Esercitazione 4

Practice make perfect

Programma:

- Pattern Matching
- Algoritmi di ordinamento
- Scoping delle funzioni

1-2

Pattern matching

Es 1.13.6

PROBLEMA *(importante)*
Scrivere una funzione che verifica se un array P (pattern) e' contenuto nell'array T (testo).

verifica se P e' contenuto in T a partire dalla posizione inizio;

match:
P[0, ...dimP] = T[inizio, ... , inizio+dimP]

scorri T, cercando
un match a partire da T[0],
poi a partire da T[1], etc.
fino a trovare o
fino all'ultima posizione utile.

1-4

```
//verifica se P e' contenuto in T a partire dalla posizione inizio
//Pre-condizione: inizio + dim P <= dim T
bool match_da (char T[], char P[], int dimP, int inizio ) {
        bool ok = true;

        // i primi i elementi sono uguali:
        // P[0 ... i-1] = T[inizio ... inizio+i-1]
        for (int i=0;i<dimP && ok ; i++ )
             if (P[i]!=T[inizio+i])  ok=false;
        return ok;}
```

```
//verifica se il pattern P e' contenuto nell'array T
bool trova_match (char T[], int dimT, char P[], int dimP){
        bool trovato= false;

        //non ho trovato match  da alcun indice prima di inizio
        for (int inizio=0; inizio <=(dimT - dimP) && !trovato ;inizio++)
                        if (match_da (T, P, dimP, inizio) )
                           trovato = true;
        return trovato; }
```

1-5

```
//verifica se P e' contenuto in T a partire dalla posizione inizio
//Pre-condizione: inizio + dim P <= dim T
bool match_da (char T[], char P[], int dimP, int inizio ) {

        //P[0 ... i-1] = T[inizio ... inizio+i-1]
         for (int i=0;i<dimP ; i++ )
                  if (P[i]!=T[inizio+i])
                     return false;

        return true;}
```

BIS

1-6

Nota che trova_match ha la solita struttura della funzione trova:

```
//trova il carattere ch nell'array A
bool trova (char A[], int dim, char ch, ){
        bool trovato = false;

        //prima di i non ci sono occorrenze di ch
        //uscita: trovato e A[i] = ch, opp. non trovato e i=dim
        for (int i=0; (i<dim && !trovato); i++)
                        if (A[i]== ch)
                           trovato = true;
        return trovato;
}
```

Nota: usare un booleano per uscire aiuta a ragionare sul ciclo

1-7

PROBLEMA *(v. semplice)*

dati due array P (pattern) e T,
verifica se tutti gli elementi di P sono presenti in T,
in posizione qualsiasi.

verifico se trovo P[0] in T,
poi se trovo P[1], etc...

fino a ...?

Scorro P
Ad ogni P[i], chiamo trova(T,P[i])

1-8

```
// verifica se il carattere ch e' presente nell'array A
bool trova (char A[], dim, char ch);
```

```
//verifica se tutti gli elementi di P sono presenti in T
bool tutti_presenti (char T[], int dimT, char P[], int dimP){

        // i primi i elementi  di P stanno in T
        for (int i=0;i<dimP; i++ )
                if (! trova(T, dimT, P[i]))
                return false;
        return true;}
```

BIS

```
bool tutti_presenti2 (char T[], int dimT, char P[], int dimP){
  bool ok=true;  //fin qui tutto bene

  //i primi i elementi  di P stanno in T
  for (int i=0;i<dimP && ok ;i++ )
      if (! trova(T, dimT, P[i]))
          ok = false;       //ok = trova (....)
  return ok;}
```

1-9

Algoritmi di ordinamento

PROBLEMA:
dato un array, scrivere una funzione che lo ordina in modo crescente:
A[0]<=A[1]<=A[2] <= .... <=A[dim-1]

Il modo piu' semplice di disegnare un algoritmo e' partire dalla definizione del problema...

IDEA: metto al primo posto il valore minimo di A,
al secondo il secondo piu' piccolo valore di A,
etc...

for (int i=0 ; i<dim; i++)
    metti l'i-esimo piu' piccolo elemento alla posizone i

1-11

Che input? array
Che output? array

Trucco: separo l'array in due parti.
A sin. ho l'array ordinato, a dx i valori da ordinare:

A[0, ..., top-1] A[top, ... dim-1]

array gia' ordinato array restante

Idea: cerco il minimo in A[top, ... dim-1]
e lo scambio con A[top].

FINE: quando top=dim-1

1-12

SELECTION SORT (l'algo piu' semplice)

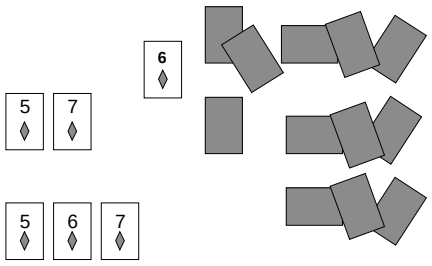
```
//calcola l'indice del minimo in A[top, ..., dim-1]
int indice_min (int A[], int top, int dim);


void ordina (int A[], int dim) {
        for (int top=0; top<dim-1 ;i++ )
                {int i_min = indice_min (A,top,dim);
                 scambia (A[top],A[i_min]); }
}
```

Sempre vero: A[0 ... i-1] e' ordinato
Esco con: i=dim

1-13

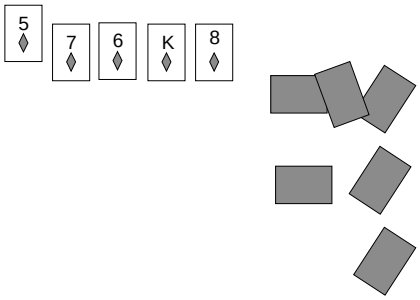
Insertion sort

Come ordinare le carte



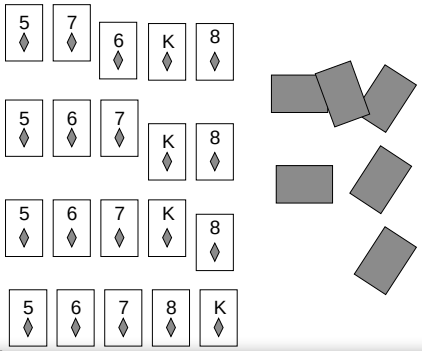
1-15

Ordinare una mano di carte



1-16

Ordinare una mano di carte



1-17

L'idea

Inserisci la j-esima carta

Prendi una carta N

Trova la posizione in cui va inserita

Inserisci N:
- Fai posto (sposta a destra)
- Metti N al suo posto

1-18

```
//top indica la prima posizione libera in array
for (top = 0; top<dim && ... ; top++)
{   int N;
    cin>> N;
    int pos = trova_pos(A, top , N);
    sposta_a_dx(A, pos, top)
    A[pos] = N
}
```

Inserisce una carta

SEMPRE VERO: A[0, ... top-1] ordinato
ESCE CON: top = dim,
quindi A[0, ..., dim-1] ordinato

1-19

```
int trova_pos (int A[], int top, int N) {
      int i = 0;
      while (i<top &&  A[i]<N)  i=i+1;
      return i;}

//Prima della posizione i, tutti gli elementi sono < N
// Sempre vero: A[i-1] < N
//Uscita: A[i]>=N oppure i=top.
//In entrambi i casi N va messo in A[i]


void sposta_a_dx (int A[], int inizio, int top) {
      for(int i=top; i>inizio; i--)
            A[i]=A[i-1];}
//sposta  a destra, lasciando un vuoto in A[pos]
```

1-20

Insertion sort per **ordinare un array**

Che input? array
Che output? stesso array

Trucco: divido l'array in due parti

- A[0 , ... , top-1] e' la parte ordinata
- A[top, ... , dim-1] sono i dati da inserire

Inizio: top =1 (perche' A[0] gia' ok)
Fine: top = dim

1-21

```
void insertion (int A[], int dim)
 //L'array e' diviso in due:
 //A[0 ...  top-1] e'  ordinato (e ha dimensione top)
 //A[top ... dim] sono i valori da inserire

 for (top = 1; top<dim; top++)
 {   int N= A[top];                    Prendi N
     int pos = trova_pos(A, top , N);  Trova dove inserire N
     sposta_a_dx(A, pos, top);         Fai posto (sposta a dx)
     A[pos] = N                        Inserisci N
 }
```

Insertion of *top*th card

1-22

fatto tutto insieme

```
void insertion (int A[], int dim)
 //L'array e' diviso in due:
 //A[0 ...  top-1] e'  ordinato (e ha dimensione top)
 //A[top ... dim] sono i valori da inserire
 for (top = 1; top<dim; top++)
 {   int N= A[top];
     int i = 0;
     while (i < top && A[i]<N)
          i=i+1;                       } Trova il posto dove inserire N

     for (int k = top; k>i; k--)
          A[k] = A[k-1]                } Fai posto (sposta a dx)

     A[i] = N                          Inserisci N
 }
```

Insertion of *top*th card

1-23

idea dell' insertion sort:	idea del selection sort:
•prendi il prossimo •mettilo al suo posto	•scegli l'elemento (min) • mettilo all'inizio
scegliere: costo costante mettere a posto: lineare	scegliere: costo lineare mettere: costante

1-24

Attenzione!
Ben distinguere tra:

- Verificare se un array e' in ordine
- Ordinare un array.

1-25

Morale

un compito complesso e' realizzato mettendo insieme operazioni semplici

Nella cassetta degli attrezzi:

Cerca il min/max

Cancella da un array l'elemento alla posizione i:

Sposto tutto a sin, da … a …

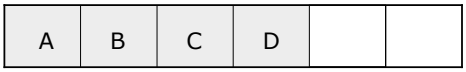
Inserisci in un array un elemento x alla posizione i:

- Sposto tutto a dx, da … a …
- Inserisco x.

(inizio da destra!)

1-27

Inserisci X alla posizione pos=1



1-28

Inserisci X alla posizione pos=1 **sposta a dx**

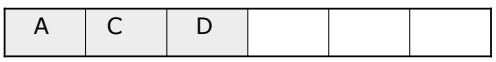
inserisci


```
for (int i=top;  i>inizio; i--)
      A[i]=A[i-1];
```

1-29

Cancella l'elemento alla posizione 1

```
//sposta a sin
for (int i= inizio; i< top; i++)
      A[i]=A[i+1];
```



1-30

```
void cancella (char A[], int pos, int & top){
//sposta a sin
for (int i= pos; i< top; i++)
      A[i]=A[i+1];
 top = top -1;  //aggiorna la dimensione
 }
```

```
void inserisci (char A[], char x, int pos, int & top){
//sposta a dx
for (int i=top;  i>pos; i--)
      A[i]=A[i-1];
 top = top+1;  // aggiorna la dimensione
  A[pos]=x;  // inserisci
 }
```


Scope e Flusso di controllo

scope.c

Scope e Flusso di controllo

```
#include<iostream>
using namespace std;

int x=10;
void F(int);
void G(int);

main() {
   int x=3;
   cout << "Sono main() ";
   cout << "qui x vale " << x << endl;
   cout <<"Ora chiamo F(), con parametro attuale x " <<
   endl;
   F(x);
   cout << "Sono di nuovo main()"<<endl;
}
```


e se il parametro formale lo chiamavo x ??

```
void G(int q) {
   cout <<"Sono G().Il mio parametro vale" << q;
   cout << "mentre qui x vale " << x << endl;
   return;
}

void F (int p) {
    cout<<"Sono F(). Il mio parametro p vale"<< p;
    cout << " mentre qui x vale " << x <<endl;
    cout << "Ora chiamo G() con parametro attuale 5 " <<endl;
    G(5);
    cout << "Sono di nuovo F()"<< endl;
    return;
}
```


Output:

Sono main() qui x vale 3

Ora chiamo F(), con parametro 3

Sono F(). Il mio parametro p vale 3
mentre qui x vale 10

Ora chiamo G(), con parametro
attuale 5

Sono G(). Il mio parametro vale 5
mentre qui x vale 10

5 !!

Sono di nuovo F()

Sono di nuovo main()

Capire gli array
per capire l'aritmetica
(e viceversa)

… fatti della vita

A e' un puntatore

A[0] = *A

A[1] = *(A+1)

A[i] = *(A+i)

& A[0] = A

& A[1] = A+1

& A[i] = A+i

1-37

abbiamo:
int M[][4];

Che differenza tra...

1. * M
2. M[0]
3. & M[0][0]

Qual e' il puntatore al primo elemento della riga i ?

- &M[i][0]
- M[i]

1-38

Cosa e' questo?

int A[10] [365] [31] [4] [8]

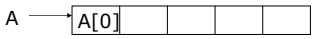
Un array di int [365][31][4][8]

1-39

Puntatori e Array

int A [5]; array di 5 elementi di tipo

1dim



A punta ad A[0] quindi

A ha tipo int *

*A = A[0] ha tipo int

A+1 = &A[1] ha tipo int *

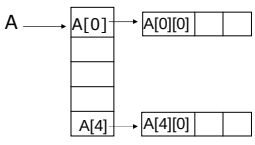
1-40

int A [5] [3];

come si legge??
…sfoglia il tipo dall'interno:…

A è un array di 5 elementi di tipo → int -- [3]

2dim



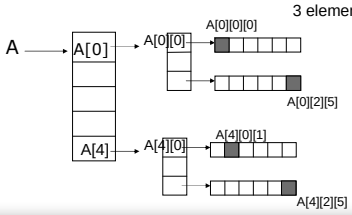
1-41

int A [5] [3] [6];

come si legge??
…sfoglia il tipo dall'interno:…

A è un array di 5 elementi di tipo → int -- [3] [6]

A[i] sono array di 3 elementi di tipo → int --[6]



3 dim

1-42