*

ci sono 4 parole con piu' di 11 lettere

542 ~/EsC ---> IstVert.bin

Sempre caro mi fu quest'ermo colle,
e questa siepe, che da tanta parte
dell'ultimo orizzonte il guardo esclude.
Ma sedendo e mirando, interminati
spazi di l da quella, e sovrumani
silenzi, e profondissima quiete
io nel pensier mi fingo; ove per poco
il cor non si spaura. E come il vento
odo stormir tra queste piante, io quello
infinito silenzio a questa voce
vo comparando: e mi sovvien l'eterno,
e le morte stagioni, e la presente
e viva, e il suon di lei. Cos tra questa
immensit s'annega il pensier mio:
e il naufragar m' dolce in questo mare.

*
*
*
*
*
*
*

```

*      *                *
*      *                *
*      *      *        *      *
*      *      *      *      *      *
*      *      *      *      *      *      *
*      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *
1      2      3      4      5      6      7      8      9      10
12     22     10     8      9      12     8      7      5      2

```

ci sono 4 parole con piu' di 11 lettere

543 ~/EsC --->

// lineelunghe.c es. 1.17

* stampa solo linee lunghe piu' di MAX

* getline come in longest20.c

*/

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

```
#define MAXLINE 1000
```

```
#define MAX 30
```

```
int getline(char line[], int maxline);
```

```
main()
```

```
{
```

```
    int len;
```

```
    char line[MAXLINE];
```

```
    while ((len = getline(line,MAXLINE)) > 0)
```

```
        if (len > MAX)
```

```
            printf("%s\n",line);
```

```
    return 0;
```

```
}
```

```
int getline(char s[], int lim)
```

```
{
```

```
    int c, i, j;
```

```

    j = 0;
    for (i=0; (c = getchar()) != EOF && c != '\n'; ++i)
        if (i < lim - 2){
            s[j] = c;
            ++j;
        }
        if (c == '\n'){
            s[j] = c;
            ++j;
            ++i;
        }
    s[j] = '\0';
    return i;
}

```

```

# if 0
540 ~/EsC ---> gcc lineelunghe.c
541 ~/EsC ---> a.out
Silvia rimembri ancora
quel tempo della tua vita mortale
quando belta' splendea <return><^D>
quel tempo della tua vita mortale

```

```

#endif
/* longest20.c es. 1.16
* come longestline ma
* MAXLINE=20
* se linea troppo lunga stampa quanto e' lunga
* e i primi MAXLINE-2 caratteri
* in main cambia solo la stampa
* in getline si aggiunge j
* che si ferma quando si arriva a MAXLINE -1
* per poter aggiungere \0 e stampare

```

```

*/

#include <stdio.h>
#define MAXLINE 20

int getline(char line[], int maxline);
void copy(char to[], char from[]);

main()
{
    int len, max;
    char line[MAXLINE];
    char longest[MAXLINE];

    max = 0;
    while ((len = getline(line,MAXLINE)) > 0) {
        if (len > max){
            max = len;
            copy(longest,line);
        }
    }
    if (max > 0)
        printf("%d %s\n", max, longest);
    return 0;
}

int getline(char s[], int lim)
{
    int c, i, j;

    j = 0;
    for (i=0; (c = getchar()) != EOF && c != '\n'; ++i)
        if (i < lim - 1){
            s[j] = c;
            ++j;
        }
    if (c == '\n'){
        s[j] = c;

```

```

        ++j;
        ++i;
    }
    s[j] = '\0';
    return i;
}

void copy(char to[], char from[])
{
    int i;
    i = 0;
    while((to[i] = from[i]) != '\0')
        ++i;
}

/*
523 ~/EsC ---> gcc longest20.c
524 ~/EsC ---> a.out
1234567890123456789012
23 1234567890123456789

525 ~/EsC ---> a.out
abcd
ab
abcde
6 abcde

582 ~/EsC --->
*/
/* longestline.c
* ripreso dal libro sect. 1.9
*
*/

```

```

#include <stdio.h>
#define MAXLINE 1000

int getline(char line[], int maxline);
void copy(char to[], char from[]);

main()
{
    int len, max;
    char line[MAXLINE];
    char longest[MAXLINE];

    max = 0;
    while ((len = getline(line,MAXLINE)) > 0)
        if (len > max){
            max = len;
            copy(longest,line);
        }

    if (max > 0)
        printf("%s", longest);
    return 0;
}

int getline(char s[], int lim)
{
    int c, i;

    for (i=0; i < lim-1 && (c=getchar())!=EOF && c!='\n';++i)
        s[i] = c;
    if (c == '\n'){
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return i;
}

