Ascolta

Nota: fa parte del compito includere anche un makefile che, invocato dalla linea di comando con il comando *make*, crea gli eseguibili di nome *esercizio1*, *esercizio2* (ed eventualmente *esercizio3*) che risolvono i singoli esercizi. A meno che non sia indicato esplicitamente, il tempo a disposizione per il compito è di due ore.

Durante l'esame gli studenti potranno accedere al materiale di questo corso DIR.

Compito dell'8/2/22

1a) Si utilizzi la struttura

```
typedef struct el {
  int val;
  struct el *next;
} intero;
```

per rappresentare un elemento di una lista di interi. Scrivere una funzione

```
intero *crea_lista(char *s)
```

che data una stringa *s* contenente degli interi separati dal carattere `:` restituisce una lista contenente tutti gli interi che appaiono in *s* mantenendo l'ordine in cui compaiono. Ad esempio, se la stringa contiene "123:45:-5:32:13" la lista risultante deve contenere gli interi [123, 45, -5, 32, 13]. Suggerimento: usare *strtok* per fare il parsing di *s*.

1b) Scrivere una funzione

```
intero *somma(intero *lis1, intero *lis2)
```

che date due liste di interi costruisce e restituisce una terza lista i cui elementi sono la somma degli elementi in *lis1* e *lis2*. La lunghezza della terza lista deve essere pari al minimo delle lunghezze di *lis1* e *lis2*. Ad esempio, se lis1 contiene [23, 4, 7, -2] e lis2 contiene [3, 7, 5] la lista risultato deve contenere [26, 11, 12]

1c) Scrivere una funzione *main* che

- verifica che sia *argc==3*
- crea due liste *lis1* e *lis2* invocando *crea_lista* su *argv[1]* e *argv[2]*
- stampa le liste *lis1* e *lis2*
- calcola *lis3 = somma(lis1, lis2)*
- stampa *lis3*
- dealloca tutte le liste (verificare con valgrind) e termina

---

2a) Si utilizzi la struttura

```
typedef struct el {
  char *s;
  struct el *next;
} stringa;
```

per rappresentare un elemento di una lista di stringhe. si scriva una funzione

stringa *leggilinee(FILE *f)

che legge, mediante `getline`, le linee contenute nel file `f` e restituisce una lista contenenti tali linee nello stesso ordine in cui appaiono nel file `f`.

2b) Si scriva una funzione

```
stringa *elimina(stringa *lis, bool (*fun)(char *))
```

che cancella dalla lista *lis* e dealloca tutte le stringhe per cui la funzione *fun* vale `true`. La funzione *elimina* deve restituire il puntatore alla primo elemento della lista modificata.

2c) Scrivere una funzione *main* che

?

- verifica che sia *argc==3*

- crea la lista *lis* invocando *leggilinee* sul file di nome *argv[1]*
- cancella dal *lis* le stringhe di lunghezza pari invocando la funzione *elimina* passando *lis* e una funzione (che dovete scrivere) che restituisce `true` se gli viene passata una stringa di lunghezza pari. Ricordate che le stringhe lette dal file contengono un carattere `\n` finale che viene contato nella lunghezza
- stampa nel file di nome *argv[2]* le linee rimaste nella lista *lis*.
- dealloca tutta la memoria utilizzata (verificare con valgrind) e termina

Ad esempio, passando al programma il file contenente le linee

```
sei
due
otto
tre
dodici
```

deve essere creato il file

```
otto
dodici
```

in quanto le linee contenenti `sei`, `due`, `tre` hanno lunghezza pari (ricordate che c'e anche lo `\n`) e quindi vengono cancellate al punto *2b*.

Tempo a disposizione: 2 ore

Compito del 21/9/21

1a) Si utilizzi la struttura

```
typedef struct el {
  int val;
  struct el *next;
} intero;
```

per rappresentare un elemento di una lista di interi. Scrivere una funzione

```
intero *crea_lista(FILE *f)
```

che dato file binario `f` contenente interi a 32 bit, costruisce e restituisce una lista contenente gli interi presenti nel file, mantenendo il loro ordine (cioè gli interi nella lista devono essere nello stesso ordine con cui appaiono nel file).

1b) Si scriva una funzione

```
intero *cancella_coppie(intero *lis, int n)
```

che data una lista di interi `lis` cancella tutte le coppie di interi consecutivi la cui somma è `n`. La funzione deve restituire il puntatore al primo elemento della lista modificata. Ad esempio, se `lis` è uguale a [4 6 7 2 8 3 11 -5 15] la chiamata `cancella_coppie(lis,10)` dovrebbe restituire la lista [7 3 11].

1c) Scrivere una funzione *main* che

- verifica che sia *argc==2*
- crea una lista `lis` invocando `crea_lista` sul file *argv[1]*
- stampa sul terminale il contenuto della lista `lis`
- invoca la funzione `cancella_coppie` sulla lista `lis` con `n=10` e visualizza il risultato sul terminale
- visualizza sul terminale la somma degli elementi nella lista risultato
- dealloca tutta la memoria utilizzata (verificare con valgrind) e termina

Per testare il programma potete usare i file in formato binario `sommaXX` disponibili su DIR. Il valore nel nome del file è la somma degli elementi che si ottiene invocando `cancella_coppie` con `n=10`. Ricordo che per visualizzare gli interi contenuti in un file binario potete usare dalla linea di comando l'istruzione `od -An -td4 nomefile`.

2a) Si consideri la struct

```
typedef struct {
  char *nome;
  char *professione;
  int reddito;
} professionista;
```

che permette di rappresentare un oggetto di tipo `professionista`. Scrivere una funzione

professionista *leggi_pro(FILE *f, int *n)

che dato un puntatore ad un file di testo contenente i dati di un gruppo di professionisti, crea e restituisce un array di oggetti di tipo `professionista`. La funzione deve restituire il puntatore al primo elemento dell'array creato e, attraverso il parametro `n`, il numero di elementi dell'array.

Nel file di testo `f` ogni riga contiene i dati una singola persona separati dal carattere `;`. Ad esempio un possibile contenuto potrebbe essere:

```
Andrea Costa;Architetto;2100
Claudia Neri;Responsabile Sicurezza;2800
George Niang;Sviluppatore Senior;2600
Elena Zukova;Chirurgo Plastico;3400
Andrew Lee;Architetto;1800
Federico Di Meglio;Chirurgo Plastico;3250
```

Per la lettura del file si consiglia di utilizzare `getline` (per leggere una singola linea) e `strtok` (per ottenere i singoli campi di una linea)

2b) Si scriva una funzione

```
void stampa_pro(professionista* p, FILE *f)
```

per stampare i dati del professionista `*p` nel file *f* utilizzando il seguente formato:

```
--- begin
Nome: Andrea Costa
Professione: Architetto
Reddito medio mensile: 2100
--- end
```

2c) Scrivere una funzione

```
void ordina(professionista *a, int n)
```

che dato un array `a` contenente `n` elementi li ordina lessicograficamente rispetto alla professione, in caso di elementi con la stessa professione questi devono essere ordinati per reddito crescente. Nel caso dell'esempio visto sopra, al termine dell'ordinamento l'array deve contenere gli elementi nell'ordine:

```
Andrew Lee;Architetto;1800
Andrea Costa;Architetto;2100
Federico Di Meglio;Chirurgo Plastico;3250
Elena Zukova;Chirurgo Plastico;3400
Claudia Neri;Responsabile Sicurezza;2800
George Niang;Sviluppatore Senior;2600
```

L'ordinamento **deve** essere fatto utilizzando la funzione di libreria **qsort**.

2d) Si scriva poi una funzione *main* che

- verifica che sia `argc==3`
- ottiene un array `a` invocando la funzione `leggi_pro` passando il file di nome `argv[1]`
- invoca la funzione `ordina` del punto **2c** sull'array `a`
- scrive gli elementi dell'array `a` riordinato sul file di nome `argv[2]` invocando la funzione `stampa_pro` del punto **2b**
- dealloca tutta la memoria utilizzata (verificare con valgrind) e termina

Per testare il programma potete usare i due file `lista1.txt` e `lista2.txt` disponibili su DIR.

