Corso: Tecniche di programmazione
Quiz: Tecniche di Programmazione - I appello del 30/06/2025

PORTALE ESAMI

	Marco Cellini **s341045**
Iniziato	30 giugno 2025, 11:02
Stato	Completato
Terminato	30 giugno 2025, 12:33
Tempo impiegato	1 ora 31 min.
Valutazione	Non ancora valutato
Riepilogo del tentativo	i 1 2 3 4 5 6 i

Informazione

Prova scritta di esame di Tecniche di Programmazione:
Primo appello 2024/2025

L'esame consta di 6 domande, per un massimo di 33 punti totali, ripartite in due parti:

- 3 domande di teoria (15 punti complessivi): domande 1, 2, 3
- 3 domande di programmazione (18 punti complessivi): domande 4, 5, 6

L'esame è superato se sono vere (tutte e tre) le seguenti condizioni:

- Voto complessivo >=18
- Voto complessivo per le domande 1,2,3 (teoria) >= 5
- Voto complessivo per le domande 4,5 (programmazione) >= 5

Nelle domande di programmazione, la complessità della soluzione proposta può essere oggetto di valutazione: se ad esempio un problema avesse una soluzione lineare, un algoritmo quadratico potrebbe essere valutato leggermente meno dell'algoritmo lineare. Si può scorrere tra le domande avanti e indietro. Non viene quindi imposta una sequenza. Le risposte alle domande di teoria e il codice C per le domande di programmazione vanno scritti negli spazi riservati, nell'editor di testo, sotto ogni domanda. Si tratta di un semplice editor di testo, senza alcuna funzionalità IDE (nessun rientro automatico, nessun run-debug, nessun completamento automatico, ecc.). Attenzione a evitare l'uso del tabulatore (tasto TAB).

Dopo la prova scritta, si riceverà via e-mail il file .pdf dell'esame. Affinché questo sia venga valutato, è necessario caricare nella sezione Elaborati del Portale della Didattica una relazione con auto-correzione, seguendo le istruzioni riportate nel Portale della Didattica. La scadenza (inderogabile) è il **2 luglio 2025, 23:59**.

Domanda 1
Completo
Punteggio max.: 5,00

ESERCIZIO 1 (5 punti)

Si ordini in maniera ascendente mediante radix sort il seguente vettore A di interi su al massimo 3 cifre in base 16:

200 63 3A7 52B 2 67A 7A3 23B 11

Domande & risposte:

- Quali sono le cifre della base 16?

- Si enunci brevemente la condizione che rende stabile il radix sort:

- passo 1 colonna di peso 16^0: riportare come sequenza di interi il contenuto iniziale del vettore C delle occorrenze semplici e delle occorrenze multiple

- passo 2 colonna di peso 16^1: riportare come sequenza di interi il contenuto del vettore A dopo l'esecuzione del passo di counting sort

_

- se al posto del counting sort si usasse l'insertion sort, quale sarebbe la complessità asintotica di caso peggiore del radix sort se si devono ordinare N interi in base 16 su d cifre?

o Theta (dN)

o O(16N^2)

o O(dN^2)

o Theta (dN^2)

Breve giustificazione della risposta:

Domande & risposte:

- Quali sono le cifre della base 16?

0 1 2 3 4 5 6 7 8 9 A B C D E F

- Si enunci brevemente la condizione che rende stabile il radix sort:

Il radix sort è stabile se l'algoritmo di ordinamento che si applica per l'ordinamento delle singole cifre è stabile, normalemente con il radix sort viene usato il counting sort il quale è stabile.

