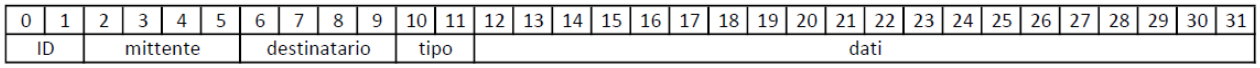
Pacchetto su Seriale: organizzazione dei messaggi, costruzione di un pacchetto a livelli 3-7 del modello ISO/OSI

1. Considerazioni preliminari

Lo scopo di questa attività è riorganizzare il formato dei messaggi scambiati fra i sensori/attuatori e l'applicazione Python, preparando le cose in modo tale che possano funzionare con RF24, che ha un payload di 32 byte.

Si costruirà un pacchetto a livello 3, predisposto per eventuali operazione di routing, e una parte dati a livello 7. Nelle parti che seguono verranno descritti sia il pacchetto e sia la parte dati.

1.1. Struttura del pacchetto e della parte dati



sensore luce/temperatura
motore

S1	valore sensore		non usato
A1	dir	velocità	non usato

Per ragioni di semplicità e debugging, tutti i dati saranno inizialmente rappresentati come caratteri Ascii.

- ***Network id:*** *rappresenta la rete logica; il primo carattere è 'A' o 'B', dipendentemente dalla classe (4ia o 4ib), mentre il secondo carattere la lettera corrispondente al vostro numero di registro (1=A, 2=B, ecc)*
- ***mittente:*** *rappresenta il codice assegnato al mittente, che può essere un sensore o il programma Python, nel caso dell'attuatore.*
- ***destinatario:*** *rappresenta il codice assegnato al destinatario, che può essere un attuatore o il programma Python, nel caso del sensore.*
- ***tipo:*** *il contenuto della parte dati varia dipendentemente dal tipo di messaggio scambiato;* ***S1*** *è il tipo assegnato ai messaggi provenienti dal sensore di luce,* ***A1*** *ai messaggi diretti all'attuatore (motore); nel caso vengano definiti altri tipi di sensori od attuatori, si procederà a definire tipi e strutture adeguati*

Tipo S1: *la parte dati è così strutturata:*

> ***valore sensore:*** *un numero di 4 cifre, riempito di zeri a sinistra*
> ***non usato:*** *riempito di un carattere predefinito*

Tipo A1: *la parte dati è così strutturata:*

> ***direzione: 'A'*** *(avanti-orario) o* ***'I'*** *(indietro-antiorario)*
> ***velocità:*** *un numero di 3 cifre, riempito di zeri a sinistra*
> ***non usato:*** *riempito di un carattere predefinito*

1.2. *Costruzione ed invio del pacchetto in C*

Il compilatore usato dall'IDE di Arduino è un compilatore C/C++ e come tale dispone di oggetti di alto livello tipo String. Nella costruzione del pacchetto abbiamo però necessità di controllare il contenuto del paylod in modo preciso e quindi non possiamo fare uso delle caratteristiche C++ del compilatore ma solo delle caratteristiche C.

Questo significa che non opereremo sulle stringhe usando l'oggetto String (C++) bensì tramite array di caratteri, con o senza carattere terminatore.

Inoltre andremo ad utilizzare nuovi tipi di dato, costruiti tramite ***struct*** *(vedi questo link).*

In particolare la struttura del pacchetto sarà definita come segue:

```
/*****************************************************************
  struttura del pacchetto di 32 byte
*****************************************************************/
struct pacchetto {
  char id[2];
  char mittente[4];
  char destinatario[4];
  char tipo[2];
  char dati[20];
};
```

Quella del tipo S1 sarà definita come segue:

```
/*****************************************************************
  struttura del pacchetto di 32 byte - tipo S1
*****************************************************************/
struct pacchettoS1 {
  char id[2];
  char mittente[4];
  char destinatario[4];
  char tipo[2];
  char valoreSensore[4];
  char vuoto[16]
};
```

Quella del tipo A1 sarà definita come segue:

```
/*****************************************************************
  struttura del pacchetto di 32 byte - tipo A1
*****************************************************************/
struct pacchettoA1 {
  char id[2];
  char mittente[4];
  char destinatario[4];
  char tipo[2];
  char direzione[1];
  char velocita[3];
  char vuoto[16];
};
```

*Per dichiarare una variabile **msg** il cui tipo sia **pacchettoS1** si scriverà:*

```
struct pacchettoS1 msg;
```

