**PROGRAMMAZIONE PROCEDURALE C++**

**Architettura del calcolatore**

-**Software** 🡪sono delle applicazioni, specifica operazioni che vengono eseguiti dall’hardware

-**Architettura esterna** (hardware) 🡪 **Dispositivi di uscita**: schermo, audio, stampanti

**Dispositivi di ingresso**: tastiera, mouse

**Unità centrale**: processore, RAM

**Memorie di massa**: disco rigido, dischi floppy, CD/DVD

-**Architettura di Von Neuamann** 🡪 architettura di tutti i calcolatori moderni

Composta da: **processore (CPU) – RAM – interfacce periferiche**

Leggere *schema di funzionamento* **(slide 12, 13)**

Nel passaggio finale trasferisce i dati permanentemente sulle memorie di massa

**-Processore** 🡪 **CPU** ( “Cental Processing Unit” )

è il circuito che effettua i calcoli ed è responsabile delle operazioni, sovrascrive i dati nella RAM

comanda l’interfaccia attraverso il bus di sistema che collega le componenti

ex: Core i7 2.8 GH

**-RAM** 🡪 ( “Random Access Memory” )

è la memoria di lavoro per la CPU e le sue eleborazioni avvengono esclusivamente nella RAM

la RAM è volatile: perde lo stato in assenza di tensione

è composta da numerosi circuiti **(celle),** essi sono capaci di contenere un 🡪**bit** di informazione

unità di memorizzazione

parlando della struttura 🡪 della RAM:

bit organizzati in **registri** 🡪 16 / 32 / 64 bit, ogni registro ha un suo indirizzo

**-Interfacce periferiche** 🡪 circuiti a cui sono connesse le periferiche:

memorie di massa

dispositivi di ingresso/uscita

funzionamento: l’interfaccia si occupa di gestire la comunicazione con la periferica

**-BIT** 🡪 rappresentare informazioni con i bit:

con un bit 🡪 due numeri (0 e 1) con due bit 🡪 quattro numeri (00, 01, 10, 11) …

unità di misura della memoria (si usa anche per i dischi):

bit: 0/1 🡪 byte: 8bit 🡪 KiloByte: 1024 byte 🡪 Megabyte: 1 mln byte 🡪 GigaByte: 1 milardo di byte

**-rappresentazione delle informazioni 🡪** sono rappresentate attraverso **sequenze di bit**

**Codifica**: insieme di regole per rappresentare oggetti in altri oggetti

**-rappresentazione dei numeri** 🡪

rappresentazione dei numeri interi positivi 🡪 ogni bit è una cifra

rappresentazione dei numeri interi relativi 🡪

**-rappresentazione dei caratteri** 🡪 tramite il codice ASCII a 7 bit

al carattere è associato un numero tra 0 e 127 🡪 coincide con esso

vale anche per le cifre (0-9) e per i caratteri speciale (+)

migliorato il codice ASCII a 8 bit (extended ASCII)

al carattere è associato un numero tra 0 e 255 🡪 coincide con esso

maggior numero di caratteri rappresentabili

**OGGI: codice UNICODE (16 bit o superiore)**

**PROGRAMMAZIONE PROCEDURALE C++**

**Linguaggi di programmazione**

-Linguaggi per impartire istruzioni al processore 🡪 **programma:**

sequenza di istruzioni:

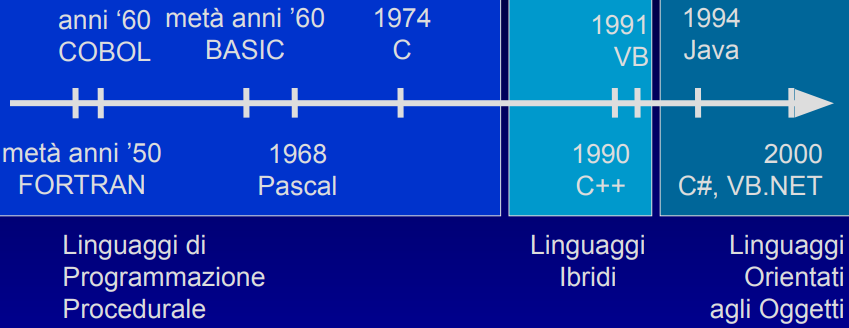
vengono forniti dei dati, vengono calcolati, vengono restituituti