- passo 1 colonna di peso 16^0: riportare come sequenza di interi il contenuto iniziale del vettore C delle occorrenze semplici e delle occorrenze multiple

occorrenze semplici: 1 1 1 2 0 0 0 1 0 0 1 2 0 0 0 0

occorrenze multiple: 1 2 3 5 5 5 5 6 6 6 7 9 9 9 9 9

- passo 2 colonna di peso 16^1: riportare come sequenza di interi il contenuto del vettore A dopo l'esecuzione del passo di counting sort

occorrenze semplici: 2 1 1 1 0 0 1 1 0 0 2 0 0 0 0 0

occorrenze multiple: 2 3 4 5 5 5 6 7 7 7 9 9 9 9 9 9

- se al posto del counting sort si usasse l'insertion sort, quale sarebbe la complessità asintotica di caso peggiore del radix sort se si devono ordinare N interi in base 16 su d cifre?

o Theta (dN)

o O(16N^2)

o O(dN^2)

Xo Theta (dN^2)

Breve giustificazione della risposta:

L'insertion sort nel caso peggiore ha complessità N^2 e il radix lo applica d volte quante sono le cifre da ordinare quindi, nel caso peggiore --> Theta(dN^2)

Domanda 2

Completo

Punteggio max.: 5,00

ESERCIZIO 2 (5 punti)

Siano date le seguenti funzioni:

```
int unknown(int arr[], int n) {
  int i, j;
  for (i = 0; i < n-1; i++) {
     for (j = i + 1; j < n; j++) {
        if (arr[i] == arr[j])
          return 1;
     }
  }
  return 0;
}
```

```
int unknown1(int *arr[], int n) {
  int i, k=1;
  for (i = 0; i < n-1; i++) {
     if (arr[i] != arr[i+1])
        k++;
  }
  return k;
}
```

Domande & risposte:

Si determini la complessità asintotica di caso **peggiore** della funzione unknown:

☐ O(n) ☐ O(n^2) ☐ O(nlogn)

☐ Theta (n) ☐ Theta (n^2) ☐ Theta (nlogn)

Breve spiegazione della risposta:

Si determini la complessità asintotica di caso **medio** della funzione unknown:

☐ O(n) ☐ O(n^2) ☐ O(nlogn)

☐ Theta (n) ☐ Theta (n^2) ☐ Theta (nlogn)

Breve spiegazione della risposta:

Le complessità asintotiche di caso **peggiore** e di caso **medio** funzione unknown1 sono **uguali** o **diverse**?

☐ uguali ☐ diverse

Breve spiegazione della risposta:

Cosa calcola la funzione unknown?

Cosa calcola la funzione unknown1? Cosa si deve assumere del vettore arr affinché la funzione operi correttamente?

Domande & risposte:

Si determini la complessità asintotica di caso **peggiore** della funzione unknown:

☐ O(n) ☐ O(n^2) ☐ O(nlogn)

☐ Theta (n) X☐ Theta (n^2) ☐ Theta (nlogn)

Breve spiegazione della risposta:

Nel caso peggiore, ovvero che la condizione di uscita non strutturata non sia mai vera allora il ciclo esterno ed interno vengono eseguiti completamente avendo una complessità uguale a Theta(n^2)

Si determini la complessità asintotica di caso **medio** della funzione unknown:

☐ O(n)	X☐ O(n^2)	☐ O(nlogn)
☐ Theta (n)	☐ Theta (n^2)	☐ Theta (nlogn)

Breve spiegazione della risposta:

Nel caso medio vi è una probabilità che la condizione di uscita sia vera e quindi la complessità diventa O(n^2).

Le complessità asintotiche di caso **peggiore** e di caso **medio** funzione unknown1 sono **uguali** o **diverse**?

X☐ uguali ☐ diverse

Breve spiegazione della risposta:

Non c'è nessuna condizione che anticipa la terminazione del ciclo, esso termina solo quando sarà arrivato fino alla penultima cella del vettore, complessità Theta di N.

Cosa calcola la funzione unknown?

Verifica se ci sono numeri duplicati nell'array, se vero ritorna 1 se no 0.

Cosa calcola la funzione unknown1? Cosa si deve assumere del vettore arr affinché la funzione operi correttamente?

Conta quanti puntatori univoci ci sono nel vettore di puntatori arr, affinche la funzioni operi correttamente arr vede essere un vettore di puntatori o una matrice ed n deve essere il numero di righe della matrice o il numeri di puntatori presenti.

