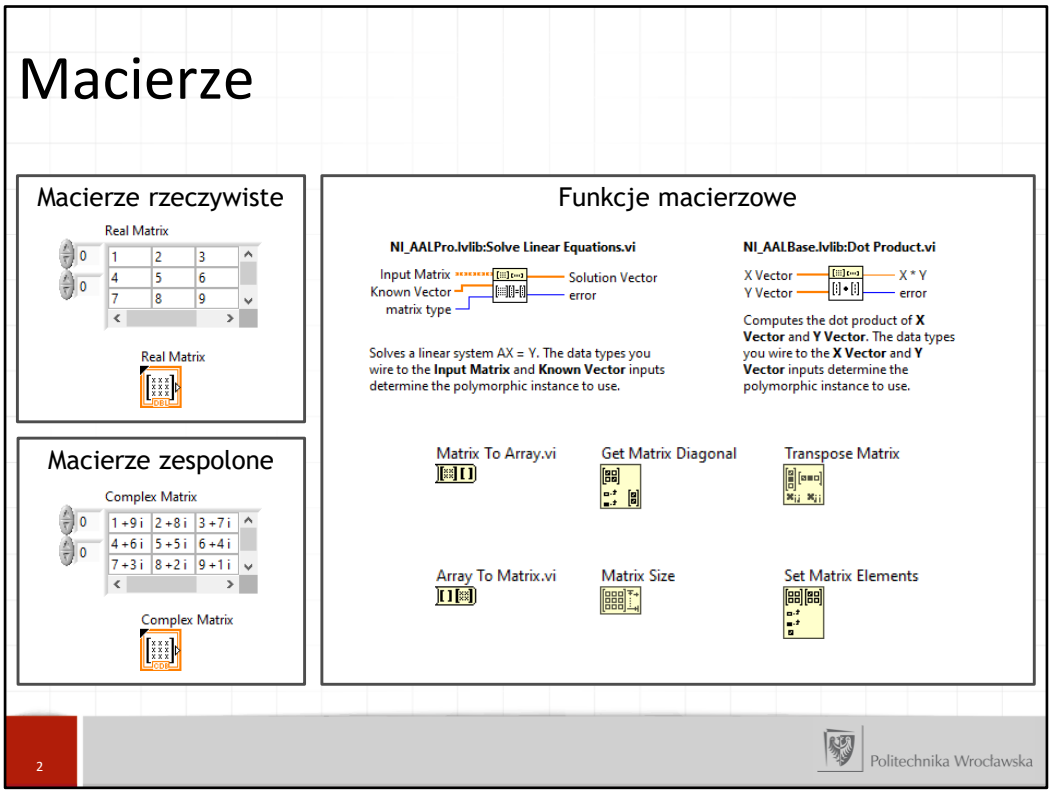
Wykład 5
05.04.2024 r.
Macierze, tablice, klastry
Obsługa plików

Adrian Kaim

Politechnika Wrocławska



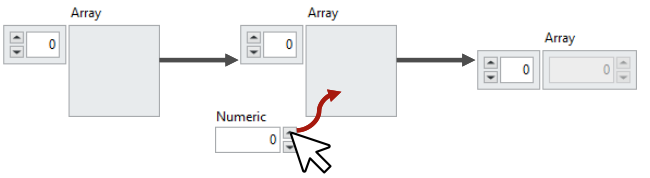
Macierze służą do przechowywania danych numerycznych w reprezentacji zmiennoprzecinkowej – rzeczywistej lub zespolonej, są zawsze dwuwymiarowe.

Macierze często wykorzystywane są do wykonywania operacji w algebrze liniowej, do czego służy gotowy zestaw funkcji dostępny w palecie.

Tablice

Tworzenie tablicy

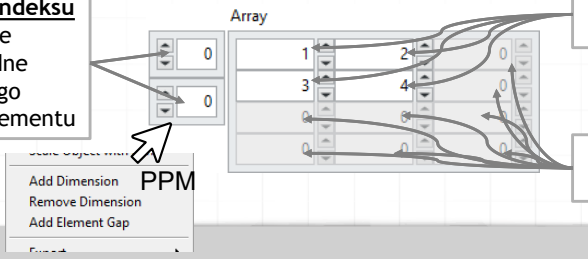
1. Stwórz pustą tablicę
2. Przeciągnij wybrany typ danych



Analogicznie w przypadku stałych

- Wiele elementów tego samego typu
- Dowolny typ danych
- Jeden lub więcej wymiarów
- Maksimum 2^{31}-1 elementów
- Elementy indeksowane od 0

