Istituto Professionale per L'industria L'Artigianato
"Antonio Guastaferro"

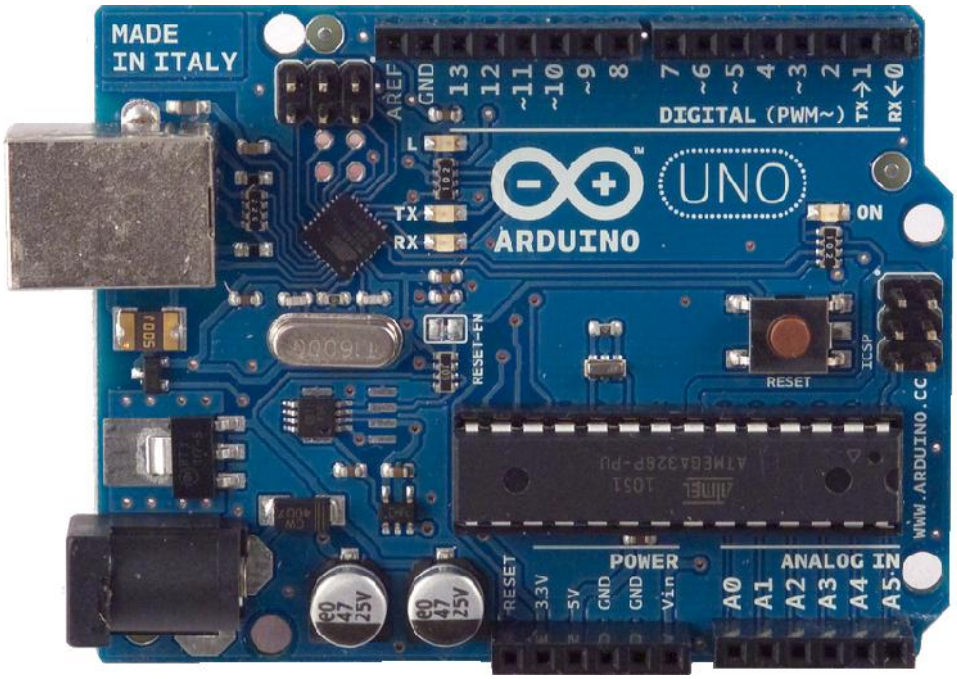
63074 SAN BENEDETTO DEL TRONTO (AP)

Classe : 4A_MAN 5A_IPAI
A.S. : 2020-2021
Docente : Tufoni Franco
Disciplina : Tecnologie elettriche-elettroniche, dell'automazione e applicazioni

Arduino

Hardware - Software

Applicazioni



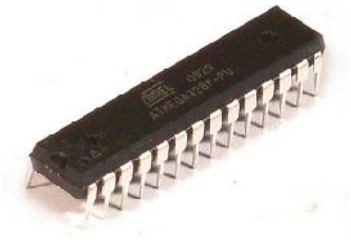
Open Source

Vers. 2.0

Indice

1.1 Microcontrollore - Generalità

Il **Microcontrollore** o **Microcontroller** o **MCU** è un dispositivo elettronico integrato su singolo chip, nato come evoluzione alternativa al Microprocessore.



E' progettato per interagire direttamente con il mondo esterno tramite un programma residente nella propria memoria interna e mediante l'uso di *pin* specializzati o configurabili dal programmatore.

Sono disponibili in 3 fasce di capacità elaborativa (ampiezza del *bus* dati): 8 bit, 16 bit e 32 bit. L'ampia gamma di funzioni di comando e controllo disponibili, sia analogiche che digitali, integrate sullo stesso chip, permette l'impiego delle MCU in sostituzione di schede elettroniche cablate tradizionali ben più complesse e costose.

Per i microcontrollori sono rilasciati sistemi di sviluppo amatoriali e professionali anche in modalità *open source*.

L'architettura del Microcontrollore prevede un insieme di moduli fissi, comuni a tutti i modelli, e una serie di possibili estensioni in funzione del costruttore, del prezzo e della fascia applicativa):

- Unità di elaborazione (CPU o Core)
- Memoria di programma (ROM, OTP, EPROM, FLASH)
- Memoria dati (RAM e EEPROM)
- Porte di I/O configurabili
- Gestione Interrupt
- Oscillatore interno o esterno
- Moduli aggiuntivi
  - Contatori e temporizzatori
  - Moduli di comunicazione: (USART, I2C, SPI, USB, Ethernet,IrDA, CAN, Wi-Fi, Zigbee)
  - Interfacce analogiche o tecnologia mista: ADC, DAC, PWM, Comparatori analogici
  - Interfacce di visualizzazione e controllo: (LCD, Touch sensor)

Da quanto fin qui esposto risulta evidente quale sia **la differenza tra un microprocessore e un microcontrollore:** il primo contiene esclusivamente l'unità centrale di calcolo (CPU) e quindi per poter funzionare necessita di una memoria ROM esterna nella quale viene scritto il programma, di una RAM per i dati e di alcuni integrati per l'interfacciamento; nei microcontrollori tutto ciò è contenuto all'interno di un singolo chip.

Il principio di funzionamento di un microcontrollore è molto semplice, e coincide con quello di un computer o elaboratore elettronico e può essere riassunto in solo tre operazioni eseguite dalla CPU:

- legge l'istruzione contenuta nella memoria programma;
- la interpreta;
- la esegue.

Riepilogo

Il microcontrollore è un dispositivo che raggruppa su **un unico chip** tutto il necessario per un sistema a microprocessore.

1. CPU (core)
2. Memoria RAM
3. Memoria EPROM, EEPROM, OTP
4. Porte I/O
5. Timers e contatori
6. Porte di comunicazione seriale speciali ("Bus CAN","Bus I2C")
7. Convertitori A/D

1.2 Vantaggi del sistema a microcontrollore

Il successo e l'enorme crescita del Mercato di questi componenti sono dovuti a questi fattori:

1. basso costo (consente di sostituire 1 o più circuiti integrati tradizionali a costo inferiore);
2. ampia scalabilità di prestazioni, di complessità (da 8 pin a 144 pin) e velocità (da 1 MHz a 200 MHz);
3. vasta gamma di dotazioni in periferiche e moduli specializzati;
4. Ridotto il numero di componenti esterni, ovvero semplicità di realizzazione, minore complessità del circuito stampato, minor spazio occupato;
5. facilità di programmazione dovuta anche ai numerosi tool di sviluppo disponibili.
6. Ampia (e spesso libera) disponibilità di librerie, codici di esempio e documentazione
7. riconvertibilità del progetto (riprogrammando il dispositivo);
8. Grande flessibilità applicativa .
9. Brevi tempi di introduzione sul mercato del prodotto finito.
10. risparmio energetico (le versioni CMOS supportano il modo di funzionamento stand-by: è possibile bloccare, *via software*, l'attività della CPU e quindi ottenere correnti di alimentazione molto basse);
11. protezione contro le copiature (la maggiore parte del single-chip offre la possibilità di proteggere da lettura il programma contenuto nella ROM);

1.3 Applicazioni

Le principali applicazioni sono:

- **automotive** (auto e altri mezzi di trasporto), che utilizza decine, in alcuni casi anche centinaia, di componenti per singola unità industriale venduta;
- **telefonia mobile** e **telecomunicazioni**;
- **prodotti medicali;**
- **sistemi acquisizione ed elaborazione dati** (centraline rilevamento dati ambientali);
- **consumer;**
- **automatismi e Demotica;**
- **sistemi di sicurezza;**
- **elettrodomestici.**

Spesso utilizziamo questi dispositivi senza rendercene conto, come per le smartcard delle carte di credito o per le cartoline musicali di auguri.

1.4 DSP

Parallelamente al Microcontrollore, e in continua evoluzione di potenza e di mercato,

esistono i DSP (Digital Signal Processor) che incorporano moduli specializzati nel trattamento in digitale di segnali analogici. I campi tipici di utilizzo sono nel controllo di azionamenti (motori), di componenti per auto e avionica, di trattamento di segnali multimediali (codifica/decodifica audio e video, streaming e, campo principe, nella telefonia mobile.

Il DSP ha tipicamente una struttura a 32 bit, e prossimamente a 64 bit.

1.5 Cenni storici

Il primo computer on-chip ottimizzato per applicazioni di controllo è stato il modello 8048 di Intel, rilasciato nel 1975, con RAM e ROM sullo stesso chip. Questo componente è stato utilizzato in più di un miliardo di tastiere per PC e numerose altre applicazioni.

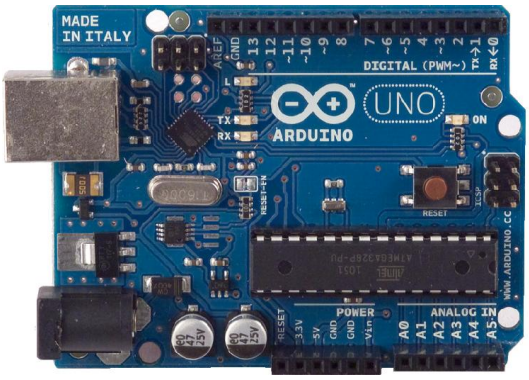
Nei primi anni di sviluppo del Microcontrollore, la maggior parte dei modelli era commercializzata in due varianti. La più economica era dotata di memoria di programma programmata in fabbrica (ROM) su specifiche del cliente oppure programmabile dall'utente una sola volta (OTP, One Time Programming). La seconda, più costosa, aveva la memoria di programma cancellabile EPROM mediante esposizione a luce ultravioletta del chip tramite la finestrella trasparente che lo sovrastava.

Nel 1993, Microchip ha introdotto il modello di MCU PIC16C84, caratterizzato da memoria programma in EEPROM, ovvero cancellabile elettricamente, che permetteva sia lo sviluppo veloce del prototipo del prodotto finito, sia la modifica del Firmware a circuito montato (In-System Programming).
La semplificazione del contenitore (package), senza finestrella in quarzo, ha contribuito a ridurre il costo finale del componente.

Nello stesso anno, Atmel ha rilasciato il primo MCU che utilizzava una memoria di tipo Flash, ancora più semplice e veloce da programmare/modificare, più compatta e con un ciclo di vita (cancellazioni) molto più elevato. Questo ha segnato l'inizio del massiccio utilizzo del Microcontrollore nelle più disparate applicazioni.

2.1 Arduino – Sistema di sviluppo programmabile

Arduino è un sistema (framework) open source che permette la prototipazione rapida e l'apprendimento veloce dei principi fondamentali dell'elettronica e della programmazione.



È composto da una piattaforma hardware per il physical computing sviluppata presso l'Interaction Design Institute, un istituto di formazione post-dottorale con sede a Ivrea, fondato da Olivetti e Telecom Italia. Il nome della scheda deriva da quello di un bar di Ivrea (che richiama a sua volta il nome di Arduino d'Ivrea, Re d'Italia nel 1002) frequentato da alcuni dei fondatori del progetto.

Questa si basa su un circuito stampato che integra un **microcontrollore** con pin connessi alle porte I/O, un regolatore di tensione e un'interfaccia USB che permette la comunicazione con il computer. A questo hardware viene affiancato un ambiente di sviluppo integrato (**IDE**) multipiattaforma (*Linux, Apple Macintosh e Windows*). Questo software permette di scrivere programmi (*sketch*) con un linguaggio semplice e intuitivo derivato da C/C++ chiamato **Wiring** (*cablare, collegare con cavi*). Sito di riferimento (www.arduino.cc).

Arduino può essere utilizzato per lo sviluppo di oggetti interattivi stand-alone (*funzionare da solo*)ma può anche interagire, tramite collegamento, con software residenti su computer, come Adobe Flash, Processing, Max/MSP, Pure Data, SuperCollider.

La piattaforma hardware Arduino è distribuita in versione pre-assemblata, acquistabile in internet o in negozi specializzati. La particolarità del progetto è che le informazioni sull'hardware e soprattutto i progetti sono disponibili per chiunque: si tratta quindi di un hardware open source, distribuito nei termini della licenza Creative Commons Attribution-ShareAlike 2.5. In questo modo, chi lo desideri può legalmente auto-costruirsi un clone di Arduino o derivarne una versione modificata, scaricando gratuitamente lo schema elettrico e l'elenco dei componenti elettronici necessari. Questa possibilità ha consentito lo sviluppo di prodotti Arduino compatibili da parte di piccole e medie aziende in tutto il mondo, e infatti oggi è possibile scegliere tra un'enorme quantità di schede Arduino compatibili.

Ciò che accomuna questi prodotti inerenti elettronica sperimentale e sviluppo è il codice sorgente per l'ambiente di sviluppo integrato e la libreria residente che sono resi disponibili, e concessi in uso, secondo i termini legali di una licenza libera, GPLv2.

Grazie alla base software comune, ideata dai creatori del progetto, per la comunità Arduino è stato possibile sviluppare programmi per connettere a questo hardware più o meno qualsiasi oggetto elettronico, computer, sensori, display o attuatori.

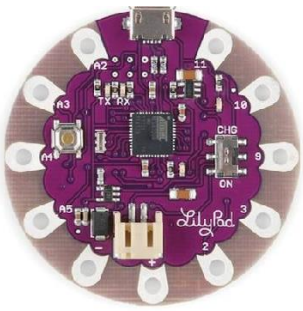
Dopo anni di sperimentazione è oggi possibile fruire di un database di informazioni vastissimo. Sito di riferimento (www.arduino.cc).

Il team di Arduino è composto da Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis. Il progetto prese avvio in Italia a Ivrea nel 2005, con lo scopo di rendere disponibile, a progetti di Interaction design realizzati da studenti, un dispositivo per il controllo che fosse più economico rispetto ai sistemi di prototipazione allora disponibili.

I progettisti riuscirono a creare una piattaforma di semplice utilizzo ma che, al tempo stesso, permetteva una significativa riduzione dei costi rispetto ad altri prodotti disponibili sul mercato. A ottobre 2012 in tutto il mondo erano già stati venduti più di 100.000 esemplari di Arduino.

2.2 Arduino – Hardware

L'hardware originale Arduino è interamente realizzato in Italia dalla Smart Projects, mentre i cloni della scheda possono essere realizzati da chiunque in qualsiasi parte del mondo.
Una scheda Arduino tipica consiste in un microcontroller a 8-bit AVR prodotto dalla Atmel, con l'aggiunta di componenti complementari per facilitarne l'incorporazione in altri circuiti. In queste schede sono usati chip della serie megaAVR - nello specifico i modelli ATmega8, ATmega168, ATmega328, ATmega1280 e ATmega2560.
Molte schede includono un regolatore lineare di tensione a 5 volt e un oscillatore a cristallo a 16 MHz, sebbene alcune implementazioni, come ad esempio la piccola **LilyPad**, abbiano un clock di 8 MHz e facciano a meno dello stabilizzatore di tensione.
Il microcontroller della scheda è pre-programmato con un bootloader che semplifica il caricamento dei programmi sulla memoria flash incorporata nel chip *(Tipo il BIOS del PC)*. Le versioni attuali di Arduino sono gestite via USB: la versione Uno, utilizza un microcontrollore Atmega8U2 programmato come convertitore USB-seriale mentre le precedenti versioni usavano chip adattatori USB-seriale. Alcune varianti, come la Arduino Mini e la versione non ufficiale Boarduino, usano una scheda o un cavo adattatore USB-seriale staccabile.



Per implementare il comportamento interattivo, Arduino è fornito di funzionalità di Input/Output (I/O), grazie alle quali esso riceve i segnali raccolti da *sensori* esterni, tramite un programma memorizzato nel microcontrollore è in grado di pilotare dispositivi esterni (*attuatori*).
A tale scopo, Arduino è dotato di connettori di Input/Output collocati sulla parte superiore della scheda, mediante connettori femmina da 0,1". La direzione di funzionamento, I/O, è decisa dallo sketch programmato sull'IDE.
Inoltre, sono disponibili commercialmente molte schede applicative plug-in, note come "**shields**".