Compito del 14/7/21

1a) Si consideri la struct

```
typedef struct inte {
  int val;
  struct inte *next;
} intero;
```

che permette di rappresentare una lista di interi. Si scriva una funzione

```
intero *crea_lista_ordinata(char *s, int n)
```

che data una stringa `s` contenente solamente caratteri 0 e 1, la spezza in blocchi di `n` caratteri e converte ogni blocco in un intero considerandolo come sequenza binaria (si noti che l'ultimo blocco può avere meno di `n` caratteri). Ad esempio se `s` è uguale a "00111000001101" e `n` è uguale a 3, i blocchi risultanti sono "001", "110", "000", "011", "01" e si ottengono di conseguenza gli interi 1,6,0,3,1. La funzione deve restituire una lista contenente questi interi ordinati dal più grande al più piccolo; nell'esempio di prima deve essere restituita la lista:

```
6 -> 3 -> 1 -> 1 -> 0
```

1b) Si scriva poi una funzione *main* che

- verifica che sia `argc==5`
- crea una lista `lis1` chiamando la funzione del punto **1a** passando come argomenti `argv[1]` e `atoi(argv[2])`
- crea una lista `lis2` chiamando la funzione del punto **1a** passando come argomenti `argv[3]` e `atoi(argv[4])`
- visualizza le liste `lis1` e `lis2` su sdterr
- calcola la lista `lis3` contenente gli elementi di `lis1` e `lis2` ordinati in ordine decrescente (quindi `lis3` ha una lunghezza pari alla somma delle lunghezze di `lis1` e `lis2` ed è ordinata in maniera decrescente). Per il calcolo di `lis3` si scriva una funzione a parte utilizzando una strategia a vostra scelta.
- dealloca tutta la memoria utilizzata (verificare con valgrind) e termina

---

2a) Si scriva una funzione

```
int *contiene(char *a[], int n, char c, int *num)
```

che dato un array `a[]` contenente `n` stringhe alloca e restituisce un array contenente le posizioni delle stringhe di `a[]` che contengono il carattere `c`. Ad esempio se

```
a = ["turchia", "svizzera", "galles", "austria", "belgio", "spagna", "inghilterra"]
```

e il parametro `c` contiene il carattere `'g'` la funzione deve restituire l'array [2, 4, 5, 6]. Se invece `c` contiene il carattere `'t'` la funzione deve restituire l'array [0,3,6]. La funzione deve scrivere in `num` il numero di elementi dell'array di interi e restituire con return il puntatore al suo primo elemento. *Nota* è ammesso fare due passate sull'array *a*.

2b) Si scriva poi una funzione *main* che

- verifica che sia `argc==2`
- apre il file di nome `argv[1]` e legge con fscanf tutte le stringhe in esso contenute mettendone una copia in un array *a[]* creato appositamente
- invoca la funzione del punto **2a** sull'array `a[]` passando come terzo argomento il carattere `r`; l'array di interi ottenuto deve essere scritto in formato binario nel file `r.bin`
- invoca la funzione del punto **2a** sull'array `a[]` passando come terzo argomento il carattere `s`; l'array di interi ottenuto deve essere scritto in formato binario nel file `s.bin`
- visualizza su *stderr* le posizioni delle stringhe `a[]` che contengono sia la lettera `r` che la stringa `s`. Scegliete voi il metodo con cui calcolare tali posizioni. Nel caso dell'esempio precedente, dovrebbero essere visualizzate su *stderr* le posizioni `1` e `3`
- dealloca tutta la memoria utilizzata (verificare con valgrind) e termina

Si raccomanda di testare il programma con più esempi, e in particolare con qualche esempio in cui non compaiono le lettere `r` o `s` (in tal caso i file `r.bin` e `s.bin` devono essere creati vuoti), oppure le lettere `r` e `s` compaiono ma mai nella stessa stringa (in tal caso non deve essere visualizzato nulla su *stderr*).

Per visualizzare sul terminale gli interi contenuti in un file binario, potete usare il comando `od -An -t d4 <nomefile>`.

Tempo a disposizione: 2 ore

Compito del 23/6/21

1a) Si consideri la struct

```
typedef struct {
  int primo;
  int secondo;
} coppia;
```

che permette di rappresentare una coppia di interi. Scrivere una funzione

```
coppia *leggi_coppie(FILE *f, int *n)
```

che dato un puntatore ad un file binario contenente un numero pari di interi a 32 bit, crea e restituisce un array di coppie utilizzando gli interi del file (i primi due interi formano la prima coppia, gli interi 3 e 4 la seconda coppia, e così via). La funzione deve restituire il puntatore al primo elemento dell'array creato e, attraverso il parametro, *n* il numero di coppie presenti nell'array.

1b) Si consideri la funzione

```
int somma(coppia c) {
   return (c.secondo + c.primo);
}
```

che data una coppia restituisce la somma delle componenti. Si scriva una funzione

```
coppia *ordina_somma(coppia a[], int n)
```

che dato un array *a[]* contenente *n* coppie, alloca e restituisce un nuovo array di *n* coppie che contiene le stesse coppie di *a[]* ordinate per somma decrescente. Ad esempio, se l'array *a[]* contiene

```
[(3,1), (5,2), (-4,3), (2,8), (1,-1)]
```

l'invocazione della funzione deve restituire l'array

```
[(2,8), (5,2), (3,1), (1,-1), (-4,3)]
```

senza modificare l'array di input *a[]*. La funzione *ordina_somma* deve eseguire l'ordinamento utilizzando la funzione di libreria *qsort*.

1c) Si scriva una funzione *main* che

- apre in lettura il file di nome *argv[1]*
- crea un array di coppie invocando *leggi_coppie* e passando il file appena aperto di nome *argv[1]*
- invoca la funzione `ordina_somma` per creare l'array di coppie ordinato rispetto alla somma
- scrive in un file `prima.bin` l'array di coppie non ordinato, in formato binario, 4 byte per intero quindi 8 byte per ogni coppia
- scrive in un file `dopo.bin` l'array di coppie ordinato, in formato binario, 4 byte per intero quindi 8 byte per ogni coppia
- dealloca tutta la memoria utilizzata (verificare con valgrind) e termina

Per visualizzare sul terminale gli interi contenuti in un file binario, potete usare il comando `od -An -t d4 <nomefile>`.

Per creare un file di coppie in formato binario contenente i valori passati sulla linea di comando potete usare il programma riportato qui sotto.

```
#include <stdio.h>    // permette di usare scanf printf
#include <stdlib.h>   // conversioni stringa/numero rand() abs() exit()
#include <stdbool.h>  // gestisce tipo bool (per variabili booleane)
#include <assert.h>   // permette di usare la funzione assert
#include "coppie.h"

int main(int argc, char *argv[])
{
    if(argc < 4 || argc%2!=0) { \\>
        printf("Uso: %s nomefile i1 i2 i3 i4 i5 i6 ....\n",argv[0]);
        return 1;
    }
    FILE *f = fopen(argv[1],"wb");
    for(int i=2;i<argv;i++) {  \\>
      int n = atoi(argv[i]);
      if(fwrite(&n,sizeof(int),1,f)!=1) {
         perror("Errore scrittura"); exit(1);
      }
    }
    fclose(f);
    return 0;
}
```

2a) Si consideri la struct

```
typedef struct par {
  char *stringa;
  struct par *next;
} parola;
```

che permette di rappresentare una lista di stringhe.

Si scriva una funzione

```
  parola *crea_lista(char *s)
```

che data una stringa `s` crea una lista di stringhe ordinata lessicograficamente contenente i token restituiti dalla funzione `strtok` usando come delimitatore il carattere `:`. Ad esempio, se la stringa `s` contiene `casa:telo-mare:albero::sedia` deve essere restituita la lista contenente nell'ordine le stringhe `albero`, `casa`, `sedia`, `telo-mare`.

2b) Si scriva una funzione

```
  parola *elimina_sefinale(parola *, char c)
```

che data una lista di stringhe elimina e dealloca gli elementi contenenti una stringa che termina con il carattere `c`. Ad esempio, se si chiama `elimina_sefinale` sulla lista del punto **2a)** con il secondo argomento uguale al carattere `a`, deve essere restituita la lista contenente solamente `albero`, `telo-mare`.

