

Fondamenti di Informatica ◇ 2019-20

Lezioni, Esercitazioni e Laboratorio

Cristiana Bolchini

Indice

16-09-2019	Lezione: introduzione al corso	1
17-09-2019	Lezione: la rappresentazione dell'informazione – parte 1	2
19-09-2019	Lezione: la rappresentazione dell'informazione – parte 2	3
23-09-2019	Lezione: algoritmi	4
4.1	Tempo espresso in ore, minuti e secondi	4
4.2	Operazioni di somma e sottrazione in modulo e segno	5
4.3	Valore assoluto	5
24-09-2019	Lezione: introduzione al C: struttura, tipi, operatori	8
5.1	Tempo in ore, minuti e secondi	8
26-09-2019	Lezione: costrutto di selezione if, if-else, if-else-if	9
6.1	Positivo, negativo o nullo	9
6.2	Anno bisestile	9
6.3	Terna pitagorica – Proposto	9
6.4	Ordinamento crescente/decrescente – Proposto	9
26-09-2019	Laboratorio: introduzione all'ambiente di lavoro	10
30-09-2019	Lezione: costrutto ciclo while e do-while	11
8.1	Minimo, massimo e valor medio	11
8.2	Fattoriale	11
01-10-2019	Esercitazione: costrutti elementari di controllo	12
9.1	Conversione in binario, allo specchio	12
9.2	Potenza del 2	12
9.3	Numero di cifre di un valore	12
9.4	Einstein	12
9.5	Conversione in binario, senza array – Proposto e risolto	12
9.6	Conversione in binario, allo specchio, con dimensione	13
07-10-2019	Lezione: array monodimensionali e ciclo for	14
10.1	5 dati in ingresso in ordine inverso	14
10.2	Sopra soglia	14
10.3	Conversione in binario – Proposto	14
08-10-2019	Esercitazione: array monodimensionali	15
11.1	Numero allo specchio	15
11.2	Conta lettere	15
11.3	Niente primi	15
10-10-2019	Lezione: array bidimensionali, struct e typedef	16
12.1	Matrice identità	16
12.2	Date: definizione di tipo	16
12.3	Studente: definizione di tipo	16
14-10-2019	Lezione: stringhe	17
13.1	Stringa allo specchio	17

13.2	Stringa allo specchio senza vocali	17
13.3	Anagrammi	17
15-10-2019	Analisi esercizi: array	18
17-10-2019	Esercitazione: array e stringhe	21
15.1	Determinante di una matrice dimensione 3	21
15.2	Nome di file da percorso, nome ed estensione	21
15.3	Da intero a stringa	21
15.4	Area di un poligono	21
15.5	Quadrato magico	21
17-10-2019	Laboratorio: algoritmi e array	23
21-10-2019	Lezione: sottoprogrammi	26
17.1	Combinazioni	26
17.2	Massimo di un array	26
17.3	Triangolo	26
22-10-2019	Lezione: sottoprogrammi	27
18.1	Massimo, minimo e media dei valori di un array	27
24-10-2017	Esercitazione: sottoprogrammi	28
19.1	Lunghezza di una stringa	28
19.2	Matrice identità	28
19.3	Vertici di un poligono	28
19.4	Vertice?	28
19.5	Combinazione stringhe – Proposto	28
19.6	Cifra del numero – Proposto	29
24-10-2019	Laboratorio: sottoprogrammi	30
28-10-2019	Esercitazione: sottoprogrammi ☒	33
21.1	Confronta lunghezza delle stringhe	33
21.2	Selection sort	33
21.3	Matrice trasposta	33
29-10-2019	Esercitazione: sottoprogrammi ☒	34
22.1	Prepara un cocktail	34
22.2	Controlla Sudoku	34
22.3	Zoo	34
31-10-2019	Esercitazione: array, stringhe	36
23.1	Lunghezza di una stringa	36
23.2	Stringa nella stringa	36
23.3	Cifre divisori	36
23.4	Insieme unione ordinato	36
07-11-2019	Lezione: file di testo e binari	37
24.1	Numeri pari e dispari	37
24.2	Numeri pari e dispari binari	37
24.3	Numeri primi .. chissà quanti	37
11-11-2019	Esercitazione: file ☒	38
25.1	Conta vocaboli	38
25.2	Copia file lowercase	38
25.3	Record di un Videogame	38
12-11-2019	Lezione: allocazione dinamica	39
26.1	Visualizza sequenza di numeri interi in ordine inverso	39

26.2	Stringa di vocali	39
26.3	Primi nel file	39
26.4	Anagramma – Proposto	39
14-11-2019	Lezione: liste concatenate semplici	40
27.1	Duplica stringa	40
27.2	itos	40
27.3	Tipo da lista	40
14-11-2019	Laboratorio: sottoprogrammi	41
18-11-2019	Lezione: liste concatenate semplici	46
29.1	Visualizza in ordine inverso	46
29.2	push	46
29.3	append	46
29.4	emptylist	46
29.5	itos	46
19-11-2019	Esercitazione: liste concatenate semplici	47
30.1	Quanta memoria!	47
30.2	Insieme unione	47
30.3	Elimina doppiati adiacenti	47
30.4	Da lista a insieme	48
30.5	Riempi lista	48
21-11-2019	Lezione: ricorsione	49
31.1	Fattoriale – versione ricorsiva	49
31.2	Lunghezza di una lista – versione ricorsiva	49
31.3	Lunghezza di una stringa – versione ricorsiva	49
31.4	Carattere nella stringa – versione ricorsiva	49
31.5	Variabili static	49
31.6	Scompatta lista	50
31.7	Paint ... semplificato – Proposto e poi risolto	50
21-11-2019	Laboratorio: lavoro di gruppo	51
26-11-2019	Esercitazione: riepilogo ∞	53
33.1	Fibonacci	53
33.2	Conta iniziali vocaboli di un file	53
33.3	Mergesort	53
33.4	Date di nascita	54
33.5	Paint	54
28-11-2019	Esercitazione: riepilogo ∞	55
34.1	Array to list	55
34.2	Convoluzione 1D	55
34.3	Accesso utente banca	55
34.4	Vicinanza media di una matrice	56
34.5	Separa stringa (proposto)	56
02-12-2019	Analisi esercizi: riepilogo ∞	57
03-12-2019	Lezione: parametri da riga di comando argv, argc e variabili globali	60
36.1	genera	60
36.2	conta primi	60
05-12-2019	Esercitazione: parametri da riga di comando	61
37.1	Crop	61

37.2	Unisci liste ordinate	61
37.3	Genera numeri binari	61
37.4	Trova somma	62
05-12-2019	Laboratorio: lavoro di gruppo	63
09-12-2019	Lezione: architettura del calcolatore (hw & so) – parte 1	66
10-12-2019	Lezione: architettura del calcolatore (hw & so) – parte 2	67
12-12-2019	Esercitazione: temi d'esame ∞	68
41.1	ISBN-10	68
41.2	Elimina le sottosequenze	68
41.3	Conta le vette	69
41.4	Da intero a lista	69
41.5	Cifra più frequente	69
16-12-2019	Sfida di programmazione a squadre	71
20-01-2020	Appello del 20-01-2020	73
43.1	Dominanti in matrice	74
43.2	Valori completi ed unici	74
43.3	Minimo e massimo di una stringa	74
43.4	Lista ruotata	74
17-02-2020	Appello del 17-02-2020	75
xx-07-2020	Appello del 03-07-2020	75
xx-07-2020	Appello del 17-02-2020	75
xx-09-2020	Appello del xx-09-2020	75

dal problema all'algoritmo

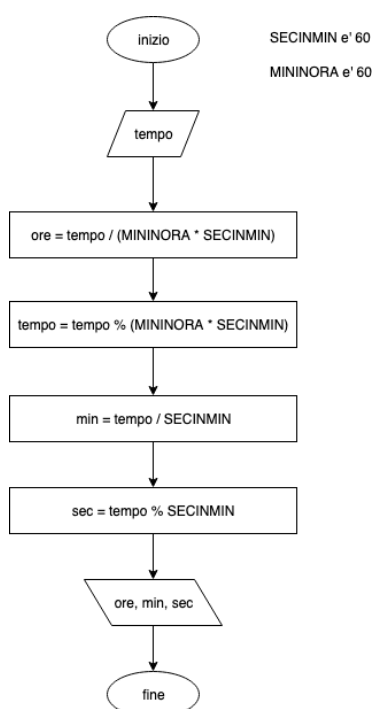
- ◇ proprietà dell'algoritmo
 - non ambiguo
 - deterministico
 - termina in un numero finito di passi
- ◇ tipologia di istruzioni
 - istruzioni effettive
 - istruzioni di controllo

diagrammi di flusso

- ◇ blocchi elementari
 - inizio
 - fine
 - istruzioni semplici
 - visualizzazione risultati
 - blocco di selezione

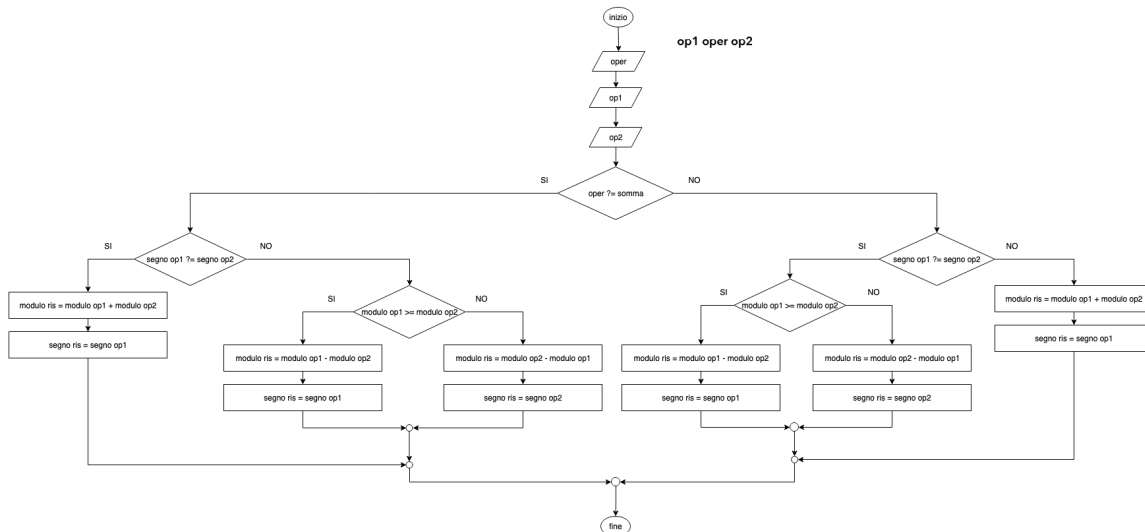
Esercizio 4.1: Tempo espresso in ore, minuti e secondi

Si realizzi l'algoritmo che acquisito un valore intero (senz'altro positivo) che rappresenta una quantità di tempo espressa in secondi, di calcola e visualizza la stessa quantità di tempo espressa in ore, minuti e secondi. Per esempio, se si inserisce 3812, l'algoritmo calcola e visualizza 1 (ora) 3 (minuti) 32 (sec).



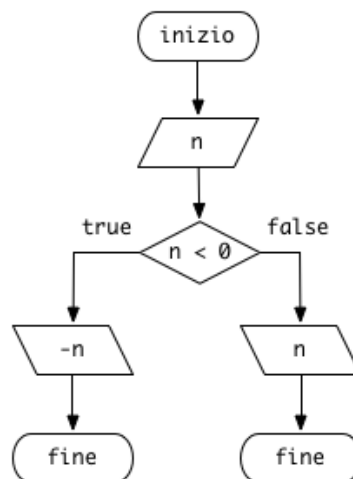
Esercizio 4.2: Operazioni di somma e sottrazione in modulo e segno

Si realizzi l'algoritmo che acquisito un operatore ('+' o '-') e due operandi effettua l'operazione e visualizza il risultato.

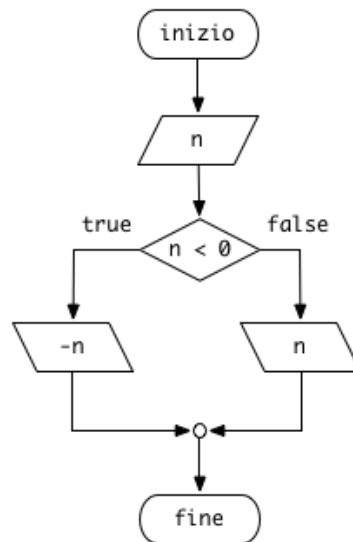


Esercizio 4.3: Valore assoluto

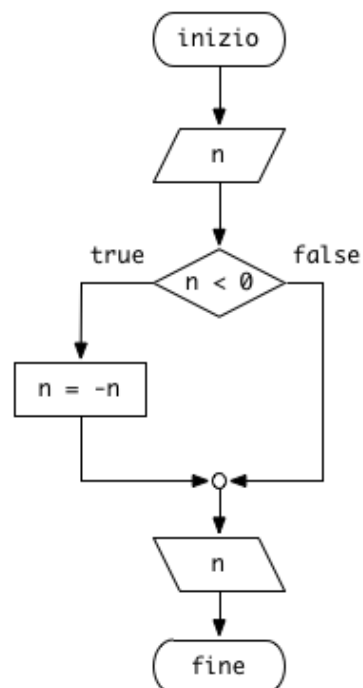
Si realizzi l'algoritmo che acquisito un valore intero calcola e visualizza il suo valore assoluto.



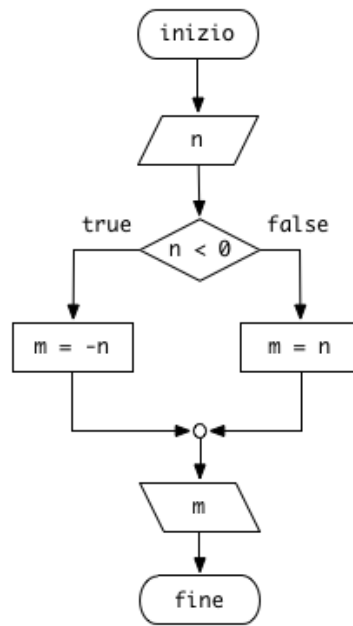
In questo caso l'algoritmo non calcola il valore assoluto, ma lo visualizza direttamente. Guardando la soluzione come una scatola nera, si ottiene il risultato richiesto, quindi soddisfa il comportamento atteso. Non rispetta la richiesta di calcolare il valore e poi visualizzarlo, ma questo aspetto è invisibile esternamente. Ci sono più *fine*, aspetto poco pulito perché diventa difficile riconciliare l'algoritmo.



Anche in questo caso non si calcola il valore assoluto, ma si visualizza il risultato. È più pulito l'utilizzo di un unico punto di *fine*.



Questo algoritmo effettivamente calcola il valore assoluto e poi lo visualizza. Di fatto sovrascrive il valore iniziale con il valore assoluto, quindi eventualmente si perde questa informazione. Dal punto di vista della scatola nera, anche questa soluzione fa ciò che è richiesto.



