# Elaborazione testi - livello stringa

- Un testo può essere costruito o modificato a livello di stringhe se:
  - E' possibile identificare sottostringhe (sequenze di caratteri) sulle quali applicare operazioni di tipo unitario
  - Le operazioni su stringhe debbono trovarsi in libreria, oppure essere chiamate funzioni realizzate dal programmatore
- Spesso le sottostringhe sono sequenza di caratteri separate da spazi (facile l'input)
- Talvolta le sottostringhe includono spazi e/o sono delimitate da altri caratteri

# Formattazione di testo

- Formulazione:
  - E’ dato un file testo, visto come un insieme di righe, scomponibili in sottostringhe (di non più di 20 caratteri) separate da spazi (oppure '\t' o '\n')
  - Si realizzi una funzione C che, letto il file, ne copi il contenuto in un altro file (i nomi dei file sono ricevuti come parametri), dopo aver:
    - Ridotto le sequenze di più spazi ad un solo spazio
    - Inserito (in sostituzione di spazi) o eliminato caratteri a-capo ('\n') in modo tale che ogni riga abbia la massima lunghezza possibile, minore o uguale a lmax (terzo parametro della funzione)

# Formattazione di testo

- Soluzione:
  - E’ possibile operare sia a livello di caratteri che di stringhe
  - Una possibile soluzione per la gestione di stringhe parte dal fatto che l’input mediante scanf("%s") permette di isolare in modo automatico stringhe separate da spazi (oppure '\t' o '\n')
    - Iterazione di input di stringhe, con contestuale output e conteggio della lunghezza di una riga
    - Prima di fare l’output di una stringa, si decide (in base al conteggio di lunghezza riga) se occorre stampare un carattere a-capo

# Codice

```
void formatta (char nin[], char nout[],
               int lmax) {
  const int STRLEN=21;
  FILE *fin=fopen(nin, "r");
  FILE *fout=fopen(nout, "w");
  char parola[STRLEN];
  int l;

  l=0;
  ...
```

```
  while (fscanf(fin, "%s",parola)==1) {
    if (l+1+strlen(parola) > lmax) {
      fprintf(fout, "\n%s",parola);
      l=strlen(parola);
    }
    else {
      fprintf(fout,"%s%s",
                    l==0? "":" ",parola);
      l+=1+strlen(parola);
    }
  }
  fclose(fin); fclose(fout);
}
```

# Codice

```
void formatta (char nin[], char nout[],
               int lmax) {
  const int STRLEN=21;
  FILE *fin=fopen(nin, "r");
  FILE *fout=fopen(nout, "w");
  char parola[STRLEN];
  while (fscanf(fin, "%s",parola)==1) {
    if (l+1+strlen(parola) > lmax) {
      fprintf(fout, "\n%s",parola);
      l=strlen(parola);
    }
    else {
      fprintf(fout,"%s%s",
                    l==0? "":" ",parola);
      l+=1+strlen(parola);
    }
  }
  fclose(fin); fclose(fout);
}
```

Attenzione: test per non stampare uno spazio prima della prima parola
Espressione condizionale C:
`cond ? exprTrue : exprFalse`

# Problemi di verifica e selezione

- Verifica: decidere se un insieme di informazioni o dati rispettano un determinato criterio di accettazione:
  - La risposta ad un problema di verifica è Booleana (SÌ/NO)
  - Si possono verificare dati singoli oppure sequenze (insiemi) di dati
  - Un problema può consistere in una sola verifica o in più verifiche (su dati diversi)

- Selezione: separare i dati che rispettano un criterio di accettazione/verifica (rispetto a quelli che non lo rispettano)

# Criteri di accettazione

- I criteri possono essere:
  - Espressioni logiche (logica proposizionale)
  - Condizione (proprietà) p su di un insieme di dati S espressa mediante:
    - **Quantificatore universale** $\forall$: $\forall$ x $\in$S | p è vera
      - Per tutti gli  x appartenenti ad S la proprietà p è vera
      - Tutti gli elementi di S soddisfano p
    - **Quantificatore esistenziale** $\exists$: $\exists$ x $\in$S | p è vera
      - Esiste almeno un x appartenente ad S per cui la proprietà p è vera
      - (Almeno) un elemento di S soddisfa p

