Programmi con argomenti

Quando si compila un programma C viene generato un file eseguibile (.exe sotto dos). Questo programma può venire eseguito direttamente scrivendo il suo nome sulla linea di comando (finestra PromptMSDOS sotto Windows). Come è noto, ai programmi eseguibili è possibile passare degli argomenti. Per esempio, il programma copy.exe, che fa parte del sistema operativo, viene eseguito scrivendo due nomi di file dopo il nome del programma. Il programma è in grado di leggere questi due nomi, e comportarsi di conseguenza, copiando il primo file sul secondo. Il punto fondamentale è che un programma esguibile deve in generale essere in grado di poter usare i parametri con cui è stato lanciato, ossia le stringhe che seguono il nome del programma sulla linea di comando.

In C, è possibile accedere a queste informazioni modificando la intestazione della funzione `main`. I programmi visti fino ad ora usavano una intestazione di questo tipo:

```
int main() ...
```

Per poter usare gli argomenti con cui il programma è stato lanciato, questa linea deve diventare:

```
int main(int argn, char *argv[]) ...
```

L'effetto di questa modifica è che all'interno del programma risultano definite una variabile di tipo intero `argn`, e un array di stringhe `argv`. La variabile intera contiene il numero di argomenti con cui il programma è stato lanciato più uno, quindi è semplicemente il numero di stringhe che si trovano sulla linea di comando. Il vettore di stringhe contiene le varie stringhe che compongono la linea di comando: in particolare, `argv[0]` è il nome del programma stesso, mentre `argv[1]` è il suo primo argomento, `argv[2]` è il secondo, ecc.

Le pagine seguenti mostrano come si possono usare queste variabili.

Numero di argomenti ricevuti

La regola mnemonica per ricordare il significato delle due variabili `argn` e `argv` è che `argn` è il numero di stringhe che si trovano sulla linea di comando (incluso il nome del programma stesso) mentre `argv` è un vettore che contiene le varie stringhe, in ordine a partire dal nome del programma.

Visto che la prima stringa è il nome del comando, il numero di argomenti che il programma ha ricevuto è semplicemente il valore di `argn` meno uno. Il programma seguente quanti.c stampa questo numero.

```
/*
  Stampa il numero di argomenti ricevuti.
*/

#include<stdlib.h>

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

  printf("Ho ricevuto %d argomenti\n", argc-1);

  return 0;
}
```

Per testare il programma occorre generare il file eseguibile (`quanti.exe` sotto dos e `quanti` oppure `a.out` sotto unix), e poi lanciare il programma stesso, usando una linea di comando come quella qui sotto:

```
  quanti argomentouno argomentodue ...
```

Il programma dovrebbe stampare il numero di argomenti ricevuti. Si noti che all'inizio della esecuzione del programma il valore di `argc` è già quello giusto: a scrivere il numero di argomenti in questa variabile ha già provveduto il sistema operativo.

Usare gli argomenti ricevuti

Una volta determinato il numero di argomenti, il programma dovrebbe essere in grado di usarli. Come si è detto, gli argomenti del programma si trovano nel vettore di stringhe `argv`. Il primo argomento è la stringa `argv[1]`, mentre l'ultimo argomento è la stringa `argv[argc-1]`.

Il programma seguente quali.c stampa, oltre al numero di argomenti, il loro valore.

```
/*
  Stampa il numero di argomenti ricevuti,
e dice quali sono.
*/

#include<stdlib.h>

int main(int argc, char* argv[]) {
  int i;

  printf("Ho ricevuto %d argomenti\n", argc-1);

  printf("Questi argomenti sono:\n");

  for(i=1; i<=argc-1; i++)
    printf("%s\n", argv[i]);

  return 0;
}
```

Il ciclo viene ovviamente fatto partendo da `1`, dato che `argv[1]` contiene il primo argomento, fino ad arrivare a `argc-1`, dato che l'ultimo argomento si trova in `argv[argc-1]`.

Per eseguire il programma, occorre generare un file eseguibile, e poi eseguirlo da linea di comando passando degli argomenti.

