System Visual Basic dla Aplikacji w algorytmice i wizualizacji

Andrzej Burewicz
Nikodem Miranowicz

System Visual Basic dla Aplikacji w algorytmice i wizualizacji

Zakład Dydaktyki Chemii
Uniwersytet im. Adama Mickiewicza
Poznań 2005

Andrzej Burewicz, Nikodem Miranowicz

SYSTEM VISUAL BASIC DLA APLIKACJI W ALGORYTMICE I WIZUALIZACJI

Recenzent: prof.dr hab. Marek Kręglewski

Projekt okładki: dr Nikodem Miranowicz
Opracowanie komputerowe: dr Nikodem Miranowicz

ISBN: 83-89723-17-4

Druk i oprawa: Sowa–Druk na życzenie
www.sowadruk.pl, tel. 022 431-81-40

Zakład Dydaktyki Chemii
Uniwersytet im. Adama Mickiewicza
Poznań 2005

Spis treści

Wstęp

Jednym z istotnych elementów kursu Technologii Informacyjnej są zagadnienia związane z nauką podstaw algorytmiki. Wyróżnionym zadaniem jest też „zapisywanie algorytmów w postaci procedur, które może wykonać komputer - podstawowe struktury języków opisu algorytmów".

Rozpatrując ogólnie dostępne metody programowania, które można wykorzystać między innymi w kursie algorytmiki zwrócono uwagę na obecność wielu z nich w systemach operacyjnych i popularnych aplikacjach. Wyróżniają się wśród nich **AppleScript** w systemach **MacOS**, **Visual Basic dla Aplikacji** w aplikacjach pakietu **Microsoft Office** (**Windows** i **MacOS**) oraz **JavaScript** implementowany w przeglądarkach internetowych.

W rezultacie opracowano materiały metodyczne dostępne także w formie serwisu Internetowego (HTTP://ZDCH.AMU.EDU.PL/ALGORYTMY/) przeznaczone dla osób, które na praktycznych przykładach chcą nauczyć się implementacji podstawowych algorytmów oraz podstaw wizualizacji w systemie **Visual Basic dla Aplikacji**. Zasadniczym zaś zadaniem tych materiałów jest przedstawienie reguł programowania na praktycznych przykładach zadań rozważanych w kursie Technologii Informacyjnej.

Opracowany skrypt omawia przede wszystkim zasady implementacji algorytmów w postaci programów w wybranym języku programowania oraz odpowiednie gotowe ich przykłady. Istotna część materiału omawia ogólne zagadnienia algorytmiki. Ponieważ na rynku wydawniczym jest obecnych wiele podręczników w szerszym lub węższym ujęciu opisujących algorytmikę, postanowiono w tym skrypcie zaprezentować rozwiązania, które będą się skupiały bardziej na implementacji algorytmów, czyli reprezentowaniu w danym języku programowania założeń matematycznych danego algorytmu i skutecznym skompilowaniu tak uzyskanego

kodu. Zagadnienia szczegółowe dotyczące samego algorytmu opisane są w odpowiednio okrojonej wersji, ale niezbędnej dla poprawnego programowania.

Microsoft Visual Basic dla Aplikacji (**VBA**) jest językiem programowania stosowanym do rozbudowy i sterowania pakietowymi aplikacjami biurowymi Microsoft i ich integracji z dostępnymi danymi i systemami. **VBA** udostępnia szeroki zestaw narzędzi programistycznych opartych na aplikacji **Microsoft Visual Basic**. Istota stosowalności **VBA** opiera się na założeniu, że program ten umożliwia rozbudowę dostępnych aplikacji za jego pomocą i że daje on większe możliwości tworzenia bardziej efektywnych aplikacji niż budowa narzędzi od podstaw.

W **VBA** zawarte są elementy zintegrowanego środowiska programistycznego (IDE – ang. *Integrated Development Environment*) wykorzystujące składniki analogiczne do tych z systemu **Microsoft Visual Basic** - **Project Window**, **Properties Window**, **Debugging**, jak również **Microsoft Forms** umożliwiające łatwe tworzenie interfejsu użytkownika. Zintegrowanie tych narzędzi w nadrzędnej aplikacji znacznie usprawnia i przyspiesza proces tworzenie programów sterujących aplikacjami.

Książka niniejsza przeznaczona jest do wspomagania zajęć z Technologii Informacyjnej i Informatyki w szkolnictwie wyższym ze szczególnym wyróżnieniem studiów chemicznych. Zawiera zbiór zadań z podstawowym ich omówieniem. Zakłada się, że prowadzący zajęcia przedstawiać będzie uzupełniające materiały i informacje niezbędne do właściwego zrozumienia prezentowanych zagadnień. Czytelnicy, którzy poznawać będą zadania z tej książki samodzielnie powinni systematycznie uzupełniać z innych źródeł swą wiedzę teoretyczną dotyczącą zwłaszcza programowania strukturalnego i zorientowanego obiektowo, programowania w **Visual Basic** oraz zastosowań **Visual Basic dla Aplikacji**. Książka ta stanowi praktyczny wstęp do programowania w jednym z prostszych języków programowania i zastosowania go w podstawowej wizualizacji chemicznej.

Autorzy

I. Przygotowanie systemu Visual Basic dla Aplikacji

Przystępując do realizacji zadań w niniejszym zbiorze tworzyć będziemy projekty w programie **Microsoft Word** (lub **Microsoft Excel**). Moduł **Macro** tego programu umożliwi nam wprowadzenie kodu programu, zweryfikowanie jego poprawności, skompilowanie go i uruchomienie.

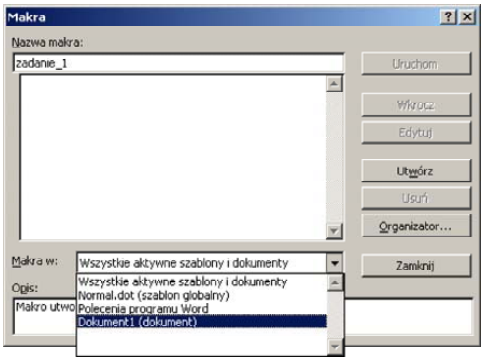
Zadania programistyczne opisane w skrypcie zostały przygotowane do realizacji na komputerach wyposażonych w program **Microsoft Word, Microsoft Excel** lub **Microsoft PowerPoint** (czyli komputerach PC (tzw. Wintel) z systemem **Windows** lub komputerach Macintosh z systemem **Mac OS 9** lub **Mac OS X**). Wykonywanie ćwiczeń wymaga zainstalowania modułu **VBA**.

Przy użyciu instalacji standardowej pakietu **Microsoft Office** instalowany jest moduł „*Visual Basic dla Aplikacji*" (Microsoft stosuje nazwę **Visual Basic dla Aplikacji** nawet w polskiej wersji swego pakietu i odpowiednich materiałach informacyjnych, wielu autorów stosuje jednak spolszczoną nieznacznie wersję tej nazwy **Visual Basic dla Aplikacji** i taką też przyjęliśmy do stosowania w niniejszej książce). Do grupy funkcji instalowanych domyślnie przy pierwszym użyciu z zakresu przydatnego w naszym kontekście należy „*Pomoc dla języka Visual Basic dla Aplikacji*", jednak jest to pomoc wyłącznie w języku angielskim, stąd jej zastosowanie może być dla wielu czytelników ograniczone.

Gdyby konieczne było zainstalowanie funkcji „*Pomoc dla języka Visual Basic dla Aplikacji*" należy w panelu sterowania kliknij dwukrotnie ikonę **Dodaj/Usuń programy**, a następnie na karcie **Instalowanie/Odinstalowanie** kliknąć pozycję **Microsoft Office**, a następnie kliknąć przycisk **Dodaj/Usuń** lub **Zmień**. W kolejnym należy kliknąć **Dodaj lub usuń funkcje**, kliknąć przycisk **Dalej**, a następnie rozwinąć na liście funkcji pozycję **Współużytkowane funkcji pakietu Office**. Rozwinąć

pozycję **Visual Basic dla Aplikacji**, a następnie kliknąć ikonę obok pozycji **Pomoc programu Visual Basic**, aby wyświetlić opcje instalacyjne. Na koniec kliknąć należy polecenie **Uruchom z mojego komputera** lub **Uruchom z sieci**, a następnie kliknąć przycisk **Aktualizuj**.

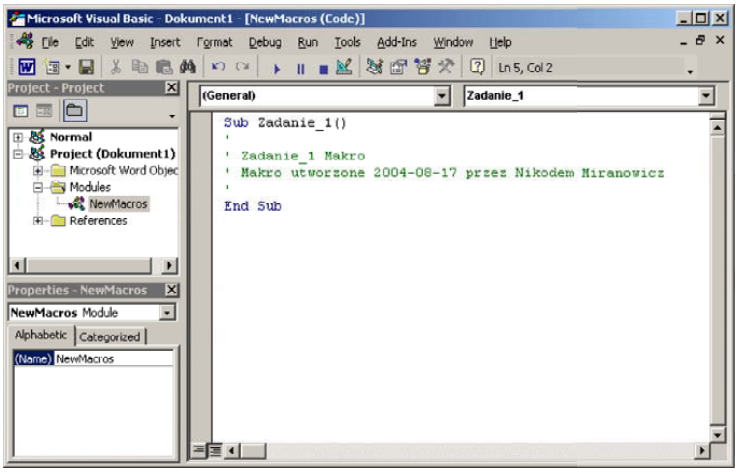
Uruchommy program **Microsoft Word**. Po uruchomieniu program przedstawia na ekranie okno dokumentu. W tym dokumencie prezentowane będą wyniki działania programów lecz sam program wpisać będzie trzeba w oknie edytora **Makro**. Należy wybrać opcję **Narzędzia/Makro/Makra...** (**Tools/Macro/Macros...)** .



Przedstawione zostanie okno wyboru lub definicji pliku makroinstrukcji. Wpisać należy nazwę procedury (np. "Zadanie_1" - uwaga, nie należy używać spacji w nazwie zamiast tego zaś kreskę dolną). Uwaga. Ważne jest, aby tworzone w ramach tych ćwiczeń makra nie zapisywać w szablonie globalnym, który jest opcją domyślną, a pliku dokumentu. Dlatego też należy wybrać opcję **Makra w: Dokument1 (dokument).** Aby utworzyć makro z tymi cechami należy wybrać przycisk **[Utwórz]** (**[Create]**).

Macro Editor wpisze tylko zasadnicze części procedury:

```
Sub zadanie_1()
'
' Zadanie_1 Makro
' Makro utworzone 2004-08-17 przez Nikodem Miranowicz
'
End Sub
```



Zadanie 1. Program wypisujący na ekranie pojedyncze zdanie.

Wypisywanie treści na ekranie jest kluczowym sposobem komunikacji komputera z użytkownikiem. Zanim nauczymy się wykorzystywać moc obliczeniową komputera należy się zapoznać z podstawowymi metodami operowania tekstem.

Realizacja zadania 1

```
Sub zadanie1()
ActiveDocument.Content = _
"Program napisany w Visual Basic " & _
"dla Aplikacji pozostawia tekst w dokumencie"
End Sub
```

Moduł **Visual Basic dla Aplikacji** przeznaczony jest do realizowania zadań na zawartości dokumentów programu macierzystego - w naszym przypadku programu **Word**. Polecenie *ActiveDocument.Content* ma na celu określenie zawartości (*Content*) aktywnego dokumentu (*ActiveDocument*) i w rezultacie spowoduje, że aktualny dokument programu **Word** zostanie wymazany a w nim zostanie przedstawiona określona treść.

Uwaga. W przedstawionym przykładzie zastosowano tzw. znak kontynuacji (składający się ze znaku spacji i znaku podkreślenia), który pozwala długie polecenia pisać w kilku wierszach.

Projekt zostanie skompilowany i uruchomiony po wybraniu opcji menu **Run**. Ponieważ wynikiem działania programu jest zmiana zawartości dokumentu **Word**, więc aby zobaczyć ten wynik należy przejść do podglądu tego dokumentu.

Zadanie 2. Program wypisujący na ekranie trzy kolejne zdania.

Na program ten składa się kilka instrukcji wyprowadzania znaków. Instrukcje tych programów wykonywane są sekwencyjnie tzn. każda instrukcja jest realizowana zgodnie z kolejnością występowania w programie (od strony lewej do prawej i z góry w dół).

Realizacja zadania 2

```
Sub zadanie2()
ActiveDocument.Content = ""
ActiveDocument.Content.InsertAfter Text:= _
"Jest taki dzień w tygodniu ... Poniedziałek" & vbLf
ActiveDocument.Content.InsertAfter Text:= _
"Jest taki dzień w tygodniu  ... Wtorek" & vbLf
ActiveDocument.Content.InsertAfter Text:= _
"Jest taki dzień w tygodniu ... Środa" & vbLf
End Sub
```

Jak to już wyżej zaznaczono polecenie "*ActiveDocument.Content* =" określa zawartość całego dokumentu zmazuje więc przy tym wszystko, co w nim było dotychczas określone. Aby dodać do dokumentu nowe treści, należy więc wybrać polecenie "*ActiveDocument.Content.InsertAfter Text:=*" umieszczające tekst (*Text*) przez wstawienie go na końcu (*InsertAfter*) aktualnej zawartości (*Content*) aktywnego dokumentu (*ActiveDocument*). Dodatkowo zaś należy zastosować specjalny określnik zakończenia linii (*LineFeed*) - znacznik nowej linii w **Visual Basic** przyjmujący postać *vbLf*.

W przykładzie tym wyjaśnienia wymaga stosowanie znaków równości. W pierwszej linii kodu zastosowano znak równości jako operator przypisania zawartości aktywnego dokumentu przypisuje się (lub podstawia się pod niego) określony za znakiem równości tekst umieszczony w cudzysłowie. W kolejnej linii wobec tego samego obiektu *ActiveDocument.Content* stosowana jest metoda *InsertAfter* stosująca się do jednego wybranego argumentu – *Text*. Konstrukcja „Text:=" jest tzw. nazywanym argumentem (ang. *named argument*) i ułatwia wybranie argumentu dla metody *InsertAfter*.

Zadanie 3. Program wypisujący na ekranie siedem kolejnych zdań odnoszących się do kolejnych dni tygodnia z wydzieleniem pierwszej części zdań w postaci zmiennej tekstowej.

Podstawowym sposobem optymalizacji programu jest wydzielanie z materiału opisywanego przez program wartości stałych i zmiennych.

Zmienną jest nazwane miejsce magazynowania danych mogące zawierać dane, które mogą być modyfikowane podczas wykonywania programu. Każda zmienna ma nazwę, która jednoznacznie identyfikuje zmienną w jej zakresie. Stałą można zdefiniować podobnie z tą jednak zasadniczą różnicą, że stałe, zgodnie zresztą z ich nazwą nie mogą być modyfikowane podczas wykonywania programu.