GOTTLOB FREGE
1879

# Dualità $\forall, \exists$ ($\neg$ operatore di negazione in logica)

- $\neg(\forall\ x \in S \mid p \text{ è vera}) \Leftrightarrow \exists\ x \in S \mid p \text{ è falsa}$

  non è vero che per tutti gli x appartenenti ad S la proprietà p è vera

  EQUIVALE A

  esiste almeno un x appartenente ad S per cui la proprietà p è falsa

- $\neg(\exists\ x \in S \mid p \text{ è vera}) \Leftrightarrow \forall\ x \in S \mid p \text{ è falsa}$

  non è vero che esiste almeno un x appartenente ad S per cui la proprietà p è vera

  EQUIVALE A

  per tutti gli x appartenenti ad S la proprietà p è falsa

# Esempio: monotonicità di una sequenza

Data una sequenza di N interi $S = (x_0, x_1, \dots, x_{N-1})$:

- Proprietà p: è una sequenza monotona crescente

$$\forall\, x_i, x_{i+1} \in S \mid (x_i \leq x_{i+1}) \quad 0 \leq i < N-1$$

per tutte le coppie di elementi adiacenti vale la relazione d'ordine ≤

- Proprietà $\neg$ p: NON è una sequenza monotona crescente

$$\exists\, x_i, x_{i+1} \in S \mid (x_i > x_{i+1}) \quad 0 \leq i < N-1$$

esiste almeno una coppia di elementi adiacenti per cui non vale la relazione d'ordine ≤, quindi vale >

# Criteri di accettazione

- Realizzazione in C:
  - Variabile intera utilizzata come logica (i.e., come se avesse valore vero/falso) e inizializzata al verdetto del quantificatore universale
    - E.g., 1 per vero, 0 per falso
    - Inizializzo col valore corrispondente al verdetto che assumo come corretto rispetto alla proprietà
  - Costrutto iterativo per enumerare tutti gli elementi dell'insieme S
  - Per ogni iterazione: controllo se il verdetto iniziale è confermato o contraddetto
    - La variabile logica viene modificata solo in caso di verdetto contraddetto

# Codice

```
...
int Monotona = 1; // verdetto inizializzato a 1 - assumo che sia monotona (vero)
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);

for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj)      // test sul verdetto tra due dati adiacenti
    Monotona = 0;  // eventuale modifica del verdetto se l'ho contraddetto
  xi = xj;
}
```

# Errore comune

```
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);
for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj)
    Monotona = 0;
  else
    Monotona = 1;
  xi = xj;
}
```

il verdetto dipende dall'esito dell'ultimo test

# Variante: uscita anticipata strutturata

```
...
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);

for (i=1; i<N && Monotona; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj)
    Monotona = 0;
  xi = xj;
}
```

uscita strutturata con flag
Ho trovato un caso in cui il verdetto è falso quindi posso fermarmi

# Variante: uscita anticipata non strutturata (1)

```
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);
for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj) {
    Monotona = 0;
    break;
  }
  xi = xj;
}
```

uscita non strutturata dal ciclo for

# Variante: uscita anticipata non strutturata (2)

```
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);
for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj) {
    Monotona = 0;
    return 0;
  }
  xi = xj;
}
return 1;
```

uscita non strutturata dalla funzione

# Variante: uscita anticipata non strutturata (3)

```
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);
for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d"
  if(xi > xj) {
    Monotona = 0;
    return 0;
  }
  xi = xj;
}
return 1;
```

verdetto ridondante sostituito dal valore di ritorno

# Verifiche su sequenze

- Verificare una sequenza di dati significa decidere se la sequenza rispetta un criterio di accettazione
- Esempi di criteri:
  - Sommatoria/media dei dati: superiore o inferiore a limite
  - Confronto tra dati adiacenti: sequenza di dati crescenti, differenza inferiore a massimo
  - Regole su gruppi di dati adiacenti:
    - Sono vietate più di 2 consonanti consecutive, i segni di punteggiatura sono seguiti da spazio, 'p' e 'b' non possono essere precedute da 'n' …
    - Un pacchetto di dati trasmessi deve rispettare regole di format
    - …

