

Programmare in C++

A.S. 2015/2016

Alessandro Saltini

Liceo Scientifico Statale "A. Tassoni"

- ▶ L'informatica **non** è
 - ▶ saper usare un computer
 - ▶ saper costruire/riparare un computer
 - ▶ usare programmi scritti da altri
- ▶ L'informatica **è**
 - ▶ una branca della matematica
 - ▶ lo studio dell'**informazione**
 - ▶ lo studio degli **algoritmi**
 - ▶ lo studio dei **linguaggi di programmazione**

- ▶ L'informazione si misura in **bit** (binary digit)
- ▶ 1 bit è la quantità di informazione necessaria a determinare una quantità che può essere **0** o **1**
- ▶ Il **byte** è un multiplo del bit: $1 \text{ B} = 8 \text{ bit}$
- ▶ Due scale di multipli del byte:
 - ▶ decimale: kB (10^3), MB (10^6), GB (10^9), TB (10^{12}), ...
 - ▶ **binaria**: KiB (2^{10}), MiB (2^{20}), GiB (2^{30}), TiB (2^{40}), ...

- ▶ L'algebra Booleana è l'algebra dei bit
- ▶ È un **modello** della logica classica: 1 = vero, 0 = falso
- ▶ Insieme di base $B = \{ 0, 1 \}$
- ▶ Tre operazioni fondamentali:
 - ▶ **not** (non): $\neg : B \rightarrow B$
 - ▶ **and** (et): $\wedge : B^2 \rightarrow B$
 - ▶ **or** (vel): $\vee : B^2 \rightarrow B$

► not (non): \neg

► $\neg 1 = 0$

► $\neg 0 = 1$

► and (et): \wedge

► $1 \wedge 1 = 1$

► $1 \wedge 0 = 0$

► $0 \wedge 1 = 0$

► $0 \wedge 0 = 0$

► or (vel): \vee

► $1 \vee 1 = 1$

► $1 \vee 0 = 1$

► $0 \vee 1 = 1$

► $0 \vee 0 = 0$

- ▶ Combinando queste tre operazioni si possono ottenere tutte le altre operazioni possibili
- ▶ In realtà basta **una** sola operazione, meno intuitiva:
 - ▶ nand (\uparrow)
 - ▶ nor (\downarrow)
- ▶ Esistono circuiti **elettrici** che realizzano materialmente queste operazioni logiche
 - ▶ segnale "alto" = 1
 - ▶ segnale "basso" = 0
- ▶ Sono l'elemento di base dei computer

- ▶ Un numero in rappresentazione **decimale** è espresso come combinazione di potenze di 10

$$1064 = 1 \cdot 10^3 + 0 \cdot 10^2 + 6 \cdot 10^1 + 4 \cdot 10^0$$

- ▶ I coefficienti sono compresi tra 0 e 9 (**minori** di 10)
- ▶ Il massimo numero con n cifre decimali è $10^n - 1$
- ▶ I numeri esistono indipendentemente dalla loro rappresentazione, è solo un modo di scriverli

- ▶ La rappresentazione **binaria** utilizza le potenze di 2

$$10110 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

- ▶ I coefficienti sono soltanto 0 e 1 (**minori** di 2)

- ▶ Il massimo numero con n cifre binarie è $2^n - 1$

- ▶ Ogni cifra è rappresentabile da un **bit**

- ▶ n cifre $\Rightarrow n$ bit

- ▶ I computer memorizzano i numeri in binario
- ▶ Le operazioni tra essi vengono svolte da appositi circuiti, basati sulle operazioni Booleane
- ▶ Ogni operazione richiede un certo tempo
- ▶ Limiti di memoria/tempo impediscono di operare con numeri arbitrariamente grandi

- ▶ I numeri negativi devono memorizzare anche il segno
- ▶ Costo di 1 bit aggiuntivo
 - ▶ $s = 0 \Rightarrow +$
 - ▶ $s = 1 \Rightarrow -$
- ▶ Spesso si ricorre a rappresentazioni alternative
 - ▶ rimozione di ambiguità tra $+0$ e -0
 - ▶ facilità di calcolo
 - ▶ occupano comunque 1 bit in più

- ▶ La parte frazionaria è problematica da rappresentare

$$1.1011 = 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4}$$

- ▶ Non tutti i numeri con rappresentazione decimale finita hanno rappresentazione binaria finita
- ▶ Non possiamo memorizzare infinite cifre
 - ▶ Impossibile rappresentare i numeri irrazionali
 - ▶ Non tutti i numeri razionali sono rappresentabili

