Questa pagina riporta esempi di prove di esame con relative soluzioni per il corso di Programmazione 1 e Laboratorio 2022/2023 tenuto all'interno del corso di studio in Informatica, Università di Catania, a cura dei Proff. Giovanni Maria Farinella, Antonino Furnari, Fabrizio Messina

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Studio in Informatica, A.A. 2022-2023
Programmazione 1 e Laboratorio
Esempio di Prova di Esame Svolta
10 gennaio 2023

Descrizione del programma

Si scriva un programma C che:

- **A** prenda un input da tastiera (argomenti della funzione main()) un intero k in [10,15], un carattere w in ['a'-'z'] e due interi N ed M in [4,8]; se gli argomenti a riga di comando non rispondono ai suddetti requisiti, il programma stampa un messaggio di errore sullo standard error e termina la propria esecuzione con codice di errore "-1";
- **B** allochi dinamicamente una matrice S di stringhe (char*) di dimensioni N x M;
- **C** riempia la matrice S con NxM stringhe di lunghezza k composte di caratteri pseudo-casuali in [a-z];
- **D** ordini ogni colonna della matrice S in modo crescente (ordinamento lessicografico) con un algoritmo di ordinamento a scelta tra Insertion Sort e Bubble Sort;
- **E** stampi la matrice sullo standard output;
- **F** stampi sullo standard output la stringa (e gli indici all'interno della matrice) che contiene il maggior numero di occorrenze del simbolo w. Queste ultime vanno sostituite, sullo standard output, con il carattere '*'.

Specifiche

Il programma potrà essere articolato in un unico file sorgente, ma dovrà contenere almeno le seguenti funzioni con opportuni parametri formali:

- **readInput**: funzione che prende in input il numero argc e il vettore argv della funzione main(), che controlli la presenza ed i requisiti degli argomenti k, w, N ed M, e che li inserisca in un record (struct) da restituire allo user code (funzione main). La funzione deve gestire correttamente gli errori relativi a input non corretti;
- **allocateS**: funzione per allocazione dinamica della matrice di dimensioni NxM, tale matrice va restituita come dato di ritorno al chiamante (funzione main());
- **genString**: funzione che restituisce una stringa della lunghezza specificata con caratteri pseudo-casuali in un ben determinato insieme specificato mediante opportuni parametri formali;
- **fillS**: funzione di riempimento della matrice S come specificato nel punto C;
- **sortS**: funzione di ordinamento della matrice come specificato nel punto D; NB: si faccia

uso, al suo interno, della funzione di libreria strcmp();
- **printMatrix**: funzione per la stampa della matrice S;
- **printMax**: funzione per la stampa della stringa contenente il maggior numero di occorrenze del simbolo w come specificato nel punto F.

Note

Durata della prova: 120 minuti

Generazione di numeri pseudocasuali:

- Si consideri la seguente funzione get_random() per la generazione di numeri pseudo-casuali interi positivi (qualora necessaria):

```c
// Scaricabile da: https://pastebin.com/f6eAKNQy
unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}
```

- NB: Ai fini della generazione di numeri in virgola mobile, si faccia uso della costante UINT_MAX (<limits.h>) unitamente alla funzione get_random().

È VIETATO usare variabili globali.

Output di controllo

Eseguendo il programma con il comando "`./main 10 x 5 4`", esso dovrà produrre il seguente output:

```
nsmpvthnts bfdtfmjqqy eivkuwpadm jcughtergo
nwexgnpgdc ctrsulhgcz jlhxsmeokj lagozauhdg
sfgdnichgb riyourjykp lfjtetzycd lewtavgvyg
uvbayzwszq tywvyxgjwy quxsqbjzri tvqxaciajv
xnmeouozlj xxxkqifjeu uwwhvqfhok xxgnebvjuy

Stringa con 3 occorrenze di 'x' trovata agli indici 4, 1:
***kqifjeu
```

Soluzione

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

```
unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}

struct data {
    int k;
    char w;
    int N;
    int M;
};

struct data readInput(char *argv[], int argc) {
    if(argc!=5) {
        fprintf(stderr, "Errore, il numero di argomenti deve essere pari a 4.\n");
        exit(-1);
    }

    struct data d = {0, 0, 0, 0};

    char **end = malloc(sizeof(char*));

    *end = NULL;

    d.k = (int) strtol(argv[1], end, 0);

    if(**end) {
        fprintf(stderr, "Errore, il primo parametro deve essere un intero.\n");
        exit(-1);
    }

    if(d.k<10 || d.k>15) {
        fprintf(stderr, "Errore, il primo parametro deve essere compreso tra 10 e
15.\n");
        exit(-1);
    }

    if(strlen(argv[2])!=1) {
        fprintf(stderr, "Errore, il secondo parametro deve essere un carattere.\n");
        exit(-1);
    }

    d.w = argv[2][0];

    if(d.w<'a' || d.w>'z') {
        fprintf(stderr, "Errore, il carattere deve essere nel range a-z.\n");
        exit(-1);
```

```
    }

    d.N = (int) strtol(argv[3], end, 0);

    if(**end) {
        fprintf(stderr, "Errore, il terzo parametro deve essere un intero.\n");
        exit(-1);
    }

    if(d.N<4 || d.N>8) {
        fprintf(stderr, "Errore, il terzo parametro deve essere nel range [4, 8].\n");
        exit(-1);
    }

    d.M = (int) strtol(argv[4], end, 0);

    if(**end) {
        fprintf(stderr, "Errore, il quarto parametro deve essere un intero.\n");
        exit(-1);
    }

    if(d.M<4 || d.M>8) {
        fprintf(stderr, "Errore, il quarto parametro deve essere nel range [4, 8].\n");
        exit(-1);
    }

    return d;
}

char*** allocateS(int N, int M) {
    char ***S = calloc(N, sizeof(char**));
    for(int i=0; i< N; i++) {
        S[i] = calloc(M, sizeof(char*));
    }

    return S;
}

char* genString(int len, char min, char max) {
    char *s = calloc(len, sizeof(char*));

    for(int i=0; i<len; i++) {
        s[i] = get_random() % (max-min+1) + min;
    }

    return s;
}

void fillS(char ***S, int N, int M, int k) {
    for(int i = 0; i<N; i++) {
```

```
        for(int j=0; j<M; j++) {
            S[i][j] = genString(k, 'a', 'z');
        }
    }
}

void sortS(char ***S, int N, int M) {
    for(int j=0; j<M; j++) { //scorro le colonne
        for(int pass=0; pass<N-1; pass++) {
            for(int i=0; i<N-1-pass; i++) {
                if(strcmp(S[i][j], S[i+1][j])>0) {
                    char *tmp = S[i][j];
                    S[i][j] = S[i+1][j];
                    S[i+1][j] = tmp;
                }
            }
        }
    }
}

void printMatrix(char ***S, int N, int M) {
    for(int i=0; i<N; i++) {
        for(int j=0; j<M; j++) {
            printf("%s ", S[i][j]);
        }
        printf("\n");
    }
}

int conta(char* s, char w){
    int c = 0;
    for(int i=0; i<strlen(s); i++) {
        if(s[i]==w)
            c++;
    }

    return c;
}

void printMax(char ***S, int N, int M, char w) {
    int i_max = 0;
    int j_max = 0;
    int max = 0;

    for(int i =0; i<N; i++) {
        for(int j=0; j<M; j++) {
            int count = conta(S[i][j], w);
            if(count>max) {
                max = count;
                i_max = i;
```

```
                j_max = j;
            }
        }
    }

    printf("\nStringa con %d occorrenze di '%c' trovata agli indici %d, %d: ", max, w,
i_max, j_max);

    for(int i=0; i<strlen(S[i_max][j_max]); i++) {
        char c = S[i_max][j_max][i];
        c = (c==w) ? '*' : c;
        printf("%c", c);
    }

    printf("\n");


}

int main(int argc, char* argv[]) {
    struct data d = readInput(argv, argc);
    char ***S = allocateS(d.N, d.M);
    fillS(S, d.N, d.M, d.k);
    sortS(S, d.N, d.M);
    printMatrix(S, d.N, d.M);
    printMax(S, d.N, d.M, d.w);
}
```