*Per copiare qualcosa in uno dei campi di **msg** (ad esempio i caratteri "AB" in **id** si utilizzerà la funzione **memcpy** (vedi questo link):*

***N.B. memcpy** riceve come parametri due puntatori a void (void *: puntatori non tipizzati); **id** è un puntatore ad un'array di caratteri, così come "AB" è una stringa C, quindi un puntatore ad array di caratteri; il casting da **char *** a **void *** viene fatto automaticamente dal compilatore*

```
memcpy(msg.id, "AB",sizeof(msg.id);
```

o meglio

```
#define ID "AB"
memcpy(msg.id,ID,sizeof(msg.id);
```

*Le variabili intere andranno convertite in stringhe di lunghezza opportuna, riempite di zeri sulla sinistra. Per fare ciò è possibile utilizzare la funzione **sprintf**. Ad esempio, per ottenere la stringa "0027" partendo dal numero 27:*

```
int i = 27;
char s[5];
sprintf(s, "%04d", i);
```

*La scrittura su seriale avverrà utilizzando il metodo **write** che permette di controllare il numero di byte che viene scritto. Tale metodo richiede come parametro un puntatore a byte. Volendo ad esempio scrivere **msg** sulla seriale:*

```
Serial.write((byte *)&msg, sizeof(msg));
```

1.3. *Ricezione e decodifica del pacchetto in C*

*La lettura da seriale avverrà utilizzando il metodo **readBytes** che permette di controllare il numero di byte che viene letto. Tale metodo richiede come parametro un puntatore a byte. Volendo ad esempio leggere quanto presente nel buffer della seriale e trasferirlo in **msg**:*

```
Serial.readBytes((byte *) &msg, sizeof(msg));
```

*Ovviamente la lettura avverrà dopo essersi assicurati che nel buffer sia presente un numero di byte maggiore o uguale alla dimensione del pacchetto, utilizzando il metodo **Serial.available()**, che fornisce appunto il numero di bytes disponibili nel buffer.*

*In seguito si dovrà procedere alla verifica del contenuto del pacchetto, in particolare che **id** sia quello voluto e il destinatario sia quello corretto. Ovviamente questi controlli sono privi di senso finché si utilizzerà una connessione punto-a-punto seriale, mentre risulteranno indispensabili quando si utilizzerà RF24.*

*Il controllo dovrà essere effettuato utilizzando le funzioni **strncmp() o memcmp();** non si potrà invece usare **strcmp()** in quanto nel pacchetto sono contenute sequenze di caratteri senza alcun carattere terminatore.*

1.4. *Costruzione ed invio del pacchetto in Python*

*La costruzione del pacchetto in Python avviene utilizzando la libreria **struct** (vedi questo link), che dispone del metodo **pack,** per la costruzione di una stringa di bytes.*

Tale metodo riceve un primo parametro, contenente i parametri di formattazione, e poi l'elenco di tutte le variabili che devono essere concatenate a formare la stringa di bytes.

Nel nostro caso, avendo solo variabili di tipo stringa, potremmo fare a meno di utilizzare tale libreria, ma questo ci permetterà poi di perfezionare la struttura del pacchetto introducendo anche variabili di altro tipo.

Ad esempio, volendo concatenare "AB" come network id, "P001" come mittente e "A678" come destinatario producendo poi un array di byte, si potrebbe semplicemente scrivere, senza fare uso di struct:

```
ID="AB"
MITTENTE="P001"
DESTINATARIO="A678"
msg=ID+MITTENTE+DESTINATARIO
buffer=msg.encode()
```

*Usando invece **struct:***

```
ID=b"AB"
MITTENTE=b"P001"
DESTINATARIO=b"A678"
buffer=struct.pack("2s 4s 4s",ID,MITTENTE,DESTINATARIO)
```

Le variabili passate come parametro devono essere di tipo array di bytes.

*Dovendo convertire un intero in array di bytes, riempiendolo di zeri a sinistra, si potrà utilizzare il metodo **zfill**. Ad esempio, per ottenere l'array di bytes b"027" partendo dal numero 27:*

```
i=27
s= str(i).zfill(3).encode()
```

1.5. *Ricezione e decodifica del pacchetto in Python*

*La lettura da seriale avverrà utilizzando il metodo **read.** Ovviamente la lettura avverrà dopo essersi assicurati che nel buffer sia presente un numero di byte maggiore o uguale alla dimensione del pacchetto testando la proprietà **in_waiting** che fornisce appunto il numero di bytes disponibili nel buffer.*

