Fondamenti di Informatica T-1
Modulo 2

Contenuti

Scopo di questa esercitazione:

- Comprendere la complessità del problema “ordinamento”… cerchiamo di valutare il “costo” di una soluzione (confrontandola con un’altra…)
- Modifiche al codice degli algoritmi di ordinamento, per supportare tipi di dato “complessi”

Esercizio 1
(ordinamento)

Naive Sort con conteggio degli scambi e dei confronti

- Come facciamo a valutare la “bontà” di un algoritmo?
  - Idea: contiamo quante volte eseguiamo le operazioni “costose” di un algoritmo
  - Con Naïve Sort, le operazioni costose possono essere i *confronti* e gli *scambi*.

Esercizio 1
(ordinamento)

Naive Sort con conteggio degli scambi e dei confronti

- In un apposito modulo **`ordinamento.h`** / **`ordinamento.c`**, realizzare l'algoritmo Naive Sort contando quanti *confronti* e quanti *scambi* vengono effettuati

- Per comodità, definiamo i contatori come variabili *globali statiche*, in modo da potervi accedere da più funzioni

Esercizio 1 - Soluzione
(ordinamento)

```
#include <stdio.h>

int swap_count = 0;
int comp_count = 0;

void incSwap() { swap_count++; }
void incComp() { comp_count++; }

void resetCounters()
{
   swap_count = 0;
   comp_count = 0;
}

void printCounters()
{
   printf("Numero di confronti effettuati: %d\n", comp_count);
   printf("Numero di scambi effettuati: %d\n", swap_count);
}
```

Esercizio 1 - Soluzione
(ordinamento)

```
int trovaPosMax(int v[], int n) {
   int i, posMax=0;
   for (i=1; i<n; i++) {
        incComp();
        if (v[posMax]<v[i]) {
               posMax=i;
               //incComp(); // ????
        }
   }
   return posMax;
}
void scambia(int * a, int * b) {
   int temp;
   temp = *a;
   *a = *b;
   *b = temp;
   incSwap();
}
```

Esercizio 1 - Soluzione

(ordinamento)

```
void naiveSort(int v[], int n) {
   int p;

   while (n>1) {
        p = trovaPosMax(v,n);

        // incComp();  // ???
        if ( p < n-1 )
               scambia( &v[p], &v[n-1]);
        n--;
   }
}
```

Esercizio 1 - Soluzione
(ordinamento)

```
int main(void) {
   int v[] = {6,5,4,3,2,1};
   int i;

   naiveSort(v, 6);

   for (i=0; i<6; i++)
        printf("%d ", v[i]);

   printf("\n");
   printCounters();

   system("PAUSE");
   return (0);
}
```

Esercizio 2
(ordinamento)

- Implementare e modificare l'algoritmo Bubble Sort visto a lezione al fine di contare i confronti e gli scambi eseguiti

- Realizzare un programma che legga un vettore di MAXDIM elementi e ne esegua
  - L'ordinamento con Naive Sort
  - L'ordinamento con Bubble Sort

  e stampi a video il numero di confronti e scambi effettuato da ogni algoritmo

- Qual'è il caso "peggiore" per bubble sort?

Esercizio 2 - Soluzione
(ordinamento)

Per comodità:

```
void bubbleSort(int v[], int n) {
   int i;
   int ordinato = 0;

   while (n>1 && ordinato==0) {
        ordinato = 1;
        for (i=0; i<n-1; i++) {
               if ( v[i] > v[i+1] ) {
                      scambia( &v[i], &v[i+1]);
                      ordinato = 0;
               }
        }
        n--;
   }
}
```

Esercizio 2 - Soluzione
(ordinamento)

```
void bubbleSort(int v[], int n) {
   int i;
   int ordinato = 0;

   while (n>1 && ordinato==0) {
        ordinato = 1;
        for (i=0; i<n-1; i++) {
                incComp();
                if ( v[i] > v[i+1] ) {
                        scambia( &v[i], &v[i+1]);
                        ordinato = 0;
                }
        }
        n--;
   }
}
```

Esercizio 2 - Soluzione
(ordinamento)

```
… // riprendo parte dell'esercizio 1

int main(void) {
    int v[6];
    int i;

    for (i=0; i<6; i++) scanf("%d", &v[i]);
    naiveSort(v, 6);
    for (i=0; i<6; i++) printf("%d ", v[i]);
    printf("\n");
    printCounters();

    resetCounters();
    for (i=0; i<6; i++) scanf("%d", &v[i]);
    bubbleSort(v, 6);
    for (i=0; i<6; i++) printf("%d ", v[i]);
    printf("\n");
    printCounters();

    system("PAUSE");
    return (0); }
```

Esercizio 3
(ordinamento)

- Implementare e modificare l'algoritmo Insert Sort visto a lezione al fine di ordinare un array di float

- Realizzare un programma che legga un vettore di MAXDIM elementi di tipo float, lo ordini usando l'algoritmo InsertSort e stampi a video l'array ordinato.