2c) Scrivere una funzione *main* che:

- verifica che sia *argc==3*
- crea una lista `lis` invocando `crea_lista` passando come argomento la stringa `argv[1]`
- esegue `lis = elimina_sefinale(lis, c)` assegnando a `c` tutti i caratteri della stringa `argv[2]` (quindi se `argv[2]` ha lunghezza 4 devono essere fatte 4 invocazioni a `elimina_sefinale`).
- stampa su `stderr` le lunghezze di tutte le stringhe che si trovano ancora in `lis`
- dealloca tutti gli elementi rimasti in `lis` e termina; verificare con valgrind che tutta la memoria venga correttamente deallocata.

Tempo a disposizione: 2 ore

Compito del 2/2/21

1a) Utilizzare la struttura

```
typedef struct el {
  int val;
  struct el *next;
} intero;
```

per rappresentare un elemento di una lista di interi. Scrivere una funzione

```
intero *crea_lista(FILE *f)
```

che dato un file di testo `f` contenente interi, restituisce una lista contenente tutti gli interi del file mantenendo l'ordine in cui compaiono. Ad esempio, se il file contiene

```
41 45
10
7 13

85
```

la lista risultante deve essere [41, 45, 10, 7, 13, 85]. *Suggerimento:* usare *fscanf(f,"%d", ...)* per leggere gli interi dal file.

1b) Scrivere una funzione

```
int maxdiff(intero *lis)
```

che data una lista di interi `lis` restituisce la massima differenza fra due elementi consecutivi della lista. Nell'esempio del punto precedente la massima differenza è 35 (differenza fra il secondo e il terzo elemento)

1c) Scrivere una funzione

```
intero *cancella5(intero *lis)
```

che data una lista di interi `lis` cancella e dealloca tutti gli elementi che contengono un intero multiplo di 5

1d) Scrivere una funzione *main* che:

- verifica che sia *argc==2*
- apre in lettura il file *argv[1]* e invoca la funzione *crea_lista* passandogli il file *argv[1]*
- visualizza su *stderr* gli elementi della lista (usando un while o una funzione apposita)
- invoca la funzione *maxdiff* e visualizza il risultato su *stderr*
- invoca la funzione *cancella5* e visualizza su *stderr* la lista risultante
- chiude i file, dealloca tutta la memoria utilizzata (verificare con valgrind), e termina

Si consiglia di testare il programma anche con il file contenente i seguenti interi

```
5 10 15 17 21 25 27 35 40 45 50 55 61 65 71
```

che dovrebbe visualizzare -2 come massima differenza e terminare visualizzando la lista `17 21 27 61 71`.

2a) Scrivere una funzione

```
int map(int a[], int n, int fun(int, int))
```

che dato un array a[] di n>1 interi, e una funzione `fun` calcola le quantità b(0), b(1), b(2), ... b(n-2)

```
    b(0) = fun(a[0],a[1])
    b(1) = fun(b(0),a[2])
    b(2) = fun(b(1),a[3])
    ...
```

fino a

```
    b(n-2) = fun(b(n-3),a[n-1])
```

e restituisce il valore b(n-2)

Ad esempio se a = {2,4,0,1} e la funzione somma e' definita da

```
int somma(int a, int b) {
  return a+b;
}
```

allora l'invocazione map(a,4,somma) deve restituire 7 (uguale a 2+4+0+1)

2b) Scrivere un main che * verifica che sia argc>3 * trasforma gli elementi argv[1]... argv[argc-1] in un array a[] di argc-1 interi

- invoca la funzione map passandogli l'array a[] e come parametro fun la funzione f(a,b)=a-b stampando il risultato su stdout
- invoca la funzione map passandogli l'array a[] e come parametro fun la funzione f(a,b)=1+a*b stampando il risultato su stdout

Tempo a disposizione: 2 ore

Compito del 29/1/20

1a) Si consideri la struct

```
typedef struct str {
  char *s;
  struct str *next;
} stringa;
```

che permette di rappresentare una lista di stringhe.

Scrivere un file `stringhe.c`, e il relativo `stringhe.h`, contenente le seguenti funzioni. Nel seguito `lis` è utilizzata per indicare una variabile che rappresenta una lista di stringhe (che puo' essere vuota), mentre `el` indica una variabile che punta ad una singola stringa.

```
// crea una un elemento di tipo stringa contenente una copia di *s
// per duplicare un array di caratteri usare strdup()
stringa *crea_stringa(char *s)

// inserisce l'elemento el in testa alla lista lis
stringa *inserisci(stringa *lis, stringa *el)

// stampa sul file f la stringa in el,
void stampa_stringa(FILE *f, stringa *el, bool reverse)

// stampa sul file f tutte le stringhe di lis separate da \n
// questa funzione *deve* utilizzare stampa_stringa()
void stampa_lista(FILE *f, stringa *lis)

// dealloca tutti gli elementi della lista lis
// e anche tutti gli array di caratteri a cui questi elementi puntano
void distruggi_lista(stringa *lis)
```

1b) Si scriva una funzione:

```
stringa *duplica_se_pari(stringa *lis, int (*fun)(char *))
```

che data una lista di stringhe *lis* e una funzione *fun* che prende in input un array di caratteri e restituisce un intero, duplica tutti gli elementi della lista per cui la funzione *fun* restituisce un intero pari. La funzione deve resituire il puntatore al primo elemento della lista modificata. Ad esempio, se la lista contiene le stringhe

```
casa
aaa
 7yx yyy
```

le cui lunghezze rispettive sono 4, 3, e 8, invocando la funzione *duplica_se* passando questa lista e la funzione *strlen* devono essere duplicati il primo e il terzo elemento della lista. Di conseguenza al termine dell'esecuzione della funzione la lista deve contenere le stringhe

```
casa
casa
aaa
 7yx yyy
 7yx yyy
```

Si noti che duplicare un elemento significa anche duplicare la sequenza di caratteri in esso contenuto. La funzione deve restituire il puntatore al primo elemento della lista modificata.

▶ Commento al testo

1c) Scrivere un file `main.c` che esegue le seguenti azioni:

- apre in lettura il file di testo di nome `argv[1]`

- legge una linea alla volta da tale file utilizzando la funzione *getline*, e crea una lista di stringhe in cui ogni elemento contiene una linea del file chiamando ripetutamente le funzioni *crea_stringa* e *inserisci*. Importante: l'eventuale `\n` presente in fondo alla linea non deve essere memorizzato nella lista.
- definisce una funzione *int conta_maiuscole(char *s)* che restituisce il numero di caratteri maiuscoli nella stringa s, usare la funzione di libreria *isupper()* per stabilire de un dato carattere è pari.
- invoca la funzione *duplica_se_pari* passando come secondo argomento la funzione *conta_maiuscole*
- apre in scrittura il file di testo di nome `argv[2]` scrivendoci il contenuto della lista ottenuta al passo precedente,
- chiude tutti i file e dealloca la lista di stringhe utilizzando *distruggi_lista*.

Fa parte dell'esercizio scrivere il makefile in modo che main.c e stringhe.c vengano compilati separatamente e vengano successivamente linkati per creare l'eseguibile *esercizio1*. Si utilizzi *valgrind* per verificare che tutta la memoria venga correttamente deallocata.

Esempio di esecuzione. Se il file di input contiene:

```
Hello world

tre
Hello World
Ciao
CIAO
```

il file di output dovrebbe essere:

```
CIAO
CIAO
Hello World
Hello World
Ciao
tre
tre


Hello world
```

Si noti che gli elementi vengono ribaltati perché in lettura gli elementi sono inseriti in cima alla lista. La seconda linea del file di input è la stringa vuota, che il programma deve gestire correttamente (viene duplicata perché contiene un numero pari di maiuscole).

