Wstęp

Oferowane programy użytkowe przeznaczone są dla wielu odbiorców, mają zatem charakter ogólny lub odpowiadają potrzebom potencjalnie najliczniejszej grupy użytkowników, często nie zaspokajając potrzeb bardziej wyspecjalizowanych użytkowników. Nawet dla specjalistów napisanie dużego programu od podstaw byłoby bardzo pracochłonne. Optymalnym rozwiązaniem jest zatem tworzenie przez użytkownika własnych programów i dołączanie ich do istniejącego pakietu oprogramowania, co umożliwiają języki programowania tzw. średniego i wysokiego poziomu. W językach tych program jest kodowany przez użytkownika, zgodnie z zasadami jego składni, co więcej „komputer" sam może kodować program „śledząc" operacje wykonywane przez użytkownika. Tak przygotowany kod źródłowy jest przetwarzany na program w kodzie maszynowym przez kompilator danego języka. Pisanie, kompilowanie i sprawdzanie działania programów wykonuje się pod kontrolą odpowiedniego pakietu programów tworzących środowisko programistyczne – arkusz kalkulacyjny Excel jest wyposażony w środowisko umożliwiające programowanie w języku Visual Basic dla Aplikacji (VBA).

VBA umożliwia zatem zautomatyzowanie często używanych procedur i wykonywanych procesów, tworzenie rozwiązań dostosowanych do potrzeb użytkownika oraz implementowanie aplikacji korzystających z Excela jako środowiska pracy. Jest to język służący do automatyzacji aplikacji firmy Microsoft. Jest on modyfikacją zorientowanego obiektowo języka Visual Basic (VB), więc struktura tych języków jest bardzo podobna. Obiektami w VBA są elementy, z których jest zbudowany skoroszyt Excela .

VBA powstał na początku lat 90-tych, do tego czasu automatyzacja aplikacji była naprawdę trudna, gdyż dla każdej aplikacji, którą chciano zautomatyzować należało nauczyć się innego języka. Aby zautomatyzować Excela, należało znać język jego makr; w przypadku programu Microsoft Word, trzeba było poznać WordBasic itp. Rozwiązanie tego problemu daje właśnie VBA, którego korzyści to:

- dostosowanie do własnych potrzeb środowiska pracy Excela, pasków narzędzi, menu oraz formularzy,
- zautomatyzowanie powtarzających się zadań,
- uproszczenie korzystania z szablonów,
- tworzenie raportów,
- wykonywać odpowiednie operacje na danych oraz poddawać je analizom.

Makra

Rejestrowanie makra

Podczas przetwarzania danych i wykonywania obliczeń często powtarza się wielokrotnie takie same sekwencje operacji. Znacznym ułatwieniem pracy byłoby zakodowanie ich jako programu (*makroinstrukcji*) wykonywanego na żądanie, np. przez wybranie z listy odpowiedniego menu, naciśnięcie klawiszy skrótu lub przycisku na pasku narzędzi. Najprostszym sposobem dokonania tego w Excelu jest skorzystanie z tzw. rejestratora makr, który pozwala na „nagranie" serii wykonywanych kroków, które następnie automatycznie konwertuje na kod VBA. Aby zarejestrować makro w VBA nie trzeba więc wcale znać tego języka.

Makro to zapisana pod określoną nazwą seria poleceń, które mogą być wykonywane w Excelu.

Ćwiczenie 1. Rejestracja prostego makra.

Celem ćwiczenia jest utworzenie makra, zmieniającego czcionkę i kolor zaznaczonych komórek. W tym celu należy wykonać poniższe kroki:

1. Otwórz nowy skoroszyt.
2. Wpisz do komórki A1 swoje imię, B1 - nazwisko, C1 – nazwę miasta, D1 – ulicy.
3. Zaznacz komórkę A1.
4. Wybierz z *Narzędzia/Makro/Zarejestruj nowe makro.*
5. Nadaj makru nazwę *Czcionka.* W okienku `Przechowuj makro w:` powinno pozostać *Ten skoroszyt.* Po wciśnięciu *Enter* rozpoczynamy rejestrację, na pasku stanu pojawiło się słowo *Rejestruj* i pasek narzędzi *Zatrzymaj rejestrowanie.*
6. Wybierz polecenie *Format/Komórki.* Ustaw czcionkę 18 pkt i niebieski.
7. Kliknij przycisk *Zatrzymaj rejestrowanie* lub *Narzędzia/Makro/Zatrzymaj rejestrowanie* – jeśli pasek narzędzi *Zatrzymaj rejestrowanie* jest wyłączony, wybierz polecenie *Narzędzia, Makro, Zatrzymaj rejestrowanie.*

Kod makra zastał zapisany w dodatkowym arkuszu, nazywanym modułem, utworzonym i dołączonym do wskazanego skoroszytu. Ponieważ jako miejsce przechowywania makra wybrano *Ten skoroszyt*, to kod makra zapisano w skoroszycie roboczym (Makra.xls).

Ogólnie możemy rozróżnić dwa podstawowe typy modułów: moduły klasy i moduły standardowe.

Moduł standardowy jest to zbiór deklaracji i procedur VBA przechowywanych razem jako jedna całość. Każda procedura w module może być typu *Function* lub *Sub.*

Moduły klasy zawierają definicje nowych, własnych obiektów utworzonych przez programistę w VBA. Ponieważ przede wszystkim tworzone są moduły standardowe nazywane są one dla uproszczenia **modułami**. W czasie rejestrowania makra, moduł tworzony jest automatycznie (o ile jeszcze nie istnieje), jeżeli chcemy, możemy umieszczać w kodzie dodatkowe moduły.

Nazwa makra może składać się z 255 znaków (litery, cyfry, podkreślenia), musi zaczynać się od litery. Pozostałe znaki w nazwie mogą być literami, cyframi oraz podkreśleniami. W nazwie makra nie można umieszczać spacji.