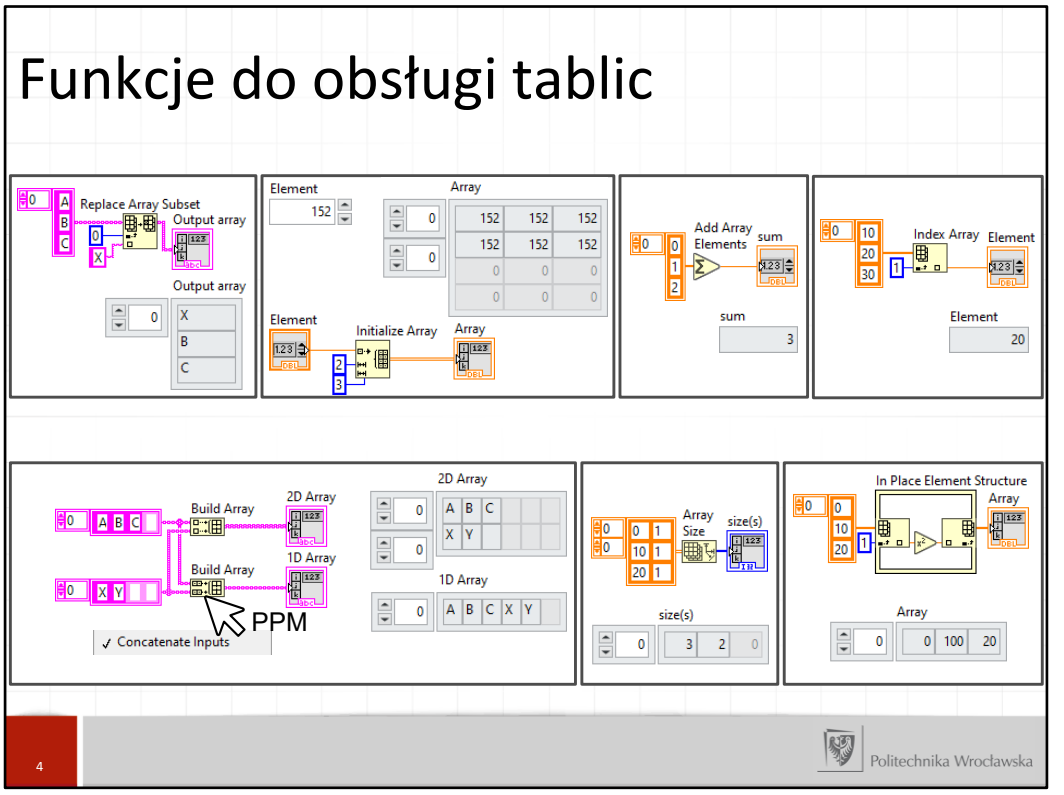
Wyświetlacz indeksu
Wskazuje współrzędne pierwszego widocznego elementu



Elementy zainicjalizowane

Elementy niezainicjalizowane

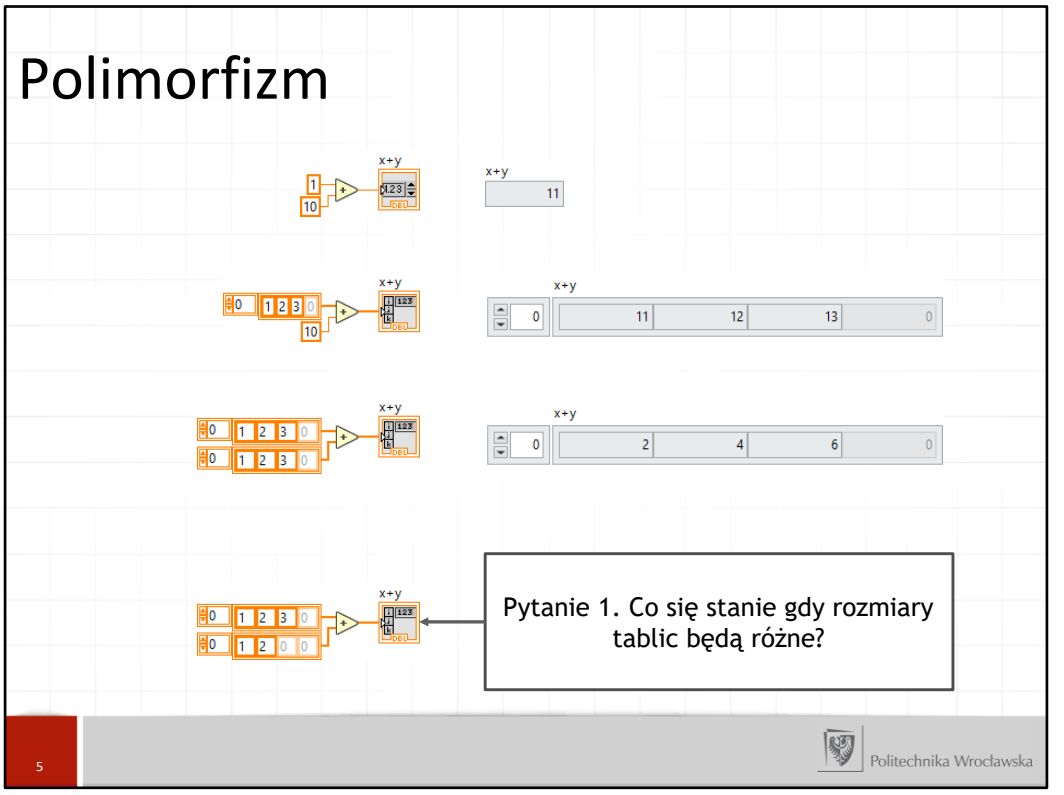
Politechnika Wrocławska



Przykłady funkcji operujących na tablicach:

- *Replace Array Subset* – pozwala zmieniać wartość wskazanego elementu tablicy;
- *Initialize Array* – pozwala stworzyć tablicę o zadanym rozmiarze i jednakowych elementach;
- *Add Array Elements* – pozwala zsumować elementy tablicy;
- *Index Array* – pozwala odczytać wartość wskazanego elementu tablicy;
- *Build Array* – pozwala łączyć elementy w tablicę. W przypadku łączenia tablic, możliwe jest stworzenie tablicy o wyższym wymiarze lub dołączenie elementów drugiej tablicy na końcu pierwszej. Przełączanie między trybami możliwe jest po naciśnięciu prawego przycisku myszy.
- *Array Size* – zwraca rozmiar tablicy we wszystkich wymiarach.

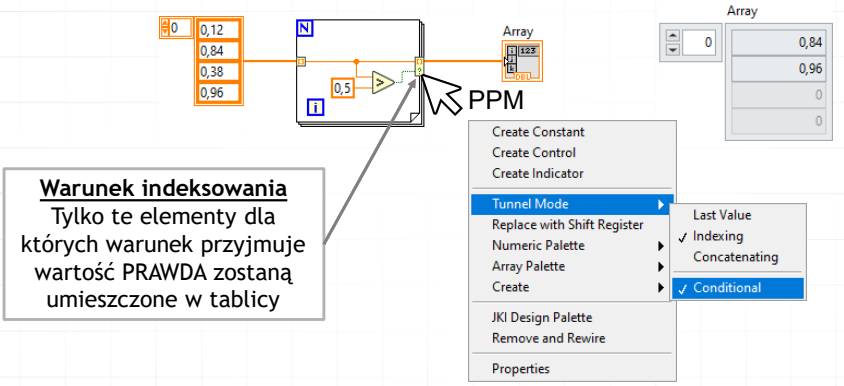
Do większości funkcji (w tym tablicowych) dane przekazywane są na zasadzie kopiowania. Czasem może okazać się, że tworzenie kopii dużych zestawów danych pochłania za dużo zasobów – w takim przypadku skorzystać można ze struktury *In Place Element Structure*, która pozwala na przeprowadzanie operacji w ramach oryginalnej tablicy.



Polimorfizm pozwala, by pozornie jedna funkcja (o zdefiniowanym opisie, wyglądzie i układzie terminali) wykonywała wiele różnych operacji, w zależności od konfiguracji i przyłączonych danych. W powyższym przykładzie, funkcja „suma" zachowa się inaczej w zależności czy jej argumentami będą wartości skalarne czy tablice.

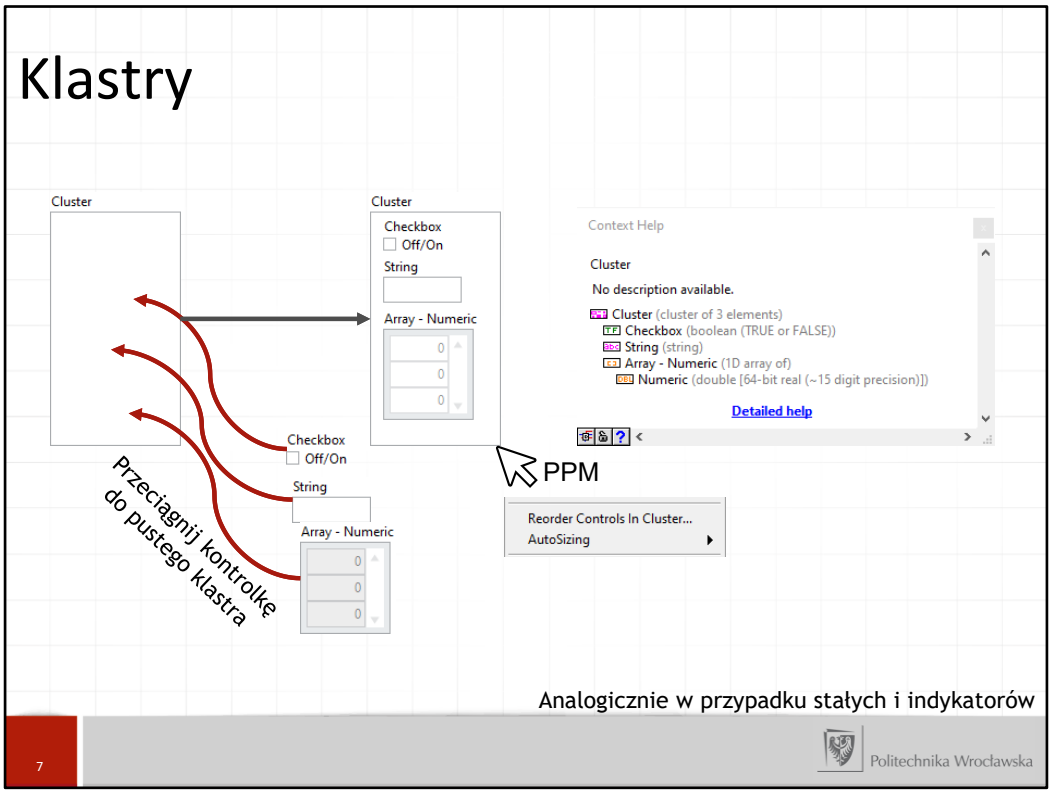
Możliwe jest tworzenie własnych funkcji polimorficznych, zbierających kilka zwykłych *SubVI* w jedną całość. Możliwe jest także stworzenie tzw. *Malleable VI*, w których w ramach jednej definicji obsługiwać można różne typy danych wprowadzane na tym samym terminalu typu *Variant*.

Auto-indexing z warunkiem



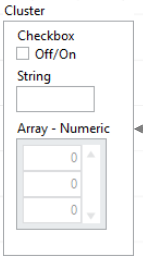
Warunek indeksowania
Tylko te elementy dla których warunek przyjmuje wartość PRAWDA zostaną umieszczone w tablicy

Politechnika Wrocławska



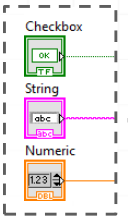
Klastry pozwalają na przechowywanie wielu danych o różnych typach. Tworzenie klastra przebiega podobnie do tablicy – przeciągając wybrane kontrolki/indykatory/stałe do jego wnętrza. Należy pamiętać, że kolejność dodawania elementów ma wpływ na to, jaki typ danych zostanie stworzony – dodanie najpierw kontrolki X, a potem Y, stworzy całkowicie inny klaster niż w kolejności YX. Kolejność elementów klastra można sprawdzić w okienku pomocy kontekstowej oraz edytować klikając prawym przyciskiem myszy i wybierając *Reorder Controls in Cluster*.

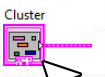
Klastry – dlaczego?



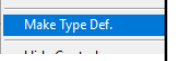
Organizacja *Front Panelu*

Organizacja *Block Diagramu*

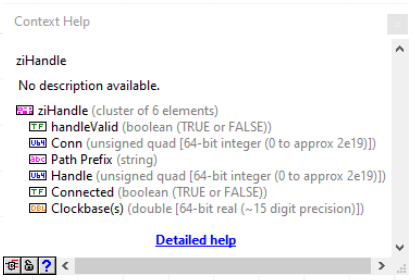




PPM

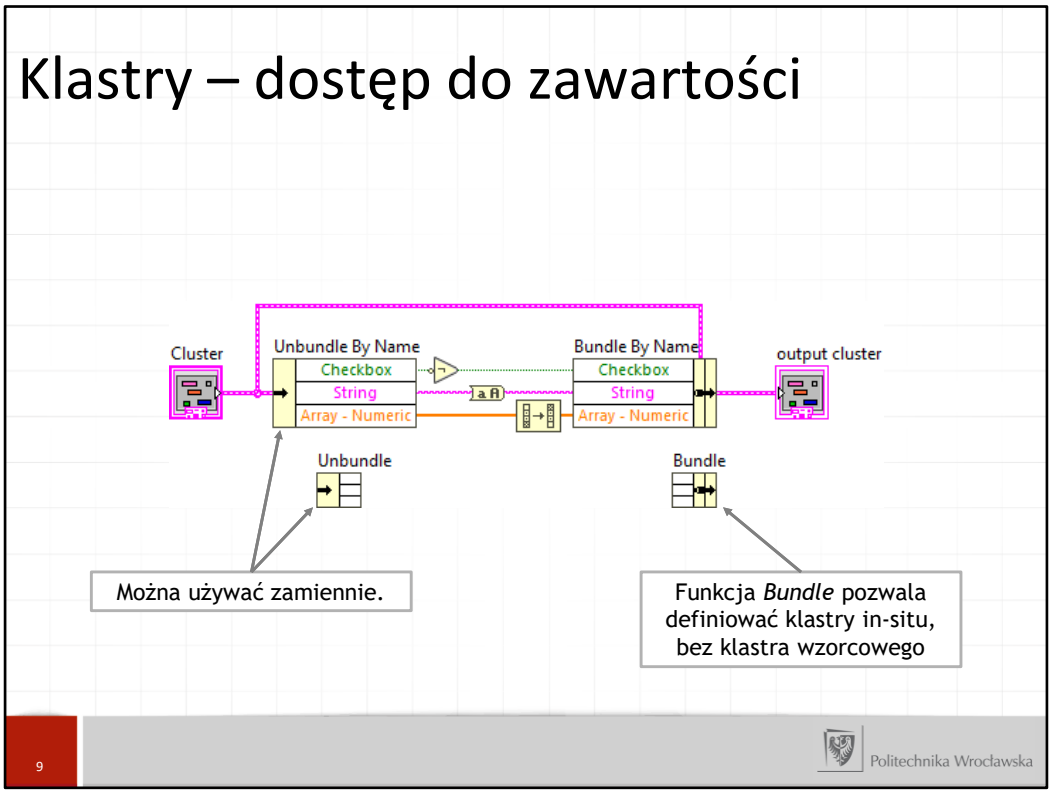


Definiowanie własnych typów danych, które można wykorzystać jako stałe, kontrolki i indykatory

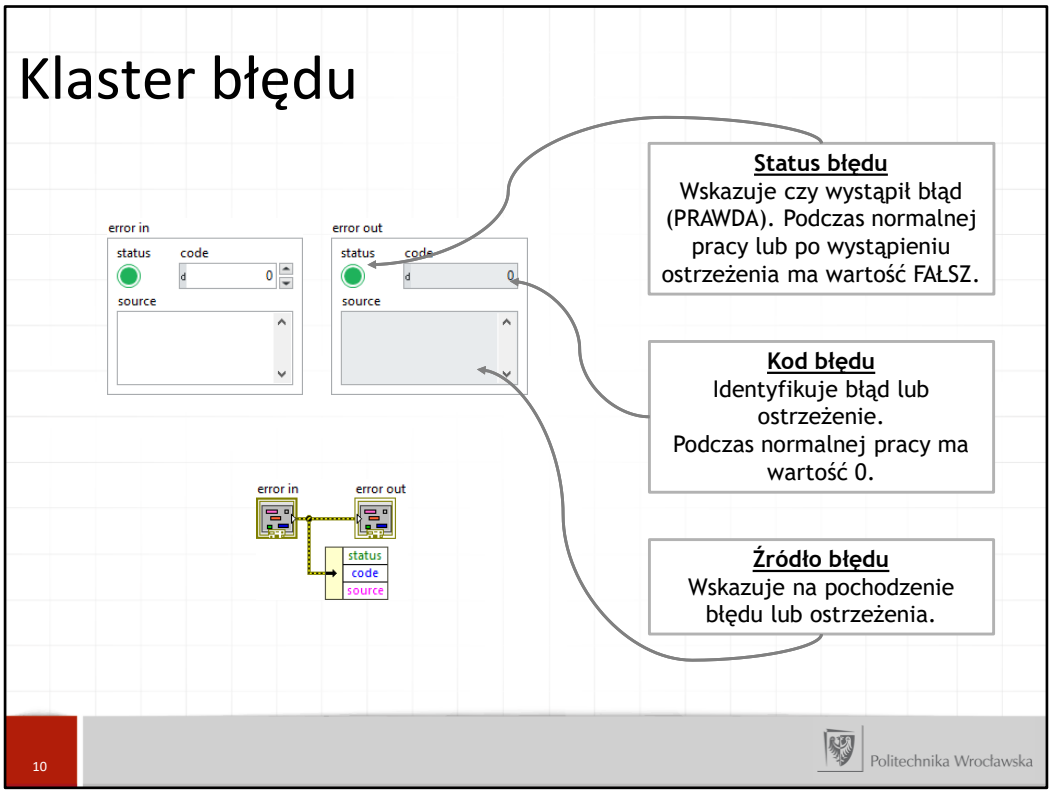


Organizacja danych, które zawsze wykorzystywane są razem

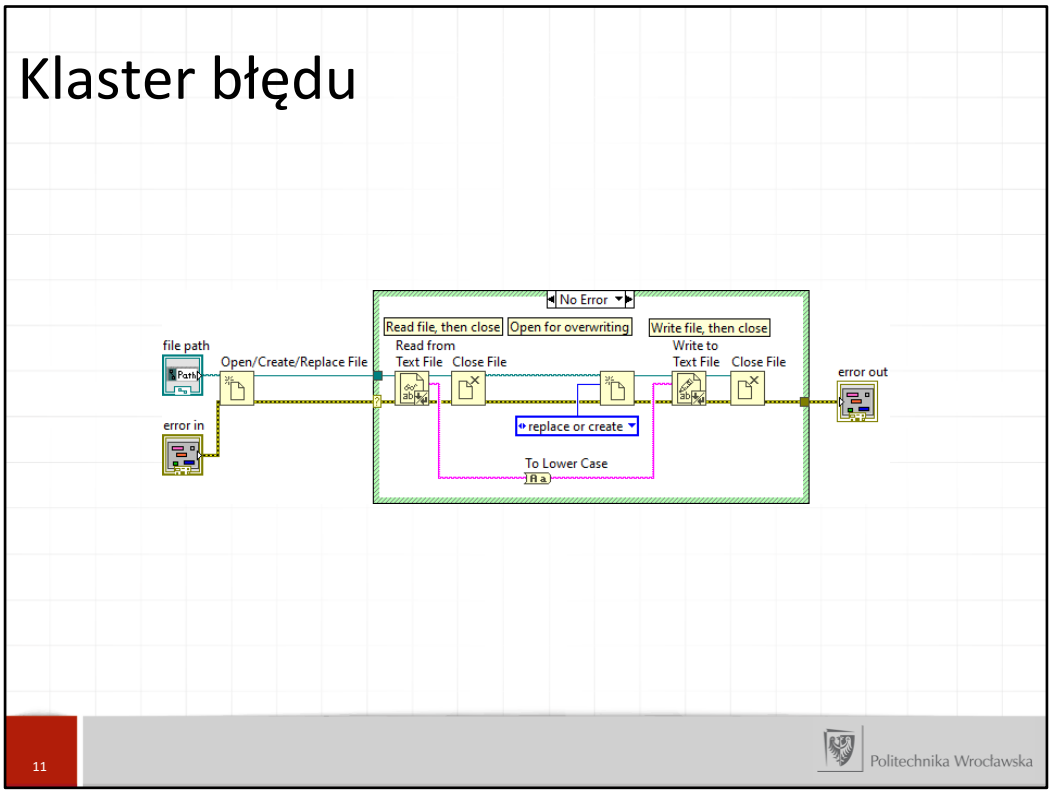
Politechnika Wrocławska



Aby uzyskać dostęp do elementów klastra, należy skorzystać z funkcji *Bundle* i *Unbundle*, a także ich odpowiedników „*By Name*”. Użycie wersji *By Name* pozwala na wskazanie konkretnych elementów klastra. Możliwe (w przypadku *Bunde By Name* konieczne) jest wskazanie klastra wzorcowego, określającego nazwy elementów i ich format.

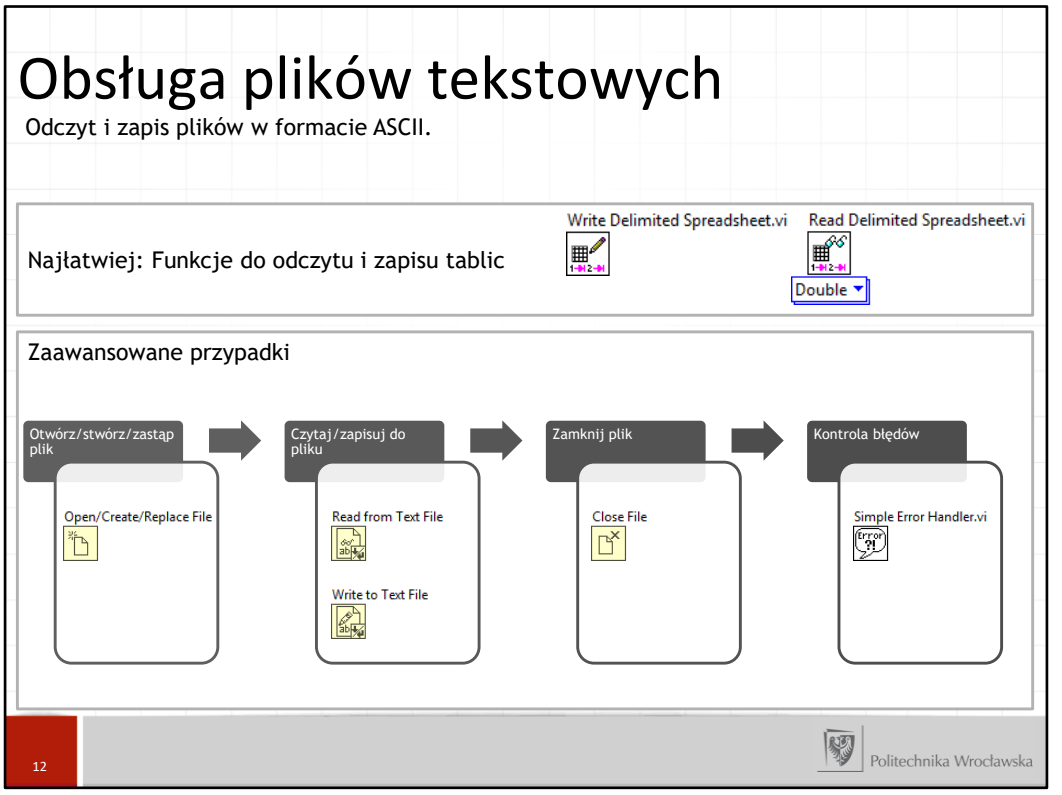


Klaster błędu to szczególny rodzaj klastra, który przechowując informacje o występowaniu błędów i ich szczegółach, pozwala na ich obsługę w samym programie.



Przykładowe zastosowanie kastra błędu – kod zawarty w strukturze *Case* wykona się tylko gdy poprzednie operacje (otwarcie pliku) nie wygenerowało błędu.

Dodatkowo, łącząc przewód klastra błędu pomiędzy kolejnymi funkcjami, można wpływać na kolejność wykonywania operacji.

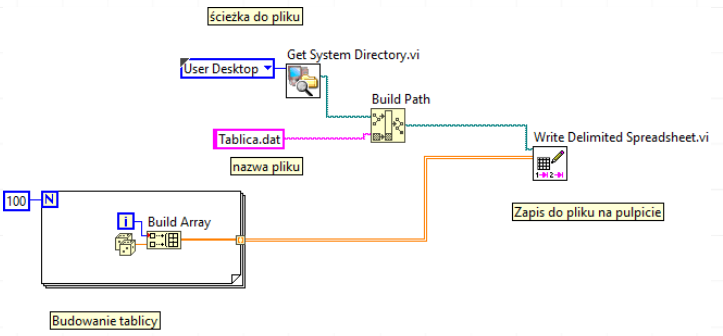


LabVIEW pozwala na interakcję z danymi zawartymi w plikach. Najczęściej spotykanym typem są pliki tekstowe w kodowaniu ASCII.

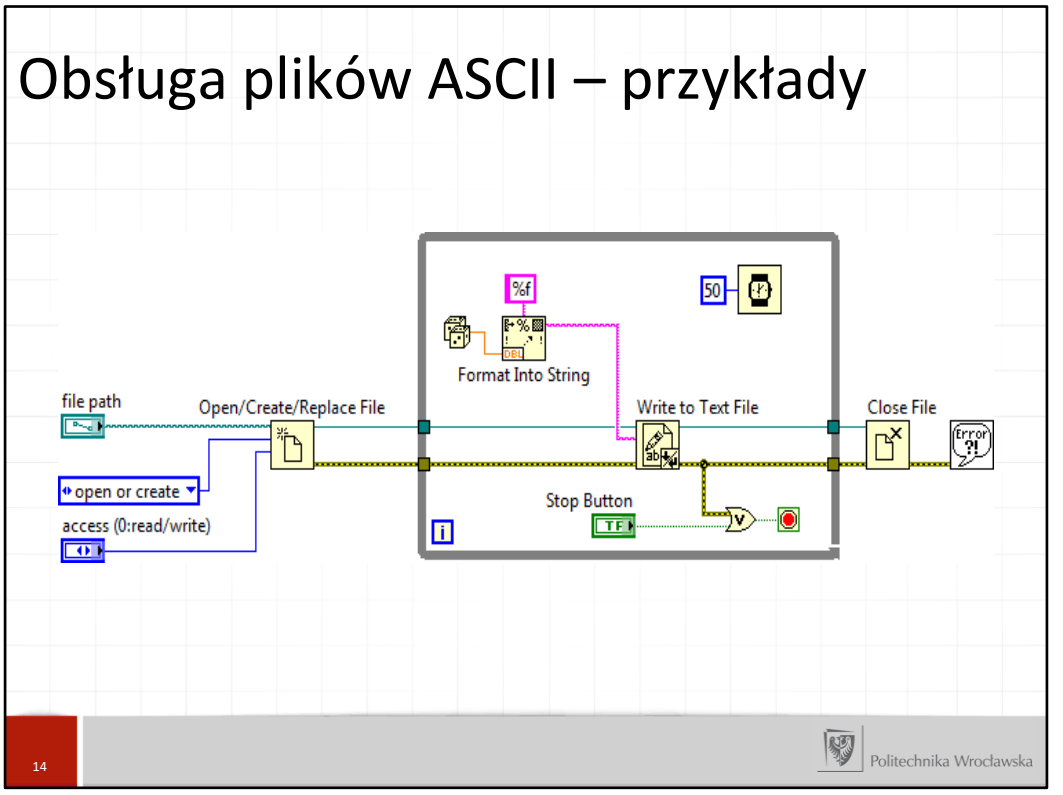
W przypadku gdy pliki przechowują dane zawarte w tablicach, można skorzystać z gotowych funkcji *Write/Read Delimited Spreadsheet.vi*, dołączając jedynie informacje o formatowaniu danych.

W bardziej zaawansowanych przypadkach należy skorzystać z funkcji pozwalających na uzyskanie bardziej bezpośredniego dostępu do plików.

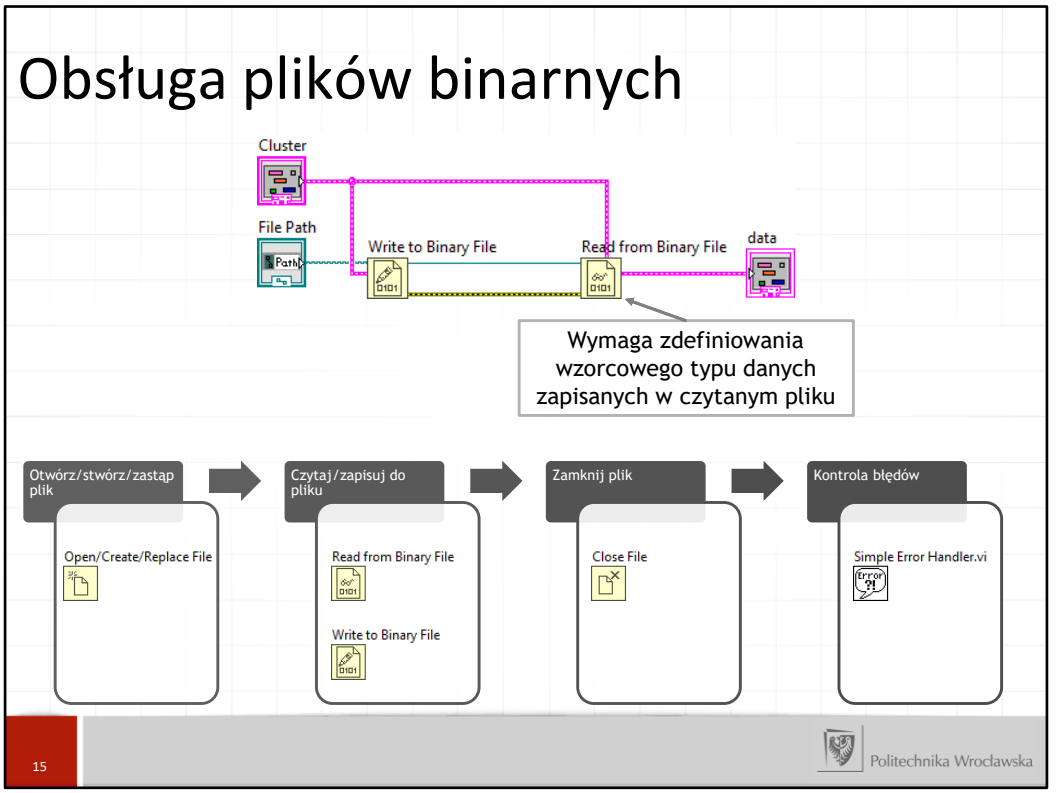
Obsługa plików ASCII – przykłady



Przykład programu zapisującego dane numeryczne w postaci tablicy 2D do pliku, korzystając z funkcji *Write Delimited Spreadsheet.vi*.



Przykład programu zapisującego dane do pliku, korzystając z funkcji dających bezpośredni dostęp do plików.



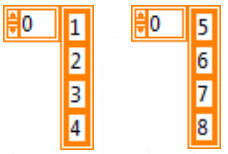
Prócz plików kodowanych tekstowo, możliwe jest także odczytywane i zapisywanie danych binarnie. Pliki takie przyjmują wówczas kodowanie analogiczne do tego, zgodnie z którym dane przechowywane są w pamięci RAM. Pliki takie przechowywać mogą dane dowolnego typu, choć najczęściej są one organizowane w klastrach i tablicach.

Należy pamiętać, że do odczytania plików binarnych konieczna jest dokładna znajomość sposobu w jaki zapisano te dane – należy prowadzić skrupulatną dokumentację kodu. Pomocne może być stworzenie definicji typu (*Type Def.*), którą łatwo wykorzystywać w różnych miejscach kodu a także w innych programach.

Zaletą wykorzystania plików binarnych jest wyeliminowanie konwersji i rzutowań, mogących zmieniać wartości zapisywanych i odczytywanych wielkości. Jednocześnie, kodowanie binarne jest często bardziej zwięzłe i pozwala oszczędzać przestrzeń dyskową – przykład: liczba *-100* w zapisie tekstowym (ASCII) zajmuje 4 bajty, a w binarnej reprezentacji int8 (liczba całkowita, 8-bitowa i ze znakiem) tylko jeden.

Zadanie 1

Stwórz dwa wektory i oblicz ich iloczyn skalarny i wektorowy bez wykorzystywania wbudowanej funkcji LabVIEW.

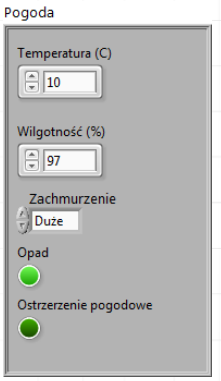


Politechnika Wrocławska

Zadanie 2

Stwórz SubVI, który będzie włączał indykator Ostrzeżenie pogodowe, gdy temperatura spadnie poniżej 0°C.

Jako dane wejściowe i wyjściowe SubVI wykorzystaj klaster zawierający dane o pogodzie (pobierz ze strony). Plik *.ctl należy przeciągnąć na block diagram. Używa się go tak samo jak zwykłych kontrolek i indykatorów.



Zadanie 3

Stwórz tablicę, w której w pierwszej kolumnie będą wartości od 0 do 2π, w drugiej wartości sin, a w trzeciej cos. Następnie zapisz utworzoną tablicę do pliku „trygonometria.dat” na pulpicie.

Zadanie 4

Rozwiąż układ równań:

$2x+3y+z=8$

$3x+y-2z=3$

$4x+y-9z=2$

Zadanie 5

Napisz program, do graficznej reprezentacji wyników uzyskanych w pomiarze charakterystyki prądowo-napięciowej.

Plik z danymi ściągnij ze strony. (Uwaga na znak dziesiętny).