Lezioni del 28-03-2014,02-04-2014 e 03-04-2014.

LABORATORIO 2

ESERCIZIO 1 – Student Book.

Definisco una sola classe composta da una parte pubblica e da una privata.

Parte pubblica:

Inserire nome corso, inserire tre voti e calcolare il massimo.

Inoltre nella parte pubblica c'è anche il **costruttore**: è un meccanismo di programmazione per costruire gli oggetti quando li istanzio. Ogni oggetto può avere un suo stato (dato dai valori delle property),***il costruttore rende più snella l'assegnazione delle variabili private***. Il primo modo è usarele funzioni set, ma è lungo e barboso, con il costruttore posso passare da subito i valori che verranno assegnatialle variabili membro.

Per farlo devo dichiarare un funzione speciale (non ha il tipo di ritorno, neanche void), e deve godere delle seguenti proprietà:

- deve essere nella parte pubblica.
- non avere alcun parametro di ritorno (neanche void).
- avere lo stesso nome della classe.
- prendere in input i futuri valori delle property.

Nota 1: nel prototipo o signature di una funzione non sono obbligatori i nomi delle variabili che gli verranno passate, basta avere il tipo di variabile. Il compilatore prende solo il tipo e butta viail nome della variabile (questo è vero SOLO nel prototipo).

```
void pippo (string); <=> void pippo (string topolino);
```

Il supporto a runtime sa che quando istanzio un oggetto automaticamente richiama il costruttore e quindi passa levariabili all'interno delle (). Il costruttore si può solo usare all'istanziamento, poi devo per forza usarele funzioni di set.

BEST PRACTICE: a ogni if mettere sempre un else, in modo da considerare sempre tutti i casi.

- se lo strutturo bene il main viene molto snello, richiama solo in ordine sequenziale le diverse funzioni.

Nota 2:

```
cin >> grade1 >>grade2>>grade3;
        è equivalente a
 cin >> grade1;
 cin >> grade2;
 cin >> grade3;
```

Parte privata:

nome corso e massimo valore del voto -> queste determinano lo stato dell'oggetto.

ESERCIZIO 2: Insertionsort.

```
//*****************************************************//
//*********    FILE>Sort.h      **********************//
//*****************************************************//
#ifndef SORT_H_INCLUDED
#define SORT_H_INCLUDED
// questa classe funziona per ordinare qualsiasi array di 10.
class Sort {
public:
void setValues (intval[], intsizeVal);
//^^^prassi: primo parametro array, secondo parametro:dimensione.
intgetValues (int index);
    void sortValues();
    void displayValues(int index);           // fa solo ilcout.

private:
int values[10];
};
#endif // SORT_H_INCLUDED

//*****************************************************//
//*********    FILE> Sort.cpp   **********************//
//*****************************************************//
#include <iostream>
#include "Sort.h"
usingnamespacestd;

/* quando passo size_val>10 vado a sovrascrivere qualcos'altro... fin che rimango nellamia zona di
memoria va tutto bene, se vado al di fuori il supporto a runtime mi genera un exception e mi
crassha.

//-----------------------------------------------//
void Sort::setValues(intval[], intsizeVal) {
//^^ogni volta che passo un vettore passo sempre il puntatore alla prima cella.
    // mi storo l'array che viene fornito alla classe.
for (inti=0; i<sizeVal; i++ ) {
        values[i]=val[i];
    }
}
//-----------------------------------------------//
void Sort::sortValues() {
inttemp;

    for (intnext=1; next<10; ++next) {   //questo lo uso per ciclare su tutti gli elementi.
temp=values[next];      // mi salvo il valore e l'indice.
intmoveItem=next;

        // se è maggiore li devo scambiare, altrimenti ho finito.
while ((moveItem>0) && (values[moveItem-1]> temp)) {
//^^^^ ciclo finito quando sono verificate entrambe le condizioni
values[moveItem]=values[moveItem-1];
/* è così che li shifto a dx tutti quanti, li shifto fin quando non trovo il posto giusto, non dico
"prima trovo la posizione giusta e poi li scambio esposto tutti gli altri"; qui identifico man mano
se devo shiftare */
moveItem--;
        }
        values[moveItem]=temp;
    }
}
//-----------------------------------------------//
int Sort::getValues(int index) {
    return values[index];
}

//-----------------------------------------------//
void Sort::displayValues(int index) {
cout<<index <<getValues(index)<<endl;
}
```