2a) Scrivere una funzione

int coppie_uni(int n)

che dato un intero positivo *n* restituisce quante sono le coppie di uni consecutivi che appaiono all'interno della rappresentazione binaria di *n*. Ad esempio (tra parentesi la rappresentazione binaria dell'input):

coppie_uni(8) -> 0 (1000) coppie_uni(17) -> 0 (10001) coppie_uni(11) -> 1 (1011) coppie_uni(27) -> 2 (11011) coppie_uni(14) -> 2 (1110) coppie_uni(15) -> 3 (1111)

▶ Commento al testo

2b) Scrivere un programma che legge dalla riga di comando il nome di un file *nomefile* e un intero positivo *n* e crea un file *nomefile* contenente l'elenco di tutti gli interi positivi minori o uguali a *n* la cui rappresentazione binaria contiene esattamente una coppia di uni consecutivi. Ad esempio scrivendo

esercizio2 pippo 30

deve essere creato il file binario *pippo* contenente gli interi (di fianco ad ogni intero indico tra parentesi la rapprentazione binaria, voi non dovete scriverla):

```
3 (11)
6 (110)
11 (1011)
12 (1100)
13 (1101)
19 (10011)
22 (10110)
24 (11000)
25 (11001)
26 (11010)
```

Il file creato deve essere in formato binario quindi usare 4 byte per ogni intero; nell'esempio precedente il file *pippo* deve avere lunghezza 36 byte. Potete usare il comando `od -An -t d4` per visualizzare sul terminale gli interi contenuti nel file.

Compito del 25/9/19

1a) Si consideri la struct

```
typedef struct el {
  int val;
  struct el *next;
} intero;
```

che rappresenta un elemento di una tradizionale lista di interi, e la struct

```
typdef struct {
   intero *primo;
   intero *ultimo;
} priult;
```

che rappresentano una coppia di puntatori al primo e all'ultimo elemento di una lista di interi. Ad esempio, se la lista contiene gli elementi

```
2 -> 3 -> 5 -> NULL
```

dove la freccia indica il campo *next*, tale lista è rappresentata da una struttura *priult* in cui il campo *primo* punta all'elemento *2* e il campo *ultimo* all'elemento *5*. Se la lista contiene un solo elemento entrambi i campi *primo* e *ultimo* punteranno a quell'elemento, se la lista è vuota entrambi i campi *primo* e *ultimo* devono essere uguali a *NULL*. Il vantaggio di mantenere un puntatore sia al primo che all'ultimo elemento è che per aggiungere un elemento in fondo alla lista possiamo (= dobbiamo) usare il puntatore *ultimo* e quindi non è necessario scorrere tutti gli elementi di una lista.

Scrivere un file `priult.c`, e il relativo `priult.h`, contenente le seguenti funzioni.

```
// crea un oggetto priult che rappresenta una lista vuota
// quindi con i campi primo e ultimo uguali a NULL
priutl *crea_lista(void)

// aggiunge l'intero n in testa alla lista rappresentata da lis
void inserisci_testa(priult *lis, int n)

// aggiunge l'intero n in coda alla lista rappresentata da lis
void inserisci_coda(priult *lis, int n)

// stampa sul file f gli interi della lista lis separati da uno spazio
// e con uno \n in fondo al termine della lista
void stampa_lista(FILE *f, priult *lis)

// dealloca tutti gli elementi della lista lis
// e successivamente anche la struct priult a cui punta lis
void distruggi_lista(priult *lis)
```

Per effettuare il test di queste funzioni potete utilizzare il file main.c riportato qui sotto. **Fa parte dell'esercizio scrivere il makefile in modo che main.c e priult.c vengano compilati separatamente**. Si utilizzi *valgrind* per verificare che tutta la memoria venga correttamente deallocata.

```
// main.c per testare esercizio sulle liste di coppie
// da compilare separatamente da coppie.c
#include <stdio.h>    // permette di usare scanf printf
#include <stdlib.h>   // conversioni stringa/numero rand() abs() exit() etc ...
#include <stdbool.h>  // gestisce tipo bool (per variabili booleane)
#include <assert.h>   // permette di usare la funzione assert
#include "priult.h"

int main(int argc, char *argv[])
{
    priult *lis = crea_lista();
    stampa_lista(stderr,lis); // non dovrebbe stampare nulla
    aggiungi_testa(lis, 7);
    stampa_lista(stderr,lis); // -> 7
    aggiungi_coda(lis, 5);
    stampa_lista(stderr,lis); // -> 7 5
    aggiungi_testa(lis, 1);
    aggiungi_testa(lis, 2);
    aggiungi_coda(lis, 4);
    stampa_lista(stderr,lis); // -> 2 1 7 5 4
    distruggi_lista(lis);  // verificare con valgrind che venga deallocato tutto

    // questa parte serve a testare la domanda 1b
    // if(argc!=2) {puts("Inserisci il nome di un file"); exit(1);}
    // lis = crea_lista();
    // esegui_comandi(lis, argv[1]);
    // stampa_lista(stderr,lis);
    // distruggi_lista(lis);

    return 0;
}
```

1b) Scrivere una funzione

```
esegui_comandi(priult *lis, char *nomefile)
```

che data la lista *lis* e il nome di un file di testo contenente coppie di interi, apre il file e legge una coppia alla volta. Per ogni coppia letta *(a,b)* il valore *a* indica un intero da inserire nella lista; se *b* è pari il valore *a* va inserito in testa, mentre se *b* è dispari il valore *a* va inserito in coda. Ad esempio, se il file contenesse le coppie

```
2 0
5 0
1 1
4 0
10 3
```

dopo ogni operazione la lista dovrebbe avere i seguenti contenuti: [2], [5 2], [5 2 1], [4 5 2 1], [4 5 2 1 10].

1c) Aggiungere ai file *priutl.c* e *priult.h* la funzione

```
void cancella_ultimo(priult *lis)
```

che cancella l'ultimo elemento della lista **lis*, se **lis* è vuota non viene modificata. Aggiungere al main il seguente loop per verificare la correttezza della funzione *cancella_ultimo*

```
while(lis->primo!=NULL) {
  stampa_lista(stderr,lis);
  cancella_ultimo(lis);
}
```

e verificare con *valgrind* che tutta la memoria sia stata correttamente deallocata.

2) Scrivere un programma *inverti* che invocato dalla linea di comando scrivendo

```
inverti infile outfile
```

legge tutte linee dal file di testo *infile* e le riscrive su *outfile* in ordine inverso. Ad esempio, se *infile* contiene

```
hello world!
tre
xx yy zz
uno

ciao
```

il file di output deve contenere

```
ciao

uno
xx yy zz
tre
hello world!
```

Si usi una strategia a scelta per eseguire l'inversione delle linee, ma il vostro programma deve funzionare con un file con un numero quasiasi di linee arbitrariamente lunghe (quindi non usate array di lunghezza prefissata). E' obbligatorio usare *getline* per leggere le linee dal file di input; e si verifichi con *valgrind* che al termine dell'esecuzione tutta la memoria sia stata correttamente deallocata.

3) Scrivere un programma *uni* che invocato dalla linea di comando scrivendo

```
uni a1 a2 ... ak
```

dove a1 a2 ... ak sono interi positivi, visualizzi il massimo valore che si ottiene effettuando l'or bit a bit di due degli interi a1 a2 ... ak. Ad esempio, l'invocazione

```
uni 1 5 4 3
```

dato che le rappresentazioni binarie degli interi sulla linea di comando sono

```
1 101 100 11
```

deve essere visualizzato il valore 7 (111 in binario) corrispondente a (101 | 11).