W przykładzie będącym rozwinięciem zadań poprzednich wydzielamy z wypisywanych tekstów powtarzający się element i włączamy go do poleceń wypisywania treści na ekranie w postaci symbolicznej. Symbolem tego fragmentu tekstu będzie "s1", lecz nazewnictwo zmiennych pozwala nam na większą kreatywność. Symbol – zmienną należy wcześniej odpowiednio zadeklarować i przypisać jej wartość tekstową, potem zaś w odpowiedni sposób dodać do wypisywania tekstów.

Przypisanie wartości zmiennej dokonuje się przez instrukcję zawierającą nazwę zmiennej, znak równości, wartość zmiennej:

```
s1 = "Jest taki dzień w tygodniu..."
```

Realizacja zadania 3a

```
Sub zadanie3()
s1 = "Jest taki dzień w tygodniu... "
ActiveDocument.Content = ""
ActiveDocument.Content.InsertAfter Text:=s1 & "Poniedziałek" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Wtorek" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Środa" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Czwartek" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Piątek" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Sobota" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Niedziela" & vbLf
End Sub
```

Zadanie 3b. Program wypisujący na ekranie siedem kolejnych zdań odnoszących się do kolejnych dni tygodnia z wydzieleniem pierwszej części zdań w postaci zmiennej tekstowej oraz alternatywną zmienną.

Realizacja zadania 3b

```
Sub zadanie3()
s1 = "Jest taki dzień w tygodniu... "
s2 = "Mój ulubiony dzień w tygodniu to ... "
ActiveDocument.Content = ""
ActiveDocument.Content.InsertAfter Text:=s1 & "Poniedziałek" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Wtorek" & vbLf
ActiveDocument.Content.InsertAfter Text:=s2 & "Środa" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Czwartek" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Piątek" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Sobota" & vbLf
ActiveDocument.Content.InsertAfter Text:=s1 & "Niedziela" & vbLf
End Sub
```

Zadanie 4. Program wypisujący na ekranie po trzy zdania odnoszące się do kolejnych dni tygodnia z wydzieleniem pierwszej części zdań w postaci zmiennej tekstowej oraz wydzieleniem części kodu dotyczącego wypisywania w postaci procedury.

Fragment kodu, który miałby się powtarzać wielokrotnie w układzie sekwencyjnym, można wydzielić do osobnej grupy poleceń mającej postać procedury lub funkcji i odwoływać się do niej w wybranych momentach. Funkcja różni się od procedury tym, że wyniki działań wykonanych przez funkcję mogą być zwracane do bloku je wywołującego przez nazwę funkcji zaś w procedurze przez parametry procedury. W naszym przypadku nie potrzebujemy aby wyodrębniona w funkcji czy procedurze część programu zwracała jakiekolwiek dane dlatego zastosujemy konstrukcję procedury wywoływanej poleceniem *Call*.

Realizacja zadania 4

```
Sub zadanie4()
ActiveDocument.Content = ""
Call Tydzien ("Poniedziałek")
Call Tydzien ("Wtorek")
Call Tydzien ("Środa")
Call Tydzien ("Czwartek")
Call Tydzien ("Piątek")
Call Tydzien ("Sobota")
Call Tydzien ("Niedziela")
End Sub

Sub Tydzien(s)
s1 = "Jest taki dzień w tygodniu... "
s3 = "Pierwsza litera nazwy tego dnia tygodnia to: "
ActiveDocument.Content.InsertAfter Text:=s1
ActiveDocument.Content.InsertAfter Text:=s & vbLf
ActiveDocument.Content.InsertAfter Text:=s3 & Mid(s, 1, 1) & vbLf
End Sub
```

Dodatkowym, nowym elementem w programie jest specyficzna konstrukcja ostatniej linii programu. Zadaniem tej linii jest wypisanie, oprócz tekstu opisu, pierwszej litery cytowanego słowa. Słowo to zawarte w zmiennej "s" jest traktowane przez program jako łańcuch znaków. Przy takim traktowaniu tekstów przez język programowania łatwe jest wydzielanie fragmentów tekstu. Wyspecjalizowana funkcja Visual Basic *Mid(t,x,y)* pozwala z łańcucha **t** wydzielić **y** znaków zaczynając od znaku **x**.

W tym miejscu warto wspomnieć o innych funkcjach tekstowych: *Left(t,y)* i *Right(t,y)*, które podobnie jak *Mid* wydzielają określoną liczbę znaków odpowiednio od lewej lub prawej strony danego łańcucha (w omawianym zadaniu proponowaną funkcję *Mid(s,1,1)* można z tym samym rezultatem zastąpić funkcją *Left(s,1)*); *Trim(t)*, *LTrim(t)* i *RTrim(t)* usuwających spacje wiodące odpowiednio z obu stron, lewej strony lub prawej strony łańcucha znaków; *String(t,x)* i *Space(x)* tworzące łańcuch składający się ze wskazanej liczby określonych znaków lub spacji; *UCase(t)* i *LCase(t)* zmieniający łańcuch znaków w majuskułę lub minuskułę; *InStr(x,t1,t2,v)* wyszukujący określony łańcuch znaków w innym łańcuchu znaków; *StrComp(t1,t2,v)* porównujący dwa łańcuchy znaków; *Len(t)* podający długość łańcucha znaków.

Zadanie 5. Program wypisujący teksty przez procedurę tydzien() zgodnie z ogólnym opisem jak w zadaniu 4, lecz wypisujący w tej samej procedurze ocenę tylko jednego dnia.

Istotnym elementem programów komputerowych jest warunkowość określonych działań. Warunkowość ta określa, że pewne instrukcje wykonywane są tylko w odpowiednio przewidzianych sytuacjach. Potocznie warunek określa słowo "jeśli", które przeniesione na instrukcje języków programowania przybiera formę "if".

Instrukcje warunkowe umożliwiają - zgodnie z nazwą - warunkowe wykonywanie instrukcji - w zależności od skuteczności wypełnienia zadanego warunku. Na przykład warunkiem może być "Jeśli [omawiany dzień] jest środą to..." . Warunek taki wyrażony w programie brzmi: *if s="Środa"*. Po instrukcji warunkowej umieszczona powinna być instrukcja, która będzie wykonywana w przypadku spełnienia zadanego warunku.

Realizacja zadania 5

```
Sub zadanie5()
ActiveDocument.Content = ""
Call Tydzien ("Poniedziałek")
Call Tydzien ("Wtorek")
Call Tydzien ("Środa")
Call Tydzien ("Czwartek")
Call Tydzien ("Piątek")
Call Tydzien ("Sobota")
Call Tydzien ("Niedziela")
End Sub

Sub Tydzien(s)
s1 = "Jest taki dzień w tygodniu... "
s2 = "To mój ulubiony dzień w tygodniu "
s3 = "Pierwsza litera nazwy tego dnia tygodnia to: "
ActiveDocument.Content.InsertAfter Text:=s1
ActiveDocument.Content.InsertAfter Text:=s & vbLf
 ActiveDocument.Content.InsertAfter Text:=s3 & Mid(s, 1, 1) & vbLf
If s = "Środa" Then
        ActiveDocument.Content.InsertAfter Text:=s2 & vbLf
End If
End Function
```

Zadanie 6. Program zgodny z ogólnym opisem jak w zadaniu 5 wypisujący teksty przez procedurę tydzien() przekazując do niej zmienną liczbową.

Kolejna modyfikacja programu uwzględnia skrócenie poleceń przez przekazywanie przez procedurę nie nazwy dnia, lecz jego kolejności w tygodniu. Możliwe jest to przez zastosowanie tablicy zmiennych, w której kolejne wartości zmiennej zapisane są w kolejnych pozycjach tablicy.

Definicja tablicy zmiennych jest rozwinięciem definicji zmiennej prostej. Słowo kluczowe *Dim* określa powstanie zmiennej tablicowej (konieczne jest określenie rozmiarów tablicy). Deklaracja tablicy zmiennych uzupełniona jest przez deklarację typu zmiennych stosowanych w tablicy. Sformułowanie *As String* deklaruje stosowanie w tablicy zmiennych typu tekstowego. Typ danych w **Visual Basic** może, ale nie musi być określony. Jeśli zostanie użyta zmienna, która nie była zadeklarowana w sposób jawny (tak jak w tym zadaniu), w języku **Visual Basic** jest ona deklarowana niejawnie

Realizacja zadania 6

```
Sub zadanie6()
ActiveDocument.Content = ""
Call Tydzien (1)
Call Tydzien (2)
Call Tydzien (3)
Call Tydzien (4)
Call Tydzien (5)
Call Tydzien (6)
Call Tydzien (7)
End Sub

Sub Tydzien(m)
Dim st(7) As String
st(1) = "Poniedziałek"
st(2) = "Wtorek"
st(3) = "Środa"
st(4) = "Czwartek"
st(5) = "Piątek"
st(6) = "Sobota"
st(7) = "Niedziela"
s1 = "Jest taki dzień w tygodniu..."
s2 = "To mój ulubiony dzień w tygodniu"
s3 = "Pierwsza litera nazwy tego dnia
tygodnia to:"
ActiveDocument.Content.InsertAfter  Text:= _
s1
ActiveDocument.Content.InsertAfter Text:= _
st(m) & vbLf
ActiveDocument.Content.InsertAfter Text:= _
s3 & Mid(st(m), 1, 1) & vbLf

If m = 3 Then
ActiveDocument.Content.InsertAfter Text:= _
s2 & vbLf
End If
End Sub
```

(tak jak w zadaniach poprzednich) jako zmienna typu *Variant***,** co nie zawsze jest korzystne.

Warto pamiętać, że możliwe jest deklarowanie zmiennych liczbowych o różnym zakresie i dokładności (*Integer*, *Long Integer*, *Single*, *Double*, Decimal, *Byte*) zmiennych tekstowych (*Fixed String*, *Variable String*) i innych zmiennych specyficznych (*Currency*, *Date*, *Object*, *Variant*).

Odwoływanie się do zmiennych zawartych w tablicy wymaga podania pozycji danej zmiennej w tablicy.

Odwoływanie się do procedury Tydzien() przez parametr liczbowy wymaga zmiany deklaracji procedury. W konsekwencji też musi ulec zmianie układ instrukcji warunkowej.

Zadanie 7. Program wypisujący w pętli odpowiednie teksty odnoszące się do kolejnych dni tygodnia.

Zestaw poleceń z głównej części programu przekazujący do procedury Tydzien() kolejne wartości liczbowe może być łatwo zoptymalizowany przez umieszczenie instrukcji wywoływania procedury w pętli, gdzie zmieniana będzie tylko wartość liczbowa parametru procedury.

Struktura pętli umożliwia wykonywanie ciągu instrukcji, aż do momentu zajścia warunku zakończenia pętli. W konstrukcji programistycznej przyjmuje ona formę *For n = 1 To 7 ... Next*, którą potocznie można opisać następująco: "dla wartości n zmieniającej się od jeden do siedem wykonaj ... następna wartość n".

Realizacja zadania 7

```
Sub zadanie7()
ActiveDocument.Content = ""
For n = 1 To 7
Call Tydzien (n)
Next
End Sub

Sub Tydzien(m)
Dim st(7) As String
st(1) = "Poniedziałek"
st(2) = "Wtorek"
st(3) = "Środa"
st(4) = "Czwartek"
st(5) = "Piątek"
st(6) = "Sobota"
st(7) = "Niedziela"
s1 = "Jest taki dzień w tygodniu..."
s2 = "To mój ulubiony dzień w tygodniu"
s3 = "Pierwsza litera nazwy tego dnia tygodnia to:"
ActiveDocument.Content.InsertAfter Text:=s1
ActiveDocument.Content.InsertAfter Text:=st(m) & vbLf
ActiveDocument.Content.InsertAfter Text:=s3 & Mid(st(m), 1, 1) & vbLf
If m = 3 Then
        ActiveDocument.Content.InsertAfter Text:=s2 & vbLf
End If
End Sub
```

Zadanie 8. Program wypisujący w pętli odpowiednie teksty odnoszące się do kolejnych dni tygodnia i warunkowo dodający dla każdego inny komentarz.

Inną formą potocznego określania warunku jest "zależnie od przypadku...". Ta warunkowa konstrukcja o innej charakterystyce niż warunek "jeśli", daje możliwość uwzględnienia większej różnorodności sytuacji warunkowych w prostszej konstrukcji programistycznej.

Realizacja zadania 8

Instrukcja warunkowa zastosowana w tym zadaniu ma ogólną postać „S*elect case (zmienna)*”, po czym następuje "*case wartość pierwszego przypadku*:”, dla którego - jeśli spełniony - instrukcje opisane są dalej, aż do wystąpienia kolejnej instrukcji "*case*", lub instrukcji "*end select*", kończącej instrukcję warunkową dla danego przypadku. Dalej może nastąpić opisu kolejnego przypadku, na końcu zaś opisywania wszystkich przypadków podać należy instrukcję "*end select*".

W zadaniu tym zastosowano również nową instrukcję - *ActiveDocument.Content.InsertParagraphAfter*, która formą i funkcjonalnością przypomina stosowaną już instrukcję *ActiveDocument.Content.InsertAfter* i jak można się domyślić jest zastępnikiem instrukcji *ActiveDocument.Content.InsertAfter Text:= vbLf*.

```
Sub zadanie8()
ActiveDocument.Content = ""
For n = 1 To 7
  Call Tydzien (n)
Next
End Sub

Sub Tydzien(m)
Dim st(7) As String
st(1) = "Poniedziałek"
st(2) = "Wtorek"
st(3) = "Środa"
st(4) = "Czwartek"
st(5) = "Piątek"
st(6) = "Sobota"
st(7) = "Niedziela"
s1 = "Jest taki dzień w tygodniu..."
s2 = "To mój ulubiony dzień w tygodniu"
s3 = "Pierwsza litera nazwy tego dnia tygodnia to:"
ActiveDocument.Content.InsertAfter Text:=s1
ActiveDocument.Content.InsertAfter Text:=st(m) & vbLf
ActiveDocument.Content.InsertAfter Text:=s3 & Mid(st(m), 1, 1) & vbLf
If m = 3 Then
  ActiveDocument.Content.InsertAfter Text:=s2 & vbLf
End If
Select Case m
Case 1:
  ActiveDocument.Content.InsertAfter Text:= _
  ("- Czy naprawdę nikt nie lubi Poniedziałków?")
Case 2:
  ActiveDocument.Content.InsertAfter Text:=("- Wtorek jest wtórny do Poniedziałku.")
Case 3:
  ActiveDocument.Content.InsertAfter Text:=("-Środa jest Środ...kiem tygodnia.")
Case 4:
  ActiveDocument.Content.InsertAfter Text:=("- Czwartek jest czwarty.")
Case 5:
  ActiveDocument.Content.InsertAfter Text:=("- No i oczywiście Piątek piąty.")
Case 6:
  ActiveDocument.Content.InsertAfter Text:=("- W Sobotę zaczyna się swoboda.")
Case 7:
  ActiveDocument.Content.InsertAfter Text:= _
  ("- Niedziela to zapowiedź nowego tygodnia.")
End Select
ActiveDocument.Content.InsertParagraphAfter
End Sub
```

Zadanie uzupełniające 8a:

Na podobieństwo zadania 8 napisz program przedstawiający za pomocą tekstów na ekranie kolejne planety układu słonecznego - ich nazwy, kolejność i wybrane cechy charakterystyczne. Informacje zbierz z Internetu, np.:

- HTTP://PORTALWIEDZY.ONET.PL/
- lub HTTP://ENCYKLOPEDIA.WP.PL/

Zadanie uzupełniające 8b:

Na podobieństwo zadania 8 napisz program przedstawiający na ekranie, za pomocą tekstów, kolejne miesiące roku - ich nazwy, kolejność i przysłowia na ich temat.