Domanda 3

Completo

Punteggio max.: 5,00

Assumendo una memoria da 4 GB byte-addressable allineata con celle di 1 Byte e parole di 4 Byte, una codifica per gli interi su 32 bit e puntatori che occupano 8 Byte, analizzare il seguente spezzone di codice:

```
typedef enum {ATTESA, ATTIVO, ERRORE} statoComponente;

typedef struct {
    char id [3]; float pressione, velocita; char modalita; statoComponente stato;
} datiComponente;

typedef struct datiComponente2 * t_componente;
typedef struct {
    char id [3]; char modalita; float pressione; statoComponente stato;
    float velocita; t_componente pros;
} datiComponente2;

datiComponente comp, compArray[2], *tPtr = compArray;
t_componente comp2;
```

e fornire lo spazio occupato in memoria (in Byte) dalle seguenti espressioni, motivando la risposta:

ESPRESSIONE	DIMENSIONE (BYTE)	MOTIVAZIONE
ERRORE		
comp		
comp.id		
comp2		
comp2.prossimo		
tPtr		
*tPtr		
compArray		
compArray+1		
*(compArray+1)		

ESPRESSIONE		DIMENSIONE (BYTE)	MOTIVAZIONE
ERRORE	|	4	gli enum sono interi con valori prefissati >= 0 incrementali
comp	|	20	struct con problema di padding, si sprecano 4 byte non mettendo uno dopo l'altro i char
comp.id	|	3	array di 3 caratteri da 1 byte l'uno
comp2	|	8	puntatore a struct
comp2.prossimo	|	8	puntatore a struct
tPtr	|	8	puntatore a struct
*tPtr	|	20	compArray[0] struct di datiComponente
compArray	|	40	2 struct di datiComponente

compArray+1	|	8	puntatore a compArray[1]
*(compArray+1)	|	20	compArray[1] struct di datiComponente

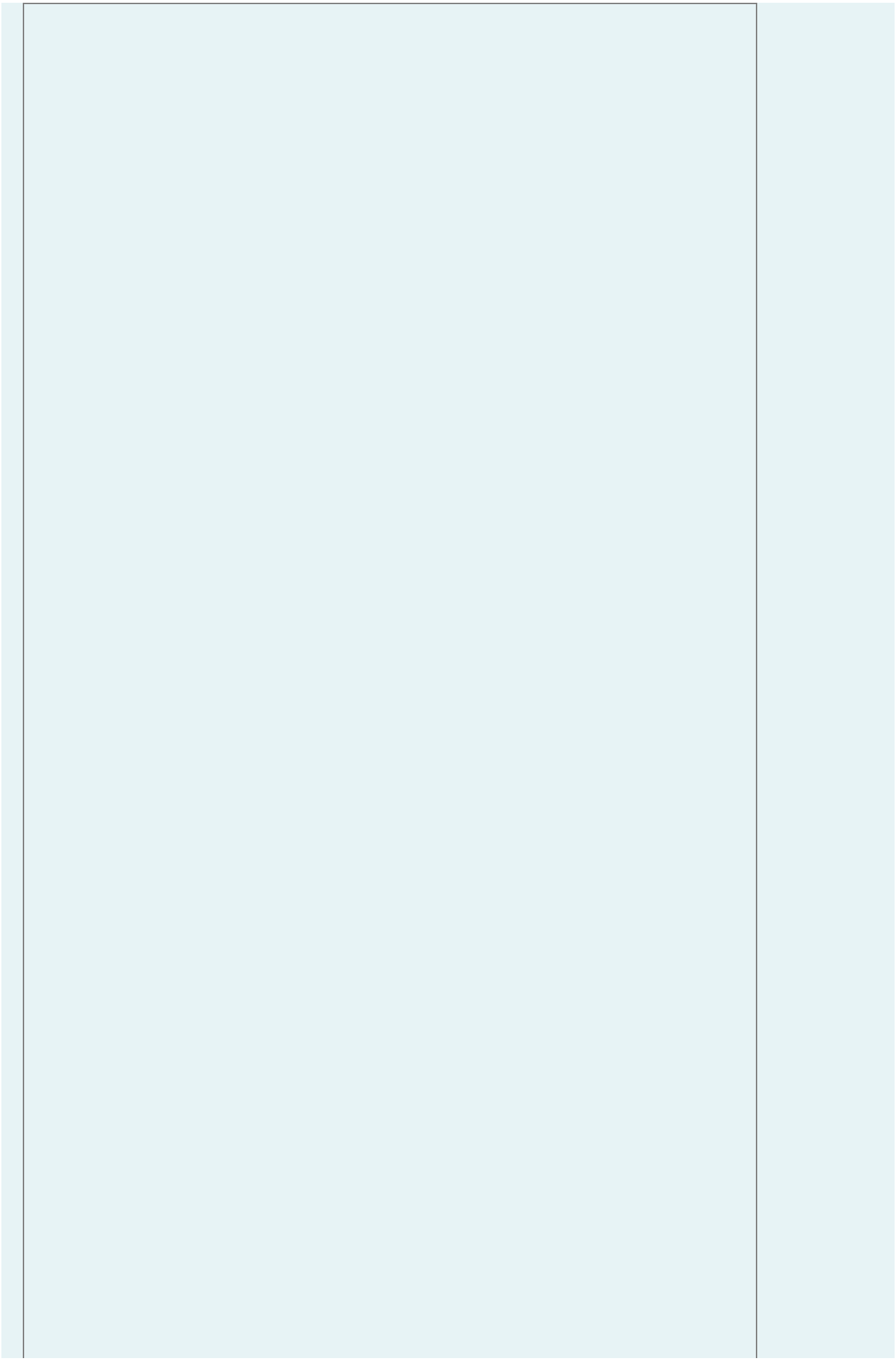
Si prega di non rimuovere la colonna con i caratteri |.