```

```

void copy(char to[], char from[])
{
    int i;
    i = 0;
    while((to[i] = from[i]) != '0')
        ++i;
}

/*
546 ~/EsC ---> longestline.bin
Sempre caro mi fu quest'ermo colle,
e questa siepe, che da tanta parte
dell'ultimo orizzonte il guardo esclude.
Ma sedendo e mirando, interminati
spazi di l da quella, e sovrumani
silenzi, e profondissima quiete
io nel pensier mi fingo; ove per poco
il cor non si spaura. E come il vento
odo stormir tra queste piante, io quello
infinito silenzio a questa voce
vo comparando: e mi sovvien l'eterno,
e le morte stagioni, e la presente
e viva, e il suon di lei. Cos tra questa
immensit s'annega il pensier mio:
e il naufragar m' dolce in questo mare.
e viva, e il suon di lei. Cos tra questa
547 ~/EsC --->
*/
#include <stdio.h>

int i, j, k;
int l = 7;
main(){
    for(i = 3;i <= 999;i += 2){
        for(j=2;j*j<=i;j++);
        /*printf("la radice di %d e' %d\n",i,j);*/
        for(k=3;k<j && i % k;k += 2 );
            if (k >= j){

```

```

        printf("%3d  ",i);
        if (l == 0){
            l = 7;
            printf("\n",i);
        }
        else --l;
    }
}
printf("\n",i);
}

```

```

/*
for(i=2;i*i<=n;i++);
printf("%d\n",--i);
*/
/* pwrstr.c
* o famo strano, senza tante fisime sui tipi
*/

# include <stdio.h>

int power(int,int);
int power2(int,int);
main(){
int i=2;
printf("%d %d %d\n",power(i,3),power1(i,4),power2(i,5));
}
int power(int base,int n)      /* canonica */
{
int i,p;
p=1;
for(i=1;i<=n;i++)
p=p*base;
return p;
}
int power1(int base,int n)     /* senza prototipo */

```

```

{
int i,p;
p=1;
for(i=1;i<=n;i++)
p=p*base;
return p;
}
power2(base,n)          /* senza ribadire i tipi */
{
int i,p;
p=1;
for(i=1;i<=n;i++)
p=p*base;
return p;
}

/*
511 ~/EsC ---> gcc -o pwrstr.bin pwrstr.c
512 ~/EsC ---> pwrstr.bin
8 16 32
513 ~/EsC --->
*//* reverse.c    es. 1.19
*/

#include <stdio.h>
#define MAXLINE 1000

int getline(char line[], int maxline);
void rev(char s[]);
main()
{
    char line[MAXLINE];
    while (getline(line, MAXLINE) > 0){
        rev(line);
        printf("%s",line);
    }
}

```

```

/* rovescia la stringa s */

void rev(char s[]){
    int i, j;
    char temp;
    i = 0;

    while(s[i] != '\0')
        ++i;          /* cerca \n e misura lunghezza riga */
    --i;
    if (s[i] == '\n')
        --i;
    j = 0;
    while ( j < i){
        temp = s[j];
        s[j] = s[i];
        s[i] = temp;
        --i;
        ++j;
    }
}

int getline(char s[], int lim)
{
    int c, i, j;

    j = 0;
    for (i=0; (c = getchar()) != EOF && c != '\n'; ++i)
        if (i < lim - 2){
            s[j] = c;
            ++j;
        }
    if (c == '\n'){
        s[j] = c;
        ++j;
        ++i;
    }
}

```

```

    }
    s[j] = '\0';
    return i;
}

```

```
/*
```

```

574 ~/EsC ---> reverse.bin <shift>+<Ins>
Sempre caro mi fu quest'ermo colle,
e questa siepe, che da tanta parte
dell'ultimo orizzonte il guardo esclude.
Ma sedendo e mirando, interminati
spazi di l da quella, e sovrumani
silenzi, e profondissima quiete
io nel pensier mi fingo; ove per poco
il cor non si spaura. E come il vento
odo stormir tra queste piante, io quello
infinito silenzio a questa voce
vo comparando: e mi sovvien l'eterno,
e le morte stagioni, e la presente
e viva, e il suon di lei. Cos tra questa
immensit s'annega il pensier mio:
e il naufragar m' dolce in questo mare. <^D>
,elloc omre'tseuq uf im orac erpmeS
etrap atnat ad ehc ,epeis atseuq e
.edulcse odraug li etnozziro omitlu'lled
itanimretni ,odnarim e odnedes aM
inamurvos e ,alleuq ad l id izaps
eteiuq amissidnoforp e ,iznelis
ocop rep evo ;ognif im reisnep len oi
otnev li emoc E .aruaps is non roc li
olleuq oi ,etnaip etseuq art rimrots odo
ecov atseuq a oiznelis otinifni
,onrete'l neivvos im e :odnarapmoc ov
etneserp al e ,inoigats etrom el e
atseuq art soC .iel id nous li e ,aviv e