Wykonywanie makra

Podczas wykonywania makra realizowane są te same kroki, co podczas rejestrowania. Chcąc więc uruchomić makro należy:

1. Zaznaczyć komórkę lub obszar (np.B1, a później C2:E3), na którym makro ma zostać wykonanc.
2. Wybrać *Narzędzia/Makro/Makra* lub **Alt+F8**.
3. Wybrać nazwę makra i przycisk *Uruchom.*

Zauważmy, że po wykonaniu makra Czcionka, krój i rozmiar liter we wszystkich komórkach zaznaczonego obszaru (C2:E3) ulega zmianie, pomimo że podczas rejestrowania makra zmieniana była tylko jedna komórka.

W czasie tworzenia makra, Excel konwertuje wykonywane czynności na kod VBA. Aby zapoznać się z podstawowymi funkcjami środowiska VBA obejrzmy kod makra. W tym celu:

1. Wybierz polecenie *Narzędzia/Makro/Makra*, wyświetlając okno dialogowe *Makro.*
2. Wybierz nazwę makra, a następnie przycisk *Edycja -> Mc Visual Basic Editor.*

Po wykonaniu powyższych czynności zobaczymy kod podobny do poniższego

```
Sub Czcionka()
'
' Czcionka Makro
' Makro zarejestrowane 2002-09-05, autor AM
'
'
    With Selection.Font
        .Name = "Arial CE"
        .FontStyle = "Normalny"
        .Size = 18
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = 5
    End With
End Sub
```

Zauważmy, że w powyższym makrze zostało zarejestrowanych więcej operacji niż wykonaliśmy, ponieważ program rejestrujący makro zapisuje również instrukcje kodujące ustawienia domniemane, nie wykonywane naprawdę podczas rejestracji makra, czyli w omawianym przypadku zostały zapisane wszystkie dane na temat czcionki znajdujące się w oknie dialogowym *Formatuj komórki*. Część informacji jest zbędna i spowalnia wykonywanie makra, co jest istotne w przypadku dużych kodów.

Kod makra jest to tzw. podprogram typu proceduralnego, którego postać ogólna , to:

```
Sub Nazwa_makra()
     Komentarze (opcjonalnie)
     Instrukcje
End Sub
```

Słowa kluczowe VBA `Sub` (ang. *subroutine*, czyli podprogram) i `End Sub` ograniczają kod makra. Po słowie kluczowym następuje nazwa, czyli w ćwiczeniu 1, będzie to

```
Sub Czcionka()
```

Komentarze (wyświetlane w kolorze zielonym) są informacjami dla programisty ignorowanymi przez kompilator. Aby tekst był interpretowany jako komentarz należy poprzedzić go apostrofem (‘). Komentarz można umieścić w dowolnym miejscu, również w wierszu zawierającym instrukcję (oczywiście wyłącznie za instrukcją). W tworzonym makrze są to: `nazwa, data rejestracji i autor.`

Instrukcje tworzą program, który jest wykonywany po skompilowaniu. Składają się one ze słów kluczowych (wyświetlanych w kolorze niebieskim), identyfikatorów (nazw obiektów, metod i właściwości).

Zasadnicza część przykładowego makra zaczyna się od słowa `with`. Słowa kluczowe `With` i `End With` tworzą instrukcję odsłaniającą, którą omówimy później, zaś `Selection` jest obiektem oznaczającym wybrany obszar arkusza, co powoduje, że poniższe instrukcje wykonywane będą na zaznaczonym bloku, bez względu czy jest to jedna komórka czy cały arkusz.

W trakcje edycji możemy dokonywać zmian w kodzie makra, co jest szczególnie przydatne, gdy popełniliśmy pomyłkę w trakcie rejestracji jego lub chcemy zmodyfikować istniejące makro.

Ćwiczenie 2. Usunąć z kodu makra Czcionka zbędne instrukcje (to czego nie przestawialiśmy), następnie dodać własne komentarze, przestawić rozmiar czcionki na 14 oraz zapisać, skompilować i uruchomić zmodyfikowane makro.

Zakres działania makra

W czasie tworzenia makra, w oknie *Rejestruj makro*, mamy do wyboru różne miejsca jego przechowywania i w konsekwencji widoczności. Mamy mianowicie trzy możliwości:

- **ten skoroszyt** – zarejestrowane makro będzie widoczne tylko wówczas gdy ów skoroszyt będzie otwarty (opcja domyślna omówiona w ćwiczeniu 1),
- **nowy skoroszyt** – taki skoroszyt zostanie automatycznie utworzony i w nim widoczne makro,
- **skoroszyt makr osobistych** – to specjalny, ukryty skoroszyt, zarezerwowany wyłącznie dla makr. Gdy pierwszy raz zapisujemy w tym miejscu makro, zostaje utworzony plik *PERSONAL.XLS* w folderze XLSTART, i nazwa makra (np. na liście makr) jest poprzedzona nazwą tego pliku. Od tej chwili plik ten będzie otwierany wraz z uruchomieniem Excela, stąd zapisane w nim makra będą stale dostępne, we wszystkich dokumentach.

Edycja lub usuwanie makra

Aby edytować lub usunąć makro ze skoroszytu makr trzeba najpierw odkryć skoroszyt makr, który jest skoroszytem ukrytym. W tym celu z menu *Okno/Odkryj* wybieramy nazwę pliku czyli *Personal.xls*. Dalej przechodzimy do edycji jak w poprzednim przypadku. Aby usunąć makro wybieramy *Narzędzia/Makro/Makra,* wskazujemy nazwę i przycisk *Usuń.*