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Studio in Informatica, A.A. 2022-2023
Programmazione 1 e Laboratorio
Esempio di Prova di Esame Svolta
10 gennaio 2023

Descrizione del programma

Si scriva un programma C che:

- **A** chieda all'utente di inserire un intero “n” (si assuma n<256), una stringa "s1", una stringa “s2”, e un carattere "c" da tastiera. Si verifichi che entrambe le stringhe siano di lunghezza “n”. In caso contrario, si stampi un errore su standard error e si termini il programma con un opportuno codice di terminazione;
- **B** costruisca una nuova stringa “s3” ottenuta sostituendo tutte le occorrenze del carattere "c" in “s1” con i caratteri che si trovano in “s2” nelle posizioni corrispondenti;
- **C** definisca una nuova stringa “s4” ottenuta invertendo l’ordine dei caratteri in “s2”;
- **D** concateni le stringhe “s3” e “s4” in una nuova stringa “s5” e la ordini in ordine

lessicografico ascendente usando un algoritmo di ordinamento a scelta;
- **E** stampi a schermo la stringa ordinata. I caratteri i cui codici numerici relativi (fare cast dei caratteri a int) siano dispari, vanno sostituiti con "*".

Specifiche

Il programma potrà essere articolato in un unico file sorgente, ma dovrà contenere almeno le seguenti funzioni con opportuni parametri formali:

- **readInput**: funzione che permette di leggere gli input da tastiera e restituisce opportunamente i valori letti al chiamante. La funzione deve gestire correttamente gli errori relativi a input non corretti e gestire la terminazione del programma in caso di errori.
- **replaceChar**: funzione che riceve come input le stringhe "s1", "s2" e il carattere "c" e restituisce la stringa con le occorrenze di "c" in "s1" sostituite dai corrispondenti valori in "s2. Ad esempio, se s1="abcacba", s2="fhsuika" e c='a', allora il risultato sarà "fbcucba";
- **invertString**: funzione che costruisce una stringa a partire da una stringa in input invertendono i caratteri;
- **sort**: funzione che permetta di ordinare una stringa in ordine lessicografico;
- **printResult**: funzione che stampa a schermo la stringa ottenuta come specificato nel punto E.

Note

Durata della prova: 120 minuti

Generazione di numeri pseudocasuali:

- Si consideri la seguente funzione get_random() per la generazione di numeri pseudo-casuali interi positivi (qualora necessaria):

- NB: Ai fini della generazione di numeri in virgola mobile, si faccia uso della costante UINT_MAX (<limits.h>) unitamente alla funzione get_random().

È VIETATO usare variabili globali.

```c
// Scaricabile da: https://pastebin.com/f6eAKNQy
unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}
```

Output di controllo

Eseguendo il programma con i seguenti input:

```
jsudbehdnbvsju9
kduw76_lposhndb
s
```

Il programma dovrà stampare il seguente output:

```
6***bbbdddddd*hhhjj*lnn*p****v*
```

N.B. Si consiglia di creare un file “input.txt” contenente i dati di input specificati sopra e di testare il programma sviluppato mediante la redirezione dello standard input.

Soluzione

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

void readInput(char *s1, char *s2, int *n, char *c) {
    printf("%s", "Si inserica un intero n: ");
    scanf("%d", n);

    printf("Si inserisca una stringa di lunghezza %d: ", *n);
    scanf("%s", s1);

    printf("Si inserisca una stringa di lunghezza %d: ", *n);
    scanf("%s", s2);

    if(strlen(s1)!=*n || strlen(s2)!=*n) {
        fprintf(stderr, "Errore! Le stringhe non sono di lunghezza %d\n", *n);
        exit(-1);
    }

    while(!getchar());

    printf("Inserire un carattere: ");
    scanf("%c", c);
}

char* replaceChar(char *s1, char *s2, char c) {
    int n = strlen(s1);
    char *out = calloc(n, sizeof(char)+1);

    for(int i = 0; i<n; i++) {
        if(s1[i]==c)
            out[i] = s2[i];
        else
            out[i] = s1[i];
```

```
    }

    return out;
}

char* invertString(char *s) {
    char* out = calloc(strlen(s)+1, sizeof(char));
    for(int i=0; i<strlen(s); i++) {
        out[strlen(s)-i-1] = s[i];
    }
    out[strlen(s)] = 0;
    return out;
}

void sort(char *s) {
    int n = strlen(s);

    for(int pass = 0; pass < n-1; pass++) {
        for (int i=0; i<n-1-pass; i++) {
            if(s[i]>s[i+1]) {
                char c = s[i];
                s[i] = s[i+1];
                s[i+1] = c;
            }
        }
    }
}

void printResult(char *s) {
    for(int i=0; i<strlen(s); i++) {
        if(s[i] % 2==1)
            printf("*");
        else
            printf("%c", s[i]);
    }

    printf("\n");
}

int main() {
    char s1[255];
    char s2[255];
    int n;
    char c;

    readInput(s1, s2, &n, &c);

    char *s3 = replaceChar(s1, s2, c);
    char *s4 = invertString(s2);
    char *s5 = strcat(s3, s4);
```