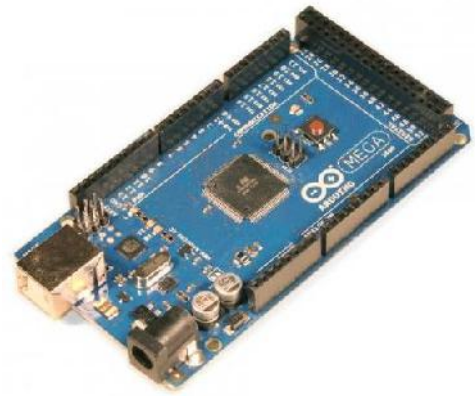
Shield controllo Motori DC e Passo-Passo

RTC-Shield Orologio

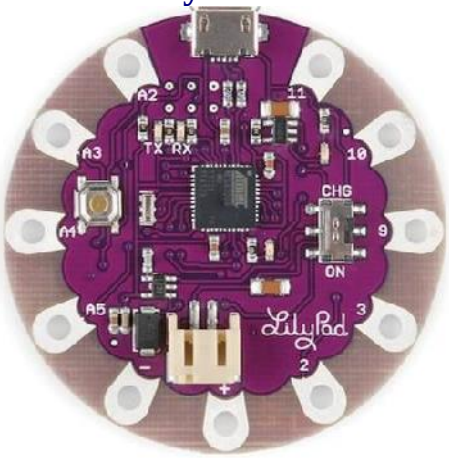
Alcuni canali I/O possono produrre segnali Pulse-width modulation (PWM). Attraverso i segnali PWM è possibile, ad esempio, regolare l'intensità di luminosità di un LED o la velocità di rotazione di un motorino elettrico.
Il sistema è in grado di acquisire segnali analogici, sono presenti pin dedicati (collegati ad un convertitore A/D a 10 bit), i valori di tensione letti da sensori esterni (range: 0V ⎤ 5V), sono convertiti in 1024 livelli discreti (da 0 a 1023).
I connettori dedicati ai segnali analogici possono essere riprogrammati (sempre dal codice dello sketch sull'IDE) per funzionare come normali entrate/uscite digitali.
L'alimentazione della scheda può avvenire attraverso la porta USB del computer, o attraverso la maggior parte degli alimentatori USB, oppure attraverso un adattatore in corrente continua a 9 volt, con connettore cilindrico (diametro 2,1 mm e positivo centrale). In quest'ultimo caso, la scheda commuta automaticamente sull'alimentazione esterna quando il connettore dell'alimentatore esterno è inserito, mentre commuta autonomamente sull'alimentazione USB in caso di disconnessione del connettore. Alcune versioni meno recenti, necessitano di essere commutate a mano, azionando uno switch ubicato tra la porta USB e l'ingresso dell'alimentazione esterna

2.3 Arduino – Versioni

In tabella sono riportati i principali modelli con le rispettive caratteristiche tecniche

Modello	Specifiche Tecniche	
Arduino Uno 	Microcontrollore	ATmega328
	Tensione di funzionamento	5V
	Tensione di Alimentazione (raccomandata)	7-12V
	Massima Tensione supportata (non raccomandata)	20V
	I/O digitali	14 (6 dei quali con uscita PWM)
	Ingressi analogici	6
	Corrente in uscita per I/O Pin	40 mA
	Corrente in uscita per 3.3V Pin	50 mA
	Memoria Flash	32 KB (ATmega328) di cui 0.5 KB usata bootloader
	SRAM	2 KB (ATmega328)
	EEPROM	1 KB (ATmega328)
	Velocità di clock	16 MHz
Arduino Mega 2560 	Microcontrollore	ATmega2560
	Tensione di funzionamento	5 V
	Tensione di ingresso (raccomandato)	7-12V
	Tensione di ingresso (limiti)	6-20V
	I/O digitali	54 (di cui 14 anche come uscite PWM)
	Ingressi analogici	16
	Corrente DC per ogni pin I/O	40 mA
	Corrente DC per pin 3.3V	50 mA
	Memoria flash	256 KB di cui 8 KB utilizzati dal bootloader
	SRAM	8 KB
	EEPROM	4 KB
	Velocità di clock	16 MHz
Arduino Micro 	Microcontrollore	ATmega32u4
	Tensione Operativa	5V
	Tensione di Alimentazione (raccomandato)	7-12V
	Tensione di Alimentazione	6-20V
	Pin I/O Digitali	20
	Canali PWM	7
	Canali Analogici in Ingresso	12
	Massima Corrente Pin I/O	40 mA
	Max corrente per Pin a 3.3V	50 mA
	Memoria Flash	32 KB (ATmega32u4) di cui 4 KB usati dal bootloader
	SRAM	2.5 KB (ATmega32u4)
	EEPROM	1 KB (ATmega32u4)
	Velocità di Clock	16 MHz

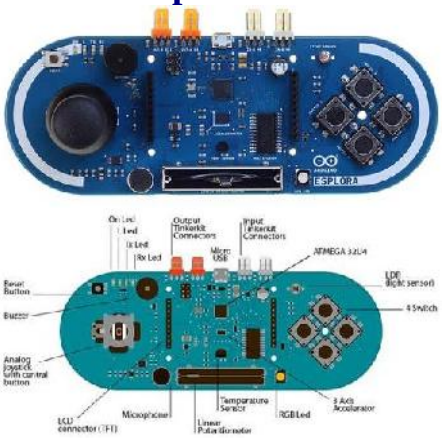
Arduino LilyPad



Scheda circolare dal diametro di 50mm, per circa 8mm di spessore, per applicazione su indumenti

Microcontroller	ATmega32u4
Operating Voltage	3.3V
Input Voltage	3.8V to 5V
Digital I/O Pins	9
PWM Channels	4
Analog Input Channels	4
DC Current per I/O Pin	40 mA
Flash Memory	32 KB (ATmega32u4) of which 4 KB used by bootloader
SRAM	2.5 KB (ATmega32u4)
EEPROM	1 KB (ATmega32u4)
Clock Speed	8 MHz

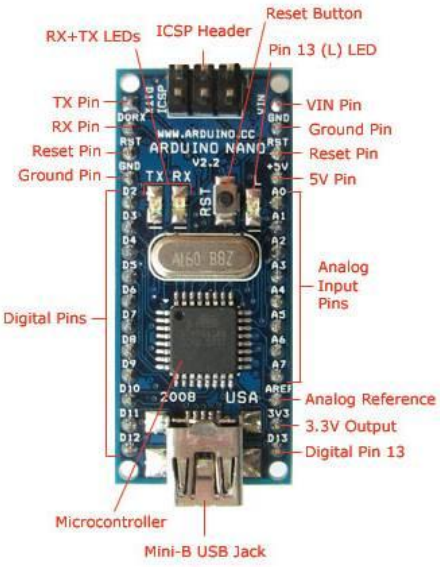
Arduino Esplora



Microcontroller	ATmega32u4
Operating Voltage	5V
Flash Memory	32 KB of which 4 KB used by bootloader
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

Questa scheda ha la forma di un controller da gioco ed è completa di tutto, sono presenti sensori e attuatori (un sensore di luce, uno di temperatura, un accelerometro a tre assi, un joystick, pulsanti, uno slider, un LED RGB e un buzzer). Esplora può emulare un mouse o una tastiera, ed è pensato per creare un controller personalizzato per strumenti di modellazione 3D, software musicali ecc..

Arduino Nano

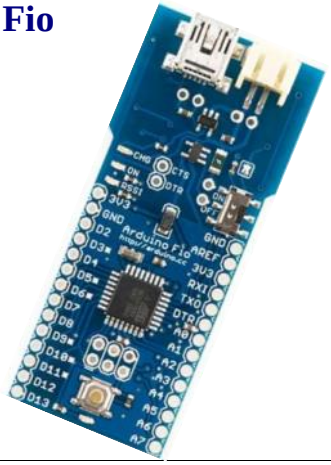


Microcontroller	Atmel ATmega168 or ATmega328
Operating Voltage (logic level)	5 V
Input Voltage (recommended)	7-12 V
Input Voltage (limits)	6-20 V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	8
DC Current per I/O Pin	40 mA
Flash Memory	16 KB (ATmega168) or 32 KB (ATmega328) of which 2 KB used by bootloader
SRAM	1 KB (ATmega168) or 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) or 1 KB (ATmega328)
Clock Speed	16 MHz
Dimensions	0.73" x 1.70"

Arduino Due

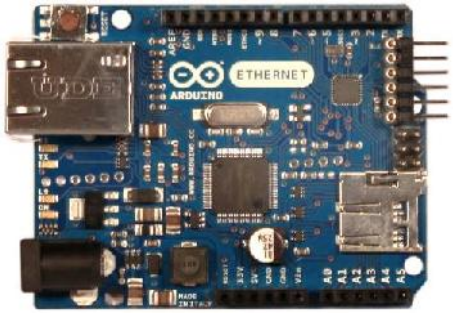
Microcontroller	AT91SAM3X8E (32bit)
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Outputs Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz

Arduino Fio



Microcontroller	ATmega328P
Operating Voltage	3.3V
Input Voltage	3.35 -12 V
Input Voltage for Charge	3.7 - 7 V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	8
DC Current per I/O Pin	40 mA
Flash Memory	32 KB (of which 2 KB used by bootloader)
SRAM	2 KB
EEPROM	1 KB
Clock Speed	8 MHz

Arduino Ethernet

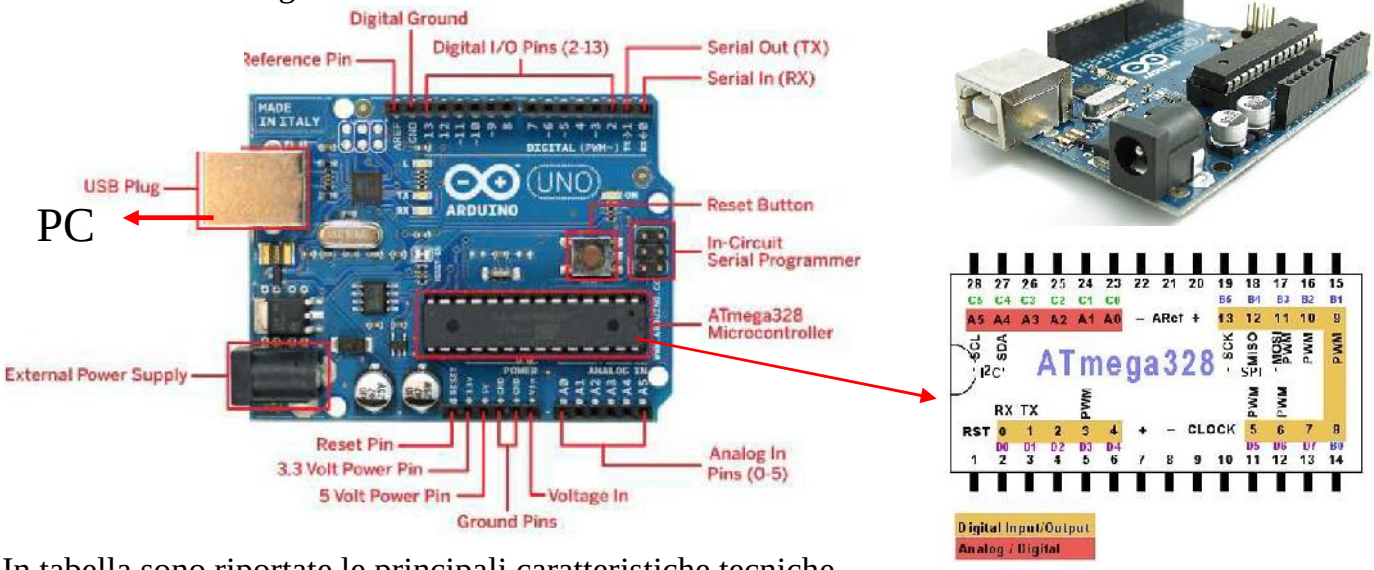


W5100 TCP/IP Embedded Ethernet Controller
Power Over Ethernet ready Magnetic Jack
Micro SD card, with active voltage translators

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage Plug (recommended)	7-12V
Input Voltage Plug (limits)	6-20V
Input Voltage PoE (limits)	36-57V
Digital I/O Pins	14 (of which 4 provide PWM output)
Arduino Pins reserved:	*10 to 13 used for SPI* *4 used for SD card* *2 W5100 interrupt (when bridged)*
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

2.4 Arduino Uno - Descrizione

Arduino UNO è basato sul microcontrollore ATmega328 (in formato DIP, in fig. 2.4.1 e riportato lo schema a blocchi interno) e dispone di 14 pin di I/O (di cui 6 usabili come uscite PWM), 6 ingressi analogici, un oscillatore a 16MHz, un connettore per la programmazione In-Circuit ed un Plug di alimentazione.



In tabella sono riportate le principali caratteristiche tecniche.

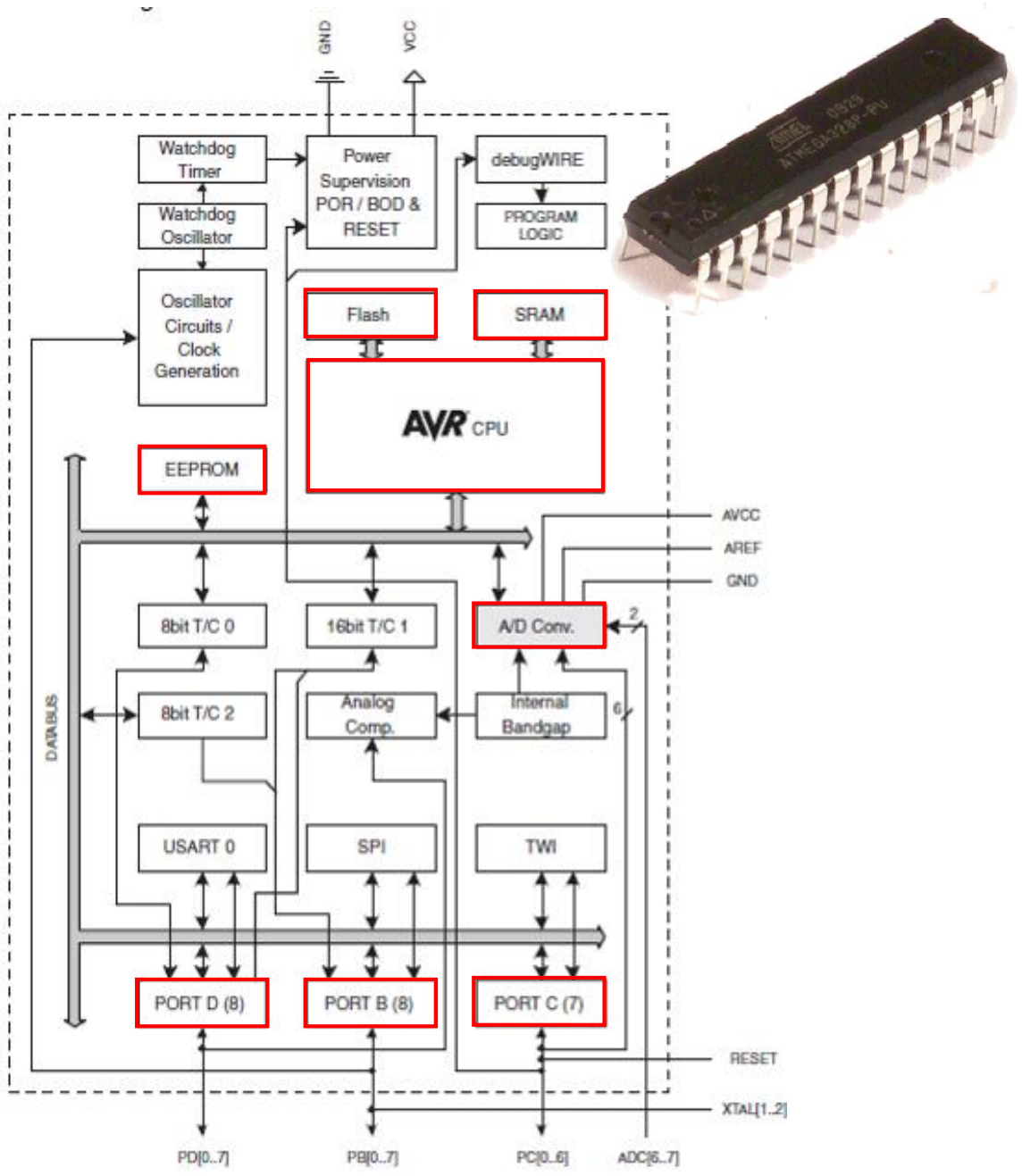
Microcontrollore	ATmega328
Tensione di funzionamento	5V
Tensione di Alimentazione (raccomandata)	7-12V
Massima Tensione supportata (non raccomandata)	20V
I/O digitali	14 (6 dei quali con uscita PWM)
Ingressi analogici	6
Corrente in uscita per I/O Pin	40 mA
Corrente in uscita per 3.3V Pin	50 mA
Memoria Flash	32 KB (ATmega328) di cui 0.5 KB usata bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocità di clock	16 MHz

L'alimentazione alla scheda può avvenire tramite la porta USB ma è disponibile un connettore plug che accetta, in ingresso, una tensione non regolata con valore compreso tra 7 e 12, in questo caso un semplice alimentatore non stabilizzato impostato sul valore di 9volt è l'ideale ma nulla vieta di alimentare la scheda tramite una batteria a 9 o 12 volt.
La sorgente di alimentazione viene riconosciuta in automatico, e nessun deviatore deve essere impostato. La porta USB è comunque protetta da accidentali corto circuiti nella scheda e comunque non vengono prelevati più di 500mA.
Il clock è ottenuto tramite un quarzo a 16MHz e questo stabilirà anche l'intervallo di tempo per l'esecuzione di una istruzione in quanto quasi tutte le istruzioni necessitano di un ciclo di clock per la loro esecuzione.