# Verifica di ordine alfabetico

- Formulazione:
  - Un file testo contiene un elenco di persone, ognuna delle quali è rappresentata, su una riga del file (al max. 50 caratteri), da cognome e nome (eventualmente contenenti spazi)
  - Si scriva una funzione C che, ricevuto come parametro il puntatore al file (già aperto), verifichi se i dati sono in ordine alfabetico, ritornando 1 in caso affermativo (0 in caso negativo)
- Soluzione:
  - Si tratta di analizzare iterativamente i dati, confrontando progressivamente i due ultimi acquisiti
  - Criterio di accettazione: quantificazione (tutte le righe devono soddisfare la condizione)
  - Verdetto: valore di ritorno della funzione di verifica
  - Uscita anticipata non strutturata

# Verifica di ordine alfabetico

- Struttura dati:
  - Due stringhe, rispettivamente per l'ultima (riga1) e la penultima riga (riga0) acquisite dal file
- Algoritmo:
  - Si leggono progressivamente le righe del file, confrontando le ultime due:
    - Se viene rispettato il criterio di ordinamento si procede
    - Non appena viene violato il criterio di ordinamento si ritorna 0
    - Al termine di ogni iterazione l'ultima riga diviene la penultima
      - Non devo salvare tutto il file!
    - Se non è mai stato violato il criterio si ritorna 1
  - Occorre gestire a parte la lettura della prima riga

# Codice

```
int verificaOrdine (FILE *fp) {
  const int MAXC=50;
  char riga0[MAXC+1], riga1[MAXC+1];

  fgets(riga0,MAXC,fp);
  while (fgets(riga1,MAXC,fp)!=NULL) {
    if (strcmp(riga1,riga0)<0) // se è vera, riga1 viene prima di riga0
      return 0;
    strcpy(riga0,riga1);
  }
  return 1;
}
```

# Verifica di congruenza dati

- Formulazione:
  - Un file testo contiene una sequenza di temperature (in gradi Celsius) rilevate da un sensore termico nell'arco di una giornata (a distanza di 5 minuti l'una dall'altra)
    - Le temperature (numeri reali) sono separate da spazi o a-capo
  - Si scriva una funzione C che, ricevuto come parametro il puntatore al file (già aperto), verifichi che, in ogni intervallo di 10 minuti, la temperatura non vari di più di 5 gradi, ritornando:
    - 1 in caso affermativo
    - 0 in caso negativo, da interpretare come malfunzionamento di sensore o controllo termico

# Verifica di congruenza dati

- Soluzione:
  - Analizzare iterativamente i dati, verificando progressivamente gli ultimi tre dati acquisiti (per coprire 10 minuti)
  - Criterio di accettazione:
    - Condizione booleana sui primi 2 dati
    - Quantificazione sui restanti
  - Verdetto: valore di ritorno della funzione di verifica
  - Uscita anticipata non strutturata

# Verifica di congruenza dati

- Struttura dati:
  - 3 variabili reali (float) per le ultime tre letture: t0, t1, t2
    - t2 è l'ultimo dato acquisito
- Algoritmo:
  - Si leggono iterativamente i dati, controllando gli ultimi tre acquisiti (i primi due sono letti a parte)
    - Non appena |t2-t0|>5 e |t2-t1|>5, si ritorna 0
      - |t1-t0|>5 è già stata verificata
    - Se viene rispettato il criterio di ordinamento, aggiornano t0 e t1
    - Se non è mai stato violato il criterio si ritorna 1

# Codice

```
int verificaTemperature (FILE *fp) {
  float t0, t1, t2;

  fscanf(fp, "%f%f",&t0,&t1);
  if (abs(t1-t0)>5)
    return 0;
  while (fscanf(fp, "%f",&t2)==1) {
    if (abs(t2-t0)>5 || abs(t2-t1)>5)
      return 0;
    t0=t1;
    t1=t2;
  }
  return 1;
}
```