*L'estrazione dei campi contenuti nel pacchetto avverrà tramite il metodo **unpack**, che funziona in modo analogo al metodo **pack**: riceve la stringa di formattazione e l'array di bytes ricevuto e restituisce un array (o meglio una tupla) con ciascun campo estratto. Ogni campo è sempre un array di bytes.*

*In seguito si dovrà procedere alla verifica del contenuto del pacchetto, in particolare che **id** sia quello voluto e il destinatario sia quello corretto. Ovviamente questi controlli sono privi di senso finché si utilizzerà una connessione punto-a-punto seriale, mentre risulteranno indispensabili quando si utilizzerà RF24.*

2. Struttura dei programmi

2.1. Programma Arduino sensore

- *Lettura dato del sensore*
- *Trasformazione del dato numerico in stringa con zeri a sinistra*
- *Riempimento pacchetto con i dati costanti (id, mittente, destinatario, tipo, vuoto):* ***usare costanti definite con #define***
- *Invio pacchetto*
- *attesa*

2.2. Programma Python ricezione dati sensore

- *Controllo che nel buffer ci sia un pacchetto completo*
- *Lettura pacchetto*
- *Estrazione campi dal pacchetto (unpack)*
- *Controllo che l'id sia corretto, altrimenti il pacchetto è scartato (usare una costante)*
- *Controllo che il destinatario sia corretto, altrimenti il pacchetto è scartato (usare una costante)*
- *Visualizzazione su console del valore ricevuto*
- *Attesa prossimo pacchetto*

2.3. Programma Python di invio comandi al motore

- *Lettura comandi da console*
- *Trasformazione del dato numerico in stringa con zeri a sinistra*
- *Riempimento pacchetto (pack) con i dati costanti (id, mittente, destinatario, tipo, vuoto):* ***usare costanti***
- *Invio pacchetto*
- *attesa*

2.4. Programma Arduino ricezione dei comandi per il motore

- *Controllo che nel buffer ci sia un pacchetto completo*
- *Lettura pacchetto*
- *Controllo che l'id sia corretto, altrimenti il pacchetto è scartato (usare una costante)*
- *Controllo che il destinatario sia corretto, altrimenti il pacchetto è scartato (usare una costante)*
- *Visualizzazione su monitor seriale i valori ricevuti*
- *Controllo del motore*
- *Attesa prossimo pacchetto*

3. Attività da svolgere

3.1. Programma Arduino sensore

Riscrivete il programma in modo che venga inviato il dato del sensore con la struttura del pacchetto sopra discussa.

Assegnate come network id il codice descritto nella sezione *"1.1 Struttura del pacchetto"*, come mittente e come destinatario due indirizzi valori di 4 caratteri a vostra scelta e come tipo il valore "S1".

Provate il programma con il monitor seriale. Prestate attenzione al fatto che, utilizzando per la scrittura su seriale il metodo **write** e non il metodo **println**, si vedranno apparire sul monitor seriale i pacchetti inviati uno di seguito all'altro.

Inizializzate la parte "*vuoto"* del pacchetto con un carattere visibile, in modo da poterlo vedere col monitor seriale.

Non passate a modificare il programma Python se il programma Arduino non funziona perfettamente.

3.2. Programma Python ricezione dati sensore

Riscrivete il programma che riceve il dato dal sensore in modo che riceva e decodifichi il pacchetto ricevuto.

Controllate che il pacchetto abbia come id il valore atteso, altrimenti deve essere scartato.

Controllate che il destinatario del pacchetto sia l'indirizzo che avete stabilito per il programma Python, altrimenti deve essere ignorato.

Estraete quindi il valore del sensore e stampatelo.

3.3. Programma Python di invio comandi al motore

Riscrivete il programma in modo che prepari il pacchetto contenente i comandi per il motore

Assegnate come network id del pacchetto il codice descritto nella sezione *"1.1 Struttura del pacchetto"*, come mittente e come destinatario due indirizzi valori di 4 caratteri a vostra scelta e come tipo il valore "A1".

Stampate su console, a scopo di debugging, il pacchetto che viene inviato su seriale,

Non passate a modificare il programma su Arduino se il programma Python non funziona perfettamente.

3.4. Programma Arduino ricezione dei comandi per il motore

Riscrivete il programma in modo che riceva e decodifichi il pacchetto ricevuto.

Controllate che il pacchetto abbia come id il valore atteso, altrimenti deve essere scartato

Controllate che il destinatario del pacchetto sia l'indirizzo che avete stabilito per l'attuatore, altrimenti deve essere ignorato.

Estraete quindi il valore di direzione e velocita, stampateli a scopo di debugging sul monitor seriale ed inviateli al motore.