Programování

RNDr. Jana Linhardová

Obsah

1. Základní pojmy

1.1. Vymezení pojmu algoritmu

Pojem „algoritmus“ pochází ze začátku 9.stol. z Arábie. Arabský matematik al Chwárizmí napsal v letech 800 - 825 knihu postupů pro počítání s čísly. Kniha se v latinském překladu jmenovala „Algoritmi decit“, tedy „Tak praví al Chwárizmí“

Všichni máme s konkrétními algoritmy bohaté zkušenosti. Denně provádíme celou řadu činností zcela mechanicky, podle návodů, které jsme se dříve naučili. Algoritmus je vlastně takový návod, podle kterého se máme řídit, abychom vyřešili každou úlohu daného typu.

Při programování se jím myslí teoretický princip řešení problému, který bude následně přesně napsán v konkrétním programovacím jazyce. Obecně se ale algoritmus může objevit v jakémkoli jiném odvětví. Jako jistý druh algoritmu se může chápat i např. kuchyňský recept.

Algoritmus je konečná posloupnost elementárních příkazů, jejichž provádění umožňuje pro každá přípustná vstupní data mechanickým způsobem získat po konečném počtu kroků příslušná výstupní data.

Vlastnosti algoritmu :

Hromadnost

Algoritmus je navržen pro řešení celé třídy úloh, jednotlivé úlohy jsou určeny svými vstupními daty. (Nenavrhujeme algoritmus pro úlohu sečti čísla 5 a 8, ale navrhujeme algoritmus pro sčítání dvou čísel. Konkrétní úloha je dána vstupními daty např. 5 a 8)

Konečnost

Algoritmus je konečný, když pro přípustná vstupní data po konečném počtu kroků skončí

Správnost

Těžko se dokazuje. Často dokazujeme spíše nesprávnost tak, že najdeme přípustná vstupní data, pro něž algoritmus dá špatný výsledek.

Parciální (částečná) správnost

Jestliže výpočet pro přípustná vstupní data skončí, pak skončí se správným výsledkem
Všimněte se, že částečná správnost & konečnost → správnost

Korektnost

Algoritmus je korektní, jestliže při jeho návrhu nebyla opomenuta žádná z možností, které mohou během výpočtu nastat.

Př. Popiš algoritmus přecházení přes ulici. (Jaká jsou přípustná vstupní data?)
Př. Popiš algoritmus, jak si uvařit čaj.
Př. Popiš algoritmus pro sčítání dvou čísel
Př. Algoritmus nalezení extrému v posloupnosti
Př. Eukleidův algoritmus nalezení největšího společného dělitele dvou přirozených čísel

1.2. Programování

Programování je řešení úloh pomocí počítače

Programování je dovednost. A nelze ji tedy získat jinak, než vlastními pokusy !!!

1.3. Programovací jazyk

Pro zápis algoritmů může posloužit přirozený jazyk, případně doplněný matematickými symboly. Tento zápis však nelze předložit počítači. Počítač "rozumí" strojovému kódu.
Je třeba nalézt takovou formu zápisu, které by rozuměl počítač a přitom byl bližší přirozenému jazyku než strojový kód.

Programovací jazyk je určen pro zápis programu. Je to jazyk umělý, formální, jednoduchý.

Programovací jazyky nižší - jazyk je v podstatě ekvivalentem strojového kódu - Assembler
Programovací jazyky vyšší - více se blíží přirozenému jazyku
Syntaxe – pravidla zápisu
Sémantika - význam

1.4. Zdrojový text programu

Zdrojový text programu je zápis programu v programovacím jazyku

1.5. Překladače programovacích jazyků

I když je programovací jazyk velmi strohý, počítač mu „nerozumí“. Počítač „rozumí“ strojovému kódu.
Překladače jsou programy, které umí číst zdrojový text v daném programovacím jazyku a překládat jej do strojového kódu.

2. Základní prvky jazyka

2.1. Symboly

Písmena, číslice, spec. symboly (+ - * < > …..)

2.2. Klíčová (vyhrazená) slova

Tvoří kostru programu, jsou to speciální slova, mající určitý speciální význam, lze je použít pouze v tomto významu. Jsou to většinou anglická slova nebo jejich zkratky.
Př. If, Then, Else, Case, Do, Loop, While, Until, …

2.3. Konstanty

Číselné … 12 3.14 0
Řetězcové … "Ahoj !" "Jan Novák" "583 21 35 48"
Pojmenované konstanty:

Některé jsou součástí jazyka VBNewLine, VBYesNo, VBRed, VBBlue…
Vlastní konstanty si může programátor deklarovat deklaračním příkazem Const
Např. Const Pi = 3.14
Const Pozdrav = "Ahoj "

2.4. Proměnné

Proměnná je vyhrazené místo v paměti počítače sloužící k uchování hodnoty v po dobu běhu programu. Hodnota proměnné se může v průběhu programu měnit.