Domanda 4

Completo

Punteggio max.: 4,00

ESERCIZIO 4 (4 punti)



Un programma legge da un file un elenco di parole, ciascuna lunga al massimo 15 caratteri. Il nome del file viene passato come primo parametro da riga di comando. Il programma invoca la funzione **readFile**, che conta quante parole ci sono per ogni lunghezza (da 1 a 15 caratteri), memorizzando i risultati in un vettore attraverso la corrispondenza indice-valore. Il programma chiama poi una funzione **printHistogram**, che stampa un istogramma: per ogni lunghezza (da 1 a 15) stampa un numero di asterischi '*' pari al numero di parole di quella lunghezza. Tale funzione inoltre individua la lunghezza di parola più frequente e la restituisce come risultato. Completa il seguente codice, inserendo le espressioni corrispondenti alle caselle numerate nella tabella sottostante:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LUNGH 10

void readFile( 1 , 2 ){
   char input[ 3 ];
   FILE * fp = fopen(filename, "r");
   while(fscanf( 4 ) == 1){
      cc[ 5 -1]++;
}
   fclose(fp);
}
int printHistogram(int * cc){
   int i, j, max = 0;
   for(i = 0; i < MAX_LUNGH; i++) {
      printf("%4d", i+1);
      for (j = 0; j < 6 ; j++) printf("*"); printf("\n");
      if(cc[i] > cc[max]) 7 ;
   }
   return max +1;
}
int main(int argc, char * argv[]) {
   int cc[MAX_LUNGH] = {0};
   char parola[MAX_LUNGH];
   readFile( 8 , cc);
   printf("lunghezza con piu' parole: %d\n", printHistogram(cc));
}
```

<1>		<5>	
<2>		<6>	
<3>		<7>	
<4>		<8>	

<1>	char *filename	<5>	strlen(input)
<2>	int cc[]	<6>	i
<3>	MAX_LUNGH	<7>	max = i
<4>	fp, "%s", input	<8>	argv[1]

Domanda 5

Completo

Punteggio max.: 6,00

ESERCIZIO 5 (6 punti)

Per ciascuno dei programmi in tabella, dire se il codice può essere compilato ed eseguito: in caso affermativo, riportare cosa verrebbe stampato a video, altrimenti spiegare quale problema non consente di compilare e/o eseguire. Assumere un’architettura a 32 bit Little Endian, in cui gli interi sono codificati su 4Byte.

1

