0.1 INTRODUCTION TO PROGRAMMING

text for some intro to new lecture unit

0.1.1 Tasks: Make robot move

1. Connest both DC motors to RobDuino controller according to tbl. 1:

Table 1: Motors connections to RobDuino Output pins.

MOTOR	RobDuino Output pins
Left DC Motor - con. A	D7
Left DC Motor - con. B	D6
Right DC Motor - con. A	D5
Right DC Motor - con. B	D4

2. Write simple programming instructions to move the robot forward. Make right sequence of programming instructions (e.g. `digitalWrite()` and `delay(time_in_ms)` functions) to achive:
    1. move the robot forward,
    2. do it for 3000 ms and
    3. stop the robot.

0.1.2 Questions:

You probably ended up with something like prog. 1:

Program 1: Introduction to Programming.

```
void setup()
{
 pinMode(4, OUTPUT);
 pinMode(5, OUTPUT);
 pinMode(6, OUTPUT);
 pinMode(7, OUTPUT);

 digitalWrite(7, HIGH);
 digitalWrite(6, LOW);
 digitalWrite(5, HIGH);
 digitalWrite(4, LOW);

  delay(3000);

  digitalWrite(7, LOW);
 digitalWrite(6, LOW);
 digitalWrite(5, LOW);
 digitalWrite(4, LOW);
}

void loop()
{

}
```

1. Is this code “easy readable”?
2. Why is readable code important?

0.1.3 PROGRAMMING CODE EXPLAINED

1. Zaporedje
2. Izbira
3. Ponavljanje

0.1.3.1 Kako pisati pregledno kodo programa?

- Clean CODE

Proces pisanja kode je izredno NE-linearen, naše misli skačejo na različne težave in različne potrebe, ki se utegnejo pripetiti med programiranjem. Zato ni čudno, da bo prva delujoča koda zapletena in raztresena. Zato jo moramo po končnem testiranju NUJNO urediti.

```
Ko napišete delujočo kodo in ste jo stestirali, ste na pol-poti.
   Potrebno jo je še urediti in narediti berljivo! (Uncle Bob)
```

Organizacija programa naj bo podobna pisanju članka v časopisu:
1. Začnemo z naslovom, nato 2. napišete povzetek 3. nato sledijo odstavki, ki razkrivajo zgodbo v podrobnosti in 4. na koncu je zaključek z rezultati.

Taka ureditev omogoča bralcu, da besedilo lahko zapusti takoj, ko vsaj približno razume namen vsebine. Zamislite si, da berete časopis z novicami, a preberete le tiste, ki vas zanimajo, ostale pa le preletite.

0.1.3.2 Manj je več Krajše koščke programa je lažje razumeti. Zato se moramo potruditi, da vsako zaključeno celoto strnimo v podprogram ali funkcijo.

0.1.3.3 Funkcije Pri funkcijah naj bi se držali nekaj previl:

0.1.3.3.1 Koda v funkcijah naj bo kratka Funkcija naj naredi le eno stvar. To pomeni, da iz kode, ki je v funkciji ne moremo izvleči programske stavke in jih logično ločiti v svojo funkcijo. Seveda pa, moramo vse te majhne funkcije primerno poimenovati.

```
1    Imena funkcije naj bodo GLAGOLI in ne samostalniki, ker funkcije
        OPRAVLJAJO neko nalogo. (Uncle BOB)
```

0.1.3.3.2 Oblikovanje funkcij v razrede Pri oblikovanju funkcij lahko opazimo, da funkcije operirajo s podatki. Če se ti podatki ponavljajo ali pa so podobni moramo razmisliti o uporabi RAZREDA (callses). Naprimer krmiljenje DC motorja je tak primer. Lahko imamo več motorjev in za vsakega posebej želimo nastavljati smer in hitrost. V ta namen bi bilo smiselno pripraviti class:

```
1    class Motor
2    {
3      public:
4        int smer;
5        int hitrost;
6    };
```

0.1.3.4 Da je koda pregledana je verjetno bolj pomembno kot, da deluje... zakaj? Če imamo delujočo kodo in je ta nepregledna, se lahko zgodi, da ko se bodo zahteve spremenile (posodobili bomo program) bomo skušali kodo popraviti in je ne bomo mogli. Da o možnosti, da bi nam jo popravil nekdo tretji sploh ne razmišljamo. Če pa je koda pregledna, pa ne deluje nam jo lahko pomaga rešiti bolj izkušenj programer.

Pregledno kodo lahko uporabi nekdo drug in je prenosljiva.

0.1.3.5 Koda naj gre iz višjega nivoja v nižji

Med posameznimi vrsticami naj ne bo velikih prehodov med nivoji programiranja. Naprimer ne mešajmo deklaracij objektov z deklaracijami konstant.

0.1.3.6 Razlagalna spremenljivka

Te spremenljivke določimo zato, da bodo if-stavki bolj berljivi. Samo spremenljivko določimo predhodno in ji damo tako ime, ki nakazuje na neko logično stanje. Izogibamo se negaciji.

```
1    stikaloJeSklenjeno = digitalRead(3);
2    if (stikaloJeSklenjeno) digitalWtire(3, HIGH);
```

0.1.3.7 Kakšna so naša pričakovanja glede programske kode?

0.1.3.7.1 Vmesne različice naj bodo delujoče

Postaviti si moramo kratke roke ob katerih bomo izdali delujočo kodo. Koda je lahko še podhranjena z uporabnimi funkcijami, a vse funkcije morajo delovati. Izdajanje vmesne različice naj vsebuje: - vse delujoče elemente kode, - njihovo dokumentacijo in - koda naj bo urejena ter - vsak njen del stestiran.

0.1.3.7.2 Dodajanje novih funkcij v program ne sme upočasniti dela

Dodajanje novih funkcij v program ne sme upočasniti dela, če se to zgodi, je verjetno zaradi tega, ker smo pred tem naredili zmedo v programski kodi. Še en razlog več zakaj **mora** biti koda urejena.

0.1.3.8 Spremembe programske kode morajo biti enostavne

Že iz besedne zveze SOFT-WARE je razvidno, da je to MEHAK - IZDELEK in ga je zato enostavno spremeniti. Zato vsaj majhne spremembe ne smejo biti težava in morajo biti hitro implementirane. K temu koraku pripomore zopet: - pregledna koda in - dobro napisan testni program

0.1.3.9 Program naj bo s časom vedno boljši

0.1.3.10 Popravljanje kode brez strahu

Kadar imamo občutek, da bi morali kodo izboljšati, jo dokumentirati ali narediti preglednejšo - imamo verjetno prav. Vendar se tega dela lahko ustrašimo, češ, da bomo kodo morda uničili. Tega se ne smemo nikoli ustrašiti! V veliko pomoč nam je lahko dober testni algoritem kode. Tako brez težav počasi spreminjamo kodo in jo sproti testiramo. Tak proces je zanesljiv in enostaven.

0.1.3.11 Seznanjanje svojega sodelavca s kodo Pametno je seznanjanje svojih sodelavcev z vašim delom (programiranjem) zato, da vas lahko nadomestijo, če ste vi odsotni z dela. Poleg tega pa je to dobra praksa pregleda kode in tako pogosto kodo izboljšamo z idejami sodelavcev.

0.1.3.12 Testiranje kode

1. Ne napiši kode dokler nisi napisal testa zanjo in je le-ta spodletel, ker koda ne obstaja
2. Ne napiši daljšega testa kode, le toliko, da je dovolj, da spodleti.
3. Na napiši daljše kode, le toliko, da popraviš spodleteli test.

0.1.3.13 Arhitektura kode Iz arhutekture kode mora biti jasno za kakšen projekt gre. Podobno kot lahko iz tlorisa stavbe lahko povemo za kater namen je zgrajena. Enako je, če pogledamo kako je urejena arhitektura računalniške matične plošče.

Ker gre pri robotiki v najosnovnejšem primeru za S-R-A loop bi verjetno bilo primerno, da je tudi arhitektura kode taka.

0.1.4 Summary:

0.1.4.1 <++>

0.1.5 Issues:

0.1.5.1 <++>