Per fare delle verifiche potete usare il programma *uno.py* riportato qui sotto che risolve l'esercizio in python. Il programma va invocato scrivendo:

```
 python3 uni.py a1 a2 ... ak
```

```
#!/usr/bin/env python3
# programma uni.py
import random, sys, itertools

if len(sys.argv)<2:  #>
  print("Uso:\n\t %s a1 a2 ... ak" % sys.argv[0])
else:
  a = [int(x) for x in sys.argv[1:]]
  sys.stdout.write("Rappresentazioni binarie: ")
  [print("{0:b}".format(x),end=" ") for x in a]
  maxx = max([c[0] | c[1]  for c in itertools.combinations(a,2)] )
  print("\nRisultato:",maxx, "({0:b})".format(maxx) )
```

Tempo a disposizione 2 ore e 30 minuti

Compito del 10/7/19

1a) Si scriva una funzione

```
int *leggi_array(char *nomefile, int *num)
```

che dato il nome di un file *nomefile* alloca e restituisce un array di interi ottenuto leggendo gli interi da *nomefile* in formato binario (4 byte per ogni intero). La funzione deve restituire il puntatore al primo elemento dell'array e memorizzare in *num* il numero di elementi dell'array (che sarà uguale alla lunghezza del file diviso 4). Se il file non esiste o non può essere letto la funzione deve restituire NULL e porre *num* a zero

1b) Si utilizzi la struct

```
typedef struct cop {
  int primo;
  int secondo;
  struct cop *next;
} coppia;
```

che permette di rappresentare un elemento di una lista di coppie di interi. Si scriva una funzione

```
coppia *unisci(int *a1, int n1, int *a2, int n2)
```

che dati i due array *a1* e *a2* con rispettivamente *n1* e *n2* elementi restituisce la lista di coppie i cui elementi sono ottenuti prendendo *primo* da *a1* e *secondo* da *a2* rispettando l'ordine. Ad esempio, se *a1 = {1,4,0,5}* e *a2 = {3, -1, 1, 4, 4, 8}* la lista restituita deve contenere le coppie

```
(1,3) (4,-1) (0,1) (5,4)
```

in quest'ordine. Si noti che, come nell'esempio precedente, il numero di elementi della lista deve essere pari al numero di elementi dell'array più corto; gli elementi finali dell'array più lungo sono semplicemente ignorati.

1c) Si scriva una funzione

```
void scrivilistafun(FILE *f, coppia *lis, int (*fun)(int))
```

che dato una lista di coppie *lis*, scrive in formato testo nel file *f* i valori ottenuti applicando la funzione *fun* ad ognuno degli interi di ogni coppia. Ad esempio, se *lis* contiene le 4 coppie riportate sopra e la funzione *fun* restituisce il quadrato del suo input, nel file *f* devono essere scritte le 4 righe

```
1 9
16 1
0 1
25 16
```

1d) Si scriva un *main* che legge dalla linea di comando i nomi di due file binari, e crea due array di interi chiamando la funzione *leggi_array* passando i nomi dei due file letti sulla riga di comando. Successivamente il main deve creare una lista di coppie chiamando la funzione *unisci* passando come argomenti i due array precedentemente ottenuti. La lista di coppie deve essere infine passata alla funzione *scrivilistafun* con *stderr* come primo argomento e la funzione che restituisce il quadrato del suo input come terzo argomento. Prima di terminare il programma deve deallocare tutta la memoria utilizzata; utilizzare *valgrind* per la verifica.

Per creare un file di interi in formato binario contenente dei valori scelti da voi, potete usare il programma *genera_fileinteri* definito dal seguente main

```
#include <stdio.h>    // permette di usare scanf printf
#include <stdlib.h>   // conversioni stringa/numero rand() abs() exit() etc ...
#include <stdbool.h>  // gestisce tipo bool (per variabili booleane)
#include <assert.h>   // permette di usare la funzione assert

int main(int argc, char *argv[])
{
    if(argc<3 || argc%2!=0) {
        printf("Uso: %s nomefile i1 i2 i3 ....\n",argv[0]);
        return 1;
    }
    FILE *f = fopen(argv[1],"wb");
    for(int i=2;i<argc;i++) {
      int n = atoi(argv[i]);
      if(fwrite(&n,sizeof(int),1,f)!=1) {
         perror("Errore scrittura"); exit(1);
      }
    }
    fclose(f);
    return 0;
}
```

2a) Scrivere una procedura

```
void triangolo(FILE *f, int n)
```

che dato un intero positivo *n* scrive nel file *f* *n+1* righe in formato testo. Per *i*=0,1,2,...,*n* l'iesima riga deve contenere tutte le stringhe composte da *una* lettera **a** e da *i* lettere **b**. Ad esempio invocando

triangolo(stdout, 5)

devono essere scritte su *stdout* le seguenti 6 righe di testo

```
a
ab ba
abb bab bba
abbb babb bbab bbba
abbbb babbb bbabb bbbab bbbba
abbbbb babbbb bbabbb bbbabb bbbbab bbbbba
```

2b) Scrivere una procedura

char **elenco(int n, int *num)

che dato un intero positivo *n* crea e restituisce un array contenente le stringhe viste sopra. La procedura deve restituire il puntatore al primo elemento dell'array e scrivere nella variabile *num* il numero di elementi nell'array (come abbiamo sempre fatto). Ad esempio, invocando

```
elenco(5,&num)
```

deve essere creato un array contenente le 21 stringhe mostrate sopra (in un ordine qualsiasi) e il valore 21 deve essere memorizzato nella variabile *num*.

2c) Scrivere un *main* che legge un intero positivo *n* dalla linea di comando, invoca `triangolo(stdout, n)` e successivamente `elenco(n,&num)` ottenendo un array di stringhe *a* che deve poi essere ordinato lessicograficamente con *qsort* e *strcmp*. Infine le stringhe così ordinate devono essere scritte su *stderr*. Se ad esempio *n* = 2, su *stderr* devono essere scritte le stringhe:

```
a
ab
abb
ba
bab
bba
```

Prima di terminare il programma deve deallocare tutta la memoria utilizzata; utilizzare *valgrind* per la verifica.

Tempo a disposizione: 2 ore e 30 minuti.

Compito del 26/6/19

1a) Si consideri la struct

```
typedef struct cop {
  int primo;
  int secondo;
  struct cop *next;
} coppia;
```

che permette di rappresentare una lista di coppie di interi.

Scrivere in un file coppie.c, e il relativo coppie.h, contenente le seguenti funzioni. Nel seguito `lis` è utilizzata per indicare una variabile che rappresenta una lista di coppie (che puo' essere vuota), mentre `el` indica una variable che punta ad una singola coppia.

```
// crea una coppia contenente i valori x e y
coppia *crea_coppia(int x, int y)

// inserisce l'elemento el in cima alla lista lis
coppia *inserisci_coppia(coppia *lis, coppia *el)

// legge 2 interi dal file binario f (8 byte in totale) e crea una coppia
// con i valori corrispondenti chiamando la funzione crea_coppia()
// restituisce NULL se non e' possibile leggere i due interi dal file f
coppia *crea_coppia_bin(FILE *f)

// crea una lista con tutti gli interi presenti nel file
// il cui nome e' nomefile. Si supponga che gli interi
// siano memorizzati nel file in formato binario e che quindi
// devono essere letti con crea_coppie_bin().
// L'ordine con cui le coppie appaiono nel file non e' importante
// se non e' possibile leggere dal file la funzione deve
// restituire la lista vuota
coppia *crea_lista(char *nomefile)

// stampa sul file f gli interi della coppia el separati da uno spazio
void stampa_coppia(FILE *f, coppia *el)

// stampa sul file f tutte le coppie di lis separate da \n
// questa funzione *deve* utilizzare stampa_coppia()
void stampa_lista_coppie(FILE *f, coppia *lis)

// restituisce un puntatore alla coppia della lista lis
// la cui somma delle componenti (primo+secondo) ha valore massimo
// se ci sono più coppie con la stessa somma delle componenti
// restituire quella più lontana dalla testa della lista.
coppia *cerca_coppia_maxsomma(coppia *lis)

// restituisce un puntatore alla prima coppia della lista lis
// per cui la funzione f applicata ai campi primo e secondo
// restituisce true. Se non esiste una tale coppia la funzione
// deve restituire NULL.
coppia *cerca_coppia(coppia *lis, bool (*f)(int, int))

// dealloca tutti gli elementi della lista lis
void distruggi_lista_coppie(coppia *lis)
```