Deklarace proměnné

Chceme-li v programu používat proměnnou, je vhodné o tom počítač uvědomit. K tomuto účelu slouží deklarace proměnné.
Deklarací proměnné určíme její typ a identifikátor. Deklarací je proměnné přidělena (alokována) paměť.
Ve Visual Basicu je po deklaraci hodnota proměnné inicializována - je jí přiřazena počáteční hodnota.

Deklarační příkaz

Dim *identifikator_promenne* **As Typ**

Př. **Dim** Pom **As** Integer
Typ určuje velikost alokované paměti pro proměnnou, způsob reprezentace a obor hodnot. *Identifikátor_proměnné* slouží k pojmenování proměnné
Zvykneme si psát deklarační příkazy pro všechny potřebné proměnné před ostatní příkazy

Implicitní deklarace

Specialitou VisualBasicu je možnost implicitní deklarace, což je deklarace prvním použitím. Pokud tedy nepoužijeme deklarační příkaz, bude proměnná deklarována automaticky při svém prvním výskytu. Tento způsob je ale obvykle zdrojem těžko odhalitelných chyb, proto budeme vždy deklarovat explicitně – deklaračním příkazem.
Uvedením Option Explicit na začátku kódu potlačíte možnost implicitní deklarace.

Pravidla pro tvorbu identifikátorů

Identifikátor

- Může být tvořen pouze alfanumerickými znaky (jen písmena a číslice) a znakem _ (podtržítko)
- Musí začínat písmenem
- Musí být různý od klíčových slov
- Délka max. 256 (40 pro identifikátory podprogramů) znaků
- Nerozlišují se velká a malá písmena (není key sensitive)
  Ahoj, AHOJ, aHoj je totéž
- Nesmí obsahovat mezeru

Doporučení:
Nepoužívejte diakritiku
Vytvářejte mnemotechnické identifikátory

Příklady správných identifikátorů:

AB S1 Pom_prom CisloAutobusu

Příklady špatných identifikátorů:

5AB Xy#Z

2.5. Datové typy

Datové typy určují :

Způsob reprezentace
Velikost vyhrazené paměti
Obor hodnot

2.6. Základní datové typy

Typ Integer

Celá čísla
Velikost paměti 2B (= 16b)
-32768 až 32767 (2^{15} = 32768, 1b se využije pro znaménko)

Typ Long

Celá čísla
Velikost paměti 4B, rozsah hodnot $\pm 2^{31}$

Typ Single

Desetinná čísla
Velikost paměti 4B
Konstanty se píší s desetinnou tečkou 1.24 0.18

Typ Double

Desetinná čísla (přesnější)
Velikost paměti 8B

Typ String

Řetězec znaků (kódovaných pomocí ASCII)
Velikost paměti dle aktuální hodnoty (max 65400 znaků u 16-b apl.)
Řetězcová konstanta se zapisuje do uvozovek

Typ Boolean

Slouží k uchování dvoustavových logických hodnot
Velikost paměti 2B
Konstanty True a False (klíčová slova)

Typ Variant

Velikost paměti 16B
Obecný datový typ, který může obsahovat libovolný datový typ
Specifikum Visual Basicu
Neuvedete-li typ při deklaraci proměnné, bude automaticky Variant – nedělejte to! Chcete-li mít proměnnou typu Variant, měli byste k tomu mít důvod (zatím jej nemáte).

2.7. Výrazy

Elementární výrazy

- **Konstanta** 5 "Ahoj"
- **Proměnná** (identifikátor)
  Hodnotou je hodnota této proměnné
- **Volání funkce**
  Hodnotou je výsledek volání této funkce

Složitější výrazy

Vytváří se pomocí operátorů a závorek.

Aritmetické operátory

-	unární minus
*	násobení
\	celočíselné dělení
/	dělení
Mod	zbytek po celočíselném dělení
+	sčítání
-	odčítání

Použijí se pro operace s číselnými výrazy, hodnota výsledného výrazu bude také číslo.
Operátory násobení mají vyšší prioritu než operátory sčítání. Pořadí provádění operací lze změnit závorkami. Závorky můžeme do sebe libovolně vnořovat. Lze použít pouze kulaté závorky.
Př. A * (I + 1) – ((B + 3) Mod 2) A a B jsou proměnné některého číselného typu

Příklady použití aritmetických operátorů:

Výraz	Výsledek
4 + 5	9
4 * 5	20
4 / 5	0.8
15 \ 6	2 (výsledek celočíselného dělení)
15 mod 6	3 (zbytek po celočíselném dělení)
(3 * (4 – 2)) + 1	7