```
    sort(s5);

    printResult(s5);
}
```

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Studio in Informatica, A.A. 2022-2023
Programmazione 1 e Laboratorio
Esempio di Prova di Esame Svolta
10 gennaio 2023

Descrizione del programma

Scrivere un programma in C che:

- **A** prenda in input da tastiera (argomenti della funzione main) un intero positivo N in [10,20] ed un carattere w; se gli argomenti a riga di comando non rispondono ai suddetti requisiti, il programma stampa un messaggio di errore sullo standard error e termina la propria esecuzione;
- **B** generi, mediante successivi inserimenti, una lista concatenata semplice (ordinata in modo crescente - ordinamento lessicografico) che contenga N stringhe con caratteri pseudo-casuali in [‘a’-’z’] di lunghezza pseudo-casuale L nell'intervallo [5,15];
- **C** stampi sullo standard output l'intera lista;
- **D** stampi sullo standard output il numero totale di occorrenze del carattere w in tutte le stringhe della lista.

Specifiche

Il programma potrà essere articolato in un unico file sorgente, ma dovrà contenere almeno le seguenti funzioni con opportuni parametri formali:

- **readInput**: funzione che prenda in input gli argomenti della funzione main argv ed argc, controlli i requisiti dei parametri e restituisca al chiamante un record (struct) che contenga tutti i parametri.
- **genString**: funzione che restituisca una stringa di caratteri pseudo-casuali appartenenti ad un determinato intervallo specificato mediante opportuni parametri formali.
- **insertString:** funzione che inserisca una stringa nella lista; per tale funzione e’ possibile usare la funzione di libreria strcmp();
- **genList:** funzione che si occupa di creare e riempire la lista di stringhe come richiesto nel punto B;
- **printList:** funzione che si occupa di stampare la lista a schermo come specificato nel punto C;
- **printOcc:** funzione che stampa sullo standard output il numero totale di occorrenze

del carattere w in tutte le stringhe della lista, come specificato nel punto D.

Note

Durata della prova: 120 minuti

Generazione di numeri pseudocasuali:

- Si consideri la seguente funzione get_random() per la generazione di numeri pseudo-casuali interi positivi (qualora necessaria):

```c
// Scaricabile da: https://pastebin.com/f6eAKNQy
unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}
```

- NB: Ai fini della generazione di numeri in virgola mobile, si faccia uso della costante UINT_MAX (<limits.h>) unitamente alla funzione get_random().

È VIETATO usare variabili globali.

Output di controllo

Eseguendo il programma con il comando `./main 14 c`, esso dovrà stampare il seguente output:

```
dnichgbctrsulhg
ebvju
hterg
hxsmeokjlag
nsmpvthnt
tywvyxgjwy
wexgnpgd
wwhvqfhokjcu
xnmeou
xxxkqifjeulfjte
zauhdgsf
zeivkuwpa
zljriyourjykpj
zycdxxg
Numero totale di occorrenze di c: 4
```

Soluzione

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

```c
#include<string.h>

unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}

struct parametri {
    int N;
    char w;
};

struct node {
    char data[16];
    struct node *next;
};

typedef struct node Node;

void insertHead(Node **head, char *string) {
    if(*head==NULL) {
        *head = malloc(sizeof(Node));
        strcpy((*head)->data, string);
        (*head)->next=NULL;
    } else {
        Node *new = malloc(sizeof(Node));
        strcpy(new->data, string);
        new->next = *head;
        *head = new;
    }
}

void insertString(Node **head, char *string) {
    if(*head==NULL) {
        insertHead(head, string);
        return;
    }

    Node *ptr = *head;
    Node *prev = NULL;

    while(ptr && strcmp(ptr->data, string)<0) {
        prev = ptr;
        ptr = ptr->next;
    }

    if(prev==NULL) {
```

```
        insertHead(head, string);
        return;
    }

    Node *new = malloc(sizeof(Node));

    strcpy(new->data, string);
    prev->next = new;
    new->next = ptr;
}

void printList(Node *head) {
    Node *tmp = head;

    while(tmp) {
        printf("%s\n", tmp->data);
        tmp = tmp->next;
    }
}

char* genString(int len, const char range[2]) {
    char* s=calloc(len, sizeof(char*));

    for(int i=0; i<len; i++) {
        s[i] = get_random() % (range[1] - range[0]+1)+range[0];
    }

    return s;
}

struct parametri readInput(char*argv[], int argc) {
    if(argc!=3) {
        fprintf(stderr, "Errore, bisogna passare due parametri da riga di comando.\n");
        exit(-1);
    }

    char **end = malloc(sizeof(char*));
    int N = (int) strtod(argv[1], end);

    if(N<10 || N>20) {
        fprintf(stderr, "Errore, il primo parametro deve essere compreso tra 10 e
20.\n");
        exit(-1);
    }

    if(!end) {
        fprintf(stderr, "Errore, il primo parametro deve essere un intero.\n");
        exit(-1);
    }
```

```
    if(strlen(argv[2])!=1) {
        fprintf(stderr, "Errore, il secondo parametro deve essere un carattere.\n");
        exit(-1);
    }

    struct parametri p = {N, argv[2][0]};

    return p;
}

Node** genList(int N) {
    Node **head = malloc(sizeof(Node*));
    *head = NULL;

    char range[2] = {'a', 'z'};

    char *tmp = NULL;

    for(int i=0; i<N; i++) {
        int L = get_random() % (15-5+1)+5;
        tmp = genString(L, range);
        insertString(head, tmp);
        free(tmp);
    }

    return head;
}

void printOcc(Node *head, char w) {
    Node *tmp = head;

    int count = 0;

    while(tmp) {
        char* s = tmp->data;

        for(int i=0; i<strlen(s); i++) {
            if(s[i]==w)
                count++;
        }

        tmp = tmp->next;
    }

    printf("Numero totale di occorrenze di %c: %d\n", w, count);
}

int main(int argc, char* argv[]) {
    struct parametri p = readInput(argv, argc);
    Node** head = genList(p.N);
```