Przypisywanie klawiszy skrótów

Do często używanych makr wygodnie jest przypisywać klawisze skrótów. **Klawisz skrótu** to kombinacja klawiszy, której naciśnięcie spowoduje wykonanie określonego polecenia. Skrót w Excelu składa się z klawisza *Ctrl* i wybranej przez użytkownika litery. Jeżeli klawisz skrótu, który przypiszesz do makra, będzie taki sam jak któryś z domyślnych skrótów Excela, to ten drugi zostanie przykryty na czas, gdy skoroszyt zawierający to makro będzie otwarty, co może to powodować „lekkie" zmieszanie.
Klawisz skrótu możemy przypisać zarówno podczas tworzenia makra jak i po jego utworzeniu. Chcąc skorzystać z pierwszej możliwości, w trakcie definiowania makra, gdy

wprowadzamy nazwę makra, wpisujemy też literę, której chcemy użyć w skrócie, do pola tekstowego *Klawisz skrótu*. Gdy makro już istnieje postępujemy zgodnie z podanymi krokami:

1. Otwieramy skoroszyt, w którym jest makro.
2. Wybieramy polecenie *Narzędzia/Makro/Makra.*
3. W oknie dialogowym wybieramy nazwę makra, do którego chcemy przypisać klawisz skrótu, a następnie wybieramy *Opcje.*
4. Podajemy literę w polu *Klawisz skrótu.*

Z pomocą rejestratora można tworzyć różne makra, jednak ma on pewne ograniczenia, nie można bowiem tworzyć procedur, w których potrzebne jest:

- poproszenie użytkownika o podanie informacji w czasie wykonywania makra,
- wykonanie różnych operacji w zależności od poleceń użytkownika czy wartości komórek,
- wyświetlanie okien dialogowych,
- wyświetlanie i używanie niestandardowych formularzy.

Powyższe niedogodności nie występują, gdy bezpośrednio programujemy w VBA, zatem w dalszej części przejdziemy do opisu języka VBA.

Podstawowe elementy języka VBA

Podstawowym elementem programu napisanego w VBA są **słowa kluczowe**. Są to wybrane z języka angielskiego słowa, ich skróty lub zbitki skrótów, interpretowane jednoznacznie przez kompilator języka. Znaczenie kilku z nich (np. `Sub`, `End Sub`, `With`, `End With`) przedstawiliśmy omawiając makro z Ćwiczenia 1.

Kolejnym elementem są identyfikatory czyli **nazwy** zmiennych, stałych, obiektów, procedur, funkcji i programów za pomocą których można się do nich odwoływać.
Nazwy muszą zaczynać się od litery, mogą zawierać litery, liczby oraz znaki podkreślenia, nie mogą zawierać spacji, kropek, wykrzykników oraz @,&,$,# i nie mogą zawierać więcej niż 255 znaków.
Kompilator traktuje tak samo małe i wielkie litery czyli NAZWA, Nazwa i naZwa są traktowane jako te same.

Kolejnym elementem programu są **liczby.** Najczęściej stosujemy liczby całkowite (zapis np. 45, -3) lub w rozwinięciu dziesiętnym (np. 23.45). Nawet gdy w opcjach środowiska Windows wybrano jako separator dziesiętny przecinek, to bezpośrednio w kodzie programu separatorem dziesiętnym pozostaje kropka. Liczby bardzo małe lub bardzo duże zapisuje się w postaci zmiennoprzecinkowej (logarytmicznej) nazywanej potocznie *naukową* (np. 2,36E+12, 3,45E-10).

Łańcuchy alfanumeryczne stanowiące kolejny element języka to ciągi znaków ujęte w cudzysłowy, służące do przekazywania danych tekstowych (np. „Dane wejściowe"). Rozróżniane są oczywiście litery małe i duże oraz są dostępne polskie znaki.

Kod programu może również zawierać wspomniane wcześniej **komentarze**, czyli informacje i wyjaśnienia dla programistów, ignorowane przez kompilator. Można je umieszczać w dowolnym miejscu programu jako tekst poprzedzony apostrofem.

Jednostką kodu VBA są **procedury**, czyli serie poleceń, które służą do wykonania określonego zdania lub obliczenia pożądanej wartości. Każda procedura w skoroszycie ma unikalną nazwę, dzięki której można ją identyfikować. Są dwa typy procedur: *Sub* i *Function*. Dla uproszczenia procedury typu Function nazywamy krótko funkcjami, zaś typu Sub procedurami. Różnią się one głównie sposobem komunikacji między Użytkownikiem i komputerem.

Procedury *Sub* wykonują jedną lub więcej operacji, po czym nie zwracają żadnej wartości (przykładem zarejestrowane makro). Makra potrafią rejestrować kroki jedynie w procedurach typu *Sub*.

Przykład. Zmiana czcionki na 14, Wingdings i kolor czerwony .

```
Sub CiekawaCzcionka()
    With Selection.Font
        .Name = "Wingdings"
        .Size = 14
        .ColorIndex = 3
    End With
End Sub
```

Procedury typu **Function** (*Funkcje*) potrafią zwracać wartość. Jest ona zazwyczaj rezultatem wykonywanych obliczeń lub testu (wartości „prawda" lub „fałsz"). Zdefiniowana w ten sposób funkcja zostaje zapisana i jest dostępna w taki sam sposób jak funkcje wbudowane. Poniższy listing przedstawia prostą funkcję do ubruttowienia „kwoty"

Przykład.

```
Public Function Brutto(Kwota)
    Brutto = Kwota * 1.2
End Function
```

Powyższa funkcja używa argumentu `Kwota`. Argumenty mogą być stosowane zarówno w procedurach `Sub` jak i funkcjach i mogą być liczbą lub adresem komórki, która zawiera odpowiednią daną.

Tworzenie procedury wymaga dodania do zeszytu modułu z procedurą. Dodanie modułu wymagane jest tylko jeden raz do każdej tworzonej aplikacji, choć można mieć więcej modułów. Aby to zrobić należy

- otworzyć skoroszyt,
- *Narzędzia/Makro/Edytor Visual Basic,*
- w oknie *Project Explorer* zaznaczamy *This Workbook*, i z menu podręcznego wybieramy polecenie *Insert Module.*

Aby wpisać procedurę należy wpisać jej kod w oknie `Module1(Code)` lub częściowo skorzystać z możliwości wstawiania czyli

- wybieramy polecenie *Insert/Procedure,*
- podajemy jej nazwę i określamy typ.