Relační operátory

=
<>

<
>
<=
>=

Slouží k určení platnosti relace (vztahu, porovnání) mezi dvěma výrazy (operandy). Operandy by měly být odpovídajících typů (např. oba číselné nebo oba řetězcové).
Výraz vytvořený pomocí relačních operátorů je typu boolean - vztah buď platí - hodnota výrazu je True, nebo neplatí - hodnota výrazu je False.
Pro porovnání řetězcových operandů se bere v úvahu nastavení způsobu porovnání příkazem Option Compare

Option Compare{Binary|Text|Database}
Binary – znaky v řetězci porovnány s ohledem na jejich kód (jako čísla)
Text – textové porovnání bez ohledu na malá a velká písmena (podle abecedy)
Standardně nastaveno Binary

Příkaz Option Compare lze zapsat jen jednou v deklarační části modulu (nebo formuláře)

Logické operátory

Not	negace
And	logický součin
Or	logický součet

Logické operátory jsou uvedeny v pořadí podle priorit.
Jako operandy budeme používat pouze výrazy typu boolean. Výraz vytvořený pomocí logických operátorů je také typu boolean.

Př. Nechť B1, B2 a B3 jsou výrazy typu boolean, A, B a C jsou číselné výrazy.
B1 = B2 And B3
A > B And B > C
A <> 12 Or B < C And C = 0 (Pro větší přehlednost lze závorkovat)
(A <> 12) Or ((B < C) And (C = 0))

Priority mezi jednotlivými typy operátorů jsou v pořadí

aritmetické operátory
relační operátory
logické operátory.

Řetězcový operátor

&	řetězení (spojení řetězců)

Př. Nechť S a S1 jsou řetězcové proměnné
“Abcd ” & S & “ xyz”
S1 & S & “ ” & 5 & “.”

2.8. Příkazy

Příkaz přiřazení

P = V

P identifikátor proměnné
V výraz (jeho typ by měl odpovídat typu proměnné P)
= symbol pro přiřazení

Provedení příkazu přiřazení

Vyhodnotí se hodnota výrazu V, tato hodnota se vloží do proměnné P
Př.

```
Dim I As Integer
Dim A As Integer
Dim B As Integer
Dim Pom As Integer
Dim S As String
Dim S1 As String

I = 5                              Do proměnné I je vložena hodnota 5
I = I + 1                          Vyhodnotí se výraz I+1 a jeho hodnota je vložena
                                   do proměnné I (v proměnné I bude po provedení
                                   tohoto příkazu hodnota 6)
Pom = A * (I + 1) - ((B + 3) Mod 2)

S = "Jak se máš "
S1 = S & "Pepo"
S1 = S1 & " ?"          V proměnné S bude nyní hodnota "Jak se máš Pepo ?"
```

Další příkazy

Volání standardních nebo obecných podprogramů typu Sub.
Př. Print
Msgbox
Randomize

Pořadí provádění příkazů

Příkazy v programu jsou prováděny v tom pořadí, jak jsou za sebou napsány. (Existují i výjimky, ale ty nás zatím nemusí zajímat.)
VB je řádkově orientovaný programovací jazyk. To znamená, že každý příkaz musí být zapsán na samostatném řádku.
Je-li třeba zapsat další příkaz na tentýž řádek, musíme příkazy oddělit dvojtečkou
Př. A = 17 **:** B = B+1 **:** Print C
Je-li řádek příliš dlouhý, lze jej pro přehlednost rozdělit. Na místo, kde může být mezera, vložíme mezeru a podtržítko, pak pokračujeme v zápisu příkazu na novém řádku.

2.9. Komentáře

Zvykněte si psát do zdrojového kódu komentáře – lepší čitelnost, srozumitelnost i po čase
Překladač komentáře ignoruje
Řádek, na kterém je komentář, začíná Rem nebo ‘

3. Vývojové prostředí VB

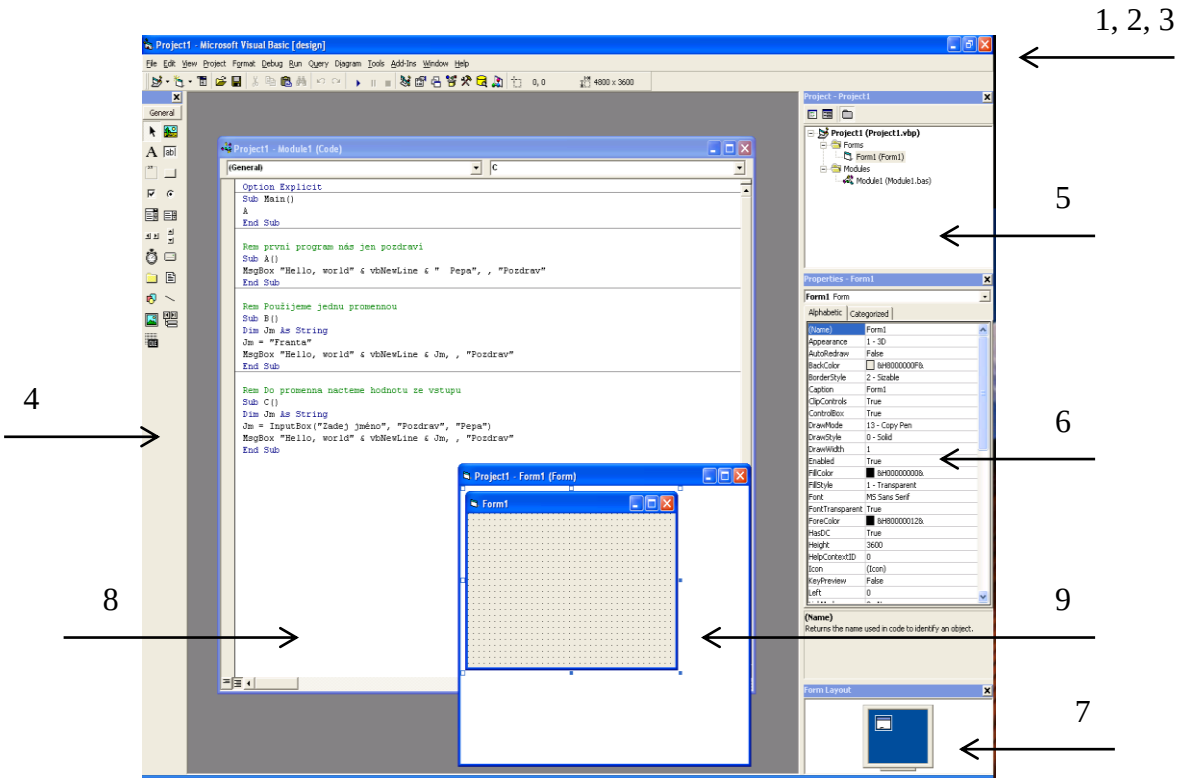
Vývojové prostředí VB se podobá jiným aplikacím ve Windows, které již znáte.
Obsahuje obvyklé ovládací prvky:

1. Titulkový pruh
2. Nabídkovou lištu (menu bar)
3. Panel nástrojů (Toolbar)

Obsahuje také speciální ovládací prvky pro návrh aplikací ve VB:

4. Souprava nástrojů (Toolbox)
5. Průzkumník projektu (Project Explorer)
6. Okno vlastností ovládacích prvků (Properties Window)
7. Okno rozvržení formuláře (Form Layout)
8. Okno kódu (Code Window)
9. Okno návrhu formuláře (Form Designer)

Pokud některé z oken nevidíte, máte k dispozici příkaz k jeho zobrazení v nabídce ***Wiew*** z nabídkové lišty.



Při ladění budeme mít k dispozici další okna (např. Immediate, Watch, …)

obrázek 1

Automatické pomůcky při psaní kódu

Automatická kontrola syntaxe - po opuštění řádku v editoru kódu
Rozlišení pomocí barevného písma
- Modrá – klíčová slova
- Zelená – komentáře
- Červená – chybná syntaxe
- Černá - ostatní

Nabídka parametrů - při zápisu procedur se automaticky nabízejí parametry
Automatické formátování příkazů
Jednotný zápis identifikátorů (Velmi užitečné, vyzkoušejte)

4. Příkazy větvení

Řekli jsme, že příkazy zapsané ve zdrojovém kódu programu se vykonávají v tom pořadí, v jakém jsou za sebou napsány. Při běhu programu ale často potřebujeme, aby se podle platnosti nějaké podmínky výpočet programu rozvětvil. K tomu slouží podmíněný příkaz.

4.1. Úplný podmíněný příkaz

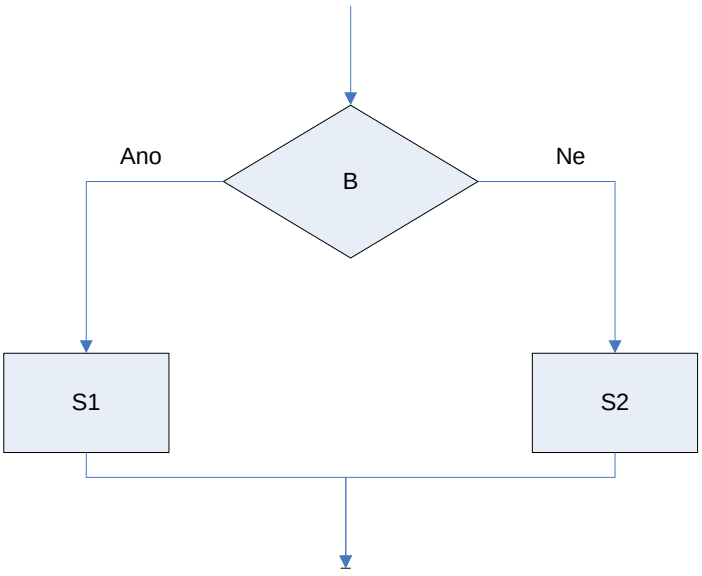
Syntaxe:
První způsob zápisu:

```
If B Then S1 Else S2
```

