

Programmazione I e laboratorio: prova pratica

Tempo a disposizione: 2 ore

Corso B

17/12/2019

Implementare il programma richiesto usando il linguaggio standard ANSI C e sottomettere la soluzione sulla piattaforma di esame.

Il codice consegnato sarà valutato solo se supera almeno il 51% dei test case. La valutazione terrà conto della correttezza del programma, strutture dati usate, uso efficiente di memoria, efficienza dell'implementazione algoritmica, stile di programmazione, controlli dell'input.

Non è consentito l'uso di nessun materiale o strumento tecnologico oltre il computer del laboratorio. Si può usare qualsiasi IDE installato sul computer.

Non è consentita la collaborazione tra studenti.

Se si desidera abbandonare l'esame occorre dirlo esplicitamente ai docenti per evitare la consegna del codice.

Compilazione Si ricorda di compilare con l'opzione ANSI C e che per abilitare i messaggi di diagnostica del compilatore, bisogna compilare il codice usando le opzioni `-g -Wall` di gcc:

```
gcc -std=c89 -Wall -g sorgente.c -o eseguibile
```

Provare la propria soluzione in locale. Valutare la correttezza della soluzione sulla propria macchina accertandosi che rispetti gli input/output contenuti nel TestSet. I file di input e output per i test sono nominati secondo lo schema:

```
input0.txt output0.txt
```

```
input1.txt output1.txt
```

...

Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirectione dell'input. Ad esempio

```
./eseguibile < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che `eseguibile` sia il file ottenuto dalla compilazione del vostro codice. Per effettuare un controllo automatico sul primo file input `input0.txt` e trovare le differenze fra l'output prodotto dal vostro programma e quello corretto potete eseguire la seguente sequenza di comandi

```
./eseguibile < input0.txt | diff - output0.txt
```

Esercizio

Scrivere un programma che gestisca i risultati di un concorso di programmazione. Ci sono P prove distinte che possono essere raggruppate in categorie diverse, e N studenti partecipanti. I partecipanti ricevono un voto in ogni prova. Alla fine del concorso gli organizzatori devono scegliere un vincitore per ogni categoria e un vincitore globale.

L'input è così composto:

- Il primo valore immesso è un numero intero P che corrisponde al numero di prove seguito da un'interlinea. Il programma deve controllare che questo numero sia un intero maggiore di 0. In caso contrario, il programma esce col messaggio **input non corretto**.
- Segue una riga con P interi separati da spazio, che rappresentano le categorie corrispondenti alle varie prove. Non si sanno in anticipo quante categorie ci sono, e i valori interi sono codici che possono prendere qualsiasi valore positivo. Se si riscontrano valori negativi o invalidi il programma esce col messaggio **input non corretto**.
- Una riga contenente il numero N di studenti partecipanti. Il programma deve controllare che questo numero sia un intero maggiore di 0. In caso contrario, il programma esce col messaggio **input non corretto**.
- N righe che contengono i voti degli studenti per ogni prova, separati da spazio. La i -esima prova corrisponde all' i -esima categoria precedentemente introdotta. Anche questi valori sono interi, nell'intervallo $[0, 100]$. Se si riscontrano valori invalidi il programma esce col messaggio **input non corretto**.

Il programma deve leggere l'input e poi stampare, per ogni categoria, l'indice dello studente che ha vinto la categoria (gli indici partono da 1, nell'ordine di inserimento dei voti). Il voto per categoria viene calcolato facendo la media tra tutti i voti presi nelle prove di quella categoria. Vince il voto più alto. Se ci sono più di uno studente con voto massimo vince l'indice minore. Le categorie vengono stampate una per riga, nell'ordine in cui compaiono la prima volta nell'input.

Alla fine viene indicato anche l'indice dello studente che a vinto globalmente (che ha la media più alta su tutte le prove).

Si ricorda che non vi deve essere spreco nell'uso della memoria. Sarà valutata anche l'organizzazione del codice: uso delle funzioni, commenti, indentazione, semplicità.

Esempio

Input

```
5
2 5 2 1 5
4
80 70 100 10 58
77 65 86 98 74
84 100 76 100 97
96 98 50 75 89
```

Output

