

C++ · Esercizi 2011

Fondamenti
di informatica

Michele Tomaiuolo

tomamic@ce.unipr.it

<http://www.ce.unipr.it/people/tomamic>



Esercitazione 1

- Blocchi condizionali
- Cicli semplici



Ken Thompson, Dennis Ritchie (Unix)

1.1 Cerchio

- Chiedere all'utente il valore del raggio r di un cerchio
- Mostrare il valore dell'area e della circonferenza

```
3.141592653589793238462643383279
5028841971693993751058209749445923
07816406286208998628034825342117067
9821      48086      5132
823      06647      09384
46      09550      58223
17      25359      4081
          2848      1117
          4502      8410
          2701      9385
          21105     55964
          46229     48954
          9303      81964
          4288      10975
          66593     34461
          284756    48233
          78678     31652      71
          2019091   456485      66
          9234603   48610454326648
          2133936   0726024914127
          3724587   00660631558
          817488    152092096
```

Definire una costante $PI = 3.14159$

1.2 Condizioni

- Chiedere all'utente tre numeri interi: a , b , c
- Determinare qual è il minore dei tre

Controllare prima di tutto se a è minore degli altri due, altrimenti controllare quale è il minore tra b e c

1.3 Cubi, ciclo

- Leggere, attraverso un ciclo, una sequenza di numeri
- Per ciascun numero immesso, mostrare il suo valore al cubo
- La sequenza termina quando l'utente immette il valore 0



Provare ad usare sia il ciclo while che il ciclo do-while

1.4 Caratteri ASCII

- Leggere un carattere
- Determinare se è una cifra, una lettera minuscola, oppure una lettera maiuscola



Nei confronti non usare esplicitamente i codici ASCII, ma simboli tra apici singoli ('a' , 'z' ecc.)

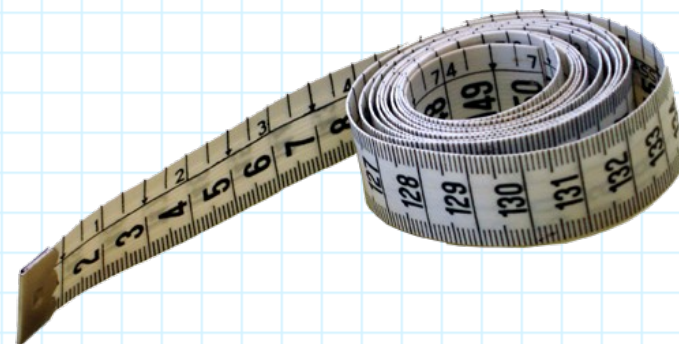
1.5 Aritmetica

- Chiedere all'utente due valori interi a , b
- Mostrare il risultato delle seguenti operazioni:
 - $a + b$
 - $a - b$
 - $a * b$
 - a / b
 - $a \% b$



Ripetere le prime 4 operazioni usando dei float

1.6 Lunghezza parole



- Leggere una sequenza di parole, attraverso un ciclo,
- La sequenza termina quando l'utente immette la parola "end"
- Visualizzare la lunghezza media delle parole

*Lunghezza di una variabile `word`
di tipo stringa: `word.size()`*

Provare ad usare sia il ciclo `while` che il ciclo `do-while`



1.7 Fattoriale

- Leggere un numero intero positivo n
- Calcolare il fattoriale del numero

Moltiplicare tra loro i primi n numeri

*Memorizzare in una variabile il risultato parziale,
ad ogni ciclo, moltiplicarla per il nuovo numero*

Esercitazione 2

- Cicli annidati
- Numeri pseudo-casuali
- Testo



Bjarne Stroustrup (C++)

2.1 Conversione gradi

- Visualizzare la tabella di conversione tra gradi centigradi e Fahrenheit
 - Intervallo tra 0 e 100°C, a passi di un grado
 - Scrivere una coppia di valori su ogni riga

$$\text{Conversione: } F = 32 + (9/5) \cdot C$$

```
cout << setw(6) << fixed  
<< setprecision(2) << x
```

x: almeno 6 posti, 2 decimali



2.2 Triangolo di cifre

- Leggere un numero intero positivo n
- Per ciascun valore x tra 1 ed n ...
- Stampare una riga con x ripetizioni di x

```
1  
2 2  
3 3 3  
4 4 4 4
```

Usare due cicli for annidati

*All'inizio non considerare n , ma fissare x
e scrivere una sola riga: $x = 3 \rightarrow \text{"333"}$*

Poi racchiudere tutto in un ciclo for esterno

2.3 Testo, lettere

- Leggere una riga di testo
 - `string line; getline(cin, line);`
- Contare il numero di lettere maiuscole ed il numero di lettere minuscole presenti

`line.size()` è la lunghezza di `line`
`line[i]` è il char in posizione `i`
Attenzione: indice da 0 a size-1

2.4 Lancio dadi

- Due giocatori lanciano un dado a testa
- L'attaccante vince solo se ottiene un numero strettamente maggiore del difensore
- Simulare 100 tentativi di attacco e stimare la probabilità di successo dell'attaccante