Scrivi la metà di un numero su file

Questo esercizio consiste nella scrittura della metè di un numero su file. La cosa nuova rispetto ai programmi visti nella parte sui file è che qui il nome del file su cui scrivere e il numero da dimezzare vengono passati al programma da linea di comando (anzichè essere codificati nel programma, oppure letti da tastiera).

In particolare, vogliamo che usare come nome del file il primo argomento passato, e come numero il secondo. Per quello che riguarda l'uso del primo argomento non ci sono problemi: infatti, per specificare quale file si vuole aprire per scrivere, bisogna passarlo alla funzione `fopen` come primo

parametro. Dal momento che questa funzione prende una stringa come nome del file, e il primo argomento del programma è `argv[1]`, che è già una stringa, tutto quello che serve è aprire il file usando una istruzione del tipo:

```
  fd=fopen(argv[1], "w");
```

Questa istruzione apre in scrittura il file che ha il nome specificato come primo argomento del programma (`argv[1]`).

Per poter dimezzare e scrivere il numero abbiamo appunto bisogno di avere in memoria il numero stesso. Si noti che `argv[2]` è una *stringa che rappresenta il numero*, e non il numero stesso. Infatti, è di tipo `char *` e non di tipo `int` come serve a noi. Per essere più chiari su questo punto: `argv[2]` è un vettore di caratteri, in cui il primo elemento `argv[2][0]` è la prima cifra del numero, il secondo carattere `argv[2][1]` è la seconda cifra del numero, ecc.

Se per esempio passiamo il numero `29102` al programma, allora il vettore `argv[2]` contiene il carattere `2` in prima posizione, il carattere `9` in seconda posizione, `1` in terza, ecc. Si noti che gli interi *non* si rappresentano in memoria in questo modo: per esempio, il numero di byte usati per rappresentare un intero è costante, indipendente dal numero di cifre del numero.

Per poter convertire una stringa in numero, usiamo la funzione `atoi`, che converte la stringa in intero nel modo che ci si aspetta, per cui per esempio la stringa `"29102"` viene convertita nel numero `29102`. Siamo ora in grado di scrivere il programma completo meta.c il cui codice è riportato qui sotto.

```
/*
  Dimezza un intero e lo scrive su file: il nome
del file e il valore del numero sono passati al
programma come argomenti.
*/

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

int main(int argn, char *argv[]) {
  int x;
  FILE *fd;


                /* verifica se gli argomenti sono in numero giusto */
  if(argn-1!=2) {
    printf("Numero inesatto di argomenti\n");
    exit(1);
  }


                /* converte il secondo argomento in un intero */
  x=atoi(argv[2]);


                /* apre il file in scrittura */
  fd=fopen(argv[1], "w");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }
```

```
                /* scrive il numero */
  fprintf(fd, "%d\n", x/2);


                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Dal momento che il programma, per poter dimezzare il numero e scriverlo su file, deve ricevere sia il nome del file che il numero, è opportuno controllare se effettivamente sono stati passati due argomenti e non un numero inferiore.

Copia di un file

Questo esercizio consiste nella copiatura di un file di testo in un file di altro nome. Il nome del vecchio file, e del nuovo, devono essere passati come argomenti al programma.

Il programma finale copia.c è riportato qui sotto. Dopo aver controllato che al programma siano stati effettivamente passati due argomenti vengono aperti i cui nomi sono il primo e il secondo argomento che il programma ha ricevuto.

Un particolarità di questo programma è che il file da scrivere viene prima aperto in lettura. Se il file già esiste questa apertura non genera errori. In questo modo, è possibile verificare la esistenza del file prima di scrivere: se il risultato della `fopen` è diverso da `NULL` allora il file esiste già, per cui si può terminare il programma senza cancellare il contenuto del vecchio file.