# Selezione di dati

- Contestualmente alla verifica di più dati (o sequenze/insiemi) di dati, è possibile discriminare i dati che corrispondono al criterio di verifica, rispetto agli altri
- La selezione può essere vista come una variante della verifica:
  - I dati vengono dapprima verificati
  - Quelli che corrispondono al criterio di accettazione vengono scelti
- La selezione è solitamente un processo iterativo:
  - Più dati (o insiemi di dati) vengono verificati
  - Una parte di questi viene selezionata

# Borse di studio

- Formulazione:
  - Un file testo contiene un elenco di studenti, per ognuno dei quali
    - Una riga riporta il numero di matricola (preceduto da '#')
    - La riga successiva riporta cognome e nome
    - Le righe successive riportano i voti di esami superati, uno per riga
  - Si scriva un programma C che
    - Legga il file e acquisisca da tastiera due numeri mmin (reale) e nmin (intero)
    - Selezioni gli studenti con media non inferiore a mmin e numero di esami superati non inferiore a nmin
    - Scriva nome e numero di matricola degli studenti selezionati su un secondo file (i nomi dei file sono acquisiti da tastiera)

# Borse di studio

- Soluzione:
  - Analizzare iterativamente i dati, considerando di volta in volta i dati relativi a uno studente, del quale occorre calcolare numero di esami e media
- Struttura dati: bastano variabili scalari:
  - 2 stringhe per nomi di file, cognome e nome e matricola: s0, s1
  - 2 puntatori a file (da aprire contemporaneamente, uno in lettura e uno in scrittura): fin, fout
  - 2 interi per il conteggio degli esami (ne), e soglia minima (nmin)
  - 3 reali per voto corrente (voto), media dei voti (media) e media minima (mmin)

# Borse di studio

- Algoritmo:
  - La verifica dei dati consiste in:
    - Lettura dei dati relativi a uno studente
    - Calcolo di numero esami e media dello studente
    - Confronto con le soglie minime previste : se vengono superate, scrivere su file
  - La soluzione consiste in una doppia iterazione:
    - Esterna (per ogni studente): acquisizione di matricola, cognome e nome
    - Interna (per ogni voto): calcolo numero esami e media
  - La separazione tra le informazioni relative a due studenti successivi si ottiene ricoscendo il carattere '#' (inizio sezione relativa a nuovo studente)

# Codice

```c
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingresso: "); scanf("%s",s0);
  printf("File uscita: "); scanf("%s",s1);
  fin = fopen(s0, "r"); fout = fopen(s1,"w");
  printf("soglie min. n. esami e media: ");
  scanf("%d%f", &nmin, &mmin);
  ...
```

```c
  while (fgets(s0,MAXL,fin)!=NULL) {
    fgets(s1,MAXL,fin);
    ne=0; media=0.0;
    while (fscanf(fin, "%f",&voto)==1) {
      media += voto; ne++;
    }
    media = media/ne;
    if (media >= mmin && ne>=nmin) {
      fprintf(fout, "%s%s",s0,s1);
    }
  }
  fclose(fin); fclose(fout);
}
```

# Codice

```
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingres
  printf("File uscita
  fin = fopen(s0, "r"
  printf("soglie min. n. esami e media: ");
  scanf("%d%f", &nmin, &mmin);
  ...
```

```
  while (fgets(s0,MAXL,fin)!=NULL) {
    fgets(s1,MAX ,fin);
    ne=0; media  ;
    while (fsc          "%f",&voto)==1) {
      media +=          ++;
    }
    media = r
  }
  fclose(fin); fclose(fout);
}
```

Legge riga contenente matricola
(senza verificare presenza di '#')