```
    printList(*head);
    printOcc(*head, p.w);
}
```

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Studio in Informatica, A.A. 2022-2023
Programmazione 1 e Laboratorio
Esempio di Prova di Esame Svolta
10 gennaio 2023

Descrizione del programma

Scrivere un programma in C che:

- **A** prenda in input da tastiera un intero positivo “n”;
- **B** legga un file di testo "input.txt". Si assuma che il file di testo contenga sequenze di parole separate da spazi. Si assuma una lunghezza massima per ciascuna parola pari di 100 caratteri, mentre una lunghezza massima di ciascuna riga pari a 1000 caratteri. Il programma dovrà creare una lista di struct a partire dal contenuto del file. Ogni struct deve rappresentare una riga del file di input e deve contenere due campi: una stringa "word” e un intero "count". La stringa "word" deve contenere la prima parola nella riga del file di input, mentre "count" deve contenere il numero di occorrenze di tale parola nella riga;
- **C** una volta creata la lista, elimini da essa le parole aventi un numero di occorrenze inferiore a “n”;
- **D** stampi a schermo le parole presenti nella lista con il relativo numero di occorrenze.

Specifiche

Il programma potrà essere articolato in un unico file sorgente, ma dovrà contenere almeno le seguenti funzioni con opportuni parametri formali:

- **readN**: funzione che permette di leggere l’intero n come parametro passato da riga di comando. Questa funzione dovrà gestire correttamente eventuali errori di input, stampare messaggi di errore su standard error e terminare il programma opportunamente ove necessario;
- **readFile**: funzione che apre il file di input (il cui nome è passato mediante un apposito parametro formale), legge le righe e restituisce la lista concatenata di struct;
- **filterList**: funzione che elimina dalla lista le parole con meno di “n” occorrenze;
- **printList**: funzione che stampa il contenuto della lista concatenata.

Note

Durata della prova: 120 minuti
Generazione di numeri pseudocasuali:

- Si consideri la seguente funzione get_random() per la generazione di numeri pseudo-casuali interi positivi (qualora necessaria):

```
// Scaricabile da: https://pastebin.com/f6eAKNQy
unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}
```

- NB: Ai fini della generazione di numeri in virgola mobile, si faccia uso della costante UINT_MAX (<limits.h>) unitamente alla funzione get_random().

È VIETATO usare variabili globali.

Output di controllo

Si consideri il seguente file “input.txt”:

```
frase questa frase è una bella frase
testo ma non è molto utile
albero il nostro albero non può essere un albero qualsiasi
eccomi qui qui sei tu eccomi eccomi eccomi
ancora tu ma non dovevamo ancora tu tu tu ancora?
ecco alto ello alto ecco
eccomi qui qui sei tu eccomi eccomi eccomi
```

L’esecuzione del programma con n=3 (`./main 3`) dovrà produrre il seguente output:

```
frase 3
albero 3
eccomi 4
eccomi 4
```

Soluzione

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

typedef struct {
    char word[100];
```

```
    int count;
} Riga;

struct node {
    Riga data;
    struct node *next;
};

typedef struct node Node;

bool insertHead(Node **head, Riga r) {
    Node *newNode = malloc(sizeof(Node));
    if(newNode==NULL)
        return false;
    newNode->data = r;

    newNode->next = *head;
    *head = newNode;

    return true;
}

bool insertTail(Node **head, Riga r) {
    if(*head==NULL) {
        return insertHead(head, r);
    }

    Node *ptr = *head;

    while(ptr->next) {
        ptr = ptr->next;
    }

    Node *newNode = malloc(sizeof(Node));

    if(newNode==NULL)
        return false;

    newNode->data = r;
    newNode->next = NULL;

    ptr->next = newNode;

    return true;
}

Node* readFile(char *name) {
    Node **head = malloc(sizeof(Node*));
    *head = NULL;
    FILE *f = fopen(name, "r");
```

```
    char s[1000];

    bool done = false;

    while(!done) {
        char *out = fgets(s, 1000, f);
        if(out!=NULL) {
            char *tok = strtok(s, " \n");
            char *firstTok = tok;
            int count = 0;
            do {
                if(strcmp(firstTok, tok)==0) {
                    count++;
                }

            } while((tok=strtok(NULL, " \n")));
            Riga r;
            r.count = count;
            strcpy(r.word, firstTok);
            insertTail(head, r);
        } else {
            done = true;
        }
    }

    return *head;
}

void printList(Node *head) {
    Node *ptr = head;

    while(ptr) {
        Riga *r = &ptr->data;
        printf("%s %d\n", r->word, r->count);
        ptr = ptr->next;
    }
}

bool deleteHead(Node **head) {
    if(*head) {
        Node *tmp = *head;
        *head = (*head)->next;
        free(tmp);
        return true;
    } else {
        return false;
    }
}
```

```c
void filterList(Node **head, int n) {
    if(*head==NULL) {
        return;
    }

    Node *ptr = *head;
    Node *prevPtr = NULL;

    while(ptr) {
        if(ptr->data.count<n) {
            if(prevPtr==NULL) {
                deleteHead(head);
                ptr = *head;
                continue;
            } else {
                Node *tmp = ptr;
                prevPtr->next = ptr->next;
                free(tmp);
                ptr = prevPtr;
            }
        }

        prevPtr = ptr;
        ptr = ptr->next;
    }
}

int readN(int argc, char *argv[]) {
    if(argc!=2) {
        fprintf(stderr, "Errore, bisogna passare un parametro intero da riga di
comando.\n");
        exit(-1);
    }

    char **end = malloc(sizeof(char*));
    int n = (int) strtol(argv[1], end, 0);

    if(!end || n < 0) {
        fprintf(stderr, "Errore, il parametro passato deve essere intero positivo.\n");
        exit(-1);
    }

    return n;
}

int main(int argc, char *argv[]) {
    int n = readN(argc, argv);

    Node **head = malloc(sizeof(Node*));
```

```
    *head = readFile("input.txt");
    filterList(head, n);
    printList(*head);
}
```

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Studio in Informatica, A.A. 2022-2023
Programmazione 1 e Laboratorio
Esempio di Prova di Esame Svolta
10 gennaio 2023

Descrizione del programma

Scrivere un programma in C che:

- **A** prenda un input da tastiera (argomenti della funzione main) costituito da un intero positivo N in [10,20], e due numeri in virgola mobile positivi x,y. Dovra' essere 5.0 ≤ x < y ≤ 30.0; se gli argomenti a riga di comando non rispondono ai suddetti requisiti, il programma stampa un messaggio di errore sullo standard error e termina la propria esecuzione;
- **B** allochi dinamicamente una matrice A di numeri double pseudo-casuali in [x,y] di dimensioni N x N;
- **C** calcoli il minimo degli elementi della diagonale principale della matrice (sia mind), e il massimo valore degli elementi della diagonale secondaria della matrice stessa (sia maxd); si restituisca inoltre il numero di elementi della matrice aventi valori in [mind, maxd];
- **D** allochi dinamicamente un array di double e lo riempia con tutti gli elementi di A nell'intervallo [mind, maxd];
- **E** ordini l'array mediante un algoritmo a scelta tra selection sort e insertion sort;
- **F** stampi l'array sullo standard output, insieme alla media aritmetica dei suoi elementi.