```
#include <stdio.h>
int main(){
  char * vet = "one";
  for (int i = 0; vet[i]!='\0'; i++)
      printf("%c", *(vet+i) +1);
  return 0;
}
```

2

```
#include <stdio.h>
int main(){
  char * vet = "one";
  for (int i = 0; vet[i]!='\0'; i++)
      printf("%s", vet+i);
  return 0;
}
```

3

```
#include <stdio.h>
int main(){
   int val = 0x88112233;
   int *p = &val;
   printf("%x", *p);
   return 0;
}
```

4

```
#include <stdio.h>
int main(){
   int val = 0x88112233;
   char *p = (char *)&val;
   printf("%x", *p);
   return 0;
}
```

5

```
#include <stdio.h>
int main(){
   int val = 0x44112233;
   char *p = (char *)&val;
   for(int i = sizeof(val)-1; i>= 0; i--)
      printf("%x", p[i]);
   return 0;
}
```

6

```
#include <stdio.h>
int main(){
   int val = 0x44112233;
   char *p = (char *)&val;
   for(int i = sizeof(val)-1; i>= 0; i--)
      printf("%x", *(val + i));
   return 0;
}
```

	ESEGUE? (Sì/NO)	**SE Sì, SCRIVERE COSA VIENE MOSTRATO A VIDEO SE NO, SPIEGARE PERCHE'**
N.1		
N.2		
N.3		
N.4		
N.5		
N.6		

	ESEGUE? (SI/NO)	**SE Sì, SCRIVERE COSA VIENE MOSTRATO A VIDEO SE NO, SPIEGARE PERCHE'**
N.1	SI	pof
N.2	SI	onenee
N.3	SI	88112233
N.4	SI	3

N.5	SI	44112233
N.6	NO	val non è un array con *(val + i) stiamo controllando la i-esima cella del vettore di interi

Domanda 6

Completo

Punteggio max.: 8,00

ESERCIZIO 6 (8 punti)

In un programma in C, un bambino e una attività svolta sono rappresentati dalle seguenti strutture:

```
typedef struct {
  char id[7];            // codice identificativo del bambino
  attivita* elenco;      // puntatore all'elenco delle attività svolte
  int n;                 // numero di attività svolte (elementi in elenco)
} bambino;