-**Evoluzione** dei linguaggi di programmazione🡪 i linguaggi della stessa famiglia hanno = concetti

**C / C++**

Alcuni linguaggi hanno delle differenze sulle funzionalità:

**sintassi e semantica**



-**Sintassi:** sarebbe la ***grammatica*** del linguaggio 🡪 insieme di regole per stabilire le frasi corrette

EX: in italiano dopo il punto va la maiuscola

EX: in C++ le istruzioni si concludono con **;**

**-Semantica:** sarebbe il ***significato*** del linguaggio 🡪 insieme di regole x il calcolatore per eseguire programmi

EX: in italiano “ciao come va?”

EX: in C++ cout << “ciao”;

verificare successivamente con **dei controlli** sulla sintassi e semantica, effettuando dei test

-**Ciclo di vita di un programma**: è necessario oltre alla sintassi e semantica, un metodo dato che è complesso creare un software 🡪 ci sono vari punti da seguire:

**punto di partenza**: descrizione e analisi del problema

**punto intermedio**: concepire una strategia

**punto di arrivo**: l’applicazione funziona correttamente

**algoritmo:** strategia per risolvere il problema, e decidere che passi fare

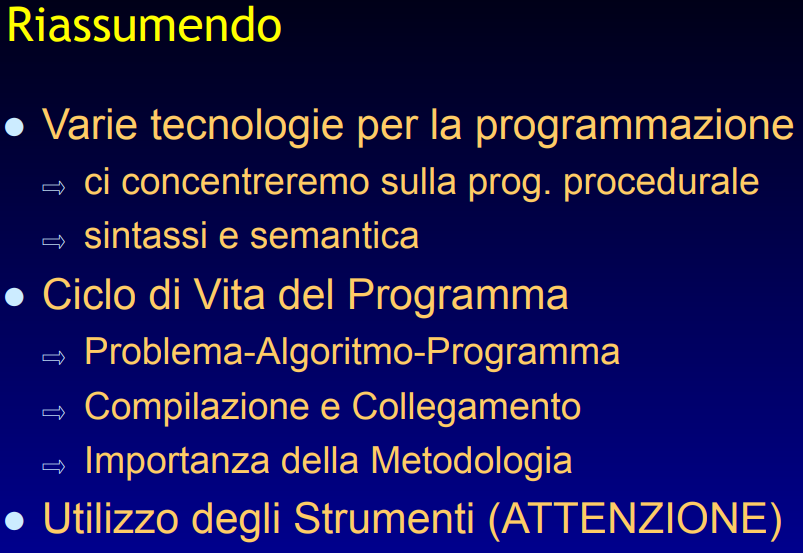
-**Utilizzo degli strumenti**: fondamentale per creare un programma 🡪 **disponde di vari strumenti**:

1. editor di testo

2. compilatore e link

3. lanciatore del programma

4. debugger (o usare stampe)

****

**ALGORITMI E PROGRAMMAZIONE**

-**Risoluzione problemi / problemi non risolvibili** 🡪 tramite calcolatore

-**Algoritmo** 🡪 strategia e insieme **FINITO** di passi in qualsiasi situazione di vita (EX: cucinare…)

dati 🡪 **algoritmo** 🡪 risultati

Proprietà dell’algoritmo 🡪 **realizzabilità , non ambiguità , finitezza**

-**Linguaggio di programmazione** 🡪 per eseguire **le istruzioni impartite dall’elaboratore**

esecuzione di un programma sul calcolatore (**processo o task**)