Po wpisaniu polecenia i naciśnięciu spacji zostanie wyświetlona składnia polecenia – opcja ta nazywana jest *Auto QuickInfo*.

Ćwiczenie. Celem ćwiczenia jest utworzenie procedury w edytorze VBA. W oknie `Module (Code)` proszę wpisać kod pierwszej procedury postaci:

```
Public Sub Pierwsza()
     MsgBox „To jest procedura wpisana w kodzie VBA"
End Sub
```

W procedurze występuje funkcja `MsgBox`, którą omówimy później. Aby obejrzeć efekt jej działania, należy uruchomić makro `Pierwsza`. Wcześniej opisaliśmy parę sposobów uruchamiania makra, kolejnym stosowanym z poziomu edytora jest menu *Run/Run Macro* lub przycisk *Run* lub *F5*.

Aby zapisać nową procedurę należy zapisać skoroszyt, w którym się ona znajduje. Można to zrobić także z poziomu edytora (*File/Save Zeszyt1*).

Zmienne

Do przechowywania w pamięci wartości danych służą **zmienne** określonego typu. Mogą one reprezentować dane, których wartość jest inna przy każdorazowym uruchomieniu programu, a także może zmieniać się podczas jego działania.

Ćwiczenie. Proszę wpisać poniższą procedurę (korzystając z opcji *Insert*), a następnie ją uruchomić

```
Public Sub Imie()
    InputBox "Wprowadź imię"
End Sub
```

Wpisana dana (imię) nie została zapamiętana, ponieważ nie zdefiniowaliśmy żadnej zmiennej do przechowania tej wartości. Aby zdefiniować zmienną trzeba podać jej typ i nazwę. **Typ zmiennej** określa jakie dane mogą być przy jej pomocy przechowywane

Nazwa typu	**Opis, rozmiar**	**Zakres**
Byte	1 bajt = 8 bitów	0-255
Boolean	wartość logiczna, 2 bajty	True lub False
Integer	Liczba całkowita, 2 bajty	-32 768=-2^16/2 do 32 767
Long (long integer)	Liczba całkowita długa, 4 B	
Single	L. zmiennoprzecinkowa, 4B	
Nazwa typu	**Opis, rozmiar**	**Zakres**
Double	L. zmiennoprzecinkowa, 8B	
Currency	Waluta, 8 bajtów	
Decimal	L. dziesiętna, 14 B	
Date	Data, 8B	1.1.100-31.12.9999

Object	Wskaźnik, 4B	Dowolne odwołanie do obiektu
String (variable length)	Łańcuch znaków o zmiennej długości, 10B +1B/znak	Od 0 do około 2 mld
String (fixed length)	Tablica znaków (łańcuch znaków o stałej długości)	Od 1 do około 65 400
Variant (number)	16B	Dowolna wartość numeryczna nie przekraczająca zakresu zmiennej typu Double
Variant (text)	22B + 1B/znak	Jak String o zmiennej długości

Tworzenie zmiennych zwane jest *deklarowaniem* zmiennych. Chcąc utworzyć zmienną korzystamy z instrukcji `Dim`, o następującej składni:

```
Dim NazwaZmiennej As TypZmiennej
```

Zasady dotyczące nazw zmiennych są takie same jak w przypadku procedur (lepiej nie używać polskich liter, choć nie jest to zakazane)

Ćwiczenie. Zmodyfikować kod procedury poprzez dodanie deklaracji zmiennej

```
Public Sub Imie()
    Dim Imie As String

    Imie = InputBox("Wprowadź imię:")
       'po zdefiniowaniu przypisujemy zmiennej Imie wartość
    MsgBox "Cześć " & Imie
End Sub
```

Jeżeli w instrukcji `Dim` nie podamy typu zmiennej automatycznie zostanie nadany domyślny typ Variant (zajmuje dużo pamięci i spowalnia prace programu, gdyż komputer musi sprawdzać jakie zmienne są przechowywane w zmiennych typu Variant).

Konwencja nazywania zmiennych zakłada, aby poprzedzać nazwę zmiennej przedrostkiem określającym jej typ (czyli zamiast *Imie* lepiej *sImie,* a dla liczby całkowitej np. *iNumer*).

W powyższym ćwiczeniu został także dodany komentarz, czyli tekst poprzedzony apostrofem.

Deklarowanie tablic

Tablice stanowią zbiór wartości tego samego typu, współdzielących tę samą nazwę. Poszczególne elementy tablicy identyfikowane są przy pomocy numeru indeksu (liczby określającej położenie elementu w tablicy). Składnia deklaracji tablicy:

```
Dim NazwaTablicy(n) As TypZmiennej
```

gdzie n oznacza ilość-1 elementów tablicy. Ilość – 1 (wartości zmniejszone są o 1, gdyż pierwszy element ma indeks 0). Tablice przydają się podczas działań na podobnych danych.

Jeżeli chcemy przetwarzać 15 wyników doświadczenia, możemy zadeklarować 15 zmiennych lub jedna tablicę

```
Dim iWyniki(15) As Integer
```

Oprócz tablic jednowymiarowych (wektorów) w VBA mamy do dyspozycji tablice wielowymiarowe. Np. tablicę dwuwymiarową (macierz, arkusz, tabelę) o wielkości 4x4 deklarujemy następująco

```
Dim iTablica(3,3) As Integer
```

Inną opcją, która możemy zastosować jest deklarowanie tablicy bez podania jej wymiaru (można w czasie trwania zlecić użytkownikowi podanie wymiaru), co więcej wymiar możemy zmieniać już po utworzeniu. VBA tworzy wówczas tzw. *tablicę dynamiczną.*

```
Dim DynTablica() As TypZmiennej
```

