

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Tworzenie makr w VBA dla Excela 2003/2007. Ćwiczenia

Autor: Mirosław Lewandowski

ISBN: 978-83-246-1222-2

Format: A5, stron: 192



Gotowe makra w Excelu! Programy, które ułatwią Ci życie!

- Poznaj niezwykle możliwości języka Visual Basic for Applications
- Naucz się korzystać z makr poszerzających funkcje Excela
- Zaimportuj gotowe programy automatyzujące uciążliwe zadania

Visual Basic for Applications, dostępny w programie Excel język programowania, jest narzędziem bardzo przydatnym w codziennej pracy z arkuszami kalkulacyjnymi. Jego możliwości są naprawdę ogromne, a umiejętne posługiwanie się nim pozwala na znaczne skrócenie czasu wykonywania uciążliwych, codziennych czynności. Automatyzacja zadań, szczególnie ważna w przypadku pracy z rozbudowanymi arkuszami, znacznie ułatwia nam życie i pracę oraz sprawia, że unikamy wielu pomyłek. Jednak nie każdy z nas ma czas i ochotę uczyć się programowania, by tworzyć własne makra i w ten sposób dostosowywać aplikację do własnych potrzeb. Bardzo pomocna staje się więc „ściąga”, w której można znaleźć gotowe programy z objaśnieniami konkretnych zastosowań.

„Tworzenie makr w VBA dla Excela 2003/2007. Ćwiczenia” to właśnie książka, której Ci potrzeba! Znajdziesz w niej wiele programów upraszczających pracę w Excelu i wskazówki związane z ich wykorzystaniem. Dowiesz się, jak używać rejestratora makr, jak konstruować gotowy program ze stałych elementów i jak sprawić, by jego działanie odpowiadało temu, co chcesz osiągnąć. Nauczysz się deklarować zmienne, wykorzystywać pętle i wyszukiwać potrzebne Ci dane. Wszystkie te zadania zostały podane w formie praktycznych ćwiczeń, dzięki czemu bez zbędnych teoretycznych wywodów zapoznasz się z ich działaniem. Ponadto w książce tej uwzględniono sugestie i pytania czytelników jej poprzedniego wydania, co pozwoliło na jeszcze lepsze dostosowanie jej treści do potrzeb użytkowników Excela.

- Rejestrowanie makr
- Uruchamianie zapisanych projektów
- Zmienne i stałe
- Zmienne tablicowe
- Deklarowanie zmiennych
- Pętle
- Komunikaty
- Obsługa błędów
- Funkcje użytkownika
- Zmiana danych w komórkach
- Warunkowa zmiana wyglądu arkusza
- Okno edytora VBA

Niech Twój Excel pracuje dla Ciebie!



Spis treści

	Dla kogo jest ta książka?	5
Rozdział 1.	Zabawy z rejestratorem makr	7
	Wprowadzenie	7
	Dla użytkowników Office 2007	8
	Nowy wygląd — nowe problemy	10
	Rejestrowanie makr	12
	Uruchamianie zapisanych projektów	17
	Szybkie sortowanie danych	25
Rozdział 2.	Podstawy	39
	Interakcja ze skoroszytem. Zmienne i stałe	40
	Deklarowanie zmiennych i ich zasięg	47
	Zmienne tablicowe	53
	Co będzie, jeśli?	55
	Pętle	60
	Idź do, idź i wróć	68
	Dialog z użytkownikiem	71
	Obsługa błędów	83
	Makro a funkcja	86
Rozdział 3.	Przykłady	93
	Liczby słowne	93
	Wygląd zależny od warunków	99
	Nawigacja między arkuszami	102
	Wspomaganie pracy Excela	104

Generowanie dźwięku	116
Obliczanie głębi ostrości	120
Arkusze ofert	133
Rozdział 4. Dodatki	141
Okno edytora VBA	141
Procedury zdarzeniowe	144
Właściwości formantów formularza	158



Podstawy



Pierwszy rozdział podpowiadał, jak można sobie ułatwić codzienną pracę z Excelem i zautomatyzować często powtarzane czynności. Właściwie niezbyt przydała się wiedza na temat VBA — wystarczyło Ci uruchomienie rejestratora makr i pokazanie, czego oczekujesz od komputera.