Zadanie 9. Program wykonujący podstawowe operacje matematyczne.

To zadanie i zadania dalsze opisują zasady wykorzystanie operacji matematycznych.

Omawiane języki programowania potrafią realizować podstawowe działania matematyczne, co przedstawiono w zadaniu 9.

Realizacja zadania 9

```
Sub zadanie9()
x = 3
s = "Wynik działania (dla x=3): "
ActiveDocument.Content = ""
y = x + 2
ActiveDocument.Content.InsertAfter Text:=s & " y=x+2 to " & y & vbLf
y = x - 2
ActiveDocument.Content.InsertAfter Text:=s & " y=x-2 to " & y & vbLf
y = x * 2
ActiveDocument.Content.InsertAfter Text:=s & " y=x*2 to " & y & vbLf
y = x / 2
ActiveDocument.Content.InsertAfter Text:=s & " y=x/2 to " & y & vbLf
End Sub
```

Obliczenia matematyczne wykonywane mogą być bezpośrednio na liczbach, lecz także i na zmiennych.

W powyższych przypadkach pod zdefiniowaną zmienną "y" podstawiana jest wartość zmiennej "x" poddana dodatkowej operacji matematycznej: dodawania, odejmowania, mnożenia lub dzielenia.

Zadanie 10. Program wykonujący obliczenia matematyczne z funkcjami.

Cenna jest też możliwość realizacji bardziej złożonych obliczeń matematycznych. Instrukcje potęgowania, pierwiastkowania, funkcje trygonometryczne są dostępne w każdym języku programowania.

Realizacja zadania 10

Wykorzystanie pętli programistycznej

```
Sub zadanie10()
x = 3
s = "Wynik działania (dla x=3): "
ActiveDocument.Content = ""
y = x^2
ActiveDocument.Content.InsertAfter Text:=s & " y=x^2 to " & y & vbLf
y = x^3
ActiveDocument.Content.InsertAfter Text:=s & " y=x^3 to " & y & vbLf
y = Sqr(x)
ActiveDocument.Content.InsertAfter Text:=s & " y=Sqr(x) to " & y & vbLf
y = x^0.5
ActiveDocument.Content.InsertAfter Text:=s & " y=x^0.5 to " & y & vbLf
y = Sin(x)
ActiveDocument.Content.InsertAfter Text:=s & " y=Sin(x) to " & y & vbLf
y = Cos(x)
ActiveDocument.Content.InsertAfter Text:=s & " y=Cos(x) to " & y & vbLf
End Sub
```

Konstrukcja pętli programistycznej omówiona przy okazji zadania 7. jest szczególnie użyteczna przy realizacji obliczeń matematycznych. Poniżej przedstawione są dwie ogólne formy pętli: pętla typu "for" i pętla typu "do ... while". Każdą z nich można jednak konstruować na wiele sposobów.

Zadanie 11. Program obliczania silni z 10 w pętli "for" z kresem ze względu na zmienną x.

Zadanie obliczania silni jest dobrym przykładem dla zastosowania pętli. Zmienna stosowana do sterowania pętlą może stanowić wartość składową obliczeń.

Realizacja zadania 11

```
Sub zadanie11()
'obliczanie silni z liczb w pętli for
y = 1
ActiveDocument.Content = ""
For x = 1 To 10
y = y * x
ActiveDocument.Content.InsertAfter Text:="silnia z " & x & " = " & y & vbLf
Next x
End Sub
```

Program powyższy realizuje obliczenia zamknięte w pętli 10 razy.

Zadanie 12. Program obliczania silni w pętli "while ... wend".

Inna forma pętli to pętla "*while... wend*" realizująca warunek opisany w sposób potoczny następująco: "tak długo jak (spełniany jest warunek) wykonuj".

Realizacja zadania 12

```
Sub zadanie12()
'obliczanie silni z liczb w pętli while
x = 0
y = 1
While x < 10
x = x + 1
y = y * x
Wend
ActiveDocument.Content = "Wynik obliczeń silni z " & x & " = " & y
End Sub
```

Zadanie 13. Losowanie "rzut kostką"

Losowanie przez rzut kostką daje wynik pomiędzy 1 a 6. Realizacja takiego losowania powinna się opierać na „czynniku losowym”. Czynnik taki symuluje funkcja *Rnd*. Generuje ona wartości rzeczywiste od 0 do 1 w sposób pseudolosowy (uruchomienie tej funkcji daje przypadkowe wartości lecz porównanie ich z takimi samymi wartościami losowymi na innym komputerze wykazuje „zadziwiające podobieństwo” – aby zerwać ten „przypadkowy” związek należy uruchomić jednokrotnie funkcję *Randomize*). Wygenerowane wartości (od 0 do 1) należy sprowadzić do oczekiwanych wartości (od 1 to 6) poprzez przemnożenie wyniku przez 6 i zwiększenie go o 1. W ten sposób wygenerowane będą wartości rzeczywiste od 1 do 6. Konieczne jest jeszcze sprowadzenie wygenerowanych wartości rzeczywistych do wartości całkowitych – dokonać tego można za pomocą funkcji *INT* ucinającej część rzeczywistą i pozostawiającej wartość całkowitą.

Realizacja zadania 13

```
Sub zadanie13()
' losowanie "rzut kostką"
ActiveDocument.Content = ""
Randomize
For x = 1 To 10
y = Int(Rnd * 6 + 1)
If y = 6 Then ActiveDocument.Content.InsertAfter Text:="Brawo! "
ActiveDocument.Content.InsertAfter Text:=y & vbLf
Next x
End Sub
```

Zadanie 14. Wypisywanie znaków z tablicy znaków

Zamiany wartości liczbowych generowanych np. w pętli na znaki tekstu można dokonywać przez konstrukcje warunkowa *If* lub *Case*. Jednak w określonych sytuacjach można skorzystać z tablicy znaków, która uporządkowuje wszystkie znaki alfanumeryczne właśnie w konstrukcję tablicy znaków przyporządkowując im kolejne liczby. Przez funkcję *Chr* można przywołać odpowiedni znak podając jego pozycję. Znaki alfabetu w minuskule znajdują się na pozycjach od 97 do 122 a w majuskule od 65 do 90.

Realizacja zadania 14

```
Sub zadanie14()
'wypisywanie znaków z tablicy
ActiveDocument.Content=""
For x = 97 To 122
        ActiveDocument.Content.InsertAfter text:= Chr(x)
Next x
End Sub
```

Zadania powyższe, oczywiście nie wyczerpują opisu możliwości języka programowania **Visual Basic**, a stanowią tylko wprowadzenie do jego składni. Efektywność programów w najwyższej mierze zależy od właściwego algorytmu działań stosowanego przez programistę. Stąd też dalszym etapem poznawania wybranego języka programowania powinno być poznanie podstaw algorytmiki w odpowiednich przykładach. Aby natomiast tworzyć użyteczne programy w środowisku **Windows** czy **Mac OS X** (jak i innych współczesnych systemach komputerowych), konieczne jest dokonanie ważnego kroku w poznawaniu zasad programowania i wkroczenie na terytorium programowania obiektowego.

II. Algorytmy - znajdowanie elementów

Znajdowanie w danym zbiorze elementów o określonych właściwościach polega na lokalizowaniu miejsca danego elementu w zbiorze (czyli inaczej stwierdzeniu, w jakim miejscu zbioru znajduje się określony element np. litera "A", liczba "12", największa liczba w tym zbiorze itp). Właściwości poszukiwanych elementów dotyczą wartości znaku traktowanej indywidualnie, jak i w odniesieniu do innych elementów (jak na przykład w kontekście "mniejszy", "większy" itp.). Omawiane w tym dziale będą iteracyjne i rekurencyjne algorytmy "wyszukiwania liniowego" lub inaczej "wyszukiwania sekwencyjnego".

O iteracji i rekurencji nie będziemy tu wiele pisać. Proponujemy odwołać się do odpowiednich pozycji literaturowych[1]. Jedyne, co przypomnimy to odpowiednie definicje:

- "Iteracja, metoda matematyczna polegająca na wielokrotnym kolejnym zastosowaniu tego samego algorytmu postępowania, przy czym wynik i-tej operacji stanowi dane wejściowe dla kolejnej, (i+1)-szej operacji." (Słownik języka polskiego PWN)
- "Rekurencja, rekursja (angielskie recursion), cecha algorytmu, polegająca na tym, że w którymś kroku algorytmu następuje odwołanie do całego algorytmu." (Słownik języka polskiego PWN)
- Wyszukiwanie liniowe - "Prosty (...) algorytm wyszukiwania, operujący poprzez sekwencyjne badanie każdego elementu w liście, dopóki element docelowy nie zostanie znaleziony, lub ostatni element zostanie całkowicie przeanalizowany ." (Słownik komputerowy, Microsoft Press, PLJ, Warszawa 2000)

[1] Banachowski L., Diks K., Rytter W., *Algorytmy i struktury danych,* WNT, Warszawa 2003; Sysło M., *Algorytmy*, WSiP, Warszawa 1997; Wirth N., *Algorytmy + struktury danych = programy,* WNT , Warszawa 2002; Wróblewski P., *Algorytmy, struktury danych i techniki programowania,* Helion, Warszawa 2001; i inne

Zadanie 15. Znajdowanie określonego znaku w łańcuchu

Jako pierwsze ćwiczenie z tego działu wybierzmy proste zadanie, które pozwoli nam w łatwy sposób przejść do zadań dotyczących algorytmów. W zasadzie zadanie niniejsze wymaga ułożenia pewnego algorytmu, lecz jest to algorytm prosty, a w większości współczesnych języków programowania znaleźć możemy kilka innych, bardziej bezpośrednich rozwiązań tego zadania. Oto zadanie: "Przygotować program w wybranym języku programowania realizujący w sposób algorytmiczny znajdowanie danego znaku w danym łańcuchu znaków". (Ponieważ w dalszej części materiału ćwiczenia będą zawsze dotyczyły "przygotowania odpowiedniego programu w wybranym języku programowania realizujących w sposób algorytmiczny określone zadania", skrócimy umownie treść zadania do postaci: "Znaleźć dany znak w danym łańcuchu znaków".)

Innymi słowy nasze zadanie to znaleźć literę w słowie, bądź w zestawie słów. Zadanie wydaje się proste i takim jest rzeczywiście, choć nie uchroni nas to od ułożenia odpowiedniego algorytmu działań. Ponieważ nie można oczekiwać, iż "dany łańcuch znaków" będzie w określony sposób uporządkowany, dlatego realizacja zadania będzie wymagała spojrzenia na każdy kolejny znak w łańcuchu i porównania go ze wzorcem - "danym znakiem".

- Algorytm"1. algorytm m IV, D. -u, Ms. ~mie; lm M. –y 1. inform. - dokładny przepis wykonania w określonym porządku skończonej liczby operacji, pozwalający na rozwiązanie każdego zadania danego typu 2. mat. reguła przekształcania wyrażeń matematycznych przez powtarzanie tych samych działań na kolejno otrzymywanych wynikach działań poprzednich (Słownik Języka Polskiego PWN); śrdwłc. z ar.". słowo "algorytm" wywodzi się od nazwiska matematyka arabskiego z IX w., Al-Chuwarizmiego.
- Łańcuch, "struktura danych składająca się z sekwencji znaków zwykle reprezentujących czytelny dla człowieka tekst." (Słownik komputerowy, Microsoft Press, PLJ, Warszawa 2000)

Realizacja zadania 15

Po uruchomieniu modułu **Makro** programu **Word** i utworzeniu odpowiedniego projektu zadania można wpisywać treść programu. Konieczne jest zachowanie linii definiujących procedurę i ją zamykających:

Sub zadanie15()
End Sub

Pomiędzy powyższymi oznaczeniami wprowadzimy zasadniczą część kodu reprezentującego algorytm.

Zmienne "s1", "s2" (niezbędne w naszym programie bowiem reprezentujące odpowiednio dany znak do znalezienia dany łańcuch znaków, wśród których będziemy poszukiwać) należy odpowiednio w programie zdefiniować. Stąd na początku naszego kodu znaleźć się powinny dwie następujące linie:

s1="s"
s2="abecadło z pieca spadło"

Zasadnicza część programu oparta jest na pętli. Pętla ta jest niezbędna, aby móc wybierać kolejne znaki z łańcucha i poddawać je porównaniu. Pętla ta powinna więc obejmować pozycje łańcucha od pierwszej do ostatniej. Pierwsza pozycja łańcucha jest oznaczona wartością "1", ostatnia zaś może być określona przez polecenie "Len(s2)", które podaje długość łańcucha znaków "s2". Pętla tego programu powinna mieć więc postać: *For x = 1 To Len(s2) ... Next x*

Teraz, gdy potrafimy już przejrzeć całość łańcucha znaków spróbujmy weryfikować zgodność znalezionego tam znaku ze znakiem poszukiwanym. Znak z danej pozycji "x" będzie przedstawiany przez funkcję: *Mid(s2, x, 1)*, zaś porównanie, będzie dokonane przez funkcję: *If Mid(s2, x, 1) = s1 Then*.

Wynik wyszukiwań kryjący się pod zmienną "x" można wypisać w dokumencie Word przez polecenie: *ActiveDocument.Content = x*

Znacznie lepiej będzie taka odpowiedź wyglądała jeśli obudujemy ją odpowiednimi słowami np. "znak 's' znaleziono na pozycji 18", co może być uzyskane przez polecenie: *ActiveDocument.Content ="znak '" & s1 & "' znaleziono na pozycji " & x*

Znaki "&" pozwalają łączyć teksty w cudzysłowach ze zmiennymi.

Oto całość programu:

```
Sub zadanie15()
s1 = "s"                                'dany znak do znalezienia
s2 = "abecadło z pieca spadło"          'dany łańcuch
For x = 1 To Len(s2)
If Mid(s2, x, 1) = s1 Then ActiveDocument.Content = _
" znak '" & s1 & "' znaleziono na pozycji " & x
Next x
End Sub
```

Program po wpisaniu jest gotowy do uruchomienia. Projekt zostanie skompilowany i uruchomiony po wybraniu opcji **Run/RunSub/UserForm** lub jej odpowiednika w postaci odpowiedniej ikony. Wynik działania programu widoczny będzie w dokumencie programu **Word**. Aby go zobaczyć przełącz się do oglądu dokumentu **Word**.