```
Categoria 2:  vince candidato 1.
Categoria 5:  vince candidato 3.
Categoria 1:  vince candidato 3.
Vincitore globale:  candidato 3.
```



UNIVERSITÀ
DI PISA

PROGRAMMAZIONE 1
Corso B

documentazione per l'esame

printf

convertitore	descrizione
d	stampa intero decimale con segno
i	stessa cosa di d — sarà diverso per scanf
o	stampa intero senza segno in ottale
u	stampa intero decimale senza segno
x oppure X	stampa intero decimale senza segno in esadecimale (x usa a-f, X usa A-F come cifre)
h, l oppure ll	prima di qualsiasi convertitore intero per short, long e long long (length modifiers)
e oppure E	stampa floating point in notazione esponenziale
f	stampa floating point in notazione fixed point
g oppure G	come f o e (E) a seconda della grandezza del numero
L	prima di qualsiasi convertitore floating-point per indicare long double
c	stampa caratteri
s	stampa stringhe
p	stampa l'indirizzo delle variabili
%	stampa %

	precisione
interi	numero minimo di cifre da stampare (default 1). Se maggiore delle cifre dell'intero si estendono gli eventuali 0 iniziali, 0 dopo la virgola oppure si premettono spazi per giustificare a destra
reali	con e, E, f è il numero di cifre dopo la virgola; con g e G é il numero di cifre significative da stampare
stringhe	il numero massimo di caratteri da stampare dall'inizio della stringa

printf

flag	descrizione
-	giustifica a sinistra l'output nel campo corrispondente
+	stampa il segno + (-) davanti ai numeri positivi (negativi)
spazio	stampa uno spazio davanti ai valori positivi al posto del + se non stampati con flag +
#	prefigge 0 ad output con %o (ottale), prefigge 0x (0X) ad output con %x (%X), forza la stampa del punto decimale negli output con %e, %E, %g, %G, %f anche in assenza di parte decimale
0	tiempo un campo con 0 iniziali davanti al dato stampato

escape set	descrizione
\'	stampa il singolo apice
\"	stampa apice doppio
\?	stampa punto interrogativo
\\	stampa il backslash
\a	genera un alert sonoro o visuale
\b	muove il cursore indietro di una posizione sulla linea corrente
\f	muove il cursore all'inizio della successiva pagina logica
\n	muove il cursore all'inizio della linea successiva
\r	muove il cursore all'inizio della linea corrente
\t	muove il cursore alla posizione del prossimo tab orizzontale
\v	muove il cursore alla posizione del prossimo tab verticale

scanf

convertitore	descrizione
d	legge intero decimale opzionalmente con segno
i	legge intero decimale opzionalmente con segno, ottale o esadecimale
o	legge intero ottale
u	legge intero decimale senza segno
x oppure X	legge intero esadecimale (x usa a-f, X usa A-F come cifre)
h, l oppure ll	prima di qualsiasi convertitore intero per short, long e long long (length modifiers)
e, E, f, g o G	legge valori floating point
l oppure L	prima di qualsiasi convertitore floating-point per indicare double o long double
c	legge caratteri
s	legge stringhe
[char set]	legge input finché trova caratteri contenuti nel set
n	memorizza il numero di caratteri letti fino a quel momento
p	legge un indirizzo
%	salta i caratteri % nell'input

<math.h>

funzione	descrizione
sqrt(x)	radice quadrata
cbrt(x)	radice cubica
exp(x)	funzione esponenziale
log(x)	logaritmo naturale di x
log10(x)	logaritmo in base 10 di x
fabs(x)	valore assoluto di x come floating number
ceil(x)	arrotonda al più piccolo intero non minore di x
floor(x)	arrotonda al più grande intero non maggiore di x
pow(x, y)	x elevato alla y
fmod(x, y)	resto di x/y come floating number
sin(x)	seno di x in radianti
cos(x)	coseno di x in radianti
tan(x)	tangente di x radianti

file accesso casuale

i file ad accesso casuale sono organizzati in slot di dimensione fissata e consentono di accedervi senza dover scorrere tutto il file — si può calcolare la posizione di un record come offset dall'inizio del file e di una chiave di accesso al record