Suggerimenti:

- Sarà necessario modificare i prototipi delle funzioni usate…
- Sarà necessario controllare che le operazione di confronto siano ancora effettivametne valide…
- Sarà necessario controllare che le operazioni di assegnamento siano *compatibili* col nuovo tipo…

Esercizio 3
(ordinamento)

Per comodità:

```
void insOrd(int v[], int pos) {
   int i = pos-1, x = v[pos];

   while (i>=0 && x<v[i]) {
        v[i+1]= v[i]; /* crea lo spazio */
        i--;
   }
   v[i+1]=x; /* inserisce l'elemento */
}

void insertSort(int v[], int n) {
   int k;
   for (k=1; k<n; k++)
        insOrd(v,k);
}
```

Esercizio 3 - Soluzione
(ordinamento)

```
void insOrdF(float v[], int pos) {
   int i = pos-1;
   float x = v[pos];

   while (i>=0 && x < v[i]) {
        v[i+1] = v[i]; /* crea lo spazio */
        i--;
   }
   v[i+1]=x; /* inserisce l'elemento */
}

void insertSortF(float v[], int n) {
   int k;
   for (k=1; k<n; k++)
        insOrdF(v,k);
}
```

Esercizio 4
(ordinamento)

Un sito web del turismo trentino tiene un elenco aggiornato delle stazioni sciistiche e del manto nevoso (in cm, un intero). Si deve realizzare un programma che chieda in ingresso, per MAXDIM località, il nome di una località (al più 20 caratteri senza spazi), e l'altezza del manto nevoso (un intero).

- A tal fine si definisca una apposita struttura dati **`Stazione`**
- Si definisca un array di MAXDIM elementi di tipo **`Stazione,`** e si chiedano all'utente i dati relativi a MAXDIM località (nome e neve), memorizzandoli nell'array
- Si realizzi una funzione **`compare(Stazione s1, Stazione s2)`** che restituisca **`-1, 0`** o **`1`** a seconda che il manto nevoso in **`s1`** sia rispettivametne minore, uguale o maggiore al manto nevoso in **`s2`**
- Si modifichi l'algoritmo Merge Sort visto a lezione, e lo si utilizzi per ordinare le località in base alla neve presente (suggerimento: si usi la funzione **`compare(…)`**)
- Si stampi a video l'elenco ordinato delle località

Esercizio 4

(ordinamento)

Per comodità:

```
void merge(int v[], int i1, int i2, int fine, int vout[]) {
    int i=i1, j=i2, k=i1;

    while ( i <= i2-1 && j <= fine ) {
          if (v[i] < v[j]) {
                    vout[k] = v[i];
                    i++;
          }
          else {
                    vout[k] = v[j];
                    j++;
          }
          k++;
    }
    while (i<=i2-1) {
          vout[k]=v[i];
          i++; k++;
    }
    while (j<=fine) {
          vout[k]=v[j];
          j++; k++;
    }
    for (i=i1; i<=fine; i++) v[i] = vout[i];
}
```

Esercizio 4
(ordinamento)

Per comodità:

```
void mergeSort(int v[], int iniz, int fine, int vout[]) {
   int mid;

   if ( iniz < fine ) {
        mid = (fine + iniz) / 2;
        mergeSort(v, iniz, mid, vout);
        mergeSort(v, mid+1, fine, vout);
        merge(v, iniz, mid+1, fine, vout);
   }
}
```

Esercizio 4 - Soluzione
(ordinamento)

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


typedef struct stazione {
   char nome[21];
   int neve;
} Stazione;

int compare(Stazione s1, Stazione s2) {
   if (s1.neve < s2.neve)
        return -1;
   else
        if (s1.neve == s2.neve)
                return 0;
   return 1;
}
```

Esercizio 4 - Soluzione
(ordinamento)