Questo algoritmo ha il comportamento richiesto e non perde il dato iniziale. L'approccio è migliore non tanto in relazione allo specifico problema (estremamente semplice) ma in termini di "buone" soluzioni, considerando che

- ◇ può essere necessario non tanto visualizzare il valore assoluto, ma utilizzarlo per ulteriori computazioni
- ◇ il "costo" di un elemento in più per memorizzare il valore assoluto è irrisorio.

Lezione 4 ◇ introduzione al C: struttura, tipi, operatori

24-09-2019

- ◇ struttura di un programma
 - dichiarazione di variabili
 - elaborazione
 - visualizzazione dei risultati
 - ◇ istruzioni
 - ◇ commenti `/* */`
 - ◇ tipi di dati di base: `int`, `float`, `double`, `char`
 - ◇ `return 0`
 - ◇ `#define`
 - ◇ input / output formattato
 - acquisizione formattata `scanf`
 - visualizzazione `printf`
-

Esercizio 5.1: Tempo in ore, minuti e secondi

Scrivere un programma che acquisito un valore intero che rappresenta un lasso di tempo espresso in secondi calcola e visualizza lo stesso tempo in ore, minuti e secondi.

Argomenti: assegnamento. commenti `/* */`, operatori.

Versione alternativa. Il numero di operazioni eseguite è lo stesso e le variabili risparmiate non fanno la differenza.

Lezione 5 ◇ costrutto di selezione `if`, `if-else`, `if-else-if`

26-09-2019

- ◇ cast implicito ed esplicito
 - ◇ costrutto di selezione `if`, `if-else`, `if-else-if`
 - valutazione dell'espressione tra parentesi
 - `if (a)` equivale a `if(a != 0)` e `if(!a)` equivale a `if(0 == a)`
 - attenzione al `==` (meglio scrivere `if(_costante_ == _variabile_)`)
 - semantica di `if(a = b)`
 - ordine di valutazione delle condizioni composte
 - relazione tra `else` e `if` in assenza di parentesi
 - ◇ De Morgan
-

Esercizio 6.1: Positivo, negativo o nullo

Scrivere un programma che acquisito un valore visualizza `+` se positivo, `-` se negativo, altrimenti, seguito da un carattere a-capo `'\n'`.

Argomenti: costrutto `if`

Versione alternativa.

Versione con `#define`.

Esercizio 6.2: Anno bisestile

Scrivere un programma che acquisito un valore intero positivo che rappresenta un anno, visualizza 1 se l'anno è bisestile, 0 altrimenti.

Argomenti: algoritmo. divisore/multiplo.

Esercizio 6.3: Terna pitagorica – Proposto

Scrivere un programma che acquisiti tre valori interi visualizzi 1 se costituiscono una terna pitagorica, 0 altrimenti.

Esercizio 6.4: Ordinamento crescente/decrescente – Proposto

Scrivere un programma che acquisisca un carattere e tre valori interi. Se il carattere è `+` visualizza i tre valori in ordine crescente, se è `-` li visualizza in ordine decrescente.

Argomenti: costrutto `if`. algoritmo.

Conversione del tempo in ore, minuti e secondi

Scrivere un programma che acquisito un valore intero positivo che rappresenta una durata temporale espressa in secondi, calcoli e visualizzi lo stesso intervallo di tempo espresso in ore, minuti e secondi.

Argomenti: algoritmo

Ingresso/Uscita:

input: un numero intero

output: tre interi separati da uno spazio (seguito da un carattere 'a-capo')

Alcuni casi di test per il collaudo:

input: 453

output: 0 7 33

input: 43268

output: 12 1 8

Resto

Scrivere un programma che acquisisce un intero e calcola e visualizza il numero di monete da 2 euro, 1 euro, 50, 20, 10, 5, 2 e 1 centesimo.

Argomenti: algoritmo

Ingresso/Uscita:

input: un numero intero

output: otto numeri interi

Alcuni casi di test per il collaudo:

input: 453

output: 2 0 1 0 0 0 1 1

input: 188

output: 0 1 1 1 1 1 1 1

Lezione 6 ◇ costruito ciclico `while` e `do-while`

30-09-2019

- ◇ costruito ciclico `while`
 - valutazione dell'espressione tra parentesi
 - esecuzione del corpo solo se l'espressione è vera
 - l'istruzione che aggiorna l'espressione è tipicamente l'ultima del corpo del ciclo
 - ◇ costruito ciclico `do-while`
 - valutazione dell'espressione tra parentesi
 - esecuzione del corpo almeno una volta
-

Esercizio 8.1: Minimo, massimo e valor medio

Scrivere un programma che acquisisce una sequenza di 53 valori interi e calcola e visualizza valor minimo, valor massimo e media dei valori.

indexProgrammi!Minimo, massimo e valor medio

Argomenti: costruito `while`

Esercizio 8.2: Fattoriale

Chiedere all'utente un valore non negativo, e fino a quando non è tale ripetere la richiesta, quindi calcolare e visualizzare il fattoriale.

Argomenti: costruito `while`, costruito `do-while`.

versione alternativa che distingue il caso limite $num = 0$ e $num = 1$ inutilmente.

versione alternativa che distrugge inutilmente il valore iniziale senza un reale risparmio

Esercitazione 1 ◇ costrutti elementari di controllo

01-10-2019

Esercizio 9.1: Conversione in binario, allo specchio

Scrivere un programma che acquisito un valore intero positivo visualizza la sua rappresentazione in base 2 (con i bit in ordine inverso).

Argomenti: algoritmo. resto della divisione, #define.

Esercizio 9.2: Potenza del 2

Scrivere un programma che acquisito un valore n intero positivo calcola e stampa la prima potenza del 2 superiore ad n .

Argomenti: algoritmo. cicli, while, #define.

Esercizio 9.3: Numero di cifre di un valore

Scrivere un programma che acquisito un valore intero calcola e visualizza il numero di cifre di cui è composto.

Esercizio 9.4: Einstein

Si dice che Einstein si divertisse molto a stupire gli amici con il gioco qui riportato.

Scrivete il numero 1089 su un pezzo di carta, piegatelo e datelo ad un amico che lo metta da parte. Ciò che avete scritto non deve essere letto fino a che non termina il gioco.

A questo punto chiedere ad una persona di scrivere 3 cifre qualsiasi (ABC), specificando che la prima e l'ultima cifra devono differire di almeno due ($|A - C| \geq 2$). Per fare atmosfera, giratevi e chiudete gli occhi. Una volta scritto il numero di tre cifre, chiedete all'amico di scrivere il numero che si ottiene invertendo l'ordine delle cifre (CBA). A questo punto fate sottrarre dal numero più grande quello più piccolo: $XYZ = |ABC - CBA|$ e fate anche calcolare il numero che si ottiene invertendo le cifre del numero risultante: ZYX . Infine, fate sommare questi ultimi due numeri: $XYZ + ZYX$ e constatate che il risultato è 1089.

Fate estrarre il foglio tenuto da parte fino a questo momento: 1089!

Realizzate un programma che fa i seguenti passi:

- ◇ Chiede all'utente di inserire un numero di 3 cifre, specificando il vincolo tra la prima e l'ultima cifra, e se i vincoli non sono rispettati chiede nuovamente il valore
- ◇ Acquisisce il valore
- ◇ Visualizza il numero inserito e il numero con le cifre in ordine inverso
- ◇ Visualizza la differenza tra i due valori (deve essere un numero positivo)
- ◇ Visualizza il numero con le cifre in ordine inverso
- ◇ Visualizza il risultato della somma tra questi due valori (dovrebbe essere 1089 per qualsiasi numero di 3 cifre che rispetta il vincolo tra la prima e l'ultima cifra)

Esercizio 9.5: Conversione in binario, senza array – Proposto e risolto

Acquisire un valore intero e calcolare e visualizzare la sua rappresentazione nel sistema binario, mostrando le cifre nell'ordine corretto. Se l'utente inserisce 6, il valore visualizzato è 110. Il suggerimento è che un numero binario può anche essere visto come un numero in base dieci, ossia 110 in binario può anche essere visto come centodieci in decimale ...

Argomenti: algoritmo. resto della divisione, #define. moltiplicazioni ripetute per la base.

Esercizio 9.6: Conversione in binario, allo specchio, con dimensione

Scrivere un programma che acquisisce un primo valore intero, il dato da convertire in binario, ed un secondo dato intero, il numero di bit su cui rappresentarlo. Il programma visualizza la rappresentazione in base 2 (con i bit in ordine inverso) del valore acquisito, eventualmente aggiungendo bit di padding.

Lezione 7 ◇ array monodimensionali e ciclo `for`

07-10-2019

- ◇ array monodimensionali
 - ◇ dati di tipo omogeneo
 - ◇ dimensione dell'array costante e nota a priori
 - ◇ indice degli elementi da 0 a `dimensione - 1`
 - ◇ costrutto ciclico a conteggio `for` (a condizione iniziale)
 - ◇ le tre parti del costrutto
 - istruzioni prima di iniziare il ciclo
 - condizioni, semplici e composte
 - istruzioni a chiusura del corpo del ciclo
 - ◇ separatore per le istruzioni nella prima e terza parte del costrutto
 - ◇ parti sono facoltative
-

Esercizio 10.1: 5 dati in ingresso in ordine inverso

Scrivere un programma che acquisiti 5 numeri interi li visualizza in ordine inverso.

Argomenti: array monodimensionale. cicli a conteggio. costrutto `for`.

Versione con array.

Esercizio 10.2: Sopra soglia

Scrivere un programma che acquisisce 20 valori interi, quindi un intero valore soglia e calcola e visualizza in numero di campioni strettamente superiori alla soglia.

Versione con il costrutto `for`.

Esercizio 10.3: Conversione in binario – Proposto

Scrivere un programma che acquisisce un valore compreso tra 0 e 1023, estremi inclusi e finché non è tale lo richiede. Quindi effettua la conversione in base 2 e la visualizza.

Argomenti: algoritmo. validazione ingresso. dimensione dell'array.

Versione alternativa con ciclo `do-while`.

Versione alternativa in cui si visualizza il numero sul numero completo di bit massimo.

Esercitazione 2 ♦ array monodimensionali

08-10-2019

Esercizio 11.1: Numero allo specchio

Scrivere un programma in C che acquisito un valore intero calcola e visualizza il valore ottenuto invertendo l'ordine delle cifre che lo compongono. Ad esempio, se il programma acquisisce il valore 251, il programma visualizza 152. Si ipotizzi (non si devono far verifiche in merito) che il valore acquisito non dia problemi di overflow e sia senz'altro strettamente positivo. Se l'utente inserisce il valore 1000, il programma visualizza 1 (questo perchè CALCOLA e poi visualizza). Se l'utente inserisce 9001 il programma visualizza 1009.

Argomenti: algoritmo.

Tratto dal tema d'esame del 04/09/2015 (Variante)

Esercizio 11.2: Conta lettere

Scrivere un programma che acquisisce una sequenza di 50 caratteri e per ogni carattere letto mostra quante volte compare nella sequenza, mostrandoli in ordine alfabetico. Si consideri che i caratteri inseriti siano tutti caratteri minuscoli. Per esempio, se l'utente inserisce `sequenzadiprova` (solo 16 caratteri in questo caso, per questioni di leggibilità), il programma visualizza

```
a 2
d 1
e 2
i 1
n 1
o 1
p 1
q 1
r 1
s 1
u 1
v 1
z 1
```

Argomenti: array monodimensionale. cicli a conteggio. contatori di occorrenze. codice ASCII di un carattere. costruito `for`.

Esercizio 11.3: Niente primi

Scrivere un programma che acquisisce una sequenza di al più 50 valori interi strettamente maggiori di 1 e che si ritiene terminata quando l'utente inserisce un valore minore o uguale a 1. Il programma, una volta acquisiti i dati, visualizza 1 se tra di essi non c'è alcun numero primo, 0 altrimenti.

Argomenti: array monodimensionale. cicli annidati. numero primo.

Lezione 8 ◇ array bidimensionali, struct e typedef

10-10-2019

- ◇ array pluri-dimensionali: dimensione 2
 - scansione mediante due cicli annidati
 - linearizzazione della memoria
 - ◇ struct
 - nome facoltativo, ma sempre utilizzato
 - array di struct
 - struct con campo array
 - ◇ typedef
-

Esercizio 12.1: Matrice identità

Si scriva un programma che acquisisce i dati di una matrice di dimensione 5x5 di valori interi. Il programma visualizza 1 se si tratta di una matrice identità, 0 altrimenti.

Argomenti: array bidimensionali. algoritmo.

Esercizio 12.2: Date: definizione di tipo

Definire un nuovo tipo di dato, cui dare nome `date_t`, per rappresentare date in termini di giorno, mese ed anno.

Argomenti: typedef.

```
1 typedef struct date_s {
2     int g, m, a;
3 } date_t;
```

Esercizio 12.3: Studente: definizione di tipo

Definire un nuovo tipo di dato, cui dare nome `student_t` per rappresentare le informazioni seguenti relative ad uno studente:

- ◇ cognome e nome: due array di caratteri di 41 caratteri ciascuno
- ◇ data di nascita e data di immatricolazione: due data
- ◇ voti esami: un array di NESAMI interi (dovete saperlo voi)
- ◇ media: voto medio (relativo agli esami superati)
- ◇ livello: 'T' o 'M' per distinguere se si è immatricolati alla triennale o alla magistrale.

Argomenti: typedef.

```
1 #define NL 41
2 #define NESAMI 18
3 typedef struct student_s {
4     char last[NL+1], first[NL+1];
5     date_t dnascita, dlaurea;
6     int grades[NEX];
7     float avg;
8     char level;
9 } student_t;
```

- ◇ sequenza di caratteri delimitata dal terminatore `'\0'`
 - ◇ allocazione dell'array con $N+1$ elementi
 - ◇ acquisizione
 - `scanf`: segnaposto `%s` e niente `&` davanti al nome dell'array
 - `gets`: serve poi solo il nome dell'array, trattandosi *sempre* di caratteri
 - ◇ scansione con condizione `s[i] != '\0'`
-

Esercizio 13.1: Stringa allo specchio

Scrivere un programma che acquisisce una stringa di al più 25 caratteri, inverte l'ordine di tutti i caratteri in essa contenuta e la visualizza.

Argomenti: stringa. algoritmo.

Esercizio 13.2: Stringa allo specchio senza vocali

Scrivere un programma che acquisisce una stringa di al più 25 caratteri, inverte l'ordine di tutti i caratteri ed elimina tutte le vocali in essa contenute. Ci sono solo caratteri minuscoli.

Argomenti: stringa. algoritmo.

versione alternativa

Esercizio 13.3: Anagrammi

Scrivere un programma che acquisisce due stringhe di al più 20 caratteri e determina se una sia l'anagramma dell'altra, visualizzando 1 in caso affermativo, 0 altrimenti.