ESERCIZIO 3>COME PASSARE MATRICI

Nel prototipo delle funzioni che richiedono una matrice fra i parametri in ingresso:

```
intnome_funzione (int val[][NUMERO_COLONNE]);
```

Il numero di righe non è richiesto, ma per le colonne è necessario conoscerlo precedentemente per dimensionare correttamente lo stack.
Ok, conosco il numero di colonne ma non conosco il numero di righe, in questo caso lo so a priori, e quindi non passo lenghtperchè l'ho scritto direttamente nel codice.

```
void Sort:: setValues (intval[][5]) {
	for(inti=0; i<5; i++) {
		for(int j=0; j<5; j++) {
			values[i][j]=val[i][j];
		}
	}
}
```

Per inizializzare un array a n dimensioni devo fare n for.
Per passare una matrice a una funzione:

```
int data[5][5]= { {...}, {...}, {....}};
nomeFunzione(nomeMatrice);// non passo nomeMatrice[] o nomeMatrice[][]
// passo sempre e comunque il puntatore alla prima cella.
```

ESERCIZIO 4> DICE PLAY

Conviene utilizzare due classi: ci sono i dadi e il tavolo da gioco con le regole (logica per cui si vince o si perde).

TIPI ENUMERATIVI

```
enum Status {CONTINUE, WON, LOST};
```

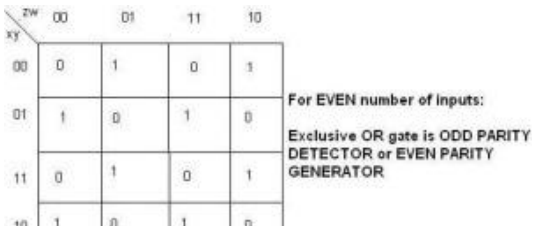
Dentro la graffa ho tutti i possibili valori che la variabile status può assumere. Questo è più pulito perché ho già tutti i possibili valori e non devo fare 3000 if per escludere gli eventuali valori indesiderati. E' come se stessi recintando i possibili valori chepuò assumere una variabile senza mettere 1000 controlli.Il supporto a runtime vedrà continue=0, won=1 e lost=2, invece io programmatore li uso come label.
Nota: quando uso la funzione`rand()` devo sempre inizializzare il seme con`srand(time(NULL));` lo si mette nel main, ogni volta che eseguo il programma prendo il tempo attuale (che cambia di volta in volta) e quindi ottengo qualcosa di più casuale.

Covering Karnaugh Maps – Alternative implementations

Come le abbiamo visto fino adesso, posso utilizzare le mappe di Karnaugh se ho al massimo 5 ingressi, dopo di che non riesco più a sfruttare il giochetto dell'adiacenza logica e fisica. Proviamo a vedere come scalare e ottimizzare le mappe (riesco a scalare ma avrò soluzione che non sono globalmente ottimizzate).

Chessboard like maps

In questo caso conviene utilizzare le porte exor (fra le possibili tecnologie in sviluppo si sta lavorando a creare porte exor con 2 o 3 transistors). La mappa di Karnaugh dell'exor è proprio una scacchiera.

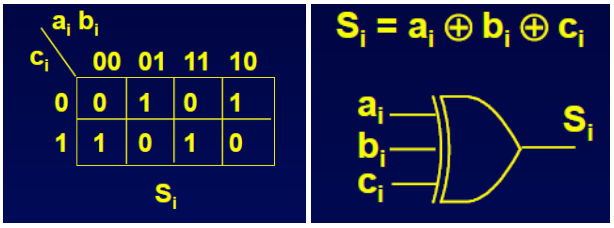


Fonte: 1http://www.exploreroots.com/dk5_clip_image004.jpg

Full-adder (parte 1).

Riceve in ingresso 3 segnali (a_i, b_i e c_i) e da in uscita due segnali (S e c_{i+1}), sul bit S fornisce la somma dei tre bit dati in ingresso e su c_{i+1} il riporto (carry). A livello RT questo sarà un blocchetto elementare, full perché esiste un'altra versione half-adder dove non c'è il segnale di riporto.

Implementation: progetto il circuito tenendo separate le due uscite, di fatto ottengo la mappa dell'exor-> è un exor a 3 ingressi (oppure riscrivendolo usando la proprietà associativa ottengo due exor a 2 ingressi).