è possibile inserire nuovi dati senza distruggere quelli già presenti

è possibile leggere i dati sia sequenzialmente che casualmente

è possibile aggiornare i dati

è possibile cancellare record

è possibile posizionare il puntatore del file ad una posizione specifica con

int fseek(FILE *, long, int)

dove il secondo parametro è l'offset (distanza) in byte da un punto del file dove cominciare a scrivere individuato dal terzo parametro che è una costante simbolica tra SEEK_SET (offset dall'inizio del file), SEEK_CUR (offset dalla posizione corrente) e SEEK_END (offset dalla fine del file) — ritorna un valore non nullo

int fprintf(FILE *, const char *, ...)

int fscanf(FILE *, const char *, ...)

scrivono e leggono il numero di caratteri necessario a rappresentare testualmente il dato (es. **1** si scrive un carattere)

int fwrite(const void *, size_t, size_t, FILE *)

int fread(const void *, size_t, size_t, FILE *)

scrivono e leggono array su e da file per un numero di byte passato come secondo argomento a partire da una posizione iniziale indicata dal puntatore al file. Il terzo argomento rappresenta il numero di elementi nell'array — variabili singole sono trattate come array di un solo elemento

ritornano tutte il numero di scritture o letture effettuate con successo

<stdio.h>

Prototipo	Descrizione
<code>int getchar(void);</code>	legge il carattere successivo dallo standard input e lo restituisce come intero
<code>int putchar(int c);</code>	stampa il carattere memorizzato in c e lo restituisce come intero
<code>int puts(const char *s);</code>	stampa una stringa costante passata come argomento seguita da newline; ritorna un intero > 0 se eseguita con successo ed EOF altrimenti
<code>int sprintf(char *s, ..);</code>	il primo parametro formale è un array di caratteri, i rimanenti parametri sono gli stessi di printf ; anziché stampare a schermo, memorizza l'output nell'array s — ritorna il numero di caratteri scritti in s , oppure EOF se si verifica un errore
<code>int sscanf(char *s, ..);</code>	il primo parametro formale è un array di caratteri, i rimanenti parametri sono gli stessi di scanf ; anziché leggere da tastiera, legge dall'array s — ritorna il numero di caratteri letti con successo, oppure EOF se si verifica un errore
<code>char *fgets(char *s, int n, FILE *stream);</code>	legge caratteri dallo stream specificato nel terzo parametro e li memorizza nell'array di caratteri s fino a che non incontra un newline oppure ha letto n-1 byte (cioè n-1 caratteri); il carattere NULL è appeso al termine dell'array — lo stream stdin è lo standard input, tipicamente la tastiera; ritorna la stringa letta e memorizzata in s — se incontra un newline, viene incluso nella stringa letta

<ctype.h>

Prototipo	Descrizione
<code>int isblank(int c);</code>	se <code>c</code> è uno spazio che separa parole in una linea di testo ritorna vero, altrimenti falso (<code>0</code>)
<code>int isdigit(int c);</code>	se <code>c</code> è una cifra ritorna vero, altrimenti falso (<code>0</code>)
<code>int isalpha(int c);</code>	se <code>c</code> è una lettera ritorna vero, altrimenti falso (<code>0</code>)
<code>int isalnum(int c);</code>	se <code>c</code> è una cifra o una lettera ritorna vero, altrimenti falso (<code>0</code>)
<code>int isxdigit(int c);</code>	se <code>c</code> è una cifra esadecimale ritorna vero, altrimenti falso (<code>0</code>)
<code>int islower(int c);</code>	se <code>c</code> è una lettera minuscola ritorna vero, altrimenti falso (<code>0</code>)
<code>int isupper(int c);</code>	se <code>c</code> è una lettera maiuscola ritorna vero, altrimenti falso (<code>0</code>)
<code>int tolower(int c);</code>	se <code>c</code> è una lettera maiuscola ritorna <code>c</code> minuscola, altrimenti <code>c</code> senza cambiamenti
<code>int toupper(int c);</code>	se <code>c</code> è una lettera minuscola ritorna <code>c</code> maiuscola, altrimenti <code>c</code> senza cambiamenti