Dado (pseudo-casuale): `1+rand() % 6`

Per valori diversi, eseguire all'inizio:

`srand(time(NULL))`

Librerie: `<cstdlib>`, `<ctime>`

2.5 Interesse composto

- Calcolare l'interesse composto di un certo *capitale* ad un certo *tasso* d'interesse, per un certo numero di *anni* (dati dall'utente)
 - Es. per 100€ di capitale, l'interesse (4,5%) del primo anno sarà di 4,5€, da aggiungere al capitale
 - | Anni | Capitale |
|------|----------|
| 0 | 100.00 |
| 1 | 104.50 |
| 2 | 109.20 |
| 3 | 114.12 |
| 4 | 119.25 |
| ... | |



2.6 Maiuscolo

- Leggere un testo da tastiera
- Riscriverlo a console, ma convertito in maiuscolo
- Trasformare le sequenze di punteggiatura e spazi in un singolo trattino



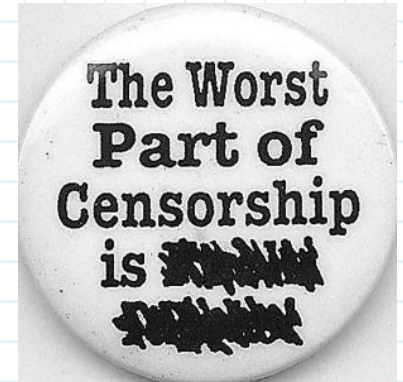
Es. "It's 2011... yet" → "IT-S-2011-YET"

*Lettere minuscole vengono dopo le maiuscole:
sottrarre la costante 'a' - 'A'*

Segnare in un bool se l'ultimo char scritto è un trattino

2.7 Filtro caratteri

- Leggere da tastiera una riga di testo
- Leggere da tastiera una stringa di caratteri da filtrare
- Visualizzare in output la riga di testo, ma con i caratteri da filtrare sostituiti da una 'X'



Usare due cicli for annidati

Per ogni carattere della riga di testo, controllare se è diverso da tutti i caratteri della stringa di filtro

Stampare quindi il carattere stesso, oppure una 'X'

Esercitazione 3

- File
- Array
- Matrici



Richard Stallman (GNU, FSF, Emacs, gcc)

3.1 File, parentesi

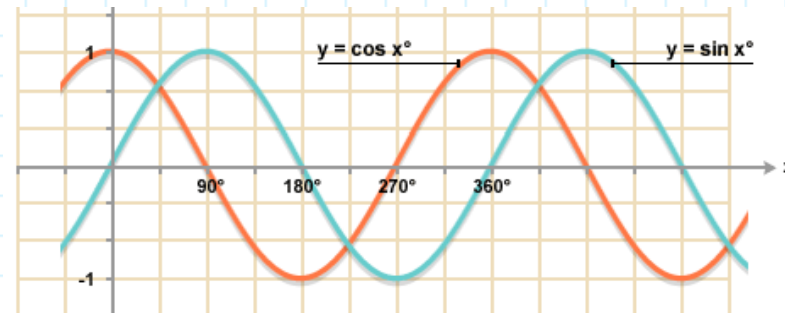
- Leggere da file un carattere alla volta
- Riscrivere il testo a console, ma...
- Escludere il testo tra parentesi quadre ' [', '] '

*Segnare in un bool se
si è letta una quadra aperta, ma
non ancora una quadra chiusa*

*Se necessario, provare prima
a leggere i caratteri da console*

3.2 Array, precalcolo

- Riempire un array con i valori di $\sin(x)$
 - 360 elementi, indice intero tra 0 e 359
 - \sin in libreria `<cmath>`, opera su radianti
 - Ottimizzazione opz.: basta calcolare $\sin(x)$ per ogni x intero tra 0° e 90° e poi sfruttare le simmetrie di \sin
- Chiedere ripetutamente un angolo all'utente, visualizzare il corrispondente valore precalcolato
- Scrivere in un file tutti i valori dell'array



3.3 Vettore

- Leggere da file una sequenza di numeri, di lunghezza arbitraria, fino alla fine del file
 - Tutti i numeri in un vettore $v \rightarrow n = v.size()$
- Ripetutamente, chiedere all'utente una coppia di numeri, a e b , compresi tra 1 ed n
- Visualizzare la somma dei valori della sequenza iniziale con indice compreso tra $a-1$ e $b-1$