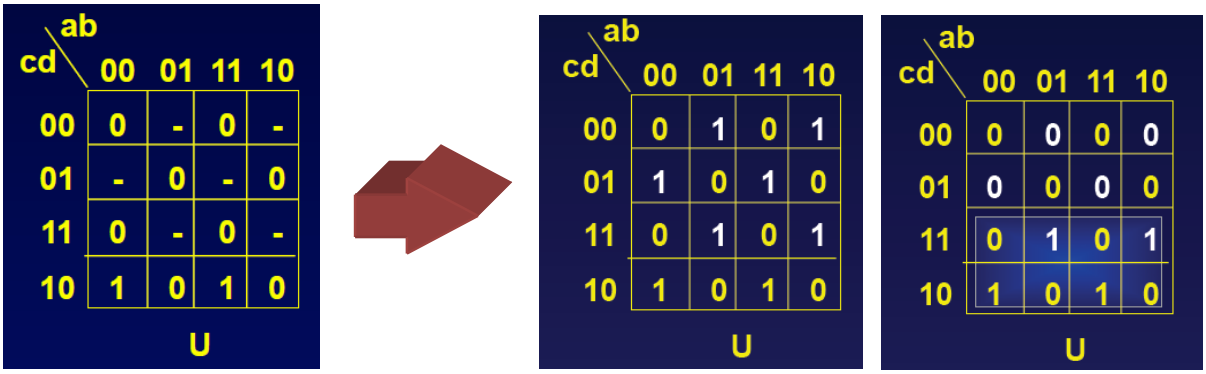


Fonte: TestGroup

Warning: attenzione, la soluzione intuitiva a volte non è ottimizzata!

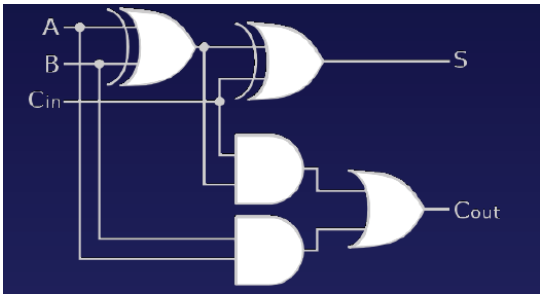
Esercizio 2

Giocando con i valori assunti dai don't care, riesco a ottenere implementazioni diverse per la stessa mappa.



Mappa da coprire. *U=a⊕ b⊕ c⊕ d* *U= (a⊕ b⊕ d)'c*

Map composition



$c_{i+1} = a_i b_i + X = a_i b_i + c_i (a_i \oplus b_i)$

In alcuni casi può essere conveniente (nel senso che è più facile da progettare manualmente) pensare a una mappa di Karnaugh come un AND o un Or o un EXOR fra altre due mappe. Copro F1 e F2 e poi le unisco e ottengo f. Adesso proviamo a implementare il bit di carry, a differenza della copertura con gli implicanti posso vedere la mappa come composizioni di due mappe (in particolare attraverso un OR). Le due mappe sono facilmente copribili, F1 da $a_i b_i$ e la seconda è X, che a sua volta è possibile vedere come composizione di altre due mappe, e quindi complessivamente f=ab+c(a exor b). In questo caso la netlist del full-adder viene qualcosa di questo tipo.

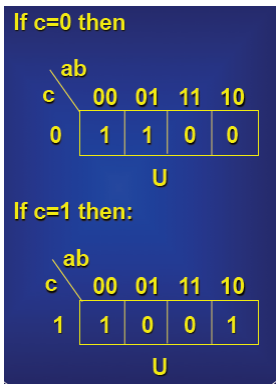
In particolare una porta è in comune a S e al carry e quindi ho ottimizzato qualcosa.

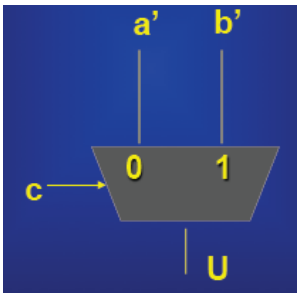
Questa tecnica è comoda se la mappa è quasi a scacchiera, se ho qualche 1 o 0 di troppo posso fare la mappa a scacchiera completa in AND con un'altra mappa che forza gli 1 o gli 0.