Jak już zdążyłeś się przekonać, rejestrator — choć bardzo pomocny — nie oferuje możliwości zapisania operacji warunkowej, przypisania zmiennej czy wyświetlenia okien dialogowych. Czynności te musieliśmy wykonywać z poziomu edytora. Dobrze byłoby zatem poznać podstawowe polecenia i struktury odpowiedzialne za wykonywanie operacji, których rejestrowanie jest niemożliwe lub przynajmniej karkołomne.

Wszystkie zamieszczone tu ćwiczenia możesz znaleźć na stronie <http://www.twojexcel.com>.

Interakcja ze skoroszytem.

Zmienne i stałe

Czytanie i umieszczanie danych

Często zdarza się, że napisane przez Ciebie makro umieszcza dane w arkuszu roboczym lub pobiera je stamtąd. VBA oferuje kilka sposobów adresowania komórek arkusza w zależności od tego, jakie dane są dla użytkownika dostępne.

Ć W I C Z E N I E

2.1 Tworzenie tabeli z poziomu VBA

Utwórz arkusz tabliczki mnożenia w zakresie od 1 do 10 według rysunku 2.1. Pomiń formatowanie.

Rysunek 2.1.
Arkusz tabliczki
mnożenia

	A	B	C	D	E	F	G	H	I	J	K
1	1	2	3	4	5	6	7	8	9	10	
2	2	4	6	8	10	12	14	16	18	20	
3	3	6	9	12	15	18	21	24	27	30	
4	4	8	12	16	20	24	28	32	36	40	
5	5	10	15	20	25	30	35	40	45	50	
6	6	12	18	24	30	36	42	48	54	60	
7	7	14	21	28	35	42	49	56	63	70	
8	8	16	24	32	40	48	56	64	72	80	
9	9	18	27	36	45	54	63	72	81	90	
10	10	20	30	40	50	60	70	80	90	100	
11											

Rozwiązanie

1. Otwórz nowy skoroszyt, uruchom edytor VBA (*Alt+F11*) i wstaw moduł (*Insert/Module*).
2. W module wprowadź następujący kod:

```
Sub tabliczka_mnozenia()
    For wiersz = 1 To 10
        For kolumna = 1 To 10
            Cells(wiersz, kolumna) = wiersz * kolumna
        Next kolumna
    Next wiersz
End Sub
```

3. Ustaw kursor w obrębie makra i naciśnij klawisz *F5*, aby uruchomić makro.

Wyjaśnienia

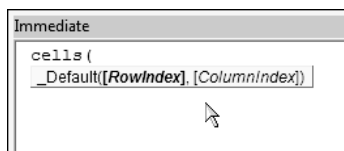
Zastosowane tu zostały instrukcje *pętli* (struktury *For...Next*). Poznasz je w dalszych rozdziałach tego podręcznika. Wpisanie wartości do komórki wykonywane jest w poniższym wierszu kodu:

```
Cells(wiersz, kolumna) = wiersz * kolumna
```

Właściwość *Cells*, określająca adres komórki, ma dwa argumenty. Jak widać w instrukcji *For...Next*, zmienne *wiersz* i *kolumna* przyjmują wartości od 1 do 10. W poleceniu *Cells* zatem zarówno *wiersz*, jak i *kolumna* są określane za pomocą wartości **liczbowych**.

Łatwo pomylić kolejność współrzędnych. Z pomocą przyjdzie wtedy edytor (rysunek 2.2), który sam podpowie, czego od Ciebie oczekuje.

Rysunek 2.2.
Podpowiedzi
edytora VBA
bywają bardzo
pomocne



Ć W I C Z E N I E

2.2 Wpływanie na wygląd komórek arkusza z poziomu VBA

Zaciemnij wnętrza komórek od A1 do J1 i od A2 do A10, jak pokazano na rysunku 2.1.

Rozwiązanie

Wprowadź do modułu następujący kod i uruchom go.

```
Sub wypełnij()  
    Range("A1", "J1").Interior.ColorIndex = 15  
    Range("A2:A10").Interior.ColorIndex = 15  
End Sub
```

Wskazówki

- ❑ Zauważ, że w różny sposób wpisano argumenty *Range*. Obydwa sposoby są poprawne.
- ❑ Jak widać, za pomocą *Range* możemy zaznaczać całe zakresy komórek.

- ❑ Jako argumentów możemy użyć zmiennych (jeżeli ich wartość będzie się składać z liter i cyfr) lub znanych Ci już poleceń Cells. Nasze makro mogłoby więc wyglądać tak:

```
Sub wypeŃnij()  
a = "A1"  
b = "J1"  
    Range(a, b).Interior.ColorIndex = 15  
    Range(Cells(1, 1), Cells(10, 1)).Interior.ColorIndex = 15  
End Sub
```

Odczytywanie wartości z komórek odbywa się w sposób odwrotny, niż są umieszczane. Przećwiczymy to na bardziej użytecznym przykładzie.

