

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 2002/XP PL. Ćwiczenia zaawansowane

Autor: Mirosław Lewandowski

ISBN: 83-7361-204-1

Format: B5, stron: 178



Najpopularniejszy arkusz kalkulacyjny – Microsoft Excel – posiada ok. 300 funkcji gotowych do wykorzystania w sytuacjach, w których mogą przydać się obliczenia. Jakby tego było mało, mamy do dyspozycji potężne narzędzie jakim jest Visual Basic for Applications (VBA) – przejrzysty i prosty język programowania, zoptymalizowany specjalnie pod kątem rozszerzania możliwości innych aplikacji.

Celem tej książki jest nauka wykorzystania VBA w połączeniu z Excelem. Założono więc, że nie chcesz zgłębiać teorii poszczególnych składników języka, lecz interesuje Cię strona praktyczna Twoich działań. Dlatego też każdy przykład został przez autora szeroko omówiony, zarówno od strony algorytmu, jak i wykorzystanego kodu.

Dowiesz się jak:

- Zautomatyzować rutynowe czynności
- Rejestrować makrodefinicje
- Korzystać z danych zawartych w skoroszytach
- Tworzyć własne funkcje
- Komunikować się z użytkownikiem
- Zamieniać liczby na format „słownie”
- Wspomagać pracę Excela
- Losować liczby



Spis treści

Zapytaj Autora przez Internet!	5
Rozdział 1. Automatyzacja często powtarzanych zadań	7
Wprowadzenie	7
Dla początkujących — rejestrowanie makr	7
Uruchamianie zapisanych projektów	9
Szybkie sortowanie danych	13
Rozdział 2. Podstawy	21
Wymiana danych między VBA a skoroszytem. Zmienne i stałe	21
Czytanie i umieszczanie danych	21
Zmienne i stałe	24
Deklarowanie zmiennych i ich zasięg	26
Zmienne lokalne	26
Zmienne modułu i zmienne publiczne	28
Zmienne tablicowe	30
Co będzie, jeśli?	32
Pętle	35
For Each	36
Do...Loop	40
Idź do, idź i wróć	43
Dialog z użytkownikiem	45
Okna komunikatów	45
Okna dialogowe	49
Formularze	50
Obsługa błędów	54
Makro a funkcja	56
Tworzenie funkcji użytkownika	56
Opisywanie funkcji użytkownika	58
Rozdział 3. Przykłady	61
Liczby słownie	61
Wygląd zależny od warunków	65
Nawigacja między arkuszami	67
Wspomaganie pracy Excela	69
Jednorazowe losowanie	69
Autostart	71
Rozdzielanie tekstu	72
Poszukiwanie dni tygodnia	74
Poszukiwanie nazw w skoroszytcie	76

Hiperłącza.....	77
Dodawanie polecenia do menu	82
Matematyka dla dziewięciolatka.....	84
Tworzenie dodatku Kalendarz w pasku narzędzi.....	103
Generowanie dźwięku	117
Obliczanie głębokości	120
Arkusz ofert.....	130
Kółko i krzyżyk.....	134
Rozdział 4. Dodatki.....	151
Okno edytora VBA.....	151
Procedury zdarzeniowe	154
Właściwości formantów formularza	162

Rozdział 2.

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, przypisywania zmiennej czy wyświetlenia okien dialogowych. 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://excel.vip.interia.pl>.

Wymiana danych między VBA a skoroszytem. Zmienne i stałe

Czytanie i umieszczanie danych

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

Ćwiczenie 2.1.

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 dokonywane jest w poniższym wierszu kodu:

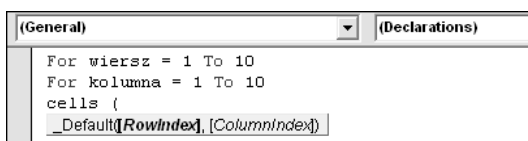
```
Cells(wiersz, kolumna) = wiersz * kolumna
```

Właściwość *Cells*, określająca adres komórki, posiada 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 (patrz rysunek 2.2), który sam podpowie, czego od Ciebie oczekuje.

Rysunek 2.2.

Podpowiedzi
edytora VBA bywają
bardzo pomocne



Ćwiczenie 2.2.

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 nam już poleceń Cells. Nasze makro mogłoby więc wyglądać tak:

```
Sub wypełnij()  
    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 do ich umieszczania. Przecwiczymy to na bardziej użytecznym przykładzie.

Ćwiczenie 2.3.

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

- ❖ zaznaczonej komórki;
- ❖ komórki z pierwszego wiersza aktywnej kolumny;
- ❖ komórki z pierwszej kolumny aktywnego 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.

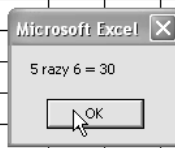
Rozwiązanie

```
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
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				60
7	7	14	21	28	35	42				70
8	8	16	24	32	40	48				80
9	9	18	27	36	45	54				90
10	10	20	30	40	50	60				100

**Wskazówki**

- ❖ W pierwszym wierszu procedury 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 niż w 10. wierszu i nie dalej niż w 10. kolumnie arkusza.
- ❖ W kolejnych trzech 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 kolejnym 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 czwartym rozdziale podręcznika.