Map partitioning

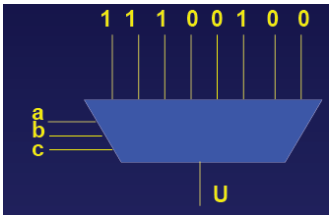
Teorema di espansione di Boole (o teorema di Shennon): *una funzione con n variabili in ingresso può essere espressa attraverso 2 funzioni da n-1 variabili in cui l'n-esima variabile è costante e viene forzata una volta a 1 e una volta a 0.*

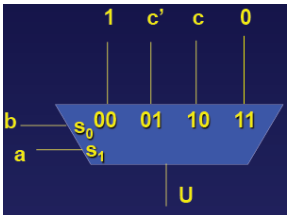
Viene utilizzato perché permette di esprimere una funzione come composizione di due funzioni più semplici. Ok, ma dove mi porta? Riprendiamo il multiplexer, un commutatore, ho un segnale di ingresso e due segnali fra cui switcciare; in modo alternativo è possibile vederlo come due reti combinatorie con un ingresso in meno (principio del divide et impera). Analogamente è possibile iterare questa frammentazione e riuscire quindi a scalare la rete, tuttavia ciò che è stato guadagnato in scalabilità è stato perso in ottimizzazione (in termini di area occupata dal silicio ad esempio).





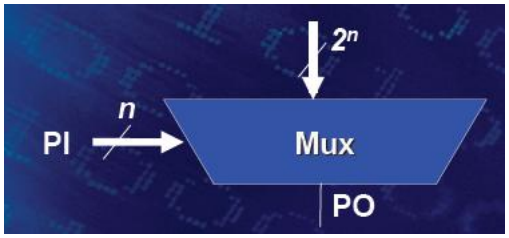
Per esempio #example, con una soluzione alternativa (anche se di costi più elevati) è quella in cui uso due mappe, una con c=0 e una con c=1, unite attraverso un mux. Ovviamente non è l'unica soluzione possibile, ce ne sono anche delle altre che tuttavia non sono molto ottimali. Chiaramente esistono soluzioni non ottimali, arrivando addirittura a implementare le tabelle di verità così come sono, senza fare alcuna operazione di minimizzazione (e.g. utilizzando gli implicanti principali).





Perché vengono adottate anche queste soluzioni?In realtà questo tipo di approccio è molto usato nelle FPGA (array di celle programmabili sul campo), in particolare per implementare le LUT (tavole di consultazione) nelle quali i segnali vengono confrontati con delle costanti (lo 0 viene ottenuto collegandolo a massa, l'1 collegandolo all'alimentazione). Cambiando gli 0 e gli 1 con i quali confronto cambio la mappa di Karnaugh e quindi si ottengono risultati diversi a parità di hardware. Se dall'esterno ho modo di andare a cambiare i bits di configurazione, allora posso riprogrammare l'FPGA e ottenere con lo

stesso hardware funzioni diverse. Il passo successivo è chesi riescono a riconfigurare gli FPGA mentre lavorano (DynamicPartialReconfiguration). In ASIC ottimizzo e quindi ottengo cose più piccole, l'FPGA è più grande ma è molto più flessibile.



2^n Multiplexer.

Fonte: TestGroup.

ROM-based synthesis

Una rom è una memoria organizzata in indirizzi, a ciascuna "parola" è associata un indirizzo, in ingresso ho un bus d'indirizzi e la logica prende la parola i-esima e me la fornisce. Posso solo leggere la ROM, non scriverla, l'operazione di lettura non cancella il valore della "parola", ROM=Read Only Memory, è il costruttore a scrivermele (invece RAM sta per Random Access Memory). Posso progettare a fatica zero un circuito a n ingressi e m uscite utilizzando la struttura di una ROMdove gli ingressi sono gli indirizzi, la memoria ha 2^n celle, ogni cella è composta da m bits,uso gli ingressi come indirizzi e butto fuori il contenuto della cella corrispondente al segnale in ingresso sui m bits di output. In pratica scrivo sulle ROM tutte le possibili combinazioni e poi è come se accedessi alla tabella di verità. In realtà la LUT è un approccio più vecchio, per esempio venivano utilizzate per implementare la funzione *arcsinh* in hardware, prima pre calcolo i valori nel range richiesto e poi li memorizzo con in una ROM con la precisione richiesta.

#ESEMPIO: soluzione alternativa per il comparatore esadecimale.