Po zadeklarowaniu tablicy możemy ją wymiarować za pomocą instrukcji `ReDim`

```
ReDim DynTablica(RozmiarTablicy)
```

gdzie `RozmiarTablicy` oznacza nowy jej wymiar. Chcąc zachować ten wymiar po poleceniu ReDim piszemy `Preserve`

```
ReDim Preserve DynTablica(RozmiarTablicy)
```

Gdy przypisujemy wartość do elementu tablicy, trzeba podać jego numer w tablicy (indeks).

```
Dim iLiczba As Integer
Dim iWynikiTestu() As Integer
Dim i As Integer

iLiczba = InputBox("Podaj liczbę studentów: ")
ReDim Preserve iWynikiTestu(iLiczba)

For i = 10 To iLiczba
     iWynikiTestu(i)=InputBox(„Podaj wynik testu: "&i)
Next
```

Używanie wartości stałych w programach

Stałe to miejsca, w których przechowujemy wartości, które się nie zmieniają np. stałe fizyczne czy chemiczne, choć w rzeczywistości możemy je zmienić, gdy zajdzie taka potrzeba. Deklaracja stałej ma postać

```
Const nazwa_stałej As typ_stałej = wartość_stałej
```

czyli na przykład

```
Const STALA_GAZOWA As Single = 8.31451
```

Często stosowaną konwencją przy nadawaniu nazw stałym jest stosowanie tylko wielkich znaków. Dzięki temu analizując kod możemy szybko zorientować się, które nazwy oznaczają zmienne (i jakiego typu), a które stałe.

Jeżeli chcemy zadeklarować wektor stałych wartości, to musimy tego dokonać wewnątrz programu, korzystając z polecenia `Array`. Oto przykład deklaracji tablicy stałych współczynników

```
Public Sub StalaTablica()
Dim wartosci As Variant
  wartosci = Array (1,2.5,6.9,9.01,10,12)
     'Tutaj powinien znajdować się kod programu
     'wykorzystującego zdefiniowaną tablicę
End Sub
```

Zasięg zmiennych

Zmienne i stałe mogą być zadeklarowane wewnątrz procedury albo na początku modułu w sekcji zwanej *General Declarations.* Lokalizacja deklaracji zmiennej określa jej zasięg. Kiedy deklarujemy zmienna wewnątrz procedury, jest ona widziana tylko przez te procedurę; żadne inne procedury nie mogą używać wartości przechowywanej w tej zmiennej. Tego typu zmienne określane są mianem *zmiennych o zasięgu procedurowym* lub *zmiennych lokalnych.*

Jeśli zadeklarujemy zmienne w sekcji *General Declarations* modułu, wówczas wszystkie procedury zlokalizowane w tym module będą mogły używać owej zmiennej lub stałej. Tego typu zmienne nazywamy *zmiennymi o zasięgu modułowym* lub *zmiennymi modułowymi.*

Taka sama koncepcja zasięgu znajduje zastosowanie również w przypadku procedur.

Istnieje jeszcze jeden rodzaj zasięgu: **zmienne publiczne**, zwane też **zmiennymi globalnymi**, które mogą być używane w dowolnej procedurze aplikacji, niezależnie od tego, czy znajduje się ona w tym samym module co zadeklarowana zmienna (lub stała) czy też innym. Do tworzenia zmiennych globalnych służy instrukcja `Public`.

```
Public NazwaZmiennej As TypZmiennej
```

Stałe publiczne zaś definiujemy następująco

```
Public Const NAZWASTALEJ TypStalej = wartosc
```

Zmienne oraz stałe globalne deklarujemy w sekcji *General Declarations,* znajdującej się na początku modułu.

Przejdźmy do omówienia zastosowanych w ćwiczeniach funkcji, ich składni i zastosowań.

Funkcja MsgBox

Funkcja ta stosowana jest do wyświetlania komunikatów w oknach dialogowych. Okno takie, po wyświetleniu na ekranie „oczekuje" na kliknięcie odpowiedniego przycisku, który się w nim znajduje . W zależności od wyboru przycisku funkcja zwraca określona wartość typu `integer`. Składnia funkcji MsgBox jest następująca

```
MsgBox(prompt [,buttons] [,title] [,helpfile, context])
```

Jedynym argumentem wymaganym jest prompt. Jest to łańcuch znaków pojawiający się w oknie dialogowym wyświetlanym przez te funkcje.

W języku polskim możemy zapisać

```
MsgBox(tekst_komunikatu,[przyciski],[nagłówek],[plik_pomocy],
[kontekst])
```

Przykład.

```
MsgBox „Czy zapisać zmiany w '"& ThisWorkbook.Name &"'", 
vbYesNoCancel + vbExcamation
```

Jeżeli zwracana przez funkcję wartość jest dla nas istotna, argumenty obejmujemy nawiasami i wprowadzamy zmienną. Wtedy musimy zadeklarować zmienną typu integer.

```
Dim iOdpowiedz As Integer
iOdpowiedz = MsgBox „Czy zapisać zmiany w '"& _
ThisWorkbook.Name &"'", vbYesNoCancel + vbExcamation
     'spacja_ jest znakiem kontynuacji wiersza.
```

Powyższy przykład daje w efekcie znane nam okienko, które pojawia się przy próbie zamknięcia skoroszytu, bez uprzedniego zapisania dokonanych zmian.

Jeśli pominiemy argument buttons (przyciski), VBA domyślnie umieści w wyświetlanym okienku przycisk OK. Argument buttons pozwala określić

- ile przycisków zostanie wyświetlonych w oknie komunikatu,
- jakie przyciski się tam znajdą,
- jaka ikona będzie wyświetlana,
- który przycisk będzie domyślny,
- jaki będzie tryb modalności okna.

Poniżej przedstawiamy dostępne wartości tego argumentu. Grupa 1 określa liczbę oraz typ przycisków, grupa 2 odpowiada za styl ikony, grupa 3 decyduje który z przycisków będzie domyślny, grupa 4 określa rodzaj modalności okna komunikatu. Dodając wartości otrzymujemy wartość argumentu funkcji button

Grupa	Wartość	Stała	Opis
Grupa 1	VbOKOnly	0	Wyświetla jedynie przycisk OK. (domyślnie)
	VbOKCancel	1	Ok. i Anuluj
	VbAbbortRetryIgnore	2	Przerwij, Ponów, Zignoruj
	VbYesNoCancel	3	Tak, Nie, Anuluj
	VbYesNo	4	Tak, Nie
	VbRetryCancel	5	Ponów, Anuluj
Grupa 2	VbCritical	16	Wyświetla ikonę Krytyczny
	VbQuestion	32	Pytanie
	VbExclamation	48	Wykrzyknik
	VbInformation	64	Pytajnik
Grupa 3	vbDefaultButton1	0	Ustawia pierwszy przycisk jako domyślny
	vbDefaultButton2	256	Ustawia drugi przycisk jako domyślny
	vbDefaultButton3	512	Ustawia trzeci przycisk jako domyślny
	vbDefaultButton4	768	Ustawia czwarty przycisk jako domyślny
Grupa 4	VbApplicationModal	0	Okno modalne dla aplikacji, użytkownik wybrać przycisk, zanim będzie kontynuował pracę w aplikacji
	VbSystemModal	4096	Okno modalne dla systemu, praca wszystkich aplikacji zostanie zawieszona do momentu aż użytkownik wybierze przycisk
Dodatkowe opcje	VbMsgBoxHelpBotton	16384	Dodaje do okienka komunikatu przycisk Pomoc
	vbMsgBoxSetForeground	65536	Okno komunikatu ustawia jako pierwsze
	vbMsgBoxRight	524288	Wykonuje komunikat w oknie od lewej do prawej
	vbMsgBoxRtlReading	1048576	Wyświetla tekst w oknie od prawej do lewej

Opcjonalny argument `title(nagłówek)` może zawierać łańcuch znaków wyświetlany na pasku tytułowym okna dialogowego (domyślnie Mc Excel).

`Helpfile (plik pomocy)` oraz `context (kontekst)` to opcjonalne argumenty używane podczas tworzenia aplikacji pomocy.

Zwracane przez funkcję `MsgBox` wartości wynoszą, zależnie od wybranego przycisku i tak otrzymujemy

Stała	Wartość	Opis
VbOK	1	OK.
VbCancel	2	Anuluj
VbAbort	3	Przerwij
VbRetry	4	Ponów
VbIgnore	5	Zignoruj
VbYes	6	Tak
VbNo	7	Nie

Funkcja InputBox

Funkcja ta wyświetla na ekranie okno dialogowe, zawierające pole tekstowe, do którego użytkownik może wprowadzić dane.

```
InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,cont
ext])
```

Podobnie jak w MsgBox jedynym niezbędnym argumentem jest `prompt`. Pozostałe argumenty

`[,title]` i `[,helpfile,context]` – jak w MsgBox

`[,default]` łańcuch znaków domyślnie wyświetlany w polu tekstowym

`[,xpos][,ypos]` opcjonalne argumenty określające położenie okna dialogowego na ekranie.

Czyli w języku polskim możemy zapisać

```
InputBox(tekst_komunikatu,        [nagłówek],[wartość_domyślna],
         [od_lewej], [od_góry], [plik_pomocy], [kontekst])
```

Wartość zwrócona przez funkcję `InputBox` możemy wpisać do komórki za pomocą właściwości `Value`.

Przykłady:

```
Public Sub DemoFunkcjaInput()
Dim iWynik As Integer
     iWynik = InputBox("Podaj swoją ulubioną liczbę: ")
     ActiveCell.Value = iWynik
          'przypisanie aktywnej komórce podanej wartości
End Sub
```

```
Public Sub Imie()
    Dim sImie As String
    sImie = InputBox("Wprowadź imię:")
       'po zdefiniowaniu przypisujemy zmiennej sImie wartość
    MsgBox "Cześć " & Imie
End Sub
```

Metoda InputBox

Metoda `Application.InputBox` jest podobna do funkcji `InputBox` zawiera jednak kilka istotnych różnic. Jej składnia jest postaci:

```
Application.InputBox(Prompt, [Title],[Default],[Left],
          [Top],[HelpFile],[HelpContextId],[Type])
```

czyli

```
Application.InputBox(tekst_komunikatu, [nagłówek],[wartość
          domyślna],[od_lewej],[od_góry],[plik_pomocy],
          [kontekst],[Type:=numer_typu])
```

Wywołanie metody zaczyna się od słowa Application, gdyż każda metoda operuje na konkretnym obiekcie, do którego należy. „Właścicielem" tej metody jest Excel, który jest aplikacją.

`Type` określa typ zwracanej wartości (dostępne typy umieszczone są w tabeli). Każdy typ zdefiniowany jest poprzez liczbę. Metoda InputBox może być przygotowana na wartości różnych typów, wówczas sumujemy wartości z tabeli (stąd „dziury" w numeracji). Gdy chcemy na przykład aby zwracana wartość mogła być zarówno tekstem jak i liczbą, to sumujemy i podajemy `Type:=3`

Wartość	Spodziewana wartość zwracana
0	Formuła
1	Liczba
2	Tekst (String)
4	Wartość logiczna (True, False)
8	Adres komórki
16	Wartość błędu
64	Tablica wartości

Ćwiczenie. Uruchomić poniższą procedurę, korzystając z funkcji InputBox, a następnie metody Application.InputBox. Po pojawieniu się okienka dialogowego wpisać w nie tekst (typ niezgodny z zadeklarowanym). Porównać rezultaty

```
Public Sub DemoMetodaInput()
Dim iWynik As Integer
   iWynik = Application.InputBox("Podaj swoja liczbę: ", , , ,
, , , 1)
   MsgBox iWynik
   ActiveCell.Value = iWynik
End Sub
```

Ponieważ podana wartość nie zgadza się z zadeklarowanym typem, w przypadku metody InputBox został wyprowadzony komunikat o błędzie. Widzimy zatem, że jedną z korzyści wynikających z używania metody `InputBox` jest wbudowana obsługa błędów. Drugą różnicą pomiędzy metodą a funkcją `InputBox` jest rezultat zwracany w momencie kliknięcia przez użytkownika przycisku *Anuluj*. Otóż funkcja `InputBox` zwraca łańcuch o zerowej długości, natomiast metoda w takim przypadku zwraca wartość logiczną `FALSE` (fałsz).

W poleceniu `Application.InputBox("Podaj swoja liczbę: ", , , , , , , 1)` przecinki to miejsca występowania argumentów, których nie podaliśmy. Zamiast nich można korzystać z możliwości stosowania tzw. argumentów nazwanych. Wówczas zamiast 7 przecinków, aby trafić w dobre miejsce, możemy podając nazwy argumentów wpisywać je w dowolnej kolejności, przypisując im wartości wg wzoru

```
Nazwa_argumentu:=wartość
```

Powyższy przykład możemy zatem zapisać:

```
Public Sub DemoMetodaInput()
Dim iWynik As Integer
   iWynik = Application.InputBox("Podaj swoja liczbę: ", Type:=1)
   MsgBox iWynik
   ActiveCell.Value = iWynik
End Sub
```

Operatory i wyrażenia

Wyrażenie jest tekstem określającym czynności wykonywane w celu utworzenia wartości zmiennej. Jego elementami są nazwy zmiennych, stałych, funkcji, wartości (liczbowe, tekstowe lub logiczne) oraz operatory. W VBA mamy operatory arytmetyczne, logiczne, relacyjne oraz łańcuchowe.

Operator konkatenacji i łączenie tekstów

Termin konkatenacja oznacza łączenie tekstów. Do tego celu można używać operatora arytmetycznego (+) lub operatora tekstowego (&), który wymusza traktowanie zmiennej jako tekst.

Ćwiczenie. Proszę zadeklarować zmienna wynik typu string i wykonać następujące przykłady wyprowadzając wynik na ekran (np. za pomocą MsgBox)

Przykład 1. Dodawanie tekstów operatorem +

```
wynik = "kwas " + "siarkowy"
```

Przykład 2. Dodawanie tekstów operatorem &

```
wynik = "kwas " &  "siarkowy"
```

KOMENTARZ DO PRZYKŁADÓW 1 – 2: Działanie operatorów + oraz & było identyczne, gdyż argumenty były rzeczywiście tekstami.

Przykład 3. Dodawanie liczb operatorem &

```
wynik = 5 & 3
```

KOMENTARZ. Użycie operatora & wymusza niejawną konwersję liczb na teksty nawet jeżeli nie umieszczono ich w apostrofach.

Przykład 4. Dodawanie liczb jako tekstów operatorem +

```
wynik = "51" + "9"
```

KOMENTARZ. Liczba w apostrofach jest tekstem.

Gdy chcemy część łańcuch przenieść do następnego wiersza używamy do tego stałej `vbNewLine`, co ilustruje poniższy przykład.

```
Public Sub DlugiLancuch()
    Dim sDlugiTekst As String

   sDlugiTekst = "Oto przykład bardzo długiego łańcucha "
   sDlugiTekst = sDlugiTekst & "znaków będącego połączeniem "
   sDlugiTekst = sDlugiTekst & "długich łańcuchów" & vbNewLine
   sDlugiTekst = sDlugiTekst & "z przejściem do nowego wiersza"
   MsgBox sDlugiTekst

End Sub
```

Operatory i wyrażenia arytmetyczne

Operatory arytmetyczne służą do tworzenia wyrażeń arytmetycznych tj. z użyciem argumentów typu liczbowego, dzielimy je na jednoargumentowe i dwuargumentowe. Operatorami jednoargumentowymi są operatory tożsamości (+) oraz zmiany znaku.

Operatory i wyrażenia logiczne

Instrukcje VBA mogą zawierać wyrażenia (zdania) logiczne, których wartością jest prawda (TRUE) lub fałsz (FALSE). Do tworzenia stosujemy operatory: *negacji* (Not), *koniunkcji* (And), *alternatywy* (Or), *różnicy symetrycznej* (Xor), *równoważności* (Eqv), *implikacji* (Imp) oraz *operator identyczności obiektów* (Is). Argumentami wyrażeń logicznych są zasadniczo zmienne typu Boolean (logiczne) lub zdania logiczne. Działają one jednak także na zmiennych liczbowych.

Operatory i wyrażenia relacyjne

Operatory relacyjne służą do porównania dwóch argumentów w wyniku czego zostanie utworzona wartość logiczna, którą można przypisać zmiennej typu Boolean. Są to operatory: *równy* (=), *nierówny* (<>), *mniejszy* (<), *większy* (>), *nie większy* (<=) oraz *nie mniejszy* (=>).

Wyrażenie ma więc postać

```
argument_1 operator argument_2
```

Argumentami mogą być zmienne (stałe) typu arytmetycznego (liczby) lub tekstowego (stringi). Do porównywania tekstów można też używać operatora `Like`

(*podobny do*), który umożliwia stosowanie znanych znaków globalnych

?	pojedynczy znak
*	sekwencja znaków
#	pojedyncza cyfra
[]	należy do listy
[!]	nie należy do listy

Instrukcje

Operacje wykonywane na danych są kodowane w postaci instrukcji złożonych ze słów kluczowych i wyrażeń. Omówimy podstawowe instrukcje VBA.

Instrukcje przypisania

Stosowaliśmy już instrukcje przypisania, która służy do nadania zmiennej wartości aktualnej i jest postaci

```
Nazwa_zmiennej = wyrażenie
```

Wykonanie instrukcji polega na obliczeniu wartości wyrażenia, a następnie wpisania wyniku do komórek pamięci przydzielonych tej zmiennej w wyniku deklaracji. W instrukcji przypisania istotny jest aspekt zgodności zadeklarowanego typu zmiennej i wyrażenia znajdującego się po prawej stronie operatora przypisania.