```
/*
  Copia un file: sorgente e destinazione sono
passati come argomenti.
*/

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

int main(int argn, char *argv[]) {
  char buf[200];
  FILE *in, *out;
  int res;


                /* controllo numero di argomenti */
  if( argn-1<2 ) {
    printf("Mancano gli argomenti\n");
    exit(1);
  }


                /* apre il file da leggere */
  in=fopen(argv[1], "r");
  if( in==NULL ) {
    perror("Errore in apertura del file da leggere");
    exit(1);
  }


                /* controlla se il file da scrivere esiste */
```

```
  out=fopen(argv[2], "r");
  if( out!=NULL ) {
    printf("Il file da scrivere esiste gia'\n");
    exit(1);
  }
  /* nota: se out==NULL allora non si *deve* fare fclose */


                /* apre il file da scrivere */
  out=fopen(argv[2], "w");
  if( out==NULL ) {
    perror("Errore in apertura del file da scrivere");
    exit(1);
  }


                /* legge e scrive ogni riga */
  while(1) {
    fgets(buf, 200, in);
    if( feof(in) )
      break;
    fputs(buf, out);
  }


                /* chiude i file */
  fclose(in);
  fclose(out);

  return 0;
}
```

Eliminazione estensione

Questo esercizio consiste nella eliminazione della estensione dal nome di un file. Il nome di un file è una stringa, che può contenere il carattere punto `'.'` Lo scopo di questo esercizio è realizzare un programma che prende una stringa da linea di comando, e stampa la parte iniziale della stringa fino al carattere punto `'.'`; se questo carattere non c'è, si deve stampare tutta la stringa (il fatto che la stringa passata è il nome di un file è in effetti irrilevante).

Per prima cosa, il programma deve poter usare gli argomenti passati da linea di comando. Per questa ragione, la intestazione della funzione `main` deve contenere l'intero e il vettore di stringhe:

```
int main(int argn, char *argv[]) ...
```

Il programma deve venire chiamato con un singolo argomento, e questo si può controllare vedendo se il numero di argomenti passati, che è `argn-1`, coincide o meno con 1. In caso contrario, si stampa un messaggio di errore e si esce.

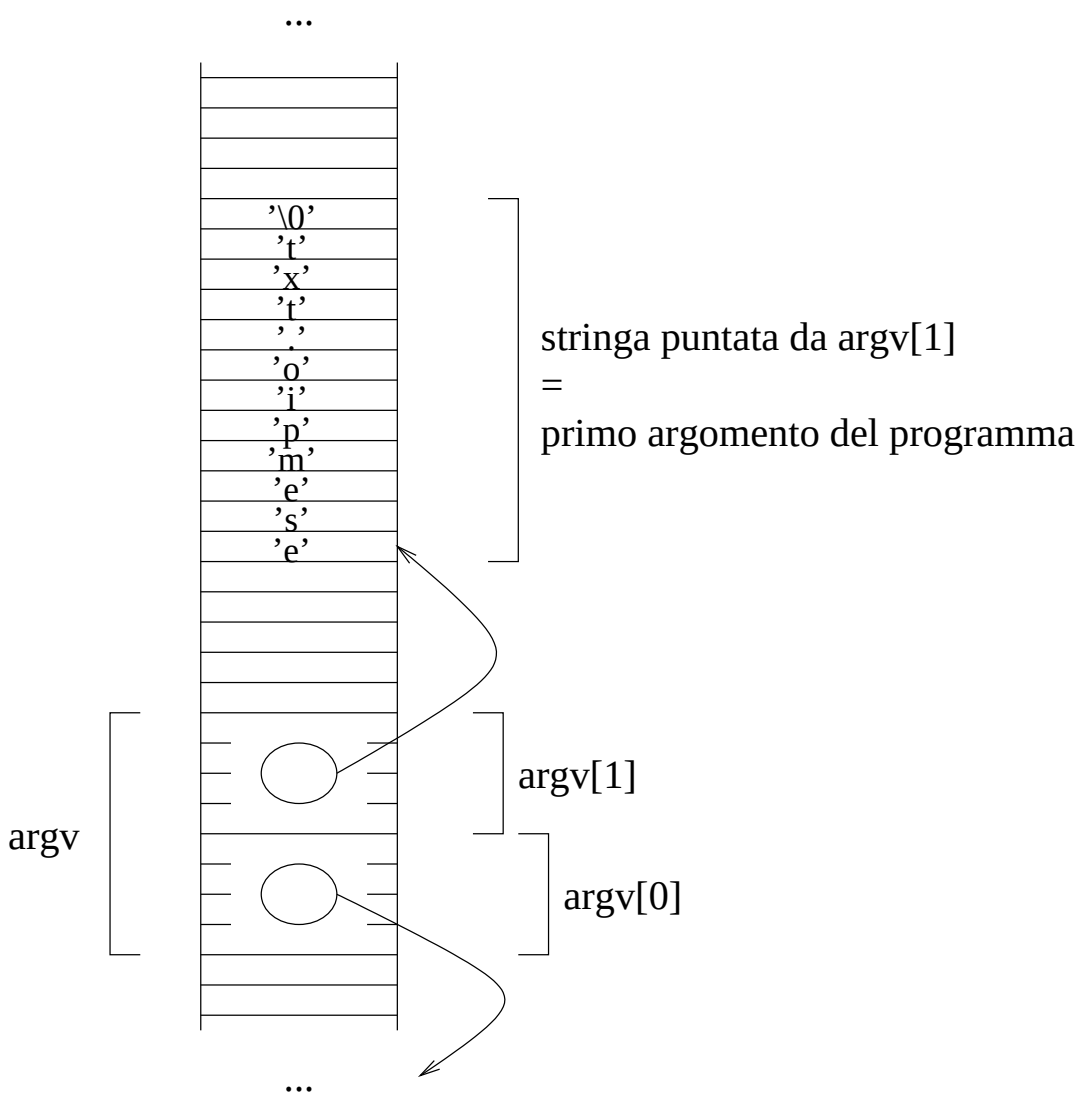
A questo punto, sappiamo che il programma è stato lanciato con un solo argomento. Il primo argomento si trova come sempre memorizzato nella stringa `argv[1]`. Si ricorda che:

> una stringa è un vettore di caratteri che contiene il carattere `'\0'` che indica che i successivi caratteri non sono significativi.

In altre parole, una stringa è un vettore di caratteri, che può essere sia statico che dinamico. I caratteri compresi fra il primo e il carattere `'\0'` sono quelli significativi della stringa. Si dice anche che il carattere `'\0'` è il *terminatore* di stringa. Dal momento che questo è il modo per indicare dove la stringa finisce, questo carattere deve apparire almeno una volta nel vettore.

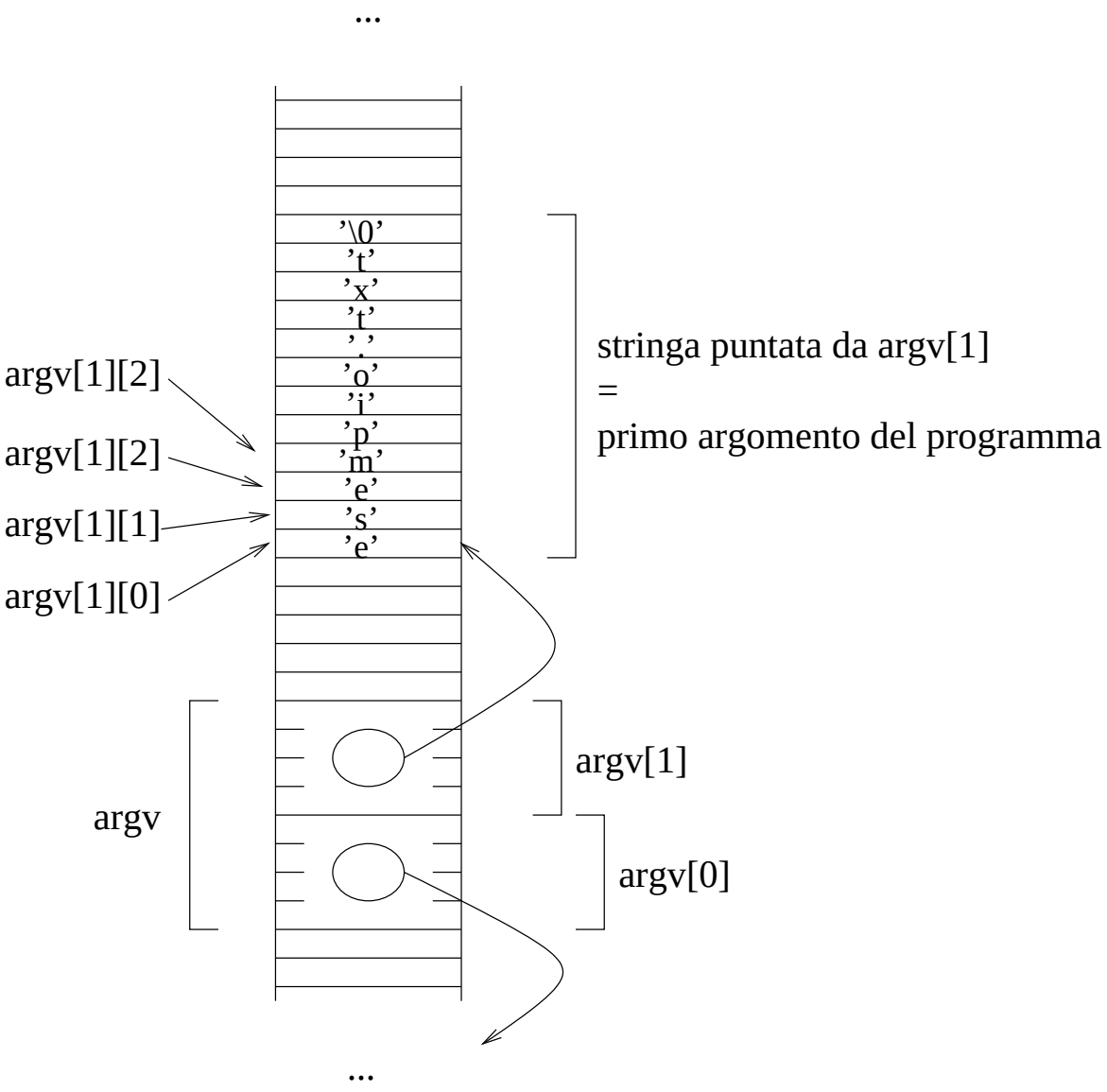
Nel nostro caso, sappiamo che `argv[1]` è la stringa che ci interessa. In particolare, questa variabile contiene un puntatore a carattere, che si può appunto considerare come un vettore allocato dinamicamente. Il vettore `argv` è un vettore di puntatori, ossia un vettore i cui elementi sono puntatori. Il punto fondamentale è che ogni elemento del vettore è un puntatore a caratteri. In particolare, `argv[1]` punta a una zona di memoria in cui si trovano, in ordine, i caratteri che compongono il primo argomento con cui il programma è stato chiamato. Inoltre, sappiamo che questa sequenza di caratteri è una stringa, ossia la sua parte significativa viene seguita immediatamente dal carattere di fine stringa `'\0'`.

Per capire meglio la situazione, vediamo come è strutturata le memoria quando il programma viene chiamato con un singolo argomento, e supponiamo che questo primo argomento sia `esempio.txt`. In altre parole, vediamo cosa succede all'avvio del programma quando viene lanciato per stampare il nome di file `esempio.txt` senza estensione.