Specifiche

Il programma potrà essere articolato in un unico file sorgente, ma dovrà contenere almeno le seguenti funzioni con opportuni parametri formali:

- **readInput**: funzione che prenda in input gli argomenti argv ed argc della funzione main, controlli i requisiti dei parametri e restituisca un record (una struct) che contiene tali parametri. Verificare che i requisiti dei parametri siano rispettati come specificato nel punto A. Nel caso in cui i requisiti non fossero specificati, stampare un opportuno messaggio di errore e terminare il programma;
- **genDouble:** genera un numero double in un intervallo specificato mediante parametri formali;
- **genMatrix**: funzione che crea e restituisce una matrice con elementi double generati dalla funzione genDouble;

- **computeMinMax:** calcola mind e maxd e restituisce il numeri di elementi che ricadono nell'intervallo [mind, maxd] come specificato nel punto C. La funzione permette di restituire al chiamante anche i valori di mind e maxd;
- **createArray:** creazione e riempimento array come specificato nel punto D;
- **sortArray:** ordinamento dell'array come specificato nel punto E;
- **printArray**: stampa dell'array come specificato nel punto F.

Note

Durata della prova: 120 minuti

Generazione di numeri pseudocasuali:

- Si consideri la seguente funzione get_random() per la generazione di numeri pseudo-casuali interi positivi (qualora necessaria):

```c
// Scaricabile da: https://pastebin.com/f6eAKNQy
unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}
```

- NB: Ai fini della generazione di numeri in virgola mobile, si faccia uso della costante UINT_MAX (<limits.h>) unitamente alla funzione get_random().

È VIETATO usare variabili globali.

Output di controllo

L'esecuzione del programma con il comando `./main 10 5 10` dovrà produrre il seguente output:

```
6.122730
6.162632
6.194943
6.208264
6.209351
6.214056
6.235616
6.243339
6.262306
6.330648
6.368935
6.378846
6.390884
6.449782
6.521540
6.715844
6.778278
```

```
6.808093
6.821375
6.865654
6.892895
6.946560
6.955153
7.069172
7.094966
7.130167
7.137448
7.204647
7.212355
7.238173
7.403624
7.678935
7.779727
7.797982
7.850210
8.079907
8.096187
8.156989
8.177093
8.179078
8.241983
8.258611
8.261388
8.285773
8.343275
8.353323
8.365839
8.377934
8.568861
8.595750
8.777560
8.845153
8.863467
Media: 7.330251
```

Soluzione

```c
#include<stdio.h>
#include<limits.h>
#include<stdlib.h>

unsigned int get_random()
{
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
```

```
}

struct parametri {
    int N;
    float x;
    float y;
};

struct parametri readInput(int argc, char* argv[]) {
    if(argc!=4) {
        fprintf(stderr, "Errore: il numero di argomenti deve essere pari a 3.\n");
        fprintf(stderr, "Uso corretto: %s N x y.\n", argv[0]);
        exit(-1);
    }

    struct parametri p;

    char **end = malloc(sizeof(char*));
    p.N = (int) strtol(argv[1], end, 0);

    if(!end) {
        fprintf(stderr, "Errore: il primo parametro deve essere un intero.\n");
        exit(-1);
    }

    if(p.N<10 || p.N>20) {
        fprintf(stderr, "Errore: il primo parametro deve essere compreso tra 10 e
20.\n");
        exit(-1);
    }

    p.x = strtof(argv[2], end);

    if(!end) {
        fprintf(stderr, "Errore: il secondo parametro deve essere un float.\n");
        exit(-1);
    }

    p.y = strtof(argv[3], end);

    if(!end) {
        fprintf(stderr, "Errore: il terzo parametro deve essere un float.\n");
        exit(-1);
    }

    if(p.x<5 || p.x>30 || p.y<5 || p.y>30) {
        fprintf(stderr, "Errore: il secondo e il terzo parametro devono essere compresi
tra 5.0 e 30.0.\n");
        exit(-1);
    }
```

```
    if(p.x>=p.y) {
        fprintf(stderr, "Errore: deve essere x < y.\n");
        exit(-1);
    }

    return p;
}

double genDouble(float x, float y) {
    return ((double) get_random() / UINT_MAX) * (y-x)+x;
}

double ** genMatrix(int N, float x, float y) {
    double **A = calloc(N, sizeof(double*));
    for(int i=0; i<N; i++) {
        A[i] = calloc(N, sizeof(double));

        for(int j=0; j<N; j++) {
            A[i][j] = genDouble(x, y);
        }
    }

    return A;
}

int computeMinMax(double **A, int N, double *mind, double *maxd) {
    *mind = A[0][0];
    *maxd = A[N-1][0];

    for(int i=0; i<N; i++) {
        if(A[i][i]<*mind)
            *mind = A[i][i];
        if(A[i][N-1-i]>*maxd)
            *maxd = A[i][N-1-i];
    }

    int count = 0;

    for(int i=0; i<N; i++) {
        for(int j=0; j<N; j++) {
            if(A[i][j]<=*maxd && A[i][j]>=*mind) {
                count++;
            }
        }
    }

    return count;
}
```

```
double* createArray(double **A, int N, int num, double mind, double maxd) {
    double *out = calloc(num, sizeof(double));

    int idx = 0;
    for(int i=0; i<N; i++) {
        for(int j=0; j<N; j++) {
            if(A[i][j]<=maxd && A[i][j]>=mind) {
                out[idx++] = A[i][j];
            }
        }
    }

    return out;
}

void sortArray(double array[], int num) {
    // selectionsort
    for(int i = 0; i<num-1; i++) {
        int smallest = i;
        for(int j=i+1; j<num; j++) {
            if(array[j] < array[smallest]) {
                smallest = j;

            }
        }

        double tmp = array[i];
        array[i] = array[smallest];
        array[smallest] = tmp;
    }
}

void printArray(double array[], int num) {
    double mean = 0;
    for(int i=0; i<num; i++) {
        printf("%f\n", array[i]);
        mean += array[i];
    }
    mean/=num;

    printf("Media: %f\n", mean);
}

int main(int argc, char* argv[]) {
    struct parametri p = readInput(argc, argv);
    double **A = genMatrix(p.N, p.x, p.y);

    double minp, maxd;

    int num = computeMinMax(A, p.N, &minp, &maxd);
```

```
    double *array = createArray(A, p.N, num, minp, maxd);

    sortArray(array, num);
    printArray(array, num);
}
```

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Studio in Informatica, A.A. 2022-2023
Programmazione 1 e Laboratorio
Esempio di Prova di Esame Svolta
10 gennaio 2023

Descrizione del programma

Scrivere un programma in C che:

- **A** prenda come argomenti da riga di comando il nome di un file di testo di input "input" (es. "input.txt") e il nome di un file di testo di output "output" (es. "output.txt"). Si assuma che i nomi dei file non superano i 255 caratteri.
- **B** legga una matrice A di puntatori float di dimensioni n x m dal file "input". Si assuma che il file contenga un numero di righe di testo pari a n e che ogni riga contenga m valori separati da spazi. Il programma dovrà inferire le dimensioni n e m dal file di input e riempire opportunamente i valori della matrice;
- **C** elimini dalla matrice i valori di A che sono superiori ai valori medi delle righe corrispondenti (far puntare a NULL i puntatori relativi ai valori identificati e liberare la relativa memoria);
- **D** inserisca i valori puntati dai puntatori non nulli in A in un array di float;
- **E** scriva i valori dell'array sul file "output" il cui nome è stato passato come argomento da riga di comando (es "output.txt").

Specifiche

Il programma potrà essere articolato in un unico file sorgente, ma dovrà contenere <u>almeno le seguenti funzioni con opportuni parametri formali:</u>

- **readParameters**: funzione che permetta di leggere i parametri da riga di comando, verificando che rispettino i criteri specificati nel punto A, gestendo eventuali messaggi di errore e terminando il programma ove opportuno;
- **createMatrix**: funzione che permetta di creare la matrice A a partire dal nome del file di input;
- **sparsify**: funzione che permetta di mettere a NULL gli opportuni puntatori di A (come da punto C);
- **collect**: funzione che permetta di creare e riempire l'array di float come descritto nel punto D;
- **writeToFile**: funzione che permetta di scrivere i valori della lista su file.

Note

Durata della prova: 120 minuti
Generazione di numeri pseudocasuali:

- Si consideri la seguente funzione get_random() per la generazione di numeri pseudo-casuali interi positivi (qualora necessaria):

```c
// Scaricabile da: https://pastebin.com/f6eAKNQy
unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}
```

- NB: Ai fini della generazione di numeri in virgola mobile, si faccia uso della costante UINT_MAX (<limits.h>) unitamente alla funzione get_random().

È VIETATO usare variabili globali.

Output di controllo

Dato il seguente file `input.txt`:

```
1.2  5.1  3.7 8.2  2.8
9.7 -8.2 -1.4 3.5  2.65
0.0  2.2  5.0 1.8 -0.3
```

Il programma dovrà generare il seguente file `output.txt`:

```
 1.20
 3.70
 2.80
-8.20
-1.40
 0.00
-0.30
```

Soluzione

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

struct parametri {
    char input[255];
    char output[255];
};
```

```c
struct parametri readParameters(int argc, char * argv[]) {
    if(argc!=3) {
        fprintf(stderr, "Errore: numero di parametri non corretto.\n");
        fprintf(stderr, "Uso corretto: %s input_file output_file,\n", argv[0]);
        exit(-1);
    }

    struct parametri p;

    strcpy(p.input, argv[1]);
    strcpy(p.output, argv[2]);

    return p;
}

float*** createMatrix(char filename[], int* n, int* m) {
    FILE *f = fopen(filename, "r");

    if(f==NULL) {
        fprintf(stderr, "Errore: impossibile aprire file %s.\n", filename);
        exit(-1);
    }

    char s[1000];
    char *end;

    *n = 0;
    *m = 0;

    do {
        end = fgets(s, 1000, f);
        (*n)++;
        if(*m==0) {
            char *tok;
            char *tmp = s;
            do {
                tok = strtok(tmp, " \n");
                tmp = NULL;
                (*m)++;
            } while(tok);
        }
    } while(end);

    (*n)--;
    (*m)--;

    fseek(f, 0, 0);

    float ***A = calloc(*n, sizeof(float**));
    for(int i = 0; i<*n; i++) {
```

```
        A[i] = calloc(*m, sizeof(float*));
        for(int j=0; j<*m; j++) {
            A[i][j] = NULL;
        }
    }

    for(int i=0; i<*n; i++) {
        end = fgets(s, 1000, f);
        char *tok;
        char *tmp = s;

        for(int j=0; j<*m; j++) {
            tok = strtok(tmp, " \n");
            tmp = NULL;
            if(tok) {
                char **e = malloc(sizeof(char*));
                A[i][j] = malloc(sizeof(float));
                *(A[i][j]) = strtof(tok, e);
                free(e);
            }
        }
    }

    return A;
}

void sparsify(float ***A, int n, int m) {
    for(int i=0; i<n; i++) { //righe
        float mean = 0;
        int count = 0;
        for(int j=0; j<m; j++) {
            if(A[i][j]) {
                mean += *A[i][j];
                count++;
            }
        }
        mean /= count;

        for(int j=0; j<m; j++) {
            if(A[i][j] && *A[i][j]>mean) {
                float *tmp = A[i][j];
                A[i][j] = NULL;
                free(tmp);
            }
        }
    }
}

float* collect(float ***A, int n, int m, int* num) {
    *num = 0;
```

```
    for(int i=0; i<n; i++) {
        for(int j=0; j<m; j++) {
            if(A[i][j])
                (*num)++;
        }
    }

    float *out = calloc(*num, sizeof(float));

    int c = 0;

    for(int i=0; i<n; i++) {
        for(int j=0; j<m; j++) {
            if(A[i][j])
                out[c++] = *A[i][j];
        }
    }

    return out;
}

void writeToFile(char output[], float *array, int n) {
    FILE *f = fopen(output, "w");

    for(int i=0; i<n; i++) {
        fprintf(f, "%5.2f\n", array[i]);
    }
}

int main(int argc, char * argv[]) {
    struct parametri p = readParameters(argc, argv);

    int N, M;
    float ***A = createMatrix(p.input, &N, &M);
    sparsify(A, N, M);

    int num;
    float *out = collect(A, N, M, &num);

    writeToFile(p.output, out, num);
}
```

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Studio in Informatica, A.A. 2022-2023
Programmazione 1 e Laboratorio
Esempio di Prova di Esame Svolta

10 gennaio 2023

Descrizione del programma

Scrivere un programma in C che:

- **A** prenda un input da tastiera (argomenti della funzione main) costituito da un intero positivo N. Il programma deve verificare che N sia inserito nel formato corretto e restituire un messaggio di errore terminando il programma opportunamente nel caso in cui il formato di N non dovesse essere corretto;
- **B** generi una sequenza di N operazioni di inserimento (push) di caratteri pseudo-casuali in [A-Z,a-z,1-9] in una struttura dati LIFO dinamica (pila o stack) da implementare mediante lista concatenata semplice (top==testa della lista), nel seguente modo:
    - ad ogni passo, si generi innanzitutto un carattere x in [1-9];
    - se x rappresenta un numero in [1-4], allora si proceda ad x operazioni di inserimento (push) di caratteri pseudo-casuali che siano vocali, seguite dall'inserimento del carattere x;
    - se x rappresenta un numero in [5-9], allora si proceda ad x operazioni di inserimento (push) di caratteri pseudo-casuali che siano consonanti, seguite dall'inserimento del carattere x;
    - in particolare, sia c il generico carattere da inserire sullo stack:
        - se c==v, si inserisca sullo stack al posto di esso il carattere '*';
        - se c==w, si inserisca sullo stack al posto di esso il carattere '?';
- **C**
    - crei un array di stringhe (puntatori a caratteri) di lunghezza N;
    - proceda nel seguente modo, fino a svuotamento dello stack: successivamente proceda ad una sequenza di operazioni di rimozione (pop()) come segue:
        - si proceda quindi, ad ogni passo con un'operazione di rimozione (pop()) del carattere x che indica (per costruzione) il numero di caratteri da rimuovere successivamente mediante x operazioni pop();
        - si memorizzi sull'array di caratteri ogni stringa composta dal carattere x e dai successivi x caratteri;

- **D** stampi, sullo standard output, il contenuto dell'array di stringhe;

Specifiche

Il programma potrà essere articolato in un unico file sorgente, ma dovrà contenere almeno le seguenti funzioni con opportuni parametri formali:

- **readInput:** funzione che prende in input l'array di puntatori a carattere argv della funzione main, controlli che gli argomenti richiesti siano nei limiti specificati, e restituisca il record (struct) che contiene tali parametri; se il controllo non va a buon fine, stampa un messaggio sullo standard error e termina il programma.
- **genVowel:** funzione che produca un carattere vocale pseudo-casuale;
- **genConsonant:** funzione che produca un carattere consonante pseudo-casuale;
- **push e pop:** funzioni che consentono di inserire un elemento sullo stack o rimuovere un elemento da esso;
- **fillStack:** funzione con opportuni parametri formali che rappresenti

l'implementazione della procedura descritta nel punto B;
- **emptyStack:** funzione con opportuni parametri formali che sia conforme alla procedura descritta nel punto C;
- **printArray:** funzione per la stampa dell'array di stringhe prodotto al punto C.

Note

Durata della prova: 120 minuti

Generazione di numeri pseudocasuali:

· Si consideri la seguente funzione get_random() per la generazione di numeri pseudo-casuali interi positivi (qualora necessaria):

```c
// Scaricabile da: https://pastebin.com/f6eAKNQy
unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}
```

· NB: Ai fini della generazione di numeri in virgola mobile, si faccia uso della costante UINT_MAX (<limits.h>) unitamente alla funzione get_random().

È VIETATO usare variabili globali.

Output di controllo

Il programma eseguito mediante il seguente comando: `./main 12` dovrà restituire il seguente output:

```
R*tKm
Iou
SyDZMh
o
LZfWB
KMjzCNsPR
GtVcSGGXt
iIIE
XpcQXLqQ?
qKhWXfjH
y?QsSSX
q?VJMSQl
```

Soluzione

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

```c
unsigned int get_random()
{
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}


struct parametri {
    int N;
};

struct parametri readInput(char * argv[], int argc) {
    if(argc!=2) {
        fprintf(stderr, "Errore: il numero di parametri deve essere pari a 1.\n");
        exit(-1);
    }

    char** end = malloc(sizeof(char*));

    int x = (int) strtol(argv[1], end, 0);

    if(!*end || x<0){
        fprintf(stderr, "Errore: il parametro inserito deve essere un intero
positivo.\n");
        exit(-1);
    }

    struct parametri p = {x};

    return p;
}

struct node {
    char data;
    struct node* next;
};

typedef struct node Node;

bool empty(Node *head) {
    return head==NULL;
}

void push(Node** head, char x) {
    Node *new = malloc(sizeof(Node));
```

```
    new->data = x;
    new->next = *head;
    *head = new;
}

char pop(Node** head) {
    if(empty(*head)) {
        fprintf(stderr, "Errore: pila vuota!\n");
        exit(-1);
    }

    char x = (*head)->data;
    Node* tmp = *head;
    (*head) = (*head)->next;

    free(tmp);

    return x;
}

char getRandomChar(char *s) {
    return s[get_random() %strlen(s)];
}

char getVowel() {
    return getRandomChar("AEIOUaeiou");
}

char getConsonant() {
    return getRandomChar("QWRTYPDSFGHJKLZXCVBNMqwrtypdsfghjklzxcvbnm");
}

int char2int(char c) {
    char tmp[2];
    tmp[0] = c;
    tmp[1] = 0;
    char **end = malloc(sizeof(char*));
    int x = (int) strtol(tmp, end, 0);
    free(end);
    return x;
}

Node** fillStack(int N) {
    Node **head = malloc(sizeof(Node*));
    *head = NULL;

    int i = 0;


    for(int j = 0; j<N; j++) {
```

```
        char x = getRandomChar("123456789");
        if(x>='1' && x <= '4') {
            for(int i=0; i< char2int(x); i++) {
                push(head, getVowel());
            }
            push(head, x);
        } else {
            for(int i=0; i< char2int(x); i++) {
                char c = getConsonant();
                if(c=='v')
                    c='*';
                if(c=='w')
                    c='?';
                push(head, c);
            }
            push(head, x);
        }
    }

    return head;
}

char** emptyStack(Node **head, int N) {
    char** array = calloc(N, sizeof(char*));
    int i = 0;
    while(!empty(*head)) {
        char c = pop(head);
        int x = char2int(c);
        array[i] = calloc(x, sizeof(char)+1);
        for(int j=0; j<x; j++) {
            array[i][j] = pop(head);
        }
        i++;
    }

    return array;
}

void printArray(char** array, int N) {
    for(int i=0; i<N; i++) {
        printf("%s\n",array[i]);
    }
}

int main(int argc, char* argv[]) {
    struct parametri p = readInput(argv, argc);
    Node **head = fillStack(p.N);
    char** array = emptyStack(head, p.N);
    printArray(array, p.N);
}
```

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Studio in Informatica, A.A. 2022-2023
Programmazione 1 e Laboratorio
Esempio di Prova di Esame Svolta
10 gennaio 2023

Descrizione del programma

Scrivere un programma in C che:

- **A** prenda in input come argomenti da linea di comando tre interi positivi “n”, “m” e “p”. Il programma deve controllare che i parametri inseriti sono nel numero corretto e che si tratti di interi positivi, stampando errori su standard output e terminando il programma qualora i parametri non dovessero rispettare i formati previsti;
- **B** definisca un array tridimensionale di interi “A” di dimensioni “n x m x p” in allocazione dinamica e lo inizializza con n*m*p valori inseriti da riga di comando separati da spazi;
- **C** inizializzi un array di puntatori a interi "B" di lunghezza "n", in modo che l’elemento i-esimo di “B”, “B[i]” punti al valore massimo contenuto nella matrice di dimensione “m x p” individuata da "A[i]".
- **D** inserisca gli elementi puntati da “B” in una pila;
- **E** scriva in un file di testo “out.txt” i valori puntati da “B” in senso inverso (effettuare operazioni di “pop” sulla pila).

Specifiche

Il programma potrà essere articolato in un unico file sorgente, ma dovrà contenere almeno le seguenti funzioni con opportuni parametri formali:

- **readInput**: funzione che permette di leggere gli input da riga di comando e gestisca opportunamente gli errori, come specificato nel testo;
- **initArray**: funzione che permette di dichiarare l’array A e inizializzarlo con i valori inseriti da tastiera;
- **initB**: funzione che permette di inizializzare l’array B come indicato nel testo;
- **initStack**: funzione che permette di inizializzare la pila dai valori di B;
- **saveStack**: funzione che permette di scrivere i valori della pila su file in senso inverso.

Note

Durata della prova: 120 minuti

Generazione di numeri pseudocasuali:

- Si consideri la seguente funzione get_random() per la generazione di numeri pseudo-casuali interi positivi (qualora necessaria):

```
// Scaricabile da: https://pastebin.com/f6eAKNQy
```

```
unsigned int get_random() {
    static unsigned int m_w = 123456;
    static unsigned int m_z = 789123;
    m_z = 36969 * (m_z & 65535) + (m_z >> 16);
    m_w = 18000 * (m_w & 65535) + (m_w >> 16);
    return (m_z << 16) + m_w;
}
```

- NB: Ai fini della generazione di numeri in virgola mobile, si faccia uso della costante UINT_MAX (<limits.h>) unitamente alla funzione get_random().

È VIETATO usare variabili globali.

Output di controllo

Dato il seguente file `input.txt`:

```
7 2 8 22 9 0 -2 3 -2 0 9 8 2 5 44 -3 92 -12 -52 9 28 27 26 4 0 9 2 7 -1 0
```

Ed eseguendo il programma con il seguente comando:

```
./main 5 2 3 < input.txt
```

Il programma scriverà il seguente contenuto sul file `out.txt`:

```
9
28
92
9
22
```

Soluzione

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

struct parameters {
    int n;
    int m;
    int p;
};

int get(char* s) {
    char **end = malloc(sizeof(char*));
    int num = (int) strtol(s, end, 0);
```

```
    if(!*end || num<=0) {
        fprintf(stderr, "Errore: i parametri inseriti devono essere dei numeri interi
positivi.\n");
        exit(-1);
    }

    return num;
}

struct parameters readInput(int argc, char * argv[]) {
    if(argc!=4) {
        fprintf(stderr, "Errore: il numero di parametri deve essere tre.\n");
        fprintf(stderr, "Uso corretto: %s n m p.\n", argv[0]);
        exit(-1);
    }

    struct parameters p = {get(argv[1]), get(argv[2]), get(argv[3])};

    return p;
}

int*** initArray(int n, int m, int p) {
    int ***A = calloc(n, sizeof(int**));
    for (int i =0; i<n; i++) {
        A[i] = calloc(m, sizeof(int*));
        for(int j=0; j<m; j++) {
            A[i][j] = calloc(p, sizeof(int));
            for(int k=0; k<p; k++) {
                scanf("%d", &A[i][j][k]);
            }
        }
    }

    return A;
}

int** initB(int ***A, int n, int m, int p) {
    int ** B = calloc(n, sizeof(int*));

    for(int i=0; i<n; i++) {
        int* max = &A[i][0][0];

        for(int j=0; j<m; j++) {
            for(int k=0; k<p; k++) {
                if(A[i][j][k]>*max)
                    max=&A[i][j][k];
            }
        }

        B[i] = max;
```

```
    }

    return B;
}

struct node {
    int data;
    struct node* next;
};

typedef struct node Node;

void push(Node **head, int x) {
    Node *tmp = *head;

    *head = malloc(sizeof(Node*));
    (*head)->next = tmp;
    (*head)->data = x;
}

bool empty(Node *head) {
    return head==NULL;
}

int pop(Node **head) {
    if(empty(*head))
        return -1;
    int out = (*head)->data;
    Node* tmp = *head;
    *head = (*head)->next;
    free(tmp);
    return out;
}

Node** initStack(int **B, int n) {
    Node** head= malloc(sizeof(Node*));
    *head = NULL;

    for(int i=0; i<n; i++) {
        push(head, *B[i]);
    }
    return head;
}

void saveStack(Node **head) {
    FILE *f = fopen("out.txt", "w");

    while(!empty(*head)) {
        int x = pop(head);
        fprintf(f, "%d\n", x);
```

```
    }
}

int main(int argc, char * argv[]) {
    struct parameters p = readInput(argc, argv);
    int ***A = initArray(p.n, p.m, p.p);
    int **B = initB(A, p.n, p.m, p.p);

    Node** head = initStack(B, p.n);
    saveStack(head);
}
```