Descrizione connettori e relative funzioni

VIN	Questo pin replica la tensione fornita in ingresso sul connettore plug. Può essere usato per alimentare altri circuiti che dispongano già di un regolatore di tensione
5V	Questo pin fornisce i 5volt dello stabilizzatore di tensione interno alla scheda. E' utile per alimentare altri circuiti compatibili con i 5volt
3v3	Questo pin fornisce i 3,3volt dello stabilizzatore interno alla scheda. E' utile per alimentare circuiti compatibili con tensioni di 3,3Volt. La massima corrente prelevabile è di 50mA.
GND	Pin di massa (GND).
Memoria:	Il microcontrollore ATmega328 dispone di 32 KB di memoria programma di cui 0.5 KB usati per il bootloader. Dispone in oltre di 2 KB di SRAM e 1 KB di EEPROM utilizzabile, quest'ultima, per il salvataggio di dati permanenti (mantiene i dati anche in assenza di alimentazione).
Ingressi/Uscite	Ciascuno dei 14 pin può essere usato come pin di input o output e gestisce una corrente massima di 40mA, inoltre dispone di una resistenza di Pull-UP del valore di 20-50Kohms (attivabile tramite programmazione).
Seriale:	Pin TX(1) e RX(0). Questi pin fanno capo all'USART interno al microcontrollore e sono connessi al convertitore USB-Seriale della scheda.
Interruptus esterni:	Pin 2 e 3. Questi pin possono essere configurati come trigger per eventi esterni come ad esempio il rilevamento di un fronte di salita o di discesa di un segnale in ingresso.
PWM:	pin 3, 5, 6, 9, 10, e 11. Questi pin possono essere configurati via software per generare segnali PWM con risoluzione di 8 bit. Tramite un semplice filtro RC è possibile ottenere tensioni continue di valore variabile.
SPI:	Pin 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Questi pin possono essere programmati per una comuniazione SPI
LED:	Pin 13. Questo pin è connesso ad un LED interno alla scheda utile per rapide diagnostiche.
Ingressi analogici	Arduino UNO dispone di 6 ingressi analogici A0,A1,A2,A3,A4,A5 con risoluzione di 10bit (Conv. A/D a 10 bit) e tensione di ingresso 0-5volt di default; tuttavia è possibile usare l'ingresso Aref per modificare il range di misura.
I^2C:	4 (SDA) and 5 (SCL). Pin usati per la comunicazione nello standard I^2C (due fili) in abbinamento alla libreria Wire.
Reset.	Portato a livello logico basso resetta il microcontrollore. Questa funzione può essere attivata anche tramite il pulsante presente nella scheda.
Comunicazione	Il microcontrollore ATmega328 utilizza il modulo UART interno per comunicare, con livelli logici 0-5volt, via seriale con altri dispositivi o con il PC. Questi segnali sono disponibili sui pin esterni (TX e RX) e sono connessi anche al convertitore USB-Seriale della scheda permettendo una comunicazione tramite la porta USB del PC. A differenza del chip della FDTI per il quale era necessario installare appositi driver, con l'utilizzo dell'integrato ATmega8U2 questo non è più necessario in quanto vengono usati i driver comuni della periferica USB già disponibili con il sistema operativo. Tuttavia, con sistemi operativi Windows per la corretta creazione di una porta COM virtuale, è necessario installare un driver aggiuntivo.
Compatibilità	Windows, Mac OS X e Linux

Protezione da una sovra-alimentazione della USB	Arduino ha un polifusibile autoripristinante che protegge la porta USB del computer da corto circuiti e sovra-alimentazione. Anche se la maggior parte dei computer già prevedono una loro protezione interna, il fusibile fornisce un ulteriore livello di protezione. Se più di 200 mA attraversano la porta USB, il fusibile automaticamente interromperà la connessione fino a quando il corto o il picco non sia rimosso.
Bootloader	Il microcontrollore ATmega368 della scheda Arduino Uno ha un bootloader precaricato che permette di caricare il codice senza la necessità di uno specifico programmatore esterno e comunica utilizzando il protocollo originale STK500. Si può naturalmente evitare l'utilizzo del bootloader e programmare la ATmega368 attraverso il connettore ICSP (In-Circuit Serial Programming).



2.5 Schede Arduino compatibili

L'enorme quantità e l'estrema variabilita d'uso e di componenti rendono difficile definire univocamente una scheda Arduino compatibile. Solitamente, contiene un microcontroller a 8 16 o 32 bit AVR, PIC o ARM, con clock variabile tra gli 1 e 96 MHz. Molte schede incorporano componenti aggiuntivi pensati per i più svariati utilizzi.

2.6 Prototipazione

E' il processo di realizzazione di un oggetto a partire dalla sua ideazione teorica per erificarne il funzionamento o per fornire una piattaforma di debug (test e ricerca errori).
Un prototipo può essere velocemente modificato per testarne le potenzialità e adattarlo a situazioni diverse.

2.7 Gli Shields

Gli Shields sono estensioni hardware che si collegano ai connettori in linea.
Estendono le funzioni base con funzioni specializzate

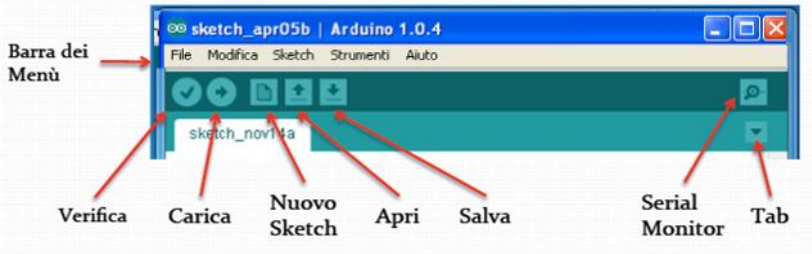
1. Ethernet, con la libreria IP
2. Motor driver, per 2 motori in CC
3. GPS, di fatto un connettore per il modulo della US GlobalSat (EM-406)
4. XBee, per la connessione ZigBee (IEEE 802.15.4, wireless personal area networks - WPAN).
5. MP3
6. Proto, universale, fai da te

3.1. Installazione e configurazione software Arduino

Per programmare la scheda è necessario installare sul PC il software (IDE) Arduino.
L'IDE Arduino è un programma che converte gli sketch in linguaggio macchina per il microcontroller.
Eseguire la seguente procedura.

1. **Download software** dal sito www.arduino.cc (disponibile per Windows, MAC e Linux e non necessita di installazione).
2. **Scompattare il file** - Conservare la struttura delle cartelle. Nelle varie cartelle sono compresi, oltre al sistema di sviluppo (IDE), tutti i file java necessari, i driver per FT232 e gli esempi.
3. **Connettere la scheda** - Come prima applicazione alimentare la scheda direttamente dalla USB; per fare questo è sufficiente inserire il cavo tra la porta USB del PC e la scheda. Non ci sono jumper o deviatori da impostare, quindi il LED di alimentazione (PWR) deve illuminarsi. Appena inserita la scheda, il sistema operativo Windows inizia l'installazione dei driver; con Vista questo passaggio è automatico, in quanto esso ricerca autonomamente i driver e li installa, operazione che richiede alcuni secondi. Per sistemi operativi più vecchi la procedura avviene manualmente. Una volta aperta la finestra di dialogo della richiesta dei driver bisogna fare clic sul pulsante sfoglia e specificare il percorso in cui trovare i driver: nel nostro caso è la cartella FDT USB driver contenuta nei file di Arduino. Fatto ciò si deve avviare l'installazione dei driver. In Windows Seven accedere a "Gestione dispositivi" e verificare lo stato delle periferiche: Se la periferica non è correttamente installata cliccateci sopra con il tasto sinistro del mouse e selezionate la voce proprietà. Utilizzate la funzione "Aggiornamento software driver" per installare manualmente i driver corretti. Se la procedura è andata a buon fine nella lista delle COM vi ritroverete la voce Arduino con indicata la porta assegnata.
4. **Avviare il software** - Aprire la cartella Arduino decompressa e doppio clic sull'icona dell'eseguibile Arduino.
5. **Configurazione di base dell'IDE**
   a. Scelta modello: *Strumenti → Tipo di Arduino → Arduino Uno*
   b. Selezione porta COM: : *Strumenti → Porta Seriale → COMxx*
   c. Impostazione Preferenze: (*File → Preferenze*) (Lingua, carattere, cartella sketch)

In Figura sono riportare le funzioni principali dell'IDE



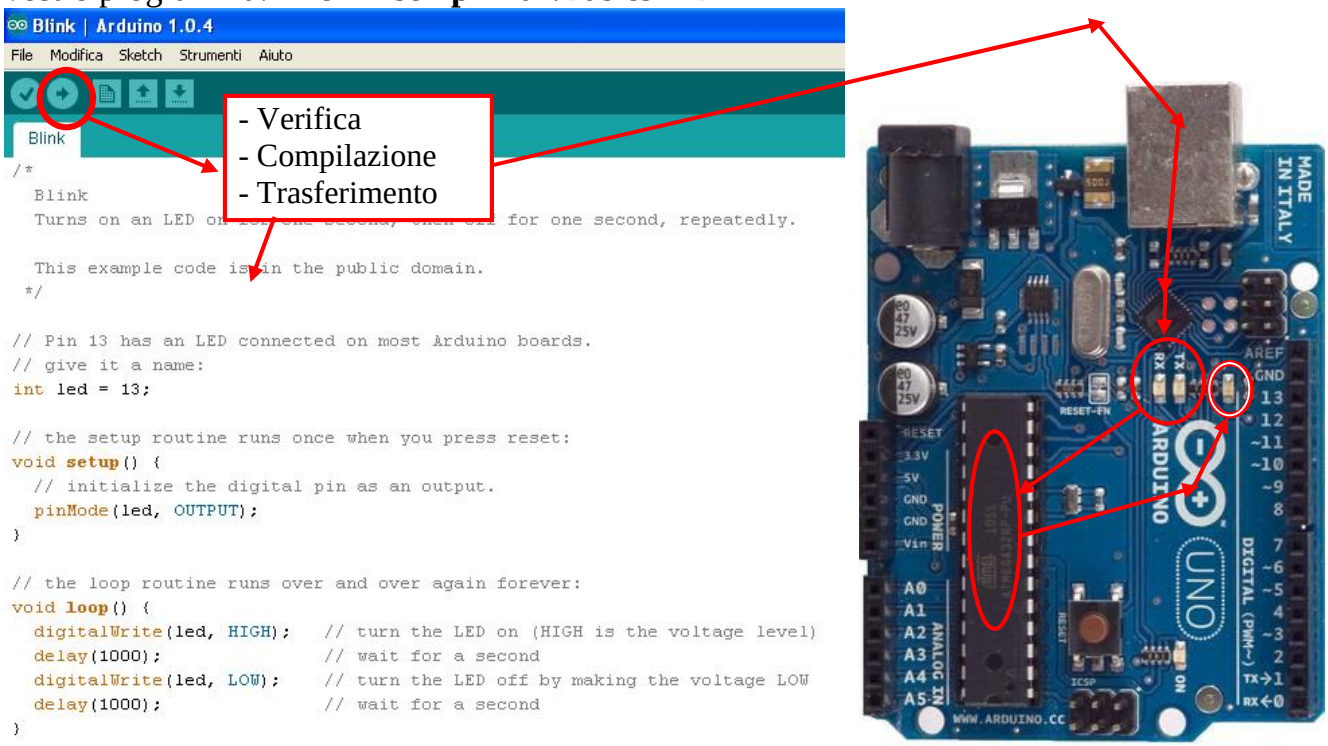
Ciclo di programmazione dell'IDE

1. Scrittura sketch (Programma)
2. Conversione codice in C e compilazione in linguaggio macchina
3. Trasferimento programma compilato sul microcontroller tramite USB
4. Esecuzione codice sulla scheda

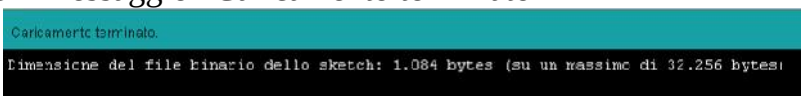
Grazie al Bootloader preinstallato non è necessario utilizzare alcun programmatore esterno ne è necessario rimuovere il microcontrollore, la connessione USB tra PC e Arduino è sufficiente per permettere la programmazione e la gestione della comunicazione. La funzione di autoreset interna alla scheda permette la programmazione con un solo click del mouse.

3.2. Primo Esempio – Lampeggio LED

Lo sketch che fa lampeggiare un LED è il primo programma da eseguire per verificare che la scheda Arduino funzioni e sia configurata correttamente.
Di solito è anche il primo esercizio di programmazione che si svolge quando si inizia a programmare un microcontrollore.
Sulla scheda è presente un LED preinstallato, identificato da una "**L**"
Aprite il programma Arduino, se è la prima volta che avviate il programma verrà immediatamente aperto uno sketch vuoto, altrimenti dal menù **File > Nuovo** aprite un nuovo sketch e salvatelo **File > Salva con nome** (*viene creata una cartella ed un file*).
Il programma Lampeggio Led potete trovarlo sul sito Arduino.cc oppure direttamente dal vostro programma: **File > Esempi > 01.Basics > Blink**



Una volta scritto il programma cliccare sul pulsante carica. L'IDE effettua la verifica, la compilazione ed il trasferimento nella memoria Flash del microcontrollore.
Nell'area vuota in basso nella finestra appaiono alcuni messaggi e, subito al di sopra di quell'area, appare il messaggio "Caricamento terminato"



Sulla scheda ci sono due LED identificati come RX e TX; e questi lampeggiano ogni volta che la scheda invia o riceve un byte. Durante il processo di upload, lampeggiano continuamente. Una volta che il codice si trova sulla memoria programma del microcontrollore, ci rimane finche non vi caricate un altro sketch. Il codice rimane anche se la scheda viene riavviata o spenta.
Se lo sketch è stato caricato correttamente, il LED "L" lampeggia (*si accende per un secondo e poi si spegne per un secondo*).
Quello che avete appena scritto ed eseguito è un programma, o uno sketch, come si chiamano i programmi di Arduino.

3.2.1 Analisi del codice

```
Blink
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
*/

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
```

Questa parte è un commento:

- se il commento si estende su più righe inizia con /* e termina con */
- se il commento si estende su una sola riga, inizia con //

```
int led = 13;
```

Imposta una variabile di nome **led** e all'interno memorizza il numero **13**

Corrisponde la numero del Pin relativo al LED "**L**"

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

void setup()

void indica ad Arduino che stiamo dichiarando una funzione di nome **setup(),** cioè un porzione di codice che esegue un'operazione specifica

La parentesi **{** aperta indica dove inizia il codice.

pinMode(led, OUTPUT)

imposta il pin digitale indicato dalla variabile led (pin n.13) come output. pinMode è un'istruzione che dice ad Arduino come usare un determinato pin. Tra parentesi tonde vengono specificati gli argomenti che possono essere numeri e lettere.

I pin digitali possono essere utilizzati sia come INPUT che come OUTPUT. Nel nostro caso poiché vogliamo far lampeggiare il diodo LED dobbiamo definire il pin di OUTPUT.

Le parole INPUT e OUTPUT sono costanti definite, che non variano mai nel linguaggio di Arduino.