L'utente preferisce indici tra 1 ed n

Invece il vettore richiede indici tra 0 e $n-1$

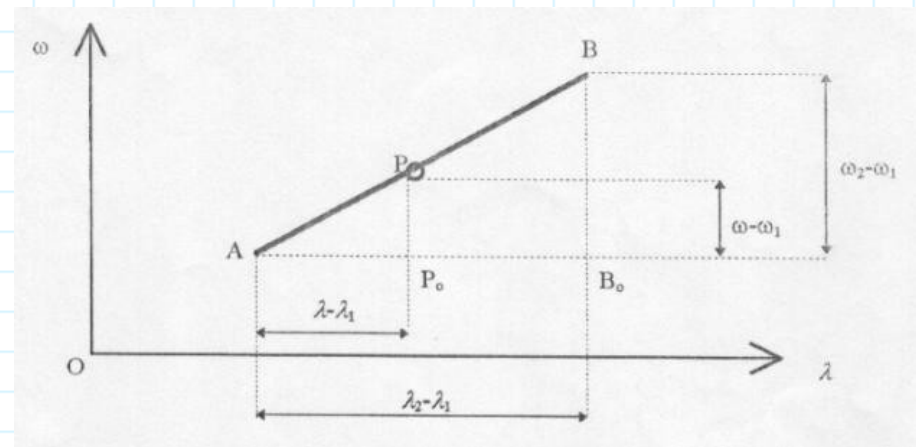
3.4 Matrice, coppie

- Allocare una matrice, dimensione `rows×cols`
 - `rows` e `cols` scelti dall'utente, ma celle in numero pari
 - `vector< vector<char> >`
`matrix(rows, vector<char>(cols));`
 - `vector<char> matrix(rows*cols); // pseudo...`
- Inserire in ordine le prime lettere dell'alfabeto, ciascuna ripetuta due volte
- Ripetutamente...
scegliere due celle a caso
e scambiare il contenuto
- Mostrare la matrice risultante



3.5 Interpolazione

- Leggere da un file una serie di numeri reali e memorizzarli in un vettore
 - Es. dati dall'esercizio 3.2
- Supporre che i numeri rappresentino il valore di una certa funzione $f(x)$, per valori interi di x
- Supporre f periodica, periodo = lunghezza del vettore
- Usare l'interpolazione (e la periodicità) per calcolare $f(x)$ per qualsiasi x reale



3.6 Map, record personali

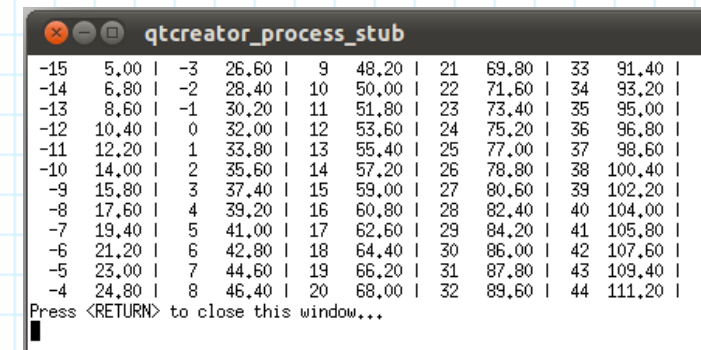
- Leggere da un file la lista di record personali di un gioco, memorizzati in ordine arbitrario
- Su ogni riga:
 - Nome giocatore (testo, anche con spazi)
 - Poi simbolo separatore ' : '
 - Infine punteggio (intero)
- Ciclicamente, chiedere all'utente un nome e mostrargli il punteggio

Usare una map
getline(file, line, ':')



3.7 Dati su più colonne

- Visualizzare la tabella di conversione °C/°F
 - Temperature tra -15 e 44°C
 - 5 coppie °C/°F per ogni riga
- Riempire la prima colonna con tutte le temp. più basse, poi continuare nella seconda colonna...



-15	5,00	-3	26,60	9	48,20	21	69,80	33	91,40
-14	6,80	-2	28,40	10	50,00	22	71,60	34	93,20
-13	8,60	-1	30,20	11	51,80	23	73,40	35	95,00
-12	10,40	0	32,00	12	53,60	24	75,20	36	96,80
-11	12,20	1	33,80	13	55,40	25	77,00	37	98,60
-10	14,00	2	35,60	14	57,20	26	78,80	38	100,40
-9	15,80	3	37,40	15	59,00	27	80,60	39	102,20
-8	17,60	4	39,20	16	60,80	28	82,40	40	104,00
-7	19,40	5	41,00	17	62,60	29	84,20	41	105,80
-6	21,20	6	42,80	18	64,40	30	86,00	42	107,60
-5	23,00	7	44,60	19	66,20	31	87,80	43	109,40
-4	24,80	8	46,40	20	68,00	32	89,60	44	111,20

Console usata sempre come macchina da scrivere (→, ↓)