- ▶ Richiamiamo la notazione scientifica

$$1064.15 = 1.06415 \cdot 10^3$$

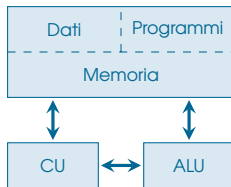
- ▶ Generalizzabile in binario come

$$110.1011 = 1.101011 \cdot 2^2$$

- ▶ La **prima** cifra della rappresentazione scientifica binaria è **sempre 1**, non serve memorizzarla

$$110.1011 = 1.101011 \cdot 2^2$$

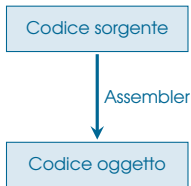
- ▶ La parte dopo la virgola è detta **mantissa** o **significando**, è un numero intero
- ▶ L'**esponente** di 2 è un numero intero
- ▶ Un numero frazionario viene rappresentato come coppia di numeri interi
 - ▶ bit del significando \Rightarrow precisione
 - ▶ bit dell'esponente \Rightarrow range



- ▶ Memoria unica per dati e programmi
- ▶ CU (Control Unit): assegna e gestisce risorse
- ▶ ALU (Arithmetic Logic Unit): compie operazioni
- ▶ ALU + CU = CPU (Central Processing Unit)

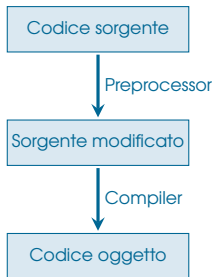
- ▶ La CPU esegue istruzioni **semplici**
- ▶ Istruzione tipo:
 - ▶ che operazione fare
 - ▶ indirizzi di memoria degli operandi
 - ▶ indirizzi di memoria dei risultati
- ▶ Le istruzioni sono codificate come numeri binari
- ▶ Le istruzioni eseguibili dipendono dal tipo di CPU

- ▶ I linguaggi **assembly** hanno una corrispondenza 1-ad-1 con le istruzioni della macchina
- ▶ Le istruzioni vengono “tradotte” da numeri binari a parole comprensibili ad un essere umano
- ▶ Un programma in assembly è una lista di istruzioni che la CPU eseguirà nell’ordine in cui appaiono
- ▶ CPU diverse hanno linguaggi assembly diversi

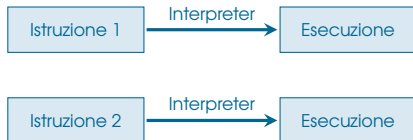


- ▶ Il **codice sorgente** è un file contenente testo
- ▶ Il **codice oggetto** è un file contenente istruzioni binarie
- ▶ Un programma detto **assembler** converte il codice sorgente in codice oggetto, eseguibile dalla CPU

- ▶ Un **linguaggio** di programmazione è un linguaggio **formale** con il quale scrivere istruzioni
 - ▶ distinzione tra istruzioni valide/invalidi
 - ▶ le istruzioni controllano la CPU
- ▶ Distinzione di **livello**
 - ▶ basso livello: **1-ad-1** con istruzioni della CPU
 - ▶ alto livello: linguaggi **astratti**
- ▶ Un **paradigma di programmazione** è uno stile secondo il quale viene scritto il codice sorgente
- ▶ Ciascun linguaggio accetta uno o più paradigmi



- ▶ Il codice sorgente viene scritto per intero
- ▶ Il **preprocessor** modifica al sorgente (facoltativo)
- ▶ Il **compiler** crea un file oggetto dal sorgente



- ▶ Ciascuna istruzione viene compilata singolarmente
- ▶ Esecuzione di programmi incompleti
- ▶ Molto più lenti dei linguaggi compilati

- ▶ Programmazione imperativa
 - ▶ lista di **comandi** eseguiti in un certo ordine
- ▶ Programmazione dichiarativa
 - ▶ lista di **relazioni** tra enti
- ▶ Programmazione ad oggetti
 - ▶ lista di **oggetti** e di **interazioni** tra essi

- ▶ Scrittura di un **algoritmo** per risolvere un problema
- ▶ Esempio: cambiare la batteria di un telecomando
 - ▶ aprire il vano batterie
 - ▶ rimuovere la vecchia batteria
 - ▶ gettare la vecchia batteria
 - ▶ inserire la nuova batteria
 - ▶ chiudere il vano batterie
- ▶ L'ordine delle istruzioni è determinante