# Codice

```
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingres
  printf("File uscita
  fin = fopen(s0, "r"); fout = fopen(s1,"w");
  printf("soglie min. n. esami e media: ");
  scanf("%d%f", &nmin, &mmin);
  ...
```

```
  while (fgets(s0,MAXL,fin)!=NULL) {
    fgets(s1,MAXL,fin);
    ne=0   media=0.0;
    wh     (fscanf(fin, "%f",&voto)==1) {
           ia += voto; ne++;

           a = media/ne;
           = mmin && ne>=nmin) {
           out, "%s%s",s0,s1);
    }
  }
  fclose(fin); fclose(fout);
}
```

Legge cognome e nome

# Codice

```
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingresso: "); scanf("%s",s0);
  printf("File uscita: "); scanf("%s",s1);
  fin = fopen(s0, "r")
  printf("soglie min. 
  scanf("%d%f", &nmin,
  ...
```

```
while (fgets(s0,MAXL,fin)!=NULL) {
  fgets(s1,MAXL,fin);
  ne=0; media=0.0;
  while (fscanf(fin, "%f",&voto)==1) {
    media += voto; ne++;
  }
  media = media/ne;
  if (media >= mm                nmin) {
    fprintf(                    s0,s1);
  }
```

Acquisisce numeri fino a quando il carattere ‘#’ lo impedisce (fscanf restituisce 0)

# Codice

```
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingresso: "); scanf("%s",s0);
  printf("File uscita: "); scanf("%s",s1);
  fin = fopen(s0, "r"); fout = fopen(s1,"w");
  printf("soglie min. n. esami e media: ");
  scanf("%d%f", &nmin, &mmin);
  ...
```

```
  while (fgets(s0,MAXL,fin)!=NULL) {
    fgets(s1,MAXL,fin);
    ne=0; media=0.0;
    while (fscanf(fin, "%f",&voto)==1) {
      media += voto; ne++;
    }
    media = media/ne;
    if (media >= mmin && ne>=nmin) {
      fprintf(fout, "%s%s",s0,s1);
    }
  }
  fclose(fin); fclose(fout);
}
```

# Problemi di ordinamento

- Un problema di ordinamento consiste nella richiesta di permutare una sequenza di dati, in modo tale che (dopo la permutazione) sia verificato un criterio di ordinamento
- Per ordinare dei dati, occorre un operatore (o una funzione) di confronto tra coppie di dati, tale da decidere quale dei due precede l'altro secondo il criterio di ordinamento
  - E' ammessa l'uguaglianza tra dati, che consente ad ognuno di precedere l'altro
  - Di solito si usano operatori relazionali (<, <=, >, >=)

# Ordinamenti totali e parziali

- Ordinamento “totale”:
  - In base al criterio scelto, un dato precede (nella sequenza ordinata) tutti i successivi, e ogni dato è confrontabile con tutti gli altri
  - Esempio: ordinamento alfabetico tra nomi
- Ordinamento “parziale”:
  - Il criterio di confronto scelto non definisce una relazione di precedenza tra alcune coppie di dati
  - Esempi:
    - Ordinamento alfabetico di un insieme di nomi e numeri (l’ordinamento alfabetico non è definito sui numeri)
    - Ordinamento di nomi in base alla lettera iniziale (non è definito l’ordine relativo tra nomi con la stessa iniziale)

# Algoritmi di ordinamento “parziale”

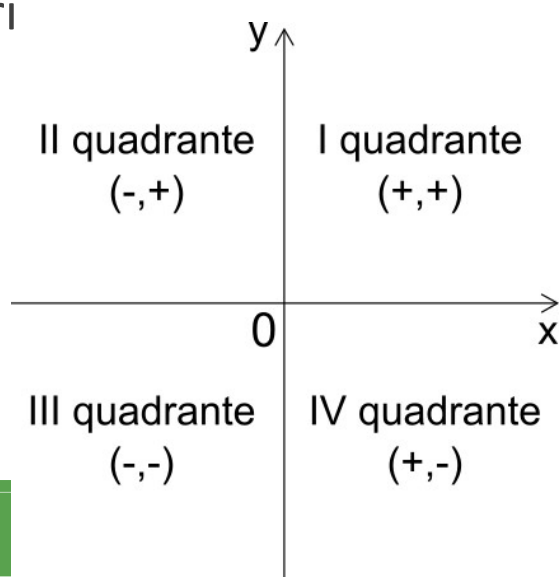
- La maggioranza degli algoritmi di ordinamento “totale” richiede l’utilizzo di vettori (per i passaggi intermedi)
  - Tali algoritmi saranno trattati con i vettori
- Semplici problemi di ordinamento “parziale” sono risolvibili mediante la ripetizione di problemi di selezione
- Sarà necessario esaminare più volte i dati: per evitare l’uso di vettori, si ripeterà più volte la lettura di un file
  - Soluzione poco efficiente ma necessaria per non usare i vettori