Zadanie 16. Znajdowanie określonej liczby w tablicy liczb

Rozwiązanie tego zadanie jest rozwinięciem zadania poprzedniego.

Algorytm rozwiązania tego zadania jest pod każdym względem identyczny z algorytmem poprzednim. Pod względem realizacji programistycznej widoczne jest natomiast kilka wyraźnych różnic. Inaczej przeszukiwana jest tablica niż łańcuch. Inaczej porównywane są liczby niż znaki/łańcuchy znaków.

Realizacja zadania 16

```
Sub zadanie16()
Dim dane(100)          'definicja tablicy liczb
s = 15                 'dana liczba do znalezienia

'wyczyszczenie zawartości dokumentu Word
ActiveDocument.Content = ""

'losowanie stu liczb
For i = 1 To 100
dane(i) = Int(Rnd * 20)
Next i

'wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
"Wylosowano sto następujących liczb:" & vbLf
For i = 1 To 100
ActiveDocument.Content.InsertAfter Text:=dane(i) & ", "
Next i

ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Liczbę " & s & " znaleziono na następujących miejscach: "

'wskazywanie pozycji liczby ze zmiennej y
For i = 1 To 100
If dane(i) = s Then ActiveDocument.Content.InsertAfter Text:=i & ", "
Next i
End Sub
```

Oto kroki realizacji tego zadania:

1. W **VBA** konieczne jest zdefiniowanie tablicy zmiennych przez polecenie *Dim()* i podanie w nawiasie rozmiaru tablicy.
2. Konieczne jest także wyczyszczenie zawartości dokumentu **Word** (polecenie: *ActiveDocument.Content* = *""*), tak aby kolejne próby działania procedury nie dodawały wyników do prób poprzednich, lecz widoczne były samodzielne.
3. Tablicę danych wypełnić można automatycznie losując liczby za pomocą funkcji *Rnd*. Ponieważ *Rnd* losuje liczby rzeczywiste z przedziału [0, 1), jeżeli potrzeba uzyskać liczby z zakresu od 0 do 19 konieczne jest pomnożenie wylosowanych liczb przez 20 i poprzez funkcję *Int* uzyskiwanie tylko ich części całkowitych.
4. Umieszczenie procedury losowania w pętli pozwoli wylosować i zapisać w tablicy kolejne liczby.
5. Druga pętla powinna pozwolić nam wypisać wylosowane liczby (możemy losowanie i wypisywanie umieścić oczywiście w jednej wspólnej pętli). Wypisywanie wartości wylosowanych uzyskać można przez polecenie *ActiveDocument.Content.InsertAfter Text:*= , które dodaje do aktywnego dokumentu (zaraz za ostatnim jego znakiem) odpowiednie teksty.
6. Trzecia pętla przeznaczona jest do ponownego przeszukania tablicy i wskazania położenia szukanej liczby.

Uwaga. W tym zadaniu i zadaniach poniższych dane do zadań są losowane za pomocą funkcji *Rnd*. Możliwe jest oczywiście wpisanie nie losowanych a ustalonych danych do tablicy na początku programu. Niektórzy użytkownicy może będą chcieli zastosować trzecie rozwiązanie, w którym dane będą podawane przez użytkownika w trakcie działania programu. Do tego zadania można wykorzystać funkcję *InputBox*, która prezentuje na ekranie okno wprowadzania danych, w którym można określić tytuł okna i zwrot zachęty w nim występujący, a także położenia okna i wartość domyślną. Oto opcjonalny fragment powyższego programu zastępujący losowanie danych wprowadzaniem ich przez użytkownika (choć wprowadzenie stu danych, jak to jest w tym przypadku, może być uciążliwe):

```
'wprowadzanie stu liczb
For i = 1 To 100
dane(i) = InputBox("Podaj wartość od 0 do 19", "wprowadzanie danych ")
Next i
```

Zadanie 17. Znajdowanie największej liczby w tablicy liczb

To zadanie definiuje poszukiwane dane zupełnie inaczej niż zadania poprzednie. Nie będziemy już poszukiwać konkretnej wartości, lecz będziemy musieli dokonać weryfikacji każdej z napotkanych liczb i wybrać tę o określonych parametrach cech. Otóż tym zadaniem jest "Znaleźć największą liczbę w danej tablicy liczb".

Sugerowany algorytm jest następujący: przeglądać liczba po liczbie w tablicy liczb i uznawać za największą te liczby, które są większe od największej, jaką dotychczas napotkaliśmy. W ten sposób po dotarciu do ostatniej liczby w tablicy powinniśmy mieć pewność, jaka liczba jest największa spośród wszystkich napotkanych.

Realizacja zadania 17

```
Sub zadanie17()
Dim dane(10)              'definicja tablicy liczb

'wyczyszczenie zawartości dokumentu Word
ActiveDocument.Content = ""

'losowanie liczb
For i = 1 To 10
dane(i) = Int(Rnd * 20)
Next i

'wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
"Wylosowano dziesięć następujących liczb:" & vbLf
For i = 1 To 10
ActiveDocument.Content.InsertAfter Text:=dane(i) & ", "
Next i
x = 0
'określenie liczby największej
For i = 1 To 10
If dane(i) > x Then x = dane(i)
Next i
ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Największą liczbą w danej tablicy jest " & x
End Sub
```

Spróbujmy przeanalizować program, aby lepiej zrozumieć algorytm poszukiwania największej liczby. Tablica "dane" zawiera 10 liczb. Przed przystąpieniem do przeglądania tych danych w pętli od pozycji "1" w tablicy do pozycji ostatniej (wyznaczonej przez wartość "10") definiujemy zmienną "x", pod którą zapamiętywać będziemy wartość największą. Załóżmy, że wylosowano liczby 6, 12, 9, 17, itp. Początkowo "x" ma wartość 0. W pierwszym kroku pętli porównujemy wartość z tablicy danych w pozycji "1" (czyli wartość "6") z wartością pamiętaną w zmiennej "x" (czyli wartością "0"), jeśli ta pierwsza jest większa, to jest to równoznaczne z tym, że jest to największa znaleziona jak dotąd w tablicy danych wartość. W drugim kroku porównujemy "x=6" z wartością w pozycji 2 z tablicy (czyli "12"). Znowu okazuje się, że napotkana wartość jest "jak na razie największa" - zapamiętujemy ją w zmiennej "x". W trzecim kroku wartość "12" pozostaje największa (bowiem wartość z pozycji 3 to "9"). I tak dalej... Na koniec pod zmienną "x" pamiętać będziemy największą napotkaną w tablicy wartość.

Zadanie 18. Znajdowanie najmniejszej liczby w tablicy liczb

Wadę przedstawionego rozwiązania poprzedniego zadania widać przy próbie analogicznego rozwiązania tego zadania.

Wadliwy w powyższym rozwiązaniu nie jest algorytm, lecz jego implementacja. Jeśli zastosujemy do tych samych danych, co powyżej ten sam program (choć zmodyfikowany do poszukiwań wartości najmniejszej) uzyskamy dziwny (?) wynik - "Najmniejszą liczbą w danej tablicy jest 0", a przecież w tablicy nie ma wartości 0! Błąd wynika z wyzerowania zmiennej x w czwartej linii programu. Najlepiej jest zmiennej x nie zerować, lecz sprowadzić do "pierwszej z brzegu" liczby w danej tablicy. Idąc zaś dalej w optymalizacji programu nie musimy naszej pętli poszukiwań rozpoczynać od, uznanego już za "chwilowo największy", elementu "dane[1]".

Realizacja zadania 18

```
Sub zadanie18()

Dim dane(10) 'definicja tablicy liczb
s = 15 'dana liczba do znalezienia

'wyczyszczenie zawartości dokumentu Word
ActiveDocument.Content = ""

'losowanie liczb
For i = 1 To 10
dane(i) = Int(Rnd * 20)
Next i

'wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
"Wylosowano dziesięć następujących liczb:" & vbLf
For i = 1 To 10
ActiveDocument.Content.InsertAfter Text:=dane(i) & ", "
Next i
x = dane(1)
'określenie liczby najmniejszej
For i = 2 To 10
If dane(i) < x Then x = dane(i)
Next i
ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Najmniejszą liczbą w danej tablicy jest " & x
End Sub
```

Zadanie 19. Znajdowanie w tablicy liczb liczby największej i liczby najmniejszej

Zadanie "Znaleźć najmniejszą i największą liczbę w danej tablicy liczb" to tylko pozornie proste rozwinięcie zadań poprzednich.

Rozwiązanie zadania znalezienia w tablicy liczb największej liczby i liczby najmniejszej faktycznie może być prostym połączeniem algorytmu poszukiwania liczby największej i algorytmu poszukiwania liczby najmniejszej, lecz tylko wtedy, gdy nie przeszkadza nam niepotrzebnie długie szukanie tych liczb. Otóż, bowiem mając N liczb w tablicy musimy dokonać N-1 porównań dla znalezienia liczby największej i N-1 porównań dla znalezienia liczby najmniejszej czyli razem (2N-2). I nie ma tu znaczenia, czy robić to będziemy osobno w dwóch pętlach, czy też obok siebie w jednej pętli. Interesujące jest natomiast pytanie, jak można znaleźć w danej tablicy liczb największą liczbę i liczbę najmniejszą wykonując mniej kroków.

Proponowane rozwiązanie polega na rozdzielaniu liczb z kolejnych par do dwóch grup - grupy liczb w danych parach większych i grupy liczb w danych parach mniejszych. Zysk, jaki w ten sposób otrzymamy, to zmniejszenie o prawie jedną trzecią liczbę porównań, jakich musimy dokonać. Algorytm ten prezentuje praktyczne zastosowanie zasady "dziel i zwyciężaj".

Realizacja zadania 19

```
Sub zadanie19()
Const z = 100
Dim dane(z)             'definicja tablicy liczb
Dim dane1(z / 2)        'definicja tablicy liczb
Dim dane2(z / 2)        'definicja tablicy liczb

'wyczyszczenie zawartości dokumentu Word
ActiveDocument.Content = ""
```

```
'losowanie liczb
For i = 1 To z
dane(i) = Int(Rnd * 20)
Next i

'wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
"Wylosowano sto następujących  liczb:" & vbLf
For i = 1 To z
ActiveDocument.Content.InsertAfter Text:=dane(i) & ", "
Next i

'rozdzielanie liczb z par (liczba pierwsza i liczba ostatnia)
While (x < z - x - 1)
' wstawinie pierwszej liczby z pary wstępnie do każdej z grup
dane2(x) = dane(x)
dane1(x) = dane(x)
' w zależności od wielkości drugiej liczby z pary
If dane(z - x - 1) < dane(x) Then
'zastępowanie nią wstawionej już liczby w grupie większych ...
dane1(x) = dane(z - x - 1)
Else
' ... lub w grupie mniejszych
dane2(x) = dane(z - x - 1)
End If
x = x + 1
Wend

' poszukiwanie największej liczby w grupie większych
y = dane2(1)
x = 2
While (x <= z / 2)
If dane2(x) > y Then y = dane2(x)
x = x + 1
Wend
ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Największą liczbą w danej tablicy jest " & y

' poszukiwanie najmniejszej liczby w grupie mniejszych
y = dane1(1)
x = 2
While (x < z / 2)
If dane1(x) <= y Then y = dane1(x)
x = x + 1
Wend
ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Najmniejszą liczbą w danej tablicy jest " & y

End Sub
```

Zadanie 20. Znajdowanie elementów w zbiorach uporządkowanych

Znajdowanie elementu o danej wartości w zbiorze uporządkowanym to definicja "wyszukiwania binarnego".

- Wyszukiwanie binarne - "Rodzaj algorytmu, który szuka obiektu o znanej nazwie według uporządkowanej listy, porównując najpierw szukany element do obiektu znajdującego się w środku listy. Następnie dzieli listę na dwie części i stwierdza, w której z nich znajduje się obiekt. Proces ten powtarzany jest aż do skutku" (Słownik komputerowy, Microsoft Press, PLJ, Warszawa 2000)

Znajdowanie danych w zbiorze nieuporządkowanym wymaga przejrzenia każdego elementu tego zbioru. Zupełnie inaczej wygląda sytuacja, gdy zbiór jest uporządkowany. W takich warunkach poszukiwanie danego elementu można znacznie przyspieszyć korzystając z informacji, jakie uzyskuje się przy każdym odczytaniu wybranego elementu zbioru. Jeśli bowiem na przykład poznamy element znajdujący się dokładnie w środku tablicy danych to, w zależności od tego, czy jest on większy czy mniejszy od oczekiwanego, dalsze poszukiwania prowadzić będziemy w pierwszej lub drugiej połowie tablicy.

Realizacja zadania 20

```
Sub zadanie20()
Const z = 100
Dim dane(z)             'definicja tablicy liczb
x = 126                 'szukana liczba
p = 1                   'początek przedziału szukania
r = z                   'koniec przedziału szukania
s = Int((p + r) / 2)    'środek przedziału szukania

'wyczyszczenie zawartości dokumentu Word
ActiveDocument.Content = ""

'losowanie liczb i wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
"Przygotowano sto następujących liczb:" & vbLf
For i = 1 To z
dane(i) = i * 3
ActiveDocument.Content.InsertAfter Text:=dane(i) & ", "
Next i

ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Szukano kolejno na pozycjach: "
While dane(s) <> x
s = Int((p + r) / 2)
ActiveDocument.Content.InsertAfter Text:=s & ", "
If dane(s) > x Then r = s - 1
If dane(s) < x Then p = s + 1
Wend
ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Znaleziono szukaną wartość na pozycji:" & s

End Sub
```

III. Algorytmy - porządkowanie elementów

Pomiędzy zadaniami dotyczącymi znajdowania elementów o określonych właściwościach, a zadaniami dotyczącymi znajdowania elementów w zbiorach uporządkowanych mamy oczywiście operację porządkowania zbiorów, czyli ułożenia zawartych w nich danych w określonym porządku. Najczęściej porządkuje się zbiory według wzrastających lub malejących wartości zawartych w nich liczb.

Porządkowanie zbioru liczb to wielce interesujące zagadnienie algorytmiki, gdyż jak się okazuje najprostsze algorytmy porządkowania nie są wcale najszybsze, i odwrotnie - algorytmy najszybsze wcale nie są takie proste.

Opisane będą w tej części najważniejsze algorytmy porządkowania zbiorów (lub inaczej sortowania). Będą to kolejno:

- Zadanie 21 - Porządkowanie zbiorów przez wybór
- Zadanie 22 - Porządkowanie zbiorów "bąbelkowe"
- Zadanie 23 - Porządkowanie zbiorów przez wstawienie
- Zadanie 24 - Porządkowanie zbiorów "szybkie"
- Zadanie 25 - Porządkowanie zbiorów przez zliczanie
- Zadanie 26 - Porządkowanie zbiorów kubełkowe

Zadanie 21. Porządkowanie zbiorów przez wybór

Pierwszy algorytm porządkowania tablicy danych przychodzi nam na myśl łatwo, gdyż jest on reprezentacją sposobu, który intuicyjnie stosuje na co dzień większość z nas. Zwykle w zbiorze liczb wyszukujemy najmniejszą z nich, przepisujemy do nowej listy, skreślamy w liście starej i spośród nie skreślonych liczb znowu wyszukujemy najmniejszą. Jest to algorytm prosty w realizacji na tym etapie ćwiczeń, gdyż opiera się w znacznej części na wyszukiwaniu wartości, co omówione zostało w zadaniu 18.