Per effettuare il test di queste funzioni potete utilizzare il file main.c riportato qui sotto. **Fa parte dell'esercizio scrivere il makefile in modo che main.c e coppie.c vengano compilati separatamente**. Si utilizzi *valgrind* per verificare che tutta la memoria venga correttamente deallocata.

```
// main.c per testare esercizio sulle liste di coppie
// da compilare separatamente da coppie.c
#include <stdio.h>    // permette di usare scanf printf
#include <stdlib.h>   // conversioni stringa/numero rand() abs() exit() etc ...
#include <stdbool.h>  // gestisce tipo bool (per variabili booleane)
#include <assert.h>   // permette di usare la funzione assert
#include "coppie.h"


bool positivi(int a, int b)
{
  return a>0 && b>0;
}

int main(int argc, char *argv[])
{
    if(argc != 2) {
        printf("Uso: %s nomefile\n",argv[0]);
        return 1;
    }
    FILE *f = fopen(argv[1],"rb");
    coppia *lista = crea_lista(argv[1]);
    stampa_lista_coppie(stdout,lista);
    puts("------");
    coppia *maxc = cerca_coppia_maxsomma(lista);
    stampa_lista_coppie(stdout,maxc);
    puts("------");
    coppia *lista2 = cerca_coppia(lista,positivi);
    stampa_lista_coppie(stdout,lista2);
    puts("------");
    distruggi_lista(lista);
    return 0;
}
```

Inoltre, per creare un file di coppie in formato binario contenente dei valori scelti da voi, potete usare il programma *genera_coppie* definito da seguente main

```
#include <stdio.h>    // permette di usare scanf printf
#include <stdlib.h>   // conversioni stringa/numero rand() abs() exit() etc ...
#include <stdbool.h>  // gestisce tipo bool (per variabili booleane)
#include <assert.h>   // permette di usare la funzione assert
#include "coppie.h"

int main(int argc, char *argv[])
{
    if(argc<4 || argc%2!=0) {
        printf("Uso: %s nomefile i1 i2 i3 i4 i5 i6 ....\n",argv[0]);
        return 1;
    }
    FILE *f = fopen(argv[1],"wb");
    for(int i=2;i<argc;i++) {
      int n = atoi(argv[i]);
      if(fwrite(&n,sizeof(int),1,f)!=1) {
         perror("Errore scrittura"); exit(1);
      }
    }
    fclose(f);
    return 0;
}
```

1b) Scrivere una funzione

```
coppia *duplica_lista(coppia *lis, bool (*f)(int, int))
```

che data una lista di coppie *lis* crea una nuova lista di coppie (senza modificare *lis*) contenente **una copia** di tutte le coppie per cui la funzione *f* restituisce true. Ad esempio se *lis* contiene le coppie (1,2) (5,-1) (2,-3) (4,4) (8,2) (-1,-1) la funzione *duplica_lista* con secondo argomento la funzione *positivi()* definita sopra deve restituire una nuova lista contenente (1,2) (4,4) (8,2).

---

2) Scrivere un programma che crea un array contenente una copia delle stringhe passate sulla linea di comando e una copia di tali stringhe ribaltate (ad esempio "Sole" -> "eloS"). Questo array di stringhe deve essere visualizzato e poi ordinato ignorando la distinzione tra maiuscole e minuscole, utilizzando la funzione *qsort()* combinata a *strcasecmp()*. L'array ordinato deve essere nuovamente visualizzato e successivamente completamente deallocato (usare valgrind per la verifica). Esempio, scrivendo:

```
esercizio2 Sole sale sTalle
```

deve essere prima visualizzato l'array non ordinato

```
Sole sale sTalle eloS elas ellaTs
```

e successivamente l'array ordinato

```
elas
ellaTs
eloS
sale
Sole
sTalle
```

Si noti che se non si ignora la distinzione tra maiuscole e minuscole le ultime 3 parole sarebbero ordinate `Sole sTalle sale` perchè in ASCII le maiuscole vengono prima delle minuscole.

▶ Suggerimento da leggere se l'ordinamento non funziona

Compito del 23/1/19

1a) Si consideri la struct

```
typedef struct str {
  char *s;
  struct str *next;
} stringa;
```

che permette di rappresentare una lista di stringhe.

Scrivere un file `stringhe.c`, e il relativo `stringhe.h`, contenente le seguenti funzioni. Nel seguito `lis` è utilizzata per indicare una variabile che rappresenta una lista di stringhe (che puo' essere vuota), mentre `el` indica una variabile che punta ad una singola stringa.

```
// crea una un elemento di tipo stringa contenente una copia di *s
// per duplicare un array di caratteri usare strdup()
stringa *crea_stringa(char *s)

// inserisce l'elemento el in fondo alla lista lis
stringa *inserisci_coda(stringa *lis, stringa *el)

// crea una nuova lista contenente copia delle stringhe contenute in *lis
// e trasformando in maiuscola la prima lettera di ogni stringa e le lettere
// immediatamente dopo uno spazio. Usare la funzione toupper()
// per convertire le maiuscole in minuscolo
stringa *maiuscola(stringa *lis)

// stampa sul file f la stringa in el,
// se reverse=true la stampa deve essere da destra a sinistra
// quindi "hello world" viene stampato "dlrow olleh"
void stampa_stringa(FILE *f, stringa *el, bool reverse)

// stampa sul file f tutte le stringhe di lis separate da \n
// Le stringhe per cui la funzione fun restituisce true
// devono essere stampate da destra a sinistra
// questa funzione *deve* utilizzare stampa_stringa()
void stampa_lista(FILE *f, stringa *lis, bool (*fun)(char *))

// dealloca tutti gli elementi della lista lis
// e anche tutti gli array di caratteri a cui questi elementi puntano
void distruggi_lista(stringa *lis)
```

1b) Scrivere un file `main.c` che esegue le seguenti azioni:

- apre in lettura il file di testo di nome `argv[1]`
- legge una linea alla volta da tale file utilizzando la funzione *getline*, e crea una lista di stringhe in cui ogni elemento contiene una linea del file chiamando ripetutamente le funzioni *crea_stringa* e *inserisci_coda*. Importante: l'eventuale `\n` presente in fondo alla linea non deve essere memorizzato nella lista.
- crea una seconda lista invocando la funzione *maiuscola* sulla lista creata in precedenza
- definisce una funzione *bool lunghezza_pari(char *)* che restituisce *true* se la stringa passata ha lunghezza pari e *false* altrimenti
- apre in scrittura il file di testo di nome `argv[2]`
- scrive su tale file gli elementi della prima lista, utilizzando la funzione *stampa_lista* passando come secondo argomento la funzione *lunghezza_pari*
- stampa su terminale (stdout) il contenuto della seconda lista. Se possibile usare sempre *stampa_lista* ma questa volta tutte le stringhe vengano stampate da sinistra a destra (dovete definirvi una nuova funzione da passare come terzo parametro a *stampa_lista*).
- chiude i file e dealloca entrambe le liste e le stringhe in esse contenute utilizzando *distruggi_lista*.

Fa parte dell'esercizio scrivere il makefile in modo che main.c e stringhe.c vengano compilati separatamente e vengano successivamente linkati per creare l'eseguibile *esercizio1*. Si utilizzi *valgrind* per verificare che tutta la memoria venga correttamente deallocata.

Esempio di esecuzione. Se il file di input contiene:

```
hello world!
tre
xx yy zz
uno.
uno
```

il file di output dovrebbe essere:

```
!dlrow olleh
tre
zz yy xx
.onu
uno
```

e l'output su terminale dovrebbe essere:

```
Hello World!
Tre
Xx Yy Zz
Uno.
Uno
```

Compito del 26/9/18

1a) Si utilizzi la struttura

```
typedef strict el {
  int val;
  struct el *next;
} intero;
```

per rappresentare un elemento di una lista di interi. Scrivere una funzione

```
intero *crea_lista(int n)
```

che dato un intero positivo `n` crea una lista contenente `n` interi uguali a 10, 20, 30, etc fino al valore n*10. Ad esempio, se `n==8` la lista restituita deve contenere i valori (10,20,30,40,50,60,70,80) in quest'ordine.

1b) Scrivere una funzione

```
intero *inserisci(intero *lis, int val, int pos)
```

che inserisce l'intero `val` all'interno della lista `lis` secondo la seguente regola: se `pos==0` l'intero va inserito in fondo alla lista; se `pos==1` l'intero va inserito immediatamente prima dell'ultimo elemento; se `pos==2` l'intero va inserito immediatamente prima del penultimo elemento e così via. Si noti che se `pos` è maggiore o uguale del numero di elementi della lista il valore `val` va inserito in cima alla lista.

1c) Scrivere una funzione

void lista2bin(intero* lis, FILE *f)

scrive nel file `f` gli interi della lista `lis` in formato binario (quindi usando 4 byte per ogni intero) utilizzando la funzione `fwrite`.

1d) Scrivere un `main` che converte in intero il parametro `argv[1]` verifica che sia positiva e lo passa alla funzione `crea_lista` ottenendo così una lista `lis` di `argv[1]` interi. Successivamente `main` deve invocare

```
lis = inserisci(lis,3,0);
lis = inserisci(lis,4,1);
```

e infine scrivere il nuovo contenuto di `lis` nel file di nome `argv[2]` utilizzando la funzione `lista2bin`. Prima di concludere `main` deve deallocare tutta la memoria utilizzata (verificare con valgrind).

2a) Si scriva una funzione

char **leggiparole(char *s, int *n)

che data una stringa `s` contenente un insieme di parole separate dal carattere : crea e restituisce un array di stringhe (quindi puntatori a char) contenente una copia delle parole di `s` nello stesso ordine in cui appaiono in `s`. La lunghezza dell'array creato deve essere memorizzata in `*n`.

Ad esempio, l'invocazione

```
leggiparole(strdup("uno:nessuno:centomila"),&n);
```

deve restituire il puntatore ad un array contenente le stringhe "uno", "nessuno", "centomila" in quest'ordine e memorizzare il valore 3 in `n`. Usare la funzione di libreria `strtok` per dividere la stringa `s` in singole parole.

2b) Scrivere una funzione

```
int sommafun(char *a[], int n, int (*fun)(char *) )
```

che dato l'array `a` di `n` stringhe e una funzione `fun` che prende in input una stringa e restituisce un intero, applica `fun` a tutti gli elementi di `lis` e restituisce la somma dei valori così ottenuti

2c) Scrivere una funzione

```
int contavocali(char *s)
```

che data una stringa `s` restituisce il numero di vocali in essa contenuta. Per vocali si intendono uno dei caratteri `aeiouyAEIOUY`.

2d) Si scriva un `main` che passa una copia di `argv[1]` a `leggiparole` ottenendo un array di stringhe `a`. Successivamente deve invocare `sommafun` passandogli l'array `a` e la funzione `contavocali` come terzo argomento. Infine, la funzione `main` deve scrivere, in un file di testo di nome `argv[2]`, il valore restituito da `sommafun` seguito dagli elementi dell'array `a` tutti su linee distinte. Ad esempio, l'invocazione

```
esercizio3 int:long:char:double prova.txt
```

deve creare un file `prova.txt` contenente le linee

```
6
int
long
char
double
```

Prima di terminare il programma deve deallocare tutta memoria utilizzata (usare valgrind per verificarlo).

Compito del 12/9/18

1a) Si scriva una funzione

```
int *stringa2array(char *s, int *n)
```

che data una stringa `s` contenente numeri interi separati da spazio crea e restituisce un array di `int` contenente gli interi presenti nella stringa. La funzione deve restituire il puntatore all'array creato e memorizzare in `n` il numero dei suoi elementi. Ad esempio, l'invocazione

```
a = stringa2array(strdup("123 -5 34 40"),&n);
```

deve assegnare ad `a` l'array {123,-5,34,40} e alla variabile `n` il valore 4. Si consiglia di usare le funzioni di libreria `strtok` per dividere la stringa `s` in token e `atoi` per convertire i singoli token in interi.

1b) Si utilizzi la struttura

```
typedef strict el {
  int *a;
  int len;
  struct el *next;
} array;
```

per rappresentare un elemento di una lista di array di interi. Si scriva una funzione

```
array *file2lista(FILE *f);
```

che legge una linea alla volta dal file `f` (usare `getline` o `fgets`), passa le linee lette alla funzione `stringa2array` e restituisce la lista contenenti tali array. Ad esempio, se il file contiene le righe

```
-23 -5 11 34
23 11 -2
1 1 1 1 1 11
```

la funzione deve restituire una lista di 3 elementi, in cui il primo elemento contiene l'array {-23, -5, 11, 34}, il secondo elemento l'array {23, 11, -2}, e il terzo elemento l'array {1,1,1,1,1,11} (ripettate l'ordine degli elementi).

1c) Scrivere una funzione

```
int somma_tutto(array *lis)
```

che data una lista di array `lis` esegue la somma di tutti gli interi contenuti negli array nella lista `lis`. Nell'esempio precedente, il valore resituito dalla funzione dovrebbe essere

```
-23 -5 +11 +34 +23 +11 -2 +1 +1 +1 +1 +1 +11  = 65
```

1d) Scrivere un *main* che legge il nome di un file dalla linea di comando, invoca su di esso la funzione `file2lista` e sulla lista ottenuta la funzione `somma_tutto`. Al termine deve deallocare tutta la memoria utilizzata per lista e array (verificare con valgrind).

2a) Scrivere una funzione

```
int contaM(char *s)
```

che data una stringa `s` conta il numero di lettere maiuscole in essa contenute. Si usi la funzione `isupper` per determinare se un dato carattere è maiuscolo.

2b) Scrivere una funzione

```
ordina_maiuscole(char *a[], int n)
```

che, utilizzano la funzione di libreria `qsort`, ordina un array `a[]` di `n` stringhe in ordine di numero crescente di lettere maiuscole in esse contenute. Ad esempio, se

```
a = {"CIAO", "Milano", "OK", "pizza"}
```

dopo la chiamata a `ordina_maiuscole` gli elementi di `a` deve essere nell'ordine

```
{"pizza", "Milano", "OK", "CIAO"}
```

2c) Scrivere un `main` che invoca la funzione `ordina_maiuscule` sull'array di stringhe contenente gli elementi `argv[1]`, `argv[2]`, etc. e stampa l'array ordinato ottenuto.

3a) Si utilizzi la struttura

```
typedef strict elx {
  char *s;
  struct elx *next;
} stringa;
```

per rappresentare un elemento di una lista di stringhe. Si scriva una funzione

stringa *leggieordina(FILE *f)

che legge, mediante `fscanf`, le stringhe contenute nel file `f` e restituisce una lista di tali stringhe ordinate per numero crescente di lettere maiuscole in esse contenute. Di conseguenza se il file contenesse

```
CIAO pizza
Milano OK
```

la lista restituita dovrebbe contenere nell'ordine le stringhe:

```
pizza Milano OK CIAO
```

3b) Si scriva un `main` che legge il nome di un file dalla line di comando, apre il relativo file passandolo a `leggieordina` e stampa gli elementi dalla lista così ottenuta. Successivamente dealloca tutta la memoria utilizzata dalla lista.

Compito del 19/6/18

1a) Scrivere una funzione

```
int *ruota(int a[], int n)
```

che dato un array di interi a[] di n elementi alloca e restituisce un nuovo array di interi che contiene gli stessi elementi di a[] ma ruotati ciclicamente di un elemento verso sinistra. Ad esempio, se a = {2,3,5,7,11} la funzione deve restituire l'array {3,5,7,11,2}

1b) Scrivere una funzione

```
int **rotazioni(int a[], int n, int *righe)
```

che dato un array a[] di n elementi alloca e restituisce una matrice contenente tutte le rotazioni *distinte* di a[] (cioè tutti i vettori riga devono essere distinti). Ad esempio, se a = {2,3,5,7} la funzione deve restituire la matrice {{2,3,5,7},{3,5,7,2},{5,7,2,3},{7, 2, 3, 5}}. Se invece a = {2,3,2,3} la funzione deve restituire la matrice {{2,3,2,3}, {3,2,3,2}} (non ci sono altre righe perchè le rotazioni si ripetono). La funzione deve scrivere in `*righe` il numero di righe presenti nella matrice restituita. Suggerimento: scrivere una funzione ausiliaria che dati due array di n elementi restituisce *true* se essi hanno tutti gli elementi uguali.