La parentesi **}**graffa chiusa indica la fine della funzione setup().

```
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

void loop()

Indica la sezione di codice principale, il nucleo del programma, che verrà ripetuto all'infinito fino a quando non si spegne la scheda.

La parentesi **{** aperta indica dove inizia il codice.

digitalWrite(LED, HIGH) accende il LED

L'istruzione digitalWrite possiede due argomenti, il primo definisce il pin, il secondo indica lo stato. digitalWrite è un'istruzione in grado di impostare un pin definito come OUTPUT ad un valore HIGH o ad un valore LOW, in modo più semplice permette di accendere o spegnere un led connesso al pin specificato nel primo argomento, nel nostro caso LED.

HIGH= +5V LOW=0V

delay(1000) aspetta un secondo

delay() interrompe per un determinato tempo l'esecuzione del programma. L'argomento, indica il numero di millisecondi di attesa. Nel nostro caso, 1000, ovvero 1 secondo.

digitalWrite(LED, LOW) spegne il LED

delay(1000) aspetta un secondo

La parentesi **}**graffa chiusa indica la fine della funzione loop().

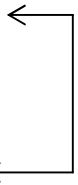
La presenza delle parentesi graffe si usano per raggruppare diverse linee di codice e si rivelano particolarmente utili quando si vuole assegnare un nome a un gruppo di istruzioni.
Nell'esempio, potete notare che sono definiti in questo modo due blocchi di codice
void setup () e *void loop ().*
Dove **void setup ()** è la preparazione, **void loop ()** è l'esecuzione.

void setup()
{
istruzioni che devono essere eseguite una sola volta all'inizio del programma;
Viene utilizzato per inizializzare i pin del microcontrollore e quindi stabilisce i pin di ingresso e di uscita, e/o inizializza la comunicazione seriale.
}

void loop()
{
ciclo in cui viene racchiuso il programma vero e proprio che deve essere eseguito ripetutamente finché la scheda non viene spenta;
}

Per riassumere, ecco cosa fa questo programma:

1. Imposta come output il pin 13 (*solo una volta all'inizio*)
2. Entra in un ciclo
3. Accende il LED collegato al pin 13
4. Aspetta un secondo
5. Spegne il LED collegato al pin 13
6. Aspetta un secondo
7. Torna all'inizio del ciclo



Nei prossimi capitoli verranno riportati tutti i dettagli della programmazione

4.1. Programmazione – Concetti base

I microcontrollori sostituiscono in misura sempre maggiore i tradizionali circuiti di regolazione, misura, e controllo. Essi sono caratterizzati da una uguale struttura hardware: CPU, ROM, RAM, PORTE.
Tramite programmazione, i micro si trasformano in integrati dedicati o specializzati alla gestione di una sola applicazione, ne consegue che l'attività maggiore di progettazione non è hardware, bensì software.
Infatti, il costo delle ore di lavoro necessarie al programmatore per "specializzare" il micro, ovvero per scrivere il programma, risulta decisamente superiore al costo del micro stesso, ecco quindi l'esigenza di razionalizzare il lavoro seguendo un criterio di programmazione ben preciso.

1. **Analisi del problema e fattibilità**: identificazione di una possibile soluzione, verifichiamo se le risorse presenti nel micro e il set d'istruzioni consentono di soddisfare tutte le richieste dell'applicazione in esame.
2. **Schema Blocchi/Elettrico**: stabiliamo i collegamenti tra il micro e il mondo esterno, associamo ad ogni porta la relativa funzione e il modo di funzionamento (ad esempio: ingresso, uscita, ingresso analogico, ecc.). Si eseguono calcoli per il dimensionamento dei componenti
3. **Tabella di verità:** associamo ad ogni porta il relativo compito, ad esempio: P1 è un ingresso con resistore pull-up collegato ad un interruttore e risulta attivo quando assume il valore di "0" logico, ecc.
4. **Flow_Chart**: realizziamo lo schema a blocchi (Flow-Chart) del programma, pianificando la sequenza delle istruzioni in funzione di eventi interni o esterni, si trova l'**algoritmo**
5. **Programma**: tramite un Computer con l'ausilio del software IDE si traduce il Flow Chart in programma, rispettando il set di istruzioni del micro, si crea il file sorgente.
6. **Compilazione e memorizzazione**: tramite il software IDE il file sorgente viene compilato e trasferito nella memoria programma del microcontrollore. La scheda Arduino permette tramite la porta USB il collegamento tra microcontrollore e Personal Computer.
7. **Collaudo**: si effettua il collaudo finale del progetto, si inserisce il micro programmato nel circuito progettato, comprensivo di tutte le sue interfacce (clock, reset, alimentazione, I/O) e si verifica se il funzionamento corrisponde alle specifiche richieste dal progetto.

4.2 Algoritmo – Flow Chart

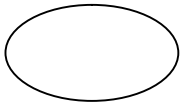
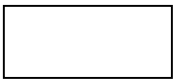
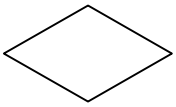
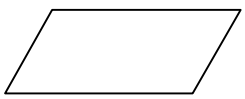
Dato un problema, un **algoritmo** è una procedura, cioè una sequenza di passi, che può essere eseguita automaticamente da una macchina in modo da risolvere il problema dato.
Un problema risolvibile mediante un algoritmo si dice computabile.
L'insieme dei valori permessi ai dati in ingresso si definisce **dominio** dell'algoritmo, mentre l'insieme dei valori che possono assumere i dati in uscita rappresenta il **codominio**.
Un **programma** prende origine comunque da un **algoritmo** che viene descritto in modo sintetico utilizzando un tipo di rappresentazione, la maniera più comune di rappresentazione è quella dei diagrammi di flusso o **Flow-Chart**.
Il Flow-Chart rappresenta l'algoritmo graficamente mediante una lista di istruzioni inserite in appositi simboli grafici, ogni simbolo viene collegato al successivo con delle linee orientate che danno luogo così alla sequenza delle operazioni da effettuare.

Il Flow-Chart da un visione immediata dell'intero algoritmo mettendo in evidenza la sequenza di esecuzione delle varie operazioni.
Un sistema programmabile (computer, microcontrollore, ecc) non ha una propria capacità di "elaborazione creativa", cioè non è in grado di eseguire alcuna operazione se non viene opportunamente istruito. Spetta quindi a noi tradurre il problema in termini formali, individuare dati e incognite (gli elementi non noti, da determinare), schematizzare tutti i passaggi, prevedere tutti i possibili casi che si possono presentare e indicare opportunamente la via da seguire, in modo che la macchina possa arrivare alla soluzione.
L'individuazione di una sequenza ordinata di istruzioni che porta alla risoluzione di un problema viene definita **algoritmo**.
Utilizziamo gli algoritmi, per esempio, quando prepariamo una pietanza in cucina, per montare un giocattolo, per sommare due numeri o per effettuare l'iscrizione a scuola.
Le operazioni definite devono poi essere tradotte in opportuni linguaggi, in modo che le istruzioni possano essere "comprese" dal Sistema programmabile (**esecutore**).
L'informatica si occupa della risoluzione di problemi mediante algoritmi.

Le istruzioni di un algoritmo devono essere:

- interpretabili in un unico modo, non possono essere ambigue;
- elementari; quelle più complesse devono essere suddivise in istruzioni più semplici,in modo che possano essere "capite" più facilmente dal sistema programmabile;
- in numero finito; se così non fosse, come potrebbe un sistema programmabile arrivare a una conclusione? Lavorerebbe all'infinito senza poter restituire alcun risultato.

In tabella sono rappresentati i simboli principali per la stesura di un Flow-Chart.

Simbolo	Descrizione
	Inizio di un programma o di una subroutine, si scrive all'interno il nome del programma o della subroutine, viene anche utilizzato per indicare fine programma o fine subroutine
	Elaborazione dati, esempio calcolo, oppure manipolazione dati, operazione incondizionata
	Blocco decisione, esso contiene una domanda, in funzione della risposta viene deviato il corso del flow-chart, tale blocco è caratterizzato da una una linea d'ingresso e da due diverse linee di uscita (Vero, Falso).
	Chiamata subroutine
	Operazione Input /Output
	Flusso logico

Per realizzare un Flow-Chart scriviamo una o più azioni/comandi all'interno di figure geometriche (ellissi, rettangoli, rombi, ecc) e colleghiamo le figure tra loro con delle linee, aggiungiamo ad ogni linea di congiunzione una freccia, essa indica la sequenza degli eventi, ovvero la sequenza utilizzata dal micro per processare le istruzioni.

4.2.1 Algoritmo non numerico

Prova a pensare alle operazioni che fai quando ti prepari una camomilla:

- scaldi l'acqua,
- sistemi un filtro in una tazza,
- versi nella tazza l'acqua calda,
- lasci in infusione per qualche minuto,
- aggiungi zucchero.

La sequenza di passi appena descritta costituisce un algoritmo, in figura il relativo diagramma di flusso.

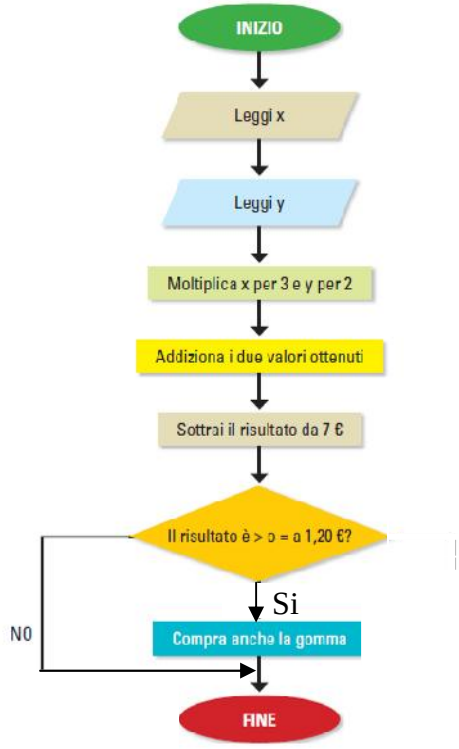
4.2.2 Algoritmo numerico

Creare l'algoritmo che rappresenta il seguente problema:

Devo acquistare tre penne che hanno un costo x e due matite di costo y.
Se ho a disposizione 7 € mi rimangono i soldi per comprarmi anche una gomma da 1,20 €?

- Possiamo definire il seguente algoritmo:
- moltiplica per tre il costo di una penna;
- moltiplica per due il costo di una matita;
- addiziona i due valori ottenuti (totale della spesa per le penne e per le matite);
- confronta la differenza tra quanto hai a disposizione (7 €) e la somma dei due valori: se è maggiore (>) o uguale (=) al costo della gomma (1,20 €) puoi comprarla, altrimenti no.

Quello appena descritto è l'algoritmo espresso in termini informali.
Formalizziamo i passaggi con il seguente diagramma di flusso.



Algoritmo numerico

Algoritmo non numerico

4.3 Gli algoritmi ed i programmi

Un sistema programmabile (Computer, microcontrollore, ecc) è una macchina in grado di elaborare dei dati in funzione di una determinata logica definita da qualcuno. La descrizione della logica in base alla quale il sistema deve elaborare i dati è detta **algoritmo** e consiste in un numero finito di passi da seguire per arrivare alla soluzione. Un algoritmo può essere espresso in varie forme, ma quelle più utilizzate sono essenzialmente due:

- linguaggio naturale
- diagramma a blocchi (Flow Chart)

Esistono essenzialmente due tecniche per la stesura di un algoritmo che hanno caratteristiche opposte

- **Top-Down** è l'approccio più classico. È il metodo secondo il quale chi scrive l'algoritmo deve avere da subito un'idea della struttura dell'algoritmo stesso: la stesura dell'algoritmo inizia dal suo inizio e procede sequenzialmente fino ad arrivare alla fine;
- **Bottom-Up** è l'approccio nato per la stesura di algoritmi relativi a problemi più complessi. Si basa sul motto latino "divide et impera", ovvero il problema viene spezzato in sottoproblemi più facili da risolvere e si inizia a stendere l'algoritmo per questi. Quindi si combinano insieme le varie parti fino ad ottenere l'algoritmo che risolve il problema nel suo complesso.

Nella realtà per la stesura di un algoritmo si fa generalmente uso di entrambe le due tecniche: per lo sviluppo delle parti per le quali è necessario avere una visione d'insieme del problema si utilizza la tecnica top-down, mentre per sviluppare le parti per le quali si sente più l'esigenza di concentrarsi sul dettaglio, si utilizza la tecnica bottom-up.

Un algoritmo, sia esso espresso in linguaggio naturale che sottoforma di diagramma a blocchi, descrive i passi necessari per la soluzione di un problema, ma non è comprensibile da un computer. A tal fine esso deve essere trasformato in un **programma**. Un programma infatti non è altro che l'implementazione di un algoritmo in un linguaggio comprensibile all'elaboratore, cioè una sequenza finita ed ordinata di istruzioni eseguibili in un tempo finito.

La strutturazione dell'algoritmo in più parti porta alla stesura di un programma suddiviso in diversi blocchi.

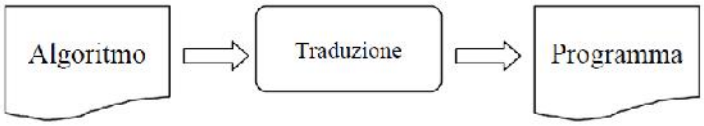
1. Programma principale detto "***main program***"
2. uno o più programmi secondari denominati "***subroutine***".

Per meglio localizzare eventuali errori di stesura del programma è consigliabile sviluppare e provare le *subroutine* singolarmente, e solo quando queste ultime funzioneranno correttamente sarà possibile unirle al main program per testare il programma completo.

È buona norma, inoltre, realizzare le subroutine e catalogarle formando una biblioteca di programmi secondari.

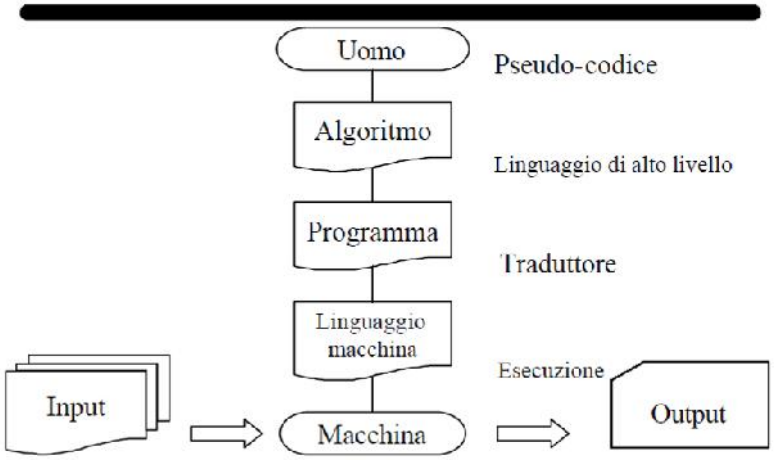
4.3.1 codifica dell'algoritmo

Affinché una macchina riesca a comprendere ed eseguire i passi specificati da un algoritmo quest'ultimo deve essere prima codificato in un opportuno programma scritto in un linguaggio di alto livello.



La CPU è in grado di comprendere solo un particolare insieme di istruzioni: il **linguaggio macchina** (o **codice macchina**) composto da particolari sequenze di valori binari. Scrivere un programma in linguaggio macchina non è né pratico né agevole.
Ecco quindi che sono nati i linguaggi di programmazione, dei linguaggi intermedi, molto vicini al linguaggio naturale (rispetto al linguaggio macchina), che rendono più facile la vita al programmatore. Una volta scritto un programma con il linguaggio di programmazione desiderato esistono degli appositi programmi che fanno la "traduzione" del programma in linguaggio macchina, in maniera da renderlo comprensibile alla CPU e, quindi, eseguibile.

L'esecuzione automatica



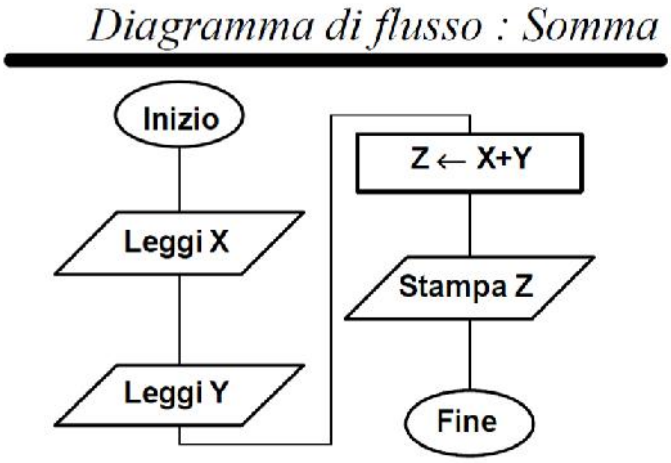
Esistono essenzialmente due metodi per effettuare la traduzione di un programma in linguaggio macchina: l'*interpretazione* e la *compilazione.*

Istruzione di assegnamento

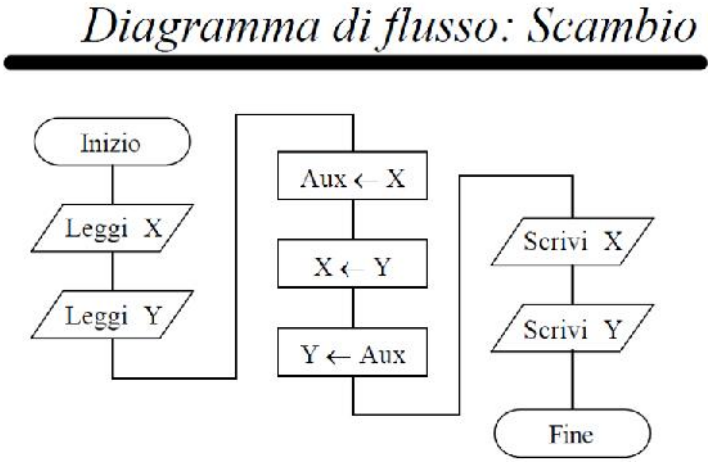
- Una *variabile* numerica è un'entità caratterizzata :
  - da un nome, e
  - da un valore (o contenuto)
    - che può cambiare nel tempo …
- Un'*espressione* è una combinazione di operatori aritmetici, costanti e variabili che può essere calcolata generando un singolo valore numerico.
  - Es.: X, X + 1, X + (Y * 3).
- Istruzione di *assegnamento* : " ← "
  - Variabile ← Espressione;
    - Es.: Z ← 3; Z ← X + 3; X ← X + 1;

4.3.2 Esempi

1) Descrivere, mediante diagramma di flusso, un algoritmo che calcoli la *somma* di due numeri letti in input.



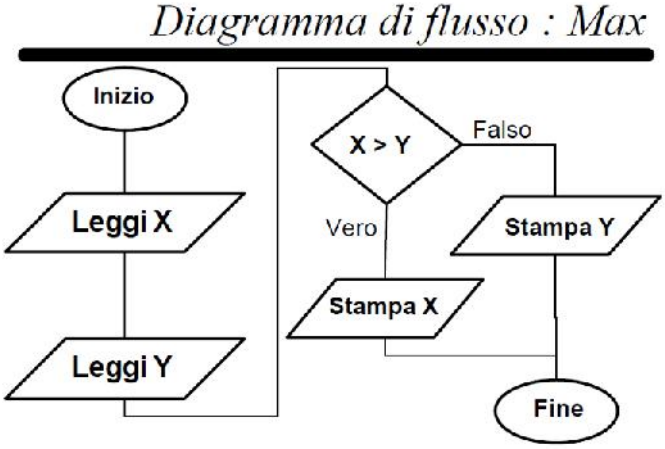
2) Descrivere, mediante diagramma di flusso, un algoritmo che scambi i valori di due variabili lette in input.



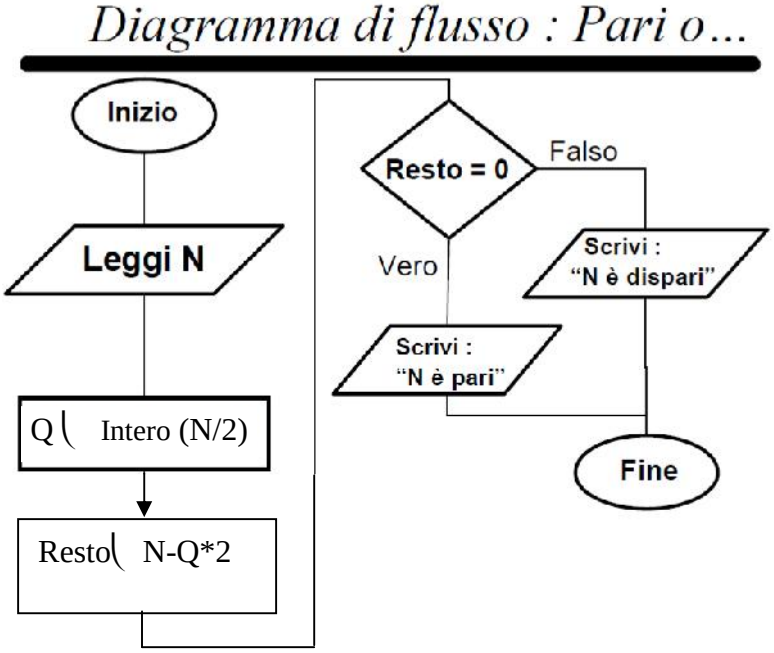
Variazioni nel flusso di esecuzione

- Ci sono dei momenti in cui il flusso di esecuzione può scegliere tra diverse direzioni;
- In genere, questi *salti* sono subordinati al verificarsi di una *condizione* (che può risultare *vera* o *falsa*);
- Si parla di *istruzioni condizionali*.

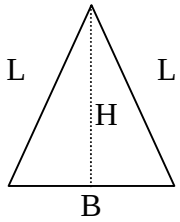
3) Descrivere, mediante diagramma di flusso, un algoritmo che determini il *massimo* tra due numeri letti in input.

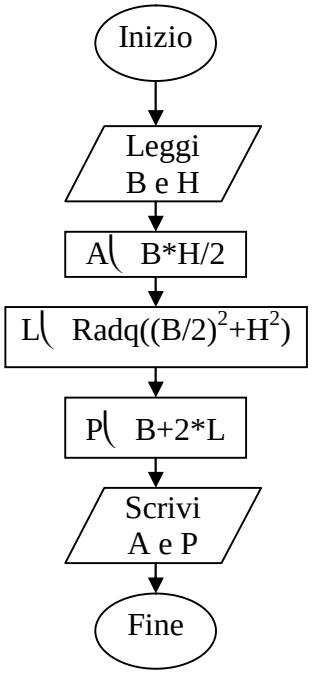


4) Descrivere, mediante diagramma di flusso, un algoritmo che determini se un numero letto in input è *pari* o *dispari*.



5) Descrivere, mediante diagramma di flusso, un algoritmo che determini l'area ed il perimetro di un triangolo isoscele. I dati di input sono la base e l'altezza.





4.4 Linguaggi di programmazione

I linguaggi di programmazione sono un insieme di parole e di regole, definite in modo formale, che consentono la programmazione di un elaboratore elettronico in modo che possa eseguire problemi o, più precisamente, algoritmi. Si definisce algoritmo una sequenza di istruzioni che risolve un problema in un numero finito di passi. È possibile classificare i linguaggi di programmazione in funzione del loro livello di astrazione (cioè di quanto siano rivolti alla macchina piuttosto che all'uomo), in linguaggi di basso, medio ed alto livello. Quello di livello più basso in assoluto prende il nome di linguaggio macchina ed è interpretato direttamente dall'elaboratore; è costituito da sequenze di numeri binari corrispondenti ai set di istruzioni del microprocessore. Nel linguaggio macchina si fa riferimento direttamente all'architettura dei calcolatore e pertanto si raggiunge il massimo livello di efficienza, sia in termini di velocità di esecuzione che di occupazione di memoria, ma i programmi sono quasi del tutto incomprensibili per il programmatore e non sono portabili, cioè non possono essere eseguiti su di una macchina differente da quella per cui sono stati scritti. I linguaggi di alto livello (C/C++, Basic, ecc), perdono sì di efficienza, ma agevolano il lavoro del programmatore, che può implementare più facilmente gli algoritmi. Infatti le parole chiave di un linguaggio ad alto livello sono espresse generalmente da termini in lingua inglese che esprimono l'azione svolta. Un linguaggio di medio livello è il cosiddetto codice mnemonico, nel quale un'istruzione consta in una o più istruzioni in linguaggio macchina. I linguaggi di medio e alto livello necessitano di un interprete specifico per la CPU.

5.1 Struttura del Programma

Introduzione

Alla scheda Arduino (Hardware) viene affiancato un ambiente di sviluppo integrato (IDE) Questo software permette di scrivere programmi con un linguaggio semplice e intuitivo derivato da **C** e **C++** chiamato **Wiring,** liberamente scaricabile e modificabile

I programmi in Arduino sono chiamati **skecth**. Un programma è una serie di istruzioni che vengono lette dall'alto verso il basso e convertite in eseguibile e poi trasferite sulla scheda dall'IDE Arduino.

Struttura

La struttura di base del linguaggio di programmazione Arduino (sketch) è abbastanza semplice e viene eseguito in almeno due parti. Queste due parti necessarie (dette anche funzioni) sono racchiuse nei seguenti blocchi di istruzioni.