Argomenti: algoritmo. stringhe. dimensione dell'array.

Numeri vicini

Scrivere un programma in C che chiede all'utente di inserire una sequenza di numeri interi terminata dallo 0 (lo 0 non fa parte della sequenza). Il programma ignora i valori negativi e valuta e stampa a video le coppie di numeri consecutivi che soddisfano tutte le condizioni che seguono:

- ◇ sono diversi tra di loro,
- ◇ sono entrambi numeri pari,
- ◇ il loro prodotto è un quadrato perfetto.

Argomenti: algoritmo. numeri pari. quadrato perfetto

Distanza di Hamming

Scrivere un programma che acquisiti due sequenze di 10 bit ciascuna (forniti una cifra alla volta), calcola e visualizza il vettore della distanza di Hamming, ed infine la distanza. Un esempio di esecuzione è il seguente:

```
1 0 0 1 0 0 1 0 0 1
1 1 1 1 0 0 0 1 0 1
0 1 1 0 0 0 1 1 0 0    4
```

Argomenti: array monodimensionale. cicli a conteggio. costruito `for`.

Carta di credito

Scrivere un programma che acquisisce un numero di carta di credito Visa costituito da 13 o 16 caratteri numerici e verifica che si tratti di un numero di carta valido oppure no. Nel primo caso visualizza 1, altrimenti 0. L'algoritmo che verifica la correttezza "sintattica" di un numero (inventato da Hans Peter Luhn di IBM) è il seguente:

- ◇ moltiplicare una cifra sì, una no, per 2 partendo dalla penultima a destra, quindi sommare tali cifre,
- ◇ sommare a tale valore, le cifre che non sono state moltiplicate per 2,
- ◇ se il valore ottenuto è un multiplo di 10, il numero è valido.



Per esempio, si consideri il numero di carta di credito seguente:
4003600000000014.

- ◇ moltiplicare una cifra sì, una no, per 2 partendo dalla penultima a destra, quindi sommare tali cifre (sono sottolineate le cifre da moltiplicare per 2):

4003600000000014

si moltiplica ogni cifra per 2:

$$1 \times 2 + 0 \times 2 + 0 \times 2 + 0 \times 2 + 0 \times 2 + 6 \times 2 + 0 \times 2 + 4 \times 2$$

si ottiene:

$$2 + 0 + 0 + 0 + 0 + 12 + 0 + 8$$

si sommano le cifre:

$$2 + 0 + 0 + 0 + 0 + 1 + 2 + 0 + 8 = 13$$

- ◇ sommare a tale valore, le cifre che non sono state moltiplicate per 2,
 $13 + 4 + 0 + 0 + 0 + 0 + 0 + 0 + 3 + 0 = 20$
- ◇ se il valore ottenuto è un multiplo di 10, il numero è valido: in questo caso le è.

Potete provare con alcuni numeri suggeriti da PayPal: 4111111111111111, 4012888888881881, 422222111212222.

Argomenti: array di caratteri. corrispondenza carattere numerico valore. cicli a conteggio. cifre di un numero.

Media mobile

Si scriva un programma che acquisiti 100 valori interi ed un valore intero n calcola e visualizza l'array di valori che costituiscono la media mobile dei dati in ingresso di finestra n . L'elemento i -esimo della media mobile viene calcolato come media degli n valori del vettore in ingresso che precedono e includono l'elemento i . Se l'elemento i è preceduto da meno di $n-1$ valori, la media si calcola su quelli.

Argomenti: array monodimensionali. calcolo della media.

Tratto dal tema d'esame del 03/07/2017 (variante)

Conta caratteri

Scrivere un programma in C che acquisisca una sequenza `str` di 20 caratteri. Per ogni carattere `car` contenuto nella stringa `str`, a partire dall'ultimo fino ad arrivare al primo, il programma visualizza (senza lasciare spazi) il carattere `car`, seguito dal numero di volte in cui compare consecutivamente in quel punto della stringa. Per esempio, se l'utente inserisce `aabbbddbbbbbhhhhhzzzz` il programma visualizza `z4h5b4d2b3a2`.

Argomenti: array monodimensionali. caratteri. algoritmo.

Tratto dal tema d'esame del 03/07/2017 (variante)

Esercitazione 3 ◇ array e stringhe

17-10-2019

Esercizio 15.1: Determinante di una matrice dimensione 3

Scrivere un programma che acquisisce i dati di una matrice di dimensione 3×3 di numeri interi, quindi calcola e visualizzare il determinante.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$\det(A) = a_{11} \times a_{22} \times a_{33} + a_{12} \times a_{23} \times a_{31} + a_{13} \times a_{21} \times a_{32} - a_{13} \times a_{22} \times a_{31} - a_{12} \times a_{21} \times a_{33} - a_{11} \times a_{23} \times a_{32}$$

Argomenti: array bidimensionale.

Esercizio 15.2: Nome di file da percorso, nome ed estensione

Scrivere un programma che acquisisce una stringa di al più 50 caratteri che contenga il nome di un file completo di percorso e visualizza il solo nome.

Argomenti: stringa. algoritmo.

Tratto dal tema d'esame del

oppure:

Esercizio 15.3: Da intero a stringa

Scrivere un programma che acquisito un intero crea una stringa che contiene le cifre dell'intero. L'intero ha al più 6 cifre.

Argomenti: stringa. algoritmo. carattere numerico.

Versione alternativa.

Esercizio 15.4: Area di un poligono

Scrivere un programma che calcola e visualizza l'area di un poligono, a partire dalle coordinate dei suoi vertici e usando la formula di Gauss. Il poligono ha al più 10 vertici, ciascuno rappresentato dalle coordinate x e y (valori interi). Il programma acquisisce prima il numero `num` di vertici del poligono, verificando che sia non superiore a 10, quindi acquisisce le coordinate dei `num` vertici, quindi procede al calcolo dell'area e la visualizza. Si dichiara un opportuno tipo di dato (`vertex_t`) per rappresentare i vertici del poligono.

Esercizio 15.5: Quadrato magico

Si scriva un programma che acquisisce i dati di una matrice di dimensione 3×3 di valori interi. Il programma visualizza 1 se si tratta di un quadrato magico seguito dal valore della somma, 0 altrimenti. Un quadrato magico è un insieme di numeri interi distinti in una tabella quadrata tale che la somma dei numeri presenti in ogni riga, in ogni colonna e in entrambe le diagonali dia sempre lo stesso valore.

Argomenti: array bidimensionali. algoritmo.

Versione alternativa.

Fattori

Scrivere un programma che acquisisce due numeri interi relativi e visualizza 1 se uno è un divisore dell'altro o viceversa, 0 altrimenti. Dopo il valore visualizzato, mettere un 'a-capo'.

Ingresso/Uscita:

input: due numeri interi

output: un intero (seguito da un carattere 'a-capo')

Alcuni casi di test per il collaudo:

input: 5542 18

output: 0

input: 5542 17

output: 1

input: 13 1950

output: 1

Padding

Si vuole rappresentare a video un valore naturale `num` utilizzando un numero a scelta di cifre `k` inserendo 0 nelle posizioni più significative, fino a raggiungere la dimensione desiderata. Per esempio, volendo rappresentare 842 su 5 cifre, si ottiene 00842.

Scrivere un programma che acquisisce due valori interi entrambi strettamente positivi (e finché non è così richiede il valore che non rispetta il vincolo) `num` e `k`, quindi rappresenta `num` su `k` cifre. Se `k` è minore del numero di cifre presenti in `num`, il programma visualizza il valore `num` come è. Dopo il valore visualizzato, mettere un 'a-capo'.

Ingresso/Uscita:

input: due numeri interi (da verificare)

output: un intero (seguito da un carattere 'a-capo')

Alcuni casi di test per il collaudo:

input: 11304 9

output: 000011304

input: -4 9000 -5 -2 2

output: 9000

input: 1 1

output: 1

Super Mario

Nella preistoria dei videogiochi in Super Mario della Nintendo, Mario deve saltare da una piramide di blocchi a quella adiacente. Proviamo a ricreare le stesse piramidi in C, in testo, utilizzando il carattere cancelletto (#) come blocco, come riportato di seguito. In realtà il carattere # è più alto che largo, quindi le piramidi saranno un po' più alte.

```
  #  #
 ## ##
### ###
#### ####
```



Notate che lo spazio tra le due piramidi è sempre costituito da 2 spazi, indipendentemente dall'altezza delle piramidi. Inoltre, alla fine delle piramidi **non ci devono essere spazi**. L'utente inserisce l'altezza delle piramidi, che deve essere un valore strettamente positivo e non superiore a 16. In caso l'utente inserisca un valore che non rispetta questi vincoli, la richiesta viene ripetuta.

Ingresso/Uscita:

input: un numero intero (da verificare)

output: una sequenza di caratteri

Scorrimento a destra – rightshift

Scrivere un programma che acquisita una stringa di al più 10 caratteri, modifica la stringa in modo tale che la stringa finale sia quella iniziale, fatta scorrere a destra di una posizione, con l'ultimo carattere riportato in testa. Se per esempio la sequenza iniziale è `attraverso`, la stringa finale sarà `oattravers`. Una volta modificata la stringa memorizzata, visualizzarla e farla seguire da un carattere 'a-capo'.

Ingresso/Uscita:

input: al più 10 caratteri

output: al più 10 caratteri (seguiti da un carattere 'a-capo')

Alcuni casi di test per il collaudo:

input: `attraverso`

output: `oattravers`

input: `ananas`

output: `sanana`

input: `trampolini`

output: `itrampolin`

Troncabile primo a destra

Scrivere un programma che acquisisce un valore intero strettamente positivo, e finché non è tale lo richiede. Il programma analizza il valore intero e visualizza 1 nel caso sia un troncabile primo a destra, 0 altrimenti. Un numero si dice troncabile primo a destra se il numero stesso e tutti i numeri che si ottengono eliminando una alla volta la cifra meno significativa del numero analizzato al passo precedente, sono numeri primi. Per esempio, se il numero iniziale è 719, i numeri che si ottengono “eliminando una alla volta la cifra meno significativa del numero analizzato al passo precedente ..” sono 71 e 7. Dopo il valore visualizzato, mettere un ‘a-capo’.

Ingresso/Uscita:

input: un intero (da verificare)

output: un intero (seguito da un carattere ‘a-capo’)

Alcuni casi di test per il collaudo:

input: 719

output: 1

input: 473

output: 0

input: -42 -18 311111

output: 0

input: 3137

output: 1

- ♦ intestazione
- ♦ tipo di dato restituito
- ♦ parametri formali e attuali
- ♦ prototipo
- ♦ return
- ♦ tipo void
- ♦ passaggio array monodimensionali e dimensione
- ♦ visibilità delle variabili

Esercizio 17.1: Combinazioni

Scrivere un programma che acquisisce due interi n e k strettamente positivi, con k non superiore a n e finchè non è tale li richiede (entrambi). Quindi calcola e visualizza il numero di combinazioni di n elementi, presi k per volta.

versione con sottoprogrammi

Esercizio 17.2: Massimo di un array

Scrivere un sottoprogramma che ricevuto in ingresso un array di numeri interi e *qualsiasi altro parametro ritenuto strettamente necessario* restituisce l'indice dell'elemento con valore massimo.

Argomenti: sottoprogrammi. passaggio array monodimensionale.

Esercizio 17.3: Triangolo

Scrivere un sottoprogramma che ricevuti in ingresso tre interi, restituisce 1 nel caso in cui si tratti delle dimensioni dei lati di un triangolo valido 0 altrimenti. Le condizioni che devono sussistere sono:

- ♦ le dimensioni sono positive
- ♦ la somma di due dimensioni non è mai superiore alla terza dimensione

- ♦ passaggio parametri per copia valore, per copia indirizzo
- ♦ record di attivazione

Esercizio 18.1: Massimo, minimo e media dei valori di un array

Scrivere un sottoprogramma che ricevuto in ingresso un array di numeri interi e *qualsiasi altro parametro ritenuto strettamente necessario trasmette* al chiamante l'indice dell'elemento con valore massimo, quello con valore minimo e la media dei valori dell'array.

Argomenti: sottoprogrammi. passaggio array monodimensionale. passaggio parametri per indirizzo.

- ◇ passaggio di array bidimensionali
- ◇ passaggio di `struct`

Esercizio 19.1: Lunghezza di una stringa

Scrivere il sottoprogramma che ricevuta in ingresso una stringa, calcola e restituisce la sua lunghezza.

Esercizio 19.2: Matrice identità

Scrivere un sottoprogramma che ricevuto in ingresso un array bidimensionale e qualsiasi altro parametro ritenuto strettamente necessario restituisce 1 se si tratta di una matrice identità, 0 altrimenti. La dimensione dell'array dichiarata è `N`.

Esercizio 19.3: Vertici di un poligono

Si consideri il seguente tipo di dato per rappresentare punti nello spazio piano.

```
1 typedef struct p {  
2     int x, y;  
3 } punto_t;
```

Si scriva un sottoprogramma che ricevuto in ingresso un array di tipo `punto_t` e qualsiasi altro parametro ritenuto strettamente necessario acquisisca i dati relativi ai vertici di un poligono.

Esercizio 19.4: Vertice?

Con riferimento al tipo prima definito, scrivere un sottoprogramma che ricevuto in ingresso un array di tipo `punto_t` e qualsiasi altro parametro ritenuto strettamente necessario, ed un punto, determini se il punto è un vertice del poligono o meno, restituendo rispettivamente 1 o 0.

Versione con il passaggio per indirizzo

Esercizio 19.5: Combinazione stringhe – Proposto

Programma che acquisisce due stringhe di al più 20 caratteri ciascuna e consente all'utente di selezionare una tra le seguenti operazioni:

- ◇ 1. verificare se la prima stringa è contenuta nella seconda (in caso positivo visualizza 1, 0 altrimenti)
- ◇ 2. verificare se la seconda stringa è contenuta nella prima (in caso positivo visualizza 1, 0 altrimenti)
- ◇ 3. concatena la seconda stringa alla prima e visualizza il risultato
- ◇ 4. concatena la prima stringa alla seconda e visualizza il risultato
- ◇ 5. concatena le stringhe in ordine alfabetico crescente e visualizza il risultato
- ◇ 6. termine programma

Il programma chiede ripetutamente all'utente quale operazione svolgere, acquisisce le stringhe, svolge l'operazione richiesta e ripresenta il menù, fino a quando l'utente decide di terminare il programma, selezionando l'opportuna voce del menù.

Versione con sottoprogrammi

Esercizio 19.6: Cifra del numero – Proposto

Scrivere un sottoprogramma `cifra` che riceve come parametri due interi `num` e `k`. Se `k` è strettamente positivo, il sottoprogramma calcola e restituisce la `k`-esima cifra del numero `num` a partire da destra. Nel caso in cui `k` non sia strettamente positivo o `k` sia maggiore del numero effettivo di cifre di `num`, il sottoprogramma restituisce `-1`.

