|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| logoKM | logoKM | | POLITECHNIKA ŚLĄSKA  WYDZIAŁ ELEKTRYCZNY  KATEDRA MECHATRONIKI | logoKM |
| Instrukcja do ćwiczenia laboratoryjnego | | | | |
| Przedmiot: | | Przetwarzanie i Wizualizacja Danych Pomiarowych | |  |
| Symbol ćwiczenia: | | PiWDP05 | |  |
| Tytuł ćwiczenia: | | **Grupowanie danych – tablice i klastry** | | |

**SPIS TREŚCI**

|  |  |  |
| --- | --- | --- |
|  | **SPIS RYSUNKÓW** | 2 |
| 1. | **CELE ĆWICZENIA** | 3 |
| 2. | **WPROWADZENIE** | 3 |
| 2.1. | **Wektory** | 3 |
| 2.2. | **Klastry** | 5 |
| 3. | **LABORATORYJNE STANOWISKO BADAWCZE** | 6 |
| 3.1. | **Obiekt badany** | 6 |
| 3.2. | **Urządzenia dodatkowe** | 6 |
| 3.3. | **Oprogramowanie** | 6 |
| 4. | **PROGRAM ĆWICZENIA – WYKAZ ZADAŃ DO REALIZACJI** | 6 |
| 5. | **PRZYKŁAD REALIZACJI ZADANIA** | 7 |
| 6. | **RAPORT** | 8 |
| 7. | **PYTANIA** | 8 |

# SPIS RYSUNKÓW

|  |  |  |
| --- | --- | --- |
| 1. | Etapy tworzenia wektora. | 3 |
| 2. | Widok niezainicjowanej i zainicjowanej kontrolki wektora 1D. | 4 |
| 3. | Metody tworzenia wektorów. | 4 |
| 4. | Widok węzłów oraz ich opis w pomocy kontekstowych | 4 |
| 5. | Metody tworzenia klastrów. | 5 |
| 6. | Klaster z zaznaczoną numeracją elementów | 5 |
| 7. | Metody realizacji zmiany wartości wybranego elementu. | 6 |
| 8. | Przykładowa realizacja zadania obliczania histogramu – kod programu. | 7 |
| 9. | Przykładowa realizacja zadania obliczania histogramu – panel programu. | 8 |
|  |  |  |
|  |  |  |

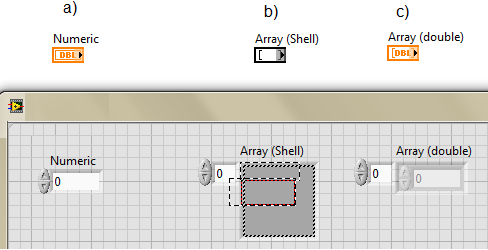
1. **CELE ĆWICZENIA**

Celem ćwiczenia jest zapoznanie się ze złożonymi strukturami danych:

* Wektory,
* Klastry.

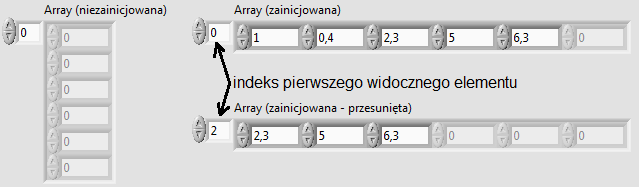
1. **WPROWADZENIE**
   1. **Wektory**

Wektory (**Array**) są złożonymi strukturami danych. Grupują dane jednego typu. Wektory składają się z wymiarów i elementów. **Elementy w wektorze są numerowane od 0!** Można tworzyć jedno i wielowymiarowe wektory. Wektory różnych typów lub rozmiarów są niekompatybilne. Wektory tworzy się w powłokach (**Shell**) kolejne kroki tworzenia wektorów są zaprezentowane na Rys. 1.



1. Etapy tworzenia wektora.

Kontrolki, wskaźniki oraz stałe typu wektorowego tworzy się w ten sam sposób. Obiekt wybranego typu przeciąga się do powłoki. W tym etapie tworzy się niezainicjowany wektor wybranego typu. Na Rys. 1 zmienna typu **double** została przeciągnięta do powłoki (**Shell**) wektora (pusta powłoka jest reprezentowana kolorem czarnym). W wyniku tej operacji powstał jednowymiarowy wektor typu double. Wektor ten jest niezainicjowany tzn. nie zawiera żadnych elementów. Zainicjowanie elementów polega na wypełnieniu elementów. Na Rys. 2 przedstawiono widok wektora niezainicjowanego oraz zainicjowanego. W przypadku wektorów 1D elementy mogą być prezentowane w postaci wektora wierszowego lub kolumnowego. W przypadku wektorów 2D kolejność wymiarów jest określona.