```
void setup()
{
        istruzioni che devono essere eseguire una sola volta all'inizio del programma;
}

void loop()
{
        ciclo in cui viene racchiuso il programma vero e proprio che deve essere eseguito
        ripetutamente finché la scheda non viene spenta;
}
```

Dove setup () è la preparazione, loop () è l'esecuzione.

setup()

Il setup () viene eseguito una volta sola appena si avvia il programma. Viene utilizzato per inizializzare i pin del microcontrollore Arduino e quindi stabilisce i pin di ingresso e di uscita, e/o inizializza la comunicazione seriale. Deve essere incluso in un programma, anche se non ci sono istruzioni da eseguire.

```
void setup ()
{
pinMode (pin, OUTPUT);  / / imposta la variabile 'pin' come uscita
}
```

loop ()

La funzione loop () fa girare consecutivamente il programma contenuto all'interno delle parentesi graffe, permettendo al programma di scambiare, rispondere e controllare la scheda Arduino.

```
void loop ()
{
digitalWrite (pin, HIGH);        / / il 'pin' livello Alto
delay (1000);                    / / un secondo di pausa
digitalWrite (pin, LOW);         / / il 'pin' livello Basso
delay (1000);                    / / un secondo di pausa
}
```

Funzioni

Una funzione è un blocco di codice a cui è attribuito un nome, è da intendersi come un blocco di istruzioni che vengono eseguite quando viene chiamata la funzione.

Delle funzioni **void setup ()** e **void loop ()** abbiamo già discusso, parleremo di altre funzioni più avanti.

Le funzioni sono utilizzate per eseguire operazioni ripetitive in modo da ridurre il codice programma ed evitare quindi confusione nel programma stesso.

Le funzioni sono dichiarate all'inizio del programma e specificate dal tipo di funzione.

La struttura della funzione è la seguente:

Tipo nomefunzione (parametri)

Quando si dichiara una funzione in primo luogo bisogna dichiarare il **tipo** della funzione, cioè **il tipo di valore restituito dalla funzione**, quindi se la funzione è di tipo 'int' (intero), vorrà dire che restituirà un valore intero.

Dopo il tipo, occorre dichiarare il nome dato alla funzione e tra parentesi i parametri che vengono passati alla funzione.

Esempio:

```
int delayVal()
```

Il tipo di valore che viene restituito dalla funzione 'int' per la funzione delayVal è di tipo intero. Se nessun valore deve essere restituito il tipo di funzione è nullo.

I tipi di dati (int, byte, long, insigne long) verranno spiegati più avanti.

Esempio: la funzione int per la variabile delayVal() è di tipo intero e viene usato per impostare un valore di ritardo in un programma leggendo il valore di un potenziometro. Prima occorre dichiarare una variabile locale v; viene letto il valore del potenziometro che dà un numero compreso tra 0-1023, poi si divide tale valore per 4 così si ha un valore finale compreso tra 0-255, e infine viene restituito il valore al programma principale.

Esempio:

```
int delayVal()
{
int v;                    // varibile temporanea intera 'v'
v = analogRead(pot);      // lettura del valore del potenziometro
v /= 4;                   // conversione da 0-1023 a 0-255
  return v;               // restituisce il valore finale
}
```

{ } parentesi graffe (si scrivono con Alt+123 "{" e Alt+125 "}" sul tastierino numerico della tastiera)

Le parentesi graffe (noto anche semplicemente come "parentesi" o "parentesi graffe") definiscono l'inizio e la fine dei blocchi di funzione e di blocchi delle istruzioni come il "void loop()" ed anche per le istruzioni "for", "if", "while", etc.

Una parentesi graffa aperta "{" deve sempre essere seguita da una parentesi graffa di chiusura "}": il numero delle parentesi aperte è uguale al numero delle parentesi graffe chiuse. Eventuali anomalie nelle parentesi graffe possono spesso portare a errori difficili da eliminare e rintracciare in un programma di grandi dimensioni.

L'ambiente di compilazione di Arduino include una comoda funzione per controllare il bilanciamento di parentesi graffe. Basta selezionare una parentesi, o anche scegliere il punto di inserimento immediatamente dopo una parentesi graffa, e la sua compagna logica sarà evidenziata.

; punto e virgola

Un punto e virgola deve essere utilizzato per terminare una istruzione e separare gli elementi o istruzioni del programma. Un punto e virgola è usato anche per separare le istruzioni sul contatore all'interno di un'istruzione for.

```
int x = 13;     // dichiarazione della variabile "x" come numero intero 13
```

Nota: Dimenticare di terminare una riga con un punto e virgola si tradurrà in un errore di compilazione. L'errore può essere evidente se si riferisce ad un punto e virgola mancante, altrimenti non risulta facile determinarlo. Se un errore di compilazione è incomprensibile o apparentemente illogico, una delle prime cose da controllare è se manca un punto e virgola al termine dell'istruzione precedente alla linea in cui il compilatore fornisce l'errore.

/*… */ Blocco commenti - Commento su più righe

I commenti a blocco o commenti su più righe sono aree di testo che vengono ignorate dal programma e sono usate per le descrizioni di grandi dimensioni di codice o di commenti che aiutano a far capire ad altre persone tutte le parti del programma. Essi iniziano con "/*" e finiscono con "*/".

```
/*      Questo e' un commento a blocchi
        Effettuato su più linee
*/
```

Poiché i commenti vengono ignorati dal programma e non occupano spazio di memoria dovrebbero essere usati con generosità e devono essere usati per "commentare" i blocchi di codice utili in fase debug ovvero durante la ricerca degli errori.

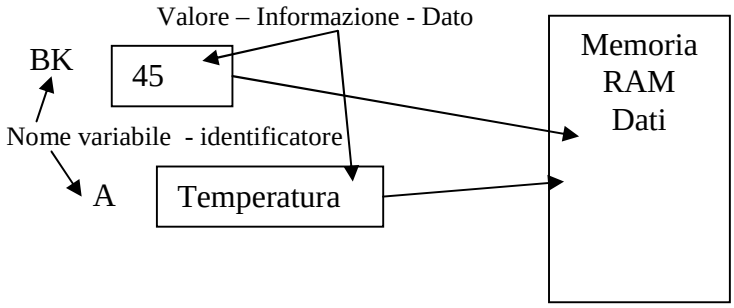
// Singola linea di commento

Il commento sulla riga singola inizia con "//" e termina quando si va a capo. Tutti i commenti vengono ignorati dal programma e non occupano spazio di memoria.

```
// questa è una singola linea di commento
int x = 13; // dichiarazione della variabile "x" come numero intero 13
```

5.2 Le variabili

Nei linguaggi di programmazione si definisce variabile una zona di memoria dati (composta da uno o più bytes) cui è associato un nome e che è utilizzata per memorizzare un valore che può essere modificato da un programma. I nomi mediante i quali è possibile fare riferimento alle variabili prendono il nome di identificatori.
Un identificatore (nome variabile) è costituito da uno o più caratteri il primo dei quali deve essere una lettera e può essere seguito da un qualsiasi carattere alfanumerico.
Le costanti sono invece dei valori che non possono essere alterati durante l'esecuzione del programma.



Le variabili identificano uno o più bytes nella memoria dati (RAM) del microcontrollore. Il numero di byte può variare a seconda del tipo, si hanno differenti tipi di variabili: intere, con la virgola, caratteri, ecc (nella memoria sono comunque numeri binari).
Nel programma bisogna dichiarare di che tipo è la variabile usata prima di utilizzarla (perché occorre dire al compilatore quanto spazio occupa e dove è la variabile nella memoria) e attribuire ad essa un nome: una dichiarazione è quindi una riga di programma del tipo:

```
int calcolo; // indica che la variabile che si chiama calcolo è un intero.
```

Nella dichiarazione si può anche dare un valore iniziale (inizializzare) alla variabile

```
int calcolo=3; // indica che la variabile che si chiama calcolo è un intero e a quel
               // punto del programma vale 3.
```

A differenza di una costante il contenuto di una variabile si può modificare durante il programma quante volte si desidera.
Nota: alle variabili deve essere dato un nome descrittivo che identifica la stessa in maniera univoca per rendere il codice più leggibile. Una variabile può essere chiamata in qualsiasi modo purché non sia già utilizzata come parola chiave nel linguaggio Arduino.
Non è possibile utilizzare lo spazio per identificare il nome delle variabili.
Ad esempio: “sensore A” produce un errore del compilatore che dovrà essere modificato con “sensore_A”.
Una variabile può essere dichiarata in differenti posizioni all’interno del programma e in tal modo si determina quali parti del programma possono utilizzare la variabile stessa (vedi più avanti)

Dichiarazione delle variabili

Tutte le variabili devono essere dichiarate prima di poter essere utilizzate. Dichiarare una variabile significa:

- definire il tipo del valore che può assumere: int, long, float, ecc…
- assegnare un nome
- opzionalmente assegnargli un valore iniziale

queste operazioni vengono fatte una volta sola nel programma, ma il valore della variabile può essere modificato in qualsiasi momento usando l'aritmetica o utilizzando delle assegnazioni.
Nell'esempio che segue viene dichiarata la variabile A di tipo intero e che il suo valore iniziale è uguale a 25: Questo comando è chiamato assegnazione semplice.

int temp=25;

Una variabile può essere dichiarata in posizioni diverse all'interno di un programma e in funzione della posizione in cui viene definita determina quali parti del programma possono utilizzarla.

Visibilità (o portata o scope) di una variabile
Una variabile può essere dichiarata

- all'inizio del programma prima di *void setup()*;
- a livello locale all'interno di funzioni;
- e talvolta, all'interno di un blocco di istruzioni ad esempio all'interno di un ciclo

Variabile globale
Una variabile per essere globale deve essere dichiarata all'inizio del programma, prima della funzione setup().
Una variabile **globale** è una variabile che può essere vista e utilizzata da ogni funzione del programma e in ogni istruzione del programma.

Variabile locale
Una variabile è detta **locale** quando è definita all'interno di una funzione o come parte di un ciclo, essa è visibile e può essere utilizzata solo all'interno della funzione in cui è stata dichiarata. E' quindi possibile avere due o più variabili con lo stesso nome in diverse parti dello stesso programma che contengono valori diversi.
Garantire che solo una funzione ha accesso alle sue variabili semplifica il programma e riduce il rischio di errori di programmazione.
L'esempio che segue mostra come dichiarare diversi tipi di variabili e dimostra la visibilità di ogni variabile.