Scrivere un programma che chiede all'utente i due valori `num` e `k` ed invoca il sottoprogramma `cifra` visualizzando poi il risultato, seguito da un carattere a-capo `'\n'`.

Argomenti: sottoprogrammi. algoritmo.

Somma a k

Scrivere un programma che acquisisce una sequenza di al più 100 valori interi e un intero strettamente positivo k . L'acquisizione della sequenza termina quando l'utente inserisce un numero negativo o nullo, oppure quando vengono acquisiti 100 valori. Il programma visualizza 1 se la sequenza contiene due valori tali che la loro somma sia k , 0 altrimenti. Dopo il valore visualizzato, mettere un 'a-capo'. Per realizzare la soluzione si sviluppi un sottoprogramma `cercasomma` che ricevuto in ingresso k , l'array contenente i dati e qualsiasi altro parametro ritenuto strettamente necessario, restituisce 1 o 0 nel caso trovi i due valori la cui somma è k .

Ingresso/Uscita:

input: una sequenza di al più 101 valori interi

output: un intero (seguito da un carattere 'a-capo')

Alcuni casi di test per il collaudo:

input: 10 15 3 7 -4 17

output: 1

input: 2 5 6 55 10 -11 100

output: 0

input: 2 2 3 -8 7

output: 0

Solo in ordine

Scrivere un programma che acquisisce una sequenza di al più 20 valori interi, chiedendo all'utente inizialmente quanti valori vorrà fornire, `num`. Il programma acquisisce `num` valori e memorizza in una opportuna struttura dati la sequenza di valori i cui elementi sono strettamente crescenti, trascurando i valori che risultano non essere ordinati. Al termine dell'acquisizione il programma visualizza la lunghezza della sequenza, seguita, su una nuova riga, dalla sequenza stessa. L'utente inserirà sempre un numero di valori coerente con la richiesta. Avvalersi di due sottoprogrammi: `fillarrord` e `viewarr`: il primo memorizza i dati ritenuti validi, il secondo visualizza il contenuto di un array.

Ingresso/Uscita:

input: sequenza di interi

output: sequenza di interi

Alcuni casi di test per il collaudo:

input: 10
3 1 4 5 -1 3 1 4 5 -1

output: 3

3 4 5

input: 6

-1 3 -1 3 -1 3

output: 2

-1 3

input: 8

9 8 7 6 5 4 3 2

output: 1

9

Quadro di parole

Scrivere un programma che acquisisce un valore intero strettamente positivo `num`, che rappresenta il numero di parole (ciascuna di al più 25 caratteri) che verranno poi fornite, e che comunque non saranno mai più di 20. Il programma acquisisce le `num` parole e le visualizza, una per riga, all'interno di un rettangolo creato dal carattere `*`.

Per esempio, se l'utente fornisce:

```
5
Hello
world
in
un
rettangolo
```

il programma visualizza:

```
*****
*Hello      *
*world      *
*in         *
*un         *
*rettangolo*
*****
```

Ingresso/Uscita:

input: un intero e una sequenza di stringhe

output: una sequenza di caratteri

Mix di due array ordinati

Scrivere un sottoprogramma che acquisisce due sequenze di valori interi, ciascuna di 20 elementi. Il programma ordina le due sequenze in senso crescente, quindi visualizza la sequenza dei valori acquisiti, in senso crescente e senza ripetizioni. Al termine dell'esecuzione, le due sequenze sono ordinate. Nel realizzare la soluzione, scrivere un sottoprogramma `sortarr` che ricevuto in ingresso un array, un intero `updown` e qualsiasi altro parametro ritenuto strettamente necessario, ordina il contenuto dell'array in senso crescente se `updown` vale 1, in senso decrescente se vale -1

Ingresso/Uscita:

input: quaranta sequenze di numeri positivi

output: una sequenza di numeri positivi

Alcuni casi di test per il collaudo:

```
input:  -1 2 4 2 5 6 8 1 0 7 3 4 9 -9 9 9 9 9 9 9
        7 3 4 5 6 7 8 9 2 1 0 5 6 8 1 0 1 2 4 2
output: -9 -1 0 1 2 3 4 5 6 7 8 9

input:  9 8 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 0 0 0
input:  1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
output: 0 1 2 3 4 5 6 7 8 9
```

Sottostringa piú lunga senza ripetizioni

Scrivere un programma che acquisita una stringa di al piú 30 caratteri, individui la sottostringa piú lunga in essa contenuta, senza caratteri ripetuti. Il programma visualizza la lunghezza di tale sottostringa, seguita da un carattere 'a-capo'.

Ingresso/Uscita:

input: una stringa

output: un intero

Alcuni casi di test per il collaudo:

input: abcabcbb

output: 3

input: alfabeto

output: 7

input: bbbbbb

output: 1

Esercizio 21.1: Confronta lunghezza delle stringhe

Scrivere un sottoprogramma che riceve due stringhe in ingresso e:

- ◇ restituisce -1 se la prima stringa è più corta della seconda
- ◇ restituisce 0 se le due stringhe hanno la stessa lunghezza
- ◇ restituisce 1 se la prima stringa è più lunga della seconda

Scrivere quindi un programma che chiede all'utente due stringhe, che si assume abbiano lunghezza massima di 50 caratteri, e stampa:

- ◇ "piu' corta" se la prima stringa è più corta della seconda
- ◇ "uguale" se le due stringhe hanno la stessa lunghezza
- ◇ "piu' lunga" se la prima stringa è più lunga della seconda

Argomenti: Stringhe. Lunghezza stringhe. Sottoprogrammi.

Esercizio 21.2: Selection sort

Il selection sort è un algoritmo di ordinamento che, dato un array di lunghezza n , ripete i seguenti passi (partendo da $i = 0$):

- ◇ seleziona il minimo elemento nella sezione di array che va da i a $n - 1$ (compreso)
- ◇ scambia l'elemento trovato con quello in posizione i
- ◇ incrementa l'indice i

Scrivere un sottoprogramma che riceve in ingresso un array di interi e tutti i parametri aggiuntivi che si ritengono necessari ed esegue un selection sort su di esso.

Scrivere quindi un programma che riceve in ingresso 10 numeri interi e li stampa ordinati in modo crescente.

Argomenti: Array monodimensionali. Algoritmi di ordinamento. Sottoprogrammi.

Esercizio 21.3: Matrice trasposta

Scrivere un sottoprogramma che data in ingresso una matrice di interi e tutti i parametri strettamente necessari, calcola e restituisce la sua trasposta.

Scrivere quindi un programma che dati in ingresso due interi, rispettivamente il numero di righe e il numero di colonne (che si assumono validi e inseriti correttamente), e una matrice di interi di quelle dimensioni (quindi `numero_righe * numero_colonne` interi) ne calcola e stampa la trasposta. Si assuma che la matrice abbia al massimo 15 righe e 20 colonne.

Argomenti: Array bidimensionali. Sottoprogrammi.

Esercitazione 6 ♦ sottoprogrammi

29-10-2019 (CB)

Esercizio 22.1: Prepara un cocktail

Si vuole scrivere un programma che, dati gli ingredienti di un cocktail, ne definisca la ricetta e la gradazione alcolica. Si assume di avere solamente ingredienti liquidi nella ricetta e che tali ingredienti non siano mai più di 10. Si ricorda, inoltre, che nelle ricette dei cocktail le quantità degli ingredienti sono definite in parti (che non necessariamente sono intere).

Definire una struttura dati che possa memorizzare le informazioni riguardanti un ingrediente di un cocktail, che sono:

- ♦ nome dell'ingrediente (dalla lunghezza massima di 50 caratteri)
- ♦ gradazione alcolica dell'ingrediente in percentuale (che si assume intera)
- ♦ numero di parti nel cocktail

Scrivere un programma che:

- ♦ prende in ingresso un intero n che corrisponde al numero di ingredienti nel cocktail, se il numero è minore di 1 o maggiore di 10, l'inserimento deve essere ripetuto;
- ♦ chiede in input, per ogni ingrediente, il nome, la gradazione alcolica e il numero di parti;
- ♦ richiede quanti litri l di cocktail si vogliono preparare;
- ♦ stampa, per ogni ingrediente, la quantità necessaria per la preparazione di l litri di cocktail;
- ♦ stampa la gradazione alcolica del cocktail

Argomenti: `struct`. `typedef`. Array monodimensionali.

Esercizio 22.2: Controlla Sudoku

Il Sudoku è un gioco di logica in cui lo scopo è quello di completare una matrice 9×9 inserendo numeri tra 1 e 9. Il gioco termina quando la matrice è piena e rispetta i seguenti vincoli:

- ♦ ogni riga contiene tutti i numeri da 1 a 9
- ♦ ogni colonna contiene tutti i numeri da 1 a 9
- ♦ ogni matrice 3×3 ottenuta raggruppando righe e colonne contigue contiene tutti i numeri da 1 a 9. In questo gruppo di matrici si includono:
 - la matrice ottenuta prendendo le righe 0-2 delle colonne 0-2;
 - la matrice ottenuta prendendo le righe 0-2 delle colonne 3-5;
 - la matrice ottenuta prendendo le righe 0-2 delle colonne 6-8;
 - la matrice ottenuta prendendo le righe 3-5 delle colonne 0-2;
 - ...

Scrivere un sottoprogramma per ogni vincolo che, data una matrice di interi 9×9 e tutti i parametri strettamente necessari, restituisce 1 se il vincolo è rispettato, 0 altrimenti. Scrivere infine un sottoprogramma che, data una matrice di interi 9×9 e tutti i parametri strettamente necessari, restituisce 1 se tale matrice rappresenta una soluzione valida per il gioco del Sudoku, 0 altrimenti.

Argomenti: Sottoprogrammi. Array bidimensionali.

Esercizio 22.3: Zoo

Definire un tipo di dato che contenga informazioni sugli animali di uno zoo. Il nuovo tipo deve contenere:

- ♦ nome dell'animale, es: "Leone" (lunghezza massima 50 caratteri)

-
- ◇ specie dell'animale, es: "Panthera Leo" (lunghezza massima 50 caratteri)
 - ◇ numero di esemplari adulti nello zoo
 - ◇ numero di cuccioli nello zoo

Scrivere i seguenti sottoprogrammi:

- ◇ dato in ingresso l'array di animali dello zoo e tutti gli altri parametri strettamente necessari, restituisce il numero totale di animali nello zoo;
- ◇ dato in ingresso l'array di animali dello zoo e tutti gli altri parametri strettamente necessari, restituisce la percentuale di cuccioli rispetto al totale degli animali nello zoo;
- ◇ dato in ingresso l'array di animali dello zoo, una stringa contenente il nome di una specie e tutti gli altri parametri strettamente necessari, restituisce il numero totale di animali di quella specie all'interno dello zoo. Si consiglia di definire un sottoprogramma di supporto che, date in ingresso due stringhe, restituisce 1 se le due stringhe sono identiche, 0 altrimenti.

Argomenti: `struct`. `typedef`. Array monodimensionali. Stringhe. Sottoprogrammi.

Esercitazione 7 ◇ array, stringhe

31-10-2019

Esercizio 23.1: Lunghezza di una stringa

Scrivere un sottoprogramma che ricevette in ingresso due stringhe restituisce 1 se non hanno caratteri in comune, 0 altrimenti.

Esercizio 23.2: Stringa nella stringa

Scrivere un sottoprogramma che ricevette in ingresso due stringhe restituisce 1 se la seconda è una sottostringa della prima (è contenuta nella prima).

Esercizio 23.3: Cifre divisori

Scrivere un sottoprogramma che ricevuto in ingresso un valore intero calcola e restituisce il numero di cifre del numero che sono divisori del numero stesso.

Esercizio 23.4: Insieme unione ordinato

Scrivere un sottoprogramma che ricevuti in ingresso due array di numeri interi e qualsiasi altro parametro ritenuto strettamente necessario, ordina entrambi gli array in senso crescente e *visualizza* l'insieme unione ordinato.

♦ utilità dei file, differenze tra file ASCII e file binari.

♦ accesso a file ASCII

- `FILE * fopen(char [], char []);`
- `int fclose(FILE *);`
- `int fscanf(FILE *, char [], ...);`
- `int fprintf(FILE *, char [], ...);`
- `char * fgets(char [], int, FILE *);`
- `int feof(FILE *);`

♦ lettura fino alla fine del file:

```
1 fscanf(fp, "%c", &car);
2 while(!feof(fp)){
3     /* utilizzo del valore letto e memorizzato in car */
4     /* ... */
5     fscanf(fp, "%c", &car);
6 }
```

♦ accesso a file binari

- `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`
- `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`

Esercizio 24.1: Numeri pari e dispari

Scrivere un programma che legge 100 valori interi dal file di testo ASCII `./dati.txt` e scrive in numeri pari nel file `./pari.txt` e quelli dispari nel file `./dispari.txt`.

Esercizio 24.2: Numeri pari e dispari binari

Scrivere un programma che legge 100 valori interi dal file binario `./dati.bin` e scrive in numeri pari nel file `./pari.txt` e quelli dispari nel file `./dispari.txt`.

Esercizio 24.3: Numeri primi .. chissà quanti

Scrivere un programma che legge dal file di testo ASCII `./dati.txt` numeri interi (non si sa quanti dati il file contiene) e scrive i numeri primi nel file `primi.txt`.

Esercizio 25.1: Conta vocaboli

Scrivere un programma che chiede all'utente il nome di un file (al più 40 caratteri) di testo e conta e visualizza il numero di parole presenti nel file. Le parole sono separate esclusivamente da spazi e sono al più di 32 caratteri.

Argomenti: file ASCII. Accesso a file con contenuto di lunghezza non nota a priori.

Quest'altra soluzione non funziona: si finisce in un ciclo infinito. Per quanto il file sia finito, l'istruzione `fscanf` non cessa di essere eseguita e legge sempre l'ultimo vocabolo, restituendo 1.

Esercizio 25.2: Copia file lowercase

Scrivere un programma che chiede all'utente il nome di un file (al più 40 caratteri) di testo e scrive un nuovo file di testo, il cui nome è uguale a quello inserito con l'aggiunta del suffisso `"_lowercase"`. Il contenuto del nuovo file deve essere identico a quello del file originale, ma con le lettere maiuscole trasformate in minuscole.

Argomenti: file ASCII. Accesso a file con contenuto di lunghezza non nota a priori.

Esercizio 25.3: Record di un Videogame

Abbiamo quasi terminato di implementare il nostro nuovo videogame, manca solo la possibilità di memorizzare i migliori record. Il file del salvataggio, che sarà in formato binario, conterrà (in successione) il numero di record che sono salvati all'interno del file e l'elenco dei record.

Come ogni videogioco che si rispetti, a ogni record è associato il nome del suo autore, di esattamente 3 lettere, e il rispettivo punteggio. Inoltre è possibile memorizzare al più i 10 record migliori.