<code>int isspace(int c);</code>	se <code>c</code> è un carattere separatore spaziale (newline <code>'\n'</code> , spazio <code>' '</code> , form feed <code>'\f'</code> , tab orizzontale <code>'\t'</code> , tab verticale <code>'\v'</code> , return <code>'\r'</code>) ritorna vero, altrimenti falso (<code>0</code>)
<code>int iscntrl(int c);</code>	se <code>c</code> è un carattere di controllo (newline <code>'\n'</code> , form feed <code>'\f'</code> , alert <code>'\a'</code> , backspace <code>'\b'</code> , tab orizzontale <code>'\t'</code> , tab verticale <code>'\v'</code> , return <code>'\r'</code>) ritorna vero, altrimenti falso (<code>0</code>)
<code>int ispunct(int c);</code>	se <code>c</code> è un carattere di stampa (cioè visibile a video) diverso da cifre, lettere e spazio (es. <code>\$</code> , <code>#</code> , <code>(</code> , <code>)</code> , <code>[</code> , <code>]</code> , <code>{</code> , <code>}</code> , <code>;</code> , <code>:</code> , <code>%</code> , ecc.) ritorna vero, altrimenti falso (<code>0</code>)
<code>int isprint(int c);</code>	se <code>c</code> è un carattere di stampa incluso spazio ritorna vero, altrimenti falso (<code>0</code>)
<code>int isgraph(int c);</code>	se <code>c</code> è un carattere di stampa escluso spazio ritorna vero, altrimenti falso (<code>0</code>)

<string.h>

Prototipo	Descrizione
<code>char *strcpy(char *s1, const char *s2);</code>	copia la stringa s2 nell'array s1 e ritorna il valore di s1
<code>char *strncpy(char *s1, const char *s2, size_t n);</code>	copia al più n caratteri della stringa s2 nell'array s1 e ritorna il valore di s1
<code>char *strcat(char *s1, const char *s2);</code>	appende la stringa s2 all'array s1 ; il primo carattere di s2 sovrascrive il carattere terminale NULL di s1 e ritorna il valore di s1
<code>char *strncat(char *s1, const char *s2, size_t n);</code>	appende al più n caratteri della stringa s2 all'array s1 ; il primo carattere di s2 sovrascrive il carattere terminale NULL di s1 e ritorna il valore di s1
<code>int strcmp(const char *s1, const char *s2);</code>	confronta s1 e s2 e ritorna 0 , >0 o <0 se s1 è uguale, maggiore o minore di s2 secondo l'ordinamento lessicografico
<code>int strncmp(const char *s1, const char *s2, size_t n);</code>	confronta fino a n caratteri della stringa s1 con la stringa s2 e ritorna 0 , >0 o <0 se s1 è uguale, maggiore o minore di s2 secondo l'ordinamento lessicografico
<code>char *strchr(const char s, int c);</code>	identifica la posizione del carattere c nella stringa s e restituisce un puntatore a c , altrimenti il puntatore NULL
<code>size_t strcspn(const char *s1, const char *s2);</code>	ritorna la lunghezza del primo segmento di s1 formato solo da caratteri non contenuti in s2
<code>size_t strspn(const char *s1, const char *s2);</code>	ritorna la lunghezza del primo segmento di s1 formato solo da caratteri contenuti in s2
<code>char *strpbrk(const char *s1, const char *s2);</code>	identifica la prima posizione in s1 che contiene un carattere di s2 e ritorna un puntatore a tale carattere, altrimenti il puntatore NULL
<code>char *strrchr(const char *s1, int c);</code>	identifica l'ultima occorrenza di c nella stringa s e ritorna un puntatore a c , altrimenti ritorna il puntatore NULL
<code>char *strstr(const char *s1, const char *s2);</code>	localizza la prima occorrenza di s2 in s1 e ritorna un puntatore a dove s2 inizia in s1 , altrimenti il puntatore NULL
<code>char *strtok(char *s1, const char *s2);</code>	una serie di chiamate divide s1 in token separati da caratteri contenuti in s2 — le chiamate successive alla prima hanno NULL come primo argomento; ciascuna chiamata ritorna un puntatore al token identificato e quando non ve ne sono più ritorna il puntatore NULL
<code>size_t strlen(const char *s);</code>	ritorna il numero di caratteri in s escluso il terminatore NULL

<string.h>

Prototipo	Descrizione
<code>void *memcpy(void *s1, const void *s2, size_t n);</code>	copia n byte dell'oggetto puntato da s2 nell'oggetto puntato da s1 e restituisce un puntatore all'oggetto risultante — indefinita se gli oggetti puntati da s1 e s2 sono anche parzialmente sovrapposti in memoria
<code>void *memmove(void *s1, const void *s2, size_t n);</code>	copia n byte dell'oggetto puntato da s2 nell'oggetto puntato da s1 e restituisce un puntatore all'oggetto risultante — la copia è usata facendo uso di un temporaneo in modo da poter copiare una parte di una stringa in un'altra parte della stessa stringa
<code>int memcmp(const void *s1, const void *s2, size_t n);</code>	confronta i primi n byte degli oggetti puntati da s1 e s2 — ritorna 0 se s1 == s2 , <0 se s1 < s2 e >0 se s1 > s2
<code>void *memchr(const void *s, int c, size_t n);</code>	ritorna un puntatore alla prima occorrenza di c (interpretato come unsigned char) in s oppure NULL se non trovata
<code>void *memset(void *s, int c, size_t n);</code>	copia c (interpretato come unsigned char) nei primi n byte dell'oggetto puntato da s e ritorna un puntatore al risultato
<code>char *strerror(int errornum)</code>	traduce i codici errore in frasi partendo dall'header <errno.h> e restituisce un puntatore a una stringa

operatore	Descrizione
&	AND — confronta due operandi bit a bit e restituisce AND dei singoli bit
	OR — confronta due operandi bit a bit e restituisce OR dei singoli bit
^	XOR — confronta due operandi bit a bit e restituisce XOR dei singoli bit (vero solo se i bit sono diversi)
<<	sposta i bit del primo operando a sinistra del numero di posizioni indicate dal secondo operando e riempie le posizioni lasciate vuote con 0
>>	sposta i bit del primo operando a destra del numero di posizioni indicate dal secondo operando e riempie le posizioni lasciate vuote con 0
~	complementa (1 in 0 e 0 in 1) tutti i bit del singolo operando
&=	$x \&= y$ è equivalente a $x = x \& y$
=	$x = y$ è equivalente a $x = x y$
^=	$x ^= y$ è equivalente a $x = x ^ y$
<<=	$x <<= y$ è equivalente a $x = x << y$
>>=	$x >>= y$ è equivalente a $x = x >> y$

modo	descrizione	modo	descrizione
r	apre un file esistente in lettura	rb	come r , ma file binario
w	crea un file in scrittura — se già esiste ne cancella il contenuto e lo apre in scrittura	wb	come w , ma file binario
a	apre un file per appenderne (scrittura) contenuto alla fine	ab	come a , ma file binario
r+	apre un file esistente per modifica (lettura e scrittura)	rb+	come r+ , ma file binario
w+	crea un file in lettura e scrittura — se già esiste ne cancella il contenuto e lo apre in scrittura	wb+	come w+ , ma file binario
a+	apre un file per modifica (lettura e scrittura), ma scritture sono fatte solo alla fine del file	ab+	come a+ , ma file binario

<stdarg.h>

Identificatore	Descrizione
va_list	è un tipo che rappresenta la lista di argomenti in modo adeguato per l'uso delle macro va_start , va_arg e va_end — un oggetto di tipo va_list deve essere definito nella funzione e usato per accedere correttamente agli argomenti
va_start(va_list, int)	è una macro da invocare prima di poter accedere agli argomenti di una lista di parametri a lunghezza variabile — inizializza gli elementi dichiarati con va_list per renderli usabili da va_start e va_end — l'intero rappresenta il numero di parametri in va_list
va_arg(va_list, type)	è una macro che accede all'elemento corrente della lista di argomenti del tipo specificato nel suo secondo parametro e sposta il puntatore in va_list al prossimo argomento
va_end(va_list)	è una macro che ripulisce la lista di argomenti a lunghezza variabile per consentire alla funzione di ritornare correttamente (gestisce opportunamente il record di attivazione)



UNIVERSITÀ
DI PISA

PROGRAMMAZIONE 1
Corso B

in bocca al lupo!!!!