Quello che succede è che la sequenza di caratteri `esempio.txt` viene memorizzata in una zona di memoria, seguita dal singolo carattere `'\0'` (questo lo si vede leggendo i caratteri nella figura dal basso verso l'alto). Questo è il modo standard di memorizzare una stringa. Il vettore `argv` contiene un insieme di puntatori, ognuno dei quali punta all'inizio della zona di memoria in cui è memorizzata una stringa. In particolare, `argv[1]` punta all'inizio della zona dove si trova la stringa `esempio.txt`.

Dal momento che `argv[1]` è un puntatore a carattere, lo si può interpretare come un vettore dinamico, per cui `argv[1][0]` è il carattere puntato da `argv[1]`, mentre `argv[1][1]` è il successivo, `argv[1][2]` quello ancora dopo, ecc.



In questo modo possiamo accedere agli elementi di tutta la stringa. Infatti, sappiamo ora che il primo elemento della stringa è `argv[1][0]`, sappiamo come accedere ai successivi, e sappiamo che la stringa termina dove si incontra il carattere `'\0'`.

A questo punto, abbiamo tutti i dati che ci servono per realizzare il facile programma di eliminazione della estensione. Tutto quello che dobbiamo fare è leggere in ordine i caratteri nel vettore dinamico `argv[1]` fino a che non si incontra il carattere punto `'.'`, oppure si arriva alla fine della stringa (ossia si arriva al carattere nullo `'\0'`.

Usiamo quindi una variabile `i` per fare la scansione della stringa. Dal momento che il primo carattere si trova nel primo elemento del vettore, che ha indice 0, questa variabile parte da zero. Dato che questa variabile deve ogni volta indicare un carattere successivo del vettore, la incrementiamo ad ogni passo. In particolare, ad ogni passo stampiamo il carattere indicato da `i` e incrementiamo la variabile `i` in modo tale che alla successiva iterazione del ciclo si passa al carattere successivo. L'uscita dal ciclo deve avvenire se si incontra il carattere punto `'.'` oppure la fine della stringa, cioè il carattere nullo `'\0'`.