```
int val;                      // 'val' è visibile da ogni funzione- Variabile Globale
void setup()
{
Istruzioni;
}
void loop()
{
  for (int i=0; i<0; i++)     // 'i' è visibile solamente all'interno del ciclo for – Variabile Locale
  {
Istruzioni;
  }
  float f;                    // 'f' è visibile solo all'interno di loop – Variabile Locale
}
```

5.3 Tipi di dati

void	Tipo utilizzato solo come ritorno di una funzione. Significa che la funzione non riporterà alcun dato come ad esempio la funzione loop().
boolean	Una variabile booleana ha due possibili valori true(vero) o false(falso) e occupa un byte di memoria. *True – False / On –Off / High - Low*

int	La variabile di tipo intero è un tipo di dato, molto utilizzato in Arduino, per memorizzare numeri senza decimali. Può assumere valori negativi. Occupa 2 byte, quindi 16 bit di memoria. Il valore può essere compreso tra -32.768 e 32.767 .
	Esempio:
	int B= 1500; // dichiara 'B' come un tipo intero e gli assegna 1500
	Nota: le variabili intere se raggiungono il valore massimo o minimo in operazioni di aritmetiche o di confronto, ad esempio se x = 32.767 e ad x aggiungiamo 1: x = x + 1 oppure x++
	il nuovo valore di x sarà -32.768.
	Quindi il range da 32.767 a -32.768 è da considerare non come una retta di numeri ma come una circonferenza il cui massimo e minimo sono consecutivi.

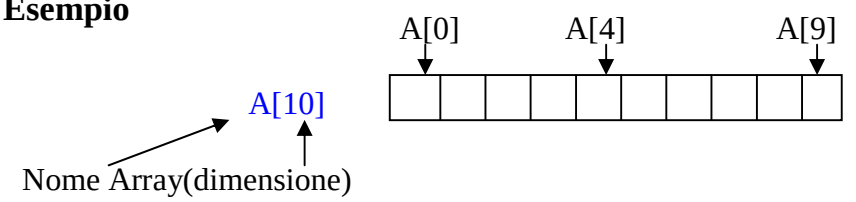
unsigned int	Semplicemente un int senza segno con un range tra 0 e 65.535.
word	Il tipo word ha lo stesso significato di un'insigned int.
byte	il byte occupa 8 bit, può contenere un valore numerico senza decimali e può assumere un valore compreso tra 0 e 255. Esempio: byte A=240
char	Il tipo char contiene uno ed un solo carattere definito secondo lo standard ASCII,quindi qualsiasi lettera (maiuscola o minuscola), cifra (da 0 a 9) e simbolo previsto dalla codifica. Arduino lo conserva in forma di numero, anche se quello che si vede è un testo. Per dichiarare una variabile char, ad esempio inizializzandola con la lettera 'r', basta scrivere: char a = 'r' Può contenere valori compresi tra -128 a 127 e occupa un byte di memoria.
unsigned char	L'unsigned char è uguale al char, però è di tipo unsigned. Comunque, per riferirsi ad una variabile unsigned di 8 bit è preferibile utilizzare byte.
long	Il tipo long è come un int, che utilizza 4 byte (32 bit) per estendere il valore da memorizzare. Infatti il valore che può contenere è compreso tra -2.147.483.648 e 2.147.483.647. Il tipo long non contiene decimali. Esempio: long totale = 90000 // dichiara 'totale' di tipo long
unsigned long	Il tipo unsigned long è un long senza segno. Il suo range va da 0 a 4.294.967.295.
float	Memorizza un numero a virgola mobile compreso tra -3.4028235E+38 e + 3.4028235E+38. Questo tipo di variabile può rappresentare numeri molto piccoli o numeri molto grandi, positivi e negativi con o senza decimali. La precisione dopo il punto decimale è di 7 cifre. I numeri in virgola mobile per la loro maggiore risoluzione rispetto

	agli interi, 4 byte cioè 32 bit. Esempio: float pi = 3,14; Nota: i numeri a virgola mobile non sono esatti, e possono portare a risultati anche strani. I calcoli matematici sono più lenti rispetto ai calcoli con variabili intere. Se possibile le variabili di tipo float sono da evitare.
double	Il tipo double a differenza di altri linguaggi è praticamente un float (quindi su 32 bit invece di 64) senza guadagni in precisione.
string (array di char) **String** (Oggetto)	Il set dei caratteri ASCII può essere usato per contenere informazioni testuali (un messaggio su un display LCD o un messaggio attraverso la comunicazione seriale). Per la memorizzazione viene utilizzato un byte per ogni carattere, più un carattere null (vuoto) per dire ad Arduino che la stringa di caratteri è finita. Il tipo string si può definire in due modi: o come array di caratteri o come un oggetto. Considerandolo come array di caratteri, la stringa è un array di caratteri con l'aggiunta di un terminatore '\0'(facoltativo, verrà aggiunto dal compilatore). **Esempi:** char str1[8]={'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'} char str2[8]={'a', 'r', 'd', 'u', 'i', 'n', 'o'} **char str3[]=”arduino”** Questi sono tre modi per poter creare una stringa. **Per semplicità di scrittura, si preferisce la terza soluzione**

5.3 Array

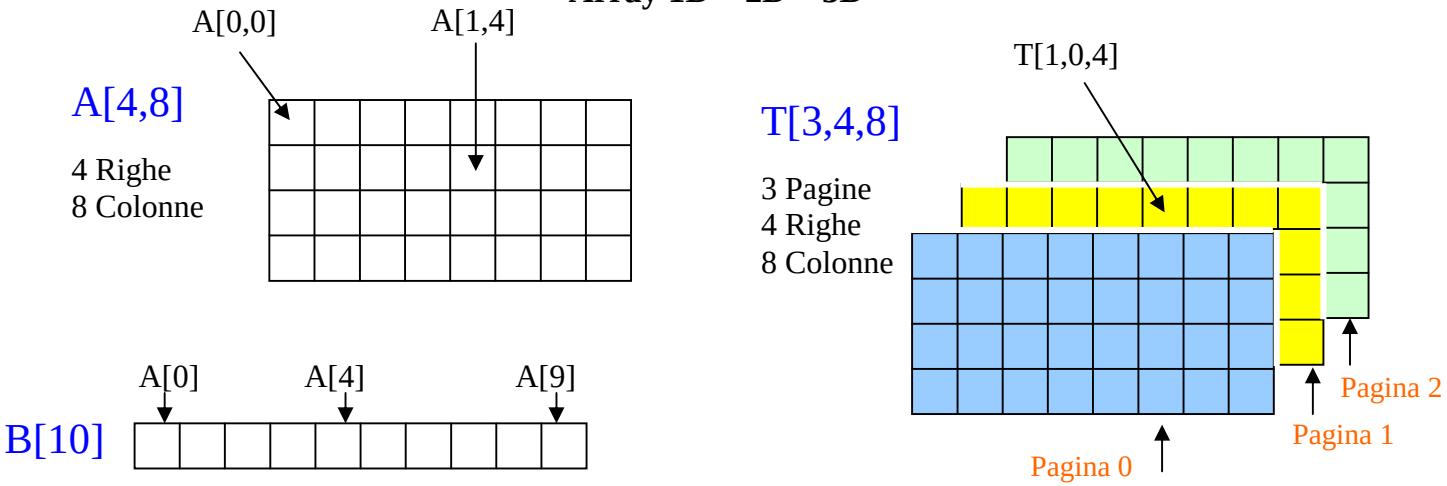
Un Array è costituito da un insieme di caselle (Vettore, matrice, tabella), le singole caselle sono dette celle dell'array. Ciascuna delle celle si comporta come una variabile tradizionale; tutte le celle sono variabili di uno stesso tipo, detto tipo base dell'array.
Ogni valore nell'array può essere richiamato indirizzando il puntatore numerico nell'indice della tabella. Gli array sono indicizzati a partire dal numero zero, infatti il primo valore posto all'inizio della matrice ha come indice numerico proprio il numero 0.

Esempio



Le singole caselle vengono identificate tramite un indice (*costante*, *variabile*) posto tra []

Array 1D – 2D – 3D



Un array deve essere dichiarato e, opzionalmente, possono essere assegnati anche i valori prima di utilizzarlo.
int B [] = {valore 0, valore 1, valore 2, ...}
Allo stesso modo è possibile dichiarare un array dichiarando il tipo di array e la dimensione e poi assegnare i valori:
int A [5]; // dichiara un array di interi di nome A avente 5 caselle
A [3] = 10; // assegna alla casella di indice 3 il valore 10
Per recuperare un valore contenuto in un array, occorre dichiarare una variabile e poi assegnarla all'indice numerico dell'array:
x = A [3]; // ad x verrà assegnato il valore 10

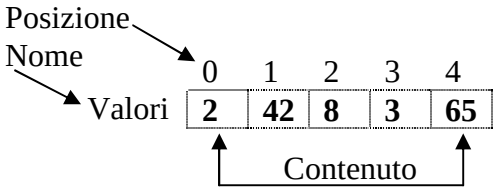
Gli array sono spesso utilizzati nei cicli for, in cui il contatore di incremento viene utilizzato anche come posizione di indice per ogni valore contenuto nella matrice.
Gli array vengono utilizzati per creare tabelle di valori facilmente accessibili. Come esempio se si vuole memorizzare diversi livelli di luminosità di un LED possiamo creare 4 variabili, una per ogni livello di luminosità. Si può utilizzare una semplice array come:

int Luce[5]={0,25,50,100};

Esempi

Un array è una collezione di variabili che sono accessibili tramite un indice numerico.

Creazione e/o dichiarazione di un array

byte Numeri[6];	//dichiarazione di un array di nome **Numeri** adatto a contenere numeri interi (0 ÷255), composto da caselle senza inizializzazione
Int Valori[ ]= {2,42,8,3,65};	//dichiarazione di un array di nome **Valori** adatto a contenere numeri interi (-32.768 ÷ 32.767) senza definirne la dimensione. Il compilatore conta gli elementi e crea un array con 5 caselle ed assegna i rispettivi valori. 
Int Sensori[6]={2,4,-8,3,2,5};	// dichiarazione e inizializzazione di un array (Nome: **Sensori**, Tipo: **Int**, Dimensione: **6** Caselle, Contenuto: **2,4,-8,3,2,5**

char messaggio[5]= “ciao”;	// Dichiarazione ed inizializzazione di un array di tipo char.
	// Occorre un carattere in più per definire il valore null che
	// determina la fine dell'array di caratteri

.

Accesso agli elementi di un arrayAccesso agli elementi di un arrayAccesso agli elementi di un arrayAccesso agli elementi di un array

Gli indici degli array partono dallo 0, cioè il primo elemento occupa la posizione 0.

In un array avente 10 elementi, l'indice 9 rappresenta l'ultimo della lista.

Sintassi per leggere un valore in un array

tipo var = mioArray[n] ;

var indica la variabile di lettura o di accesso;

mioArray[ ] è il nome dell'array di cui si vuole accedere agli elementi
n indica la posizione occupata.

L'esempio seguente utilizza un array per lo sfarfallio di un LED.
Utilizzando un ciclo for, il contatore inizia a scrivere il valore nella posizione 0 dell’indice nell'array **flicker [],** in questo caso 180, al pin PWM 10, pausa per 200ms, poi si sposta alla posizione di indice successiva.

```
int ledPin = 10; // inizializzata la variabile ledPin e assegna il valore 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60}; //memorizza i valori nell’array flicker
void setup()
{
```

```
pinMode(ledPin, OUTPUT);
}
void loop()
{
for(int i=0; i<7; i++) // esegue un ciclo utilizzando i numeri contenuti nell'array
{
analogWrite(ledPin, flicker[i]); // attiva il Pin specificato dalla variabile LedPin con il
                                 //valore contenuto nell'array con la posizione dell'indice "i"
delay(200); // pausa di 200ms
}
}
```

6.1 Operazioni e Logiche

6.1.1 Calcoli aritmetici e formule

Gli operatori aritmetici sono addizione, sottrazione, moltiplicazione e divisione.
L'operazione restituisce la somma (+), differenza (-), prodotto (*), divisione (/) e modulo (%) di due operandi.

Simbolo	Operazione	Esempio	Descrizione
+	Somma	y = k + 3	
-	Differenza	x = y - 7	
*	Prodotto	i = b* 6	
/	Divisione	a = b/5	
%	Modulo	b=a%3	Restituisce il resto della divisione

L'operazione è condotta utilizzando il tipo di dati degli operandi, così, per esempio, 9/4 il risultato è 2 invece di 2,25 perché 9 e 4 sono due interi e quindi non può fornire un risultato con il punto decimale. Questo significa anche che l'operazione può uscire fuori dalla memoria (overflow) se il risultato è più grande di ciò che può essere memorizzato nel tipo di dati. Se gli operandi sono di tipo diverso, il tipo più grande è utilizzato per il calcolo.
Per esempio, se uno dei numeri (operandi) è di tipo **float** e l'altro di tipo **int**, il microcontrollore utilizzerà per il calcolo la virgola mobile (**float**).
Quindi occorre dimensionare in modo opportuno le variabili in modo da contenere il risultato dal calcolo in modo appropriato.
Conoscere prima a che punto la variabile andrà in errore è importante; anche quando ricomincia il conteggio ciclico. Per i calcoli matematici che richiedono frazioni, è bene utilizzare le variabili float, ma con la consapevolezza del loro svantaggio:
grandi dimensioni e velocità di calcolo lento.
Nota: Le variabili possono essere convertite.
Per esempio, **i = (int) 3,6** imposterà **i** uguale a **3**.

6.1.2 Assegnazioni

Si tratta di operatori speciali che si usano per rendere più conciso il codice di programma. Esso combina un'operazione aritmetica con un'assegnazione di variabile.

Esempi:

a = a + 1 si può scrivere a++ a = a +2 si può scrivere a += 2

Attenzione! Se scrivo: value++, prima valuta la variabile value e poi la incrementa di 1; se invece scrivo: ++value prima incrementa di 1 e poi lo valuta. Lo stesso vale per –

Questi operatori speciali si trovano comunemente nei cicli for.

Le assegnazioni più comuni includono:

x + +	// è uguale a x = x + 1,	incrementa la variabile x di +1
x - -	// è uguale a x = x - 1,	decrementa x di -1
x += y	// è uguale a x = x + y,	incrementa x di + y
x -= y	// è uguale a x = x - y,	decrementa x di -y
x *= y	// è uguale a x = x * y,	moltiplica x per y
x / = y	// è uguale a x = x / y,	divide x per y

Nota: per esempio, x *= 3 da come risultato il triplo del valore di x e poi viene riassegnato alla variabile x.

6.1.3 Operatori di confronto

Alcune volte si ha bisogno di confrontare una variabile o una costante con un'altra. Il confronto si uso nelle istruzioni condizionate if, while e for per verificare se una determinata condizione è vera.

Esempi:

x == y // x è uguale a y
x != y // x è diverso da y
x <y // x è minore di y
x > y // x è maggiore di y
x <= y // x è minore o uguale a y
x >= y // x è maggiore o uguale a y

6.1.4 Gli operatori logici o operatori booleani

Gli operatori logici sono un modo per confrontare due espressioni. Si usano anche quando si vogliono combinare diverse condizioni. Restituiscono una funzione TRUE o FALSE.

Ci sono tre operatori logici AND, OR e NOT, che vengono utilizzati in istruzioni *if*:

AND logico:

if (x > 0 **& &** x < 5) // vera solo se entrambe le espressioni sono vere

OR logico:

if (x > 0 || y> 0) // vero se uno delle due espressioni è vera

NOT logico:

if (!x > 0) // vera solo se l'espressione è falsa

COSTANTI

Costanti

Il linguaggio Arduino ha una serie di parole chiave predefinite con valori speciali.
HIGH e **LOW** si usano, quando si vuole accendere o spegnere un pin di Arduino.
Input e **Output** si usano per impostare un determinato pin come ingresso o uscita.
True/false indica l fatto che una condizione o un'espressione è vera o falsa.
True/false - vero / falso
Questi sono costanti booleane che definiscono il livello logico. FALSO è facilmente

definita come 0 (zero), mentre VERO viene spesso definito come 1, ma può anche essere altro eccetto lo zero.
Quindi in un certo senso booleano, -1, 2, e -200 sono anche definiti come VERO.