Ć W I C Z E N I E

2.3 Pobieranie danych z arkusza

Na podstawie tabliczki mnożenia, utworzonej w ćwiczeniu 2.1, utwórz procedurę, która pobierając dane o adresie zaznaczonej komórki, będzie pobierać dane z:

- ❑ zaznaczonej komórki;
- ❑ komórki z pierwszego wiersza aktywnej kolumny;
- ❑ komórki z pierwszej kolumny aktywnej wiersza.

Dane zostaną wyświetlone w postaci komunikatu (rysunek 2.3). Dodatkowo niech procedura wyświetla komunikaty tylko w przypadku kliknięcia komórki w zakresie od *A1* do *J10*.

Rozwiązanie

```
Sub pobieraj_dane()  
If ActiveCell.Row <= 10 And ActiveCell.Column <= 10 Then  
    a = ActiveCell.Value  
    mnożn1 = Cells(ActiveCell.Row, 1)  
    mnożn2 = Cells(1, ActiveCell.Column)  
    MsgBox (mnożn1 & " razy " & mnożn2 & " = " & a)  
End If  
End Sub
```

Rysunek 2.3.

W oknie komunikatu wyświetlane są dane z pierwszych komórek aktywnego wiersza i kolumny oraz z aktywnej komórki

	A	B	C	D	E	F	G	H	I	J	K
1	1	2	3	4	5	6	7	8	9	10	
2	2	4	6	8	10	12	14	16	18	20	
3	3	6	9	12	15	18	21	24	27	30	
4	4	8	12	16	20	24	28	32	36	40	
5	5	10	15	20	25	30	35	40	45	50	
6	6	12	18	24	30	36	42	48	54	60	
7	7	14	21	28	35	42	49	56	63	70	
8	8	16	24	32	40	48	56	64	72	80	
9	9	18	27	36	45	54	63	72	81	90	
10	10	20	30	40	50	60	70	80	90	100	
11											

Wskazówki

- ❑ W pierwszym wierszu procedurze nadawana jest nazwa. Makro zawsze rozpoczyna się słowem kluczowym Sub, po którym podawana jest jego nazwa i ewentualnie parametry.
- ❑ W następnym wierszu zawarty jest warunek, że dalsze czynności będą wykonane tylko wtedy, gdy aktywna komórka znajduje się nie niżej niż w 10. wierszu i nie dalej niż w 10. kolumnie arkusza.
- ❑ W trzech kolejnych wierszach z aktywnej (zaznaczonej) komórki oraz pierwszych komórek kolumny i wiersza dane są pobierane i przypisywane zmiennym. Przypisanie wartości komórek zmiennym ułatwi Ci zapisanie argumentu dla polecenia MsgBox w następnym wierszu kodu. Jak widać, pobranie danych z komórek arkusza nie wymaga żadnych poleceń. Wystarczy operacja przypisania.
- ❑ Po wyświetleniu okna dialogowego (składnię polecenia MsgBox poznasz w dalszej części podręcznika) następuje zamknięcie sekwencji operacji wykonywanych po spełnieniu warunku początkowego (End If).
- ❑ Ostatnie słowo kluczowe informuje o końcu procedury (makra).

Pozostaje jeszcze pytanie: jak sprawić, aby makro było uruchamiane po każdym kliknięciu myszą? Decydują o tym procedury zdarzeniowe, których opis zawarty jest w rozdziale 4. podręcznika.

Zmienne i stałe

Korzystanie ze stałych ma sens wtedy, gdy często stosujesz tę samą wartość w procedurze. Możesz na przykład za pomocą stałej wyrazić część komunikatu często wyświetlanego w oknie dialogowym. Stałe definiuje się za pomocą słowa kluczowego `Const`:

```
Const a = "Wartość komórki wynosi: "  
Const b = 13
```

Niewątpliwą zaletą stałej jest to, że próba jej zmiany w jakikolwiek sposób jest niemożliwa i kończy się wyświetleniem komunikatu o błędzie.

W VBA rozpoznawanych jest kilka typów zmiennych:

- Boolean — zmienna logiczna — przybiera wartości `true` lub `false`.
- Byte — wartości całkowite — przybiera wartości od 0 do 255 (czyli tyle, ile jeden bajt).
- Integer — wartości całkowite — przybiera wartości od -32 768 do 32 767.
- Long — wartości całkowite — przybiera wartości od -2 147 483 648 do 2 147 483 647.



Jeżeli operujesz na adresach całego arkusza, musisz pamiętać, że pojemność zmiennej zadeklarowanej jako `Integer` jest zbyt mała. Arkusz ma bowiem 65 536 wierszy. Jeżeli więc dochodzi do deklaracji zmiennej przechowującej numer wiersza, musisz użyć typu `Long`.

- Single — wartości liczb rzeczywistych — przybiera wartości od $-3,4 \times 10^{38}$ do $3,4 \times 10^{38}$ z dokładnością sześciu cyfr po przecinku.
- Double — wartości liczb rzeczywistych — przybiera wartości od $-1,79 \times 10^{308}$ do $1,79 \times 10^{308}$ (z dokładnością do 14 cyfr po przecinku).
- Currency — wartości kwot pieniężnych — przybiera wartości od $-9,22 \times 10^{11}$ do $9,22 \times 10^{11}$ (z dokładnością do czterech miejsc po przecinku).
- String — zmienna tekstowa — może zawierać tekst do dwóch miliardów znaków.

- ❑ **Date** — data i godzina — może zawierać informacje o czasie od *01.01.100* roku do *31.12.9999* roku, przy czym data 31 grudnia 1899 jest reprezentowana przez wartość 1, 1 stycznia 1900 to wartość 2 itd. Cyfry po przecinku oznaczają — tak jak w arkuszu Excela — części doby, czyli godzinę. Czas przed 31.12.1899 reprezentowany jest przez liczby ujemne.



Zauważ, że VBA — w przeciwieństwie do arkusza Excela — może wykorzystywać daty sprzed 1 stycznia 1900 roku. Należy jednak pamiętać, że nawet prawidłowo obliczonego wyniku sprzed roku 1900 nie uda się wyświetlić w arkuszu roboczym w formacie daty. Musisz wspomóc się formatem tekstowym.

Poniższy przykład wstawi datę 15 lipca 1410 roku do komórki A1 aktywnego arkusza. Niestety, tylko w formacie tekstowym, co oznacza, że nie będziesz mógł wykonywać na tej dacie żadnych obliczeń za pomocą funkcji arkuszowych.

```
Sub data_przed_1900()  
m = DateSerial(1410, 7, 15)  
mr = Str(m)  
Cells(1, 1) = mr  
End Sub
```

- ❑ **Object** — zmienna obiektowa — może zawierać odwołanie do dowolnego obiektu, na przykład *Worksheet*, *Range*, *CommandBar* i wielu innych. Będziemy z niej często korzystać w dalszych ćwiczeniach. Istotne są tutaj dwa fakty:
 - ❑ zazwyczaj deklarujesz odpowiedni *typ* zmiennej obiektowej, a nie samą zmienną;
 - ❑ aby przypisać wartość zmiennej obiektowej, musisz skorzystać ze słowa kluczowego *Set*.

Przykład:

Aby przypisać zmiennej obiektowej arkusz swojego skoroszytu, musisz wpisać:

```
Dim zmienna_obiektowa As Worksheet  
Set zmienna_obiektowa = Sheets("Arkusz1")
```

Aby przypisać zmiennej pasek narzędzi, wpisz:

```
Dim zmienna_obiektowa As CommandBar  
Set zmienna_obiektowa = Application.CommandBars(8)
```

W tym przypadku zmiennej został przypisany pasek narzędzi *Formularze*. (W Excelu 2007 fakt zadeklarowania zmiennej obiektowej zawierającej pasek narzędzi nie ma absolutnie żadnego znaczenia).

Może więc bardziej uniwersalny przykład:

```
Dim zmienna_obiektowa As Chart
Dim zmienna_obiektowa2 As Chart
Set zmienna_obiektowa = Application.Charts("Wykres1")
Set zmienna_obiektowa2 = Application.Charts(1)
```

Jak widać, w powyższych przykładach użyłem od razu deklaracji `Dim zmienna as Worksheet`. Powinniśmy raczej unikać deklaracji `Dim zmienna as Object` i stosować ją tylko wtedy, gdy nie wiemy, jakiego obiektu się spodziewamy. Zauważ, że obiekty możemy identyfikować w kolekcji według nazw `Application.Charts("Wykres1")` lub według ich kolejności `Application.Charts(1)`. Każdy ze sposobów ma swoje zalety i wady. Przekonasz się o tym, pracując nad własnymi projektami

- ❑ `Variant` — to zmienna uniwersalna. Może zawierać zarówno wartość logiczną (`Boolean`) czy łańcuch znaków, jak i datę czy liczbę wielkości `Double`.

Visual Basic nie wymaga deklarowania zmiennych. Jeżeli tego nie zrobisz, program przypisze użytym przez Ciebie zmiennym typ `Variant`.

Po co więc to wszystko?

- ❑ Deklarowanie zmiennych pozwala programowi panować nad błędami wynikającymi z pomyłek (zamierzonych lub nie) przy wprowadzaniu danych przez użytkownika. Nie jest bowiem możliwe przypisanie łańcucha tekstowego zmiennej zadeklarowanej jako na przykład `Date`.
- ❑ Zmienne typu `Variant` rezerwują sobie nawet 20 razy (!) więcej miejsca, niż zajmowałyby zadeklarowana zmienna innego typu. Jest więc nad czym się zastanowić, szczególnie gdy ma się do wykonania kilka milionów obliczeń po każdej zmianie danych.
- ❑ Warto deklarować *wszystkie* zmienne. Wiem z doświadczenia, że zdarza się użycie zmiennej o tej samej nazwie (nadawanej najczęściej intuicyjnie) w tym samym programie do przechowywania różnych danych. Efekty takiego postępowania bywają komiczne tylko wtedy,

gdy masz wielkie poczucie humoru i dużo wolnego czasu. W przeciwnym razie lepiej zajrzeć do obszaru deklaracji zmiennych i wybrać nieużywaną jeszcze nazwę.

Deklarowanie zmiennych i ich zasięg

Zanim rozpoczniesz ćwiczenia z deklaracjami zmiennych, musisz poznać jeszcze kilka ograniczeń dotyczących ich nazw. Nazwa zmiennej:

- ❑ musi rozpoczynać się od litery;
- ❑ nie może być nazwą polecenia, funkcji ani słowa kluczowego;
- ❑ nie może zawierać spacji;
- ❑ może być kombinacją liter i cyfr.

Zmienne lokalne

Zmienne deklaruje się za pomocą słowa kluczowego `Dim` lub `Static`. Różnice między sposobami deklarowania zmiennych wyjaśni poniższe ćwiczenie.

Ć W I C Z E N I E

2.4 Deklarowanie zmiennych lokalnych

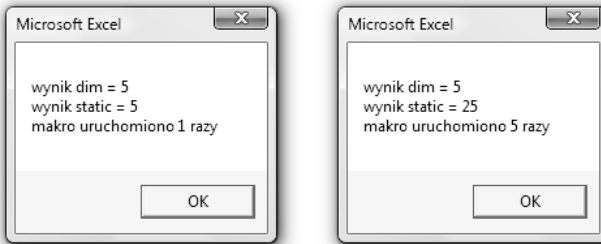
Zadeklaruj zmienne w programie za pomocą słów `Dim` i `Static`. Dodawaj zmienne do uprzednio wyliczonego wyniku i wyświetl go. Uruchom makro kilkakrotnie. Policz, ile razy uruchomiłeś makro.

Rozwiązanie

```
Sub zmienne()  
Dim a, b, wynik_dim As Integer  
Static c, d, e, wynik_static As Integer  
' dodawanie zmiennych dim  
a = 5: b = wynik_dim  
wynik_dim = a + b  
' dodawanie zmiennych static  
c = 5: d = wynik_static  
wynik_static = c + d  
e = e + 1
```

```
MsgBox ("wynik dim = " & wynik_dim & Chr(13) & "wynik static = "
& wynik_static _
& Chr(13) & "makro uruchomiono " & e & " razy")
End Sub
```

Różnice w działaniu sposobów deklarowania zmiennych przedstawia rysunek 2.4.