Druhý (častější) způsob zápisu:

```
If B then
  S1
Else
  S2
End If
```

Kostru příkazu tvoří klíčová slova - pro zvýraznění jsou podtržena
B je podmínka - výraz typu Boolean
S1 a S2 jsou příkazy nebo posloupnosti příkazů.
První způsob zápisu použijeme tehdy, pokud S1 a S2 je vždy jeden příkaz. Druhý způsob zápisu použijeme, jestliže S1 nebo S2 je několik příkazů za sebou.
Vykonání příkazu:
Vyhodnotí se podmínka B. Je-li podmínka splněna (její hodnota je True), vykoná se příkaz S1, jinak (podmínka B není splněna, její hodnota je False) se vykoná příkaz S2.
Vývojový diagram úplného podmíněného příkazu:



4.2. Neúplný podmíněný příkaz

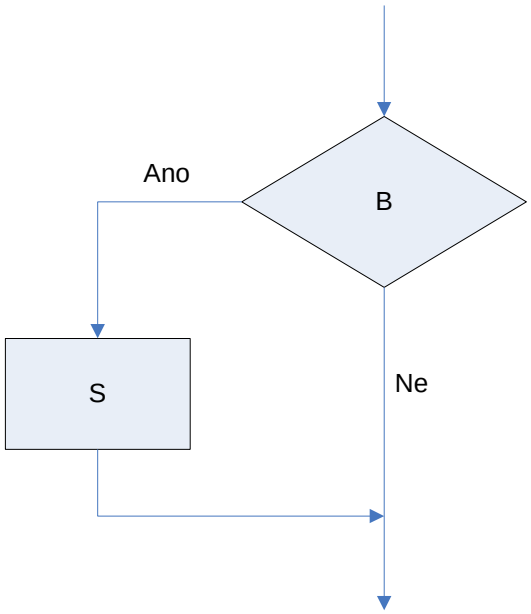
Syntaxe:
První způsob zápisu:

```
If B Then S1
```

Druhý (častější) způsob zápisu:

```
If B then
  S
End If
```

Kostru příkazu tvoří klíčová slova - pro zvýraznění jsou podtržena
B je podmínka - výraz typu Boolean
S je příkaz nebo posloupnost příkazů.
Vykonání příkazu:
Vyhodnotí se podmínka B. Je-li podmínka splněna (její hodnota je True), vykoná se příkaz S. Není-li podmínka splněna nevykoná se nic.
Vývojový diagram neúplného podmíněného příkazu:



4.3. Příkaz Select Case

Příkaz Select Case je další rozhodovací příkaz, který slouží k vícenásobnému rozvětvení výpočtu.
Umožňuje větvení programu v závislosti na hodnotě numerického nebo řetězcového výrazu.

Syntaxe:

```
Select Case výraz
Case seznam_výrazů1
příkazy1
Case seznam_výrazů2
  příkazy2
 …
Case Else
  příkazyN
End Select
```

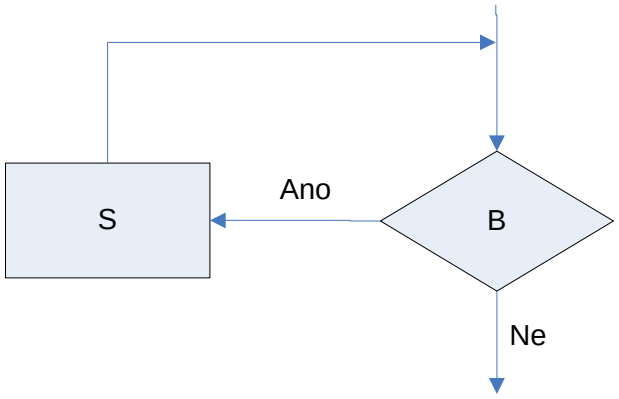
Příklad možností zápisu:

```
Dim Prom As Integer
Select case Prom
Case 1
  P1
Case 2 To 5, 8
  P2
Case 10 To 15, 18, 22, 25 To 33
  P3
Case Is > 39, 0
  P4
Case Else
  P5
End Select
```

5. Cykly s ukončovací podmínkou

Příkaz cyklu použijeme tehdy, když potřebujeme, aby se nějaká sekvence příkazů několikrát opakovala. Tuto sekvenci zapíšeme do příkazu cyklu a nemusíme ji opakovaně opisovat.

