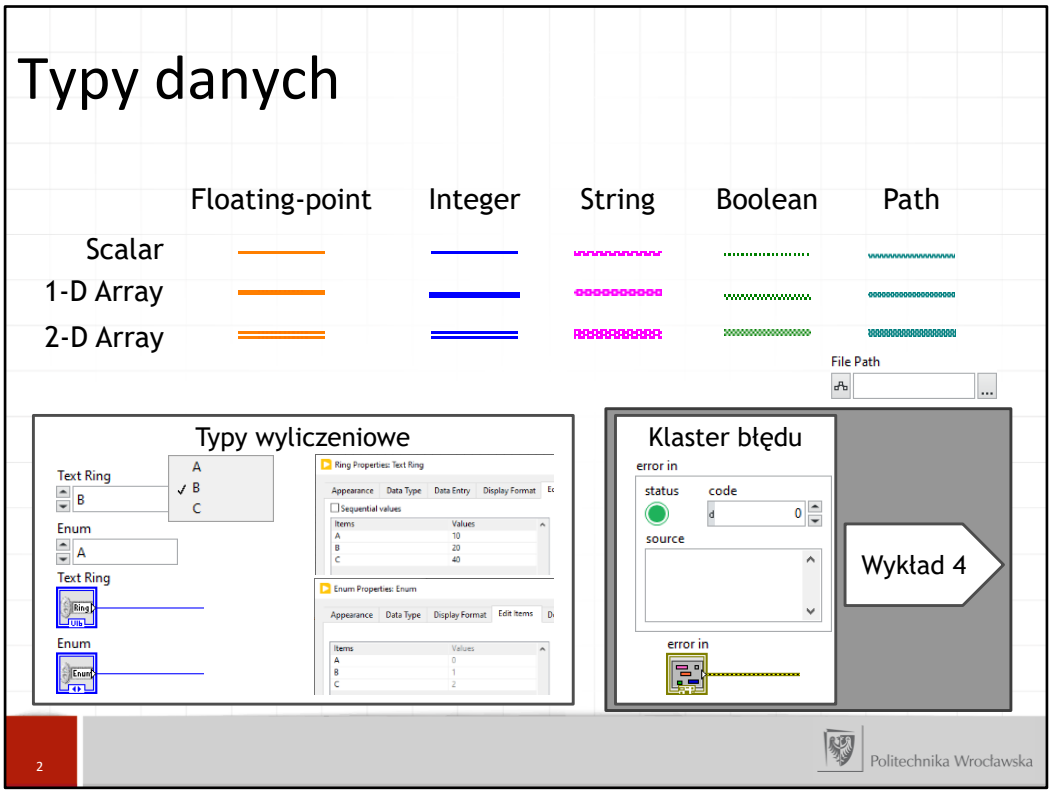
Wykład 2
08.03.2024 r.
Funkcje i struktury, cz. 1

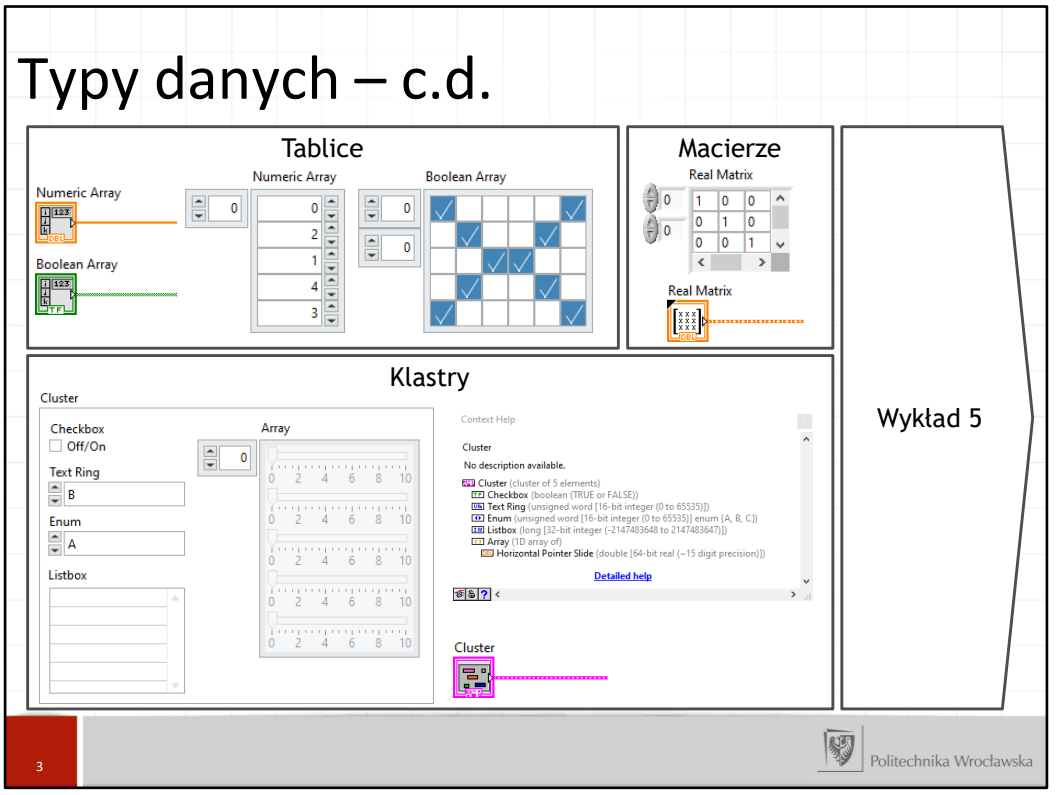
Adrian Kaim

Politechnika Wrocławska

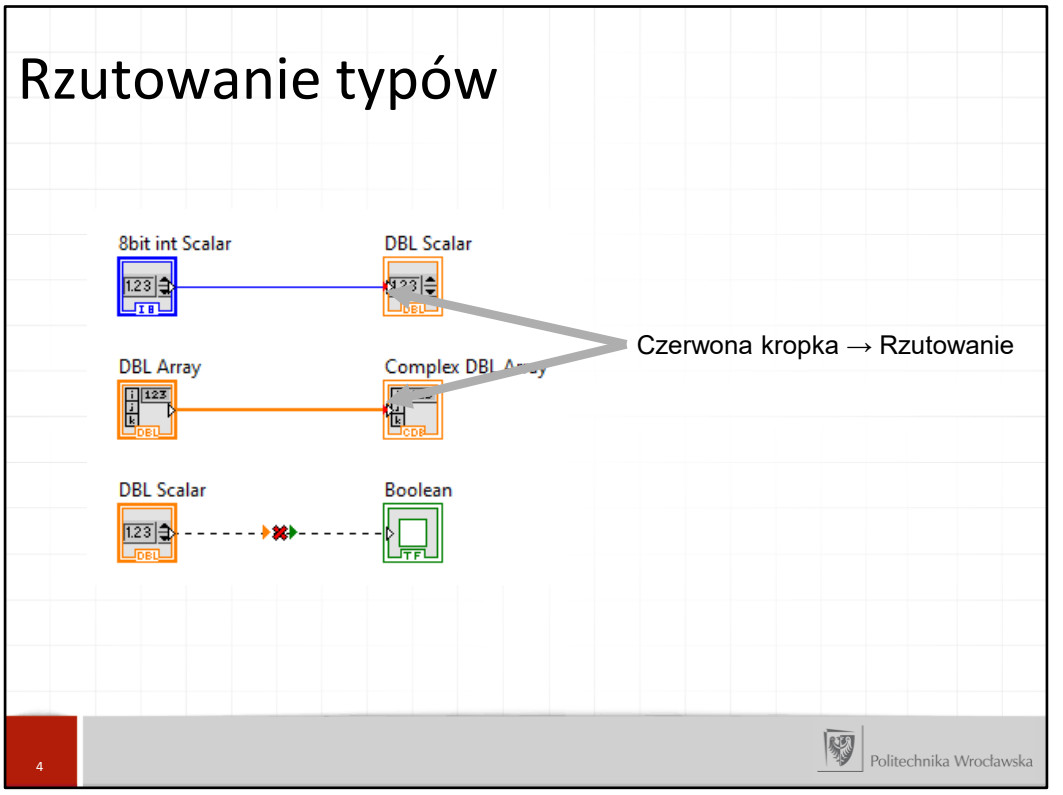


- W LabVIEW dostępnych jest kilka podstawowych typów danych: liczbowy w reprezentacji zmienno-/stało-przecinkowej, liczbowy całkowity, tekstowy, czy binarny. Dodatkowo wyróżnić można bazujące nich, bardziej szczegółowe typy takie jak bazująca na typie tekstowym ścieżka pliku lub bazujące na typie numerycznym typy wyliczeniowe. Pamięć potrzebna do przechowania danych, niezależnie od typu, alokowana jest automatycznie przez kompilator.
- Liczbowe typy danych mogą mieć różne reprezentacje, które można zmieniać w zależności od potrzeb.
- Ścieżki przepływu informacji w LabVIEW symbolizowane są przy pomocy przewodów na Block Diagramie. Każdemu z dostępnych typów danych przyporządkowany jest charakterystyczny kolor przewodu oraz ikon stałych, kontrolek i indykatorów. Przykłady kolorów przedstawiono w tabeli.
- W zależności od zorganizowania danych przesyłanych przewodem, reprezentująca go linia przyjmie różne style, zgodne z tymi przedstawionymi to w tabeli.
- Istnieją dwa rodzaje wyliczeniowych typów danych: *Enum* oraz *Ring*. Pozwalają one na wybór jednej z predefiniowanych opcji tekstu, reprezentując je przy pomocy liczb całkowitych. W typie *Enum*, numeracja zawsze rozpoczyna się od zera i wykorzystuje kolejne liczby. W typie *Ring* numeracja jest ustalana dowolnie (z zastrzeżeniem unikalności wartości).
- Dane w LabVIEW mogą być organizowane w bardziej złożone typy. Jednym z takich ustandaryzowanych typów jest klaster błędu, przechowujący wszystkie informacje konieczne do śledzenia błędów które mogą wystąpić w programie.

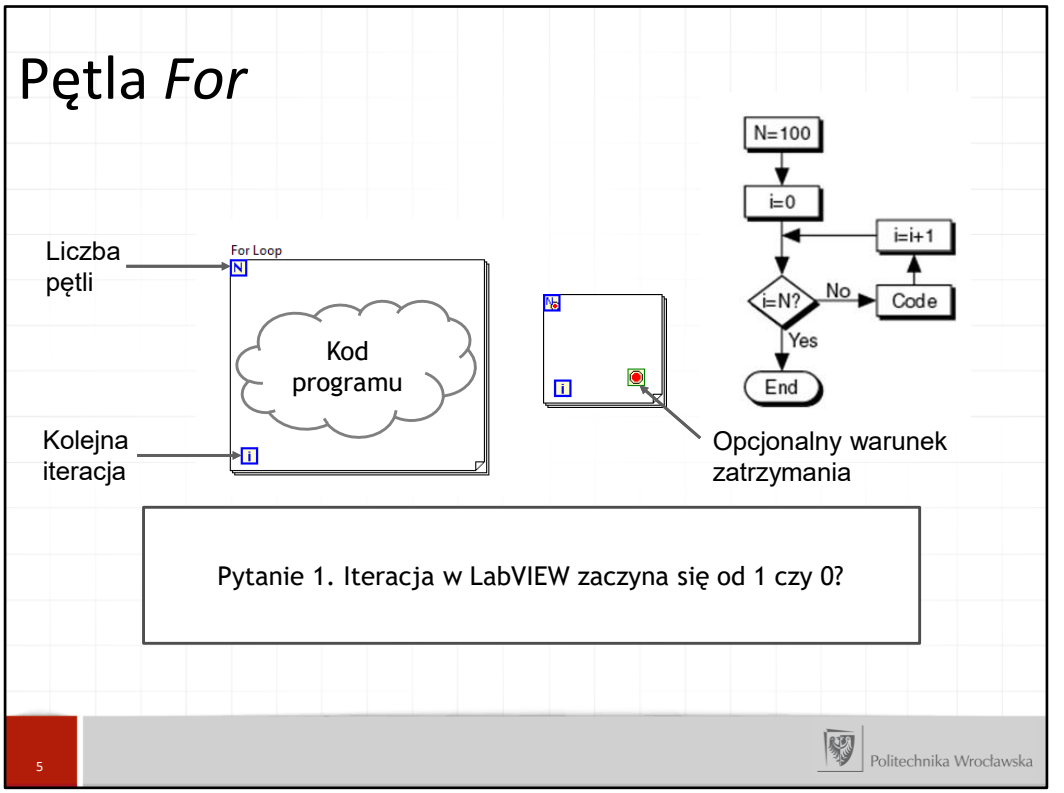
Więcej informacji na temat klastra błędów na wykładzie 4.



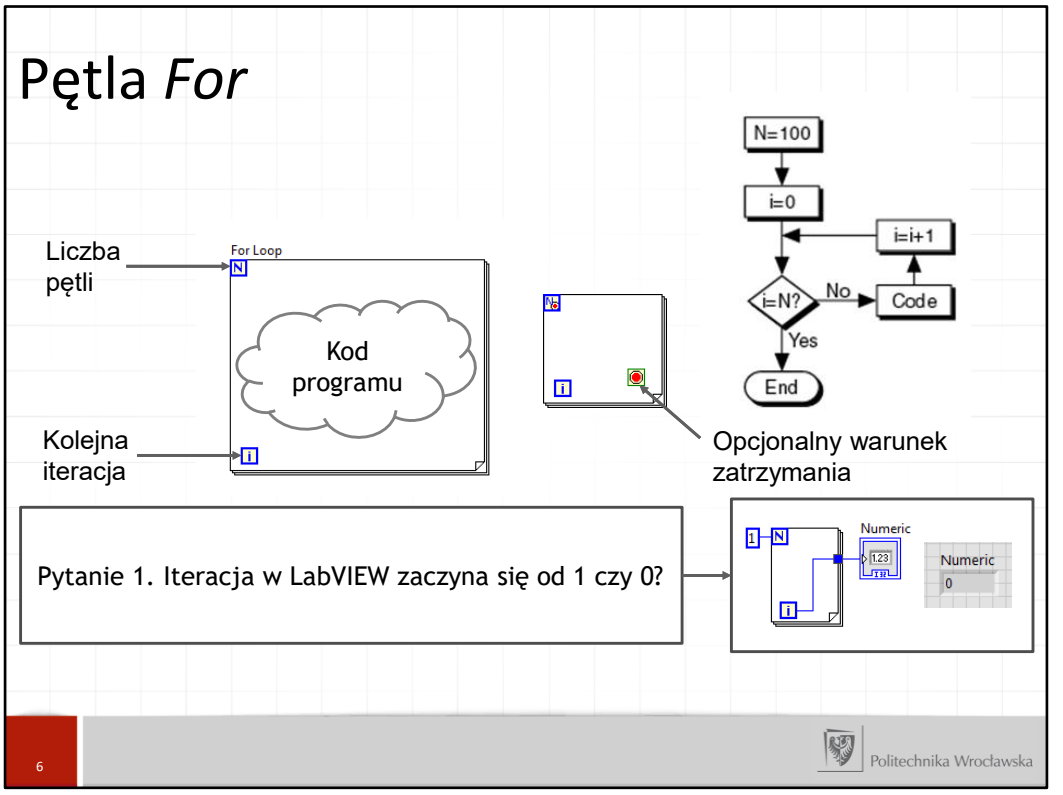
Innymi złożonymi typami danych są tablice (organizujące inne dowolne typy danych w zestawy jedno, bądź wielowymiarowe), macierze (dwuwymiarowe zestawy danych numerycznych) oraz klastry (zbierające dowolne typy danych w zestawy). Elementy składowe danego klastra mogą mieć różne typy danych. Elementy danej tablicy (lub macierzy) mogą mieć jedynie jeden typ). Więcej informacji na temat tablic, macierzy i klastrów na wykładzie 5.



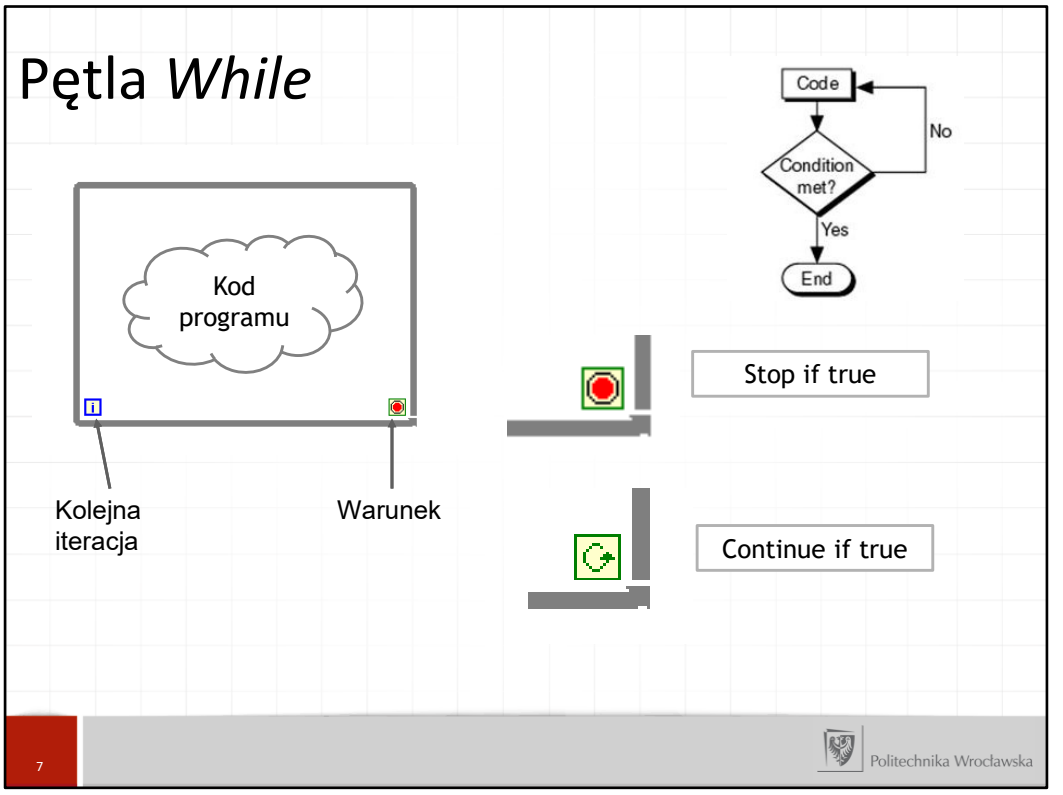
W przypadku połączenia różnych ale kompatybilnych typów danych, LabVIEW może automatycznie wykonać rzutowanie. Sytuacja taka oznaczana jest czerwoną kropką na terminalu, w którym wykonywane jest rzutowanie. Korzystając z rzutowania należy zachować ostrożność, gdyż czasem umożliwia ono nieoczekiwane działanie programu, np. gdy rzutowana wartość nie występuje w obu typach. W przypadku gdy rzutowanie nie jest możliwe (niekompatybilne typy), generowany jest błąd kompilacji, a przewód oznaczany jest czerwonym „X”.



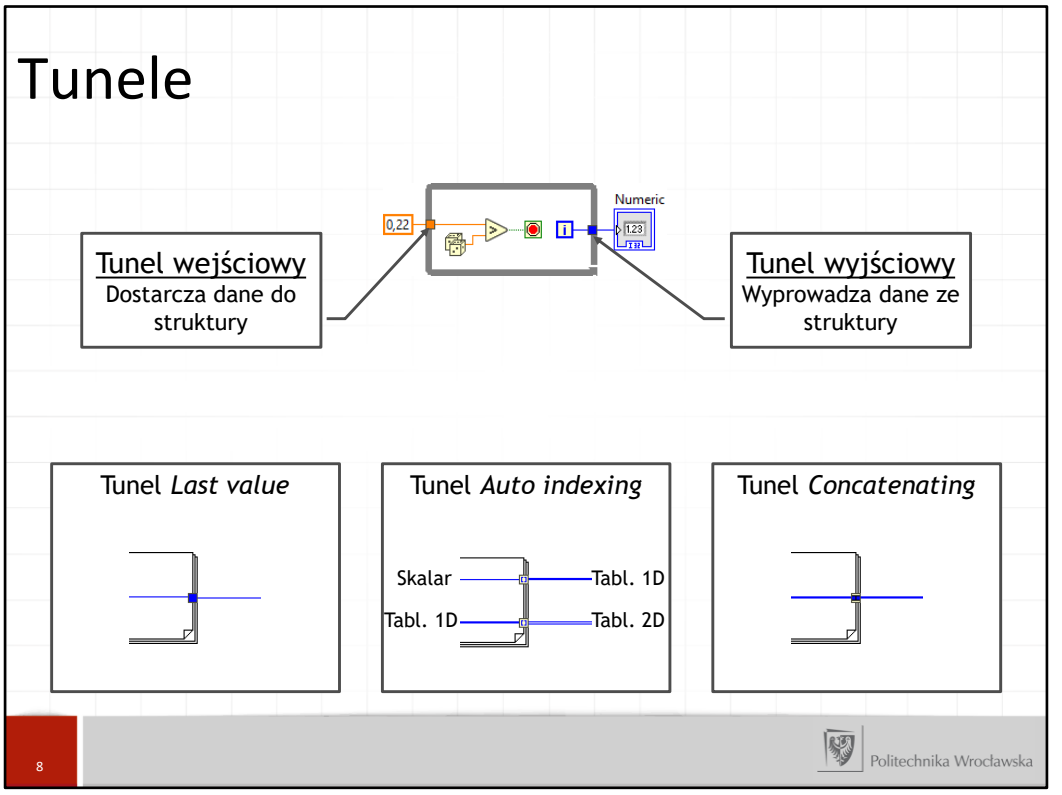
Pętla *For*, to struktura, której zadaniem jest powtarzanie wykonywania pewnego fragmentu kodu zadaną ilość razy. W LabVIEW pętla ta reprezentowana jest ramką symbolizującą stos. W lewym górnym rogu znajduje się terminal *N*, który to służy do zadawania ilości powtórzeń. Wewnątrz ramki znajduje się swobodny terminal *i*, zwracający bieżący numer iteracji. Dodatkowo (opcjonalnie), wewnątrz pętli można umieścić warunek zatrzymania, który może posłużyć do jej zatrzymania zanim osiągnięta zostanie zadana liczba powtórzeń.



Iteracja w pętlach w LabVIEW zaczyna się od 0. Można to sprawdzić zadając pojedynczą iterację ($N=1$) i wyświetlając wartość iteratora i.

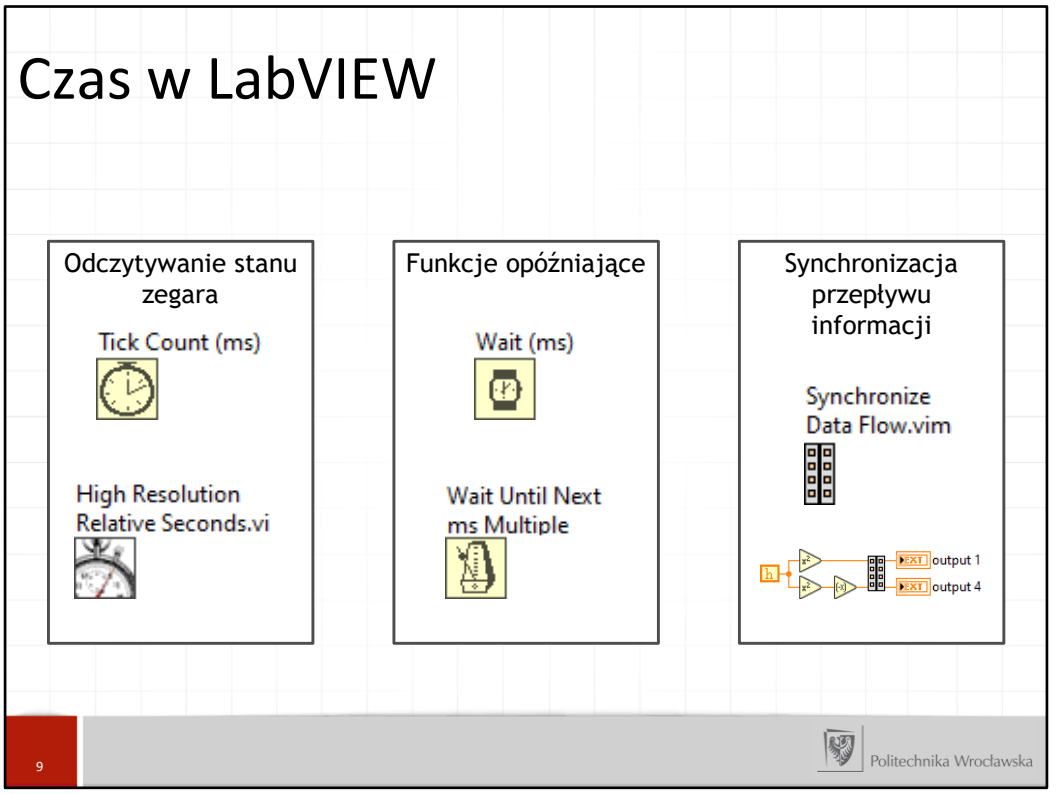


Podobną do pętli *For* strukturą, jest pętla *While*. W jej przypadku pewien fragment kodu powtarzany jest aż do spełnienia pewnego warunku. W LabVIEW pętla ta reprezentowana jest ramką symbolizującą zapętloną strzałkę. Podobnie jak w pętli *For*, wewnątrz dostępny jest terminal iteratora, a także (obowiązkowy) warunek zatrzymania. Warunek może pracować na dwa sposoby – po przekazaniu do niego wartości True może zatrzymać pętlę, lub pozwolić na kolejne powtórzenie.

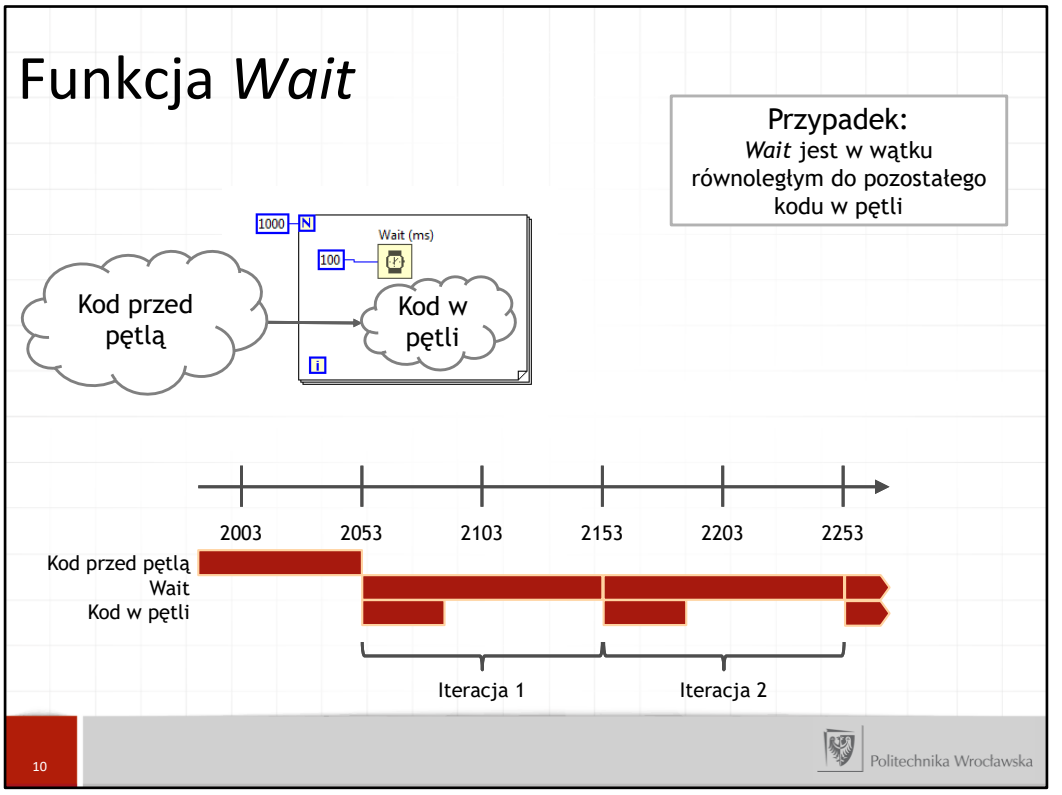


Tunele w LabVIEW pozwalają dostarczać dane do wnętrza struktury oraz wyprowadzać je na zewnątrz. W szczególności, w przypadku struktur będących pętlami, tunele można podzielić na trzy kategorie:

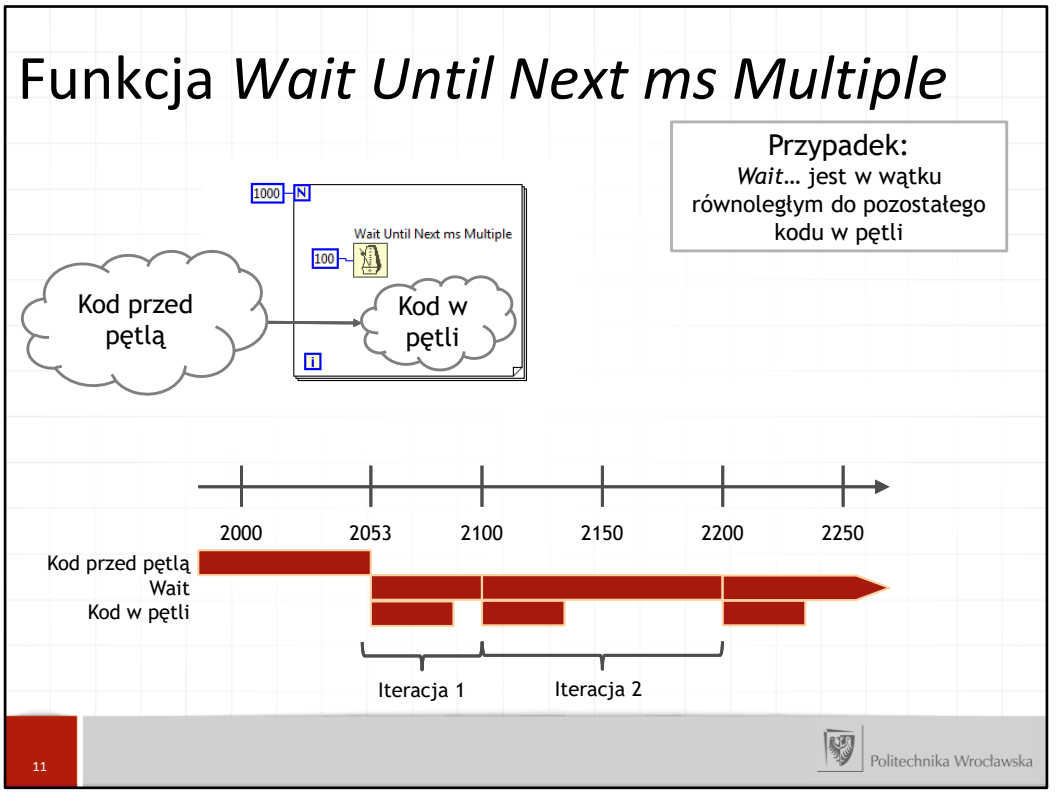
- Tunele *Last value* – najprostszy z tuneli, przekazuje dane bez jakiejkolwiek ingerencji w nie; gdy służy do wyprowadzania danych z pętli, przekazuje jedynie wartość z ostatniej iteracji
- Tunele *Auto indexing* – tunel indeksujący po elementach tablicy. Gdy do tego typu tunelu wejściowego wprowadzona zostanie tablica, w kolejnych iteracjach do wnętrza pętli przekazywane będą kolejne jej elementy (indeksując po ostatnim wymiarze). Jeżeli tego typu tunel jest tunelem wyjściowym, dane przekazywane z każdej iteracji złożone zostaną w tablicę.
- Tunele *Concatenating* – tunel, który dołącza tablicowe dane wyjściowe z każdej iteracji do takiej samej tablicy (nie dodaje kolejnego wymiaru tablicy).



W LabVIEW istnieje wiele funkcji związanych z organizacją czasu wykonywania kodu. Do odczytywania bieżącego stanu zegara procesora wykorzystać można funkcje *Tick Count (ms)* oraz *High Resolution Relative Seconds.vi*, w zależności od wymaganej dokładności. Do spowolnienia kodu wykorzystywane są funkcje typu *Wait*. Jeśli zachodzi konieczność synchronizacji wykonania różnych fragmentów kodu, można skorzystać z funkcji *Synchronize Data Flow.vim*, która to spowoduje zsynchronizowanie danych które przez nią przepływają. Dodatkowo dostępne są bardziej zaawansowane konstrukcje takie jak np.: *Timed Structures*, *Rendezvous*, *Semaphores*, czy klatki.

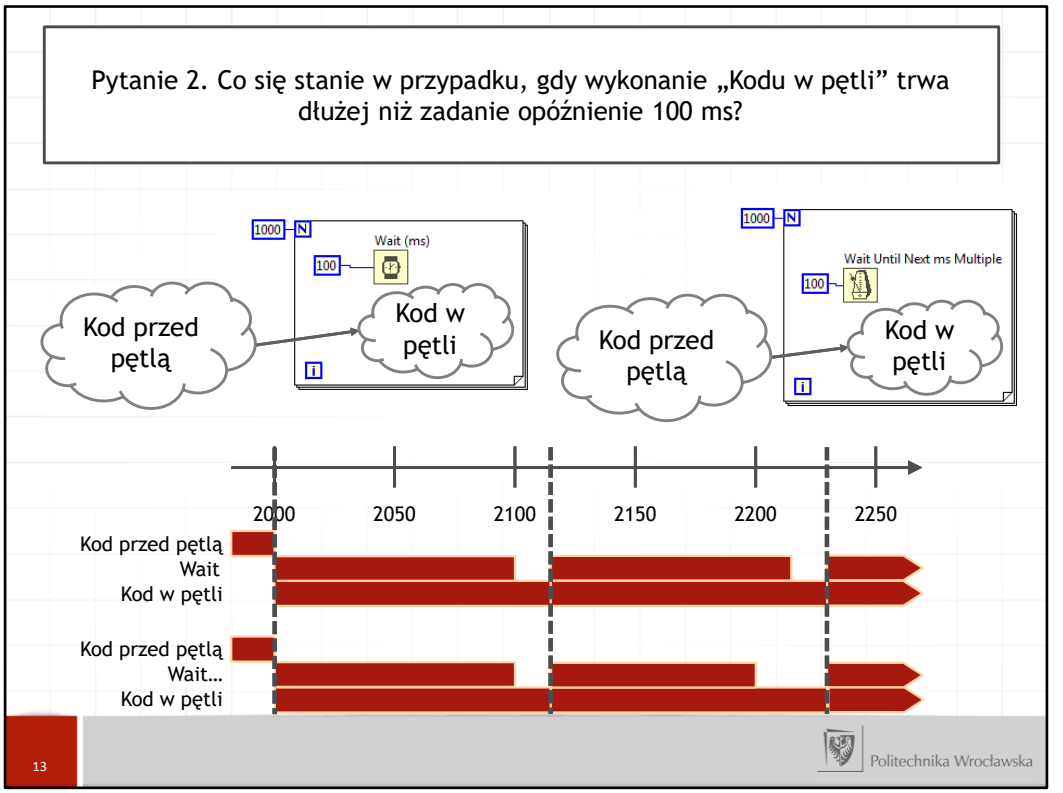


Funkcja *Wait* służy do opóźniania wykonania kodu o ustalony czas. Może dzięki temu być pomocna w spowolnieniu pętli, która przed przejściem do kolejnej iteracji będzie czekać na zakończenie wszystkich procesów wewnątrz. W przypadku, gdy funkcja *Wait* umieszczona zostanie w osobnym wątku, niezależnym od pozostałego kodu wewnątrz pętli (jak na slajdzie), należy potraktować ją jako proces równoległy w czasie względem tego kodu. Aby funkcja *Wait* opóźniała program po zakończeniu pozostałego kodu, należy ją z nim powiązać przy użyciu np.: konstrukcji synchronizujących.

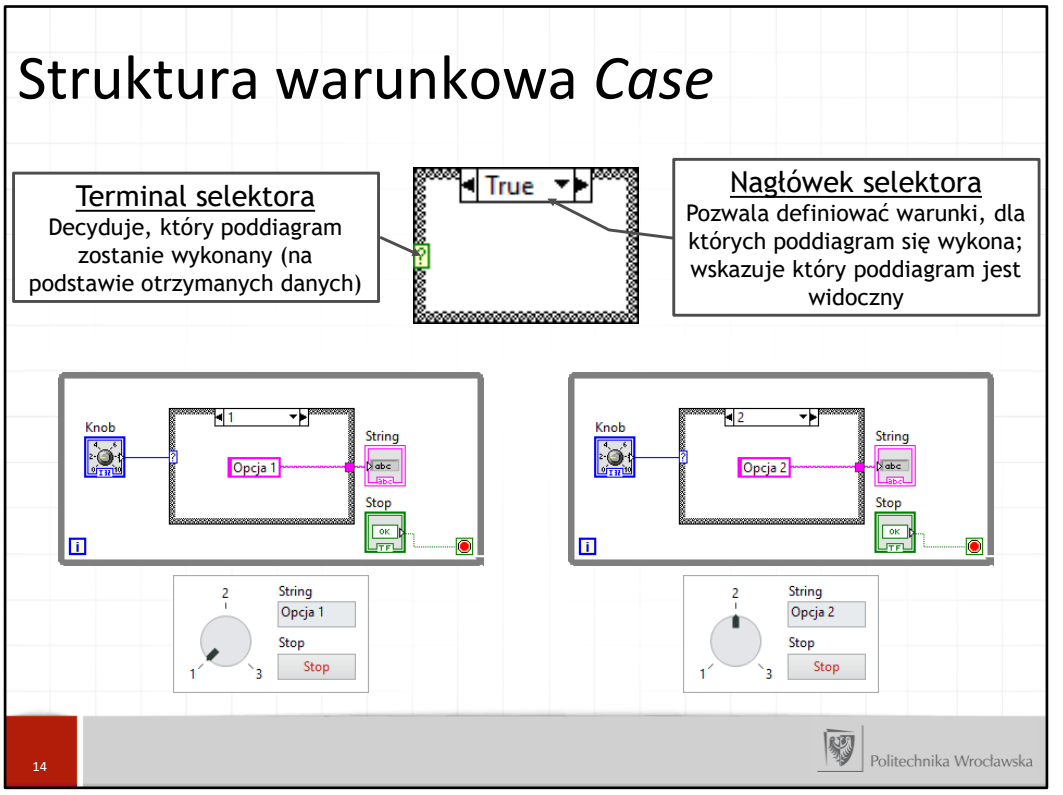


Funkcja *Wait Until Next ms Multiple* służy do opóźniania wykonania kodu do momentu, w którym zegar procesora wybije wielokrotność zadanego czasu. Może dzięki temu być pomocna np.: w synchronizowaniu różnych pętli, których kolejne iteracje będą rozpoczynać się (w przybliżeniu) w tych samym czasie. W przypadku, gdy funkcja *Wait...* umieszczona zostanie w osobnym wątku, niezależnym od pozostałego kodu wewnątrz pętli (jak na slajdzie), należy potraktować ją jako proces równoległy w czasie względem tego kodu. Aby funkcja *Wait* opóźniała program po zakończeniu pozostałego kodu, należy ją z nim powiązać przy użyciu np.: konstrukcji synchronizujących. Dodatkowo, ze względu na to, że pierwsze wejście do pętli może nastąpić w czasie innym niż wielokrotność zadanej liczby, opóźnienie w pierwszej iteracji może być dużo krótsze niż zadane.

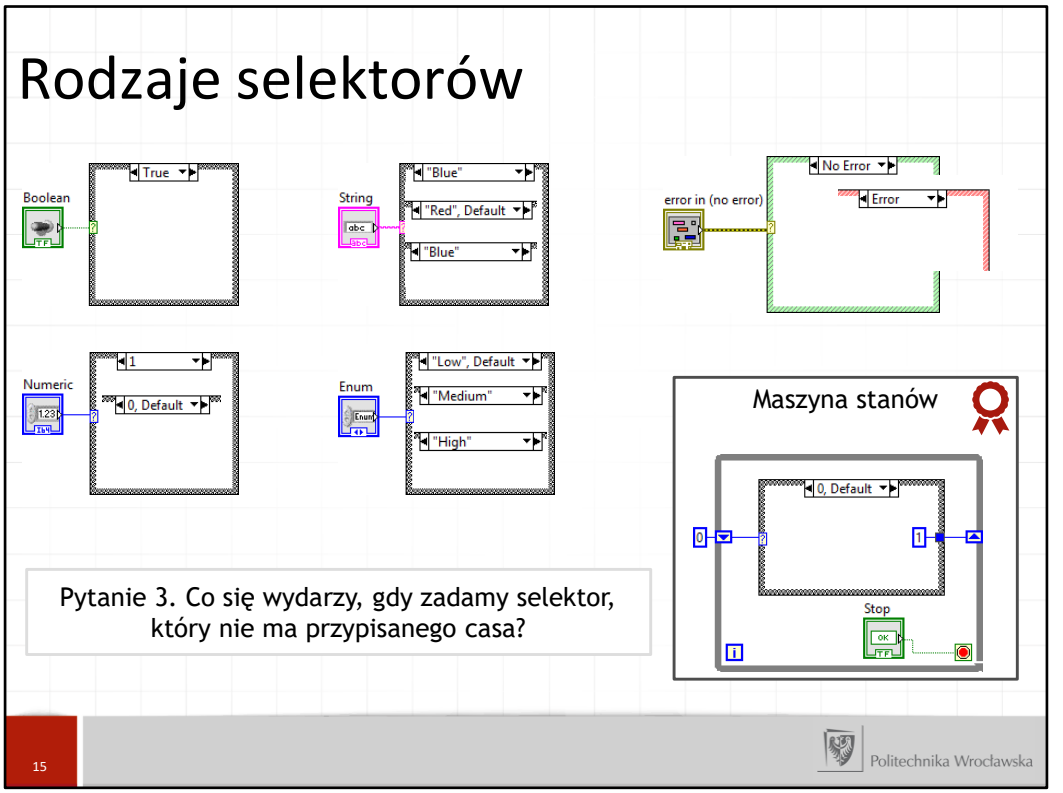
Pytanie 2. Co się stanie w przypadku, gdy wykonanie „Kodu w pętli” trwa dłużej niż zadanie opóźnienie 100 ms?



Pętla oczekuje na zakończenie wszystkich wątków wewnątrz. W opisanym przypadku, spowolnienie nie ma wpływu na szybkość iterowania pętli.



Struktura warunkowa *Case*, pozwala decydować, który fragment kodu zostanie wykonany, w zależności od tego jakie dane przekazano do selektora tej struktury. Nagłówek selektora pozwala: definiować warunki, po których spełnieniu dany poddiagram się wykona, dodawać i usuwać poddiagramy (prawy przycisk myszy), a także przełączać między wyświetlanymi poddiagramami.



Struktura *Case* automatycznie adaptuje się do przyłączonego do selektora typu danych. Obsługiwane typy to:

- binarny,
- liczbowy całkowity,
- tekstowy
- wyliczeniowy
- klaster błędu.

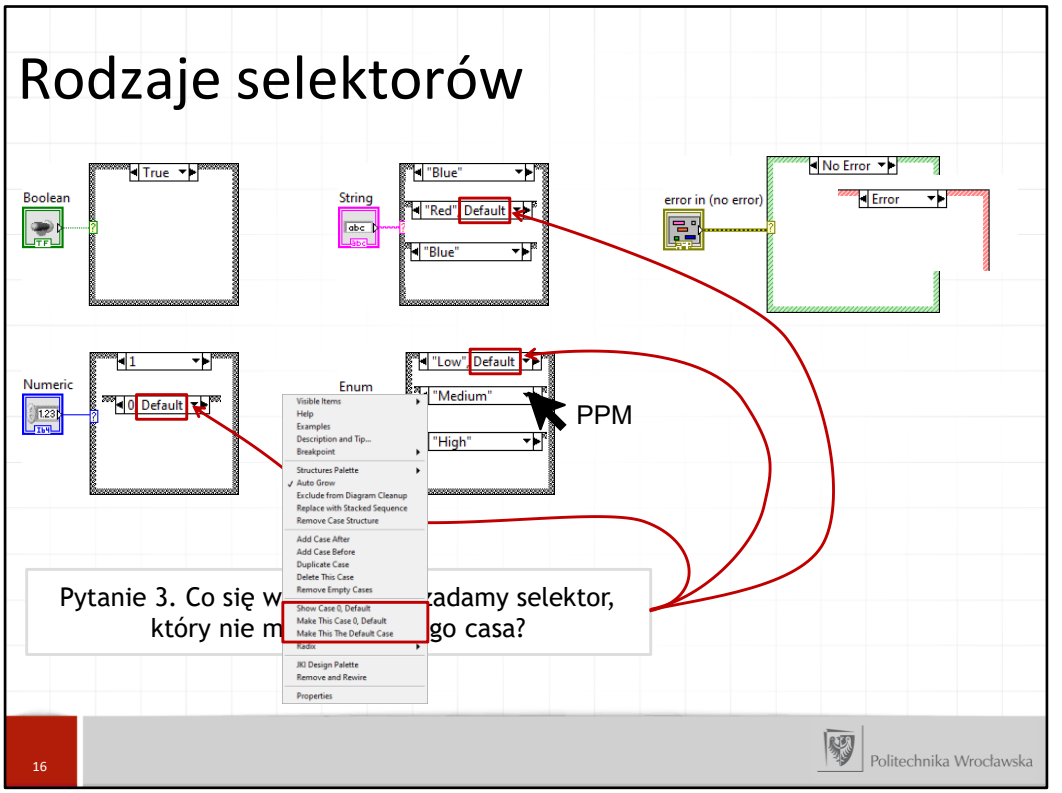
Możliwe jest ustawienie wielu warunków wskazujący na dany poddiagram jednocześnie. Kod w takim diagramie wykona się niezależnie od tego który z wymienionych po przecinku warunków został spełniony.

Dla selektora typu liczbowego, możliwe jest zdefiniowanie przedziałów liczb, które mają spełniać warunek:

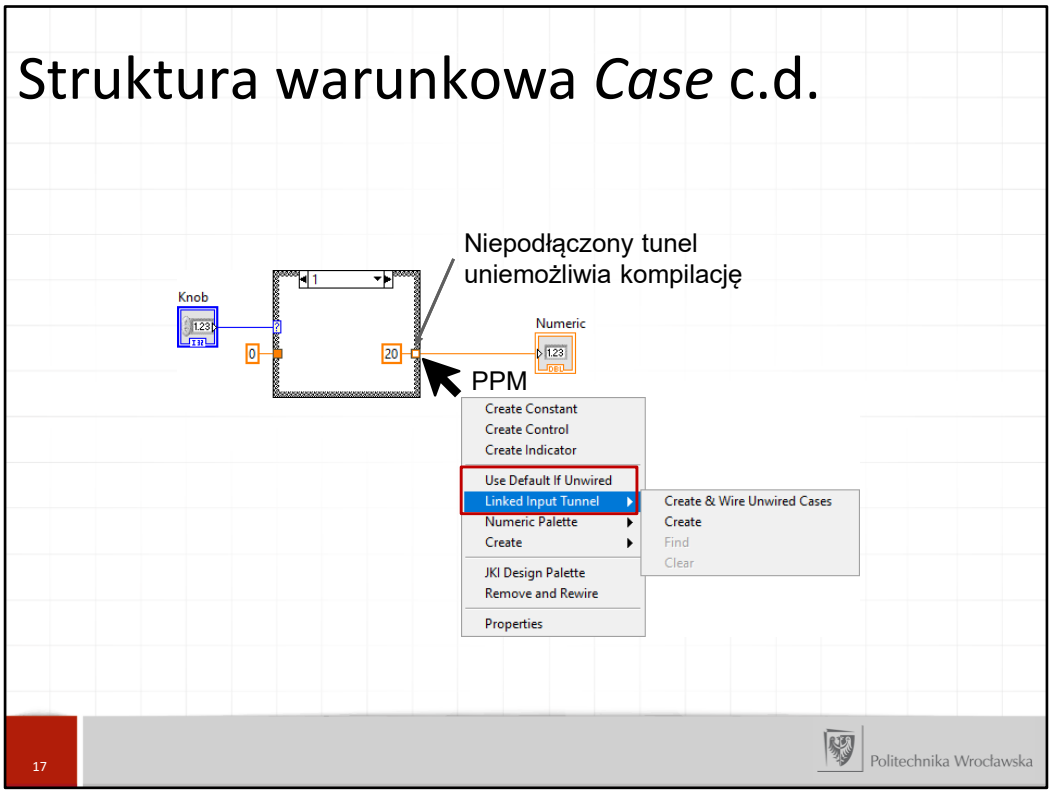
- **x..** – warunek dla wszystkich liczb większych od/równych x
- **..x** – warunek dla wszystkich liczb mniejszych od/równych x
- **x..y** – warunek dla wszystkich liczb większych od/równych x i mniejszych od/równych y

W przypadku podłączenia selektora typu wyliczeniowego *Enum*, struktura *Case*

automatycznie importuje wartości tekstowe – wartości liczbowe im odpowiadające nie są wyświetlane w nagłówku. Dodatkowo struktura sprawdza, czy dany warunek istnieje w wyliczeniu, a także pozwala automatycznie stworzyć poddiagramy dla wszystkich wyliczanych wartości.

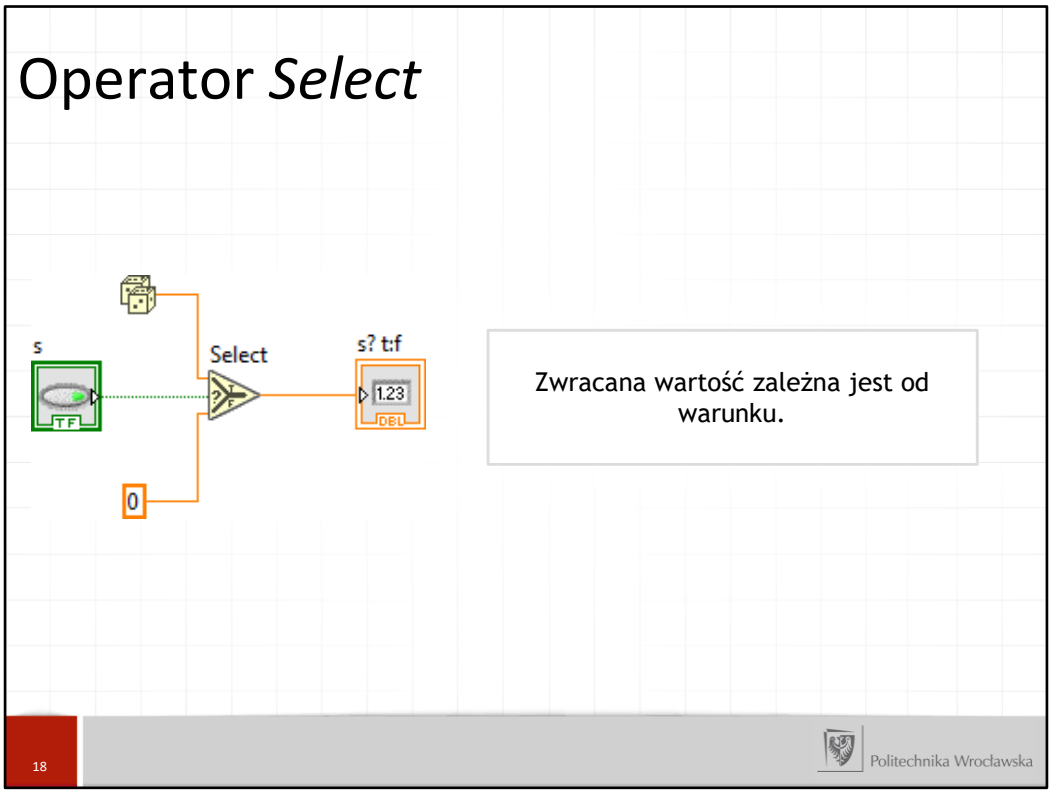


Dla niektórych typów selektorów konieczne jest określenie domyślnego poddiagramu. Można to zrobić przez kliknięcie prawym przyciskiem myszy



Jeśli w którymś z poddiagramów, tunel wyjściowy zostanie niepodłączony, to tunel taki oznaczony zostaje białym (pustym) kwadratem. W takiej sytuacji generowany jest błąd uniemożliwiający kompilację programu. Aby program zadziałał, należy wykonać jedną z czynności:

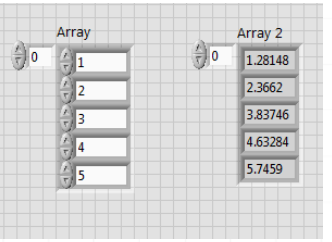
- ręcznie połączyć pusty tunel wyjściowy z kodem w odpowiednim poddiagramie;
- skorzystać z opcji *Use Default If Unwired*, by pusty tunel przyjmował wartość domyślną dla danego typu danych ;
- skorzystać z opcji *Linked Input Tunnel*, by pusty tunel został automatycznie połączony z wybranym tunelem wyjściowym.



Operator *Select* zwraca jedną z dwóch przyłączonych do niego wielkości, w zależności od wartości selektora. Obie przyłączone wielkości muszą mieć ten sam typ danych.

Zadanie 1

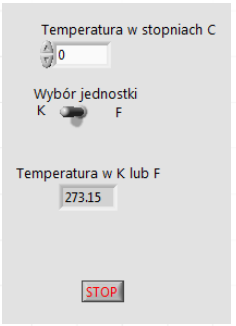
Stwórz kontrolkę składającą się z 5 elementowej tablicy liczb. Następnie wykorzystując opcję tunelowania *Auto-indexing* dodaj do każdego elementu tablicy losową liczbę z zakresu 0-1. Wynik przedstaw za pomocą indykatora.

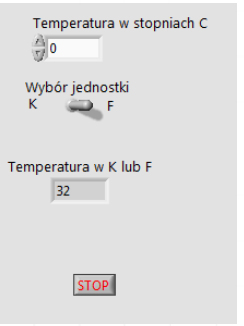


Politechnika Wrocławska

Zadanie 2

Wykorzystując strukturę warunkową *Case* napisz program, który wprowadzoną temperaturę w stopniach Celsjusza, będzie zamieniał na Kelwiny bądź stopnie Fahrenheita, w zależności od wyboru użytkownika.



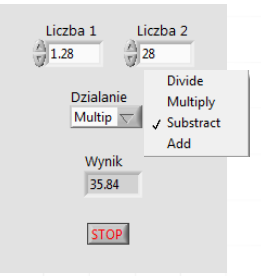


$$T\ (°F) = 1.8 \cdot T\ (°C) + 32\ (°F)$$

Zadanie 3

Wykorzystując strukturę warunkową *Case* i kontrolkę wyliczeniową napisz program wykonujący cztery podstawowe działania: dodawanie, odejmowanie, mnożenie, dzielenie.

Zadanie dodatkowe: nie dziel przez 0



Zadanie 4

Napisz program do symulowania sygnału, tak aby użytkownik mógł wybrać rodzaj generowanego sygnału (sine, square, triangle, sawtooth, DC). Wyświetl wykres z sygnałem na front panelu. Pozwól użytkownikowi na określenie parametrów sygnału.

Podpowiedź: wykorzystaj express vi simulate signal

Zadanie 5

Napisz program, który będzie obliczał wartości funkcji trygonometrycznych sin, cos, tan. Pozwól użytkownikowi zadawać wartości kąta w radianach lub stopniach.

Pamiętaj o dziedzinie funkcji tan. Podpowiedź: wykorzystaj funkcje zawarte w Mathematics->Elementary & Special Functions->Trigonometric Functions

Podpowiedź: π jest stałą w LabVIEW