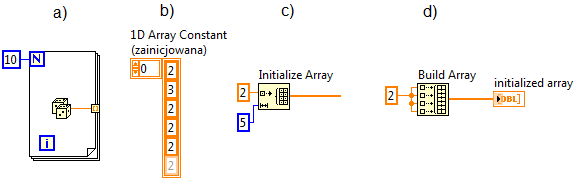


1. Widok niezainicjowanej i zainicjowanej kontrolki wektora 1D.

Pierwszym wymiarem wektora 2D jest wiersz, drugim kolumna.

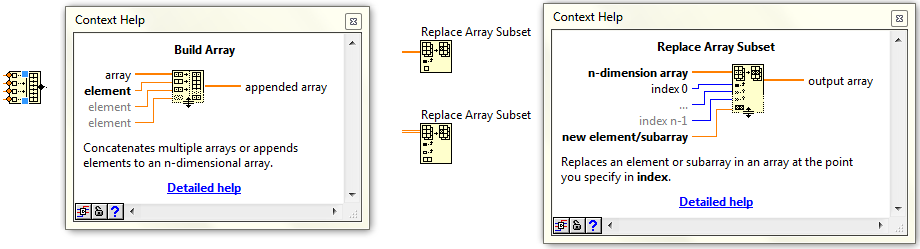
Metody tworzenia wektorów zostały przedstawione na Rys. 3:

1. Zastosowanie pętli for do tworzenia 10-cio elementowego wektora wypełnionego losowymi wartościami,
2. Stała wektor typu double,
3. Zastosowanie węzła Initialize Array do tworzenia wektora 5 elementowego wypełnionego wartościami 2,
4. Zastosowanie węzła Build Array do tworzenia czteroelementowego wektora wypełnionego wartościami 2.



1. Metody tworzenia wektorów.

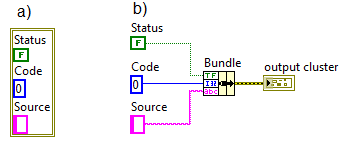
Węzły operujące na wektorach automatycznie dostosowują się do rozmiaru wektora. Ich rozmiar może być także zmieniany ręcznie – Rys. 4.



1. Widok węzłów oraz ich opis w pomocy kontekstowych
   1. **Klastry**

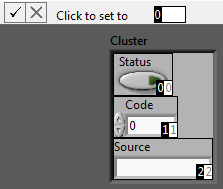
Klastry są złożonymi typami danych zawierającymi dane różnych typów. Są odpowiednikiem struktur i rekordów z tekstowych języków programowania. Wszystkie elementy klastra muszą jednak mieć ten sam „kierunek” (tzn. muszą być kontrolkami lub wskaźnikami). Podobnie jak wektory, klastry tworzy się w powłokach. Elementy klastra są ponumerowane. Dwa klastry zawierające te same elementy, jednak ułożone w różnej kolejności są różnymi klastrami. Przykładem jest klaster błędu zawierający elementy: **Status** (**Boolean**), **Kod** (**I32**), **Źródło** (**String**).

Klastry można tworzyć jako kontrolki, wskaźniki i stałe, można także grupować dane za pomocą węzłów **Bundle** oraz **Bundle by Name**. Przykłady tworzenia klastrów zostały przedstawione na Rys. 5.



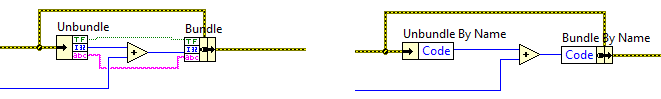
1. Metody tworzenia klastrów.

Widok klastra z zaznaczonymi numerami elementów został przedstawiony na Rys. 6. Organizację klastra można zmieniać: **PPM** 🡪 **Reorder Controls In Cluster**.