5.1. Cyklus s podmínkou před tělem cyklu



Syntaxe:

```
Do While B
   S
Loop
```

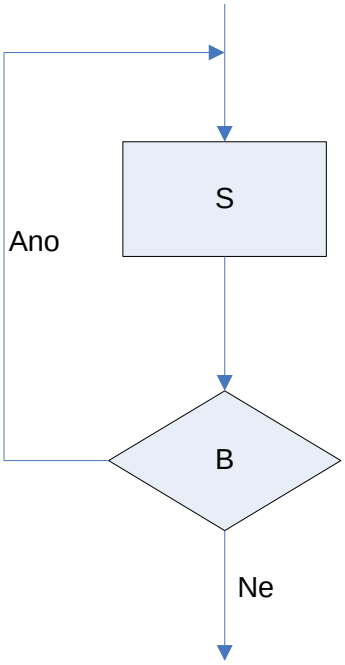
Kostru příkazu tvoří klíčová slova - pro zvýraznění jsou podtržena
B je podmínka - výraz typu Boolean
S - tělo cyklu - je příkaz nebo posloupnost příkazů.
Tělo cyklu se bude provádět v případě splněné podmínky.
Tělo cyklu se nemusí provést ani jednou.

5.2. Cyklus s podmínkou za tělem cyklu

Syntaxe:

```
Do
   S
Loop While B
```

Tělo cyklu se bude provádět v případě splněné podmínky.
Tělo cyklu se provede vždy alespoň jednou.



Je třeba, aby se v těle cyklu stalo něco, co má vliv na platnost ukončovací podmínky. Jinak se vám zřejmě podaří naprogramovat „věčný cyklus“ („nekonečnou smyčku“)

6. For cyklus

V modulu PRG1 jsme se již seznámili s cykly s ukončovací podmínku. Řekli jsme, že příkaz cyklu použijeme tehdy, když potřebujeme, aby se nějaká sekvence příkazů několikrát opakovala. Tuto sekvenci zapíšeme do příkazu cyklu a nemusíme ji opakovaně opisovat. Toto platí i pro speciální příkaz cyklu – For cyklus. **For cyklus použijeme tehdy, když předem známe počet opakování.** Ukončení opakování těla cyklu nezávisí tedy na podmínce, jako tomu bylo u příkazů cyklu s ukončovací podmínkou, ale počet opakování je předem určen.

Syntaxe:

```
For počítadlo = začátek to konec
tělo cyklu - příkazy
next počítadlo
```

Kostru příkazu tvoří klíčová slova - pro zvýraznění jsou podtržena.
Do počítadla je před provedením cyklu přiřazena hodnota začátek. Je-li hodnota počítadla menší nebo rovna hodnotě konec, provedou se příkazy těla cyklu a hodnota počítadla se zvýší o jedničku. Cyklus se opakuje, dokud hodnota počítadla nepřevýší hodnotu konec.

Počítadlo – řídící proměnná cyklu

Musí být některého numerického typu
Není „slušné“ měnit hodnotu řídící proměnné uvnitř těla cyklu
Může se stát, že tělo cyklu neproběhne ani jednou (Kdy ?)

Syntaxe ještě jinak:

```
For počítadlo = začátek to konec step krok
tělo cyklu - příkazy
next počítadlo
```

Do počítadla je před provedením cyklu přiřazena hodnota začátek. Je-li hodnota počítadla menší nebo rovna hodnotě konec, provedou se příkazy těla cyklu a hodnota počítadla se zvýší o **krok**. Cyklus se opakuje, dokud hodnota počítadla nepřevýší hodnotu konec.
Krok může být i záporná hodnota.

7. Strukturovaný datový typ pole

7.1. Datový typ pole

Datový typ pole je strukturovaný datový typ, proměnná tohoto typu je **struktura** tvořená několika prvky jednoduššího typu.
Všechny prvky pole jsou téhož typu – je to homogenní datová struktura.
Pole použijeme tehdy, když potřebujeme pracovat s několika proměnnými téhož typu, které k sobě nějak logicky patří, a potřebujeme je všechny zpracovávat stejným (nebo podobným) způsobem.
(Dosud jsme pracovali pouze s jednoduchými proměnnými - Integer, Single, Boolean, … Hodnoty těchto typů byly jednoduché, dále nedělitelné na jednodušší hodnoty.)

Pole je řada po sobě následujících paměťových míst označených jedním identifikátorem a rozlišených indexy, určujícími jejich pořadí.

Jednotlivé prvky pole jsou označeny indexy počínaje dolním indexem z deklarace.
Každý další prvek má index o jedničku větší, poslední prvek má index roven horní mezi z deklarace.
K jednotlivým prvkům pole přistupujeme prostřednictvím indexované proměnné.