Ma $\forall (x, y)$ si calcola la temperatura °C: $t=y+x\cdot h-15$

Bastano due cicli for annidati, senza array

Esercitazione 4

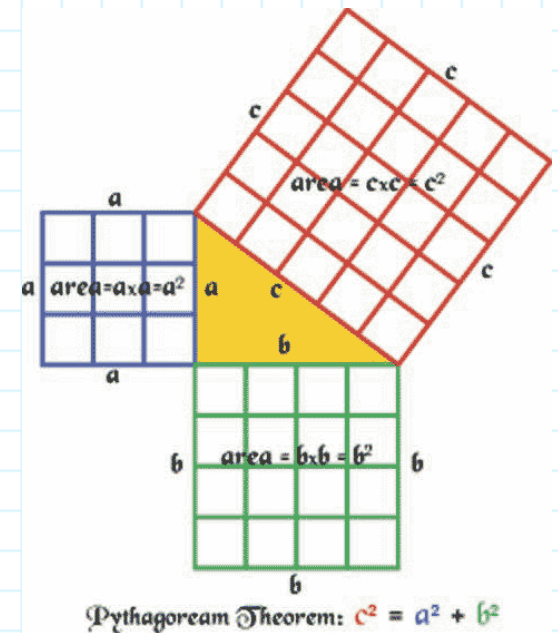
- Funzioni
- Classi



James Gosling (Java)

4.1 Funzione, ipotenusa

- Scrivere una funzione per il calcolo dell'ipotenusa di un triangolo rettangolo
- Parametri: due cateti come `float`
- Risultato: ipotenusa come `float`



*Nel main: chiedere all'utente due valori,
poi invocare la funzione con questi parametri*

4.2 Classe, punto cartesiano

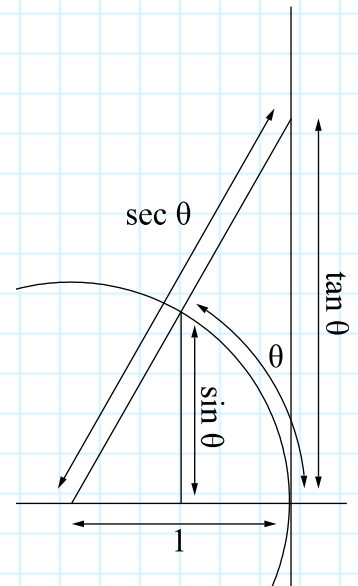
- Scrivere una classe `Point` per rappresentare dei punti sul piano cartesiano
- Coordinate cartesiane (x , y) in campi privati
- Metodi per ottenere e impostare la posizione
- Metodo per calcolare la distanza da un altro punto
- Nel main, permettere ripetutamente all'utente di spostare uno dei due punti
- Mostrare sempre la posizione aggiornata dei punti e la loro distanza

cin e cout fuori dalla classe!

4.3 Coordinate polari

- Aggiungere alla classe dell'esercizio precedente...
- Metodi per ottenere le coordinate polari del punto
- Un metodo per impostare la posizione, fornendo le coordinate polari
- Memorizzare internamente solo le coordinate cartesiane

$$\rho = \sqrt{x^2 + y^2}, \vartheta = \text{atan2}(y, x)$$
$$x = \rho * \cos \vartheta, y = \rho * \sin \vartheta$$



4.4 Vettore di punti

- Gestire un vettore di puntatori a punti
 - `vector<Point*> points;`
- Ripetutamente chiedere all'utente di scegliere tra:
 - Aggiunta di un nuovo punto
`Point* pt = new Point(x, y); // ...`
 - Eliminazione di un punto esistente
(deallocazione memoria + rimozione puntatore)
`delete points[i]; // ...`
 - Calcolo della distanza tra due punti scelti dall'utente
- Al termine di ogni operazione dell'utente, visualizzare la posizione attuale di tutti i punti

4.5 Lancio dadi

- Come es. 2.4, ma ad ogni giocata l'attaccante (A) lancia 3 dadi, il difensore (D) 2 dadi
- Una funzione (due vector come parametri) confronta a coppie i dadi: il dado migliore di A con quello migliore di D, poi il dado medio di A con quello peggiore di D
 - I dadi di ciascun giocatore vanno ordinati:
`sort(v.begin(), v.end())`
- Stimare la probabilità dei diversi risultati
 - 0, 1 o 2 punti dell'attaccante



Ripetere l'esperimento con diversi numeri di dadi

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – Unipr
<http://www.ce.unipr.it/people/tomamic/>

- [illegible]

M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – Unipr
<http://www.ce.unipr.it/people/tomamic/>

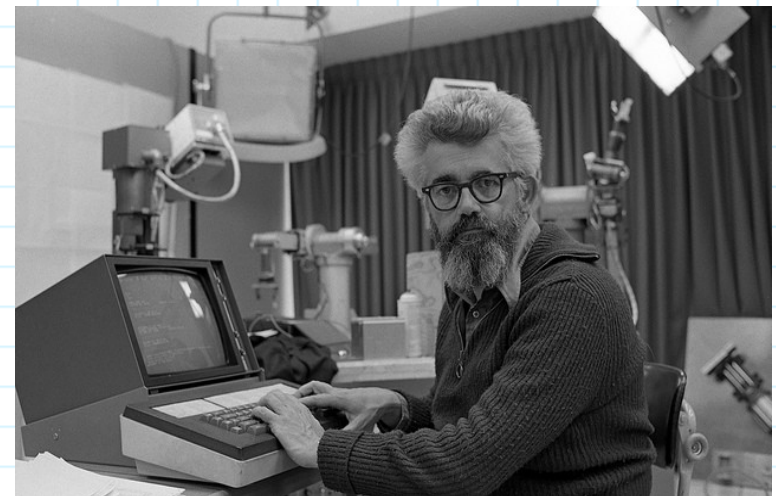
- M. Tomaiuolo – Fondamenti di informatica**
Dip. Ingegneria dell'informazione – Unipr
<http://www.ce.unipr.it/people/tomamic/>