```
if (b == TRUE);
{
doSomething;
}
```

HIGH / LOW - alta / bassa
Queste costanti definiscono il livello del pin come alto o basso e vengono utilizzati durante la lettura o la scrittura del pin digitale. HIGH viene definito come livello logico 1, ON o 5 volt mentre LOW livello logico 0, OFF o 0 volt.

```
digitalWrite (13, HIGH);
```

Input / Output - ingresso / uscita
Costanti usate con il pinMode () per definire la modalità di un pin digitale come ingresso o uscita.

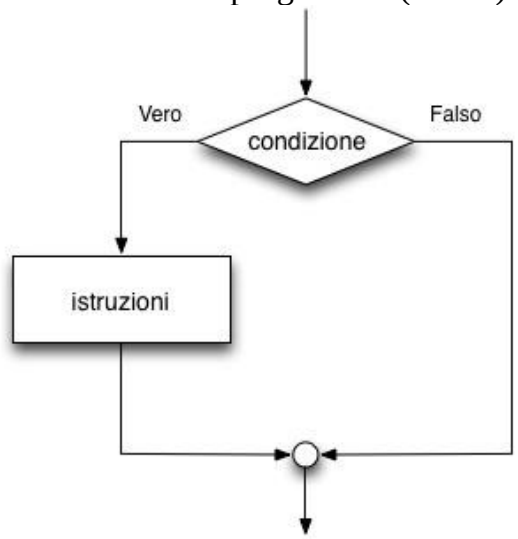
```
pinMode (13, OUTPUT);
```

Controlli di Flusso – Strutture di Controllo

Struttura selezione semplice (If)

La struttura semplice è composta da un blocco "**condizione**" che verifica se una determinata condizione logica si verifica. Se la "**condizione**" risulta "**Vera**", viene eseguita un blocco "**istruzioni**", se la "**condizione**" risulta "**Falsa**", non viene eseguito il blocco "**istruzioni**" e l'esecuzione del programma prosegue oltre l'istruzione. Il blocco *Condizione* è costituito da un'espressione con operatori relazionali (es. A>=45).

Nella stesura del programma (listato) la struttura viene tradotta con l'istruzione **IF**




```
if (A>=45)
{
  istruzioni;
}
```

Operatori di confronto:

```
x == y        // x è uguale a y
x != y        // x è diverso da y
x <y          // x è minore di y
x > y         // x è maggiore di y
x <= y        // x è minore o uguale a y
x >= y        // x è maggiore o uguale a y
```

Operatori logici

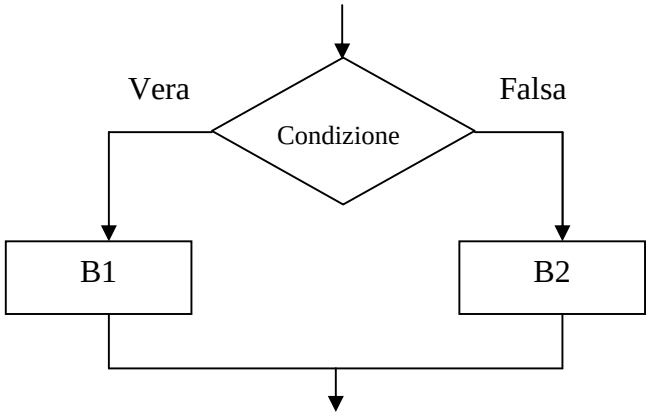
```
(x > 0 & & x < 5)    // AND logico
(x > 0 || y> 0)      //OR Logico
(!x > 0)             // NOT Logico
```

Se la condizione di controllo (A>=45), risulta vera, verrà eseguito il codice incluso tra le parentesi graffe che seguono il controllo, altrimenti se la condizione di controllo risulta falsa, non vengono eseguite le istruzioni dell'if e l'esecuzione del programma continua con la porzione di codice che segue.

Nota: Attenzione all'uso accidentale del simbolo dentro le parentesi dell'istruzione if '=', infatti (x = 10), tecnicamente è un'operazione valida, infatti assegna alla variabile x il valore 10 ed è di conseguenza una condizione sempre vera, per cui lo sketch si comporta in modo diverso da come ci si aspettava.

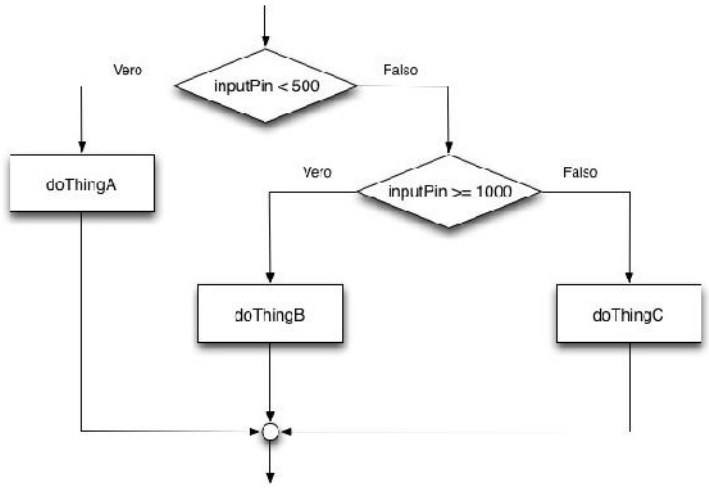
Occorre usare invece il doppio uguale '==', infatti (x == 10), esegue il confronto tra la variabile e il valore, cioè confronta solo e soltanto se x è uguale al valore 10. Occorre pensare a '=' come "assegna a" e '==' a "confronta con".

Struttura selezione doppia (If…else)




```
if (pulsante == HIGH)
{
digitalWrite (rele, acceso);
}
Else
{
digitalWrite(rele, spento);
}
```

La struttura selezione doppia è composta da tre sezioni, la prima è costituita da un blocco "**condizione**" che controlla il verificarsi di una determinata condizione logica, se la condizione risulta "**Vera**" viene eseguito il blocco "**B1**" altrimenti, se "**Falso**" viene eseguito il blocco "**B2**".
Nella stesura del programma (listato) la struttura viene tradotta con l'istruzione **IF …ELSE**
La struttura if ... else è utilizzata per verificare determinate condizioni e quindi eseguire decisioni. Se l'espressione contenuta all'interno delle parentesi tonde è vera, viene eseguito il codice di programma che segue. Se l'espressione è falsa vengono eseguite le righe di codice subito dopo l'istruzione **else**.



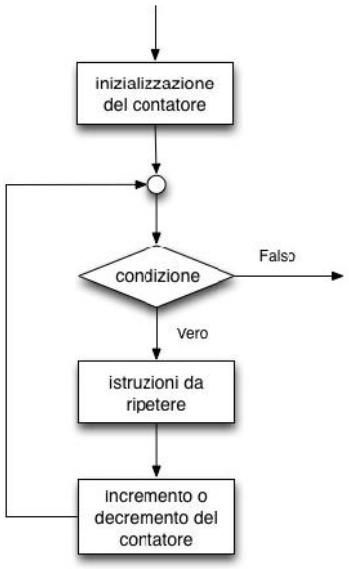
Si possono scrivere condizioni if annidate una dentro l'altra o condizioni multiple.
Attenzione: solo una serie di dichiarazioni verrà eseguito a seconda della condizione posta nello sketch.
L'istruzione else if permette di fare la scelta tra più condizioni.

```
If (inputPin <500)
{
esegui A;
}
else if (inputPin> = 1000)
{
esegui B;
}
else
{
esegui C;
}
```

Ricapitolando.
Il flusso del programma può eseguire una parte di codice oppure no (nel caso del solo if), di fare una scelta tra due parti di codice (nel caso di If – else) o di fare una scelta
tra più parti di codice (nel caso di if – else if – else).

Iterazione Enumerativa (For)

La struttura, tradotta con l'istruzione **for,** viene utilizzata per ripetere un blocco di istruzioni racchiuso tra parentesi graffe un determinato numero di volte. Viene utilizzato un contatore per incrementare e terminare il ciclo.



Sintassi del comando:
for *(inizializzazione; condizione; espressione)*

Esempio:

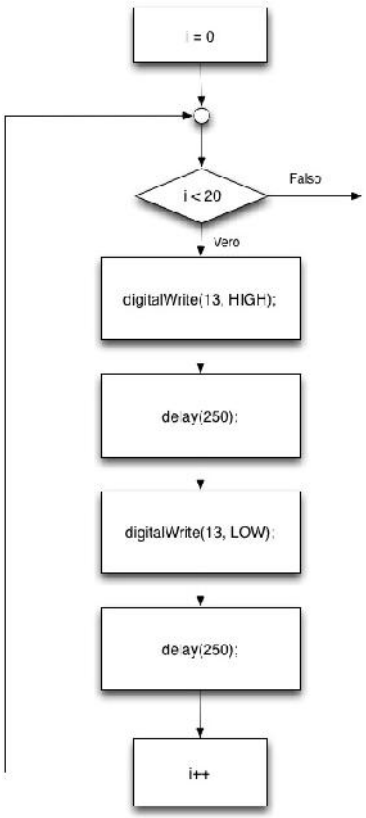
```
for (int A = 0; A < 10; A++)
{
esegui le istruzioni;
}
```

Il contenuto all'interno della parentesi definisce quante volte deve essere ripetuto il blocco di programma contenuto dentro le parentesi graffe.

Inizializzazione:(**Int A = 0**); definisce la variabile di tipo intero e viene impostata con un valore iniziale uguale a zero. L'inizializzazione di una variabile locale, o contatore di incremento, avviene all'inizio e viene definita una volta sola.

Condizione: (**A < 10**);Specifica che finché la variabile A è minore di 10 il ciclo for viene ripetuto perché la condizione è vera, pertanto vengono eseguite le istruzioni contenute all'interno delle parentesi graffe.

Espressione: `(A++);` Incrementa la variabile A di una unità. Il significato è uguale a scrivere A=A+1. Quando la variabile assume il valore di 10, quindi la condizione diventa falsa, il ciclo termina.



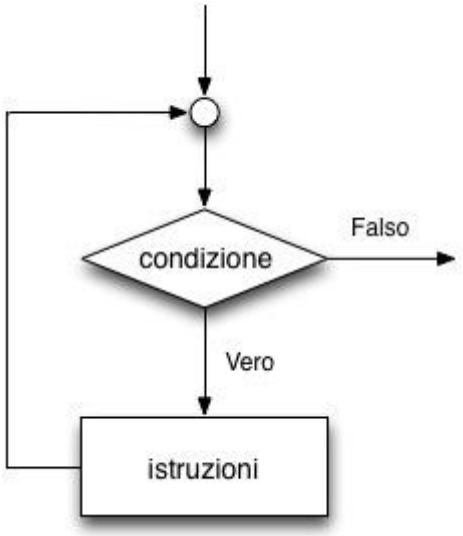
Nell'esempio seguente viene inizializzata la variabile i definita come int e posta uguale a 0; viene avviato il ciclo per vedere se i è ancora inferiore a 20 e finché non risulta vera, la variabile i viene incrementata di 1 e vengono eseguite le istruzioni racchiuse
tra parentesi graffe:

```
for (int i = 0; i <20; i ++)   // dichiara i, controlla se i< 20, incrementa i di 1
{
digitalWrite (13, HIGH);       // abilita il pin 13 come uscita alta
delay (250);                    // pausa per 1/4 di secondo
digitalWrite (13, LOW);        // abilita il pin 13 come uscita bassa
delay (250);                    // pausa per 1/4 secondo
}
```

Risultato:Il led collegato sul pin 13 lampeggia 20 volte

Struttura iterazione a controllo iniziale (While)

Il ciclo **while** esegue il blocco "istruzioni" fini a quando risulta vera l'espressione logica presente in "**condizione**", quando l'espressione risulta falsa si esce dal ciclo **while**.



Finché condizione *vera* ➔ Esegui

Esempio:

```
while (A<100)     //verifica se A<100
{
Istruzioni;
Istruzioni;
A=A+2             //La variabile viene incrementata di 2
}
```

Il ciclo **while** esegue le istruzioni racchiuse tra le parentesi graffe fino a quando la condizione racchiusa dentro le parentesi tonde diventa falsa.

Struttura iterazione a controllo finale (Do … while)

La struttura si comporta in modo simile al ciclo while ma con la differenza che la "condizione" di test viene eseguita alla fine del ciclo, così facendo le istruzioni vengono eseguite almeno una volta prima di verificare la condizione:

Ripeti B1 finché condizione *vera*

Esempio:

```
do
{
x=x+1;
} while (x < 50);
```

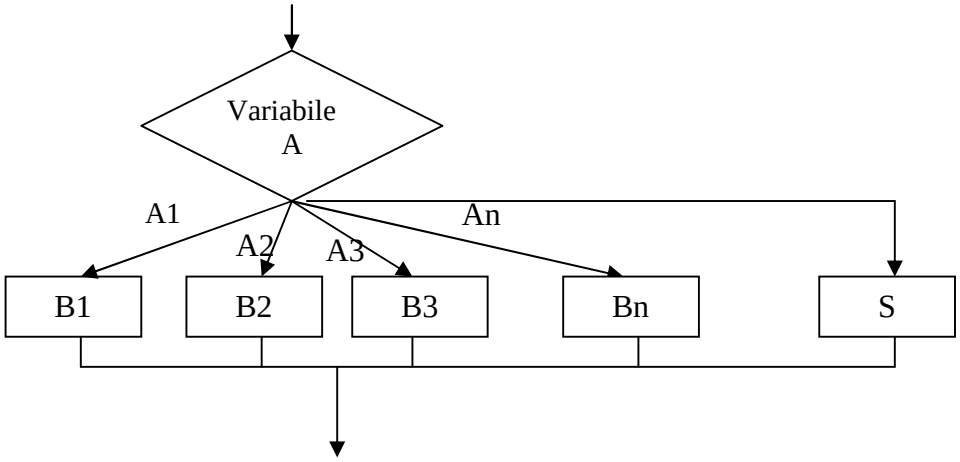
La sintassi del **do…while** prevede che vi sia dopo l'istruzione **do** il blocco "istruzioni" incluso tra parentesi graffe e dopo l'istruzione **while**, a cui segue la condizione di controllo:

Altro esempio:

```
do
{
digitalWrite(13,High);
delay(250);
digitalwrite(13,High)
delay(250);
sensor= analogRead(0);
} while (sensor < 511);
```

Struttura selezione multipla (switch ..case)

La struttura selezione multipla viene utilizzata quando in base al valore assunto da una variabile debbano essere prese decisioni diversificate.



Il controllo switch si comporta come più if in cascata. È utilizzato per quei casi in variabile può assumere più valori che devono essere controllati tutti (es. Che tasto premuto?)