typedef struct {
  char nome[30];   // nome dell'attività
  int ingressi;    // numero di volte in cui il bambino ha partecipato all'attività
} attivita;
```

Si supponga che i dati relativi a un gruppo di 100 bambini e alle loro attività siano già stati letti da un file e memorizzati nell'array: bambino b[100];

Si assuma, per semplicità, che ogni bambino abbia svolto almeno un'attività.

La funzione main invoca le seguenti funzioni:

- funzione **raggruppaBambini**: riceve in ingresso l'array di bambini b, la sua dimensione n, un vettore di interi conteggi di dimensione 10. La funzione deve raggruppare i bambini in base al numero di attività (non il numero totale di ingressi). Il gruppo 0 considera i bambini con meno di 5 attività, il gruppo 1 da 5 a 9 attività, il gruppo 2 da 10 a 14 attività e così via. L'ultimo gruppo (il 9) raggruppa tutti i bambini con più di 45 attività. La funzione salva il numero di bambini per ciascun gruppo nel vettore conteggi, dove ogni indice rappresenta un gruppo. Il prototipo della funzione deve essere:
void raggruppaBambini(bambino b[], int n, int conteggi[10]);

- funzione **attivitaPreferite**: riceve in ingresso l'array di bambini b, la sua dimensione n, un array preferite di puntatori a attività. Per ogni bambino, la funzione individua l'attività alla quale ha partecipato più volte (cioè con il valore di ingressi più alto) e ne memorizza l'indirizzo in preferite[i], per il bambino di indice i. Si può assumere che non ci siano parità nel numero massimo di ingressi per ciascun bambino. Prototipo della funzione:
void attivitaPreferite(bambino b[], int n, attività* preferite[]);

- funzione **piuAttivi**: riceve in ingresso l'array di bambini b, la sua dimensione n, un intero min, e un vettore di puntatori selezionati. La funzione seleziona i bambini che hanno svolto almeno min attività (senza considerare il numero di ingressi per attività), caricando i loro puntatori nel vettore di puntatori ricevuto come parametro. La funzione ritorna il numero di bambini selezionati. Prototipo della funzione:
void piuAttivi(bambino b[], int n, int min, bambino* selezionati[]);

Implementare le funzioni **raggruppaBambini**, **attivitaPreferite**, e **piuAttivi**, rispettando i prototipi forniti.

Attenzione! Si consiglia l'uso degli spazi al posto delle tabulazioni per l'indentazione del codice, dal momento che il carattere TAB viene utilizzato per la navigazione della pagina da parte della piattaforma.
Sono possibili, se necessarie o utili, altre funzioni oltre a quella/e richiesta/e

```
viod raggruppaBambini(bambino b[], int n, int conteggi[10]) {    int i, n_attivita;    for (i = 0; i < len; i++) {        n_attivita = b[i].n;        if (n_attivita >= 45)            conteggi[9]++;
else             conteggi[n_attivita / 5]++;    }}