Definire un nuovo tipo di dato che possa contenere le informazioni riguardanti un record. Scrivere quindi i sottoprogrammi `load` e `save` che, dati il nome del file, l'array dei migliori record e tutti i parametri strettamente necessari:

- ◇ legga dal file indicato i record (`load`)
- ◇ scriva i record nel file indicato (`save`)

Facoltativo: Scrivere un sottoprogramma `insert` che, dato l'array dei migliori record, un nuovo record e tutti i parametri strettamente necessari, inserisca il nuovo record tra i migliori, se necessario.

Argomenti: file binario. `typedef`. `sizeof`.

- ◇ allocazione dinamica: utilità
- ◇ `malloc`
- ◇ `free`
- ◇ tipo `void *`

Esercizio 26.1: Visualizza sequenza di numeri interi in ordine inverso

Scrivere un programma che chiede all'utente la quantità `n` di valori interi che intende inserire, quindi acquisisce `n` e li visualizza in ordine inverso.

Argomenti: allocazione dinamica. `malloc`. `free`. `sizeof`.

Esercizio 26.2: Stringa di vocali

Scrivere un sottoprogramma che ricevuta in ingresso una stringa, crea una nuova stringa contenente tutte e sole le vocali contenute nella stringa di ingresso – di dimensioni strettamente idonee a contenere tale stringa – e la restituisce al chiamante.

Esercizio 26.3: Primi nel file

Scrivere un sottoprogramma che ricevuto in ingresso il nome di un file restituisce al chiamante tutti i numeri primi in esso contenuti.

Argomenti: sottoprogramma. numero primo.

versione in cui si usano due `return`
versione in cui si passano entrambi i parametri per indirizzo
versione in cui si crea una struttura dati e la si restituisce per valore

Esercizio 26.4: Anagramma – Proposto

Scrivere un sottoprogramma che riceve in ingresso due stringhe (senz'altro di lunghezza uguale) e restituisce 1 se le stringhe sono una l'anagramma dell'altra, 0 altrimenti.

Argomenti: sottoprogramma. stringhe. numero primo.

- ◇ allocazione dinamica: liste concatenate semplici
- ◇ tipo di dato per elementi della lista
 - testa della lista
 - elementi generici
 - ultimo elemento della lista

Esercizio 27.1: Duplica stringa

Scrivere un sottoprogramma che ricevuta in ingresso una stringa ne crea e restituisce una con lo stesso contenuto.

Argomenti: allocazione dinamica. malloc. sizeof. strdup

Esercizio 27.2: itos

Scrivere un sottoprogramma che ricevuto in ingresso un valore intero positivo restituisce la stringa i cui caratteri sono le cifre del valore in ingresso.

Esercizio 27.3: Tipo da lista

Definire un tipo di dato opportuno per la realizzazione di una lista concatenata semplice per la gestione dei valori interi.

Operazioni da file

Scrivere un programma che legga dal file `operazioni.txt` un elenco a priori di lunghezza ignota di operazioni **tra valori interi** da eseguire, una per riga. L'operazione tra due operandi interi è presentata nel formato:

`<operando1> <operazione> <operando2>`

Tra il primo operando, l'operatore e il secondo operando c'è esattamente uno spazio, e gli operatori utilizzati sono solamente `+`, `-`, `*` o `/` (non effettuare controlli, è senz'altro così).

Il programma visualizza i risultati delle operazioni presenti nel file. Nella soluzione avvalersi di un sottoprogramma (che dovrete sviluppare) che riceve in ingresso i due operandi e l'operatore e restituisce al chiamante il risultato dell'operazione. Dopo la sequenza di risultati visualizzata, mettere un `'a-capo'`. Per esempio, se il contenuto del file `operazioni.txt` è il seguente

```
5 + 4
10 / 9
4 * 7
1520 - 5567
184 * 2
```

il programma visualizza: `9 1 28 -4047 368`

Ingresso/Uscita:

input: file di testo contenente stringhe

output: una sequenza di interi

Valore mediano di un array

Il file di testo `mediana.txt` contiene una sequenza di al più 100 interi, preceduta da un intero `n` che rappresenta il numero di valori successivamente presenti nel file (non ci saranno inconsistenze, se `n` vale 12, nel file ci sono poi 12 valori interi).

Scrivere un programma che visualizzi il valore mediano `M` dei numeri presenti nel file. Il valore mediano cercato `M` è il valore che occupa la posizione centrale nella distribuzione ordinata dei valori.

Per risolvere il problema, sviluppare due sottoprogrammi: `ordina` e `mediano`, il primo per ordinare un insieme di valori, il secondo per calcolare il valore mediano.

Dopo il valore visualizzato, mettere un 'a-capo'.

Per esempio, se il contenuto del file è il seguente:

```
10
3
5
97
45
68
32
15
20
1000
24
```

il programma visualizza 32.

Ingresso/Uscita:

input: file di testo contenente valori interi

output: un intero (seguito da un carattere 'a-capo')

Prodotto tra matrici

Scrivere un programma che acquisisce i dati di due matrici di valori interi da un file binario `mats.bin`, e ne effettua il prodotto, visualizzando il risultato (seguito dal carattere 'a-capo'). Il contenuto del file binario è il seguente:

```
numero di righe matrice 1
numero di colonne matrice 1
numero di colonne matrice 2
valori matrice 1
valori matrice 2
```

(Si noti che il numero di colonne della matrice 1 è pari al numero di righe della matrice 2 affinché il prodotto possa essere fatto.) Le matrici hanno senz'altro dimensione non superiore a 50x50.

Per esempio, se il file contiene (qua visualizzato non in binario):

```
3
4
2
12 54 99 5
2 45 68 33
10 22 888 4
197 -85
264 -8
487 1
-4 -8
```

il programma visualizza:

```
64813 -1393
45258 -726
440218 -170
```

Ingresso/Uscita:

input: un file binario

output: una sequenza di interi organizzati su più righe

Ruota immagine a destra

Un'immagine in bianco e nero è rappresentata in un sistema di calcolo come un array bidimensionale di valori interi tra compresi nell'intervallo $[0, 255]$, ciascuno rappresentante il valore di intensità del pixel in quella posizione. Scrivere un programma che acquisita un'immagine da un file il cui nome viene chiesto all'utente (ed è al più di 30 caratteri), la ruota di 90 gradi in senso orario. L'immagine ha una dimensione massima di 100×100 pixel.

Il formato del file contenente l'immagine è il seguente:

```
<numero righe immagine> <numero colonne immagine>  
<valori dei pixel dell'immagine>
```

Per esempio, se il file contiene:

```
4 6  
2 55 79 3 218 24  
41 5 46 233 96 9  
73 34 11 0 109 68  
35 85 17 45 20 219
```

il programma visualizza:

```
35 73 41 2  
85 34 5 55  
17 11 46 79  
45 0 233 3  
20 109 96 218  
219 68 9 24
```

Ingresso/Uscita:

input: un file di interi

output: sequenze di interi organizzati per righe

Somma dei quadrati delle differenze tra immagini

Confrontare due immagini (o parti di immagini) e capire quanto sono simili è un problema molto comune in Computer Vision. Un modo per capire quanto due immagini sono simili consiste nel calcolare la somma dei quadrati delle differenze pixel per pixel, metodo noto con il nome di Sum of Squared Differences o SSD. Scrivere un programma che dati due file contenenti due immagini (con le stesse dimensioni di al più 50x50 pixel) calcoli l'indice SSD, ovvero:

$$SSD = \frac{1}{n} \sum_{i=0}^{n_r} \sum_{j=0}^{n_c} (I_1(i, j) - I_2(i, j))^2 \quad (1)$$

dove n è il numero di pixel di una immagine, n_r è il numero di righe, n_c il numero di colonne e $I_x(i, j)$ è il valore del pixel (i, j) nell'immagine x . Si calcoli il valore di SSD tramite un sottoprogramma opportuno che riceve in ingresso le due immagini e restituisce il valore calcolato. Il programma chiede all'utente il nome dei due file contenenti le immagini (ciascuno al più 30 caratteri).

Il formato del file contenente un'immagine è:

```
<numero righe immagine> <numero colonne immagine>
<valori dei pixel dell'immagine>
```

Per esempio, date le immagini:

```
3 4
25 65 48 88
100 251 64 95
65 10 25 45
```

```
3 4
28 65 55 94
100 254 70 95
77 20 30 45
```

il valore visualizzato è 9547.083008

Ingresso/Uscita:

input: due file di testo contenente interi

output: un valore reale (seguito da un carattere 'a-capo')

Lezione 15 ◇ liste concatenate semplici

18-11-2019

Esercizio 29.1: Visualizza in ordine inverso

Scrivere un programma che acquisisce una sequenza di valori interi a priori di lunghezza ignota e che si ritiene terminata quando l'utente inserisce il valore 0, e la visualizza in ordine inverso.

Argomenti: lista concatenata semplice. `push`. `emptylist`. `printlist`

Esercizio 29.2: `push`

Scrivere un sottoprogramma `push` che ricevuta in ingresso una lista ed un valore intero, inserisce l'elemento *in testa* alla lista.

Argomenti: `push`. `malloc`. `sizeof`.

Esercizio 29.3: `append`

Scrivere un sottoprogramma `append` che ricevuta in ingresso una lista ed un valore intero, inserisce l'elemento *in coda* alla lista.

Argomenti: `append`. `malloc`. `sizeof`.

Esercizio 29.4: `emptylist`

Scrivere un sottoprogramma `emptylist` che ricevuta in ingresso una lista ne elimina il contenuto, restituendo la lista vuota.

Argomenti: `emptylist`. `free`.

Esercizio 29.5: `itos`

Scrivere un sottoprogramma che ricevuta in ingresso una lista la restituisce dopo aver invertito l'ordine dei suoi elementi.

Argomenti: `emptylist`.

Esercitazione 9 ◇ liste concatenate semplici

19-11-2019

Esercizio 30.1: Quanta memoria!

Si considerino i 4 stralci di codice qua riportati: per ciascuno di essi si vuole calcolare la quantità di memoria allocata per le variabili dichiarate ed utilizzate (quindi la memoria allocata nel momento in cui si arriva all'ultima istruzione del codice riportato). Si utilizzi l'operatore `sizeof`. Per i primi due stralci, il calcolo è già stato fatto, riportare il risultato nell'apposito spazio il calcolo per gli ultimi due stralci.

<pre>1 int a; 2 ... 3 scanf("%d", &a);</pre>	<pre>1 int v[NUM], i; 2 ... 3 for(i = 0; i < NUM ; i++) 4 scanf("%d", &v[i]);</pre>	<pre>1 int * p, i, ndati; 2 ... 3 if(p = (int *)malloc(ndati* sizeof(int))){ 4 for (i = 0; i < ndati; i++) 5 scanf("%d", p+i); 6 ...</pre>	<pre>1 typedef struct _s { 2 int val; 3 struct _s * next; 4 } t_listi; 5 ... 6 t_listi * head = NULL; 7 int i, n, ndati; 8 ... 9 ndati = 0; 10 scanf("%d", &n); 11 while(n != STOP){ 12 head = append(head , n); 13 ndati++; 14 scanf("%d", &n); 15 }</pre>
---	---	--	---

quantità di memoria complessiva allocata:

<pre>1 sizeof(int) 2 3 .</pre>	<pre>1 NUM * sizeof(int) + 2 sizeof(int) 3 .</pre>	<pre>1 2 3 .</pre>	<pre>1 2 3 .</pre>
--	---	----------------------------	----------------------------

Esercizio 30.2: Insieme unione

Scrivere un sottoprogramma che ricevette in ingresso due liste che rappresentano insiemi di valori interi, restituisce una nuova lista rappresentante l'unione dei due insiemi.

Argomenti: Unione di insiemi. find.

Esercizio 30.3: Elimina doppioli adiacenti

Scrivere un sottoprogramma che ricevuta in ingresso una lista elimina dalla lista le ripetizioni di elementi *adiacenti*.

Argomenti: lista concatenata semplice. deleteptr.

Per esempio, se la lista in ingresso è quella di seguito riportata e `start` è 1:

$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 1 \rightarrow |$

il sottoprogramma restituisce la lista seguente

$$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow |$$

Esercizio 30.4: Da lista a insieme

Scrivere un sottoprogramma che ricevuta in ingresso una lista la manipola facendola diventare un insieme (elimina gli elementi ripetuti).

Argomenti: lista concatenata semplice.

Per esempio, se la lista in ingresso è quella di seguito riportata e `start` è 1:

$$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 1 \rightarrow |$$

il sottoprogramma restituisce la lista seguente

$$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow |$$

Esercizio 30.5: Riempi lista

Scrivere un sottoprogramma che ricevuto in ingresso un valore intero `num` crea una lista chiedendo all'utente `num` valori interi e la restituisce all'utente. Anche se sappiamo quanti dati vogliamo memorizzare (e quindi una unica istruzione `malloc` sarebbe appropriata) lo scopo è creare una lista da usare per collaudare gli altri sottoprogrammi.

Argomenti: lista concatenata semplice. `fill`.

- ◇ passo base
- ◇ passo induttivo
- ◇ condizione di termine della ricorsione
- ◇ stack delle chiamate
- ◇ visibilità variabili locali
- ◇ variabili `static`

Esercizio 31.1: Fattoriale – versione ricorsiva

Scrivere un sottoprogramma ricorsivo che calcola e restituisce il fattoriale di un intero.

Argomenti: sottoprogramma. ricorsione.

Esercizio 31.2: Lunghezza di una lista – versione ricorsiva

Scrivere un sottoprogramma ricorsivo che ricevuta in ingresso una lista, calcola e restituisce la sua lunghezza.

Argomenti: sottoprogramma. ricorsione. `length`

Esercizio 31.3: Lunghezza di una stringa – versione ricorsiva

Scrivere un sottoprogramma ricorsivo che ricevuta in ingresso una stringa, calcola e restituisce la sua lunghezza.

Argomenti: sottoprogramma. ricorsione. `stringhe`.

Esercizio 31.4: Carattere nella stringa – versione ricorsiva

Scrivere un sottoprogramma ricorsivo che ricevuta in ingresso una stringa ed un carattere, calcola e restituisce il numero di volte che il carattere compare nella stringa.

Argomenti: sottoprogramma. ricorsione. `stringhe`. indirizzo di un elemento dell'array.

Esercizio 31.5: Variabili `static`

Programma che chiama un sottoprogramma con una variabile dichiarata `static`, che viene allocata non sullo stack e il cui valore è persistente rispetto a chiamate successive.

Esercizio 31.6: Scompatta lista

Scrivere un sottoprogramma `extract` che riceve in ingresso una lista per la gestione dei numeri interi, e un intero `start` che vale senz'altro 0 o 1 (non è necessario gestire il caso in cui non sia così). La lista *codifica* un'informazione binaria: il valore del primo elemento indica quante volte consecutive compare il bit `start`, il secondo elemento indica quante volte compare il complemento di `start`, il terzo elemento quante volte compare il bit `start` e così fino alla fine. Il sottoprogramma scompatta tale informazione e restituisce **una nuova lista** i cui elementi contengono ciascuno 0 o 1.