```
void mergeS(Stazione v[], int i1, int i2, int fine, Stazione vout[]) {
    int i=i1, j=i2, k=i1;

    while ( i <= i2-1 && j <= fine ) {
          if (compare(v[i], v[j]) < 0) {
                vout[k] = v[i];
                i++;
          }
          else {
                vout[k] = v[j];
                j++;
          }
          k++;
    }
    while (i<=i2-1) {
          vout[k]=v[i];
          i++; k++;
    }
    while (j<=fine) {
          vout[k]=v[j];
          j++; k++;
    }
    for (i=i1; i<=fine; i++) v[i] = vout[i];
}
```

Esercizio 4 - Soluzione
(ordinamento)

```
void mergeSortS(Stazione v[], int iniz, int fine, Stazione vout[]) {
    int mid;
    if ( iniz < fine ) {
          mid = (fine + iniz) / 2;
          mergeSortS(v, iniz, mid, vout);
          mergeSortS(v, mid+1, fine, vout);
          mergeS(v, iniz, mid+1, fine, vout);
    }
}

int main(void) {
    Stazione v[3];
    Stazione temp[3];
    int i;

    for (i=0; i<3; i++) {
          printf("nome: ");
          scanf("%s", v[i].nome);
          printf("Neve: ");
          scanf("%d", &(v[i].neve));
    }
    mergeSortS(v, 0, 2, temp);
    for (i=0; i<3; i++) printf("%s: %d\n", v[i].nome, v[i].neve);
    system("PAUSE"); return (0); }
```

Esercizio 5
(ordinamento)

- Implementare e modificare gli algoritmi Insert Sort, Merge Sort e Quick Sort visti a lezione al fine di contare i confronti e gli scambi eseguiti

- Realizzare un programma che legga un vettore di MAXDIM elementi e ne esegua l'ordinamento con gli algoritmi di cui al punto precedente, e stampi a video il numero di confronti e scambi effettuato da ogni algoritmo

Esercizio 5 - Soluzione
(ordinamento)

```
void insOrd(int v[], int pos) {
    int i = pos-1, x = v[pos];
    while (i>=0 && x<v[i]) {
          v[i+1]= v[i]; /* crea lo spazio */
          incSwap(); // non proprio corretto: qui non ho uno swap, ma
          i--;       // solo un assegnamento...
          incComp(); // potrebbe "difettare" di 1...
    }
    v[i+1]=x; /* inserisce l'elemento */
    // incSwap(); // ancora ???
}


void insertSort(int v[], int n) {
    int k;
    resetCounters();
    for (k=1; k<n; k++)
          insOrd(v,k);
}
```

Esercizio 5 - Soluzione
(ordinamento)

```
void mergeSort(int v[], int iniz, int fine, int vout[]) {
   int mid;

   resetCounters();
   if ( iniz < fine ) {
        mid = (fine + iniz) / 2;
        mergeSort(v, iniz, mid, vout);
        mergeSort(v, mid+1, fine, vout);
        merge(v, iniz, mid+1, fine, vout);
   }
}
```

Esercizio 5 - Soluzione
(ordinamento)

```
void merge(int v[], int i1, int i2, int fine, int vout[]) {
    int i=i1, j=i2, k=i1;

    while ( i <= i2-1 && j <= fine ) {
           incComp();
           incSwap(); // non corretto… ho solo assegnamento, e non swap…
           if (v[i] < v[j]) {
                      vout[k] = v[i];
                      i++;
           }
           else {
                      vout[k] = v[j];
                      j++;
           }
           k++;
    }
    while (i<=i2-1) {
           incSwap(); // non corretto…
           vout[k]=v[i];
           i++; k++;
    }
    while (j<=fine) {
           incSwap(); // non corretto…
           vout[k]=v[j];
           j++; k++;
    }
    for (i=i1; i<=fine; i++) v[i] = vout[i];
}
```

Esercizio 6

(riepilogo su ordinamento)

“Astrazione” degli algoritmi di ordinamento

- Implementare i diversi algoritmi di ordinamento, facendo in modo di ***astrarre completamente dal tipo*** degli elementi del vettore
- Fare anche in modo che vengano stampate delle ***statistiche sul numero di confronti e di scambi*** effettuati
- Validare la soluzione su un vettore di interi, un vettore di caratteri, un vettore di stringhe

Esercizio 6

(riepilogo su ordinamento)

- Quali sono le istruzioni utilizzate in fase di ordinamento che dipendono dal TIPO dell'elemento?
  - Confronto tra due elementi
  - Assegnamento di un elemento a un altro elemento
  - Swap?
    - dipende dal tipo a causa degli assegnamenti effettuati
    - quindi ci riconduciamo al caso precedente

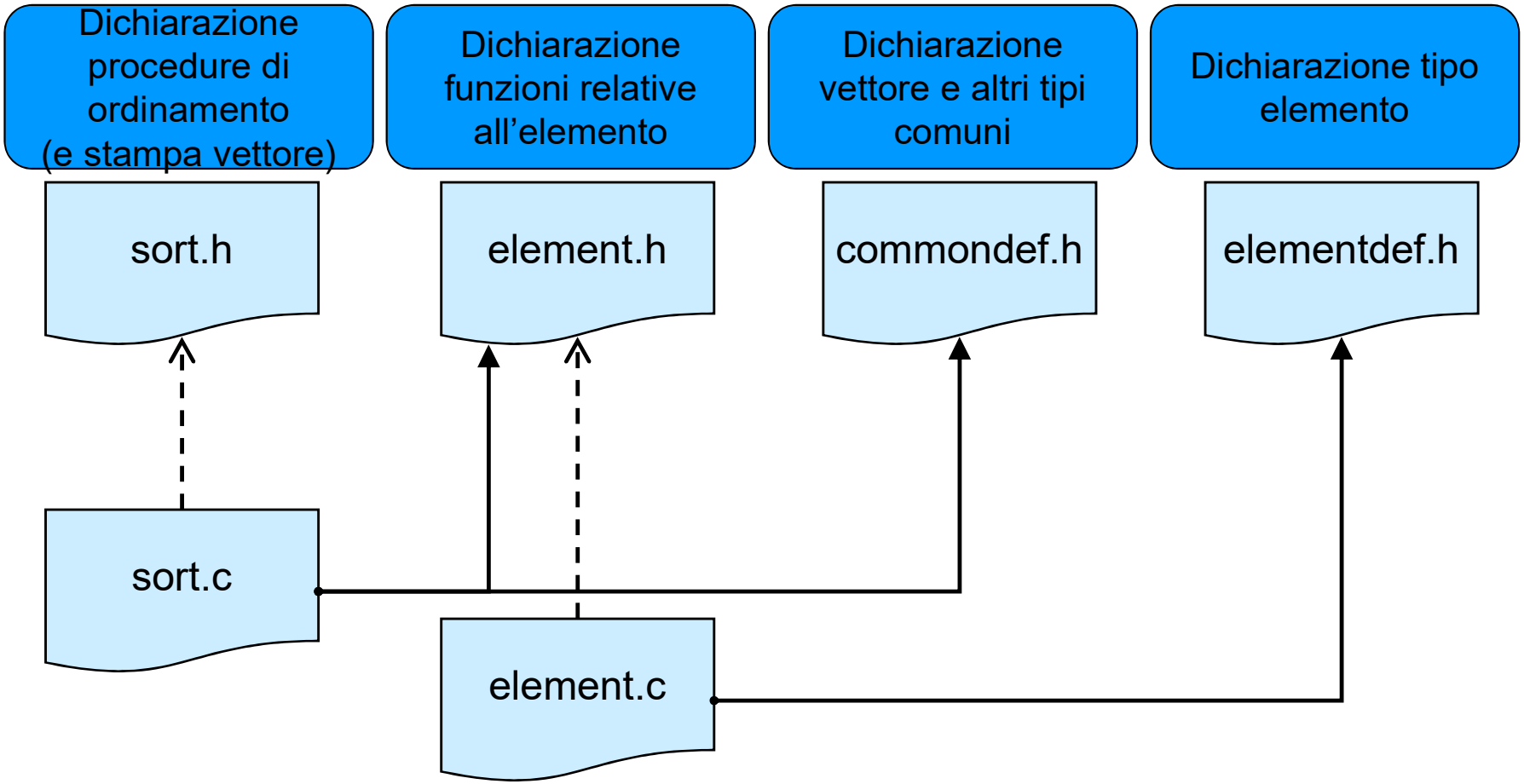
Esercizio 6
(riepilogo su ordinamento)

- Quindi dobbiamo sostituire
  - Confronti
  - Assegnamenti
- …con delle funzioni capaci di eseguire il confronto e l'assegnamento