void attivitaPreferite(bambino b[], int n, attivita *preferite[]) {    int i, j max, n_attivita;    for (i = 0; i < n; i++) {         max = 0;         n_attivita = b[i].n;
         for (j = 0; j < n_attivita; j++)             if (b[i].elenco[j].ingressi > b[i].elenco[max].ingressi)                 max = j;         preferite[i] = &b[i].elenco[max];     }}
int piuAttivi(bambino b[], int n, int min, bambino *selezionati[]) {
int i, n_attivita, selected = 0;
for (i = 0; i < n; i++) {
n_attivita = b[i].n;
if (n_attivita >= min)
selezionati[selected++] = &b[i];
} return selected; }
```

Informazione

#include <stdio.h>

FILE *fopen(char *filename, char * mode) – Opening a file (mode: “r” reading – “w” writing – “a” append)

FILE *freopen(char *filename, char * mode, FILE *file_pointer) – Reassign a file pointer to a different file

int fclose(FILE *file_pointer) – Closing a file

int feof(FILE *file_pointer) – Checks if end-of-file has been reached.

int fflush(FILE *file_pointer) - Empties file’s buffer.

int getchar(void) – Reads a character from “stdin" (keyboard)

int fgetc(FILE *file_pointer) – Gets a character from a file

char *gets(char *buffer) - Reads a line from “stdin" (keyboard)

char *fgets(char *string, int maxchar, FILE *file_pointer) - Reads a line from a file

int printf(char *format _string, ...) – Writes formatted output on "stdout" (screen)

int fprintf(FILE *file_pointer, char *format_string, ...) – Writes formatted output on a file.

int sprintf(char *string, char *format_string, ...) – Writes formatted output on a string.

int fputc(int c, FILE *file_pointer) – Writes a character on a file.

int putchar(int c) – Writes a character on “stdout” (screen).

int puts(char *string) - Writes a string on “stdout” (screen).

int fputs(char *string, FILE *file_pointer) - Writes a string on a file.

int scanf(char *format_string, args) – Reads formatted input from “stdin” (keyboard)

int fscanf(FILE *file_pointer, char *format string, args) – Reads formatted input from a file.

int sscanf(char *buffer, char *format_string, args) – Reads formatted input from a string.

EOF – end of file (costant with negative value)

NULL - null pointer (value 0)

#include <stdlib.h>

double atof(char *string) – Converts a string into a floating point value.

int atoi(char *string) – Converts a string into an integer value.

int atol(char *string) – Converts a string into a long integer value.

void exit(int val) – Terminates the program returning the value ‘val’.

EXIT_FAILURE – constant highlighting the unsuccessful termination of the program with exit(); non zero value.

EXIT_SUCCESS - constant highlighting the successful termination of the program with exit(); zero value.

#include <string.h>

char *strcpy(char *s1, char *s2) - Copies s2 in s1. Returns s1

char *strncpy(char *s1, char *s2, size_t n) – Copies the first "n" characters of s2 in s1. Returns s1.

int strcmp(char *s1, char *s2) - Compares s1 and s2 to determine the alphabetical order (<0, s1 precedes s2, 0 equal, >0 s1 follows s2)

int strncmp(char *s1, char *s2, size_t n) – Compares the first "n" characters of two strings.

int strlen(char *string) – Determines the length of a string.

char *strcat(char *s1, char *s2, size_t n) - Links s2 to s1. Returns s1

char *strncat(char *s1, char *s2, size_t n) - Links "n" characters of s2 to s1. Returns s1

char *strchr(char *string, int c) – Finds the first occurrence of the character ‘c’ in string; returns a pointer to the first occurrence of c in s, NULL if not present.

char *strrchr(char *string, int c) – Finds the last occurrence of the character ‘c’ in string.

char* strstr(char* s, char* t) – Returns a pointer to the first occurrence of t in s. returns NULL if not present.

char* strtok(char* s, const char* t) – Decomposes s in tokens, the characters that limit the tokens are contained in t. returns the pointer to the token (NULL if any is found). At the first call the string to be decomposed is s and the characters delimiting the various tokens in t. To operate on the same string, at following calls NULL has to be passed instead of s.

#include <ctype.h>

int isalnum(int c) – True if ‘c’ is alphanumeric.

int isalpha(int c) – True if ‘c’ is an alphabetic charater.

int iscntrl(int c) – True if ‘c’ is a control character.

int isdigit(int c) - True if ‘c’ is a decimal digit.

int islower(int c) - True if ‘c’ is lowercase.

int isprint(int c) - True if ‘c’ is a printable character.

int ispunct (int c) - True if ‘c’ is a punctuation character.

int isspace(int c) - True if ‘c’ is a space character.

int isupper(int c) - True if ‘c’ is uppercase.

tolower(int c) – Converts ‘c’ to lowercase.

int toupper(int c) – Convert ‘c’ to uppercase.

#include <math.h>

int abs (int n) – integer absolute value

long labs(long n) – long absolute value

double fabs (double x) – absolute value of x

double acos(double x) - arccosine

double asin(double x) - arcsin

double atan(double x) - artangent

double atan2(double y, double x) – arctangent of y/x.

double ceil(double x) – round up value of x.

double floor(double x) – round down value of x.

double cos(double x) – cos (x in radians)

double sin(double x) – sin (x in radians)

double tan(double x) – tan (x in radians)

double cosh(double x) – hyperbolic cosine

double sinh(double x) – hyperbolic sin

double tanh(double x) – hyperbolic tangent

double exp(double x) - e^x

double log(double x) - log(x).

double log10 (double x) – logarithm base 10

double pow (double x, double y) - x^y

int rand (void) – random integer between 0 and RND_MAX.

int random(int max_num) – random value between 0 and max_num.

void srand(unsigned seed) – initialize the sequence of random values

double sqrt(double x) – square root

#include <limits.h>

INT_MAX – Maximum value that can be represented by int variable.

INT_MIN – Minimum value that can be represented by int variable.

LONG_MAX - Maximum value that can be represented by long variable.

LONG_MIN - Minimum value that can be represented by long variable.

#include <float.h>

FLT_MAX, DBL_MAX - Maximum value that can be represented by float (or double) variable.

FLT_MIN, DBL_MIN - Minimum value that can be represented by float (or double) variable.