-**Analisi** 🡪 input

output

relazione tra 🡪 input e output

-**Operatori** 🡪 **aritmetici** ( + - / \*)

**Logici e relazionali** (< > == = >= <= not…) possono essere differenti tra i linguaggi di program.

-**Diagrammi di flusso** 🡪 flowchart

descrive le azioni da eseguire

ad ogni azione corrisponde un blocco (inizio, fine, ingresso, uscita, operazione…)

-**Programmazione strutturata** 🡪 di tipo TOP-DOWN

algoritmi + leggibili, test correzzioni e manutenzioni + semplici…

-**Teorema di** **Bohm-Jacopini** 🡪composto da 3 strutture di controllo:

**Sequenza – selezione – iterazione** 🡪 è possibile codificare **TUTTO**

Creato da 2 matematici italiani (corrado B, giuseppe J)

-**Metodo top-down** / **metodo bottom-up**

-**Dichiarazione dei dati in C++** 🡪 è importante la progettazione delle strutture dati (int, float, char, string)

-**interfaccia utente** 🡪 **è il mezzo** con cui l’utente comunica con l’elaboratore

in **input** i dati, in **output** i risultati

un programma per fare ciò necessita di apposite librerie



-**Struttura di selezione (if)** 🡪 permette di effettuare una scelta tra 2 alternative:

Se è **VERO** vengono eseguite le istruzioni nel ramo vero

Se è **FALSO** vengono eseguite le istruzioni nel ramo vero

La verità o la falsità della proposizione è detta valore di verità

-**La selezione multipla (Switch-Case)** 🡪 quando ci sono troppe scelte annidate

-**Concettivi di logica** 🡪 connettivo **and**, connettivo **or**, connettivo **not**

tramite essi è possibile ridurre la complessità degli algoritmi

-**Cicli** 🡪 si stabiliscono tramite una struttura iterativa ne esistono 3 tipi di cicli:

**postcondizionale** (do-while) 🡪 l’istruzione viene eseguita PRIMA del ciclo almeno una volta

**precondizionale** (while) 🡪 l’istruzione viene eseguita DOPO il ciclo

**ciclo for** 🡪 utilizziamo quando sappiamo il N di volte che dobbiamo ripetere quell’operazione

-**Numeri casuali 🡪 rand() 🡪** funzione usata per generare un **numero pseudo-casuale** (per adesso)

inclusione di 2 librerie **<cstdlib>**  , **<ctime>**

tra 0 e RAND\_MAX (il valore varia in base al compilatore, in genere è 32767)

rand()%1000 (numero casuale tra 0 e 999)

rand()%10+5 (numero casuale tra 5 e 14, il 5 prende nome di **offset**)

per ottenere dei numeri differenti, bisogna cambiare il punto iniziale (seme) con **srand()**

Per impostare un massimo si ricorre all'operatore modulo (%) che, serve per calcolare il resto di una divisione intera e questo spiega perché il valore massimo può essere N-1.

-**Stringhe** 🡪 non richiedono un terminatore di stringa, occorre la libreria **<string>**, è una classe

EX: string s1 = “ciao mondo!”;

-per accedere al singolo carattere utilizziamo la **notazione degli array**

-per vedere la lunghezza della stringa, utilizziamo **lenght()** oppure **s.size()**

-per confrontare le stringhe, funziona esattamente come gli altri tipi di dati **(<, ==, >= …)**

-per concatenare le stringhe , utilizziamo gli operatori **+, +=**

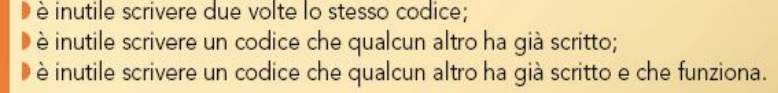
-per ricercare una sottostringa, utilizziamo il **find()**,oppure **string::npos** se la sottostringa non viene trovata