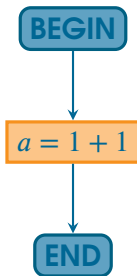
- ▶ Per rappresentare un algoritmo si utilizzano i flowcharts o diagrammi di flusso
- ▶ Un flowchart è costituito da
 - ▶ celle contenenti istruzioni
 - ▶ frecce che guidano il flusso di controllo
- ▶ I flowcharts sono un linguaggio di programmazione
- ▶ Creare flowcharts aiuta nella stesura di un algoritmo



- ▶ Istruzioni di inizio e fine diagramma
- ▶ Uno ed un solo **BEGIN** per diagramma
- ▶ A volte ammessi più **END** in un singolo diagramma



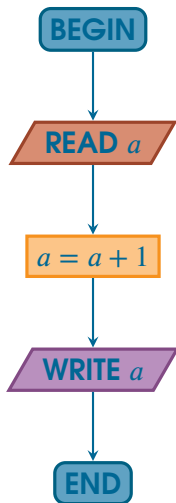
- ▶ Istruzione di processo
- ▶ Viene eseguita l'operazione indicata nella casella

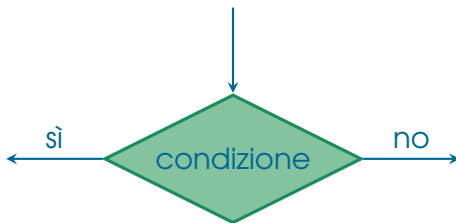


- ▶ Il programma calcola $1+1$
- ▶ Il risultato viene messo nella **variabile** a

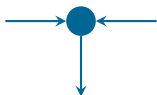


- ▶ Istruzioni di lettura/scrittura
- ▶ Lettura: un valore inserito dall'**utente** viene memorizzato nella variabile x
- ▶ Scrittura: il valore corrente della variabile x viene comunicato all'**utente**

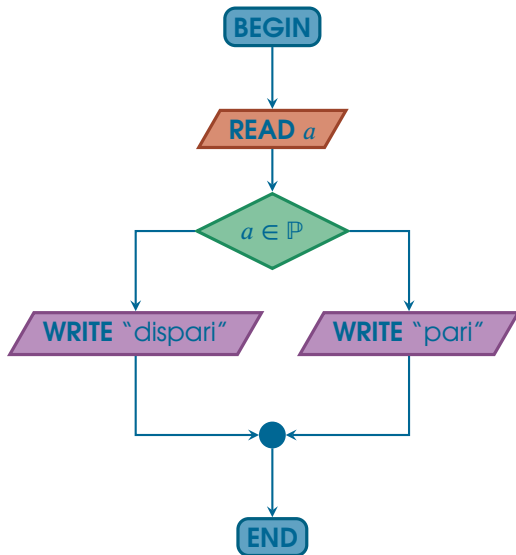




- ▶ Istruzione condizionale
- ▶ Il flusso del grafico cambia direzione a seconda che la condizione sia vera oppure falsa



- ▶ Connettore
- ▶ Permette di rincongiungere due rami separati

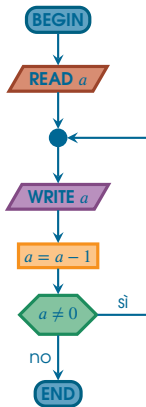


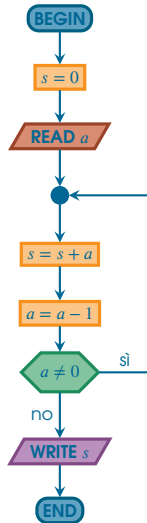
- ▶ Le regole elencate danno origine alla programmazione imperativa in senso lato
- ▶ Le frecce possono “risalire” il diagramma
 - ▶ istruzioni **goto**: ritorno ad un punto precedente
- ▶ Questo rende più difficile:
 - ▶ studio formale del programma
 - ▶ apportare modifiche al codice
 - ▶ comprensione del programma da parte di terzi

- ▶ La programmazione **strutturata** vieta i goto
- ▶ Le frecce non possono risalire il diagramma
- ▶ Viene aggiunto un nuovo simbolo
- ▶ Teorema di Böhm-Jacopini
 - ▶ **equivalenza** tra imperativa e strutturata



- ▶ Istruzione di loop
- ▶ Concettualmente identica all'istruzione condizionale, ma uno (ed uno solo) dei due flussi può risalire
- ▶ Permette di ripetere una serie di istruzioni





```
1 | #include <iostream>
2 |
3 | int main() {
4 |     std::cout << "Hello, World!" << std::endl;
5 |     return 0;
6 | }
7 |
```