# Ordinamento di punti per quadrante

- Formulazione:
  - Un file testo contiene un elenco di punti del piano, ognuno rappresentato su una riga del file mediante due numeri reali (le coordinate)
  - Si scriva una funzione C che riscriva i punti su un secondo file dopo averli ordinati in base al quadrante di appartenenza
  - NB: i punti appartenenti agli assi cartesiani possono essere attribuiti indifferentemente ad uno dei quadranti adiacenti
  - I nomi di entrambi i file sono ricevuti come parametri

# Ordinamento di punti per quadrante

- Soluzione:
  - Leggere 4 volte il file di ingresso, selezionando ogni volta (e scrivendo sul file in uscita) i punti di un quadrante
- Struttura dati e algoritmo:
  - Risulta opportuno scrivere una funzione di selezione dei dati di un quadrante: tale funzione viene chiamata 4 volte
  - Il quadrante viene identificato mediante due parametri interi (uno per le ascisse e uno per le ordinate): +1 indica valori positivi, -1 indica valori negativi
    - I quadranti sono denotati dalle coppie (1,1), (-1,1), (-1,-1), (1,-1)



# Codice

```
void ordinaPunti (char nin[], char nout[]) {
  FILE *fin, *fout;
  fout = fopen(nout, "w");
  fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,-1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,-1);
  fclose(fin);
  fclose(fout);
}
```

```
void selezionaPunti (FILE *fi, FILE *fo,
                     int sx, int sy) {
  float x,y;
  int xOK, yOK;
  while (fscanf(fi, "%f%f",&x,&y)==2) {
    xOK = x*sx>0.0 || (x==0.0 && sx>0);
    yOK = y*sy>0.0 || (y==0.0 && sy>0);
    if (xOK && yOK)
      fprintf(fo, "%f %f\n",x,y);
  }
}
```

# Codice

```
void ordinaPunti (char nin[], char nout[]) {
  FI
  fo
  fi
  selezionaPunti(fin,fout,1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,-1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,-1);
  fclose(fin);
  fclose(fout);
}
```

Ascissa(ordinata) OK
se concorde con sx(sy)

```
void selezionaPunti (
  FILE *fi, FILE *fo, int sx, int sy) {
  float x,y;
  int xOK, yOK;
  while (fscanf(fi, "%f%f",&x,&y)==2) {
    xOK = x*sx>0.0 || (x==0.0 && sx>0);
    yOK = y*sy>0.0 || (y==0.0 && sy>0);
    if (xOK && yOK)
      fprintf(fo, "%f %f\n",x,y);
  }
}
```

# Codice

```
void ordinaPunti (char nin[], char nout[]) {
  FI
  fo
  fi
  selezionaPunti(fin,fout,1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,-1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,-1);
  fclose(fin);
  fclose(fout);
}
```

```
void selezionaPunti (
  FILE *fi, FILE *fo, int sx, int sy) {
  float x,y;
  int xOK, yOK;
  while (fscanf(fi, "%f%f",&x,&y)==2) {
    xOK = x*sx>0.0 || (x==0.0 && sx>0);
    yOK = y*sy>0.0 || (y==0.0 && sy>0);
    if (xOK && yOK)
      fprintf(fo, "
```

Ascissa(ordinata) OK se concorde con sx(sy)

Ascissa(ordinata) OK se nulla e sx(sy) è positivo (convenzionalmente 0 viene interpretato come positivo)