Non è necessario definire due tipi diversi di dato, poichè il contenuto dell'elemento della lista è comunque sempre un intero.

Per esempio, se la lista in ingresso è quella di seguito riportata e `start` è 1:

$$3 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 1 \rightarrow |$$

il sottoprogramma restituisce la lista seguente

$$1 \rightarrow 1 \rightarrow 1 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 1 \rightarrow 0 \rightarrow |$$

Argomenti: lista concatenata semplice.

Tratto dal tema d'esame del 09/09/2019

Esercizio 31.7: Paint ... semplificato – Proposto e poi risolto

Si scriva un sottoprogramma `paint` che riceve in ingresso un array bidimensionale (con `NC` numero di colonne) e qualsiasi altro parametro ritenuto necessario, oltre a due interi `x` e `y` che sono le coordinate di un elemento dell'array. L'array contiene solo 0 e 1. Il sottoprogramma, modifica il valore dell'elemento di coordinate `x` e `y` mettendolo a 0, se vale 1, e propaga la trasformazione verso i 4 elementi adiacenti posti nelle posizioni a destra, sinistra, alto e in basso.

Argomenti: sottoprogramma. ricorsione.

Campi e alberi

Alberi

Si consideri il gioco "Alberi", costituito da una griglia quadrata, in cui in ogni elemento può esserci un albero o meno. Le dimensioni della griglia variano da gioco a gioco, ma sono di al più 12 elementi per lato. La griglia è occupata da aree di colore diverso, cui appartengono uno o più elementi, e le regole per la corretta presenza degli alberi sono le seguenti: i) ci devono essere 2 alberi per riga, ii) ci devono essere 2 alberi per colonna, iii) ci devono essere 2 alberi per colore, iv) negli elementi adiacenti ad un albero non ci devono essere alberi. Gli schemi più piccoli (con dimensione minore o uguale a 6) contengono 1 solo albero per riga, colonna e colore.

Si realizzi un programma che acquisisce in ingresso la specifica della disposizione delle aree e dei colori (nel formato indicato di seguito) e visualizza 1 se lo schema ricevuto in ingresso è corretto, 0 altrimenti, seguito dal carattere 'a capo'. Si organizzi il codice in sottoprogrammi.

Si suggerisce di sviluppare, tra gli altri, i seguenti sottoprogrammi:

`checkGriglia`

restituisce 1 se il numero di alberi e la loro disposizione è corretta

`checkRiga`

restituisce 1 se il numero di alberi nella riga della griglia è quello richiesto

`checkColonna`

restituisce 1 se il numero di alberi nella colonna della griglia è quello richiesto

`checkColore`

restituisce 1 se il numero di alberi in un'area di colore della griglia è quello richiesto

`checkDistanza`

restituisce 1 se quando si incontra un albero nella griglia esso non ha alberi negli elementi adiacenti.

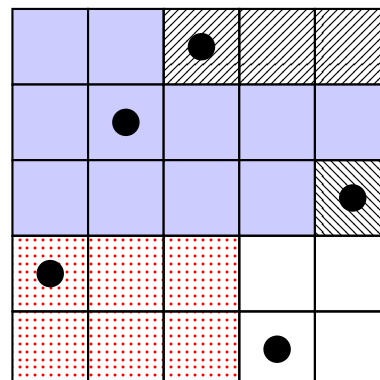
Ingresso/Uscita:

input: un insieme di numeri interi e caratteri

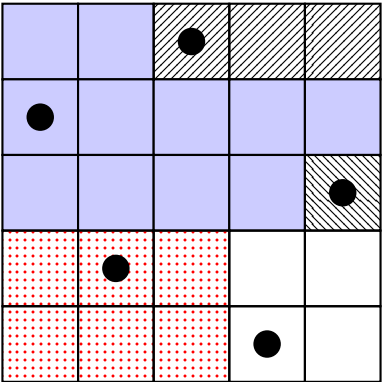
output: un intero

Alcuni casi di test per il collaudo:

```
5 % dimensione del campo di gioco
AABBB % un carattere per ogni colore
AAAAA
AAAAC
DDDEE
DDDEE
0 2 % coordinate degli alberi
1 1
2 4
3 0
4 3
0
```




```
5 % dimensione del campo di gioco
AABBB % un carattere per ogni colore
AAAAA
AAAAC
DDDEE
DDDEE
0 2 % coordinate degli alberi
1 0
2 4
3 1
4 3
1
```



Esercitazione 10 ♦ riepilogo

26-11-2019 (CB)

Esercizio 33.1: Fibonacci

La successione di Fibonacci è così definita: $F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)$ per ogni $n > 1$. Scrivere due versioni di un sottoprogramma, una che sfrutta la ricorsione, mentre l'altra no, che riceve in ingresso un intero n e restituisce l' n -esimo valore nella successione di Fibonacci.

Scrivere quindi un programma che chiede all'utente un intero $n \geq 0$, richiedendo l'inserimento finché il vincolo viene rispettato, e stampa l' n -esimo valore nella sequenza di Fibonacci calcolato utilizzando una delle due versioni definite precedentemente.

Argomenti: ricorsione. sottoprogrammi.

Esercizio 33.2: Conta iniziali vocaboli di un file

Scrivere un programma che chiede all'utente il nome di un file (al più 40 caratteri) di testo e conta e visualizza il numero di parole che iniziano con ciascuna lettera dell'alfabeto. I caratteri contenuti nel file sono solo minuscoli. Le parole sono separate esclusivamente da spazi e sono al più di 32 caratteri.

Visualizzare il messaggio:

- ♦ parole che iniziano con a: 10
- ♦ parole che iniziano con b: 3
- ♦ parole che iniziano con e: 14

Non visualizzare alcun messaggio per le lettere che non hanno parole nel testo.

Argomenti: file. stringhe.

Esercizio 33.3: Mergesort

Il mergesort è un algoritmo di ordinamento che, dato un array, ripete i seguenti passi:

- ♦ viene diviso in due parti l'array ricevuto in ingresso;
- ♦ ogni parte viene ordinata applicando ricorsivamente l'algoritmo;
- ♦ le due parti vengono fuse insieme estraendo ripetutamente il minimo da esse, in modo da ottenere una fusione ordinata.

Scrivere un sottoprogramma che riceve in ingresso un array di interi da ordinare e tutti i parametri aggiuntivi che si ritengono necessari ed esegue un merge sort su di esso.

Scrivere quindi un programma che riceve in ingresso 10 numeri interi e li stampa ordinati in modo crescente.

Argomenti: ricorsione. array. sottoprogrammi. algoritmi di ordinamento.

Esercizio 33.4: Date di nascita

Scrivere un programma che legge dal file binario `date.bin` le date di nascita di 100 persone, memorizzate come tre interi che rappresentano il giorno, il mese e l'anno di nascita. Il programma chiede all'utente una data e conta e visualizza, in base ai dati contenuti nel file `date.bin`:

- ◇ il numero di persone nate in quel giorno;
- ◇ il numero di persone che festeggiano insieme il compleanno in tale data.

Il programma visualizza i due valori interi, separati da uno spazio e seguiti da un carattere a-capo. Definire un tipo opportuno per rappresentare le date.

Argomenti: file binari. `typedef`.

Esercizio 33.5: Paint

Si scriva un sottoprogramma `paint` che riceve in ingresso un array bidimensionale (con `NC` numero di colonne) e qualsiasi altro parametro ritenuto necessario, oltre a due interi `x` e `y` che sono le coordinate di un elemento dell'array. L'array contiene solo 0 e 1. Il sottoprogramma, se l'elemento di coordinate `x` e `y` vale 1, modifica il suo valore mettendolo a 0, e propaga la trasformazione verso i 4 elementi adiacenti posti nelle posizioni a destra, sinistra, alto e in basso.

Argomenti: ricorsione. array bidimensionali.

Esercitazione 11 ♦ riepilogo

28-11-2019 (CB)

Esercizio 34.1: Array to list

Scrivere un sottoprogramma che riceve in ingresso un array di valori interi e qualsiasi altro parametro ritenuto strettamente necessario e restituisce una lista contenente tutti e soli i valori dell'array che sono minori o uguali alla media dei valori contenuti nell'array stesso.

Argomenti: array. liste. calcolo media.

Esercizio 34.2: Convoluzione 1D

Si vuole riprodurre l'operazione di convoluzione discreta a una dimensione. In pratica si tratta di applicare un array monodimensionale chiamato filtro o kernel a un altro array monodimensionale. Il risultato della convoluzione è, per ogni cella i , l'applicazione del filtro a partire dalla cella i dell'array di partenza. Per applicazione si intende la somma della moltiplicazione elemento per elemento tra gli elementi dell'array e gli elementi del filtro.

Un esempio:

```
Array: 1 2 3 4 5 6  
Filtro: 1 0 1  
Risultato: 4 6 8 10
```

Infatti, $4 = 1 * 1 + 2 * 0 + 3 * 1$, $6 = 2 * 1 + 3 * 0 + 4 * 1$, $8 = 3 * 1 + 4 * 0 + 5 * 1$, $10 = 4 * 1 + 5 * 0 + 6 * 1$. Da notare che l'applicazione del filtro viene fatta solamente se è possibile applicare l'intero filtro, e non solo una sua parte. Se non è più possibile applicare l'intero filtro, la convoluzione termina.

Scrivere un programma che chiede in ingresso il nome di un file (di lunghezza massima 50 caratteri). Il file contiene, nella prima riga, due interi, rispettivamente la dimensione dell'array n e la dimensione del filtro k , con sicuramente $n \geq k$. La seconda riga contiene n interi, che rappresentano i valori dell'array. La terza ed ultima riga contiene k interi, che rappresentano i valori del filtro. Il programma legge il file avente il formato appena proposto e applica il filtro all'array, salva il risultato in un nuovo array e ne stampa il contenuto.

Argomenti: allocazione dinamica. file.

Esercizio 34.3: Accesso utente banca

Si vuole gestire l'accesso ai dati relativi ai conti correnti dei clienti di una banca. La banca in questione è minimalista e per ogni utente memorizza solamente il nome utente e la password per l'accesso al conto e il saldo attuale. Si definisca un nuovo tipo di dato per memorizzare queste informazioni, sapendo che nome utente e password sono lunghi al più 50 caratteri.

Si scriva quindi un sottoprogramma che dati in ingresso l'array degli utenti, un nome utente, una password e tutti i parametri ritenuti necessari, cerchi l'utente nell'array che corrisponda alle credenziali (nome utente e password) ricevute in ingresso e, se esiste, ne restituisca la posizione all'interno dell'array, altrimenti restituisca -1.

Argomenti: stringhe. typedef.

Esercizio 34.4: Vicinanza media di una matrice

Scrivere un sottoprogramma che data in ingresso una matrice di numeri reali e tutti i parametri strettamente necessari, trasmette la media dei valori della matrice e la posizione (in termini di indici di riga e colonna) della cella il cui valore è il più vicino alla media. Come precisazione, per *vicinanza* tra due numeri si intende il valore assoluto della loro differenza. In caso ci fossero più valori con minima vicinanza alla media, selezionarne uno a piacere.

Scrivere quindi un programma che dati in ingresso due interi, rispettivamente il numero di righe e il numero di colonne (che si assumono validi e inseriti correttamente), e una matrice di reali di quelle dimensioni, ne calcola e stampa la media dei valori, il valore più vicino alla media e la sua posizione. Si assuma che la matrice abbia al massimo 20 righe e 20 colonne. E' consentito scrivere ulteriori sottoprogrammi di supporto per operazioni ripetute, se ritenuto necessario.

Argomenti: array bidimensionale. passaggio parametri per indirizzo.

Esercizio 34.5: Separa stringa (proposto)

Scrivere un sottoprogramma che, data in ingresso una stringa, separa la stringa in concomitanza degli spazi e restituisce una lista contenente i pezzi della stringa separata, nell'ordine in cui sono presenti nella stringa originale. Esempio:

"Questa stringa deve essere separata ! "

"Questa" → "stringa" → "deve" → "essere" → "separata" → "!" → |

Da notare che ci possono essere più spazi tra una parola e l'altra, in tal caso devono essere considerati come un unico spazio.

Si dispone del seguente tipo di lista e si considerino già disponibili (e quindi non da sviluppare) i sottoprogrammi seguenti:

```
1 typedef struct slist_s{
2
3     char * str;
4     struct slist_s * next;
5
6 } slist_t;
7
8
9 /*inserisce in testa alla lista*/
10 slist_t * push(slist_t *, char *);
11
12 /*inserisce in coda alla lista*/
13 slist_t * append(slist_t *, char *);
```

Argomenti: liste. stringhe. allocazione dinamica.

Cancella elementi frequenti dalla lista

Scrivere un sottoprogramma `delfromlist` che riceva in ingresso una lista per la gestione dei numeri interi ed un intero x , elimini dalla lista tutti quegli elementi che compaiono almeno x volte, e restituisca la lista. Se per esempio il sottoprogramma riceve in ingresso la lista di seguito riportata ed il valore 3:

$3 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow |$

il sottoprogramma restituisce la lista seguente:

$1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 5 \rightarrow 4 \rightarrow |$

Definire un tipo di dato opportuno per la lista.

Si considerino già disponibili (e quindi non da sviluppare) i sottoprogrammi seguenti:

```
1 /*inserisce in testa alla lista*/
2 ilist_t * push(ilist_t *, int);
3
4 /*inserisce in coda alla lista*/
5 ilist_t * append(ilist_t *, int);
6
7 /*elimina dalla lista il primo elemento*/
8 ilist_t * pop(ilist_t *);
9
10 /*elimina dalla lista tutti gli elementi con il valore indicato*/
11 ilist_t * delete(ilist_t *, int);
12
13 /*restituisce il riferimento all'elemento nella lista che ha il valore indicato,
14    se esiste*/
14 ilist_t * exists(ilist_t *, int);
15
16 /*restituisce il numero di elementi nella lista*/
17 int length(ilist_t *);
18
19 /*elimina la lista*/
20 ilist_t * emptylist(ilist_t *);
```

Argomenti: liste. sottoprogrammi. typedef. eliminazione da lista.

Tratto dal tema d'esame del 17/06/2019

Compatta lista

Scrivere un sottoprogramma *compactlist* che ricevuta in ingresso una lista per la gestione dei numeri interi ne crei una nuova in cui per ogni valore presente nella lista in ingresso viene memorizzato anche il numero di volte in cui esso compare. La lista creata deve avere gli elementi ordinati in senso crescente rispetto al valore (e non al numero di occorrenze). La lista così creata viene restituita al chiamante.