-per leggere una stringa anche se è composta da spazi tra una parola e l’altra, usiamo **getline(cin, stringa)**

-per le altre funzioni aggiuntive leggere plico slide

-**Funzioni** 🡪 la programmazione moderna suddivide il programma in + parti, sono chiamati **sottoprogrammi**

sono segmenti di codice che realizzano operazioni semplici o complesse:

i sottoprogrammi sono componenti (**atomi software**) da utilizzare per i programmi + complessi: 

Perché li utilizziamo? 🡪 per tre motivazioni:

**riusabilità** 🡪 utilizzare lo stesso codice per problemi diversi

**astrazione** 🡪 esprimere operazioni complesse in modo sintetico

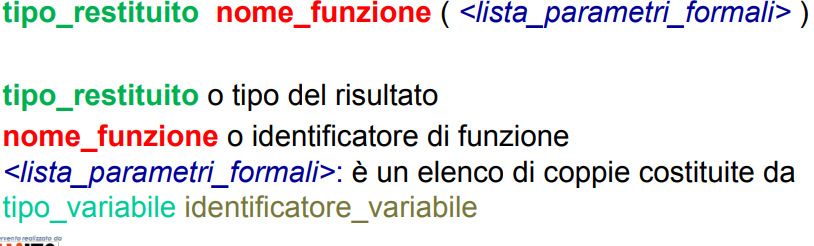
**risparmio** 🡪 scrivere una sola volta il codice usato + volte

la funzione possiede la seguente *struttura* 🡪 **testata** (header)

**istruzioni** (body, racchiuse tra le {} ) 🡪 cosituita da:

**parte dichiarativa**

testata: **parte esecutiva**

****

-EX: varie librerie che contengono funzioni importanti 🡪 **<cmath**>(matematiche) **<cctype>**(caratteri)

anche se le librerie con funzioni sono tante, *non sono sufficienti* per affrontare tutti i problemi del nostro programma, quindi dobbiamo crearne anche noi.

-per condividere i dati tra le funzioni 🡪 due metodi:

1. **variabili globali** 🡪

2. **parametri di riferimento** 🡪 **parametri del valore** (return)

**parametri di riferimento** (&)

**parametri di riferimento costanti**

-**variabili 🡪** prima di parlare delle variabili globali e delle locali, definiamo prima il **termine di variabile**, è un posto nella memoria che ha:

1. **un nome** (reddito, nome)

2. **tipo di dato** (int, double)

3. **una dimensione** (numero di byte)

4. **un ambito** 🡪 **variabili globali** (tutte le funzioni possono vederle ed usarle)

**variabili locali** (solo la funzione che le dichiara, usa/vede queste variabili)

5. **un tempo di vita** 🡪 **variabili globali** (vivono finchè il programma viene esguito)

**variabili locali** (vivono quando vengono eseguite dalle funzioni che le definiscono)

**-variabili globali 🡪** effetti negativi: 1) non sicuro 2) **contro** il principio dell’informatiob hiding

**-variabili locali 🡪** sono dichiarate all’interno del corpo della funzione, vengono distrutte all’uscita della funzione

**-parametri di riferimento 🡪 passaggio per valore**

**passaggio per riferimento (&)**

-**algoritmi notevoli** 🡪 affronteremo lo studio di alcuni algoritmi sugli array

**ricerca sequenziale** 🡪

**ricerca binaria** 🡪

-**le struct** 🡪 vengono utilizzati per gestire degli elementi NON OMOGENEI, servono a contenere dati di tipo

diverso (numeri, stringhe e array), il dato aggregato che la STRUCT descrive si chiama RECORD

-**record** 🡪 il singolo elemento del RECORD si chiama CAMPO

-**struct + funzioni** 🡪METTERE PRIMA LE STRUCT PRIMA DELLA FIRMA E DELLE FUNZIONI!