Address	Content	Address	Content
0000	1	1000	0
0001	1	1001	1
0010	1	1010	1
0011	1	1011	1
0100	0	1100	1
0101	0	1101	1
0110	0	1110	1
0111	0	1111	1

Fonte: TestGroup.

ESERCIZI VARI

- Combination circuit con 4 ingressi e 3 uscite, ricevo valore esadecimale e in uscita fornisco la radice quadrata approssimata all'intero successivo (ceiling(3.1)=4). Fatto le coperture come posso controllare che il risultato ottenuto sia corretto? Proviamo usando altre strade, con le cose viste prima. Se parto dall'espressione finale e la vedo come composizione di mappe, le compongo e se ottengo la mappa iniziale allora ho fatto le cose giuste.
- Comparatore, riceve due operandi da 2 bit ciascuno, e un segnale di controllo il quale specifica la codifica utilizzata per i due operandi (in CA2 (1) o senza segno(0)). Se A>B setto il bit di uscita. Le 3 uscite possibili sono: A_GT_B, se A=B ho A_EQ_B, se A<B allora setto alto A_LT_B. Più tempo

investo all'inizio a pensare, meno fatica faccio a progettare.Suggest: disegnare sempre la top layer, il circuito a livello alto.
Trucco: le 3 uscite non sono indipendenti, prese due mi ricavo la restante come funzioni dell'altre. Mi basta il > e = e se non è nessuna delle due allora è <-> passo da 3 a 2 mappe (risparmio il 33%).
Altra osservazione: prendo quella dell'uguale perché la più banale (anche senza mappa, uso l'exor direttamente). Alla fine devo solo fare una mappa.

Laboratorio 3

Esercizio 1 – Play With Pointers.

```
//*******************************************************//
//*********    FILE>PlayWithPointers.h          ********//
//*******************************************************//
#ifndef PLAYWITHPOINTER_H_INCLUDED
#define PLAYWITHPOINTER_H_INCLUDED

#include <iostream>
using namespace std;
class PlayWithPointer {

    public:
    void displayValue();
voidsetPointer();
voidsetValue(int); // funzione set della variabile aa

    private:
int aa;
int *aPtr;
//^^^^^ la variabile puntatore deve essere dello stesso tipo del contenuto della variabile a
// cui punto.
};
#endif // PLAYWITHPOINTER_H_INCLUDED

//*******************************************************//
//*********    FILE> PlayWithPointers.cpp  *************//
//*******************************************************//
#include <iostream>
#include "PlayWithPointer.h"
using namespace std;

//------- DISPLAY VALUES..    ----------//
void PlayWithPointer::displayValue() {

cout<< "The value of a is:"<<aa<< "and his address:"<<&aa<<"\n"
<< "the value of pointer his:" <<aPtr<< " and the value of *aPtr is:" << *aPtr<<endl;

cout<< "\n\nShowing that * and & are inverses of "
<< "each other.\n&*aPtr = " <<&*aPtr
<< "\n*&aPtr = " << *&aPtr<<endl;

/* *aPtr=DEFERENZIARE IL PUNTATORE: il supporto a runtime prende il valore di aPtr, va nella celladi
memoria che ha con indirizzo quel valore e prendiamo il valore della variabile a cui punto.
*/

}

//--- SETTER PUNTATORE ...     ------------------//
void PlayWithPointer:: setPointer() {
```

```
aPtr=&aa;
/* Assegno sempre un valore, ma questo valore è un indirizzo di memoria, in particolare è
l'indirizzo della cella a cui voglio puntare.*/
}

//--- SETTER PROPERTY aa----------------------//
void PlayWithPointer:: setValue(int x) {
aa=x;
}
```

ESERCIZIO 2 >Calculator

```
//******************************************************//
//*********    FILE>Calculator.h  *************//
//******************************************************//
#ifndef CALCULATOR_H_INCLUDED
#define CALCULATOR_H_INCLUDED

//#include <math.h>
#include <cmath>
class Calculator {

    public:
    void myCube (float *);
    void mySqrt (float *);

};

#endif // CALCULATOR_H_INCLUDED
```

PASSAGGIO BY REFERENCE E BY VALUE.

