## Ponavljanje: pullup, pulldown.

Zadano je *7* zadataka nad kojima je potrebno primjeniti dosad stečeno znanje, posebno ono vezano na spajanje jednostavnih vanjskih jedinica koje iniciraju kakvu programsku akciju.

Prouči slijedeći isječak programskog kôda (dostupan i ovdje:pastebin.com/F9b5WsfR):

1. Serial.begin(115200); // Inicijalizacija serijske komunikacije

2. String ulaz = ""; // definicija stringa u koji ce se spremati 3. // primljeni podaci

4.

5. while(Serial.available() != 0); // Cekanje na dostupnost podataka u

6. // meduspremniku (bufferu)

7.

8. while(Serial.available() > 0) { // Dok buffer nije prazan

9. ulaz += (char) Serial.read(); // Pretvoriti jedan byte podataka sa

10. // meduspremnika u znakovni oblik

11.

12. delay(5); // Odgoditi daljnje izvrsavanje programa za

13. // 5ms, kako bi se spremnik stigao napuniti

14. }

15.

16. Serial.println(ulaz); // Izvrsiti operaciju nad primljenim

17. // stringom (npr. ispisati ga)

18.

19. ulaz = ""; // Isprazniti postojece podatke, kako bi se 20. // novi podaci mogli ucitati

**Zadatak 1:** Napisati program *echo.ino* kojemu korisnik na *Serial Monitor* upisuje podatke, a Arduino iste podatke šalje nazad preko serije (koristiti gore navedeni programski kôd).

## Funkcije

Ponekad je programski kôd podijeliti u cjeline. Najčešći oblik podjele programa prema funkcijskim cjelinama jest pisanjem *funckija*: manjih dijelova programa koji imaju zadanu ulogu.

Funkcija je definirana *imenom* (kao i ime varijable), *tipom* povratne vrijednosti, *argumentima* koji definiraju kontekst, odnosno izravno utječu na rezultat funkcije, te *implementacijom* funkcije, koja je zapravo programski kôd kojim je funkcija opisana. Sintaksa je slijedeća:

1. // redom: tip povratne vrijednosti funkcije, ime funkcije, argumenti
2. int mojaFunkcija(int argument1, String argument2, byte nekiTreciAgrument){
3. // implementacija funkcije
4. Serial.println("Ovo i nije neka osobito korisna funkcija");
5. // povratak iz funkcije.
6. return 0;
7. }

Česta apstrakcija funkcija u programiranju je ona matematička. Slijede matematički i programski zapis funkcije (*djelomično!*) jednakog ponašanja:

1. float f(float x){
2. float rezultat;
3. rezultat = x\*x;
4. return rezultat;
5. }

**Zadatak 2:** Napisati funkciju koja služi primanju string-a sa serije (analogno zadatku 1). Pohrani svoj programski kôd kako bi ga mogao koristiti u slijedećim zadacima.

**Zadatak 3:** Napisati funkciju koja pali 5 LED-ica na pinovima zadanih argumentima. Moguće je umjesto svih 5 LED-ica koristiti 4 vanjske i jednu ugrađenu. U nastavku je zadan mogući oblik funkcije:

int paliLED(int pin1, int pin2, int pin3, int pin4);

Ovo zovemo i *prototipom funkcije*, a sâm zapis u potpunosti je podržan u programskim jezicima C i C++. Prototipi funkcija su vrlo korisna stvar, kako nam omogućavaju da na početku programa imamo lijep pregled svih definiranih funkcija. Pregled definicije funkcija često je korisna stvar, a dobro pisana funkcija ne zahtjeva gledanje u programski kôd kako bi se shvatilo što funkcija radi (ali ne nužno i ***kako*** ona radi, kao npr. često korištena funkcija *digitalWrite*). Iz tog razloga, imena funkcije kao i imena njezinih argumenata bi trebali biti što razumljiviji, u kontekstu finkcionalnost. Npr funkcija *writeSomethingQWERTY(int bla1, int bla2)* nije dobro imenovana, dok *sumOfTwoNumbers(int number1, int number2)* jest. Nikad nemojte funkcijama dodavati više funkcionalnost (npr. *sumAndProduct(int a, int b)*), već pišite više funkcija. Ovo su navike koje je lako usvojiti u početku, a teško ih je ispravljati naknadno kad je program već napisan. Zašto funkcija iz zadatka nije najsretnije napisana? Napiši funkciju i uz pomoć polja! Također, poželjno je da imena varijabli (pa čak i komentari) budu pisani na engleskom jeziku, sa standardnim skupom znakova (bez čžšđć itd...).