```
                /* ciclo di stampa */
  i=0;
  while( argv[1][i]!=0 && argv[1][i]!='.' ) {
    printf("%c", argv[1][i]);
    i++;
  }
```

Dal momento che la condizione di uscita è l'opposto della condizione che ci fa stare nel ciclo, la condizione del while è che il carattere sia diverso dal punto e dal fine stringa. In altre parole, finchè il carattere corrente è diverso dal punto e dal carattere nullo, dobbiamo continuare a eseguire il ciclo.

Il programma completo noext.c è riportato qui sotto.

```
/*
  Elimina da un nome di file la sua estensione.
*/

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

int main(int argn, char *argv[]) {
  int i;


                /* controllo argomenti */
  if( argn-1!=1 ) {
    printf("Numero errato di argomenti\n");
    exit(1);
  }


                /* ciclo di stampa */
  i=0;
  while( argv[1][i]!=0 && argv[1][i]!='.' ) {
    printf("%c", argv[1][i]);
    i++;
  }

  printf("\n");


  return 0;
}
```

Una possibile variante del programma memorizza la parte iniziale della stringa (fino al primo punto), in una seconda stringa. Questo può essere utile se per esempio serve accedere a un file che ha la stessa parte iniziale del nome ma una estensione diversa.

La soluzione non differisce di molto dalla prima. Intanto, ci serve una stringa per memorizzare la parte iniziale della stringa. Usiamo per esempio un vettore statico, e assumiamo che bastino cento caratteri. Il programma deve copiare i caratteri, in ordine, dalla stringa `argv[1]` al vettore di caratteri da noi dichiarato. Questa operazione di copiatura termina quando si verifica una di queste tre cose:

1. la stringa `argv[1]` è finita;
2. abbiamo trovato il carattere punto nella stringa `argv[1]`;
3. non c'è più spazio nel vettore

Bisogna ora fare attenzione al fatto che, in tutti e tre i casi, il vettore su cui copiamo la stringa deve contenere, alla fine della parte significativa, il carattere di terminazione. Questo implica che la parte di vettore su cui si possono scrivere i caratteri non sono 100, ma solo 99, visto che l'ultimo serve per il terminatore. Questo significa che, nel terzo caso, non dobbiamo fermarci al carattere di indice 100-1, ma al carattere di indice 99-1 (l'ultimo carattere del vettore è quello di indice 100-1, ma questo carattere va lasciato libero per metterci il terminatore).

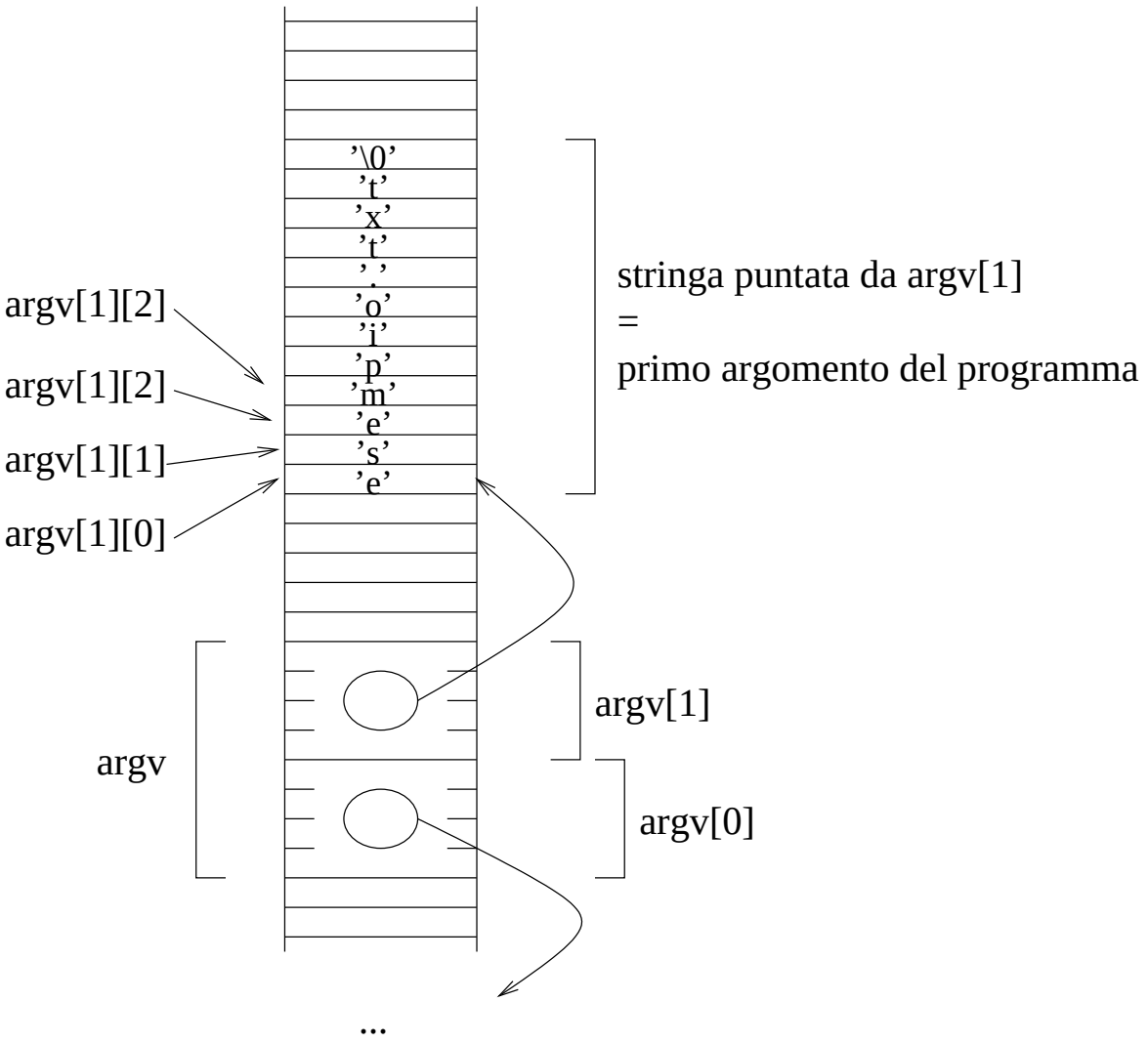
Dopo aver copiato la stringa, occorre scrivere il terminatore di stringa. Dal momento che quando si esce il valore di `i` è l'indice del primo elemento libero dell'array, questa è la posizione in cui va scritto il terminatore.

Il programma completo noextcopy.c è qui sotto.

```
/*
  Elimina da un nome di file la sua estensione.
*/

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

int main(int argn, char *argv[]) {
  char senzaext[100];
  int i;


                /* controllo argomenti */
  if( argn-1!=1 ) {
    printf("Numero errato di argomenti\n");
    exit(1);
  }


                /* copia l'argomento in senzaext fino a . oppure alla fine */
  i=0;
  while( argv[1][i]!=0 && argv[1][i]!='.' && i<=100-1-1 ) {
    senzaext[i]=argv[1][i];
    i++;
  }
  senzaext[i]=0;


                /* stampa il nome senza estensione */
  printf("%s\n",senzaext);


  return 0;
}
```

Un’altra facile variante di questo esercizio consiste nel copiare su stringa non la parte iniziale della stringa, ma solo la estensione, ossia la parte che segue il primo carattere punto `'.'` che si incontra.

È possibile modificare tutti i programmi sui file in modo tale che il nome del file (o dei file, se sono più di uno) su cui operano venga passato come argomento al programma. Ad esempio, il programma



commenti.c legge da un file il cui nome è dato come argomento, e ne stampa le righe che non cominciano con il carattere `'#'`