Rysunek 2.4. Zmiennie `wynik_dim` i `wynik_static` są wynikiem tych samych obliczeń. Jednak różny sposób ich deklarowania powoduje, że po kilku uruchomieniach makra wyniki znacznie się różnią

Wyjaśnienia

- ❑ Pierwsze trzy wiersze kodu to nagłówek procedury i deklaracje zmiennych.
- ❑ Następny wiersz jest komentarzem. Na jego początku znajduje się apostrof, więc zawartość wiersza nie jest analizowana przez program.
- ❑ W piątym wierszu następuje przypisanie wartości zmiennym. Zmienna `a` przyjmuje wartość 5, a zmienna `b` — wartość zmiennej `wynik_dim`. Zmienna `b` przyjmuje wartość 0, ponieważ `wynik_dim` nie jest znany przy pierwszym uruchomieniu makra.
- ❑ Stąd w szóstym wierszu wartość zmiennej `wynik_dim` wynosi $5+0$, czyli 5 — co jest widoczne w oknie informacyjnym na rysunku 2.4.
- ❑ W wierszach 7. – 9. powyższe czynności są powtarzane w stosunku do kolejnych zmiennych.
- ❑ W wierszu 10. zmienna `e` jest zwiększana o 1 po każdym wykonaniu programu. Wartość początkowa zmiennej `e` nie została ustalona, a jej typ to `Integer`, więc program przyjął dla niej początkową wartość zero.

- ❑ Przy kolejnym uruchomieniu makra zmiennym zadeklarowanym słowem kluczowym `dim` zostają przywrócone domyślne wartości początkowe. A zatem wartości zmiennych `b` i `wynik_dim` ponownie wynoszą zero.
- ❑ Inaczej jest ze zmiennymi zadeklarowanymi za pomocą słowa `static`. Ich wartości nie są zerowane. Przy drugim uruchomieniu możliwe jest więc powiększenie „licznika” `e` do 2, a zmienna `d` przyjmie wówczas wartość 5, obliczoną w czasie poprzedniego uruchomienia makra. W związku z tym wartość zmiennej `wynik_static` po drugim uruchomieniu makra będzie wynosić 10.
- ❑ Wartości omawianych zmiennych po pięciu uruchomieniach makra widoczne są na rysunku 2.4.
- ❑ Sposób wyświetlania komunikatów za pomocą polecenia `MsgBox` zostanie wyjaśniony w dalszej części podręcznika.

W zależności od miejsca, w którym dokonasz deklaracji, zmienne będą miały różny zasięg. Jeżeli zmienne zadeklarowano wewnątrz makra (funkcji), będą one dotyczyć tylko tego makra (funkcji) i poza nim nie będą odczytywane. Zmienne takie nazywamy *zmiennymi lokalnymi*. To wszystkie zmienne, które deklarowałeś dotychczas.

Zmienne modułu i zmienne publiczne

Budując bardziej złożony program, dojdiesz do wniosku, że w celu zmniejszenia objętości kodu dobrze jest wydzielić często powtarzane czynności (na przykład wyświetlanie komunikatów) przez umieszczenie ich w osobnych programach, uruchamianych przez inne makra tylko wtedy, gdy jest to potrzebne.

Wyjaśni to poniższe ćwiczenie.

Ć W I C Z E N I E

2.5 Deklarowanie zmiennych publicznych

Utwórz dwa makra: jedno odczytujące dane z arkusza, a drugie wyświetlające odczytane dane w oknie komunikatu. Przekaż wartości między makrami za pomocą zmiennych.

Rozwiązanie

```
Sub pobierz_dane()  
wart_komórki = Cells(1, 1)  
tekst = " Wartość komórki A1 wynosi "  
komunikat  
End Sub  
Sub komunikat()  
MsgBox (tekst & wart_komórki)  
End Sub
```

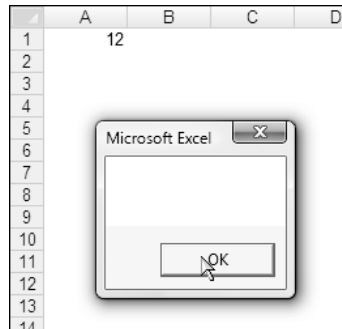
Wyjaśnienia

- ❑ Makro *pobierz_dane* odczytuje wartość komórki A1 i przypisuje ją zmiennej *wart_komórki*.
- ❑ Dodatkowo ustalana jest wartość zmiennej *tekst*, która jest wykorzystywana przez makro *komunikat*.
- ❑ Po nadaniu wartości wykonywane jest makro *komunikat*, mające na celu wyświetlenie na ekranie wartości zmiennych.

Wpisz do komórki A1 dowolną wartość i uruchom makro *pobierz_dane*. Jego efekty ilustruje rysunek 2.5.

Rysunek 2.5.

Mimo że składniowo wszystko jest w porządku, nie takiego efektu się spodziewaliśmy



W oknie komunikatu nie zostały wyświetlone żadne informacje, ponieważ nie zadeklarowano zmiennych na poziomie modułu. VBA uznał zatem, że ich wartości obowiązują tylko w obrębie makra, w którym zostały użyte.