Przykład

t0 = 3

liczba_pierwiastkow = 27

Instrukcje warunkowe

Z instrukcji warunkowej korzystamy, gdy w zależności od wyboru użytkownika powinny być wykonane przez program różne instrukcje. ***Instrukcja warunkowa*** analizuje, czy dany warunek jest prawdziwy (true) czy fałszywy (false) i zależnie od wyniku tej analizy, wykonuje odpowiednie polecenia.

Warunki logiczne pozwalają na wybór różnych dróg w programie, w zależności od wartości zmiennej, odpowiedzi uzyskanej od użytkownika, wyniku obliczeń wykonanych przez funkcję lub ustawienia właściwości. Przy pomocy warunków logicznych tworzysz swego rodzaju test, którego rezultat określa przebieg programu. Do konstrukcji warunków stosujemy omówione powyżej operatory

Gdy sprawdzamy warunek złożony (koniunkcja czy alternatywa warunków) stosujemy operatory logiczne, pozwalające na łączenie warunków logicznych.

Operator logiczny	Znaczenie
And	Jeśli wszystkie warunki są prawdą, rezultat też jest prawdą
Or	Jeśli któryś z warunków jest prawdą, rezultat też jest prawdą
Not	Jeśli wyrażenie warunkowe jest fałszem, rezultat jest prawda. Jeśli warunek jest prawdą, rezultat jest fałszem
Xor	Jeśli jeden i tylko jeden warunek jest prawdą, rezultat również jest prawdą. Jeśli oba warunki mają te samą wartość logiczną rezultat jest fałszem
Eqv	Jeśli oba warunki mają te samą wartość logiczną rezultat jest prawdą.

Instrukcja If

Jej składnia jest następująca:

```
If condistions Then
[statements]
[ElseIf condition n Then
[elseifstatements]…
[Else
[elseifstatements]]
End If
```

Wymaganą częścią tej składni jest `condition`. Jest to miejsce, w które wpisujemy testowany warunek. Wykonanie instrukcji rozpoczyna się od obliczenia wartości wyrażenia logicznego. Jeżeli jest ono prawdą, to wykonane zostaną instrukcje umieszczone po słowie `then`. Jeżeli natomiast wartością wyrażenia logicznego jest fałsz, to instrukcje te są pomijane i program jest kontynuowany.

Jeśli przy pomocy tej samej procedury `If` chcemy przetestować drugi warunek, korzystamy z klauzuli `ElseIf`. VBA najpierw testuje warunek umieszczony po instrukcji `If`, jeżeli nie jest on spełniony (otrzymujemy False), VBA rozpoczyna testowanie warunku umieszczonego w pierwszej klauzuli `ElseIf`, po czym, jeśli i ten okaże się fałszem kontynuuje testowanie kolejnych warunków `ElseIf`, do momentu aż któryś z nich okaże się prawdziwy. Jeżeli żaden z nich nie jest prawdą, VBA zakończy wykonywanie instrukcji `If`, o ile nie napotka w niej na opcjonalną klauzulę `Else` w takim, bowiem przypadku zostaną wykonane instrukcje umieszczone po słowie `Else`.

Przykład. Jeśli na pytanie „Czy potrzebny jest transport ?", użytkownik odpowie TAK, do ceny dodawane jest 10, jeśli zaś NIE, wartość pozostaje bez zmian.

```
Public Sub Transport()
Dim iOdp As Integer

iOdp = MsgBox("Czy należy zorganizować transport ?", vbYesNo)
If iOdp = vbYes Then
    Range(ActiveCell.Address).Value = 10
Else
    Range(ActiveCell.Address).Value = 0
End If
End Sub
```

`ActiveCell.Address` oznacza adres aktywnej komórki, jeżeli znamy adres np. C3 można by zapisać `Range(„C3").Value`. Aby tworzyć bardziej skomplikowane instrukcje `If`, można je składać i zagnieżdżać, co ilustruje następny przykład

Instrukcje warunkowe można zagnieżdżać w sobie, co ilustruje poniższy przykład

Przykład. Obliczanie prowizji od sprzedanego artykułu, w zależności od trzech czynników.

1 czynnik – czy dany produkt jest w sprzedaży, jeżeli Nie – prowizja wynosi 0, jeżeli Tak, to prowizja 2%,

2 czynnik – staż pracy, jeśli sprzedawca pracuje więcej niż 5 lat i mniej niż 10, to otrzymuje dodatkowo 1% prowizji, jeżeli 10 lub więcej lat, to 2%,

3 czynnik – dział, osoby pracujące w dziale meblowym dodatkowo 1% prowizji.

```
Public Sub Prowizja()
Dim sngProwizja As Single

If Range("E2") = "Tak" Then
    sngProwizja = 0.02

   If (Range("E3").Value >= 5 And Range("E3").Value < 10) Then
       sngProwizja = sngProwizja + 0.01
   ElseIf Range("E3").Value >= 10 Then
       sngProwizja = sngProwizja + 0.02
   End If

   If Range("E1").Value = "Meblowy" Then
       sngProwizja = 0.01 + sngProwizja
   End If
Else
    sngProwizja = 0
End If

Range("E5").Value = sngProwizja
End Sub
```

Na powyższym wydruku czcionką zaznaczyliśmy odpowiednie pary instrukcji. Trzeba uważać, gdyż instrukcja `If` rozróżnia wielkość znaków (wpisanie „nie" zamiast „Nie"). Chcąc temu zaradzić należy wpisać

`Ucase(Range(„E2").Value)="TAK"` oraz

`Ucase(Range(„E1").Value)="MEBLOWY".`

Select Case

Alternatywą dla wielokrotnego stosowania instrukcji `If` jest `Select Case`. Składnia `Select Case` jest następująca:

```
Select Case testexpression
[Case expressionlist-n
[statements-n]]…
[Case Else
[elsestatements]]
End Select
```