7.2. Deklarace pole

Deklarace :

```
Dim NázevPole(dolní To horní) As Typ
```
nebo
```
Dim NázevPole(horní) As Typ
```
Při tomto druhém způsobu deklarace neurčujeme dolní mez indexu, která je tím standardně nastavena na 0
Pomocí příkazu `Option Base 1` lze dolní mez nastavit na 1.

Příklady deklarací :

```
Dim P(1 To 10) As Integer
Dim MojePole(5 To 17) As Boolean
Dim DalsiPole(-5 To 5) As Long
```
nebo
```
Dim Q(10) As Long
```

7.3. Indexovaná proměnná

K jednotlivým prvkům pole přistupujeme prostřednictvím indexované proměnné, např.

```
P(1)
P(4)
MojePole(12)
Q(5)
Q(10)
```

P(X), kde X je celočíselný výraz, jehož hodnota leží v intervalu <dolní, horní>

7.4. Pole a For cyklus

Pro práci s polem se výborně hodí For cyklus.
Uveďme několik jednoduchých příkladů

- Přečtěte ze vstupu 10 čísel a vypište je do ladícího okna v opačném pořadí

```
Dim P(1 To 10) As Integer
Dim I As Integer

For I = 1 To 10
  P(I) = val(Inputbox("Zadej " & I & ". číslo"))
Next I

For I = 10 To 1 Step -1
  Debug.Print P(I)
Next I
```

- Do prvků pole P vlož hodnoty, které jsou dvojnásobkem příslušného indexu

```
For I = 1 To 10
  P(I) = 2 * I
Next I
```

- Do prvku pole vlož hodnotu, která je dvojnásobkem hodnoty prvku s indexem o jedničku menším. Do prvního prvku vlož náhodnou hodnotu

```
P(1) = Int(Rnd * 25)
For I = 2 to 10
  P(I) = P(I - 1) * 2
Next I
```

7.5. Vícerozměrné pole

Příklady deklarací :

```
Dim M(1 To 10, 1 To 5) As Integer
```

Dvourozměrné pole si můžeme představovat jako tabulku, v tomto konkrétním případě má tabulka 10 řádků a 5 sloupců

```
Dim K(1 To 2, 0 To 5, -10 To 10) As Integer
```

Trojrozměrné pole

```
Dim L(5,5,20) as Boolean
```

Trojrozměrné pole, při tomto druhém způsobu deklarace neurčujeme dolní mez indexů, která je tím standardně nastavena na 0. (Nebyla-li příkazem `Option Base 1` nastavena na 1)

```
Dim Q(5 To 15,5,20) as Single
```

K jednotlivým prvkům pole přistupujeme opět prostřednictvím indexované proměnné, která ale v tomto případě musí mít počet indexů odpovídající počtu dimenzí daného pole. Hodnota každého z indexů musí ležet ve správném intervalu podle deklarace pole.
Příklad :

```
M(1,1)
Q(10,2,10)
L(1,1,1)
L(X,Y,Z)
```

kde X, Y a Z jsou celočíselné výrazy, jejichž hodnota leží v daném intervalu dle deklarace pole L

7.6. Průchod vícerozměrným polem

Pro průchod tabulkou (dvourozměrným polem) použijeme dva do sebe vnořené For cykly

```
Option Base 1
Dim M(10,5) As Integer
Dim I As Integer
Dim J As Integer

For I = 1 To 10
  For J = 1 To 5
    M(I, J) = Int(Rnd * 2))
  Next J
Next I
Rem Vyplní tabulku nulami a jedničkami

For I = 1 To 10
  For J = 1 To 5
```

```
    Debug.Print M(I, J);
  Next J
  Debug.Print
  Rem Přechod na nový řádek
Next I
Rem Vypíše přehlednou tabulku
```

7.7. Dynamické pole

Výhodou dynamického pole je, že jeho velikost se definuje až při běhu aplikace přesně podle potřeby (například podle velikosti nebo množství vstupních dat). Navíc mohu velikost dynamického pole za běhu aplikace průběžně podle potřeby měnit.

V deklaraci dynamického pole se v závorce za identifikátorem pole neuvede ani počet dimenzí ani indexový rozsah. Pouze se zavede nová proměnná, která později bude polem, jehož jednotlivé položky budou uvedeného typu. Po této deklaraci ještě nelze pracovat s jednotlivými položkami pole, protože ještě žádné neexistují. Na rozdíl od statického pole budou indexové rozsahy dynamického pole určeny později za běhu aplikace příkazem ReDim.