Aby możliwe było przekazywanie wartości zmiennych między makrami (funkcjami), musisz je zadeklarować na poziomie modułu. Robi się to na początku modułu, przed pierwszym słowem *sub* lub *function*.

Tak zadeklarowane zmienne nazywamy (jak nietrudno się domyślić) *zmiennymi modułu*. Kompletna jego zawartość powinna więc wyglądać tak:

```
Option Explicit
Dim tekst, wart_komórki
Sub pobierz_dane()
wart_komórki = Cells(1, 1)
tekst = "Wartość komórki A1 wynosi "
wyświetl
End Sub
Sub wyświetl()
MsgBox (tekst & wart_komórki)
End Sub
```

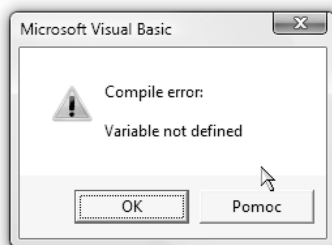
Wskazówki

- ❑ Makro *wyświetl* może oczywiście być uruchamiane przez użytkownika, z tym że nie ma on możliwości podania wymaganych przez nie zmiennych. Najbardziej funkcjonalnym jego wykorzystaniem będzie ustalanie wartości zmiennych w poszczególnych makrach i wywoływanie ich nazw tak, jak zostało to przedstawione w powyższym przykładzie. Masz więc raz napisane makro, które możesz przywoływać w dowolnym miejscu programu.
- ❑ Polecenie `Option Explicit` wymusza deklarowanie wszystkich zmiennych. Jeżeli podczas wykonywania makra zostanie wykryta niezadeklarowana zmienna, spowoduje to błąd programu (rysunek 2.6).

Rysunek 2.6.

Polecenie

Option Explicit wymusza porządek w kodzie Twojego programu. Żadna niezadeklarowana zmienna nie ma racji bytu



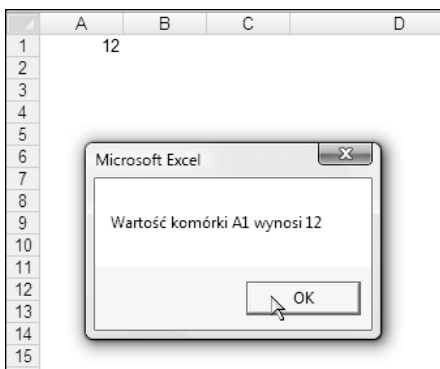
- ❑ Zauważ, że w deklaracji nie podałem typu zmiennych. Program domyślnie przypisał im typ `Variant`. W tym wypadku deklaracja miała na celu określenie nie typu, lecz zasięgu zmiennych. Teraz ich wartości będą odczytywane przez wszystkie makra i funkcje umieszczone w module.
- ❑ Możliwe jest także zadeklarowanie zmiennych modułu za pomocą słowa kluczowego `private`. Zasięg zmiennych jest taki sam: będą one dostępne w module, w którym zostały użyte.

`Private tekst, wart_komórki`

- ❑ Jeżeli wszystko poszło dobrze, na ekranie powinien pojawić się w końcu komunikat jak na rysunku 2.7.

Rysunek 2.7.

Na zakończenie obraz cieszący oko każdego projektanta. Program działa. Przypniesz, że zabawa z VBA nie jest specjalnie trudna?



Powyższe makra będą przekazywać między sobą wartości pod warunkiem, że zostały umieszczone w tym samym module. Często jednak zdarza się, że — dla zwiększenia przejrzystości — procedury (podprogramy) wywoływane przez inne programy umieszcza się w oddzielnym module. Aby zapewnić przenoszenie wartości zmiennych pomiędzy wszystkimi elementami programu, należy zadeklarować je (w dowolnym module) za pomocą słowa kluczowego `Public`.

`Public tekst, wart_komórki`

Tak zadeklarowane zmienne nazywamy *zmiennymi publicznymi*.

Zmienne tablicowe

Pomyśl, że chciałbyś przypisać kolejnym zmiennym wartości z komórek od A1 do A1000. Możesz to oczywiście zrobić, wymyślając 1000 nazw zmiennych i wpisując 1000 wierszy kodu, ale z równym powodzeniem możesz spróbować ogolić się maszynką do strzyżenia owiec.