Per esempio, se la lista in ingresso è la seguente:

$3 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow -5 \rightarrow 3 \rightarrow -5 \rightarrow 4 \rightarrow |$

il sottoprogramma restituisce la lista seguente:

$-5, 2 \rightarrow 1, 1 \rightarrow 2, 1 \rightarrow 3, 4 \rightarrow 4, 2 \rightarrow |$

Definire i due tipi di lista necessari per la realizzazione del sottoprogramma

Si considerino già disponibili (e quindi non da sviluppare) i sottoprogrammi seguenti, validi per qualsiasi tipo di lista che gestisca almeno un campo intero:

```
1 /*inserisce in testa alla lista*/
2 listtype * push(listtype *, int);
3
4 /*inserisce in coda alla lista*/
5 listtype * append(listtype *, int);
6
7 /*inserisce ordinatamente in lista, in base al valore crescente*/
8 listtype * increasing(listtype *, int);
9
10 /*inserisce ordinatamente in lista, in base al valore decrescente*/
11 listtype * decreasing(listtype *, int);
12
13 /*elimina dalla lista il primo elemento*/
14 listtype * pop(listtype *);
15
16 /*elimina dalla lista tutti gli elementi con il valore indicato*/
17 listtype * delete(listtype *, int);
18
19 /*restituisce il riferimento all'elemento nella lista che ha il valore indicato,
    se esiste, NULL altrimenti*/
20 listtype * exists(list_t *, int);
21
22 /*restituisce il numero di elementi nella lista*/
23 int length(list_t *);
24
25 /*elimina la lista*/
26 listtype * emptylist(list_t *);
```

Argomenti: liste. sottoprogrammi. typedef. creazione lista.

Tratto dal tema d'esame del 15/07/2019

Somma e prodotto di polinomi

Si consideri il tipo di dato riportato di seguito, per rappresentare i termini di polinomi.

```
1 typedef struct _term {
2     int c; /* coefficiente */
3     int p; /* potenza */
4     struct _term * next;
5 } t_term;
```

Scrivere i sottoprogrammi *sum* e *prod* che dati due polinomi in ingresso calcolino e restituiscano rispettivamente il polinomio somma e il polinomio prodotto. I polinomi ricevuti in ingresso hanno i termini ordinati per potenze decrescenti, e così devono essere anche i polinomi creati.

Si preveda di disporre dei sottoprogrammi *lunghezza*, *conta*, *instesta*, *inscoda*, *insord*, *esiste*, *del*, *svuotalista*, i cui prototipi (con anche i nomi dei parametri) e funzionalità sono riportati di seguito.

I sottoprogrammi qua riportati si riferiscono - per semplicità - al caso di lista per la gestione di dati interi: si immagini di disporre del sottoprogramma equivalente per la gestione di tipi di dati anche diversi, in base alle esigenze.

```
1 /* restituisce il numero di elementi presenti nella lista h */
2 int lunghezza(t_elem * h);
3
4 /* restituisce il numero di elementi presenti nella lista h con campo
   informazione val */
5 int conta(t_elem * h, int val);
6
7 /* crea un nuovo elemento con campo informazione val e lo inserisce in testa
   alla lista h, restituendo la testa */
8 t_elem * instesta(t_elem * h, int val);
9
10 /* crea un nuovo elemento di campo informazione val e lo inserisce in coda alla
    lista h, restituendo la testa */
11 t_elem * inscoda(t_elem * h, int val);
12
13 /* crea un nuovo elemento di campo informazione val e lo inserisce in nella
    lista h in ordine rispetto al campo intero, restituendo la testa */
14 t_elem * insord(t_elem * h, int val);
15
16 /* cerca nella lista h un termine con campo informazione val e se esiste
    restituisce il puntatore a tale termine altrimenti restituisce NULL */
17 t_term * esiste(t_elem * h, int val);
18
19 /* elimina dalla lista h un termine con campo informazione val e restituisce la
    testa della lista */
20 t_term * del(t_elem * h, int val);
21
22 /* svuota la lista h */
23 void svuotalista(t_elem * h);
```

Argomenti: liste. sottoprogrammi. polinomi.

Tratto dal tema d'esame del 03/07/2017

Lezione 17 ◇ parametri da riga di comando `argv`, `argc` e variabili globali

03-12-2019

- ◇ acquisizione dati da riga di comando
 - `argv`
 - `argc`
- ◇ variabili globali

Esercizio 36.1: *genera*

Scrivere un programma che acquisisce da riga di comando una stringa e chiama il sottoprogramma `genera`.

Tratto dal tema d'esame del 28/01/2019

Argomenti: argomenti da riga di comando.

Esercizio 36.2: *conta primi*

Scrivere una variante del programma sopra richiesto, che acquisisca da riga di comando il nome del file sia il nome del file iniziale, sia quello del file in cui salvare il risultato dell'elaborazione. Un'esecuzione, da riga di comando, di esempio è:

```
contaprimi ./dati.txt ./risultati.txt
```

Limitarsi alla parte di dichiarazione delle variabili e all'acquisizione dei dati per poter poi procedere nell'algoritmo.

Tratto dal tema d'esame del 17/06/2019

Argomenti: argomenti da riga di comando.

Esercitazione 12 ◇ parametri da riga di comando

05-12-2019

Esercizio 37.1: Crop

Scrivere un sottoprogramma `crop` che riceva in ingresso una stringa `frase` ed un carattere `ch` restituisce una nuova stringa che contiene i caratteri compresi tra la prima e la seconda occorrenza del carattere `ch`, incluso il carattere `ch`. Per esempio, se il sottoprogramma riceve in ingresso `informatica` e `i`, il sottoprogramma restituisce la stringa `informati`. Nel caso in cui la stringa `frase` non contenga due occorrenze del carattere `ch`, restituisce `NULL`. Scrivere un programma che acquisisce da riga di comando una stringa ed un carattere e chiama il sottoprogramma `crop` prima descritto e visualizza il risultato dell'elaborazione, quindi termina. Un paio di esecuzioni, da riga di comando, di esempio sono:

```
./ritagliastringa informatica a
atica

./ritagliastringa collaudo x
(null)
```

Argomenti: parametri da linea di comando. stringhe. allocazione dinamica.

Esercizio 37.2: Unisci liste ordinate

Scrivere un sottoprogramma che riceva in ingresso due liste per la gestione di numeri interi, ciascuna delle quali ordinata in ordine crescente, crei una nuova lista contenente l'unione ordinata dei valori presenti nelle due liste, priva di ripetizioni e la restituisca al chiamante.

Viene dato il seguente tipo di dato per trattare una lista di interi e la relativa funzione `append` che appende in coda alla lista:

```
1 typedef struct ilist_s {
2
3     int val;
4     struct ilist_s * next;
5
6 } ilist_t;
7
8
9 ilist_t * append(ilist_t *, int);
```

Tratto dal tema d'esame del 18/02/2016

Argomenti: liste. unione. inserimento ordinato. elimina duplicati.

Esercizio 37.3: Genera numeri binari

Scrivere un sottoprogramma che visualizza tutti i numeri binari rappresentati da una stringa costituita dai valori `0`, `1` e `x`, dove le `x` possono assumere valore sia `0` sia `1`. Quindi, se il sottoprogramma riceve in ingresso la stringa `1x0` visualizza `100` e `110` (l'ordine non è importante). Scrivere un sottoprogramma `genera` che riceve in ingresso una stringa costituita esclusivamente di `0`, `1` e `x` (è senz'altro così) e visualizza tutti i numeri binari rappresentabili.

Scrivere anche una versione ricorsiva del sottoprogramma `genera` (è possibile aggiungere un eventuale parametro).

Tratto dal tema d'esame del 28/01/2019

Argomenti: numeri binari. ricorsione. stringhe.

Esercizio 37.4: Trova somma

Scrivere un sottoprogramma che riceve in ingresso un array bidimensionale di interi `mat`, un intero `val` e qualsiasi parametro ritenuto strettamente necessario e trasmette al chiamante gli indici di riga e colonna che identificano la posizione del primo elemento (scandendo l'array per righe) che, sommato a tutti i suoi precedenti, dia come risultato un valore $> val$. Nel caso in cui tal elemento non esista, si trasmettono i valori -1, -1. Esiste una direttiva `#define NCOL 10`.

Tratto dal tema d'esame del 18/02/2019

Argomenti: array bidimensionali.

Tag Cloud o Word Cloud

Si vuole realizzare un programma che partendo da un file di testo crei una cosiddetta *Word Cloud* da visualizzarsi mediante una pagina web. Per raggiungere lo scopo, sono dati i seguenti file di testo (disponibili su piazza.com):

- ◇ `stopWords.it.txt` e `stopWords.txt` che contiene le parole che non vanno considerate in quanto non rilevanti dal punto di vista della semantica (ad esempio congiunzioni, articoli, ...) sia per la lingua italiana che per quella inglese,
- ◇ `d3.layout.cloud.js`, `d3.min.js` e `word-cloud.js` file javascript utilizzati dalla pagina web per realizzare la Word Cloud, e
- ◇ `wordcloud.htm` pagina html utilizzata per visualizzare la Word Cloud, che include la seguente direttiva:

```
<script type="text/javascript" src="words.js"></script>
```

Il file `words.js` è ciò che il programma deve creare per ottenere il risultato desiderato.

Il programma riceve in ingresso un file ASCII contenente il testo da analizzare e crea il file `words.js` risultato dell'analisi. Il formato di tale file è il seguente:

```
var tags = [  
{"key": "programma", "value" : 26},  
{"key": "C", "value" : 27},  
{"key": "variabile", "value" : 25},  
{"key": "ciclo", "value" : 25},  
{"key": "costrutto", "value" : 27},  
{"key": "tipo", "value" : 25},  
{"key": "sottoprogramma", "value" : 24},  
{"key": "parametro", "value" : 26},  
{"key": "visualizza", "value" : 26},  
{"key": "argc", "value" : 25},  
...  
{"key": "lista", "value" : 6}  
];
```

A parte la prima e l'ultima riga, ogni elemento specifica un vocabolo quante volte compare nel testo. Nel realizzare la soluzione, si tengano presente i seguenti aspetti:

- ◇ non mettere nel file finale i vocaboli che compaiono meno di 6 volte;



- ◊ tutti i vocaboli hanno al più 30 caratteri (simbolo `LEN`);
- ◊ i vocaboli contenuti nei file `stopWords.it.txt` e `stopWords.txt` sono tutti minuscoli;
- ◊ i vocaboli nel file da analizzare possono avere maiuscole, si tratta di un testo reale;
- ◊ nel file da analizzare ci possono essere segni di interpunzione (virgola, apostrofo, ...) e devono essere opportunamente gestiti;
- ◊ il file da analizzare potrebbe contenere doppi spazi, trovare i vocaboli

A titolo di esempio di ciò che dovrebbe realizzare il programma si consideri il testo iniziale qui riportato

[...] Ai tempi in cui accaddero i fatti che prendiamo a raccontare, quel borgo, già considerabile, era anche un castello, e aveva perciò l'onore dalloggiare un comandante, e il vantaggio di possedere una stabile guarnigione di soldati spagnoli, che insegnavan la modestia alle fanciulle e alle donne del paese, accarezzavan di tempo in tempo le spalle a qualche marito, a qualche padre; e, sul finir dellestate, non mancavan mai di spandersi nelle vigne, per diradar l'uve, e alleggerire a' contadini le fatiche della vendemmia. Dall'una all'altra di quelle terre [...]

i vocaboli che il programma dovrebbe considerare, prima di aver ignorato quelli presenti nel file `stopWords.it.txt` sono:

[...] ai tempi in cui accaddero i fatti che prendiamo a raccontare quel borgo già considerabile era anche un castello e aveva perciò onore alloggiare un comandante e il vantaggio di possedere una stabile guarnigione di soldati spagnoli che insegnavan la modestia alle fanciulle e alle donne del paese accarezzavan di tempo in tempo le spalle a qualche marito a qualche padre e sul finir estate non mancavan mai di spandersi nelle vigne per diradar uve e alleggerire contadini le fatiche della vendemmia una altra di quelle terre [...]

utilizzando il file `stopWords.it.txt`, il programma scarnerà i vocaboli che compaiono in tale file, arrivando ai seguenti vocaboli (non è importante che non ci siano vocaboli ripetuti)

tempi fatti prendiamo raccontare borgo considerabile castello onore alloggiare comandante
vantaggio possedere stabile guarnigione soldati spagnoli insegnavan modestia fanciulle donne
paese accarezzavan tempo tempo spalle marito padre finir estate mancavan spandersi vigne
diradar uve alleggerire contadini fatiche vendemmia terre

Si utilizzi il seguente tipo di dato:

```
typedef struct wordlist_s {
    char word[LEN+1];
    int times;
    struct wordlist_s * next;
} wordlist_t;
```

Si considerino già a disposizione i seguenti sottoprogrammi (codice disponibile su piazza.com):

```
wordlist_t * increasing(wordlist_t * h, char * w);
```

inserisce un nuovo elemento in ordine alfabetico crescente rispetto al campo `word` della struttura;

```
wordlist_t * deleteptr(wordlist_t * h, wordlist_t * e)
```

elimina un elemento dalla lista;

```
❖ wordlist_t * find(wordlist_t * h , char * w);
```

trova e restituisce della lista con un certo valore nel campo `word` della struttura, se esiste, altrimenti restituisce `NULL`.

Si utilizzino i sottoprogrammi delle librerie `string.h`, `stdlib.h`, `ctype.h` e simili quando utili, senza sviluppare ex-novo cose che già esistono.

Esercitazione 13 ◇ temi d'esame

12-12-2019 (CB)

Esercizio 41.1: ISBN-10

Scrivere un sottoprogramma `checkISBN10` che riceve in ingresso una stringa che rappresenta un codice ISBN-10 (International Standard Book Number) di 10 cifre numeriche separate da -, utilizzato per identificare univocamente un volume prima del 2007 (dal 2007 in poi il numero è di 13 cifre). Il sottoprogramma restituisce 1 se il codice ISBN è valido, 0 altrimenti. Un codice ISBN-10 è valido se la somma delle somme è un multiplo di 11. La somma delle somme si calcola addizionando ogni cifra del codice alla somma delle precedenti cifre.

Esempi:

Ingresso: 0-306-40615-2 Cifre: 0 3 0 6 4 0 6 1 5 2 Somma delle cifre: 0 3 3 9 13 13 19 20 25 27 Somma delle somme: 132 Uscita: 1
Ingresso: 0-07-881809-4 Cifre: 0 0 7 8 8 1 8 0 9 4 Somma delle cifre: 0 0 7 15 23 24 32 32 41 45 Somma delle somme: 219 Uscita: 0

Scrivere il programma che acquisisce da riga di comando la stringa dell'ISBN-10 e - utilizzando il sottoprogramma `checkISBN10` - visualizza 1 se il codice è corretto, 0 altrimenti.

Tratto dal tema d'esame del 26/01/2018

Argomenti: stringhe. sottoprogrammi. argomenti da riga di comando.

Esercizio 41.2: Elimina le sottosequenze

Definire un tipo di dato opportuno `clist_t` per realizzare una lista dinamica che gestisce caratteri (ogni elemento un carattere) e serve per gestire sequenze di caratteri. Si definisce sottosequenza una sequenza di caratteri compresa tra una parentesi tonda iniziale (e una finale). Le sottosequenze possono anche essere vuote (parentesi aperta e poi chiusa senza altri caratteri intermedi). Scrivere un sottoprogramma riceve in ingresso una lista che costituisce una sequenza e la restituisce dopo aver sostituito le sottosequenze con il carattere #.