```

```

:oim reisnep li agenna's tisnemmi
.eram otseuq ni eclod 'm ragarfuan li e
574 ~/EsC --->
*/
/* shrinkblanks.c es. 1.9
* elimina sequenze di n>=2 spazi
* senza if-then-else non ancora spiegata
*/

#include <stdio.h>

#define NONESPACE 'a'

main()
{
    int c, lastc;
    lastc = NONESPACE;
    while ((c=getchar()) != EOF) {
        if (c != ' ')
            putchar(c);
        if (c == ' ')
            if (lastc != ' ')
                putchar(c);
        lastc = c;
    }
}

/*
522 ~/EsC ---> gcc -o shrinkblanks.bin shrinkblanks.c
523 ~/EsC ---> 755 shrinkblanks.bin
524 ~/EsC ---> shrinkblanks.bin
a      b
a b
a  b
a b
c c
c c
c d

```

```

c d
    c d
    c d      halt con <^D>
525 ~/EsC --->
*/
#include <stdio.h>

int i, n;

main(){
    scanf("%d", &n);
    for(i = 2; i*i <= n; ++i);
    printf("parte intera di radice di %d = %d\n",n,--i);
    int j=2;
    while(j * j <= n)
        j++;
    printf("la parte intera della radice di %d e' \
%d\n",n,--j);
}
/*  valorediEOF.c  es. 1.7
*   stampa il valore del numero magico EOF
*   notare che esso e' registrato in stdio.h
*/
#include <stdio.h>

main()
{
    printf("EOF vale %d\n",EOF);
}

/*
532 ~/EsC ---> gcc -o valorediEOF.bin valorediEOF.c
534 ~/EsC ---> 755 valorediEOF.bin
535 ~/EsC --->  valorediEOF.bin
EOF vale -1
536 ~/EsC --->
*//* wco.c  es. 1.11

```

```

* word counting copiato dal libro
*/

# include <stdio.h>

# define IN 1
# define OUT 0

main()
{
    int c,n1,nw,nc,state;
    state=OUT;
    n1=nw=nc=0;
    while ((c=getchar()) != EOF){
        ++nc;
        if (c=='\n')
            ++n1;
        if (c==' '||c=='\t' ||c=='\n')
            state = OUT;
        else if (state == OUT){
            state = IN;
            ++nw;
        }
    }
    printf("%d %d %d\n", n1,nw,nc);
}

/*
560 ~/EsC ---> wco.bin
Silvia rimembri ancora
quel tempo della tua vita mortale
quando belta' splendea
3 12 80
561 ~/EsC ---> wco.bin
*          <return> <^D>
0 0 0
562 ~/EsC ---> wco.bin

```

```
*          <return> <return> <^D>
2 0 2
563 ~/EsC ----> wco.bin
```

```
*          <return> <tab><return> <tab><tab><return> <^D>
3 0 7
565 ~/EsC ----> wco.bin
roma          <return>
bari          <return>
pisa          <return><^D>
3 3 15
568 ~/EsC ----> wco.bin
                multi spazi
1 2 34
*/
```

8 Esercizi numerici

```
// radice quadrata col metodo di Newton
#include<stdio.h>
main(){
float a,x;
int r,i;
scanf("%d",&r);
a=r;
x=a/2.;
for(i=1;i<=10;i++){
x=(x+a/x)/2.;
}
printf("radice di %f=%f %f\n ",a,x,x*x);
}

// pi.c
// pi=4arctan(1)
// arctan(1)=1-1/3+1/5-1/7+1/9-1/11+...
// non bastano 10.000 iterazioni
```

```

#include<stdio.h>
main(){
double res,sign,i;
res=sign=1.;
for(i=3;i<=90000000;i+=2){
sign*=-1;
res+=sign/i;
}
printf("pi=%1.9e\n",4*res);
}

```

```

// 584 ~/EsC ---> cc pi.c
// 585 ~/EsC ---> a.out
// pi=3.141397

```

```

// simpson.c
// pi=4arctan(1)
// via integrale da 0 a 1 di 1/(1+x*x)
// Simpson Rule (vedi Google) con 10.000 intervalli
#include<stdio.h>
main(){
float res,i;
res=0;
for(i=.0001;i<1.;i+=.0002){
res+=4./(1.+i*i)+2./(1.+(i+.0001)*(i+.0001));
}
printf("pi=%f\n",4./30000.*(2.5+res));
}

```

```

// valore vero 3,14159

```

```

// 629 ~/EsC ---> cc simpson.c
// 630 ~/EsC ---> a.out
// pi=3.141885

```

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

```
// sinus.c
x=x/c;
  res=x;
sign=1;
for (i=3;i<=51;i+=2){
sign=sign*-1;
new=res+sign*power(x,i)/fact(i);
if(res==new)break;
res=new;
printf("res=%f new=%f\n",res,new);
}
}
float fact(int k){
int j;
float f=1.;
for (j=1;j<=k;j++)f=f*j;
return f;
}
float power(float a,int b){
float p=1.;
int l;
for(l=1;l<=b;l++)p=p*a;
return p;
}
```