LabVIEW

Vežba 2

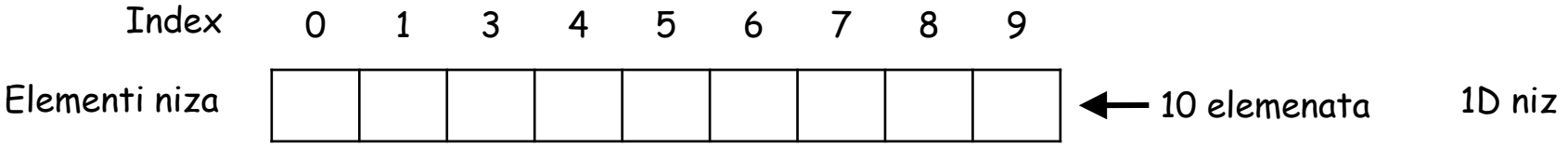
Nizovi, petlje i strukture

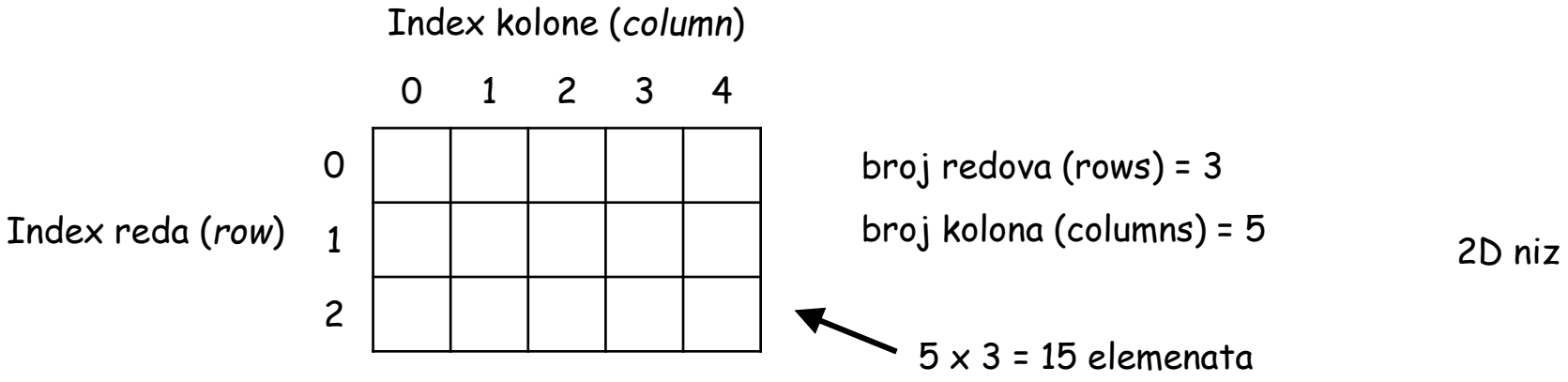
Nizovi (Arrays)

Kolekcija podataka istog tipa

- Svaki niz karakterišu njegovi elementi i dimenzija
- Niz može da ima jednu ili više dimenzija, i da sadrži do 2^{31} - 1 elemenata, u zavisnosti od memorije
- Moguće je napraviti niz numeric, boolean, path, string, waveform i cluster tipa podataka
- Elementima se pristupa na osnovu indeksa, prvi index je uvek 0

Primer:

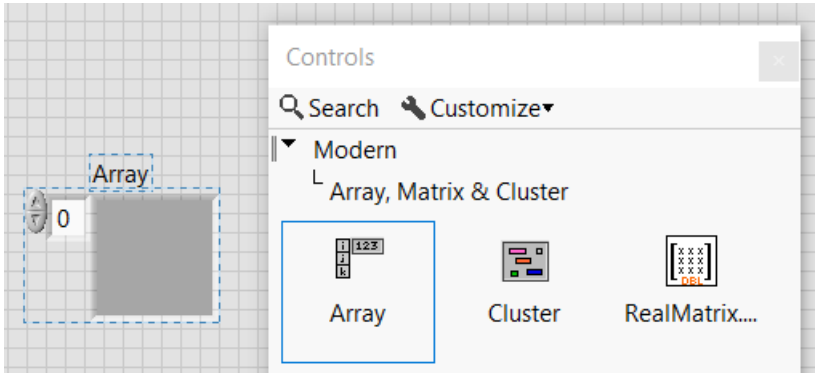




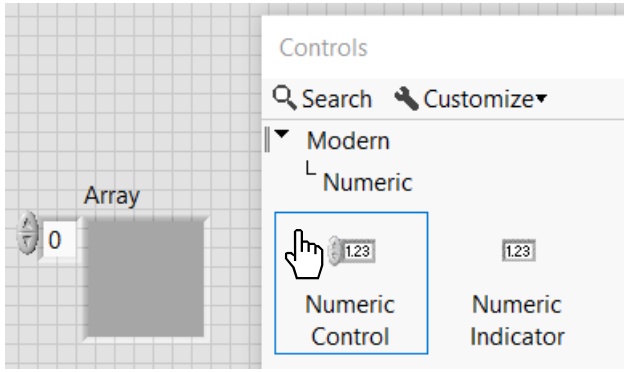
Arrays (Nizovi)

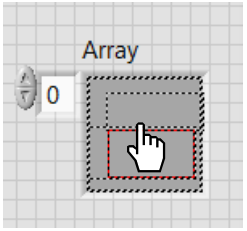
Kreiranje novog niza u vidu kontrola/indikatora na Front Panel-u:

1. ***Controls>>Array, Matrix & Clusters>>Array***

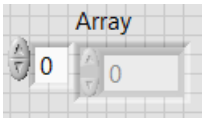


2. Prevući željenu kontrolu/indikator u objekat Array

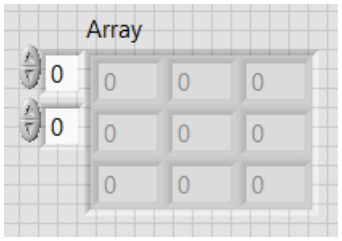




3. Dobija se 1D Array.
Za promenu na kontrolu/indikator:
Desni klik na objekat>>
Change to Control
ili
Change to Indicator



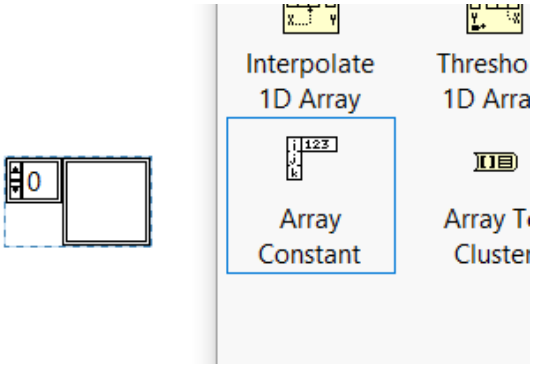
4. Po potrebi, dimenzije nizu se dodaju:
Desni klik na objekat>>
Add Dimension



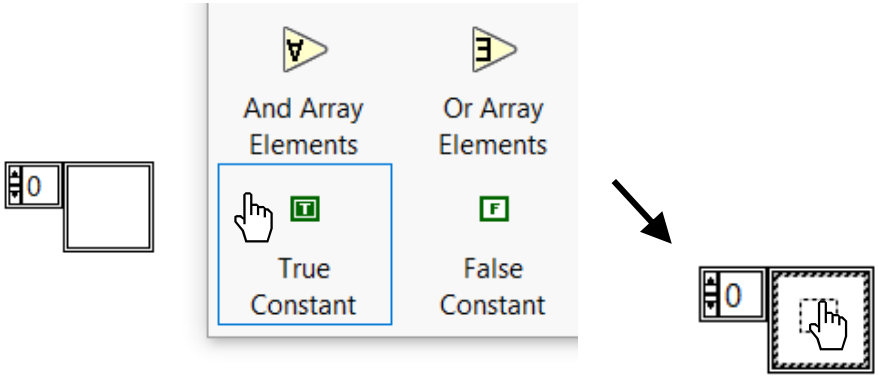
Arrays (Nizovi)

Kreiranje niza konstanti na Block Diagram-u:

1. ***Controls>>Array>>Array Constant***

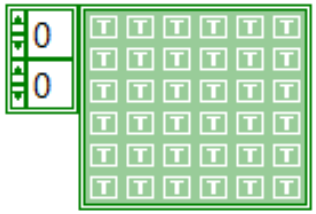


2. Prevući željenu konstantu



3. Dobija se 1D Array.
Za promenu na kontrolu/indikator:
Desni klik na objekat>>
Change to Control
ili
Change to Indicator

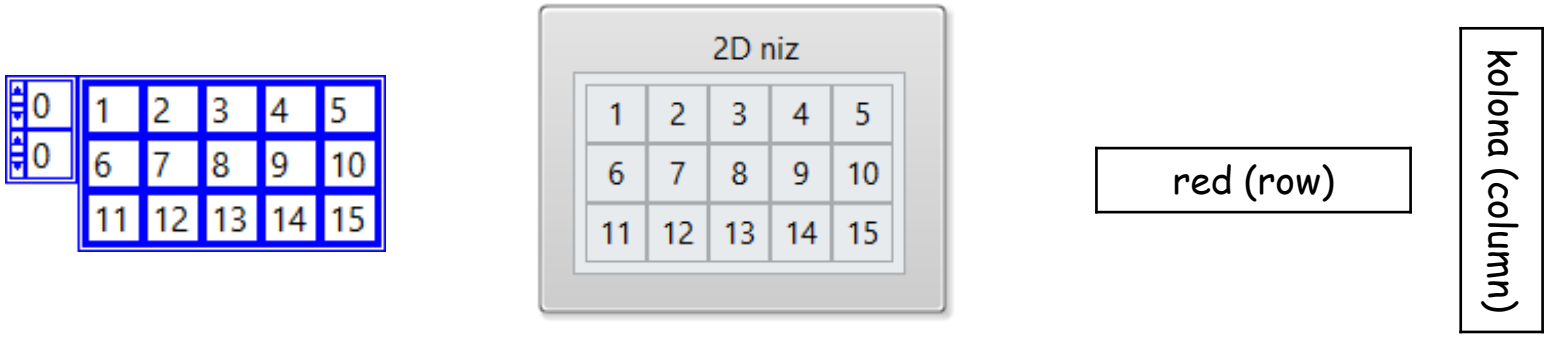
4. Po potrebi, dimenzije nizu se dodaju:
Desni klik na objekat>>
Add Dimension



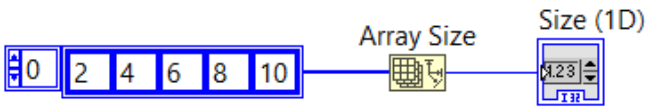
Prikaz nizova

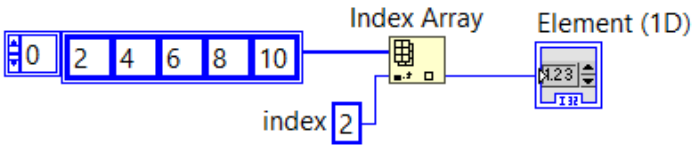
1D nizovi i na Front Panel-u i na Block Diagram-u mogu biti prikazani i u horiznotalnom i u vertikalnom položaju, i nema nikakve razlike između ova dva prikaza

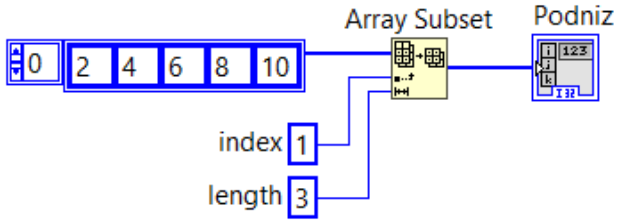
Kod 2D nizova eksplicitno je određeno da uvek pri prikazu, redovi (rows) moraju biti u horizontalnom položaju, a kolone (columns) u vetikalnom

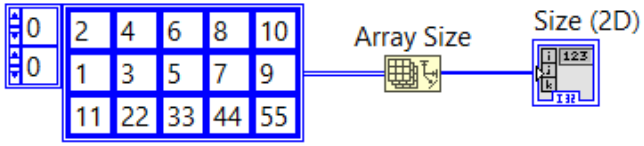


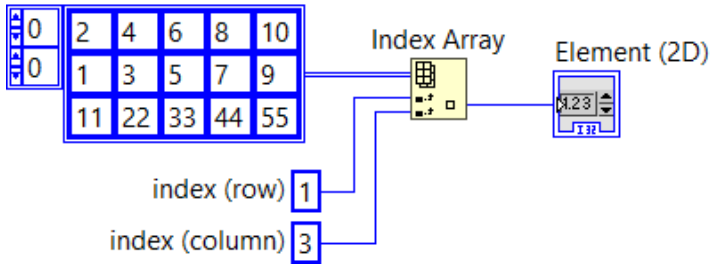
Osnovne funkcije za rad sa nizovima:



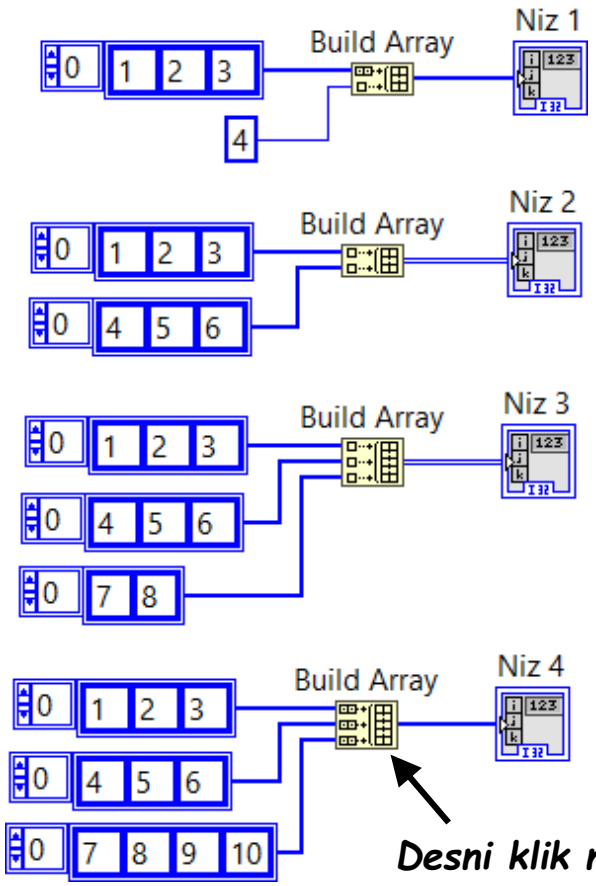


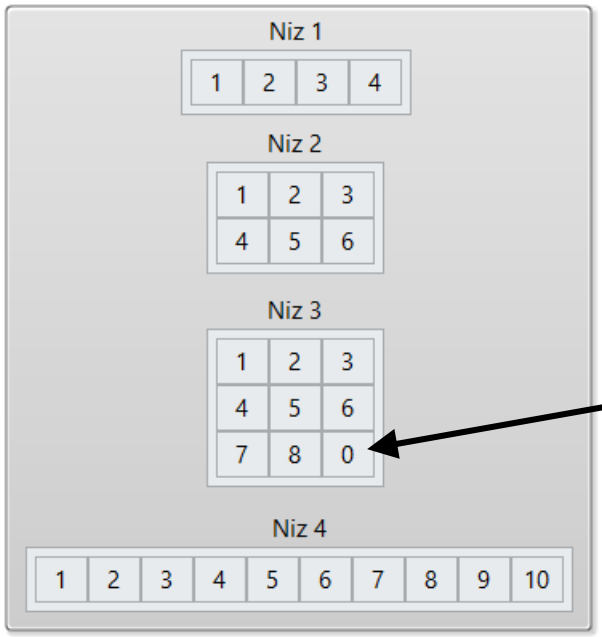






Kreiranje nizova pomoću *Build Array* funkcije





Automatski dodeljena vrednost 0 (nula)

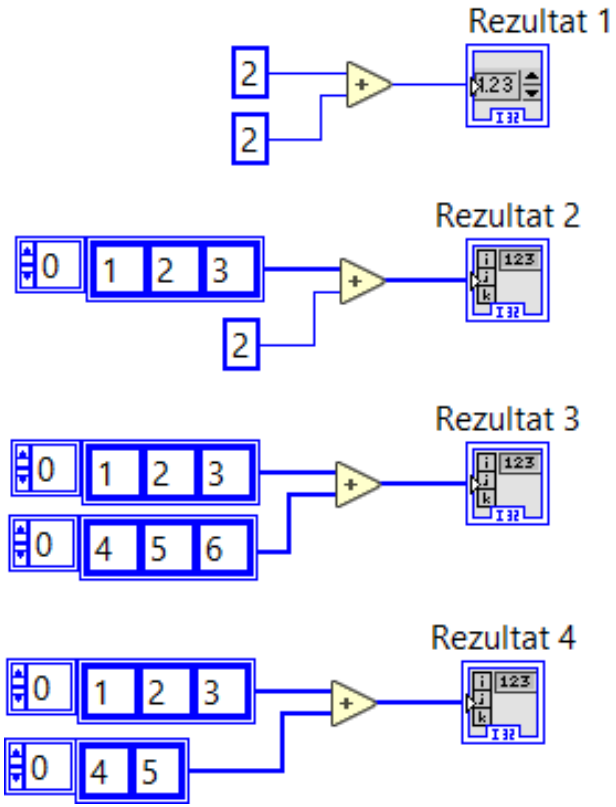
Desni klik na Build Array>>Concatenate Inputs

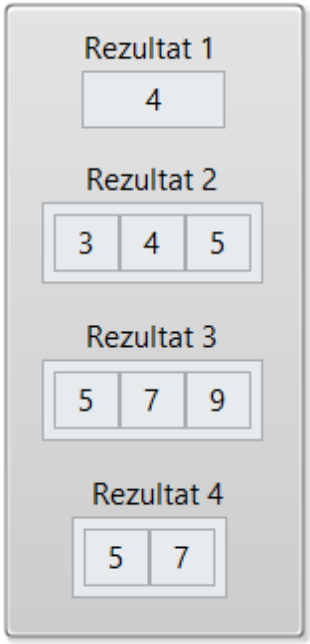
Kako bi se svi ulazni 1D nizovi smestili u jedan red, potrebno je uključiti opciju *Concatenate Inputs*, desnim klikom na *Build Array*

concatenate = objediniti

Polimorfizam aritmetičkih LabVIEW funkcija

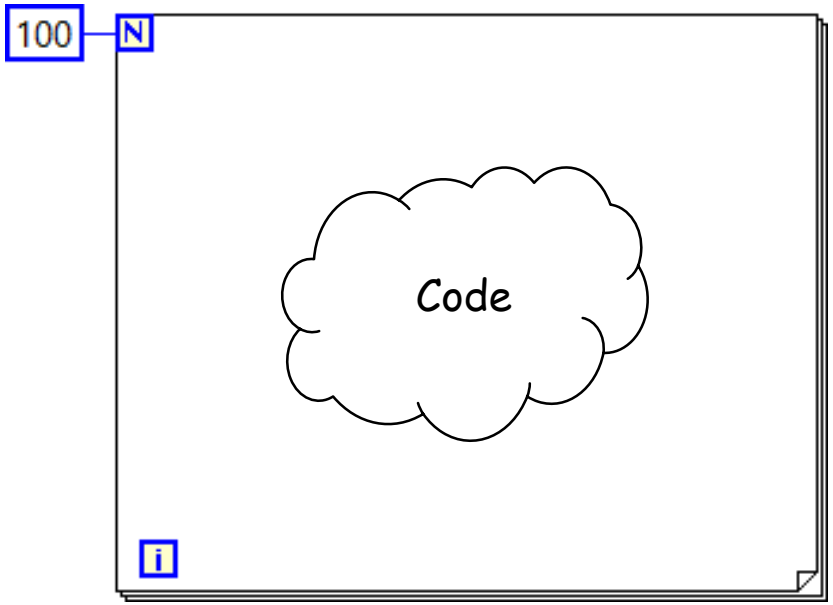
- U objektno-orijentisanom programiranju *polimorfizam* (*"višeobličnost"*) je važna osobina jer omogućuje da osnovna klasa (roditelj) definiše funkcije (metode) koje će biti zajedničke za sve izvedene klase (potomke), ali da izvedenim klasama ostavi slobodu da same implementiraju sve te funkcije.
- Ulazi LabVIEW funkcija mogu biti različitih tipova





For Loop (For petlja)

Petlja se izvršava zadati broj puta



LabVIEW

```
N = 100;
for (i = 0 ; i < N ; i++)
{
        Code;
  }
```

C jezik

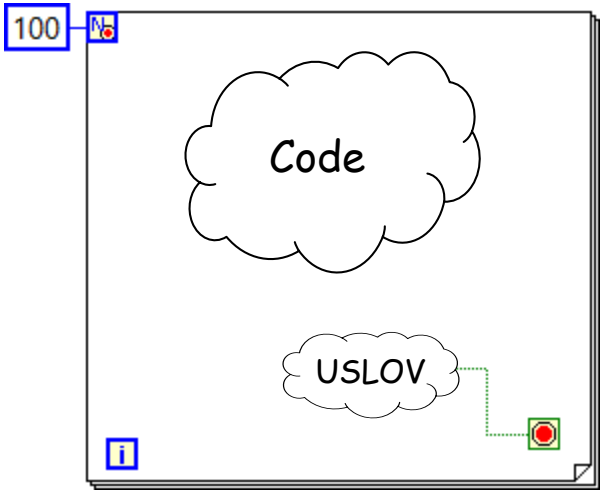
N - zadat broj iteracija

i - redni broj trenutne iteracije (kreće od 0)

For Loop (For petlja)

- Dodavanjem uslovnog terminala (*Conditional Terminal*) može se prekinuti izvršavanje *for* petlje u trenutku kada se ispuni neki USLOV, i pre nego što se izvrši zadati broj iteracija.

Desni klik na okvir petlje>>Conditional Terminal



LabVIEW

```
N = 100;
for (i = 0 ; i < N ; i++)
{
        Code;
        if (USLOV) {
              break;
        }
  }
```

C jezik

 - zadat broj iteracija

 - redni broj trenutne iteracije (kreće od 0)

Uslovni terminal (analogno ***break*** u *C* jeziku), može se pojaviti u dva oblika:

 - **Stop if True** (petlja se izvršava dokle god terminal ne primi **TRUE** vrednost)

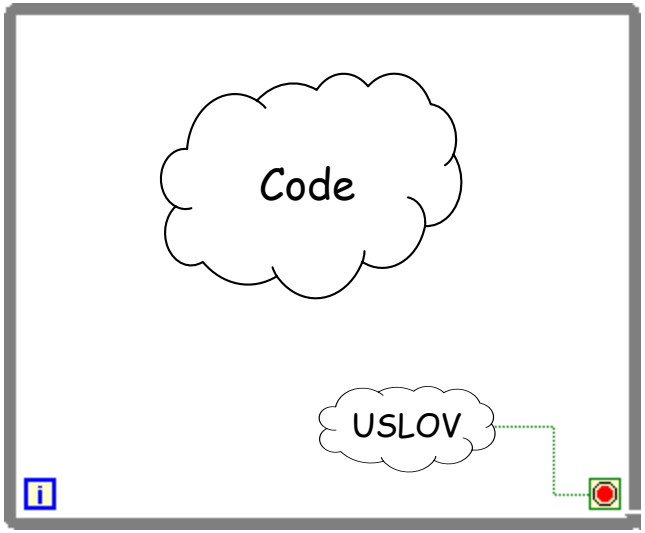
 - **Continue if True** (petlja se izvršava dokle god terminal ne primi **FALSE** vrednost)

Desnim klikom na uslovni terminal bira se željeni oblik.

While Loop (While petlja)

Petlja se izvršava dokle god se ne ispuni određeni uslov

While petlja bez obzira na ispunjenost uslova, izvršava se uvek bar jednom (analogno ***do (while)*** u *C* jeziku)



LabVIEW

```
do  {
        Code ;
} while ( !USLOV ) ;
```

C jezik

- redni broj trenutne iteracije (kreće od 0)

- **Stop if True** (petlja se izvršava dokle god terminal ne primi **TRUE** vrednost)

- **Continue if True** (petlja se izvršava dokle god terminal ne primi **FALSE** vrednost)

Paralelno izvršavanje

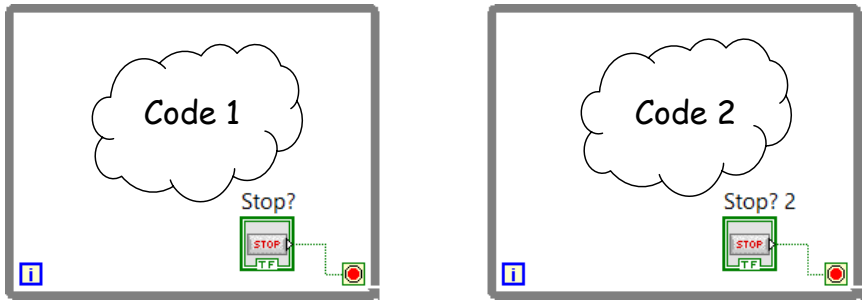
U zavisnosti od broja jezgara procesora LabVIEW automatski vrši *multithreading* i *multiprocessing*

- ***Multithreading*** – kada jedan proces ima više thread-ova (niti), koji se izvršavaju istovremeno. Thread-ovi (niti) u računarstvu predstavljaju zadatake koji se istovremeno izvršavaju u okviru jednog procesa. Instrukcije koje sačinjavaju jednu nit se izvršavaju u isto vreme sa instrukcijama koje sačinjavaju druge niti (ako u računaru postoji više procesora) ili je ta istovremenost samo prividna (ako u računaru postoji samo jedan procesor).
- ***Multiprocessing*** - kada postoje dve ili više CPU jedinice u jednom računaru (Dual Core, Quad Core...) oni dele resurse, mogu da rade paralelno i na taj način ubrzavaju izvršavanje aplikacije

- U tekstualnom programiranju, kako bi se napravila multithreading aplikacija, potrebno je kreirati više thread-ova (niti) i kod za njihovu međusobno komunikaciju.
- LabVIEW mehanizam izvršavanja je napravljen za multiprocessing. LabVIEW automatski prepoznaje mogućnosti i potrebe za multithreading-om, i mehanizam izvršavanja automatski upravlja multithreading komunikacijama.

Data-flow model programiranja je po svojoj prirodi paralelan, i svaki nezavisan *data-flow* deo koda se može paralelno izvršavati sa drugim *data-flow* delom koda

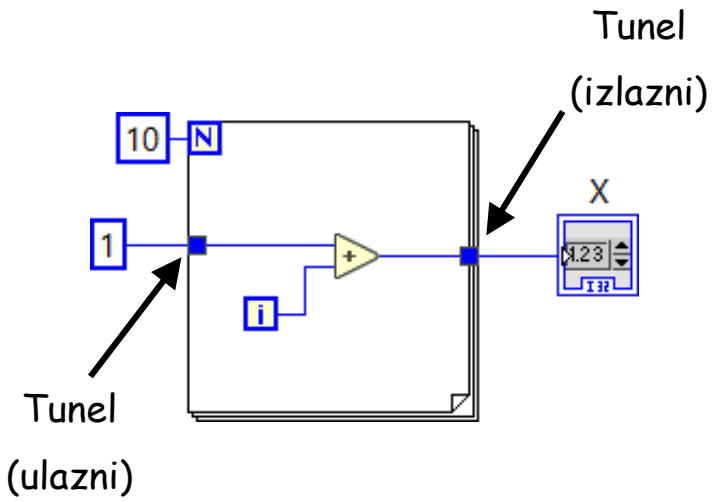
Kao primer, dve nezavisne *while* petlje:

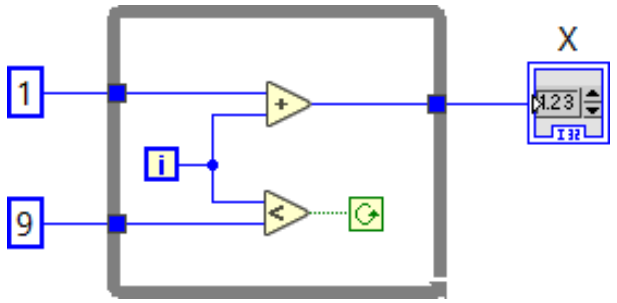


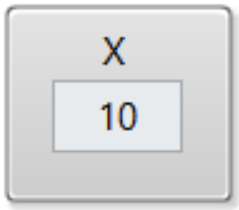
Tuneli u strukturama

Uvoze podatke u strukturu ili izvoze podatke iz strukture

- Tunel je kvadrat koji se pojavljuje na ivici strukture, a boja tunela odgovara boji tipa podatka koji prolazi kroz tunel
- Kada tunel prosleđuje podatke strukturi (ulazni tunel), struktura sa kreće sa izvršavanjem tek kada podatak stigne do tunela
- Kada se podaci prosleđuju iz strukture tunelom (izlazni tunel), podatak je dostupan tek kada se završi izvršavanje strukture





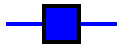


Tuneli u strukturama

Konfigurisanje tunela se vrši desnim klikom na njih

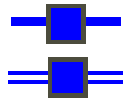
Ulazni tuneli

Ako je ulazna veličina skalar, ne postoji nikakvo dodatno konfigurisanje tunela. U ovom slučaju, jedino što tunel može je da ovu veličinu prosledi u strukturu.

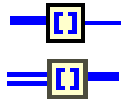


Kada je ulazna veličina niz, tada desnim klikom na tunel može da se izabere:

- **Disable Indexing** – tunel prosleđuje ceo niz ka unutrašnjosti strukture



- **Enable Indexing** – svakom novom iteracijom strukture, prosleđuje se element po element niza, počevši uvek od nultog



Tuneli u strukturama

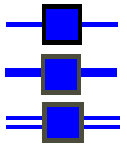
Konfigurisanje tunela se vrši desnim klikom na njih

Izlazni tuneli

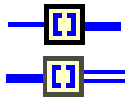
Kod izlaznih tunela, konfigurisanje je jedinstveno, bez obzira da li je vrednost koja se prosleđuje tunelu skalar, niz, i sl.

Konfigurisanje izlaznog tunela : ***Desni klik na izlazni tunel>>Tunnel Mode*** i tu se dobija:

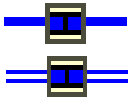
- **Last Value** – tunel prosleđuje iz strukture vrednost koja je bila u poslednjoj iteraciji te strukture



- **Indexing** – na izlazu tunela formira se niz elemenata, gde svaki element odgovara *i*-toj iteraciji

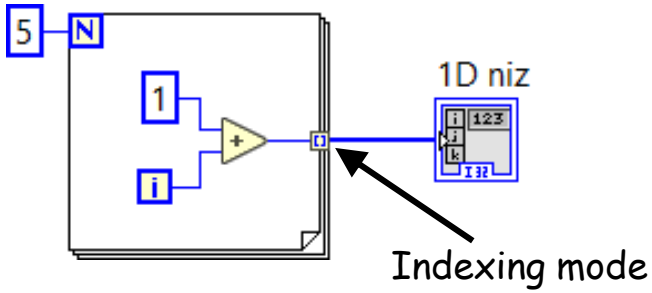


- **Concatenating** – na izlazu tunela se formira niz iste dimenzije kao niz na ulazu tunela, tako što svaka naredna iteracija dodaje trenutni niz na kraj prethodnog (analogno sa funkcijom Build Array kada je uključena opcija Concatenate Inputs)

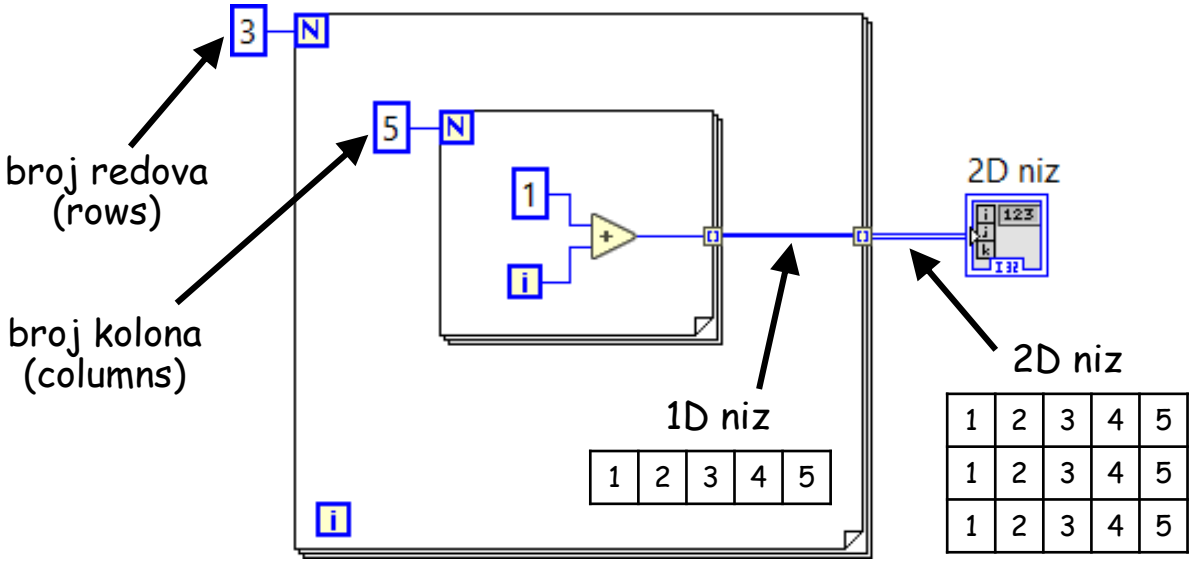


- **Conditional** – ako je ispunjen određeni uslov, izlazni tunel prosleđuje vrednost. U suprotnom ne prosleđuje ništa

Formiranje nizova pomoću petlji



1D niz				
1	2	3	4	5

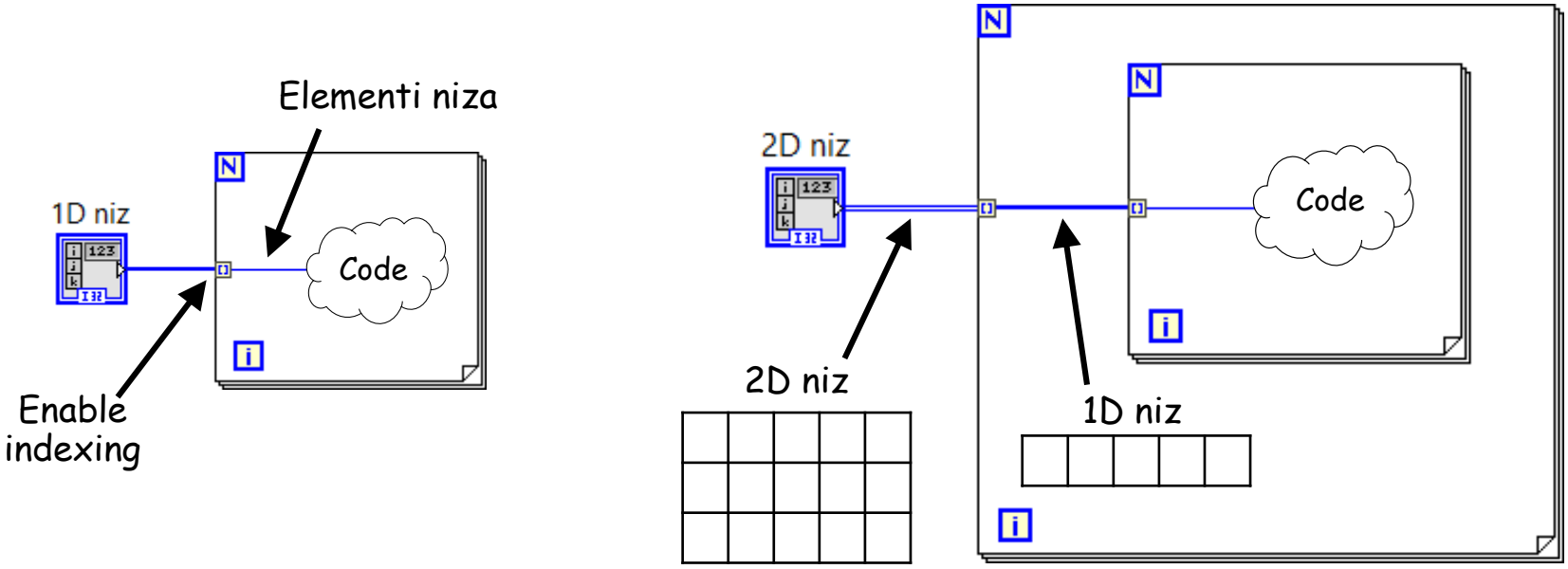


2D niz				
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

Eksplicitno pravilo za formiranje 2D niza:

- Unutrašnja petlja formira REDOVE
- Spoljašnja petlja ih spaja jedan ispod drugog (spaja ih u KOLONE)

Čitanje nizova pomoću petlji



Ako je na ulaznom tunelu odabrano *Enable Indexing*, tada se petlja izvršava onoliko puta koliko ulazni niz ima elemenata, i zbog toga nema potrebe povezivati N terminal.

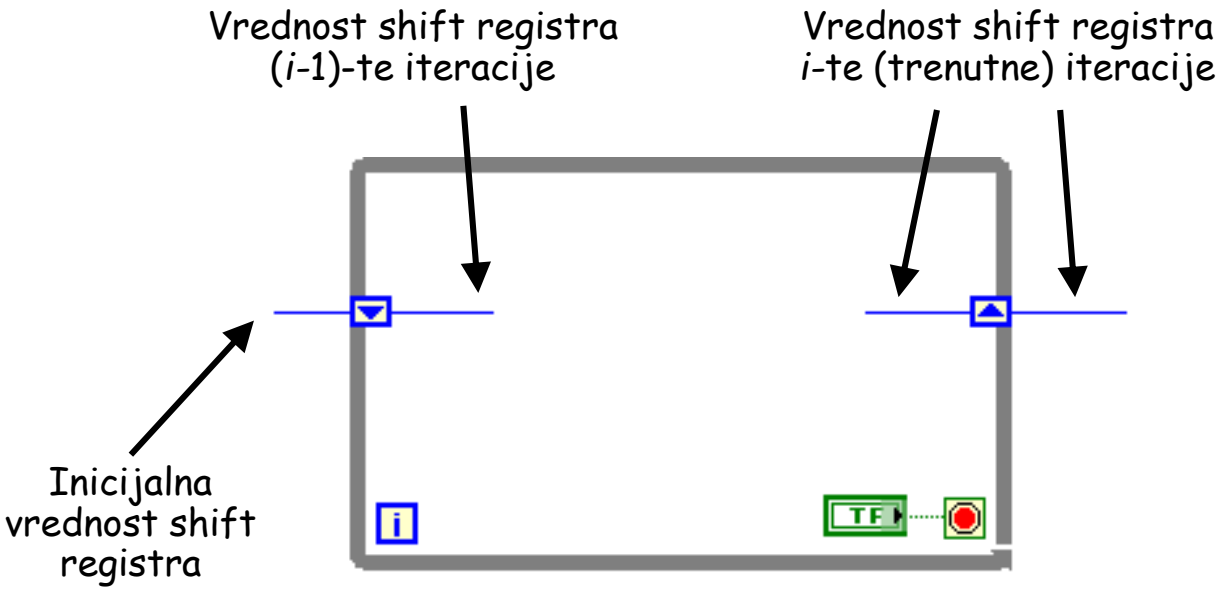
Eksplicitno pravilo za čitanje 2D niza:

- Spoljašnja petlja očitava red po red ulaznog 2D niza
- Unutrašnja petlja očitava element po element ulaznog 1D niza

Shift Registers

Registri koji služe za pristup podacima iz prethodnih iteracije petlje

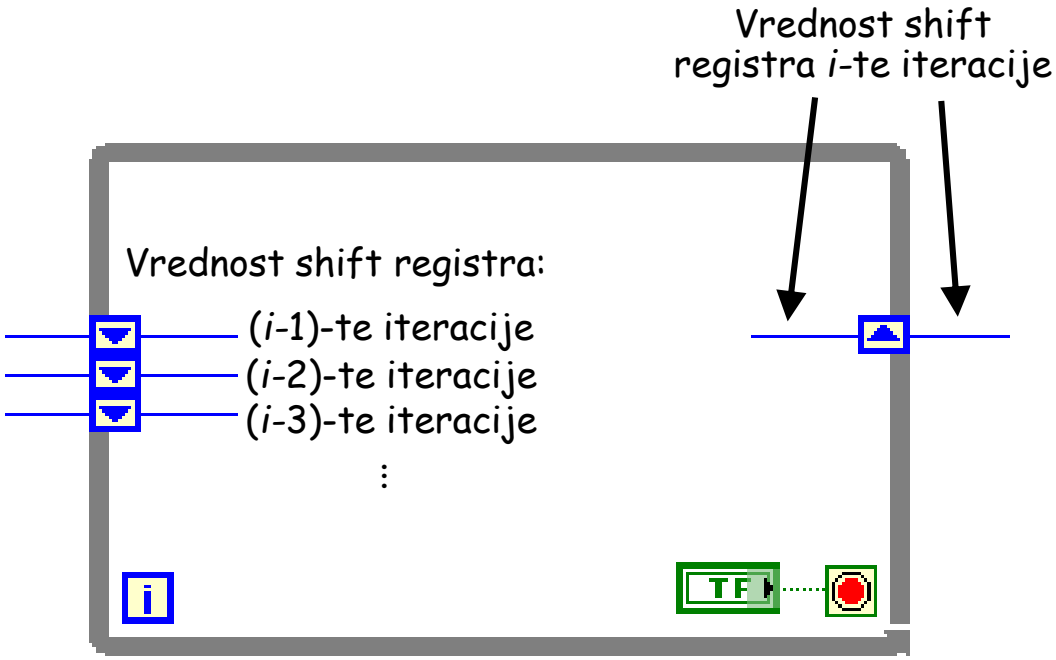
Desni klik na okvir petlje>>Add Shift Register



Inicijalna vrednost shift registra – početna vrednost shift registra koja mu se dodeljuje, ako za tim ima potrebe

Shift Register

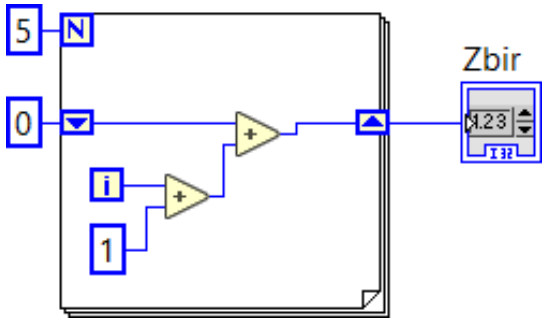
Pristup podacima iz prethodnih *n* iteracija



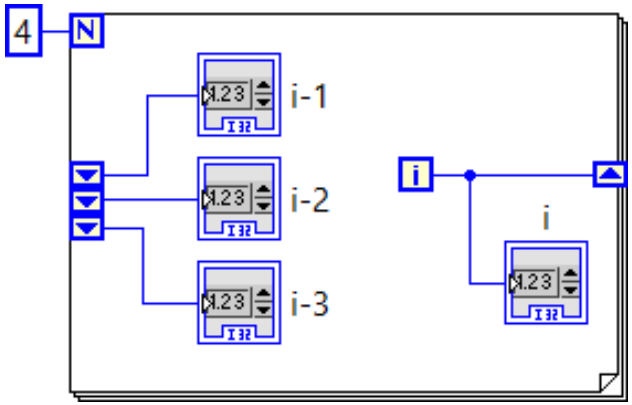
Razvlačenjem levog shift registra na dole (ili *desni klik na levi shift registar>>Add Element*) dobija se mogućnost pristupu podacima iz prethodnih *n* interacija

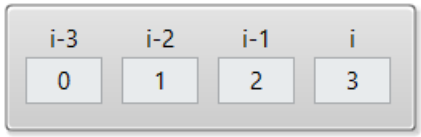
Primer 1

Naći zbir prvih pet prirodnih brojeva, tj. {1, 2, 3, 4, 5} , koristeći for petlju i shift registre



Kreirati for petlju sa 4 iteracije i prikazati redni broj svake iteracije pomoću 4 indikatora, koristeći shift registre

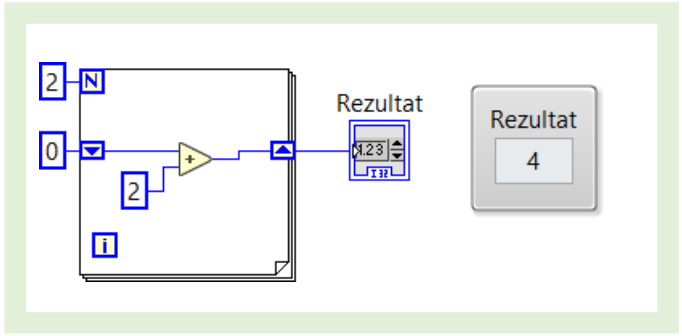




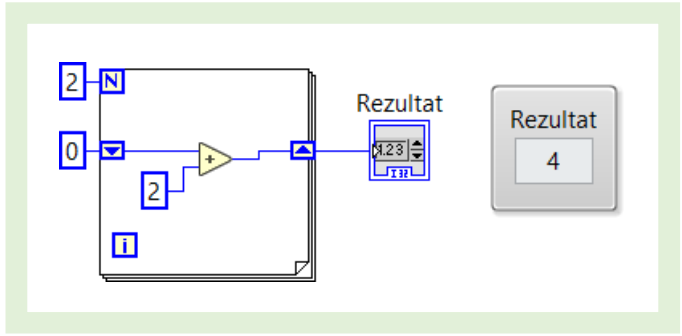
Inicijalizacija *Shift Register*-a

Prvo pokretanje VI-a → VI izvršio kod → Drugo pokretanje VI-a

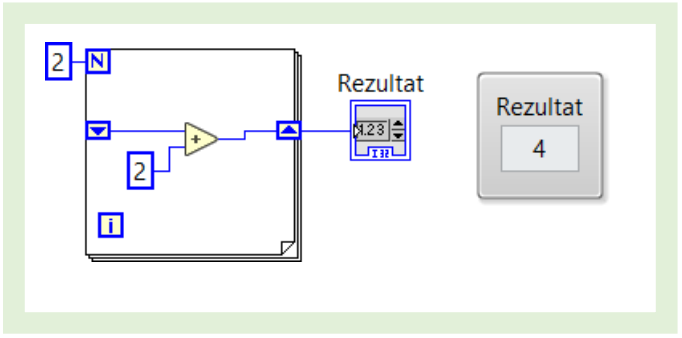
- Inicijalizovan shift registar



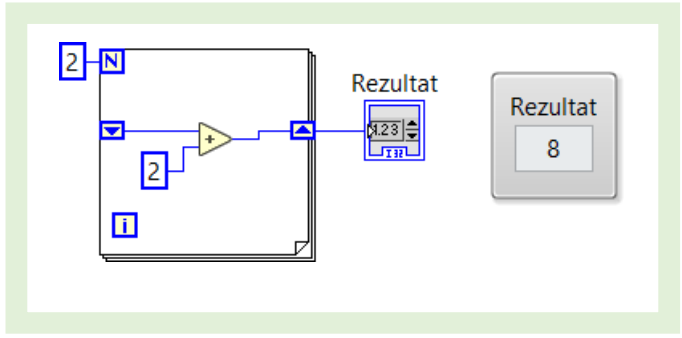
VI izvršio kod



- Neinicijalizovan shift registar



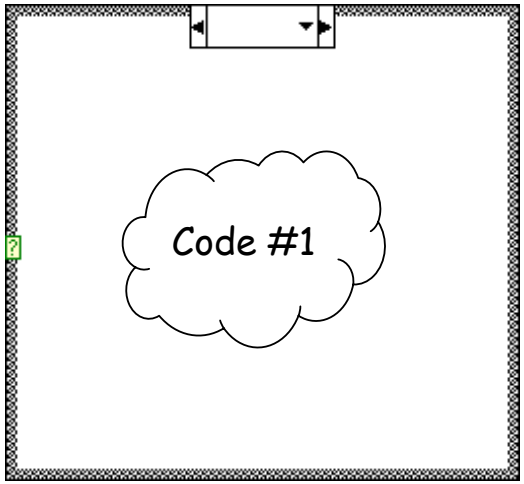
VI izvršio kod



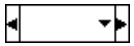
Case Structure

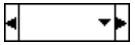
Analogno ***switch-case*** strukturi u C jeziku

Služi za odlučivanje - u zavisnosti od ulaznog parametra odlučuje se koji će deo koda da se izvrši



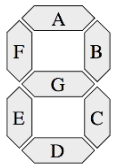
 - terminal na koji se dovodi ulazni parametar (boja zavisi od tipa podatka)

 - skup svih definisanih stanja (slučajeva) ulaznog parametra

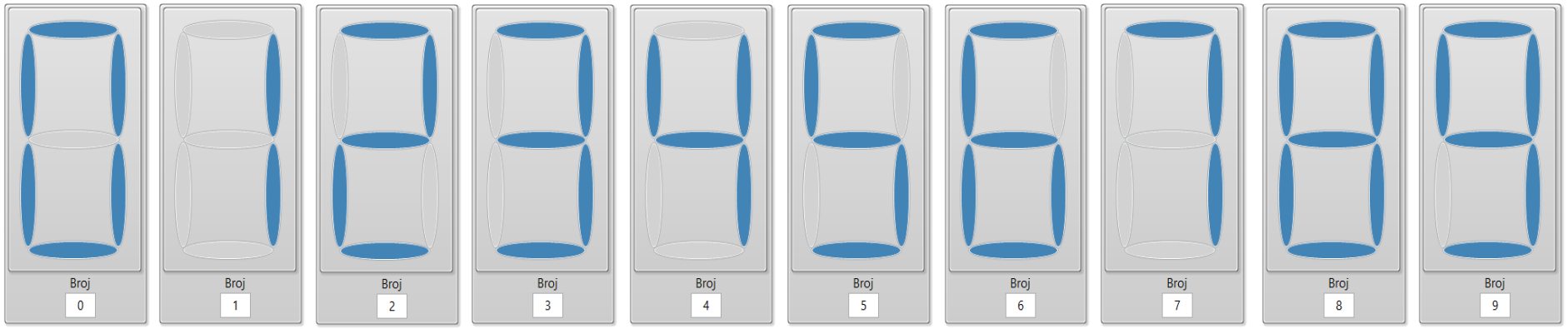
Dodavanje novog slučaja (case-a): ***Desni klik na***  ***>> Add Case After / Add Case Before***

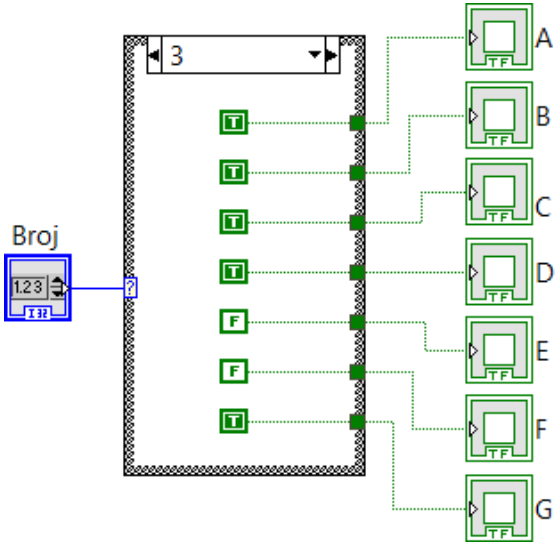
Postoji mogućnost definisanja Default Case-a, koji se pokreće ako ulazna vrednost ne odgovara ni jednom kriterijumu

Primer 2

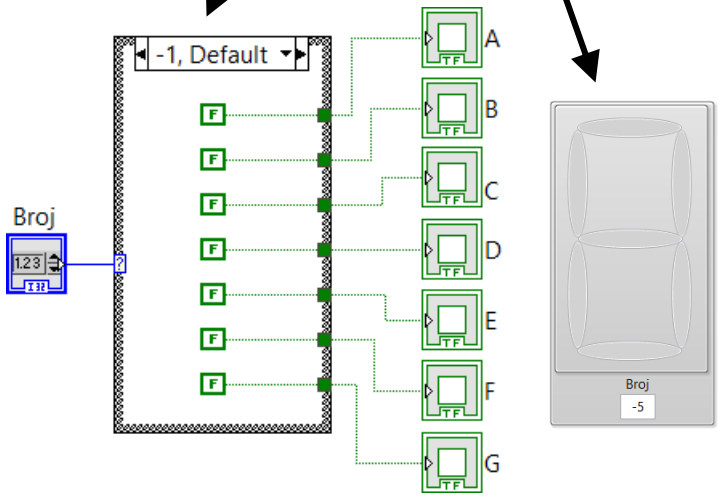


Na osnovu unetog integera *Broj*, napraviti prikaz brojeva od 0 do 9 u obliku 7-segmentnog displeja, koristeći boolean LED indikatore i case strukturu





Default Case (u slučaju da se unese broj van opsega od 0 do 9)



TRUE konstanta

FALSE konstanta

Flat Sequence

Definiše redosled izvršavanja koda

Flat Sequence-a je podeljena na frame-ove:

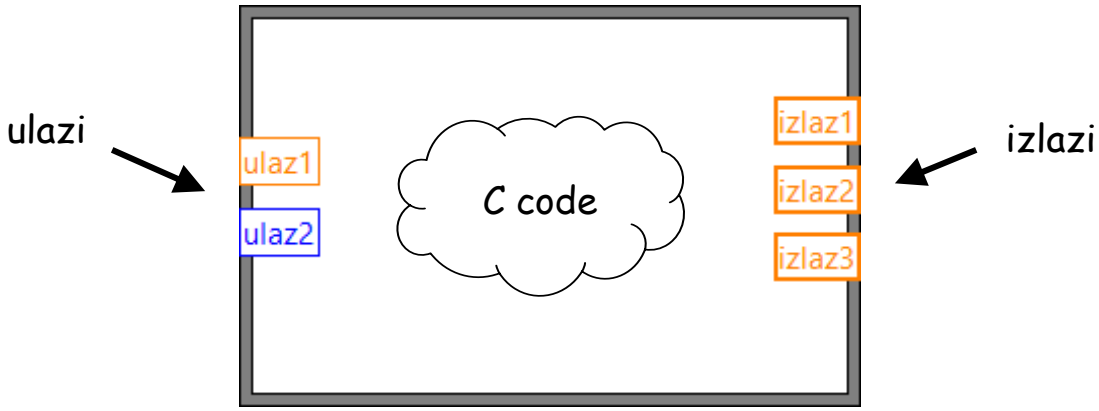
Dodavanje novog frame-a: ***Desni klik na okvir strukture>>Add Frame Before/After*** ili ***Insert Frame***

frame = ram /okvir

Formula Node

Rad sa numeričkim podacima u formi C jezika

Korisna primena kod glomaznih matematičkih izraza

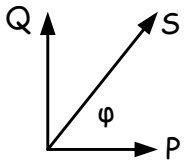


- Broj ulaza/izlaza kao i njihovi nazivi su proizvoljni
- Boja ulaza/izlaza zavisi od tipa podatka
- Ulazni numerički podatak može biti i skalar i niz

Dodavanje novog ulaza/izlaza: ***Desni klik na okvir strukture>>Add Input / Output***

Primer 3

Na osnovu zadate aktivne (P) i reaktivne (Q) električne snage, odrediti prividnu (S) snagu, faktor snage (PF) i ugao φ izražen u stepenima. Zadatak rešiti korišćenjem formula node strukture.

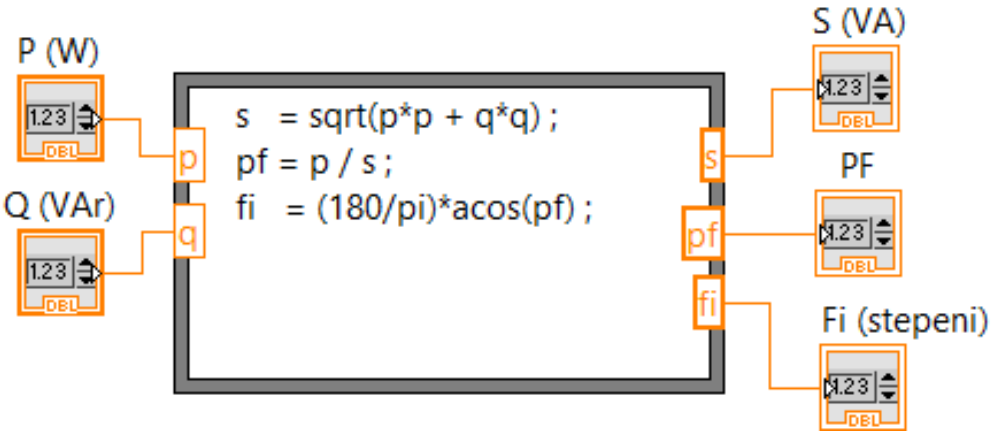


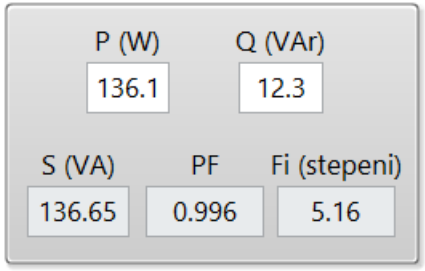
$S = P + jQ$

$S^2 = P^2 + Q^2$

$PF = \cos\varphi = P / S$

$1\ rad = (180/\pi)\ °$





pi – reč rezervisana za konstantu π unutar formula node strukture

*funkcija *acos(x)* po default-u vraća vrednost u radijanima

Zadatak 2

Unutar postojećeg projekta iz *Zadatka 1* dodati novi virtuelni instrument (VI) pod nazivom *Zadatak 2*. Unutar novog VI-a realizovati sledeće:

- Pomoću *for* petlje kreirati 1D niz od 10 elemenata, čije su vrednosti prvih 10 pozitivnih neparnih brojeva. Zatim kreirati 2D niz, pomoću *Build Array* funkcije, tako da ima tri reda. Prvi red neka su elementi kreiranog 1D niza, drugi red neka su vrednosti 5 puta veće od vrednosti elemenata 1D niza, i neka svi elementi trećeg reda imaju vrednost 9,81 . 2D niz prikazati na front panel-u.

- Koristeći *for* petlju i shift registre, naći faktorijel broja unetog u numeric (I32) kontrolu pod nazivom ***Broj n***. Rezultat prikazati u numeric indikatoru pod nazivom ***Faktorijel***.

$$n! = \prod_{i=1}^{n} i$$

- Koristeći *case* strukturu, proveriti da li je ***n!*** × **(slučajan broj između 0 i 1)** veće od 500. Ako jeste, na front panel-u ispisati: „Rezultat je veći od 500". Ako nije, ispisati: „Rezultat nije veći od 500, i iznosi: ?", gde umesto ? treba da se ispiše broj koji se dobio.

- Na osnovu unetog električnog napona (numeric kontrola ***Napon***) i struje (numeric kontrola ***Struja***), izračunati električnu otpornost pomoću Omovog zakona, koristeći *Formula Node* strukturu za računanje. Rezultat zaokružen na dva decimalna mesta prikazati na front panel-u. (R=U/I)

- Koristeći 2 *while* petlje i *flat sequence-u* simulirati bacanje dve kockice za igre na sreću. Treba da postoje dva numeric indikatora (I8), sa nazivima ***Kockica 1*** i ***Kockica 2***. Takođe treba da postoje i dva dugmeta: ***Baci kockicu 1*** i ***Baci kockicu 2***. Kockice se bacaju jedna po jedna, tako da se prvo baca kockica broj 1, pa 2. Kada se pritisne dugme ***Baci kockicu 1,*** na indikatoru ***Kockica 1*** treba da se ispiše broj na kojem je stala kockica 1. Kada se pritisne dugme ***Baci kockicu 2,*** na indikatoru ***Kockica 2*** treba da se ispiše broj na kojem je stala kockica 2. Nakon bacanja druge kockice, program završava sa izvršavanjem.

generisanje slučajnog broja može da se vrši funkcijom ***Random Number (0-1)