```
switch (var)
{
case 1:
Istruzioni; // Esegue Istruzioni se var=1
break;

case 2:
Istruzioni; // Esegue Istruzioni se var=2
break;

default:
Istruzioni; // Esegue il blocco default se var diverso da 1 e 2,  default è opzionale
}
```

L'esecuzione di uno dei blocchi alternativi B1, B2, Bn dipende da una variabile di controllo (Es. A), che può assumere uno qualunque dei valori dell'insieme ∑A1, A2, A3, ..An∗.
Se la variabile A non assume alcuno dei valori del predetto insieme, viene eseguito il blocco chiamato S
Nella stesura del programma (listato) la struttura viene tradotta con l'istruzione:

switch …. case

La forma sintattica della struttura decisionale **switch .. case** è la seguente:

INGRESSI E USCITE DIGITALI

pinMode (pin, mode)

Utilizzato in void setup (), serve per configurare un determinato pin e stabilire se deve essere un ingresso o un'uscita.

pinMode (pin, OUTPUT); // imposta il 'pin' come uscita

I pin digitali di Arduino di default sono pin di ingresso se non sono esplicitamente dichiarati come ingressi con l'istruzione pinMode (). I pin configurati come ingressi, si dice, hanno un'alta impedenza.

All'interno dell'integrato Atmega sono già presenti per ogni pin le resistenze di pull-up da 20K□ che abilitati via software permettono di settare i pin come ingresso. Le istruzioni che permettono ciò sono le seguenti:

pinMode (pin, INPUT); // imposta il 'pin' come ingresso

digitalWrite (pin, HIGH); // attiva il pin a livello alto con la resistenza di pull-up

Le resistenze di pull-up sono normalmente utilizzate per collegare gli ingressi come interruttori. Si noti che nell'esempio di cui sopra non viene configurato un pin come uscita, è semplicemente un modo per attivare il pull-up interno.

I Pin configurati come OUTPUT sono detti a bassa impedenza e sono in grado di fornire 40 mA (milliampere) di corrente ad altri dispositivi o circuiti. Questo è una corrente sufficiente per accendere un LED (non dimenticare di collegare una resistenza in serie), ma non è una corrente sufficiente per comandare la maggior parte di relè, bobine, o motori.

I cortocircuiti sui piedini di Arduino o l'eccessiva corrente possono danneggiare o distruggere il pin di uscita, o danneggiare l'intero chip Atmega. È spesso una buona prassi collegare il pin di uscita ad un dispositivo esterno collegando in serie una resistenza da 470□ o 1K

digitalRead (pin)

L'istruzione permette di leggere lo stato di un pin di input e restituisce un valore HIGH se al pin è applicato un tensione o un valore LOW se non è applicato nessun segnale. Il pin può essere specificato come una variabile o costante (0-13).

Value = digitalRead (Pin); // assegna a 'value' il valore prelevato dal pin

digitalWrite (pin, valore)

Attiva o disattiva un pin digitale, quindi l'istruzione pone il pin di uscita a livello logico HIGH o LOW. Il pin può essere specificato come una variabile o una costante (0-13).

digitalWrite (pin, HIGH); // imposta il 'pin' a livello alto

Il seguente esempio legge un tasto collegato a un ingresso digitale e gira su un LED connesso a un'uscita digitale quando il pulsante è stato premuto:

```
int led = 13; // il led è collegato al pin 13
int pin = 7; // il pulsante è collegato al pin 7
int value = 0; // viene definita una variabile per memorizzare il valore letto
21
void setup ()
{
pinMode (led, OUTPUT); // imposta il pin 13 come uscita
pinMode (pin, INPUT); // imposta il pin 7 come input
}v
oid loop ()
{
value = digitalRead (pin); // imposta 'value' pari al segnale prelevato dal pin
digitalWrite (led, value); // imposta il led al valore della variabile value
}
22
```

INGRESSI E USCITE ANALOGICHE

analogRead (pin)

Legge il valore di tensione applicato ad un pin di input analogico con una risoluzione

pari a 10 bit. Questa funzione restituisce un numero intero compreso tra 0 e 1023.
Value = analogRead (pin); // imposta value uguale a 'pin'
Nota: il pin analogico a differenza di quelli digitali, non hanno bisogno di essere prima dichiarati come INPUT o OUTPUT.

analogWrite (pin, value)

Cambia la percentuale della modulazione di larghezza di impulso (Pulse Width Modulation - PWM) su uno dei pin contrassegnati dalla sigla PWM. Sulle nuove schede Arduino con il chip ATmega168, questa funzione è abilitata sui pin 3, 5, 6, 9, 10 e 11. Le schede Arduino più vecchie con un ATmega8 supportano solo i pin 9, 10 e 11. Il valore può essere specificato da una variabile o una costante con un valore compreso tra 0 e255.
analogWrite (pin, value); // abilita il pin di uscita al valore analogico
// della variabile value
Un valore 0 genera una uscita pari a 0 volt al pin specificato; un valore di 255 genera un segnale di 5 volt al pin specificato.
Per valori tra 0 e 255, il valore di uscita sarà compreso tra 0 e 5 volt. Più alto è il valore, più spesso il pin è attivo alto (5 volt). Ad esempio, un valore 64 sarà 0 volt tre quarti del tempo, e 5 volt un quarto del tempo; mentre un valore di 128 sarà a 0 volt la metà del tempo e 5 volt la restante metà del tempo; un valore di 192 fornirà un valore di 0 volt un quarto del tempo e 5 volt tre quarti del tempo.
Poiché questa è una funzione hardware, il pin genererà un'onda quadra a impulsi costante dopo una chiamata all'istruzione analogWrite in background fino alla successiva chiamata analogWrite (o una chiamata a digitalRead o digitalWrite sul pin stesso).
Nota: il pin analogico a differenza di quelli digitali, non hanno bisogno di essere prima dichiarati come INPUT o OUTPUT.
L'esempio seguente legge un valore analogico da un pin di ingresso analogico, converte il valore dividendolo per 4, e fornisce un segnale PWM su un pin PWM:

```
int led = 10; // è collegato un LED con resistenza da 220□ al pin 10
int pin = 0; // un potenziometro o un pin analogico viene assegnato il valore 0
int value; // la variabile value sarà utilizzata per la lettura
void setup () {} // non è necessaria alcuna configurazione
void loop ()
{
value = analogRead (pin); // assegna a value il valore letto sul 'pin'
valore / = 4 // converte il rapporto 0-1023 nel rapporto 0-255
analogWrite (led, value); // il valore PWM viene assegnato al led
}
```

OROLOGIO INTERNO

delay (ms)

Mette in pausa un programma per la quantità di tempo specificato in millisecondi. Il valore 1000 è pari a 1 secondo.
Esempio: delay (1000); // attende un secondo

DelayMicroseconds(us)

Mette in pausa il programma per la quantità specificata di microsecondi.
Esempio: delayMicroseconds (1000); // attende un millesimo di secondo

Millis ()

Restituisce il numero di millisecondi da quando la scheda Arduino ha iniziato l'esecuzione del programma corrente. Il tipo di dato è un unsigned long.
value = Millis (); // imposta la variabile 'value' al numero di millisecondi Millis ()
Nota: Questo numero va in overflow (supera i limiti della memoria per cui ricomincia da zero), dopo circa 9 ore.
Duration = millis() – lastTime; //conta il tempo trascorso a partire da 'lastTime'

FUNZIONI MATEMATICHE

Arduino include molte funzioni matematiche.

Funzione	Descrizione	Esempio	Risultato
min (x,y)	Determina il minimo fra x e y	valore= min(4,45);	4
max(x,y)	Determina il massimo fra x e y	valore= max(8,145);	145
abs(x)	Determina il valore assoluto di x	B=abs(-12); E=abs(12);	12 12
constrain(x,a,b)	Ritorna il valore "x" costretto tra "a" e "b". Se x<a x=a, se x>b x=b Se a™x™b il valore non viene modificato	y=constrain(x,10,20);	10™y™20
map(val, r1,r2,n1,n2)	Associa un valore che sta nel range r1 e r2 in un nuovo range che va da n1 a n2. E' molto utile per processare valori provenienti da sensori analogici.	K=map(val,0,1023,0,255);	0™K™255
double pow(base,exp)	Determina la potenza di un numero. Si deve indicare la base e l'esponente		
Double sqrt(x)	Restituisce la radice quadrata di un numero x.		

-

Double sin(rad) - Restituisce il seno dell'angolo specificato in radianti. Esempio:

Double sine= sine(2); // circa 0.909297370

Double cos(rad) - Restituisce il coseno dell' angolo specificato in radianti.

Double tan(rad) - Restituisce il valore della tangente di un angolo specificato in radianti.

- ".

25

GENERAZIONE NUMERI CASUALI (RANDOM)

RandomSeed (seed)

Imposta un valore o un punto di partenza per generare un numero casuale (funzione random (). Esempio:

randomSeed (value); // assegna a 'value' un valore casuale

Poiché il microcontrollore Arduino è in grado di creare un numero veramente casuale, la funzione randomSeed permette di inserire una variabile, una costante, o altre funzioni casuali, per generare numeri "casuali" ancora più casuali. Ci sono una varietà di possibilità o funzioni, che possono essere utilizzati in questa funzione; può essere

utilizzato il comando millis () o anche analogRead () per leggere il rumore elettrico tramite un pin analogico.

random (max)

random (min, max)

La funzione casuale consente di avere numeri pseudo-casuali in un intervallo specificato di valori minimi e massimi.

```
value = random (100, 200); // assegna a 'value' un numero casuale
// compreso tra 100 e 200
```

Nota: utilizzare questo comando dopo aver usato la funzione randomSeed ().

L'esempio seguente crea un valore casuale tra 0 e 255 e fornisce un segnale PWM su un pin PWM pari al valore casuale:

```
int randNumber; // variabile per memorizzare il valore casuale
int Led = 10; // un LED con una resistenza da 220□ è presente
// sul pin 10
void setup () {} // nessuna configurazione è necessaria
void loop ()
{
randomSeed (Millis ()); // imposta millis () come base per generare un
randNumber = random (255); // numero casuale da 0 a 255
analogWrite (led, randNumber); // uscita segnale PWM
delay (500); // pausa per mezzo secondo
}
```

Serial.begin (rate)

Apre la porta seriale e imposta la velocità di trasmissione seriale per la trasmissione dei dati. La velocità di trasmissione tipica per comunicare con il computer è 9600, anche se sono supportati altre velocità.

```
void setup ()
{
Serial.begin (9600); // apre la porta seriale
} // Imposta velocità di trasmissione a 9600 bps
```

Nota: quando si utilizza la comunicazione seriale, i pin digitali 0 (RX) e 1 (TX) non possono essere utilizzati contemporaneamente.

Serial.println (data)

Stampa i dati alla porta seriale, seguita da un ritorno a capo automatico e avanzamento riga. Questo comando ha la stessa forma Serial.print (), ma è più facile per la lettura dei dati sul monitor seriale.

Convertitore A/D

Arduino dispone al proprio interno un convertitore Analogico/Digitale a 10 bit a 6 canali (A0 – A5) che consente di collegare direttamente al chip dispositivi esterni, i quali generano segnali che abbiano un'escursione di tensione compresa tra 0V⌉5V.
Il segnale analogico viene letto attraverso un piedino di I/O abilitato al funzionamento analogico, convertito in un numero digitale a 10 bit e memorizzato all'interno della memoria RAM (Variabile/Array).

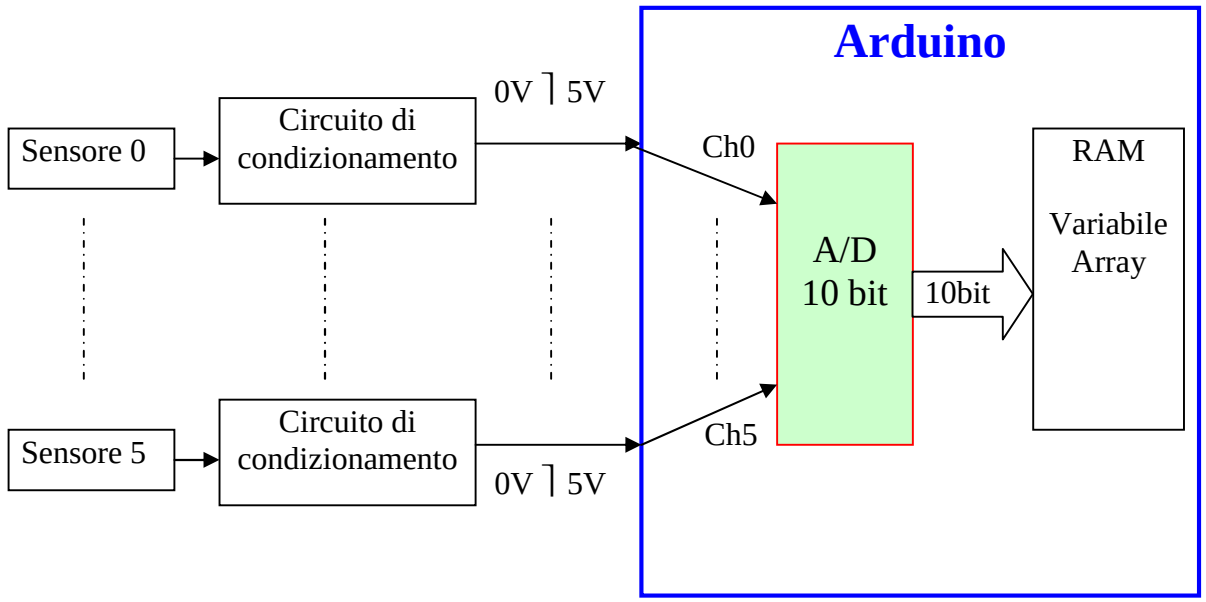
Il convertitore è caratterizzato dai seguenti parametri:

1) Risoluzione digitale 10 bit
2) VFS (tensione fondo scala)=5V
3) Range di Input=0V⌉5V
4) Range di output=0⌉1023
5) Risoluzione analogica $q \text{ X} \frac{Vfs}{2^{10}} \text{ X} \frac{5 \text{ Z} 0}{1024} \text{ X} 4{,}88mV$

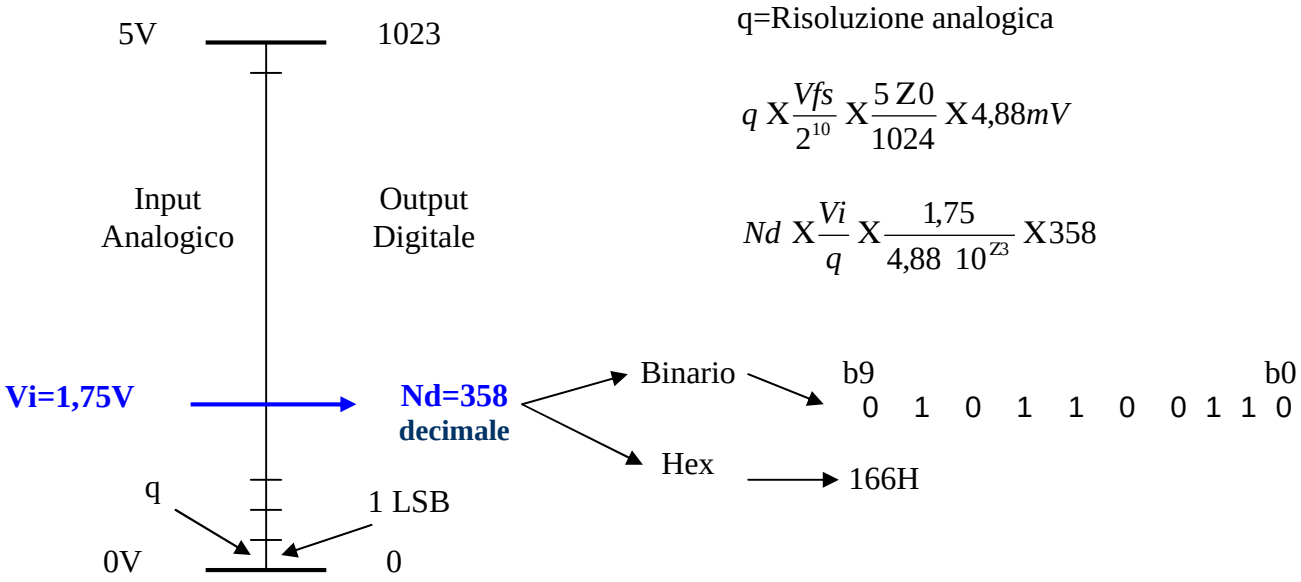
In figura sono evidenziati i pin abilitati al funzionamento analogico.

In tabella è riportata la corrispondenza canale-porta-pin

Canale	Arduino Uno			
	Porta			
A/D Channel 0	A0			
A/D Channel 1	A1			
A/D Channel 2	A2			
A/D Channel 3	A3			
A/D Channel 4	A4			
A/D Channel 5	A5			



In figura è riportato lo schema a blocchi di un sistema in grado di acquisire, convertire e memorizzare 6 grandezze analogiche senza l'ausilio di un multiplexer (Selettore) esterno.
Se i segnali sono superiori a 6 si deve inserire all'esterno di Arduino un multiplexer (Selettore), nelle prossime pagine verrà riportata la struttura di questo sistema.
In figura un esempio di conversione.
Il valore analogico Vi=1,75V viene convertito nel numero binario 0101100110 corrispondente al numero decimale 358 oppure 166H.



q=Risoluzione analogica

$$q \; X \frac{Vfs}{2^{10}} \; X \frac{5 \; Z0}{1024} \; X 4{,}88mV$$

$$Nd \; X \frac{Vi}{q} \; X \frac{1{,}75}{4{,}88 \; 10^{Z3}} \; X 358$$

Comandi

Il convertitore A/D interno viene gestito tramite il comando AnalogRead.

Sintassi di AnalogRead

Variabile=AnalogRead(Canale)

La variabile deve essere di tipo Int, mentre il canale può andare da 0 a 5 (A0 – A5)

Esempio:

```
int analogPin = 3;      // potentiometer wiper (middle terminal) connected to analog pin 3
                        // outside leads to ground and +5V
int val = 0;            // variable to store the value read

void setup()
{
 Serial.begin(9600);   // setup serial
}
void loop()
{
val = analogRead(analogPin);          // read the input pin
Serial.println(val);                  // debug value
}
```