Zmienne i stałe

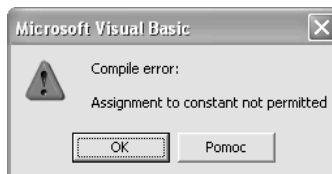
Korzystanie ze stałych ma sens wtedy, gdy często stosujesz tę samą wartość w procedurze. Możesz na przykład za pomocą stałej wyrazić część komunikatu często wyświetlanego w oknie dialogowym. Stałe definiuje się za pomocą słowa kluczowego Const:

```
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 (patrz rysunek 2.4).

Rysunek 2.4.

*Próba zmiany
wartości zadeklarowanej
jako stała niesie
opłakane skutki*



W VBA rozpoznawane jest kilka typów zmiennych:

1. Boolean — zmienna logiczna — przybiera wartości true lub false.
2. Byte — wartości całkowite — przybiera wartości od 0 do 255 (czyli tyle, ile jeden bajt).
3. Integer — wartości całkowite — przybiera wartości od -32 768 do 32 767.
4. 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.

5. Single — wartości liczb rzeczywistych — przybiera wartości od $-3,4 \times 10^{38}$ do $3,4 \times 10^{38}$ z dokładnością 6 cyfr po przecinku.
6. 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).
7. 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).
8. String — zmienna tekstowa — może zawierać tekst do 2 mld znaków.
9. Date — data i godzina — może zawierać informacje o czasie od 01.01.100 roku do 31.12.9999, 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.

10. 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 Twojego 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*. Jak widać, użyłem od razu deklaracji Dim zmienna as Worksheet. Należy raczej unikać deklaracji Dim zmienna as Object i stosować ją tylko wtedy, gdy nie wiadomo, jakiego obiektu się spodziewamy.

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

Visual Basic nie wymaga deklaracji zmiennych. Jeżeli nie dokonasz tej czynności, 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 i więcej razy (!) miejsca niż zajmowałaby zadeklarowana zmienna innego typu. Jest się więc nad czym zastanowić, szczególnie gdy nie pracujesz na superkomputerze, w tle ściągasz kilka plików z Internetu, korzystasz z czata, a z głośników sący się muzyczka z plików mp3.
- ❖ 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 wypadku 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 deklaracji zmiennych wyjaśni poniższe ćwiczenie.

Ćwiczenie 2.4.

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.

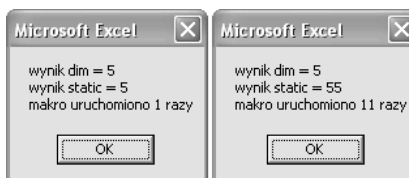
Rozwiązanie

```
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 sposobu deklaracji przedstawia rysunek 2.5.

Rysunek 2.5.

Zmienne wynik_dim i wynik_static są wynikiem tych samych obliczeń. Jednak sposób deklaracji powoduje, że po kilku uruchomieniach makra wyniki znacznie się różnią

**Wyjaśnienia**

- ❖ Pierwsze trzy wiersze kodu to nagłówki 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, bowiem wynik_dim nie jest znany przy pierwszym uruchomieniu makra.
- ❖ Stąd w wierszu szóstym wartość zmiennej wynik_dim wynosi 5+0, czyli 5 — co jest widoczne w oknie informacyjnym na rysunku 2.3.
- ❖ 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 jedenastu uruchomieniach makra widoczne są na rysunku 2.5.
- ❖ 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 tegoż 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), umieszczając je w osobnych programach, uruchamianych przez inne makra tylko wtedy, gdy jest to potrzebne.

Wyjaśni to poniższe ćwiczenie.

Ćwiczenie 2.5.

Utwórz dwa makra: jedno odczytujące dane z arkusza, a drugie wyświetlające odczytane dane w oknie komunikatu. Przekaz 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
```

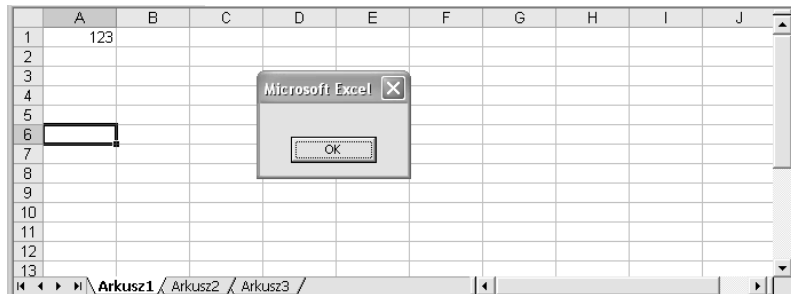
Wyjaśnienia

- ❖ 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 2.6.

Rysunek 2.6.

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. Dokonuje się tego na początku modułu, przed pierwszym słowem *sub* lub *function*. Tak zadeklarowane zmienne nazywamy (jak nietrudno się domyślić) *zmiennymi modułu*. Kompletna zawartość modułu powinna więc wyglądać tak:

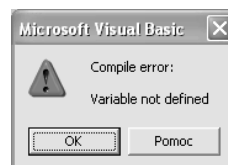
```
Option Explicit
Dim tekst, wart_komórki
Sub pobierz_dane()
wart_komórki = Cells(1, 1)
tekst = "Wartość komórki A1 wynosi "
wyświetl
End Sub
Sub wyświetl()
MsgBox (tekst & wart_komórki)
End Sub
```

Komentarz

- ❖ 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 nie zadeklarowana zmienna, spowoduje to błąd programu (patrz rysunek 2.7).

Rysunek 2.7.

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



- ❖ Zauważ, że w deklaracji nie podałem typu zmiennych. Program domyślnie przypisał im typ `Variant`. W tym wypadku deklaracja nie miała na celu określenie 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`

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*.