```
int compare(Element e1, Element e2);
void assign(Element *lvalue, Element rvalue);
```

Esercizio 6

(riepilogo su ordinamento)



Esercizio 6

(riepilogo su ordinamento)

- **`elementdef.h`**
  - Contiene la dichiarazione typedef … Element;
- **`element.h`**
  - Contiene le dichiarazioni delle funzioni per manipolare un elemento
- Quindi se cambio tipo devo aggiornare unicamente
  - **`elementdef.h`**
  - **`element.c`** (l'header rimane uguale, cambia l'implementazione in base al tipo)

Esercizio 6

(riepilogo su ordinamento)

Contenuto di element.h

- **`int compare(Element e1, Element e2);`**
  - Restituisce un numero negativo se **`e1 < e2`**, 0 se **`e1 == e2`**, un numero positivo se **`e1 > e2`**
- **`void swap(Element *e1, Element *e2);`**
  - Scambio elementi (utilizzando assign!!!)
- **`void assign(Element *lvalue, Element rvalue);`**
  - Assegna il contenuto di **`rvalue`** a **`lvalue`**
- **`void printElement(Element e);`**
  - Stampa l'elemento a video
- **`void printStatistics();`**
  - Stampa le statistiche relative a confronti e scambi
  - Suggerimento: utilizzare due variabili contatore globali

Esercizio 6 - Soluzione

(riepilogo su ordinamento)

elementdef.h

- Nel caso di interi…

```
#ifndef ELEMENTDEF
#define ELEMENTDEF
    typedef int Element;
#endif
```

- Nel caso di stringhe…

```
#ifndef ELEMENTDEF
#define ELEMENTDEF
    typedef char* Element;
#endif
```

Esercizio 6 - Soluzione

(riepilogo su ordinamento)

element.h

```
#include "elementdef.h"

#ifndef ELEMENT
#define ELEMENT
  int compare(Element e1, Element e2);
  void swap(Element *e1, Element *e2);
  void assign(Element *lvalue, Element rvalue);
  void printElement(Element e);
  void printStatistics();
#endif
```

Esercizio 6 - Soluzione

(riepilogo su ordinamento)

element.c

```
#include "elementdef.h"
#include <stdio.h>
#include <string.h> //se il tipo è stringa…

int compareCounter = 0;
int swapCounter = 0;
void incrementCompareCounter()
{
   compareCounter++;
}

void incrementSwapCounter()
{
   swapCounter++;
}
```

Esercizio 6 - Soluzione

(riepilogo su ordinamento)

element.c

```
void assign(Element *lvalue, Element rvalue)
{
  *lvalue = rvalue;
}

void swap(Element *e1, Element *e2)
{
  Element tmp;
  assign(&tmp, *e1);
  assign(e1, *e2);
  assign(e2, tmp);
  incrementSwapCounter();
}
```

Esercizio 6 - Soluzione

(riepilogo su ordinamento)

element.c

- Nel caso di interi…

```
int compare(Element e1, Element e2)
{
     incrementCompareCounter();
     return e1 - e2;
}
void printElement(Element e)
{
     printf("%d\n", e);
}
```

- Nel caso di stringhe…

```
int compare(Element e1, Element e2)
{
     incrementCompareCounter();
     return strcmp(e1, e2);
}
// e ovviamente printElement con %s
```

Esercizio 6 - Soluzione

(riepilogo su ordinamento)

commondef.h – sort.h

- commondef.h

```
#ifndef COMMONDEF
#define COMMONDEF
#define DIM 20
    typedef Element Array[DIM];
#endif
```

- sort.h

```
void printArray(Array a, int dim);
void naiveSort(Array a, int dim);
// e tutti gli altri tipi di sort…
```

Esercizio 6 - Soluzione

(riepilogo su ordinamento)

sort.c

```
#include "elementdef.h"
#include "commondef.h"
#include "element.h"
#include <stdio.h>

void printArray(Array a, int dim)
{
  int i;
  printf("---VETTORE---\n");
  for(i = 0; i < dim; i++)
     printElement(a[i]);
  printf("-------------\n");
}
```

Esercizio 6 - Soluzione

(riepilogo su ordinamento)

sort.c

```
void naiveSort(Array a, int dim) {
   int j, i, posmin;
   Element min;

   for (j = 0; j < dim; j++) {
        posmin = j;
        for ( assign(&min, a[j]), i = j + 1; i < dim; i++) {
               if(compare(a[i], min) < 0) {
                      posmin = i;
                      assign(&min, a[i]);
               }
        }
        if (posmin != j)
               swap(&a[j], &a[posmin]);
   }
}
```