W algorytmie można ten "potoczny" sposób nieco usprawnić i uniknąć tworzenia nowej listy liczb uporządkowanych. Wystarczy, że po znalezieniu pierwszej liczby najmniejszej zamienimy ją miejscami z pierwszą liczbą w zbiorze i kolejne szukanie rozpoczniemy od pozycji drugiej.

Realizacja zadania 21.

Oto odpowiedni program:

```
Sub porzadkowanie21()
Const rozmiar = 20
Dim lista(rozmiar)

'porządkowanie przez wybór
For i = 1 To rozmiar
x = i
For j = i + 1 To rozmiar
If lista(j) < lista(x) Then x = j
Next j
y = lista(x)
lista(x) = lista(i)
lista(i) = y
Next i
End Sub
```

Mimo, iż dokonuje on opisanego uporządkowania to nie widzimy jego efektu. Potrzebne jest uzupełnienie programu o fragment wypisujący dane uporządkowane. Będzie to zwykła pętla:

```
'wypisanie uporządkowanych liczb
ActiveDocument.Content.InsertAfter Text:=vbLf & _
"Uporządkowano wylosowane liczby:" & vbLf
For i = 1 To rozmiar
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i
```

Przydatne będzie również losowanie liczb do uporządkowania:

```
'losowanie i wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:="Wylosowano następujące liczby:" _
& vbLf
For i = 1 To rozmiar
lista(i) = Int(Rnd * 50)
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i
```

Oto więc cały program:

```
Sub porzadkowanie21()
Const rozmiar = 20
Dim lista(rozmiar)
ActiveDocument.Content = ""

'losowanie i wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:="Wylosowano następujące liczby:" _
 & vbLf
For i = 1 To rozmiar
lista(i) = Int(Rnd * 50)
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i

'porzadkowanie przez wybór
For i = 1 To rozmiar
x = i
For j = i + 1 To rozmiar
If lista(j) < lista(x) Then x = j
Next j
y = lista(x)
lista(x) = lista(i)
lista(i) = y
Next i

'wypisanie uporządkowanych liczb
ActiveDocument.Content.InsertAfter Text:=vbLf & _
"Uporządkowano wylosowane liczby:" & vbLf
For i = 1 To rozmiar
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i
End Sub
```

Zadanie 22. Porządkowanie zbiorów "bąbelkowe"

Inna metoda porządkowania liczb podobna jest do sposobu, w jaki grupa uczniów ustawia się (lub raczej jest ustawiana przez nauczyciela) w kolejności wg. wzrostu. Dwóch sąsiadujących uczniów porównuje się wzrostem i jeśli nie są ustawieni we właściwej kolejności zamieniają się miejscami, po czym porównywani są kolejni uczniowie. Ten sposób wymaga wielokrotnego przeglądania całej grupy (choć zwykle wiele takich działań jest realizowane przez wiele par równocześnie).

Algorytm ten najczęściej nazywany jest algorytmem "sortowania bąbelkowego" - przez analogię do bąbelków, z których większe, wynoszone są do góry i zamieniają się miejscami z mniejszymi.

Realizacja zadania 22

Implementacja algorytmu składa się z dwóch zagnieżdżonych pętli. Pętla wewnętrzna przegląda kolejne elementy tablicy, zaś pętla zewnętrza powoduje, że pętla wewnętrzna jest wykonywana wielokrotnie.

Po pierwszym przejściu gwarantowane jest, że największy element znajduje się na końcu tablicy; po drugim przejściu drugi największy element jest na pozycji przedostatniej itd. Stąd działanie wewnętrznej pętli można ograniczać za każdym razem o jedną pozycję wcześniej - nie musimy już odwiedzać ostatnich elementów tablicy: (rozmiar - i - 1). Cykl należy powtarzać aż do pełnego posortowania tablicy.

```
Sub porzadkowanie22()
Const rozmiar = 20
Dim lista(rozmiar)
ActiveDocument.Content = ""

'losowanie i wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
"Wylosowano następujące liczby:" & vbLf
For i = 1 To rozmiar
lista(i) = Int(Rnd * 50)
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i

'porządkowanie bąbelkowe
For i = 1 To rozmiar - 1
For j = 1 To rozmiar - i - 1
If lista(j) > lista(j + 1) Then
x = lista(j)
lista(j) = lista(j + 1)
lista(j + 1) = x
End If
Next j
Next i

'wypisanie uporządkowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Uporządkowano wylosowane liczby:" & vbLf
For i = 1 To rozmiar
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i
End Sub
```

Zadanie 23. Porządkowanie zbiorów przez wstawienie

Inny algorytm zwykle stosujemy przygotowując się do rozgrywki karcianej, gdy bierzemy do ręki karty ze stołu porządkując je równocześnie. Jest to jeden z prostszych i bardziej znanych algorytmów porządkowania, a nazywany jest "porządkowaniem przez wstawienie".

Algorytm ten realizowany "w ręku" jest prosty - każda kolejna karta wstawiana jest pomiędzy dwie inne, jeśli jest większa od pierwszej i mniejsza od drugiej, gdy zaś jest najmniejsza lub największa z kart trzymanych w ręku (lub jedyna, gdy w ręku kart jeszcze nie ma), jest wstawiana odpowiednio na początek lub na koniec tego zestawu.

Gdy jednak mamy ten algorytm przetransponować na ogólną sytuację z liczbami (umieszczonymi w tablicy), musimy uwzględnić przesunięcie liczb następujących po liczbie wstawianej o jedną pozycję w prawo w tablicy. Analogia do kart musi ulec zmianie - karty, powiedzmy, układamy na stole, od lewej, tuż obok siebie. Gdy więc wstawiać będziemy kolejną kartę przesunięcie tych po prawej jest konieczne.

Realizacja zadania 23

```
Sub porzadkowanie23()
Const rozmiar = 20
Dim lista(rozmiar)
ActiveDocument.Content = ""

'losowanie i wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
"Wylosowano następujące liczby:" & vbLf
For i = 1 To rozmiar
lista(i) = Int(Rnd * 50)
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i

'porządkowanie przez wstawianie
For i = 1 To rozmiar
j = i
x = lista(i)
While (j > 1) And (lista(j - 1) > x)
lista(j) = lista(j - 1)
j = j - 1
Wend
lista(j) = x
Next i

'wypisanie uporządkowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Uporządkowano wylosowane liczby:" & vbLf
For i = 1 To rozmiar
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i
End Sub
```

Zadanie 24. Porządkowanie zbiorów "szybkie"

Porządkowanie "szybkie" jest szybkie nie tylko z nazwy. Uważa się je powszechnie za jeden z najszybszych uniwersalnych algorytmów sortowania.

Faktem jest, że jest to algorytm najszybszy jednak wyłącznie dla danych czysto losowych, a szczególnie dla dużych tablic danych. W innych sytuacjach (a sytuacje rzeczywiste odbiegają często od ideału losowości) jego sprawność nie jest tak wyraźna.

Algorytm zakłada podział tablicy danych odnośnie wybranego elementu osiowego (zazwyczaj jest to pierwszy element tablicy) na dwie części - część zawierającą elementy mniejsze i część zawierającą elementy większe. W kolejnym kroku następuje podział uzyskanych części według tego samego schematu.

1. Krok pierwszy: wybieramy "punkt odniesienia" - dowolną wartość z tablicy, względem której (na prawo i na lewo) tablica będzie porządkowana. Wybieramy ją dość przypadkowo, gdyż faktycznie na początku nie wiadomo, gdzie ona się ostatecznie znajdzie (zwykle wybierana jest pierwsza wartość i pozostaje ona pierwsza do kroku szóstego, gdy wiadomo już ile wartości w tablicy jest od niej rzeczywiście mniejszych, a ile większych).
2. Krok drugi: przeglądamy tablicę od strony lewej (zakładając, że tam w końcu powinny się znaleźć tylko wartości mniejsze od wartości w "punkcie odniesienia") - w poszukiwaniu "czarnej owcy" - wartości tu niepasującej, bowiem większej od wartości w "punkcie odniesienia" (każda wartość większa powinna być z tej strony wyrzucona na stronę przeciwną, ale poczekamy z tym, aż po stronie "większych" znajdziemy wartość do zamiany)
3. Krok trzeci - tak samo, od prawej strony, szukamy "czarnej owcy" wśród wartości większych czyli wartości, która tam nie pasuje, gdyż jest mniejsza od wartości w "punkcie odniesienia" .

4. Krok czwarty - zamieniamy dwie "czarne owce" miejscami, ustawiając je we właściwych częściach.
5. Krok piąty - to powtarzanie kroków od drugiego do czwartego, aż poszukiwania z lewej i z prawej się spotkają.
6. Krok szósty - teraz, gdy wiemy już w jakim miejscu powinna się znaleźć wartość z "punktu odniesienia" (a jest to miejsce spotkania się poszukiwań z lewej i z prawej) zamieniamy tę wartość z ostatnią wartością mniejszą do wartości w "punkcie odniesienia".

Te wszystkie kroki należy od teraz zastosować dla każdej rozdzielonej części tablicy jak i rozdzielonych ich podczęści.

Porządkowanie "szybkie" opiera się jak widać na metodzie "dziel i zwyciężaj". Jest też oczywiste, że jest to algorytm rekurencyjny. Wywołania rekurencyjne kończą się, gdy któraś z kolejnych podtablic będzie zawierała tylko jeden element.

Realizacja zadania 24

Oto odpowiednia implementacja w VBA:

```
Sub porzadkowanie24()
Const rozmiar = 20
Dim lista(rozmiar)
ActiveDocument.Content = ""

'losowanie i wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
"Wylosowano następujące liczby:" & vbLf
For i = 1 To rozmiar
lista(i) = Int(Rnd * 50)
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i

'porzadkowanie "szybkie"
a = sortowanie(lista(), 1, rozmiar)

'wypisanie uporządkowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Uporządkowano wylosowane liczby:" & vbLf
For i = 1 To rozmiar
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i
End Sub
```

```
' zasadnicza część algorytmu
Function sortowanie(lista(), x1, y2)
x2 = x1
y1 = y2
If x2 >= y1 Then
        Exit Function
Else
        If x2 = y1 - 1 Then
                If lista(x2) > lista(y1) Then
                        T = lista(x2)
                        lista(x2) = lista(y1)
                        lista(y1) = T
                End If
        Exit Function
        End If
End If

z = lista((x2 + y1) / 2)
lista((x2 + y1) / 2) = lista(y1)
lista(y1) = z

While x2 < y1
        While (lista(x2) <= z) And (x2 < y1)
                x2 = x2 + 1
        Wend
        While (z <= lista(y1) And x2 < y1)
                y1 = y1 - 1
        Wend
        If x2 < y1 Then
                T = lista(x2)
                lista(x2) = lista(y1)
                lista(y1) = T
        End If
Wend

lista(y2) = lista(y1)
lista(y1) = z

a = sortowanie(lista, x1, x2 - 1)
a = sortowanie(lista, y1 + 1, y2)
End Function
```

Zadanie 25. Porządkowanie zbiorów przez zliczanie

Jedną z sytuacji, w której algorytm "QuickSort" nie okazuje się najszybszym algorytmem porządkowania jest zadanie porządkowania tablicy danych, które mogą przybierać tylko określone i znane wartości, a najlepiej, gdy danych jest więcej niż tych wartości. Przykłady zastosowań to porządkowanie kolejnych wyników rzutu kostką do gry, wartości bajtów (od 0 do 255), dziennych temperatur z dokładnością do dziesiątych części stopnia itp. W tych przypadkach najlepiej sprawdza się algorytm "porządkowania przez zliczanie".

Algorytm opiera się na zliczaniu występowania kolejnych danych źródłowych w wynikowej tablicy danych na miejscach odpowiadających tej wartości. Na przykład pojawienie się każdej wartości 10 będzie zliczane w komórce 10 wynikowej tablicy danych. Na zliczanie elementów należy przeznaczyć tablicę o wielkości równej rozpiętości liczb z tablicy do uporządkowania. Dla rzutów kostką będzie to tablica 6-cio elementowa, dla wartości bajtów 256-cio elementowa.

Realizacja zadania 25

```
Sub porzadkowanie25()
Const rozmiar = 100
Dim lista1(100)
Dim lista2(6)
ActiveDocument.Content = ""

'losowanie i wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:="Wylosowano następujące liczby:" _
 & vbLf
For i = 1 To rozmiar
lista1(i) = Int(Rnd * 6 + 1)
ActiveDocument.Content.InsertAfter Text:=lista1(i) & ", "
Next i

'porzadkowanie przez zliczanie
For X = 1 To rozmiar
lista2(lista1(X)) = lista2(lista1(X)) + 1
Next X

'wypisanie uporzadkowanych liczb
ActiveDocument.Content.InsertAfter Text:=vbLf & _
"Uporządkowano wylosowane liczby:" & vbLf
For X = 1 To 6
For y = 1 To lista2(X)
ActiveDocument.Content.InsertAfter Text:=(X) & ", "
Next y
Next X
End Sub
```

Zadanie 26. Porządkowanie zbiorów kubełkowe

Algorytm ręcznego rozdzielania poczty realizowany w urzędach pocztowych zakłada stosowanie "wstępnego" rozdzielania/sortowania listów według określonego wskaźnika (np. elementu kodu pocztowego, nazwy ulicy lub numerów domów - w zależności od rozległości danego obszaru działania danego urzędu czy listonosza). Po takim wstępnym rozdziale następuje kolejna operacja - sortowania rozdzielonej poczty w ramach każdej grupy. Występujące na poczcie "przegródki" określane są w procedurze algorytmu "kubełkami", a cały algorytm nazwany jest algorytmem kubełkowym (nazwa przeniesiona z ang. "bucket sort").

Realizacja zadania 26

```
Sub porzadkowanie26()
Const rozmiar = 100
Dim lista(100)
Dim wiadro(100, 100)
ActiveDocument.Content = ""

'losowanie i wypisanie wylosowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
"Wylosowano następujące liczby:" & vbLf
For i = 1 To rozmiar
lista(i) = Int(Rnd * 100) + 1
ActiveDocument.Content.InsertAfter Text:=lista(i) & ", "
Next i

'porządkowanie kubełkowe

' zasadnicza część algorytmu
For x = 1 To rozmiar
p = Int(lista(x) / 10)
r = wiadro(p, 0) + 1
wiadro(p, 0) = r
wiadro(p, r) = lista(x)
Next x
For z = 0 To 10
For x = 0 To wiadro(z, 0)
For y = 1 To wiadro(z, 0) - x - 1
If (wiadro(z, y) > wiadro(z, y + 1)) Then
s = wiadro(z, y)
wiadro(z, y) = wiadro(z, y + 1)
wiadro(z, y + 1) = s
End If
Next y
Next x
Next z

'wypisanie uporzadkowanych liczb
ActiveDocument.Content.InsertAfter Text:= _
vbLf & "Uporządkowano wylosowane liczby:" & vbLf
For x = 1 To 10
For y = 1 To wiadro(x, 0)
ActiveDocument.Content.InsertAfter Text:=wiadro(x, y) & ", "
Next y
Next x
End Sub
```

IV. Visual Basic dla Aplikacji w wizualizacji

Dalsze przykłady zastosowania **Visual Basic dla Aplikacji** realizowane będą w programie **Microsoft Excel** oraz **Microsoft PowerPoint**. W ten sposób opanujemy inne możliwości operowania na danych tekstowych i liczbowych oraz poznamy podstawowe metody wizualizacji. Zasadnicza różnica w programowaniu w **Visual Basic** w aplikacjach pakietu **Office** polega na innym traktowaniu dokumentów danych aplikacji. W przypadku programu **Word** jest to zasadniczo dokument sekwencyjny zawierający znaki tekstowe i inne ustawione w szeregu – nawet, jeśli wykraczają poza takie ramy jak w tabeli i kolumnach. W programie **Excel** dane gromadzone są w arkuszach składających się z komórek ułożonych w wiersze i kolumny. Dostęp do danej komórki jest niezależny od zawartości komórek pozostałych. **PowerPoint** mający stosunkowo największe możliwości graficzne, szczególnie w grafice wektorowej zachowuje dane w polach tekstowych w ramach poszczególnych slajdów.

W kolejnych zadaniach postaramy się wykorzystać tę specyfikę wymienionych programów i przygotować przykłady wizualizacji statycznej i dynamicznej na arkuszach danych i wykresach liniowych i słupkowych.

W przypadku **PowerPoint** przedstawione będą zadania zaawansowanego sterowania obiektami graficznymi oraz bardziej rozbudowanej analizy danych źródłowych w celu opracowania wizualizacji grafów prowadzącej do obrazów modeli i diagramów organizacyjnych.

Przed przystąpieniem do dalszych zadań poświęcić należy choć odrobinę czasu zagadnieniom programowania zorientowanego obiektowo, przed którymi udało się nam przez większość dotychczasowych zadań w miarę skutecznie uciekać. Zagadnienie programowania zorientowanego obiektowo jest dość szerokim zagadnieniem i choć w **Visual Basic dla Aplikacji** nie jest traktowane w pełnym wymiarze to jest bez wątpienia kluczowym pojęciem nowoczesnego programowania.

Programowanie zorientowane obiektowo jest to metodologia tworzenia programów komputerowych, która definiuje program za pomocą „obiektów" czyli elementów łączących stan (dane) i zachowanie (metody). W obiektowym języku programowania każdy aspekt kodu opiera się na elementach środowiska – obiektach. W programie np. **Excel** obiektami są skoroszyty, arkusze, zakresy komórek i pliki zewnętrzne.

Właściwości to zmienne opisujące jakieś aspekty obiektu, do którego należą. W poniższych zadaniach odwoływać się będziemy m.in. do takich właściwości arkusza jak kolor komórki (*Color*), wysokość wiersza (*RowHeight*) i innych. W kodzie **VBA** do właściwości można odwoływać się przy użyciu notacji kropkowej tzn. nazwa obiektu, kropka, nazwa właściwości (np. *ActiveSheet.Cells(a, 1).RowHeight*). Metoda z kolei to akcja, którą obiekt „wie", jak wykonać. Dostęp do metod, podobnie jak do właściwości, można uzyskać za pośrednictwem notacji kropkowej (np. *ActiveSheet.ChartObjects("wykres").Activate*).

Proponujemy rozszerzenie tych wstępnych wiadomości o cechach programowania zorientowanego obiektowo w wybranych pozycjach literaturowych.

Zadanie 27. Wypełnianie komórek losowymi liczbami

Wyniki funkcjonowania programów w **Visual Basic** prezentowane są inaczej w aplikacji Word a inaczej w **Excel**. Obiektem podstawowym w aplikacji Word był tekst aktywnego dokumentu. Aplikacja **Excel** operuje na arkuszach (*ActiveSheet*) i komórkach (*Cells*). Dlatego też, aby umieścić wynik działań w dokumencie aplikacji **Excel** należy odnieść się do *ActiveSheet.Cells(x, y)*, gdzie **x** odnosi się do numeru wiersza dla danej komórki, a **y** do numeru kolumny.

Poniższy przykład wypełni arkusz o wymiarach 10x10 liczbami tak, aby utworzyć „tabliczkę mnożenia". Algorytm wypełniania tablicy danych jest stosunkowo prosty:

1. Do kolejnych (zastosowanie pętli) komórek w danym wierszu wpisywane są kolejne wartości – w ten sposób wypełniony zostaje jeden wiersz.
2. Powyższe działanie powtórzone jest dla kolejnych (zastosowanie drugiej – zewnętrznej - pętli)

Realizacja zadania 27a

```
Sub zadanie27a()
‘tabliczka mnożenia
For y = 1 To 10
        For x = 1 To 10
                ActiveSheet.Cells(x, y) = x*y
        Next x
Next y
End Sub
```

Zmodyfikuj przykład a tak, aby wypełnić arkusz o wymiarach 256x256 danymi o wartościach wybranych losowo z liczb całkowitych od 0 do 49.

Realizacja zadania 27b

```
Sub zadanie27b()
Randomize
For y = 1 To 256
        For x = 1 To 256
                ActiveSheet.Cells(x, y) = Int(Rnd * 50)
        Next x
Next y
End Sub
```

Zadanie 28. Wyszukiwanie największej wartości w dwuwymiarowej tablicy danych

Poszukiwanie danych w tablicy realizowane jest podobnie jak tworzenie tablicy – w dwóch zawierających się w sobie pętlach. Poniższy przykład pozwala przejrzeć tablicę danych z zadania 27 i wybrać wartość największą. Wynik wpisywany jest do wolnych komórek pod tablicą - *Cells(300, 1..3)*.

Realizacja zadania 28

```
Sub zadanie28()
Max = ActiveSheet.Cells(1, 1)
For x = 1 To 256
        For y = 1 To 256
                If ActiveSheet.Cells(y, x) >= Max Then
                        Max = ActiveSheet.Cells(y, x)
                        pozycjax = x
                        pozycjay = y
                End If
        Next y
Next x
ActiveSheet.Cells(300, 1) = Max
ActiveSheet.Cells(300, 2) = pozycjax
ActiveSheet.Cells(300, 3) = pozycjay
End Sub
```

Zadanie 29. Zliczanie wartości występujących w dwuwymiarowej tablicy danych

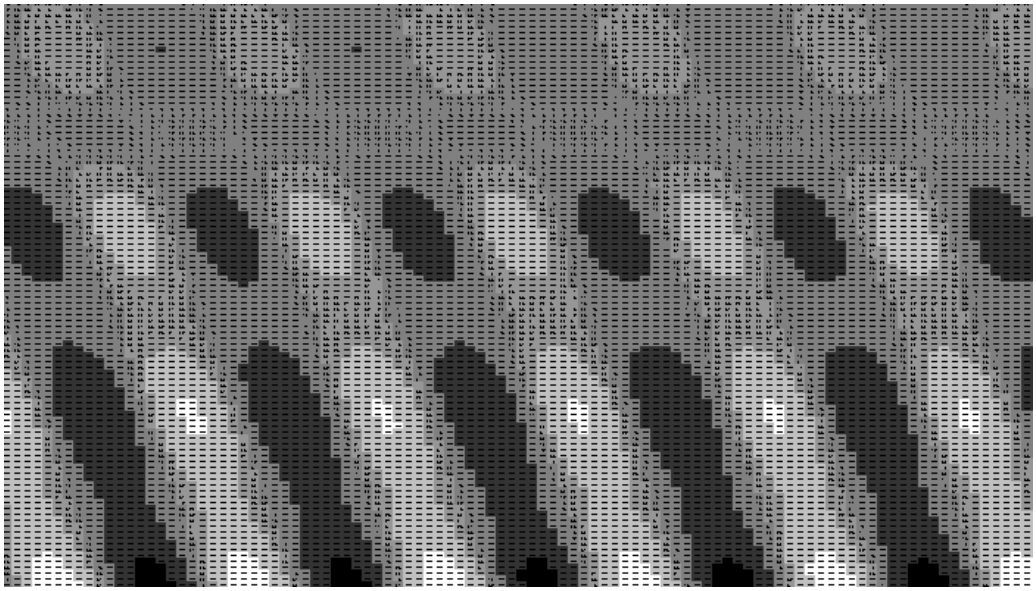
Rozwinięciem zadania 28 jest zadanie polegające na zliczaniu kolejnych wartości od 1 do 49 w utworzonej tablicy. Do powyżej stosowanych dwóch pętli dodać należy trzecią – najbardziej zewnętrzną pętlę wyliczającą kolejne zliczane wartości.

Realizacja zadania 29

```
Sub zadanie29()
For z = 1 To 49
        szukana = z
        liczba = 0
        For x = 1 To 256
                For y = 1 To 256
                        If ActiveSheet.Cells(y, x) = szukana Then liczba = liczba + 1
                Next y
        Next x
        ActiveSheet.Cells(300, z) = liczba
        ActiveSheet.Cells(301, z) = z
Next z
End Sub
```

Zadanie 30. Mapa barwna funkcji

Zadanie poniższe łączy elementy przedstawione w zadaniach poprzednich. Celem jest stworzenie mapy barwnej funkcji, której wartości wypisane są w arkuszu.



Kolejne etapy działania programu to:

1. Stworzenie tablicy danych – w sposób opisany w przykładach poprzednich.
2. Ustalanie przelicznika – znalezienie wartości maksymalnej i minimalnej w tablicy danych i na podstawie różnicy pomiędzy nimi określenie co jaką wartość nastąpić ma zmiana barwy.
3. Kolorowanie tablicy danych – przejrzenie tablicy i na podstawie określonego w poprzednim kroku przelicznika określenie jasności barwy w danej komórce. Zmiana koloru danej komórki w arkuszu jest następstwem zastosowania metody **.***Interior.Color***.**

4. Formatowanie tablicy danych – aby mapa barwna była lepiej widoczna zmieniamy szerokość każdej kolumny na 3 znaki i wysokość każdego wiersza na 12 pkt.
5. Zmiana skali oglądu arkusza – końcowy prosty krok mający na celu dosłownie zmniejszenie skali oglądu arkusza.

Realizacja zadania 30

```
Sub Mapa_barwna()
rozmiar = 100
'tworzenie tablicy danych
For a = 1 To rozmiar
  For b = 1 To rozmiar
    ActiveSheet.Cells(a, b) = 100 * ((a / 30) + Sin(a / 5)) * Sin((b - a / 5) / 3)
‘przykładowa funkcja
  Next b
Next a

'ustalanie przelicznika
Max = ActiveSheet.Cells(1, 1)
Min = ActiveSheet.Cells(1, 1)
For a = 1 To rozmiar
  For b = 1 To rozmiar
    If Max < ActiveSheet.Cells(a, b) Then Max = ActiveSheet.Cells(a, b)
    If Min > ActiveSheet.Cells(a, b) Then Min = ActiveSheet.Cells(a, b)
  Next b
Next a
Mix = 255 / (Max - Min)

'kolorowanie tablicy danych
For a = 1 To rozmiar
  For b = 1 To rozmiar
    kolor = (ActiveSheet.Cells(a, b) - Min) * Mix
    ActiveSheet.Cells(a, b).Interior.Color = RGB(kolor, kolor, kolor)
  Next b
Next a

'formatowanie tablicy danych
For a = 1 To rozmiar
   ActiveSheet.Cells(a, 1).RowHeight = 12
Next a
For b = 1 To rozmiar
   ActiveSheet.Cells(1, b).ColumnWidth = 3
Next b

‘ zmiana skali oglądu arkusza
ActiveWindow.Zoom = 25
End Sub
```

Zadanie 31. Dynamiczny wykres funkcji dwuparametrowej

Kolejny etap prac nad niestandardowymi metodami wizualizacji danych polega na próbie stworzenia programu prezentującego dynamiczny wykres.

Realizacja zadania 31

W pierwszym kroku stworzyć należy tablicę danych wartości dwuwymiarowej funkcji. Zadanie to wypełni procedura **wykres_dynamiczny:**

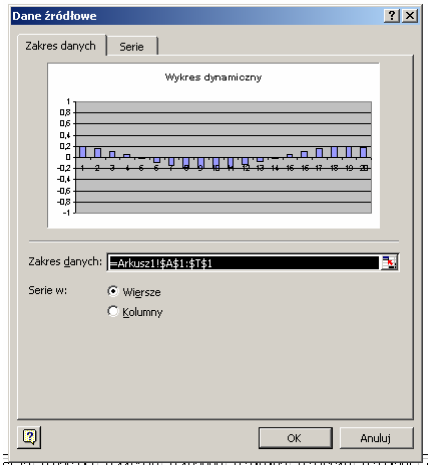
```
Sub wykres_dynamiczny()
'Tworzenie tablicy danych
For a = 1 To 31
        For b = 1 To 20
                'przykładowa funkcja
                ActiveSheet.Cells(a, b) = Sin(a / 5) * Cos(b / 3)
        Next b
Next a
End Sub
```

Druga procedura ma służyć do automatycznego budowania wykresu. W celu przygotowania tej procedury uruchomić należy funkcję rejestracji makro. Z menu **Narzędzia** wybrać należy opcję **Makro/Rejestruj nowe makro**. W tym trybie aplikacja pakietu **Office** będzie rejestrować w postaci poleceń **Visual Basic** wszystkie działania użytkownika. Na ekranie widoczny będzie niewielki pasek narzędzi – **rejestracja makro**. Aby zarejestrować polecenia tworzenia wykresu należy wykonać wszystkie działania z tym związane: zaznaczyć obszar danych, wybrać menu **Wstaw** i polecenie **Wykres**, w oknie tworzenia wykresu określić kolejno *Typ wykresu, Zakres danych, Tytuły i inne opisy* oraz *Położenie wykresu*. Na koniec w pasku narzędzi **Zatrzymaj rejestrowanie** wybrać przycisk **stop** widoczny jako niebieski kwadracik. Makro zostanie zarejestrowane pod nową nazwą, a jego treść widoczna będzie w oknie edytora **Visual Basic**.

Aby zuniwersalizować utworzoną procedurę należy doprowadzić ją poniższej postaci:

```
Sub krzywa()
ActiveSheet.Name=”dane”
Charts.Add
ActiveChart.Name=”wykres”
ActiveChart.ChartType = xlColumnClustered
ActiveChart.SetSourceData Source:= _
Sheets("Arkusz1").Range("A1:T1"), PlotBy:=xlRows
ActiveChart.Location Where:=xlLocationAsObject, Name:="dane"
With ActiveChart;
    .HasTitle = True
    .ChartTitle.Characters.Text = "Wykres dynamiczny"
    .Axes(xlCategory, xlPrimary).HasTitle = False
    .Axes(xlValue, xlPrimary).HasTitle = False
End With
ActiveChart.HasLegend = False
With ActiveChart.Axes(xlValue)
    .MinimumScale = -1
    .MaximumScale = 1
End With
End Sub
```

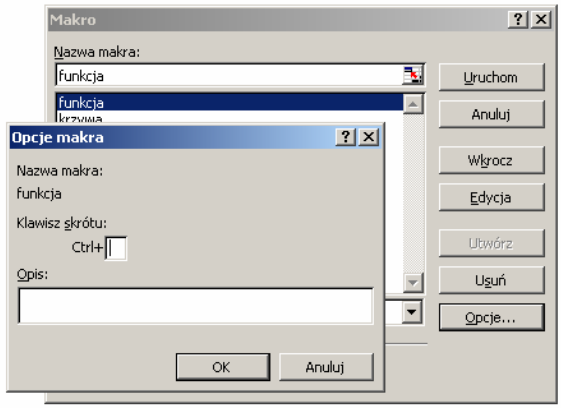
Kolejny krok to stworzenie procedury odpowiedzialnej za zmianę zakresu danych wyświetlanych na wykresie. Można tego dokonać przez modyfikację makro, w którym zarejestrowano działanie określenia zakresu danych – kliknięcie w wykres i wybranie z menu **Wykres** opcji **Dane źródłowe**. Przedstawione zostanie okno, w którym określić należy zakres danych źródłowych dla zmodyfikowanego wykresu (np. „=Arkusz1!$A$1:$T$1”)



Po modyfikacji procedura powinna wyglądać następująco:

```
Sub nastepna_linia()
‘odczytanie danych z arkusza – numer linii
l = ActiveSheet.Cells(33, 2) + 1
If l > 31 Then l = 1
ActiveSheet.Cells(33, 2) = l
‘aktualizacja wykresu
ActiveSheet.ChartObjects("wykres").Activate
ActiveChart.SetSourceData Source:= _
Sheets("dane").Range("A" & l & ":T" & l), PlotBy:=xlRows
End Sub
```

Uaktywnienie tej procedury powinno być możliwe do realizacji sprawnie – przez odpowiedni skrót klawiszowy. W tym celu należy po utworzeniu makra w oknie **Makro** wskazać jego nazwę i wybrać przycisk **Opcje...** . W oknie **Opcje makra** wybrać klawisz stanowiący podstawę skrótu klawiszowego(np. „Ctrl+f”)



Przez przekopiowanie zawartości powyższej procedury do procedury nowej pod nazwą **poprzednia_linia** i odpowiednią jej modyfikację, a także przypisanie jej odpowiedniego skrótu klawiszowego można stworzyć możliwość pokazywania wcześniejszych linii danych na wykresie.

```
Sub poprzednia_linia()
l = ActiveSheet.Cells(33, 2) - 1
If l < 1 Then l = 31
ActiveSheet.Cells(33, 2) = l
ActiveSheet.ChartObjects("wykres").Activate
ActiveChart.SetSourceData Source:= _
Sheets("dane").Range("A" & l & ":T" & l), PlotBy :=xlRows
End Sub
```

Powyższe procedury dają możliwość prezentowania kolejnych linii danych lub linii poprzednich. Prawdziwie dynamiczny wykres uzyskamy, gdy zmiany danych będą zachodziły automatycznie. Stąd zastosowanie procedury:

```
Sub kolejne_linie()
l = ActiveSheet.Cells(33, 2) + 1
If l > 31 Then l = 1
ActiveSheet.Cells(33, 2) = l

ActiveSheet.ChartObjects("wykres").Activate
ActiveChart.SetSourceData Source:= _
Sheets("dane").Range("A" & l & ":T" & l), PlotBy:=xlRows
ActiveSheet.Activate

k = ActiveSheet.Cells(34, 2)
If l > 30 Then ActiveSheet.Cells(34, 2) = k + 1
If k < 5 Then Call kolejne_linie
End Sub
```

Całość programu można usprawnić przez odpowiednią modyfikację procedury startowej:

```
Sub wykres_dynamiczny()
'Tworzenie tablicy danych
For a = 1 To 31
        For b = 1 To 20
                'przykładowa funkcja
                ActiveSheet.Cells(a, b) = Sin(a / 5) * Cos(b / 3)
        Next b
Next a
ActiveSheet.Cells(33, 2) = 1
ActiveSheet.Cells(34, 2) = 1

Call krzywa
Call kolejne_linie
End Sub
```

Zadanie 32. Wizualizacja algorytmu porządkowania

Podobne rozwiązania zastosowane w przykładach poprzednich mogą znacznie podnieść czytelność algorytmów porządkowania.

Realizacja zadania 32

Procedury porządkowania omówione wcześniej realizują porządkowanie przedstawiając wynik końcowy i dane startowe. Nie przedstawiają jednak metody porządkowania. Warto spróbować zmodyfikować te algorytmy, tak, aby uwzględniając metodę kolorowania komórek arkusza wskazywać, jakie dane są porównywane i jakie decyzje podejmowane. W tym celu uprośćmy procedurę główną i dodajmy doń legendę procedury:

```
Sub porzadkowanie()
Const rozmiar = 20
Dim lista(rozmiar)

'ustawienie szerokości kolumn
Columns("B:U").Select
Selection.ColumnWidth = 3
Columns("A:A").Select
Selection.ColumnWidth = 30

'Tworzenie opisów
i = 2
Range("a1:u6").Interior.ColorIndex = xlNone ‘usuwanie kolorowania komórek
Cells(3, 1) = "Liczba wartości:"
Cells(3, 2) = rozmiar
Cells(4, 1) = "Wartość badana:"
Cells(4, 2) = i
Cells(4, 1).Interior.Color = RGB(0, 255, 0)
Cells(5, 1) = "Wartość szukana:"
Cells(5, 2) = i + 1
Cells(5, 1).Interior.Color = RGB(255, 255, 0)
Cells(6, 1) = "Wartość mniejsza:"
Cells(6, 2) = i
Cells(6, 1).Interior.Color = RGB(0, 0, 255)

'losowanie i wypisanie wylosowanych liczb
Cells(1, 1) = "Wylosowano następujace liczby:"
For i = 1 To rozmiar
lista(i) = Int(Rnd * 50)
Cells(1, i + 1) = lista(i)
Next i
End Sub
```

Sam algorytm przeniesiony zostanie do osobnej procedury wywoływanej odpowiednim skrótem klawiszowym. Ponieważ algorytm ten został przełamany na kolejne kroki nie można zastosować klasycznej pętli – stąd liczniki są zapisywane w arkuszu danych i stamtąd za każdym krokiem odczytywane.

```
Sub Krok_wybor()
’odczyt wartości
rozmiar = Val(Cells(3, 2))
i = Val(Cells(4, 2))
j = Val(Cells(5, 2))
x = Val(Cells(6, 2))

If i > rozmiar Then Exit Sub

’kolorowanie pozycji
Range("a1:u1").Interior.ColorIndex = xlNone
Cells(1, j).Interior.Color = RGB(255, 255, 0)
Cells(1, x).Interior.Color = RGB(0, 0, 255)
Cells(1, i).Interior.Color = RGB(0, 255, 0)

’warunek porządkowania i zamiana wartości
If Cells(1, j) < Cells(1, x) Then x = j

If j > rozmiar Then
y = Cells(1, x)
Cells(1, x) = Cells(1, i)
Cells(1, i) = y
j = i
i = i + 1
x = i
End If

’zapis wartości
j = j + 1
Cells(4, 2) = i
Cells(5, 2) = j
Cells(6, 2) = x
End Sub
```

Zadanie 33. Przekształcanie krzywej miareczkowania

Zadaniem podobnym do zadania poprzedniego, jednakże o innej aplikacyjności jest procedura tworząca krzywą miareczkowania oraz licząca kolejne jej pochodne w celu precyzyjnego określenia punktu końcowego miareczkowania.

Realizacja zadania 33

Pierwsza z procedur przelicza dane w kolejnych krokach po pierwsze próbując określić rozmiar danych, oraz określić czy wartości objętości były podane w odpowiednim układzie (pozostała objętość titranta, czy objętość titranta wprowadzona do próbki). Następnie dwa kolejne kroki tworzą w arkuszu odpowiednie przeliczenia prowadzące do obliczenia pochodnych.

```
Sub krzywa_miareczkowania()

'poszukiwanie rozmiaru danych
For a = 1 To 100
        If ActiveSheet.Cells(a, 1) <> "" Then rozmiar = a
Next a

'przeliczanie objętości jeśli były podane odwrotnie
If ActiveSheet.Cells(1, 1) > ActiveSheet.Cells(2, 1) Then
         Max = -Int(-ActiveSheet.Cells(1, 1) / 10) * 10
         ActiveSheet.Cells(1, 5) = Max
         For a = 1 To rozmiar
         ActiveSheet.Cells(a, 1) = Max - ActiveSheet.Cells(a, 1)
         Next a
End If

'pierwsza pochodna
For a = 2 To rozmiar
         'obliczanie średniej objętości z sąsiednich pomiarów
         ActiveSheet.Cells(a, 3) = (ActiveSheet.Cells(a - 1, 1) + ActiveSheet.Cells(a, 1)) / 2
         'obliczanie pochodnej
         ActiveSheet.Cells(a, 4) = (ActiveSheet.Cells(a, 2) - ActiveSheet.Cells(a - 1, 2)) _
          / (ActiveSheet.Cells(a, 1) - ActiveSheet.Cells(a - 1, 1))
Next a
```

```
'druga pochodna
For a = 3 To rozmiar
        'obliczanie średniej objętości z sąsiednich pomiarów
        ActiveSheet.Cells(a, 5) = (ActiveSheet.Cells(a - 1, 3) + ActiveSheet.Cells(a, 3)) / 2
        'obliczanie pochodnej
        ActiveSheet.Cells(a, 6) = (ActiveSheet.Cells(a, 4) - ActiveSheet.Cells(a - 1, 4)) _
        / (ActiveSheet.Cells(a, 3) - ActiveSheet.Cells(a - 1, 3))
Next a
Call krzywa("A1:B" & rozmiar)
Call Dodaj_do_wykresu("C2:D" & rozmiar)
Call Dodaj_do_wykresu("E3:F" & rozmiar)
End Sub
```

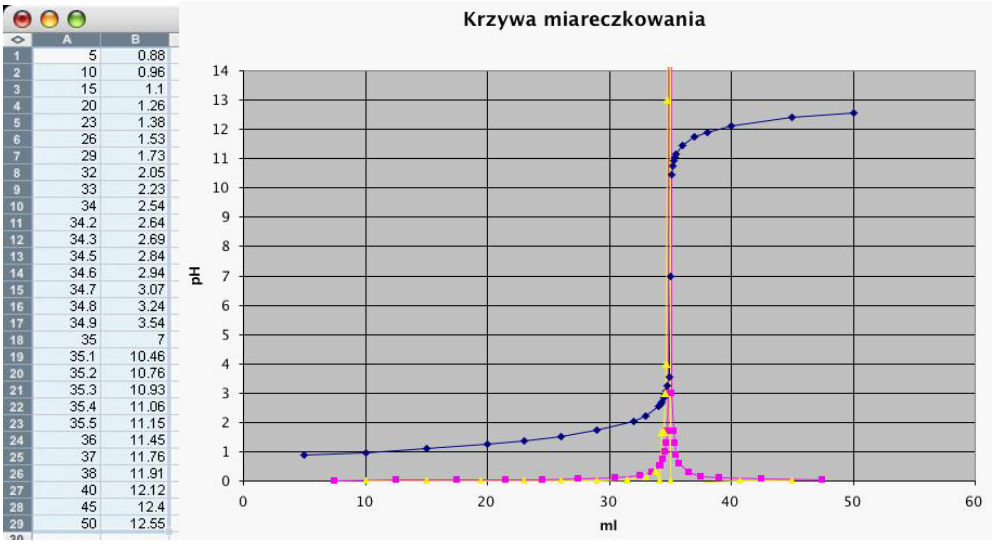
Kolejna procedura jest przekształconą rejestracją tworzenia wykresu liniowego z określonych w parametrze procedury danych.

```
Sub krzywa(zakres)
ActiveSheet.Name=”dane”
Charts.Add
ActiveChart.Name=”wykres”
ActiveChart.ChartType = xlXYScatterLines
'ActiveChart.ChartType = xlXYScatterSmooth
ActiveChart.SetSourceData Source:=Sheets("dane").Range(zakres), PlotBy:=xlColumns
ActiveChart.Location Where:=xlLocationAsObject, Name:="dane"
With ActiveChart
        .HasTitle = True
        .ChartTitle.Characters.Text = "Krzywa miareczkowania"
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "ml"
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "pH"
        .HasAxis(xlCategory, xlPrimary) = True
        .HasAxis(xlValue, xlPrimary) = True
        .Axes(xlCategory, xlPrimary).CategoryType = xlAutomatic
        .HasLegend = False
End With
With ActiveChart.Axes(xlCategory)
        .HasMajorGridlines = True
        .HasMinorGridlines = False
End With
With ActiveChart.Axes(xlValue)
        .HasMajorGridlines = True
        .HasMinorGridlines = False
End With
With ActiveChart.Axes(xlValue)
        .MinimumScale = -1
        .MaximumScale = 14
        .MinorUnitIsAuto = True
        .MajorUnit = 1
        .Crosses = xlAutomatic
        .ReversePlotOrder = False
        .ScaleType = xlLinear
        .DisplayUnit = xlNone
End With
End Sub
```

Ostatnia procedura ma zaś za zadanie dodawanie do utworzonego wykresu kolejnych dwóch wykresów: pierwszej i drugiej pochodnej.

```
Sub Dodaj_do_wykresu(zakres)
ActiveSheet.ChartObjects("wykres").Activate
ActiveChart.ChartArea.Select
ActiveChart.SeriesCollection.Add Source:=Sheets("dane").Range(zakres), _
Rowcol:=xlColumns, SeriesLabels:=False, CategoryLabels:=True, Replace:= _
False
End Sub
```

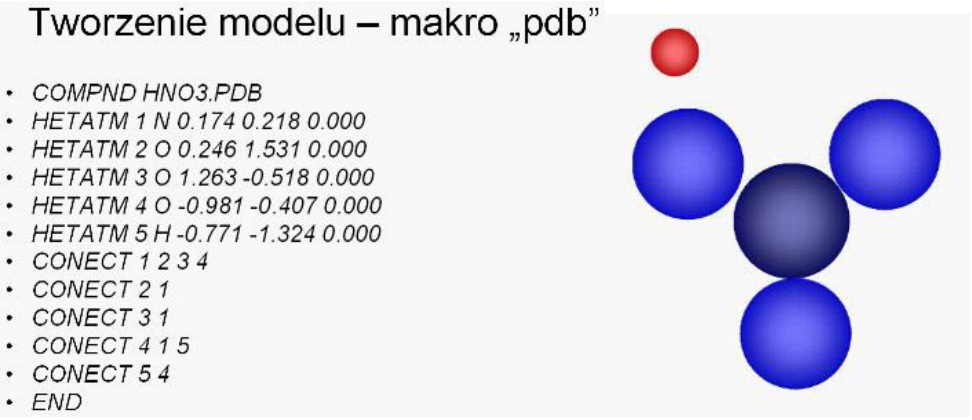
Poniżej prezentowany jest arkusz przykładowych danych oraz utworzone na jego podstawie wykresy:



Uwaga. W zadaniu tym przyjęto pewne założenia upraszczające strukturę programu. Otóż założono, że dane wprowadzane będą w pierwszych kolumnach arkusza, przy czym objętości titranta (tak pozostała objętość titranta, czy objętość titranta wprowadzona do próbki, co rozróżniane jest w pierwszym kroku) podawana jest kolumnie A, zaś pH w kolumnie B. Zmiany tych założeń wymagają odpowiednich zmian w programie.