Quando una funzione deve lavorare su più parametri e devo dare come output più variabili-> non posso fare passaggio by value (faccio una copia della variabile che voglio passare), ma mi conviene utilizzare i puntatori, **i puntatori sono variabili sia di input che di output**. Con il passaggio by reference, mando l'indirizzo della variabile, modifico effettivamente il valore e non una semplice copia e quindi qualunque funzione può accedervi e modificarla(è un parametro di output mascherato come parametro di input). PROBLEMA: nel caso di programmazione in parallelo (più flussi), questi flussi potrebbero accedere alla stessa variabile contemporaneamente e fare casini. Queste funzioni si aspettano in input un puntatore, ovvero un indirizzo. Gli posso passare o una variabile di tipo pointer (se l'ho già definita) o l'indirizzo di una variabile statica.

```
//******************************************************//
//*********    FILE> Calculator.cpp*************//
//******************************************************//
#include "Calculator.h"

void Calculator:: myCube (float *value) {
float aa;

//    aa=*value;
//    aa=aa*aa*aa;
//    *value=aa;

    *value=*value * *value * *value;
//  ^^^^^ anche questa sintassi è lecita...Inoltre non c'è il return!!!
//  inoltre in questo modo non mi perdo la variabile aa.
```

```
}


void Calculator ::mySqrt (float *value) {

*value=sqrt(*value); // ricordarsi che value è un puntatore e quindi devo deferenziarlo...
//    *value=aa;
}

//*****************************************************//
//*********    FILE> main.cpp*************//
//*****************************************************//


#include <iostream>
#include "Calculator.h"

using namespace std;

int main() {
Calculator obj;
float myValue1;

cout<< "Inserisci un float" <<endl;
cin >> myValue1;
obj.myCube(&myValue1);      //FONDAMENTALE: noi gli passiamo l'indirizzo, non il valore!
cout<< "\nil cubo e' :" << myValue1 <<endl;
obj.mySqrt (&myValue1);
cout<< "\ne facendo la radice quadra ottengo :" << myValue1 <<endl;
return 0;

    // senza avere parametri di ritorno e senza sovrascrivere qui nel main myValue1,
    // il valore di questa funzione cambia man mano.
}
```

ESERCIZIO 3 >Array e puntatori.

```
//*****************************************************//
//*********    FILE>MyArray.h *************//
//*****************************************************//
#ifndef MYARRAY_H_INCLUDED
#define MYARRAY_H_INCLUDED

#include <iostream>
//#define ARRAY_LENGHT    5
using namespace std;

class MyArray {

    private:
int *ptr;

    public:
    void setPtr (int *);
    void displayBySubScript();
    void displayByOffset();

};

#endif // MYARRAY_H_INCLUDED

//*****************************************************//
//*********    FILE> main.cpp*************//
//*****************************************************//

#include <iostream>
#include "PrintArray.h"
```

```
using namespace std;

int main() {
int aa;

int b[]={15,5,7,8,5};
PrintArrayobj(b);

/* Quando passo l'array a una funzione io passo SEMPRE l'indirizzo del primo elemento perchè gli
array sono SOLO gestiti con i puntatori, poi io posso accorgermene o no, però in realtà lui lavora
solo con gli indirizzi. Con le variabili e le struct questo non è più vero...   */

//     MODO 1...
//    obj.printBySubscrite(5);
//    obj.printByOffset(5);


//      MODO 2...
/*
    Modo alternativo per prendere la dimensione di un'array. Se non so l'array è lungo 10, posso
ricavarlo come

lenght=sizeof(b)/sizeof(*b)
#elem= dimensione di tutto l'array/dimensione del singolo elemento (in particolare accedo al primo
elemento dell'array).

                5=(32*5)/32

        oppure posso fare un define e passargli quello.
*/
obj.printByOffset((sizeof(b)/sizeof(*b)));
return 0;
}
```

ESERCIZIO 4>SelectionSort

SelectionSort: con un for ciclo sull'array non ordinato salvo l'indicedell'elemento più piccolo e poi lo scambio con primo elemento dell'array non ordinato. La funzione swap in ingresso vuole l'indice degli elementi che voglio scambiare, non i due valori!!!!

Funzione swap: devo avere una variabile d'appoggio, inoltre nella funzione swap non importa se idue elementi provengono da un unico array o sono due variabili indipendenti, la funzione è generale.

NOTA: in cout se uso setw(4) mi lascia 4 tab.
<int *aa> non è deferenziamento della variabile aa, sto solo passando un puntatore di tipo int che chiamo aa.