*Napomena: ukoliko je programski k*ô*d do sada pisao tvoj kolega, ovaj put piši ti.*

**DODATAK 1:**

Ponekad u programiranju imamo potrebu za konstantama. Općenito, postoji nekoliko vrsta konstanti. Zasad je dovoljno napomenuti dva načina pisanja konstanti, na primjeru konstante broja π (*pi*). Pokušaj u programu promijeniti ovako definiranu varijablu pi.

1. #define PI 3.1415926 // češće korišteni oblik
2. const float pi = 3.1415926;

## 2. Operacije nad bitovima (*bitwise*)

Programski jezik C i C++ omogućavaju operacije nad podacima na vrlo niskoj razini, što znači da je moguće vrlo lako pristupiti pojedinom *bitu* neke varijable. Za takve stvari dobro nam dođu još neka znanja:

### 2.1 Striktni tipovi podataka

Kako bi vršili operacije nad bitovima, korisno je znati broj bitova (odnosno bajtova, kako nije moguće zauzeti broj bitova nedjeljiv s brojem bajtova - 8, 16, 32, 64 itd...) koji određen podatak zauzima. Suprotno očekivanju, broj bitova koji sadrže podaci tipa *int*, *char* itd su podložni promjenama u broju bajtova koje zauzimaju, tj. Jako ovise o platformi (PC, Arduino, Raspberry), C/C++ kompilatoru (gcc, cc, clang, intel-c). Kako bi bili sigurni koliko bitova koristimo, moguće je koristiti slijedeće tipove podataka:

uint32\_t a; //32bitni bezpredznačni integer, vrijednosti 0 do (2^32)-1

int8\_t b; //8bitni predznačni integer, vrijednosti -(2^8) do (2^8)-1

int16\_t c; //16 bitni predznačni integer

//itd...

//...

//...

Jos jedan često korišten tip podataka je *byte* (obično veličine 8 bitova, bajt, oktet). U okviru ove vježbe je koristiti *byte*, ali vrijedno je imati na umu da razlike mogu postojati. Podatke je moguće pretvarati iz jednog tipa u drugi operacijom *cast*. Primjer pretvaranja je:

9. ulaz += (char) Serial.read(); // ovo nazivamo “castingom”

**Zadatak 3:** Unesi u program, preko serije, jedan broj s proizvoljno mnogo znamenaka. String pretvori u int metodom *StringUKojemJeBroj.toInt()*, i pohrani ga u varijablu temp1 tipa *int*. U varijablu temp2 tipa int8\_t. Varijable temp1 i temp2 ispiši na seriju. Eksperimentiraj s unešenim vrijednostima. Zašto je potrebno biti pažljiv kod pretvorbe tipova podataka?

### 2.2 bitwise operatori

U nastavku su navedene često korištene binarne operacije. *Binarna* operacije nije isto što i *bitovna*; binarna operacija znači da operacija zahtjeva dva operanda (obično *lijevi* i *desni*, kao što je operator +, \*, /). Postoje i urnarne operacije (npr. ++). Bitwise operatori **ne smiju** se miješati s logičkim operatorima, često onima koje koristimo kao uvijet grananja ili uvijet u *while* petljama, “||”, “&&” itd. Za više informacija o tome kako koji djeluje, googlaj “Arduino bitwise” ili upitaj voditelja radionice.

Operatori:

|  |  |  |
| --- | --- | --- |
| operacija | operator | primjer |
| AND | & | a & b; |
| OR | | (AltGr + W) | a = a | 1; |
| XOR | ^ (AltGr + 3) | b ^= 2; |
| LEFT SHIFT | << | a = a << 1; |
| RIGHT SHIFT | >> | a = a << 7; |

**Zadatak 4:** Napisati funkciju koja koja prima bezpredznačni integer kao argument, i prikaži njegovih nižih 5 bitova pomoću 5 LED dioda. Pokušajte koristiti što veći dio već napisanog, koristeći pozive funkcija.

Za one koji stignu sve: