Základy práce v systému LabVIEW

Ing. Roman Matějka,

Základní operátory

V dnešním díle našeho seriálu se zaměříme na obecné matematické operátory, které můžeme použít v rámci VI a systému LabVIEW. Než však začneme, zopakujme si základní datové typy, které můžeme v rámci systému LabVIEW používat.

Obrázek 1 - Menu representation s možnými datovými typy pro operátory.

Datové typy dělíme tedy na: celočíselné kladné, označované jako U (číslo udává jejich rozsah v bitech, viz předchozí díly seriálu); celočíselné (kladné i záporné), označované jako I; pak máme typy s plovoucí desetinou čárkou, SGL, DBP, EXT a jejich komplexní varianty CSG, CDB, CXT; poslední je typ s pevnou desetinou čárkou FXP.

Základní operátory naleznete v paletě **Programming → Numeric**. Tato paleta obsahuje jednak základní operátory (+, -, *, /), základní konstanty, zaokrouhlování, celočíselné dělení se zbytkem, sumu apod.

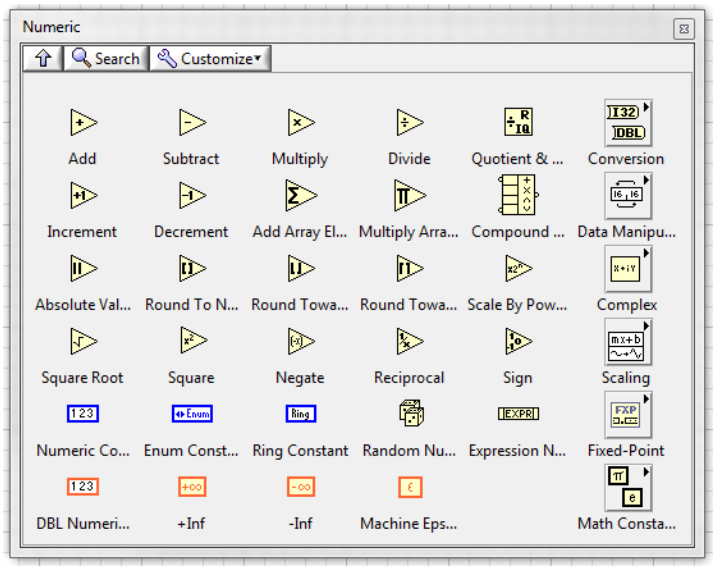
Biomedicínské inženýrství pro znalostní ekonomiku, CZ.02.2.69/0.0/0.0/16_018/0002242
Modernizace laboratoří pro biomedicínské inženýrství, CZ.02.1.01/0.0/0.0/16_017/0002244

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

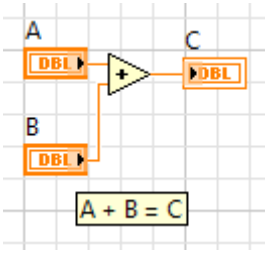
Základy práce v systému LabVIEW

Ing. Roman Matějka,



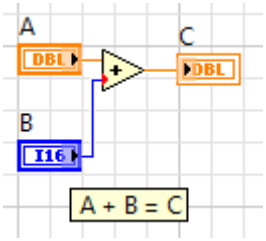
Obrázek 2 - Paleta numeric s výčtem funkcí

Uveďme například, jednoduchou operaci sčítání. V případě stejných datových typů na náš nečeká žádné úskalí, a můžeme tyto prvky bez problému spojovat.



Obrázek 3 - Sčítání dvou prvků, využívající blok Add

Problém by však mohl nastat v případě použití více datových typů. Existují dvě možnosti, jak toto řešit. První je nechat možnost na systému LabVIEW. V tomto případě se objeví u automaticky přetypované veličiny červené kolečko.



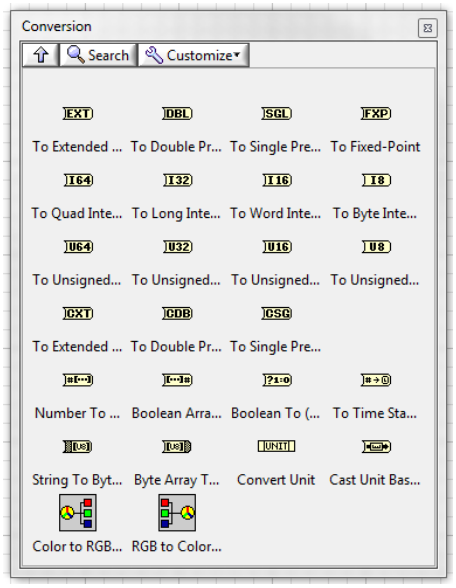
Obrázek 4 - Sčítání dvou odlišných datových typů, DBL a INT

Biomedicínské inženýrství pro znalostní ekonomiku, CZ.02.2.69/0.0/0.0/16_018/0002242
Modernizace laboratoří pro biomedicínské inženýrství, CZ.02.1.01/0.0/0.0/16_017/0002244

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

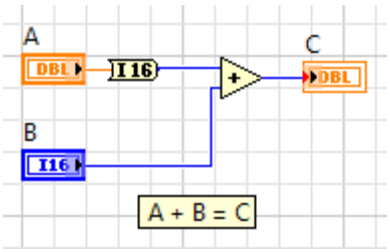
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Všimněte si, že výsledek je opět ve formě DBL, takže LabVIEW zvolilo přetypování typu I16 na DBL. V tomto případě může daná věc fungovat, ovšem mohou se vyskytnout problémy se zaokrouhlením aj. V tomto případě přichází na řadu ruční konverze. Podskupina **Conversion**



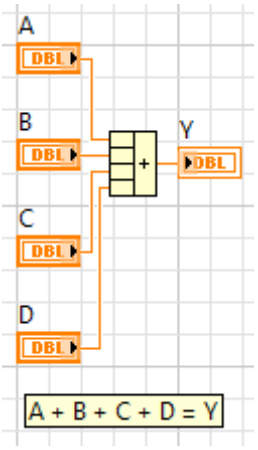
Obrázek 5 - Paleta nástrojů conversion.

V této paletě naleznete všechny potřebné prvky pro konverzi datových typů mezi sebou a můžete tak programově donutit danou konverzi dle požadavků.



Obrázek 6 - Sčítání s přetypovaným vstupem A z DBL na I16, výsledek je celočíselný, automatické přetypování do DBL.

Na závěr si zmiňme ještě jednu zajímavou funkci, **Compound Arithmetic,** tato funkce umožňuje sestavit více vstupy operátor pro základní matematické operace, což v případě většího množství takových dat, výrazně zpřehledňuje kód.



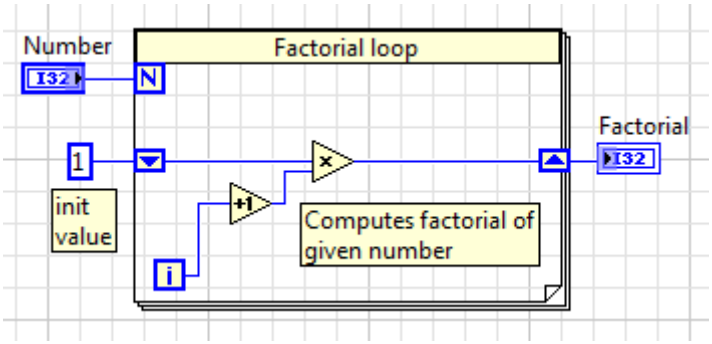
Obrázek 7 - Compound Arithmetic pro realizaci matematické operace s více vstupy.

Smyčky

Aby mohla aplikace fungovat, a uživatel ji mohl plně využít, neobejdeme se v rámci tvorby kódu k akcím, které budou realizovat periodické opakování jednotlivých části. Uvažujme například hlavní smyčku programu, která bude čekat na příchozí události od uživatele, HW aj. bude překreslovat grafické uživatelské rozhraní a bude v činnost dokud, uživatel tuto aplikaci neukončí. Tento scénář by nebylo možné realizovat pomocí tzv. smyček. V dnešním díle seriálu se zaměříme na použití smyček v kódu, konkrétně smyčky For a While.

Smyčka For

První smyčku bude smyčka For. Nalezneme ji v paletě **Programming→Structures→For Loop**. Úkolem této smyčky je N krát opakovat daný kód. Standardně se tato smyčka ukončí po zadaném počtu iterací. Ovšem je možné jí zastavit i předčasně, přidáním Conditional Terminalu v kontextové nabídce. Poté je možné dano smyčku zastavit již dříve vybabením patřičné podmínky, více v další smyčce. Ovšem stále hlavním kritériem je celkový počet iterací, které má zadané k vykonání. V rámci LabVIEW je použití smyček velmi názorné, a kód který se má vykonat, resp. vykonávat musí být umístěn uvnitř smyčky.

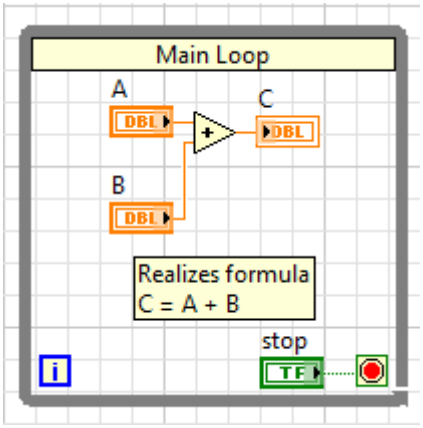


Obrázek 8 - Použití smyčky For pro výpočet faktoriálu.

Smyčka je parametrizována v terminálu „N“, který udává celkový počet iterací. Terminál „i“ vrací aktuální iteraci, indexace začíná od 0.

Smyčka While

Nyní se zaměříme na druhou ze základních smyček, a to smyčku **While**. Nalezneme ji v paletě **Programming→Structures→While Loop.** Smyslem této smyčky je opakovat daný kód po dobu splnění podmínky. V rámci LabVIEW můžeme uvažovat o dvou typech vybavení podmínky **Stop if True** a **Continue if True.** První nám bude opakovat kód do doby, než přivedeme do terminálu příznak true, druhá naopak do doby dokud tam tento příznak bude.



Obrázek 9 - Smyčka While s kódem uvnitř a připojeným tlačítkem k terminálu podmínky vykonávání.

Krom již zmíněného terminálu pro vykonání podmínky (v obrázku jako červené tlačítko, nebo v případě druhého režimu cyklická šipka) je součástí této smyčky ještě jeden terminál. Jedná se o terminál označení modře písmenem „i“. Tento terminál vrací aktuální iteraci vykonání smyčky.

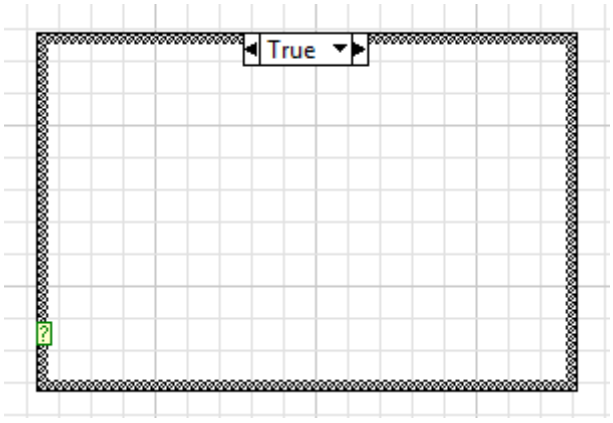
Rozhodovací algoritmy

Nezbytnou součástí programů je jeho rozvětvení na základně rozhodnutí pomocí rozhodovacích algoritmů. Vezmeme-li například jazyk C, tak základním prvkem je podmínka IF, která na základě vyhodnocení podmínky je schopná vykonat kód nebo například prvek typu SWITCH, který umožňuje širší rozvětvení. V LabVIEW tyto funkce samozřejmě nalezneme a nyní se na ně podíváme.

Struktura pro rozhodování - Case structure

Strukturu *Case Structure* nalezneme v paletě **Programming→Structures→Case Structure.** Podobně jako u smyček se nám v kódu vytvoří rám, do kterého umístíme požadovaný kód. Tato struktura obsahuje několik důležitých prvků. Jednak je to vstup pro danou podmínku, který se standardně objeví

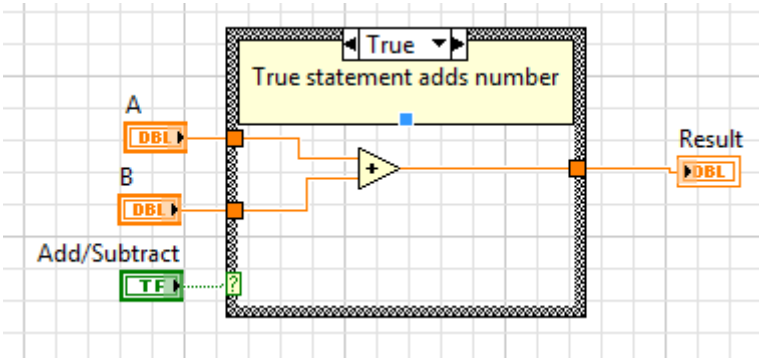
na pravé straně jako terminál s otazníkem. Druhým prvkem jsou pak jednotlivé podmínky, při kterých struktura vykonává kód. V případě booleovského typu máme standardní možnosti True/False. Zavedeme-li číselný nebo výčtový typ, máme pak mnohem více možností.



Obrázek 10 - Case Structure, vlevo terminál pro vstup, nahoře combo box pro výběr následku podmínky.

Použití s booleovským datovým typem

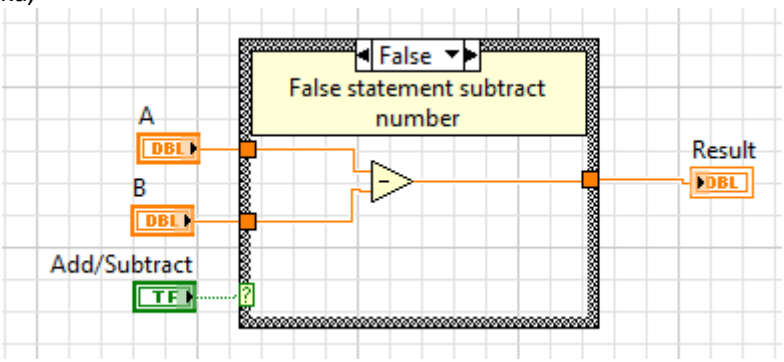
Jak již bylo zmíněno tak základním typem, který může vstoupit do terminálu je typ boolean. Tento typ nabývá pouze dvou hodnot a má tak pouze dva možné následky podmínky a to True nebo False. Daný kód musí být umístěn pod patřičným výsledkem podmínky. Na obrázku 2 a 3 jsou znázorněny realizace jednoduché funkce sčítání a odčítání, kdy tato realizace daná stavem tlačítka, jež je booleovský operátor.



Obrázek 11 - Case Structure využívající booleovský vstup. Obrázek zobrazuje kód, který se vykoná při pravdivém vyhodnocení podmínky.

Základy práce v systému LabVIEW

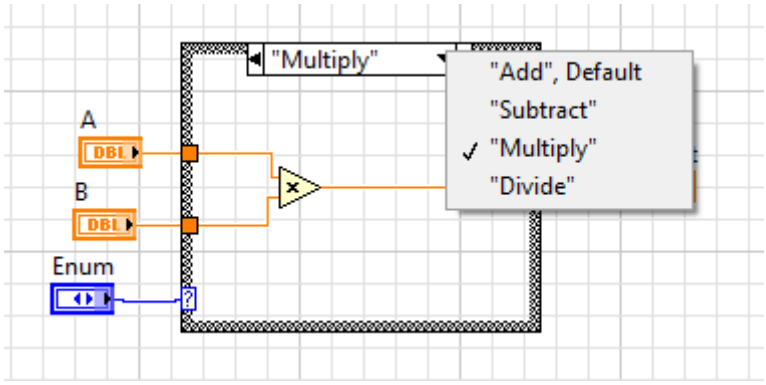
Ing. Roman Matějka,



Obrázek 12 - Case Structure využívající booleovský vstup. Obrázek zobrazuje kód, který se vykoná při nepravdivém vyhodnocení podmínky.

Použití s číselným, výčtovým nebo řetězcovým typem

Struktura Case Structure umožňuje přijímat také na vstup terminál také jiné typy než jen booleovský. Tato možnost tak poskytuje mnohem širší množství výsledků podmínky. Například v případě číselného typu to může být výsledkem pro různá čísla. Samozřejmě že nemusíme takto kvůli bezpečnosti ošetřit celý rozsah datového typu, ale je zde také výsledek typu *Default,* který se vykoná v případě že žádný výsledek nevyhovuje dané podmínce. Na obrázku 4 je další možnost, která využívá výčtový typ Enum. V tomto případě jsou výsledky podmínky dané jednotlivými prvky ve výčtovém typu.



Obrázek 13 - Použití Case Structure s výčtovým typem. Výsledky nyní obsahují jednotlivé položky výčtového typu a kód tak může být vykonán pro každou tuto položku jinak.

Datová pole

V minulých dílech našeho seriálu jsme se zaměřili na různé datové typy a ukázali si základní operace, které na nich můžeme provést. Ovšem uvažujme například nějakou měřicí aplikaci, která nám bude snímat napěťové úrovně z A/D převodníku. Samozřejmě, že se můžeme spokojit s jednou hodnotou, ovšem často budeme potřebovat zobrazit graf, nebo s většího úseku dat provést analýzu apod. V tento okamžik již jednoduchý datový typ nebude to pravé a musíme sáhnout po rozměrnějším datovém bloku – poli. V dnešním díle seriálu se právě podíváme na základní práci s polem.

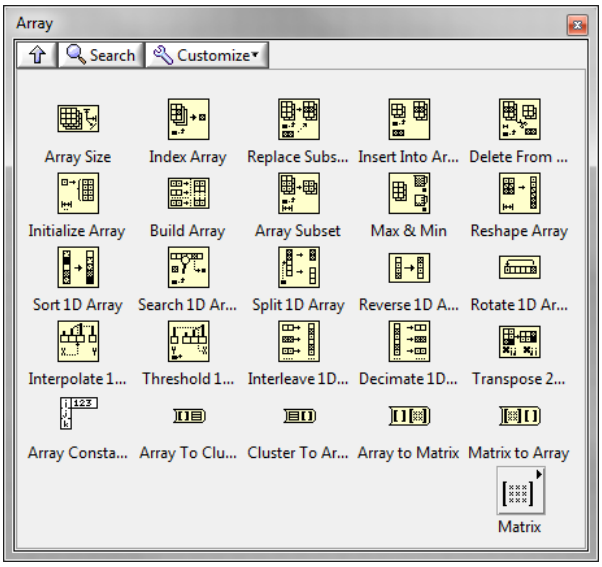
Biomedicínské inženýrství pro znalostní ekonomiku, CZ.02.2.69/0.0/0.0/16_018/0002242
Modernizace laboratoří pro biomedicínské inženýrství, CZ.02.1.01/0.0/0.0/16_017/0002244

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Ing. Roman Matějka,

Pro práci s poli obsahuje LabVIEW celou paletu nástrojů, nalezneme ji v **Programming→Arrays**.

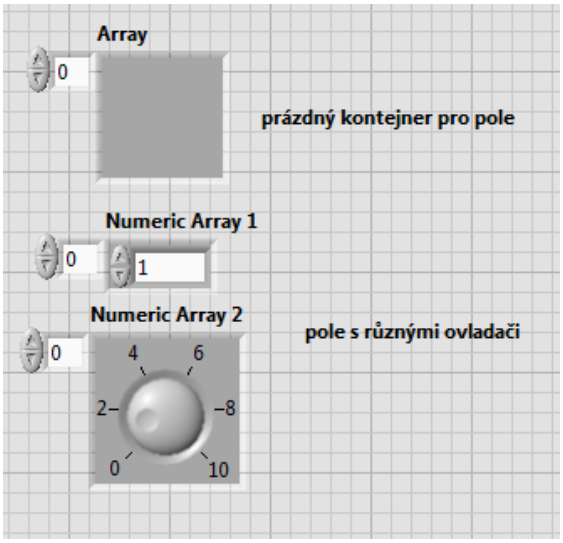


Obrázek 14 - Paleta nástrojů Arrays pro práci s poli.

Existuje několik způsobů, jak je možné vytvořit pole v rámci systému LabVIEW. V následujících sekcích se zaměříme na některé z nich:

Pole jako ovládací prvek/indikátor

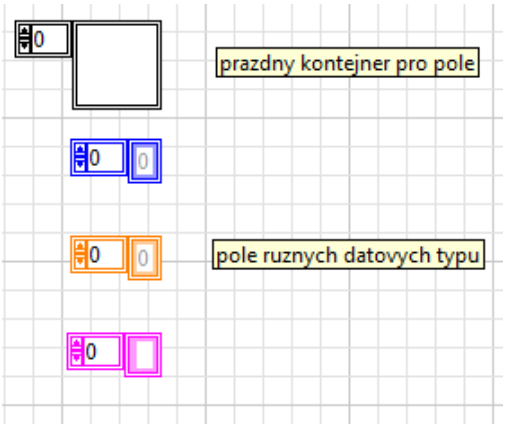
První možností je vytvoření pole jako ovládacího prvku. Ve Front panelu si vložíme prázdný kontejner pro pole, nalezneme v oblíbené paletě (Modern, Silver, Classic, System)→**Array, Matrix and cluster→Array**. Do tohoto prázdného kontejneru nyní můžeme vložit požadovaný ovladač nebo indikátor.



Obrázek 15 - Prázdný kontejner pro pole a zaplněný kontejner ovladači a indikátory.

Pole jako konstanta

Další možností je vytvoření pole jako konstanty v kódu. Postup je obdobný s tím, že použijeme v paletě **Programming→Arrays→Array constant**, což nám vytvoří prázdný blok pro datový typ. Do tohoto kontejneru opět vložíme konstantu datového typu.

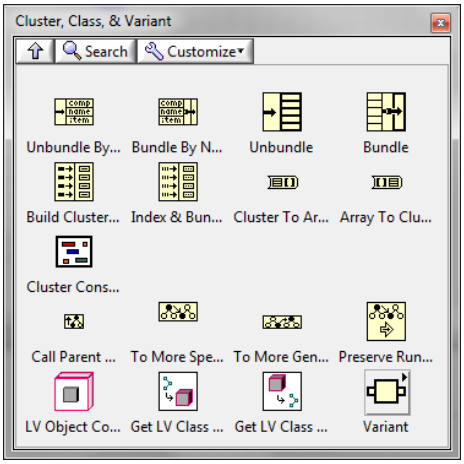


Obrázek 16 - vytvoření pole jako konstanty, prázdný kontejner, pole s různými datovými typy.

Klastry

Když vytváříme kód, tak můžeme použít mnoha datových typů, proměnných, případně je seskupit do polí a matic. Ovšem nastane situace, kdy potřebujeme vytvořit objekt, který v sobě bude obsahovat více různých datových typů. Uvažme jednoduchý příklad. Máte XY souřadnicový graf. Je jasné, že bod v tomto grafu jsme schopni sestavit pomocí dvou proměnných, X a Y. Jenže co když chceme vytvořit čáru, křivku aj. Asi Vás i napadne, že bychom mohli udělat například dvourozměrné pole, kde řádky budou vždy dvojice X a Y. Toto řešení, které je určitě funkční, udělá kód méně přehledný. Proto je vhodné použíti kontejneru, který nám umožní sestavit více datových do jedné struktury, tzv. klastru. V dnešním díle se podíváme právě na tyto klastry.

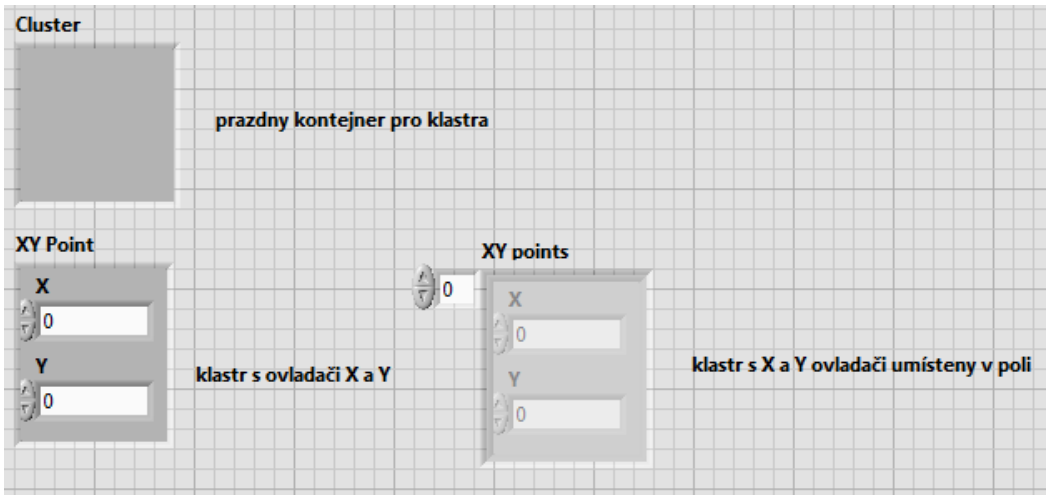
Klastry mohou seskupovat jakékoliv prvky, ovladače, indikátory, případně mohou být ve formě konstant a typových definicí. Prvky pro práci s klastry nalezneme v paletě **Programming→Clusters, Class and Variant.**



Obrázek 17 - Paleta nástrojů pro práci s klastry.

Jak vytvořit klastr

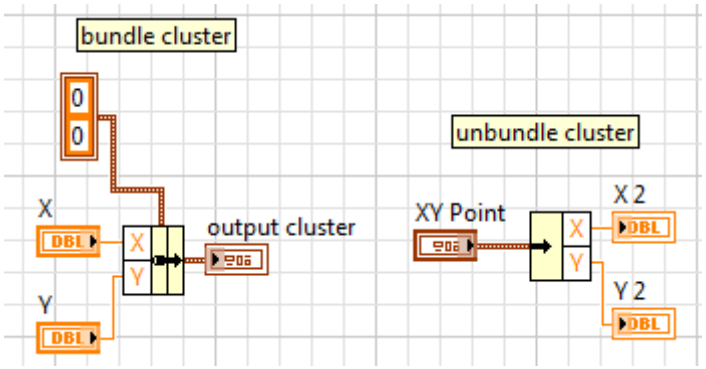
Existuje několik možností, jak vytvořit klast, avšak my se zaměříme na základní způsob. Podobně jako u polí si musíme první vytvořit určitý kontejner. Ten nalezneme v oblíbené grafické paletě v čelním panelu→**Arrays, Matrix and Clusters→Cluster.** Vytvoříme si prázdný kontejner pro tento typ. Do tohoto kontejneru nyní můžeme vkládat další prvky, tj. ovladače, indikátory, pole apod. Zároveň také celý klastr můžeme umístit do pole a vytvoříme pole s „více datovými typy“, což je například vhodné pro onen XY graf.



Obrázek 18 - Kontejner pro klastr, a vytvořené klastry.

Přístup k datům v klastru

Samozřejmě, že datový klastr by byl nesmyslný, pokud bychom neměli možnost k jeho datům přistupovat. K tomu slouží v paletě pro práci s klastry prvek **Bundle by name** a **Unbundle by name.** Bundle nám seskupí data do klastru, unbundle naopak umožní data vyčítat.



Obrázek 19 - Příkazy pro práci s klastr, bundle a unbundle.