1. Klaster z zaznaczoną numeracją elementów

Węzły **Bundle** oraz **Unbundle** można stosować do wszystkich klastrów natomiast klastry **Bundle By Name** oraz **Unbundle By Name** mogą być zastosowane tylko do klastrów, których elementy posiadają nazwy. Węzły **Bundle** i **Unbundle** operują zawsze na wszystkich elementach z zachowaniem ich organizacji. Węzły **Budle By Name** oraz **Unbundle By Name** umożliwiają operowanie na wybranych elementach oraz w dowolnej kolejności. Rys. 7 przedstawia realizację zadania zmiany wartości wybranego elementu klastra za pomocą pary **Unbundle** i **Bundle** oraz **Unbundle By Name** i **Bundle By Name**.



1. Metody realizacji zmiany wartości wybranego elementu.

Zastosowanie klastrów to grupowanie danych w celu ograniczenia liczby przewodów w pliku VI oraz utrzymanie porządku w kodzie.

Można tworzyć klastry zawierające wektory oraz klastry zawierające klastry.

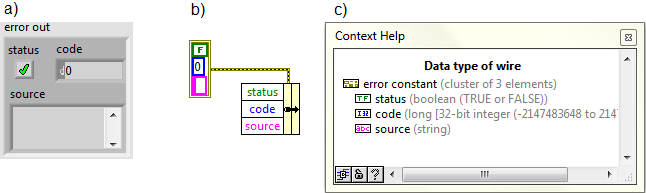
* 1. **Klaster błędu**

Klaster błędu to struktura typu Klaster zawierająca składowe:

- **status** (typu boolean): wartość **False** oznacza brak błędu,

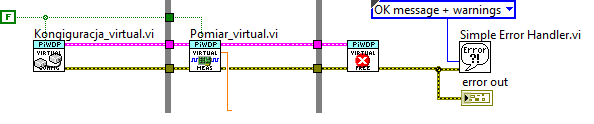
- **kod** (typu 32-integer): zawiera numer błędu, w przypadku gdy kod ma wartość niezerową a status ma wartość **True** wystąpił błąd (**error**), w przypadku gdy kod ma wartość niezerową a status ma wartość **False** wystąpiło ostrzeżenie (**warning**),

- źródło: informuje w którym subVI nastąpił błąd.



1. Wskaźnik błędu (a), stała błędu rozłożona na składowe (b), informacja o typie danych klastra błędu (c).

Klaster błędu jest preferowanym mechanizmem kontroli wykonywania programu. Stosowanie klastra błędu łączącego kolejne subVI wyłącza domyślną obsługę błędów polegającą na natychmiastowym przerwaniu działania aplikacji. W przypadku klastra błędu błąd jest propagowany przez wszystkie subVI aż do punktu, w którym program obsługuje zdarzenie. Przykład prostego programu akwizycji pracującego z prawidłowym zastosowaniem i obsługą klastra błędu przedstawiono na Rys. 6. Podczas zakończenia programu następuje obsługa błędu za pomocą subVI **Simple Error Handler.vi**.



1. Aplikacja pomiarowa, której wykonanie jest sterowane za pomocą klastra błędu.
2. **LABORATORYJNE STANOWISKO BADAWCZE**
   1. **Obiekt badany**

- Środowisko programistyczne LabVIEW,

* 1. **Urządzenia dodatkowe**

- brak,

* 1. **Oprogramowanie**
* LabVIEW 2014 lub nowszy

1. **PROGRAM ĆWICZENIA – WYKAZ ZADAŃ DO REALIZACJI**

Kolejne kroki do wykonania podczas zajęć:

* Zastosowanie klastrów:

- otworzyć projekt „*System pomiarowy DAQ*”,

- otworzyć plik „*temp konwersa.vi*”,

- przenieść powłokę klastra na pulpit,

- nazwać klaster *Temp limits*,

- w klastrze umieścić dwie kontrolki typu double o nazwach Min i Max,

- zmodyfikować kod zastępując dwie kontrolki zakresu temperatury jedną kontrolką typu klaster,

- umieścić klaster na panelu połączeń,

- zadać wartości domyślne takie same jakie miały wcześniej kontrolki,

- uzupełnić informacje w dokumentacji pliku,

- zapisać plik,

* Zastosowanie wektorów:

- z głównej pętli programu (**While**) wyprowadzić dane w postaci wektora danych (wartości zmierzone i wartości średnie),