La sequenza dei caratteri in ingresso è ben formata, ossia:

- ◇ per ogni parentesi tonda che si apre, c'è una parentesi tonda che si chiude
- ◇ non ci sono intersezioni tra coppie di parentesi.

Per esempio, `ab(acg)be()a(xx)f` è una sequenza ben formata, `ab(a(c)g)b` e `aba(c)g)b` non lo sono. Se il sottoprogramma riceve in ingresso la sequenza `ab(acg)be()a(xx)f(a)`, restituisce la sequenza `ab(#)be(#)a(#)f(#)`.

Si considerino già disponibili e non da sviluppare i sottoprogrammi seguenti:

```
1 /* inserisce in testa alla lista */
2 clist_t * push(clist_t *, char);
3
4 /* inserisce in coda alla lista */
5 clist_t * append( clist_t * , char );
6
7 /* elimina dalla lista il primo elemento */
8 clist_t * pop(clist_t *);
```

```

9
10 /* elimina dalla lista tutti gli elementi con il valore indicato */
11 clist_t * delete(clist_t *, char);
12
13 /* restituisce il riferimento all'elemento nella lista che ha il valore indicato
14    , se esiste */
15 clist_t * exists(clist_t * , char );
16
17 /* restituisce il numero di elementi nella lista */
18 int length(clist_t *);

```

Tratto dal tema d'esame del 19/02/2019

Argomenti: liste.

Esercizio 41.3: Conta le vette

Scrivere un sottoprogramma ricorsivo che ricevuto in ingresso un array di interi (e qualsiasi altro parametro ritenuto strettamente necessario) restituisce il numero di *vette* in esso presenti. Si definisce *vetta* un valore presente nell'array maggiore di tutti i valori successivi presenti nell'array. L'ultimo valore dell'array, per definizione, non è una *vetta*. Scrivere quindi un programma che acquisisce in ingresso un array di 10 interi e stampa il rispettivo numero di *vette*.

Tratto dal tema d'esame del 18/02/2016

Argomenti: ricorsione. sottoprogrammi. array.

Esercizio 41.4: Da intero a lista

Scrivere un sottoprogramma `int2list` che ricevuto in ingresso un numero intero restituisce una lista in cui ogni cifra del numero in ingresso compare tante volte quanto il suo valore. Nel caso in cui il valore ricevuto in ingresso sia negativo, il sottoprogramma restituisce la lista creata a partire dalla cifre in ordine opposto.

Se per esempio il sottoprogramma riceve in ingresso l'intero 3204, il sottoprogramma restituisce la lista seguente: $3 \rightarrow 3 \rightarrow 3 \rightarrow 2 \rightarrow 2 \rightarrow 4 \rightarrow 4 \rightarrow 4 \rightarrow 4 \rightarrow |$

Se per esempio il sottoprogramma riceve in ingresso l'intero -3204, il sottoprogramma restituisce la lista seguente: $4 \rightarrow 4 \rightarrow 4 \rightarrow 4 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 3 \rightarrow |$

Definire inoltre un tipo di dato opportuno per la lista.

Tratto dal tema d'esame del 18/02/2019

Argomenti: liste. allocazione dinamica.

Esercizio 41.5: Cifra più frequente

Scrivere un sottoprogramma che riceve in ingresso un riferimento ad un file (già aperto) e legge un valore intero (se c'è ...) e restituisce la cifra del valore letto che in esso compare più di frequente. Nel caso in cui non ci sia un valore, restituisce -1. Nel caso ci siano più cifre che compaiono lo stesso numero di volte, restituisce quella più alta. Se per esempio legge il valore 217319 restituisce 1, se legge il valore 1002932 restituisce 2.

Tratto dal tema d'esame del 28/01/2019

Argomenti: file. sottoprogrammi.

I grattacieli

I **grattacieli** è un gioco enigmistico con una griglia NxN che rappresenta una città in cui ogni casella è occupata da un edificio, la cui altezza varia da 1 a N piani.

In ogni riga o colonna non ci sono edifici di pari altezza. I numeri esterni indicano quanti edifici si vedono da quel punto (i più alti nascondono i più bassi).

Chi risolve il gioco riempie tutte le caselle della griglia cercando di rispettare tutti i vincoli.

Si realizza un **programma** che dati i numeri esterni e una griglia **completata** verifica se la soluzione è corretta, ossia:

- ◇ sono rispettati tutti i vincoli di visibilità dei grattacieli
- ◇ in ogni riga e ogni colonna, non ci sono grattacieli di ugual altezza

Specifiche

Il programma acquisisce da tastiera in ingresso il nome di un file binario (al più 13 caratteri, inclusi percorso ed estensione) che contiene:

- ◇ la dimensione della griglia (dimensione massima 10)
- ◇ i dati della griglia,
- ◇ i numeri esterni, in senso orario a partire da quello in alto a sinistra.

Il programma al termine dell'elaborazione visualizza 1 se la soluzione è valida, 0 altrimenti, seguito da UN carattere a capo ('\n').

Ipotesi di lavoro

- ◇ il file contiene tutti e soli i dati necessari (valori interi di tipo int),
- ◇ le altezze inserite e i numeri esterni sono senz'altro tutti compresi tra 1 e N

Vincoli

- ◇ non è possibile utilizzare variabili globali
- ◇ è possibile utilizzare i sottoprogrammi delle librerie standard (stdio, string, ...)
- ◇ ogni membro del gruppo deve scrivere **autonomamente** almeno un sottoprogramma

Aspetti di rilievo nella valutazione della qualità della soluzione proposta (oltre alla correttezza)

- ◇ utilizzo appropriato della memoria
- ◇ prestazioni

Compilazione ed esecuzione del codice

Sottomettere il **sorgente** (un unico file), che viene compilato con le opzioni (standard ANSI C 89):

```
-Wall -O2 -std=c89 -pedantic
```

Fate quindi attenzione a non utilizzare commenti // ma solo /* */.

Il risultato dell'esecuzione è confrontato automaticamente con il risultato atteso, carattere a carattere.

Ogni differenza è considerata un errore, quindi è fondamentale NON aggiungere stampe a video di servizio.

Esempio di contenuto del file di partenza (il file veramente usato è binario)

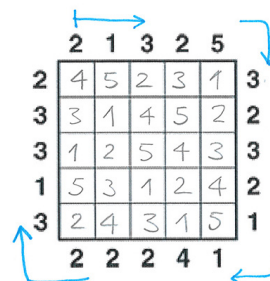
```
5 4 5 2 3 1 3 1 4 5 2 1 2 5 4 3 5 3 1 2 4 2 4 3 1 5 2 1 3 2 5 3 2 3 2 1 1 4 2 2 2 3 1 3 3 2
```

Sottomissione della soluzione

È stato creato un account per la vostra squadra (codice utente del referente) per sottomettere la soluzione al seguente indirizzo:

<http://poseidon.ws.dei.polimi.it/contests/>

L'autenticazione è tramite AUnica del Politecnico











Casi di test

Sono disponibili e verranno utilizzati 5 casi di test: per ogni caso di test è disponibile il file di ingresso e l'uscita attesa.

Prima di sottoporre la soluzione, verificate che **per ogni caso di test** il vostro programma visualizzi la soluzione corretta, e con il formato richiesto (un intero seguito da un a-capo).

Risultato dell'analisi

Il verificatore compila ed esegue il vostro programma in relazione ai 5 casi di test, e visualizza uno di questi risultati:

Risultato	Spiegazione
 CORRECT	Il programma è stato compilato ed eseguito con successo e ha fornito la risposta corretta per tutti i casi di test
 NOT CORRECT	Il programma è stato compilato ed eseguito con successo e ha fornito la risposta corretta per tutti i casi di test: la soluzione adottata però non soddisfa le specifiche date (ad esempio non si crea una nuova variabile, ma si visualizza il risultato dell'elaborazione, oppure si introduce una dimensione arbitraria di un array mentre la soluzione presuppone che non si debba memorizzare i dati ...). La soluzione non è considerata valida.
 TIME LIMIT	Il programma è stato compilato con successo ma una volta eseguito non ha completato l'elaborazione entro il tempo limite fissato. Prova a verificare che non ci siano cicli infiniti e prova ad ottimizzare l'algoritmo.
 NO OUTPUT	Il programma è stato compilato con successo ma una volta eseguito non ha prodotto alcun risultato. Questo può essere dovuto alla presenza di cicli infiniti (e ad un certo punto l'esecuzione viene interrotta).
 PRESENTATION ERROR	Il programma è stato compilato ed eseguito con successo ma il risultato prodotto non è identico a quello atteso in termini di presentazione. Spesso non sono stati rispettati i vincoli sul modo di visualizzare il risultato (ad esempio, presenza di eccessivi spazi o mancanza di un "a capo").
 WRONG ANSWER	Il programma è stato compilato ed eseguito con successo ma il risultato prodotto non è quello atteso per almeno uno dei casi di test.
 RUN ERROR	Il programma è stato compilato con successo ma durante l'esecuzione è andato in errore. Questo si verifica di solito per problemi di allocazione di memoria, di accessi a indirizzi sbagliati, di divisioni per zero.
 COMPILATION ERROR	Non è stato possibile compilare il programma. Controlla con attenzione i messaggi generati dal compilatore (facendo distinzione tra avvertimenti Warning ed errori Error). Il compilatore viene chiamato con le seguenti opzioni: <code>-Wall -O2 -std=c89 -pedantic -static</code>

Quesito 4 [7 punti]

- (4 punti) Scrivere un sottoprogramma `minmaxstr` che riceve in ingresso una stringa e trasmette al chiamante due caratteri, il minimo e il massimo nell'ordinamento alfabetico tra quelli contenuti nella stringa. Per esempio, se la stringa in ingresso è "esempio", il minimo ed il massimo sono rispettivamente `e` ed `s`. La stringa contenga tutti e soli caratteri alfabetici minuscoli.
- (2 punti) Rivedere il sottoprogramma `minmaxstr` in modo tale che i caratteri possano essere sia maiuscoli, sia minuscoli, e facendo in modo che il sottoprogramma trasmetta comunque quelli minuscoli. In tal caso, se la stringa in ingresso è "Architetto", il minimo ed il massimo sono rispettivamente `a` ed `t`.
- (1 punto) Scrivere un programma che acquisisce da riga di comando una stringa e chiama il sottoprogramma `minmaxstr`.

Quesito 5 [7 punti]

- (6 punti) Scrivere un sottoprogramma che riceve in ingresso una lista per la gestione dei numeri interi ed un valore intero `dir`. Il sottoprogramma effettua una *rotazione* a sinistra (se `dir` vale 0) o a destra (se `dir` vale 1) del contenuto della lista, restituendola modificata al chiamante. Nella rotazione a sinistra, il primo valore viene posto in fondo alla lista, nello scorrimento a destra, l'ultimo valore della lista viene messo davanti a tutti gli altri. Se per esempio il sottoprogramma riceve in ingresso una lista contenente i valori

$3 \rightarrow 6 \rightarrow 10 \rightarrow -2 \rightarrow 8$

la rotazione a sinistra produce la seguente lista

$6 \rightarrow 10 \rightarrow -2 \rightarrow 8 \rightarrow 3$

la rotazione a destra della lista iniziale produce la seguente lista

$8 \rightarrow 3 \rightarrow 6 \rightarrow 10 \rightarrow -2$

Nel caso in cui la lista è vuota o contiene un solo elemento, il sottoprogramma restituisce la lista ricevuta in ingresso. Si suggerisce di sviluppare due sottoprogrammi separati, uno che esegua la rotazione a sinistra, uno che esegua quella a destra. Non causare *memory leakage*: situazione in cui ci sono nodi non più accessibili senza aver effettivamente rilasciato la memoria.

- (1 punto) Definire un tipo di dato opportuno per gli elementi della lista.

Si considerino già disponibili e non da sviluppare i sottoprogrammi seguenti:

```
/* inserisce in testa alla lista */
elem_t * push(elem_t *, int);
/* inserisce in coda alla lista */
elem_t * append(elem_t *, int);
/* inserisce un elemento nella lista in ordine crescente */
elem_t * insert_inc(elem_t *, int);
/* inserisce un elemento nella lista in ordine decrescente */
elem_t * insert_dec(elem_t *, int);
/* elimina dalla lista il primo elemento */
elem_t * pop(elem_t *);
/* elimina dalla lista tutti gli elementi con il valore indicato */
elem_t * delete(elem_t *, int);
/* restituisce il riferimento all'elemento nella lista che ha il valore indicato, se esiste, NULL altrimenti */
elem_t * exists(elem_t *, int);
/* restituisce il numero di elementi nella lista */
int length(elem_t *);
```

Sapevo già programmare:

- ☐ No, non è vero ☐ in C ☐ in C++/C# ☐ in Python ☐ in Java ☐ in PHP/Javascript ☐ in VB* ☐ in altro linguaggio

Quesito Dominanti in matrice

Quesito Valori completi ed unici

Quesito Minimo e massimo di una stringa

Quesito Lista ruotata

Appello del 17-02-2020

Appello del 03-07-2020

Appello del 17-02-2020

Appello del xx-09-2020

Indice analitico

Allocazione dinamica, 39, 40

Costrutti

- do-while, 11
- for, 14
- if-else if, 9
- if-else, 9
- if, 9
- while, 11

#define, 12, 13

divisore/multiplo, 9

do-while, 11

File

- Accesso con numero di dati non noti a priori, 38

for, 14, 15, 18

free, 39, 46

if, 9

Lista concantenata semplice, 46

Lista concatenata semplice, 47, 48, 50

- append, 46
- deleteptr, 47
- emptylist, 46
- fill, 48
- find, 47
- length, 49
- printlist, 46
- push, 46

malloc, 39, 40, 46

Programmi

- Anagramma, 17

Anno bisestile, 9

Conta occorrenze caratteri, 15

Da decimale a binario, 14

Da intero a stringa, 21

Fattoriale, 11

Inverti stringa, 17

Matrice identità, 16

Minimo, massimo e valor medio, 27

Numero di combinazioni, 26

Numero primo, 15

Quantità di cifre di un numero intero, 12

Terna pitagorica, 9

Programmi lista concatenata semplice

- Inverti elementi di lista, 46

- Lunghezza di una lista, 49

- Unione di insiemi, 47

Ricorsione, 49

sizeof, 39, 40, 46

Sottoprogrammi

- Anagramma, 39

- Fattoriale, 49

- Lunghezza di una stringa, 49

- Numero primo, 39

Sottoprogrammi ricorsivi

- Fattoriale, 49

- Lunghezza di una lista, 49

- Lunghezza di una stringa, 49

Stringhe

- strdup, 40

typedef, 16

Variabili static, 49

while, 11, 12