Zadanie 34. Wizualizacja modelu PDB w PowerPoint

Ostatnie dwa zadania realizowane będą w **Microsoft PowerPoint**. Wspomagają one wizualizację w charakterystyczny sposób. Oba uczą jak odczytywać dane tekstowe, poddawać je analizie i na jej podstawie budować obraz je reprezentujący. W pierwszym z nich odczytywane będą dane z pliku PDB (Protein Data Bank) opisującego model cząsteczki związku chemicznego umieszczone polu treści danego slajdu. Na tej podstawie budowana będzie obrazowa reprezentacja tego modelu. Konieczne będzie odczytanie umieszczonych w polu treści slajdu danych opisujących typ atomu oraz trzy współrzędne przestrzenne jego lokalizacji. Wykorzystany będzie moduł rysowania zawarty w pakiecie Office w celu rysowania okręgów wypełnionych gradientem środkowym tworzących obraz kul.



Uwaga. W zadaniu tym przyjęto pewne założenia upraszczające strukturę programu. Dane PDB, jak to zaznaczono, umieszczone są w polu treści slajdu, które ma ustaloną nazwę *Shapes("Rectangle 3")*. Zmiana tego założenia wymaga odpowiednich zmian w programie.

Realizacja zadania 34

Pierwsza procedura (kulka()) jest to procedura utworzona przez rejestrację makro i odpowiednie przekształcenie mająca za zadanie budowanie obrazu pojedynczej kulki o określonym położeniu **x, y**, promieniu **r** i składowych kolorach **cr, cg, cb** oraz nazwie **n.**

```
Sub kulka(x, y, r, cr, cg, cb, n)
c1 = 115
c2 = 70
ActiveWindow.Selection.SlideRange.Shapes._
        AddShape(msoShapeOval, x, y, r, r).Name = n
ActiveWindow.Selection.SlideRange.Shapes(n).Select
With ActiveWindow.Selection.ShapeRange
        .Line.Visible = msoFalse
        .Fill.ForeColor.RGB = RGB(c1 + cr * c2, c1 + cg * c2, c1 + cb * c2)
        .Fill.BackColor.RGB = RGB(cr * 100, cg * 100, cb * 100)
        .Fill.Transparency = 0#
        .Fill.TwoColorGradient msoGradientFromCenter, 1
End With
End Sub
```

Procedura będzie wywołana w następujący sposób:

```
Sub przyklad()
Call kulka(100, 100, 500, 1, 0, 0,”kulka1”)
End Sub
```

Ciekawy efekt da też zadanie próbne:

```
Sub kilka_kulek()
For a = 1 To 32
        x = Sin(a * 3.14 / 32 * 2)
        y = Cos(a * 3.14 / 32 * 2)
        Call kulka(200 + x * 100, 200 + y * 100, 50, 1, 0, 0, ”k” & str(a))
Next a
End Sub
```

Procedura główna odczytująca dane PDB i budująca model jest następująca:

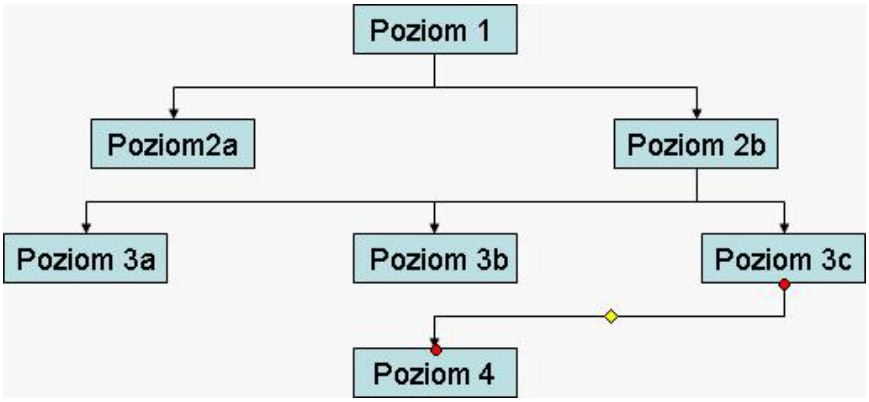
```
Sub model_pdb()
Dim m1(20) 'symbol
Dim m2(20) 'rozmiar
Dim m3(20) 'promien
Dim m4(20) 'x
Dim m5(20) 'y
Dim m6(20) 'z
Dim m7(20) 'kolejnosc
Dim grup(20)

'czytanie tekstu z pola treści slajdu
ActiveWindow.Selection.SlideRange.Shapes("Rectangle 3").Select
t = ActiveWindow.Selection.ShapeRange.TextFrame.TextRange.Text
skala = 100
srodek_x = 300
srodek_y = 200
i0 = 0
obiekt = 0
For i = 1 To Len(t)
If Mid(t, i, 1) = vbLf Then
tl = Mid(t, i0 + 1, i - i0)
i0 = i
If Mid(tl, 1, 6) = "HETATM" Then
p1 = InStr(1, tl, " ")
p2 = InStr(p1 + 1, tl, " ")
p3 = InStr(p2 + 1, tl, " ")
p4 = InStr(p3 + 1, tl, " ")
p5 = InStr(p4 + 1, tl, " ")
t1 = Mid(tl, 1, p1)
t2 = Mid(tl, p1, p2 - p1)
t3 = Trim(Mid(tl, p2, p3 - p2))
t4 = Mid(tl, p3, p4 - p3)
t5 = Mid(tl, p4, p5 - p4)
t6 = Mid(tl, p5, Len(tl) - p5)
tt = 100

Select Case t3
Case "N": pro = 0.76: c1 = 0: c2 = 0: c3 = 1
Case "O": pro = 0.73: c1 = 0: c2 = 0: c3 = 2
Case "C": pro = 0.77: c1 = 1: c2 = 1: c3 = 1
Case "H": pro = 0.32: c1 = 2: c2 = 0: c3 = 0
Case "S": pro = 1.02: c1 = 2: c2 = 2: c3 = 0
Case "Cl": pro = 0.99: c1 = 0: c2 = 0: c3 = 0
Case "Na": pro = 1.2: c1 = 1: c2 = 1: c3 = 1
Case "Mg": pro = 1.36: c1 = 1: c2 = 1: c3 = 1
Case "P": pro = 1.09: c1 = 1: c2 = 1: c3 = 0
End Select
obiekt = obiekt + 1
Call kulka(srodek_x + Val(t4) * skala, srodek_y + Val(t5) * skala, 175 * pro, c1, c2, c3, _
"k" & Str(obiekt))
grup(obiekt) = "k" & Str(obiekt)
End If
End If
Next i
ActiveWindow.Selection.SlideRange.Shapes.Range(grup).Select
ActiveWindow.Selection.ShapeRange.Group.Select
End Sub
```

Zadanie 35. Budowa diagramu organizacyjnego na podstawie zapisu umownego w PowerPoint

Jako że model strukturalny cząsteczki związku chemicznego jest rodzajem grafu, zadanie niniejszej jest w swej naturze analogiczne do poprzedniego, gdy w jego wyniku tworzony jest również graf – diagram organizacyjny. Zadanie to jest o tyle trudniejsze, że dane stanowiące punkt wyjściowy muszą zawierać informacje strukturalizowane, a także, że w wyniku uzyskać mamy zbiór obiektów – prostokątów z tekstem powiązanych łącznikami.



Podobnie jak w zadaniu poprzednim dane źródłowe zawarte będą w polu treści slajdu i opisywać będą strukturę w przedstawiony na przykładzie sposób:

Tworzenie grafu – makro „poziom”

- Poziom 1 (Poziom2a, Poziom 2b (Poziom 3a, Poziom 3b, Poziom 3c (Poziom 4)))

Uwaga. W zadaniu tym, podobnie jak w zadaniach poprzednich, przyjęto pewne założenia upraszczające strukturę programu. Pole treści slajdu, w którym umieszczone są dane źródłowe, zawsze ma w prezentacji ustaloną nazwę – *Shapes("Rectangle 3")*. Zmiana tego założenia wymaga odpowiednich zmian w programie.

Realizacja zadania 35

Procedura ma wiele etapów:

```
Sub poziomy()
Dim t1(50)
Dim t2(50)
Dim t3(50)

'czytanie tekstu z pola treści slajdu
ActiveWindow.Selection.SlideRange.Shapes("Rectangle 3").Select
t = ActiveWindow.Selection.ShapeRange.TextFrame.TextRange.Text

'analiza poziomów i tekstów
poziom = 1
obiekt = 1
nadrzedny = 0
tt = ""
For i = 1 To Len(t)
        ttt = tt
        tt = tt & Mid(t, i, 1)
        If Mid(t, i, 1) = "(" Then
                If Trim(ttt) > "" Then
                        t1(obiekt) = ttt
                        t2(obiekt) = poziom
                        t3(obiekt) = nadrzedny
                End If
                poziom = poziom + 1
                nadrzedny = obiekt
                obiekt = obiekt + 1
                tt = ""
        End If
        If Mid(t, i, 1) = ")" Then
                If Trim(ttt) > "" Then
                        t1(obiekt) = ttt
                        t2(obiekt) = poziom
                        t3(obiekt) = nadrzedny
                End If
                poziom = poziom - 1
                obiekt = obiekt + 1
                tt = ""
        End If
        If Mid(t, i, 1) = "," Then
                If Trim(ttt) > "" Then
                        t1(obiekt) = ttt
                        t2(obiekt) = poziom
                        t3(obiekt) = nadrzedny
                End If
                obiekt = obiekt + 1
                tt = ""
        End If
Next i
```

```
'liczenie poziomów
liczba_poziomow = 1
For i = 1 To obiekt - 1
        If liczba_poziomow < t2(i) Then liczba_poziomow = t2(i)
Next i

For j = 1 To liczba_poziomow
'liczenie obiektow na poziomie
        liczba_obiektow = 0
        For i = 1 To obiekt - 1
                If j = t2(i) Then liczba_obiektow = liczba_obiektow + 1
        Next i
        'przeliczanie pozycji obiektow na poziomie
        xx = (640) / liczba_obiektow
        obiekt_na_poziomie = 1
        'tworzenie i rozmieszczanie obiektow
        For i = 1 To obiekt - 1
                If j = t2(i) Then
                        Call pole(xx * obiekt_na_poziomie - xx / 2, _
                        200 + 70 * j, 100, 30, t1(i), "s" & Str(i))
                        obiekt_na_poziomie = obiekt_na_poziomie + 1
                End If
        Next i
Next j

'rysowanie lacznikow
For i = 1 To obiekt - 1
        If t3(i) > 0 Then Call lacznik("s" & Str(t3(i)), "s" & Str(i))
Next i
End Sub
```

```
Sub pole(x1, y1, x2, y2, t, n)
        ActiveWindow.Selection.SlideRange.Shapes.AddShape_
                (msoShapeRectangle, x1, y1, x2, y2).Name = n
        ActiveWindow.Selection.SlideRange.Shapes(n).Select
        ActiveWindow.Selection.ShapeRange.TextFrame.TextRange.Characters_
                (Start:=1, Length:=0).Select
        ActiveWindow.Selection.TextRange.Text = t
End Sub
```

```
Sub lacznik(a, b)
ActiveWindow.Selection.SlideRange.Shapes.AddConnector_
  (msoConnectorElbow, 153.12, 332.38, 59.5, 62.38).Select
With ActiveWindow.Selection.ShapeRange
  .Line.EndArrowheadStyle = msoArrowheadTriangle
  .Flip msoFlipHorizontal
  .Flip msoFlipVertical
  .ConnectorFormat.BeginConnect ActiveWindow.Selection.SlideRange.Shapes(a), 3
  .ConnectorFormat.EndConnect ActiveWindow.Selection.SlideRange.Shapes(b), 1
End With
End Sub
```

V. Materiały uzupełniające

W Internecie znaleźć można kilka ciekawych stron omawiających algorytmy porządkowania. Spośród nich najważniejsza w kontekście tego materiału jest strona: **HTTP://WWW.CS.UBC.CA/SPIDER/HARRISON/JAVA/INDEX.HTML** administrowana przez Jasona Harrisona. Prezentuje ona piętnaście najważniejszych algorytmów sortowania wraz z ich implementacjami w Javie oraz, co najciekawsze, apletami prezentującymi w sposób graficzny ich działanie, metodę i szybkość.

Bibliografia:

1. Burewicz A., Miranowicz N., *Implementacje klasycznych algorytmów. Nauczycielskie materiały na Mac OS i Windows*, XX Informatyka w Szkole, Wrocław 2004
2. Banachowski L., Diks K., Rytter W., *Algorytmy i struktury danych,* WNT, Warszawa 2003
3. Sysło M., *Algorytmy*, WSiP, Warszawa 1997
4. Wirth N., *Algorytmy + struktury danych = programy,* WNT , Warszawa 2002
5. Wróblewski P., *Algorytmy, struktury danych i techniki programowania,* Helion, Warszawa 2001
6. Aho A., Hopcroft J., Ullman J., *algorytmy i struktury danych*, Helion, 2003
7. Lewandowski M., *Tworzenie makr w VBA dla Excela 2002/xp pl. Ćwiczenia zaawansowane*, Helion, 2003
8. Jenach M., *Visual Basic w Accessie od podstaw*, Translator S.C., 2004
9. Langford-Wood N., Salter B., *Prosto do celu: Visual Basic. Ćwiczenia*, Read Me, 2003
10. Lewandowski M., *VBA dla Excela 2002/2003. Leksykon Kieszonkowy*, Helion, 2004
11. Walkenbach J., *EXCEL 2003 PL. programowanie w VBA. Vademecum profesjonalisty + CD-ROM*, Helion, 2004
12. Brown Ch., *ACCESS. Programowanie w VBA*, Helion, 2005