- wprowadzić dane do następnej pętli **While**,

- napisać funkcję umożliwiającą użytkownikowi wybranie dowolnego fragmentu danych i obejrzenie ich na wykresie,

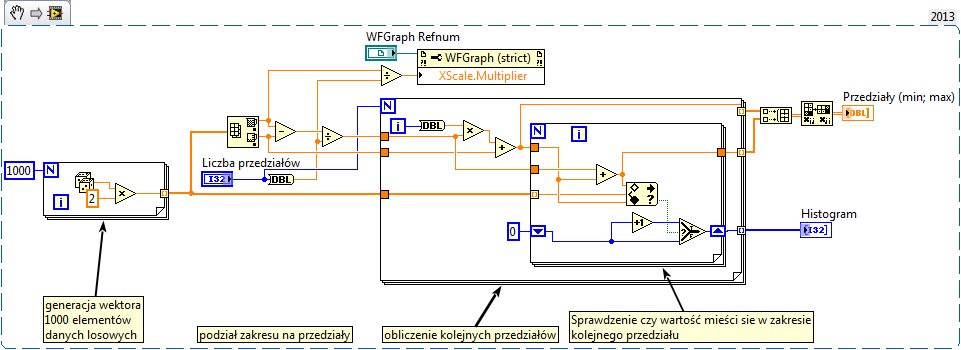
- napisać funkcję umożliwiającą użytkownikowi przeskalowania danych,

- napisać funkcję umożliwiającą użytkownikowi zmianę offsetu danych,

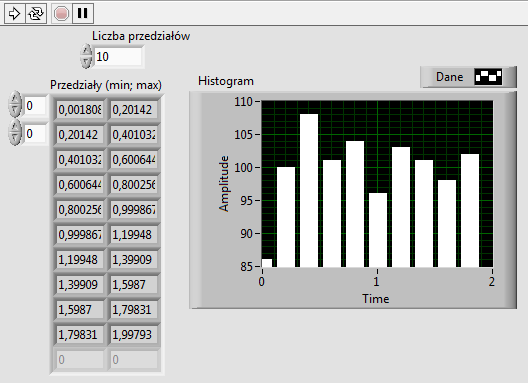
- napisać funkcję liczącą histogram danych z zadaną rozdzielczością,

1. **PRZYKŁAD REALIZACJI ZADANIA – obliczanie histogramu danych**

Przykładowa realizacja obliczania histogramu danych została przedstawiona na Rys. 8. Algorytm został podzielony na trzy części: obliczenie zakresów, przejście przez kolejne przedziały (zewnętrzna pętla **For**), sprawdzenie czy wartość mieści się w aktualnym zakresie (wewnętrzna pętla **For**). Wadą tego rozwiązania jest konieczność przeszukiwania całego zbioru danych tyle razy, ile wynosi liczba przedziałów.



1. Przykładowa realizacja zadania obliczania histogramu – kod programu.



1. Przykładowa realizacja zadania obliczania histogramu – panel programu.
2. **RAPORT**

Raport z przeprowadzonego ćwiczenia laboratoryjnego powinien zawierać opis kolejnych czynności wykonywanych w trakcie realizacji ćwiczenia, zrzuty ekranu dokumentujące wykonane kroki oraz zanotowane parametry konfiguracyjne kart DAQ (mogą być zawarte  
w tabeli)

1. **PYTANIA**
2. Wymienić podobieństwa i różnice pętli **While** oraz **FOR**.

# LITERATURA

1. LabVIEW Core 1 Course manual.
2. LabVIEW Core 1 Exercise book.
3. LabVIEW Core 2 Course manual.
4. LabVIEW Core 2 Exercise book.
5. Nota katalogowa LM35

Opracowanie: Marek Kciuk

# ZADANIA DO REALIZACJI PRZED PRZYSTĄPIENIEM DO ĆWICZENIA

- ukończyć zadania z instrukcji nr 4.

# ZADANIA DODATKOWE PO WYKONANIU ĆWICZENIA

- wyprowadzić wspólnie dane zmierzone i uśrednione z pętli akwizycji za pomocą klastra,

- zmienić funkcję obliczania histogramu z zastosowaniem węzła sortowania Sort 1D Array,

# ZAŁĄCZNIKI

## Waveform Graph vs Waveform Chart

LabVIEW