M. Tomaiuolo – Fondamenti di informatica
Dip. Ingegneria dell'informazione – Unipr
<http://www.ce.unipr.it/people/tomamic/>

Esercitazione 5

- Classi
- Ricorsione



John McCarthy (LISP)

5.1 Classe, trova le coppie

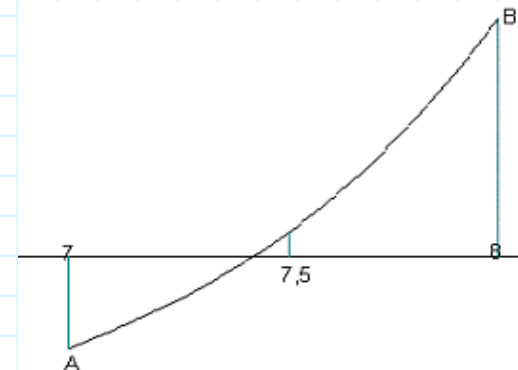
- Aggiungere alla classe dell'es. 4.6...
- Metodo per la mossa: due celle come parametri
 - Obiettivo: trovare le coppie
- Metodo per mostrare lo stato
 - Coppie già indovinate: visibili
 - Ultime celle scelte: visibili
 - Per tutte le altre celle: ' ? '
- Metodo per controllare la conclusione (`bool`)



Matrice di bool per segnare le posizioni indovinate

Oppure distinguere maiuscole e minuscole...

5.2 Bisezione, sqrt



- Scrivere una nuova funzione `sqrt`
 - Parametro `x` e risultato di tipo `float`
 - Restituire risultato con errore minore di `0.0001`
- In un ciclo, trovare approssimazioni successive:
 - Due variabili `low`, `high`: inizio e fine intervallo di stima (intervallo iniziale: da 0 ad `x`, oppure da 0 ad 1, se `x < 1`)
 - Si pone la stima `y` a metà intervallo
 - Se $y * y - x > 0$, nel seguito si considera solo la prima metà dell'intervallo (da `low` ad `y`)
 - Altrimenti solo la seconda metà (da `y` ad `high`)
 - Ad ogni iterazione si dimezza l'intervallo residuo

5.3 Ricorsione, sqrt

- Implementazione ricorsiva della funzione sqrt, con il metodo della bisezione (come esercizio 5.2)
- Parametro `x`: `float`
- Parametro inizio intervallo di stima: `float`
- Parametro fine intervallo di stima: `float`
- Bisezione: invocare ad ogni livello la funzione su un intervallo dimezzato

5.4 Aggregazione, cerchi concentrici

- Scrivere una classe per rappresentare cerchi
 - `center`: puntatore a `Point` (es. 4.2)
 - `radius`: `float`
 - Costruttore con parametri: centro (`ptr`) e raggio
 - Metodi per calcolo area e perimetro
- Gestire un vettore di puntatori a punti ed un vettore di puntatori a cerchi
- Centro dei cerchi: uno dei punti del primo vettore
 - Diversi cerchi possono condividere lo stesso centro
 - Cambiando le coordinate di un singolo punto... tutti i cerchi che lo hanno come centro risultano spostati

5.5 Composizione, poligono

- Point, metodi di lettura/scrittura stato su stream
 - `void read(istream& in);`
 - `void write(ostream& out);`
- Scrivere una classe per rappresentare poligoni
 - Vertici (punti) non condivisi con altri oggetti
 - Metodi per lettura/scrittura di tutti i vertici su stream
 - Metodo per calcolare il perimetro
 - Distruttore per deallocare eventualmente i vertici
- Istanziare un poligono
 - Leggere i vertici inseriti da console, mostrare il perimetro, infine salvare i vertici su file

5.6 Ordinamento

- Chiedere all'utente il nome di uno o più file
- Leggere le parole presenti in questi file, memorizzarle tutte in un vettore v (possibilmente senza ripetizioni)
- Mostrare a schermo le N parole più lunghe in assoluto, in ordine decrescente di lunghezza
 - Chiedere all'utente N

Scrivere una funzione booleana `order` che, dati due parametri di tipo stringa, stabilisca se sono ordinati (per lunghezza)

```
count(v.begin(), v.end(), x); // is x in v?  
sort(v.begin(), v.end(), order); // sort v
```

5.7 Notazione polacca

- Leggere una riga di testo in una stringa
- Scrivere una funzione che valuti la stringa come una espressione, nella forma: "+ 2 7" (=9)
 - Gli operandi possono essere a loro volta espressioni:
"+ * 3 4 15" (=27)
- Scrivere una seconda funzione che trasformi l'espressione nell'abituale notazione infissa:
"((3 * 4) + 15) "

Usare la ricorsione

*Supporre che i "token" siano tutti separati da spazio
e che gli operatori abbiano tutti cardinalità fissa*

Progetto 1

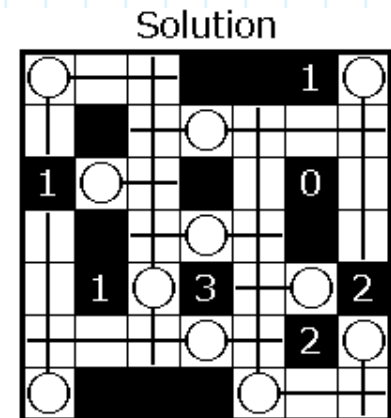
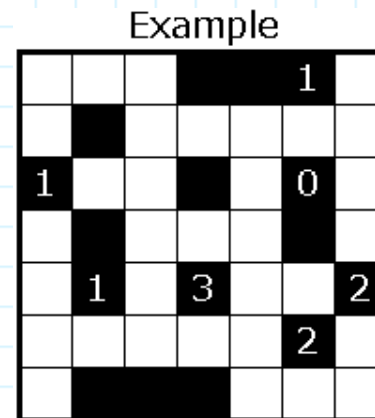
- **Classe** per incapsulare dati e regole di un gioco
 - **Campi privati + metodi pubblici** (*e privati*)
- Ciclo principale e interazione con l'utente (I/O, tramite console), ma *al di fuori della classe*



Ole-Johan Dahl e Kristen Nygaard (Simula)

“Akari” - Light up

- Scopo: disporre delle lampade (cerchi), fino ad illuminare tutte le celle



- Una lampada illumina le celle sulla sua riga e la sua colonna, fino ad una cella nera o al bordo esterno
- Ogni cella bianca può ospitare una lampada, ma:
- Un numero indica quante lampade devono trovarsi nelle celle adiacenti (in orizzontale e verticale)
- Due lampade non possono illuminarsi a vicenda

<http://www.nikoli.com/en/puzzles/bijutsukan/rule.html>



Akari - Metodi base per...

- Inserire una lampada in una cella (*bulb*)
- Controllare se le regole sono violate
- Controllare se il gioco è stato risolto
- Scrivere lo stato del gioco su uno stream generico
 - Per ciascuna cella, indicare anche se è illuminata
- *Aggiungere inoltre metodi per...*
- Leggere lo stato del gioco da uno stream generico
- Marcare una cella con un punto (*dot*)
 - L'utente intende che lì non può trovarsi una lampada

Akari - Bozza della classe

```
class AkariPuzzle
{
public:
    AkariPuzzle();
    int getRows();
    int getColumns();
    bool isSolved();
    bool isWrong();

    char get(int y, int x);
    void putBulb(int y, int x);
    void putDot(int y, int x);

    void read(istream& in);
    void write(ostream& out);
private:
    // ...
}
```

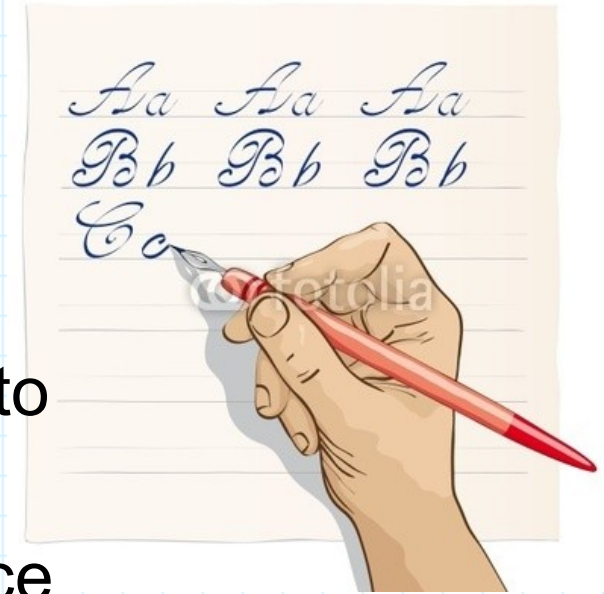
Akari - Gui

- Aggiungere una **interfaccia grafica** al progetto
 - Creare una sottoclasse di `QWidget` o `QMainWindow`
 - Interfaccia adattabile a dimensioni diverse del gioco
- **Riuso** - Definire la classe di modello per le partite in modo generico
 - Usabile sia da interfaccia grafica che da console
 - Permettere di aggiungere/rimuovere lampade e punti

*Sono fornite due classi per gestire anche
i click con il tasto destro del mouse
(`RightButtonGroup`, sulla pagina del corso)*

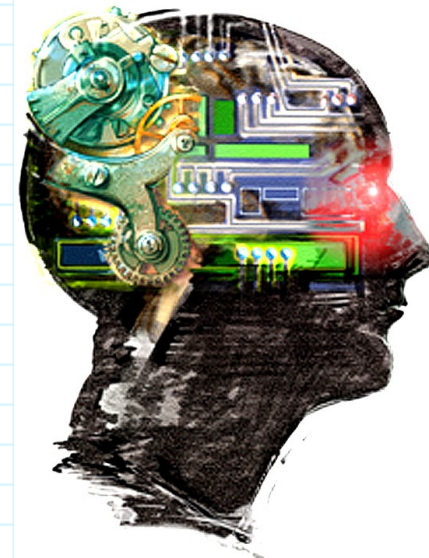


“Bella copia”



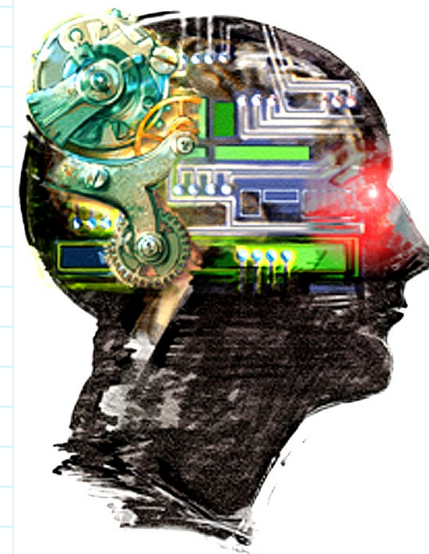
- Codice leggibile e ben organizzato
- Usare **costanti**, anziché valori numerici “*magici*” sparsi nel codice
- Nomi semplici ed **esplicativi** + regole di **stile**:
 - `nomeVariabile`, `nomeFunzione`, `NomeClasse`, `NOME_COSTANTE`
 - **Indentazione**, parentesi ecc.
- **No copia&incolla** del codice
 - Funzioni parametrizzate e/o cicli
- Programmazione strutturata
 - Un solo `return`, a fine funzione; niente `break` nei cicli

Mosse suggerite



- Metodo che calcola e suggerisce una mossa
 1. Controllare i **vincoli numerici**
 - Vincolo N già circondato da N lampade: marcare con punti tutte le celle libere
 - Vincolo N circondato da M lampade ed esattamente N-M celle libere: posizionare lampade in tutte le celle libere
 2. Cercare una **cella libera** (vuota e non illuminata), senza altre celle libere visibili (possibili fonti di luce)
 3. Cercare una **cella marcata con punto** non illuminata, con una sola cella libera visibile

Estensioni opzionali



- Suggestire mosse più difficili
 - Celle d'**angolo** nei vincoli
 - **Vincoli incrociati** (in celle vicine, in diagonale)
 - Verificare se una possibile mossa porta in un **vicolo cieco**
 - Verificare se entrambe le mosse (lampada / punto) in cella X producono un risultato **invariante** in cella Y
- Migliorare l'interfaccia utente
 - Inserire un menù e/o una barra degli strumenti
 - Comandi per apertura e salvataggio di una partita
 - Comandi per suggerimenti di diversa difficoltà
 - Ereditare da `QMainWindow`

Vicolo cieco

```
void AkariPuzzle::solveCuldesac(int y, int x) {
    AkariPuzzle copyBulb = *this; // copy the whole game
    copyBulb.putBulb(y, x);
    copyBulb.solveSimple(); // take all obvious moves (1+2+3)

    AkariPuzzle copyDot = *this; // copy the whole game
    copyDot.putDot(y, x);
    copyDot.solveSimple(); // take all obvious moves (1+2+3)

    if (copyBulb.isWrong()) {
        putDot(y, x); // bulb in x, y is a culdesac, put dot
    } else if (copyDot.isWrong()) {
        putBulb(y, x); // dot in x, y is a culdesac, put bulb
    } else {
        // find invariant cells elsewhere in the table
        // (set with the same value, in both cases)
    }
}
```

Enumerare *tutte* le soluzioni

```
// Adapt the following code to find *all* solutions
// (e.g. a random game should have one single solution)
void AkariPuzzle::solveRecursive(int i) {
    solveSimple(); // take all obvious moves (1+2+3)
    // is there an empty cell after i?
    while (i < table.size() && table[i] != EMPTY) ++i;

    if (i < table.size() && !isWrong()) {
        AkariPuzzle copy = *this; // save current status
        for (int m = 0; m <= 1 && !isSolved(); ++m) {
            if (m == 0) putBulb(i/columns, i%columns);
            else putDot(i/columns, i%columns);

            solveRecursive(i + 1);
            if (!isSolved()) *this = copy; // backtracking
        }
    }
}
```

Progetto 2

- Realizzare un gioco con diversi personaggi
 - Classe per partita
 - Classe base astratta (e sottoclassi) per personaggi
- **Polimorfismo** nei vari personaggi, per:
 - Il movimento
 - L'interazione reciproca



John Carmack (Wolfenstein 3D, Doom, Quake)

Frogger - Strada

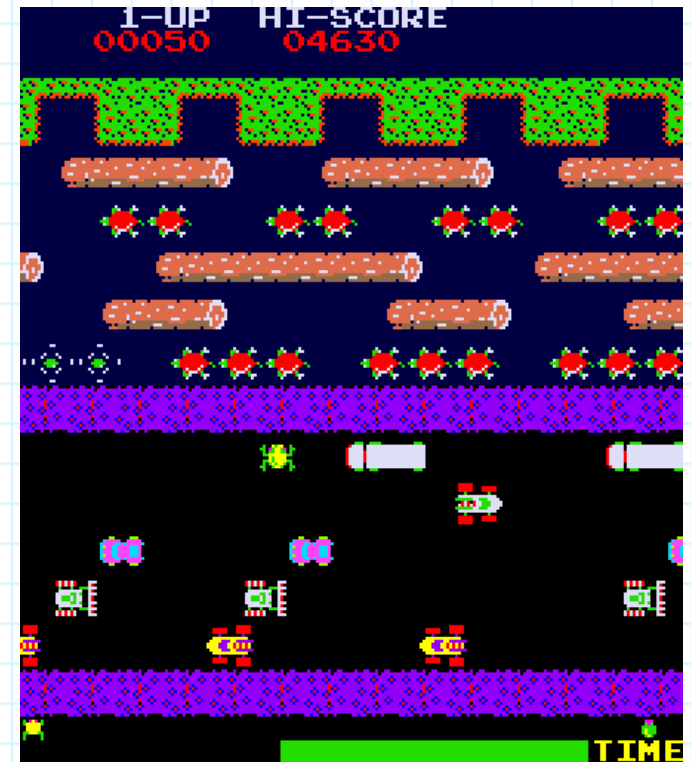
- **Rana**: guidata dall'utente in 4 direzioni: $\updownarrow \leftrightarrow$

Deve attraversare una strada percorsa da camion

Muore se investita da un camion (collisione)

- **Camion**: scorrono in uno dei 2 versi, in orizzontale \leftrightarrow

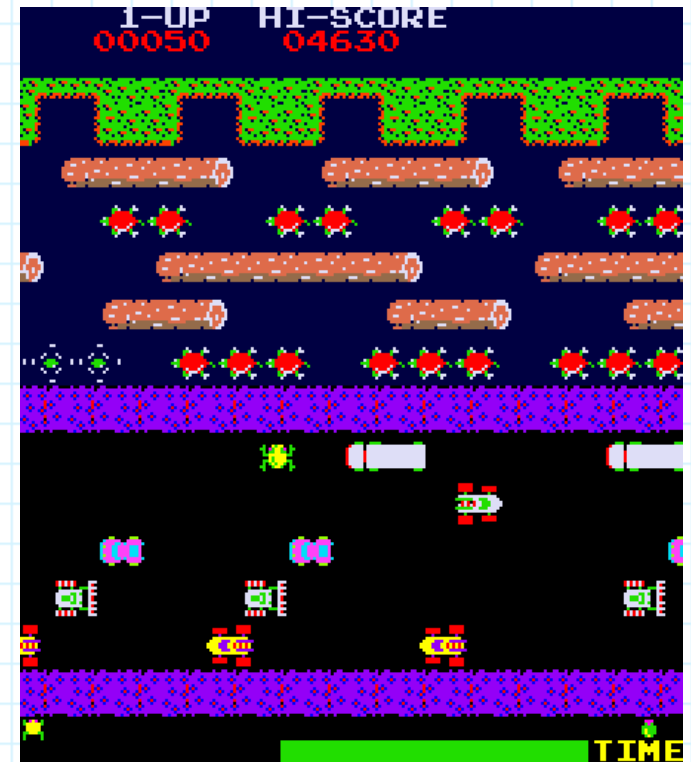
Rientrano in ciclo dalla parte opposta



È fornito un semplice framework di esempio

Frogger - Fiume

- **Rana**: deve attraversare un fiume percorso da tronchi
Può saltare sui tronchi
Muore se cade in acqua
- **Tronchi**: scorrimento \leftrightarrow
Rientrano in ciclo...
Rana scorre con tronco
- **Fiume**: a scelta... modellato come personaggio del gioco (immobile, per gestire collisioni con rana)



Rana.z > Tronco.z > Fiume.z



Frogger - Estensioni

- **Due rane** contemporaneamente in gioco
- Altri personaggi *a fantasia*, per esempio:
 - Tartarughe che scorrono sul fiume
La rana può sfruttarle per attraversare il fiume
Ma periodicamente si immergono...
 - Coccodrilli ecc.
- Limite massimo di tempo per completare una partita
- Vari bonus *a fantasia*, per punti, tempo, vite ecc.
- Icone al posto del testo nelle label (QPixmap, o stile)

Frogger - Opzionale

- Movimento dei personaggi libero e fluido (non su una griglia rigida, ma su qualsiasi pixel)
- QGraphicsScene + Item + View, oppure...
- QPainter + QWidget::paintEvent

La posizione y rimane sempre su righe prefissate...
tranne che per la rana, durante i salti tra righe

*Mantenere un ciclo simile a `Game::moveAll`
e il **polimorfismo** di `Actor::move` e `Actor::touchedBy`*