S T R U K T U R I R A N J E K O D A
Konvencije i preporuke

Da bi se minimizovala mogućnost greške u toku razvoja softvera, povećala čitljivost koda i osigurala mogućnost ponovne upotrebljivosti i proširivosti rešenja, potrebno se pridržavati sledećih konvencija:

1. Strukturiranje koda.

- Potrebno je izvršiti dekompoziciju problema metodom *top-down,* i u skladu sa identifikovanim blokovima nižeg nivoa složenosti podeliti rešenje u više funkcija/modula.
- Unutar *main* funkcije ne treba direktno smeštati kod algoritma.
- Sve funkcije i moduli moraju funkcionisati *stand-alone* (samostalno, izdvojeno). Ne treba koristiti globalne promenljive i konstante, već sve potrebne vrednosti treba proslediti kao argumente funkcija.
- Programski kod duži od 100 linija dobar je kandidat za dalju dekompoziciju i smeštanje unutar više funkcija.

2. Formatiranje koda.

- Tabulacija u dokumentu treba da bude 4 karaktera, i to 4 razmaka umesto TAB karaktera.
- Blokovi koda treba da počinju vitičastom zagradom u novom, posebnom redu, i isto tako da se i završavaju. Kod unutar svakog bloka treba da bude uvučen 4 karaktera u odnosu na blok.

```
{
    \\ kod
}
```

Ovo treba poštovati bez obzira na konstrukt koji se koristi. Primeri:
Definicija funkcije:

```
int myFunction(int a, int b)
{
    \\ kod
}
```

Iteracija:

```
for (i = 0; i < n; i++)
{
    \\ kod
}
```

Selekcija. Čak i u slučaju jedne naredbe u selekciji, treba otvoriti novi blok koda.

```
if (a == b)
{
    \\ kod
}
else
{
    \\ kod
}
```

Selekcija switch-case:

```
switch (val)
{
    case a:
        \\ kod
        break;
    case b:
        \\ kod
        break;

    default:
        \\ kod
}
```

ili:

```
switch (val)
{
    case a:
    {
        \\ kod
        break;
    }
    case b:
    {
        \\ kod
        break;
    }
    default:
    {
        \\ kod
    }
}
```

- Konstante treba pisati velikim slovima. Razmak između reči odvajati karakterom "_". Unutar koda ne treba ubacivati direktne vrednosti, već je u svim slučajevima potrebno definisati konstante. Npr.

```
#define MAX_ELEMENTS 100

...

for (i = 0; i < MAX_ELEMENTS; i++)
```

- Svaka datoteka (c, h) i funkcija mora da počinje *prologom.*

Prolog datoteke:

```
/*****************************************************************************
 *
 *  Univerzitet u Novom Sadu, Fakultet tehnickih nauka
 *  Katedra za Računarsku Tehniku i Računarske Komunikacije
 *
 *  ------------------------------------------------
 *  Ispitni zadatak iz predmeta:
 *
 *  PROGRAMSKA PODRSKA U TELEVIZIJI I OBRADI SLIKE
 *  ------------------------------------------------
 *  Naslov zadatka (npr. DVB Sniffer za EIT/SDT)
 *  ------------------------------------------------
 *
 * \file dvb.c
 * \brief
 * Opis Modula. Npr. ovaj modul realizuje prijem EIT/SDT sekcija.
 * Kreirano on Dec 2010
 *
 * @Author Petar Petrovic
 * \notes
 *
 * \history
 *  11.05.2009. Ispravljena greska pri parsiranju EIT zaglavlja.
 *****************************************************************************/
```

Prolog funkcije.

```
/*****************************************************************************/
*
* @brief    Opis funkcije.
*
* @param    inBuf - [in] Ulazni bafer sa odmercima
*           n - [in] Rezolucija FFT
*           outBuf – [out] Izlazni bafer za FFT koeficijente
*
* @return   NO_ERROR, ako nema greske
*           ERROR, u slucaju greske
*
*****************************************************************************/
```

- Potrebno je realizovati rukovanje greškama. Svaka funkcija koja može da prozvede grešku u toku izvršavanja, treba da vrati kod te greške. Greške se vraćaju kao povratna vrednost funkcije, i to:
  0 – u slučaju da nema greške
  -1 .. –n, u slučaju greške.

  Kodovi grešaka treba da se definišu kao konstante u h datoteci. Npr.

```
#define NO_ERROR 0
#define OUT_OF_RANGE -1
```

  itd.

- Imenovanje promenljivih. Lokalne promenljive treba da budu što kraće dužine, smislenog naziva, pisane malim slovima. U slučaju potrebe za dužim nazivom, koristiti kamilju notaciju (*camel notation).* Npr. **`buf`**, **`buffer`**, **`outputBuffer`**. Za brojače koristiti ***`i`***, ***`j`***, ***`k`***. Globalne promenljive izbegavati. U slučaju da moraju da postoje, imenovati ih tako da prvo slovo bude veliko, i svako slovo u reči veliko. Npr. **`OutputBuffer`**.

- Imenovanje funkcija. Funkcije imenovati korišćenjem kamilje notacije. Npr. **`calculateFourierTransform`**, ili **`calculateFFT`**

- Preglednost. Npr. pravilna deklaracija funkcije je:

```
int calculateFourierTransform(unsigned char *inBuf, int n, int *outBuf)
```

  bez razmaka iza naziva funkcije, i sa razmakom nakon zareza u listi parametara. Nakon *if* selekcije napraviti jedan razmak, npr.

```
if (a == b)
```

  a ne:

```
if(a == b)
```

  niti:

```
if ( a == b )
```

  Ukoliko izraz prelazi stranicu koda (pa je potrebno horizontalno skrolovati tekst) potrebno je prelomiti izraz u novi red, kao:

```
int myFunction(int param1,
               int param2,
               int param3)
```

odnosno:

```
if (((a == b)&&(c == d))||
    ((m==n)&&(p == q)))
```

- Ne koristiti „egzotične“ izraze u C-u, kao npr.

```
for (int i = 0; i < MAX_LENGTH; i++, ++k, calculateFFT(buf1, 5, buf2))
```

ili

```
if ((a = b + c) == 0)
```

Kod je potrebno da bude razumljiv uz što manje napora, i onome ko ga nije pisao.

- Komentari unutar koda. Koristiti komentare tipa:

```
// kod
/* tekst komentara */
// kod
```

Komentari u istoj liniji dozvoljeni su samo kod deklarisanja promenljivih, npr.

```
unsigned char *buf;    /* ulazni bafer za odmerke */
```

Komentarisati sve što nije na prvi pogled jasno. Komentar treba da bude smislen. Ne upisivati u tekst komentara ono što se vidi iz same naredbe, kao:

```
a = c + d;  /* a dobija vrednost zbira c i d - NIKAKO*/
```