1c) Scrivere un main che costruisca un array contenente gli interi atoi(argv[1]), ..., atoi(argv[argc-1]), chiama la funzione *rotazioni* su di esso, stampa la matrice risultante e dealloca tutto lo spazio utilizzato dalla matrice

1d) Scrivere una funzione

```
char ***rotazioni_stringhe(char *a[], int n, int *righe)
```

che dato un array a[] contenente *n* stringhe restituisce una matrice di stringhe, le cui righe contengono tutte le rotazioni distinte degli elementi di a[]. Ad esempio, se a = {"mare", "sole", "amore"} la funzione deve restituire la matrice {{"mare", "sole", "amore"}, {"sole", "amore", "mare"}, {"amore", "mare", "sole"}}

se invece a = {"mare", "sole", "mare", "sole"} la funzione deve restituire

```
{{"mare", "sole", "mare", "sole"},
 {"sole", "mare", "sole", "mare"}}
```

senza altre righe perché le rotazioni si ripetono. La funzione deve scrivere in `*righe` il numero di righe contenute nella matrice restituita.

Si suggerisce di realizzare questo esercizio seguendo la traccia dei punti a,b,c e quindi di scrivere una funzione

```
char **ruota_stringhe(char *a[], int n)
```

che dato un array di stringhe restituisce un nuovo array contenente copie delle stringhe di a[] ruotate ciclicamente di una posizione verso sinistra; e un'altra funzione che dati due array di stringhe restituisce *true* se essi hanno tutti gli elementi uguali.

1e) Rinominare il main precedente e scrivere un nuovo main che costruisca un array contenente una copia delle stringhe argv[1], ..., argv[argc-1], chiama la funzione *rotazioni_stringhe* su di esso, stampa la matrice risultante e dealloca tutto lo spazio utilizzato dalla matrice (controllare con valgrind).

2a) Si consideri la struct

```
typedef struct par {
  char *stringa;
  struct par *next;
} parola;
```

che permette di rappresentare una lista di stringhe. Scrivere una funzione

```
parola *delete_if(parola *lis, bool (*fun)(char *))
```

che data una lista di stringhe *lis* e una funzione *fun* cancella dalla lista *lis* tutti gli elementi contenenti stringhe per cui la funzione *fun* vale *true*. La funzione *delete_if* deve restituire il primo elemento della nuova lista o NULL se tutti gli elementi della lista vengono cancellati. Ad esempio data la funzione

```
bool lunghezza_pari(char *s) {
  return strlen(s)%2==0;
}
```

se *lis* contiene le stringhe "sole", "mare", "amore", "sale" l'invocazione

```
delete_if(lis, lunghezza_pari)
```

deve restituire la lista contenente solamente la stringa "amore". Si noti che quando si cancella un elemento di una lista si deve cancellare tutta la memoria da esso utilizzato

Per risolvere l'esercizio si consiglia di basarsi sulla seguente strategia ricorsiva:

```
if(fun(lis->stringa)) { // elemento da eliminare
  newlis = delete_if(lis->next,fun);
  cancella l'elemento a cui punta lis;
  return newlis;
}
else { // elemento da tenere
  lis->next = delete_if(lis->next,fun);
  return lis;
}
```

2b) Si scriva un main che crea una lista *lis* contenente una copia delle stringhe argv[1], argv[2], ..., argv[argc-1] (in quest'ordine) e invoca *delete_if* passandogli come argomento *lis* e *lunghezza_pari* e stampa gli elementi della lista risultante. Dopo la stampa la lista deve essere completamente deallocata.

Compito dell'11/7/18

1) Si consideri la struct

```
typedef struct cop {
  int primo;
  int secondo;
  struct cop *next;
} coppia;
```

che permette di rappresentare una lista di coppie di interi.

Scrivere in un file `coppie.c`, e il relativo `coppie.h`, contenente le seguenti funzioni. Nel seguito `lis` è utilizzata per indicare una variabile che rappresenta una lista di coppie (che puo' essere vuota), mentre `el` indica una variabile che punta ad una singola coppia.

```
// crea una coppia contenente i valori x e y
coppia *crea_coppia(int x, int y)

// inserisce l'elemento el in cima alla lista lis
coppia *inserisci_coppia(coppia *lis, coppia *el)

// stampa sul file f gli interi della coppia el separati da uno spazio
void stampa_coppia(FILE *f, coppia *el)

// stampa sul file f tutte le coppie di lis separate da \n
// questa funzione *deve* utilizzare stampa_coppia()
void stampa_lista_coppie(FILE *f, coppia *lis)

// restituisce la prima coppia della lista lis la cui somma è s
coppia *cerca_coppia(coppia *lis, int s)

// stampa sul file f tutte le coppie di lis la cui somma e' s
// questa funzione deve utilizzare cerca_coppia() e stampa_coppia()
void cerca_e_stampa_coppie(FILE *f, coppia *lis, int s)

// restituisce il numero di coppie i cui elementi passati a f() restituiscono true
int conta_coppie(coppia *lis, bool (*f)(int, int))

// dealloca tutti gli elementi della lista lis
void distruggi_lista_coppie(coppia *lis)
```

Per effettuare il test delle vostre funzioni potete utilizzare il file main.c riportato qui sotto. **Fa parte dell'esercizio scrivere il makefile in modo che main.c e coppie.c vengano compilati separatamente** e vengano successivamente linkati per creare l'eseguibile *esercizio1*. Si utilizzi *valgrind* per verificare che tutta la memoria venga correttamente deallocata.

```
// main.c per testare esercizio sulle liste di coppie
// da compilare separatamente da coppie.c
#include <stdio.h>    // permette di usare scanf printf
#include <stdlib.h>   // conversioni stringa/numero rand() abs() exit() etc ...
#include <stdbool.h>  // gestisce tipo bool (per variabili booleane)
#include <assert.h>   // permette di usare la funzione assert
#include "coppie.h"


bool uguali(int a, int b)
{
  return a==b;
}

int main(int argc, char *argv[])
{
    if(argc < 3) {
        printf("Uso: %s somma elemento [elemento]\n",argv[0]);
        return 1;
    }
    int somma = atoi(argv[1]);
    coppia *lista = NULL;
    for(int i=2;i<argc;i++) {
      int e = atoi(argv[i])
      coppia *a = crea_coppia(e, 2*e);
      lista = inserisci_coppia(a);
    }
    stampa_lista_coppie(stdout,lista);
    cerca_e_stampa_coppie(stdout,lista, 15);
    printf("Coppie: %d\n",conta_coppie(lista,uguali));
    distruggi_lista(lista);
    return 0;
}
```

2a) Scrivere una funzione

```
bool permutazione_int(int a[], int b[], int n)
```

che dati due array di n elementi restituisce true se il secondo array e' una permutazione del primo, e false altrimenti. Una possibile strategia di risoluzione è la seguente:

- Si cerca se esiste un elemento b[i] uguale a a[n-1].
- Se non esiste la funzione restituisce *false*
- Se esiste si scambia b[i] con b[n-1] e si invoca la funzione ricorsivamente sugli array di n-1 elementi

Potete utilizzare una strategia differente, ma ricordate che gli elementi di a e b possono essere interi qualsiasi.

2b) Scrivere una funzione

```
bool permutazione_generica(void *a[], void *b[], int n, int (*cmp)(void *, void *) )
```

che risolve lo stesso problema del punto precedente prendendo in input due array di puntatori generici e una funzione di confronto *cmp* che dati due elementi di a[] e b[] restituisce 0 se essi sono da considerare uguali e un valore diverso da 0 altrimenti

2c) Scrivere un main che

- verifica che argc sia dispari e maggiore di 4

- costruisce un array a[] di stringhe contenente (argc-1)/2 elementi di argv a partire da argv[1], e un array b[] contenente i rimanenti (argc-1)/2 elementi di argv
- invoca la funzione *permutazione_generica* passando gli array a[] e b[], il numero di elementi (argc-1)/2 e la funzione *strcmp*.

Ultime modifiche: giovedì, 10 febbraio 2022, 18:57