Lepiej będzie użyć w tym celu 1000-elementowej *tablicy jednowymiarowej*, której elementy mają tę samą nazwę i następujące po sobie wyróżniki liczbowe. Te wyróżniki pozwolą nam na szybkie przypisanie wartości zmiennym za pomocą pętli.

Ć W I C Z E N I E

2.6 Deklarowanie zmiennych tablicowych

Przypisz zmiennym wartości z komórek A1 do A1000 arkusza Excela.

Rozwiązanie

```
Option Explicit
Option Base 1
Sub przypisz_wartości()
Dim x As Integer
Dim tablica(1000) As Integer
For x = 1 To 1000
    tablica(x) = Cells(x, 1)
Next x
End Sub
```

Wyjaśnienia

- ❑ Option Base 1 określa, że pierwszy element tablicy będzie miał indeks 1 (zamiast domyślnego zero).
- ❑ Option Explicit powoduje konieczność deklarowania każdej zmiennej, nawet tej używanej w pętli jako licznik. Należy o tym pamiętać.
- ❑ Tablicę musisz zadeklarować bez względu na to, czy wstawiłeś na początku polecenie Option Explicit, czy nie. Jej zasięg podlega zasadom opisanym wcześniej w tym rozdziale.

- ❑ Zmienna wchodząca w skład tablicy jednowymiarowej ma postać `n(liczba)`, gdzie `liczba` określa miejsce danej w tablicy. Jeżeli chciałbyś wyświetlić zawartość 265. miejsca w tablicy utworzonej w powyższym przykładzie, wpisz:

```
MsgBox tablica(265)
```

- ❑ Elementy tablicy zadeklarowanej w sposób przedstawiony w tym przykładzie mają wyróżniki od 1 wzwyż, co nie zawsze bywa korzystne. Możliwe jest także zadeklarowanie tablicy w postaci:

```
Dim tablica (501 to 1500)
```

W tym wypadku elementy tablicy będą ponumerowane od 501 do 1500.

Tablice danych mogą mieć także więcej niż jeden wymiar — można wówczas powiedzieć, że odzwierciedlają zakres kilku kolumn i kilku wierszy arkusza, lub nawet kilku arkuszy.

Ć W I C Z E N I E

2.7 Deklarowanie tablic wielowymiarowych

Zadeklaruj tablicę dwuwymiarową dla zakresu komórek A1 do F100 i trójwymiarową dla takiego samego zakresu w trzech kolejnych arkuszach.

Rozwiązanie

Tablica dwuwymiarowa:

```
Dim tablica(5, 99) As Integer
```

lub:

```
Dim tablica(1 To 6, 1 To 100) As Integer
```

Tablica trójwymiarowa:

```
Dim tablica(2, 5, 99) As Integer
```

lub:

```
Dim tablica(1 To 3, 1 To 6, 1 To 100)
```

Wyjaśnienia

- ❑ Rozmiar tablicy dwuwymiarowej to sześć kolumn i 100 rzędów. Pierwsza wartość ma współrzędne 0,0 — liczby użyte w deklaracji to uwzględniają.
- ❑ W drugim sposobie rozwiązania zadania indeks początkowy i końcowy zarówno dla kolumny, jak i wiersza został narzucony. Pierwszy element tablicy będzie miał współrzędne 1,1, a ostatni — 6,100.
- ❑ Dopuszczalne są mieszane sposoby deklaracji tablic.

```
Dim tablica(2, 1 To 6, 99)
```

Za pomocą polecenia `Dim` możesz deklarować tablicę, której wymiar jest od początku znany. Jeżeli ilość danych w tabeli nie jest znana w momencie rozpoczęcia procedury, możesz wstępnie zadeklarować tablicę, nie podając jej wielkości:

```
Dim tablica() as String
```

a następnie po uruchomieniu makra skorzystać z instrukcji `ReDim`:

```
Sub makro()  
    ilość = Cells(3, 4) ' pobiera wielkość tablicy z komórki aktywnego arkusza  
    ReDim tablica (ilość) ' określa wielkość tablicy za pomocą zmiennej  
    ...  
End Sub
```

Taki sposób działania opisuje ćwiczenie 2.10.

Co będzie, jeśli?

Bardzo ważną konstrukcją w językach programowania są **instrukcje warunkowe**. Pozwalają one na wykonywanie określonych czynności w zależności od sytuacji, położenia kursora, wartości zmiennej czy też każdego innego zdarzenia zachodzącego w momencie wykonywania takiej instrukcji przez program. Na początek poznamy najbardziej intuicyjną strukturę: `If...Then...Else`.