Deklarace dynamického pole

```
Dim NázevPole() As Typ
```

Příklad deklarace

```
Dim P() as integer
```

Příkaz ReDim

Příkaz ReDim provádí nastavení velikosti dynamického pole nebo nastavení jeho indexových rozsahů za běhu aplikace.
Dynamické pole, jehož velikost měním příkazem ReDim, je dobré předem deklarovat některým z příkazů Dim, Private, Public a určit jeho typ.
Teprve po provedení příkazu ReDim je nastavena velikost pole, je alokováno místo v operační paměti potřebné pro danou velikost pole a můžeme začít pracovat s jednotlivými jeho prvky.
Příkaz ReDim způsobí ztrátu případného předchozího obsahu pole. Po provedení tohoto příkazu budou všechny prvky pole inicializovány (v případě číselného pole tedy vyplněny nulami).
Příklady:

```
ReDim P(10)
ReDim P(N)        '(kde N je proměnná)
ReDim P(1 to N)
ReDim P(1 to N, 2 to 5)
```

Příkaz ReDim Preserve

Přejeme-li si použít příkaz ReDim a přitom zachovat předchozí hodnoty prvků pole, je třeba použít klíčové slovo Preserve.
Příklad:

```
ReDim Preserve P(N + 1)
```

Účinek příkazu ReDim Preserve je však omezen jen na poslední index pole a nelze již měnit počet rozměrů pole.

Příkaz Erase

Příkaz Erase slouží k uvolnění paměti, kterou zabírá dynamicky alokované pole.
Po provedení příkazu Erase P nebude pole P obsahovat žádné položky obdobně jako po provedení deklarace Dim P()…
Po smazání dynamického pole příkazem Erase je tedy nutné jej před dalším použitím znovu nastavit příkazem ReDim.
Příkaz Erase lze použít i pro statické pole. V tom případě dojde pouze k opětné inicializaci (vynulování) prvků pole pevné velikosti.

Práce s dynamickým polem

Při práci s dynamickým polem je dobré si pro toto pole deklarovat proměnnou (proměnné), ve které si budete uchovávat aktuální hodnotu (hodnoty) rozsahu pole.

Příklady:

- Velikost pole nastavím podle zadané hodnoty

```
Dim P() As Integer
Dim Poc As Integer
Dim I As Integer

Poc = Val(InputBox("Zadej velikost pole: "))
ReDim P(Poc)

For I = 1 To Poc
  P(I) = Int(Rnd * 20)
Next I
```

- Pole se dynamicky zvětšuje s každým zadaným číslem

```
Dim P() As Integer
Dim Poc As Integer
Dim I As Integer

Do
  Poc = Poc + 1
```

```
  ReDim Preserve P(Poc)
  P(Poc) = Val(InputBox("Zadej číslo: "))
Loop While MsgBox("Další číslo?", vbYesNo) = vbYes

Rem Kontrolní výpis hodnot pole
For I = 1 To Poc
  Print P(I)
Next I
```

8. Složitost algoritmu

Zajímáme se o paměťovou a časovou složitost (náročnost) algoritmu.
Každou úlohu lze řešit mnoha způsoby. Kritériem pro výběr jedné z možností řešení úlohy bývá časová a paměťová náročnost daného způsobu řešení.
Nejrychlejší algoritmus nemusí být optimální z hlediska paměťové náročnosti.
Obvykle dáme přednost efektivitě časové, nevede-li k příliš extrémním paměťovým nárokům.
Příklad ze života: Hledání v telefonním seznamu
Sekvenční vyhledávání - pomalé
Vyhledávání půlením intervalu - rychlé

Závislost složitosti algoritmu na vstupních datech

Doba výpočtu a paměťové nároky nějak závisí na vstupních datech. Obvykle nezávisí na hodnotách vstupních dat ale na jejich počtu.

- počet záznamů v telefonním seznamu
- počet hodnot, mezi kterými hledáme minimum

Někdy ale mohou paměťové a časové nároky záviset na vstupní hodnotě. (Najdi všechna prvočísla menší než N.)

Časová složitost

Časovou složitost neudáváme v časových jednotkách, ale v počtu elementárních operací (kroků algoritmu), které je třeba provést se vstupními daty, abychom získali výsledek.
Rozlišujeme časovou složitost v nejhorším a v průměrném případě.
Asymptotická časová složitost – je složitost v nejhorším případě pro velká vstupní data. Značí se O(vzorec závislosti na N, kde N je velikost vstupních dat).
Obvyklé složitosti:

- O(N) – lineární
- O(N2) – kvadratická
- O(logN) - logaritmická

Paměťová složitost

Paměťovou složitost můžeme udávat v bitech nebo v Bytech. Pro jednoduchost ji ale spíše uvádíme v počtu potřebných jednoduchých proměnných.

Zpět k telefonnímu seznamu

- Náš telefonní seznam má 400 000 jmen (N=400000).
- Elementární operace – porovnání dvou jmen trvá 1 sekundu.
- Při sekvenčním prohledávání bychom nad telefonním seznamem mohli strávit 5 dní a nocí složitost O(N).
- Při vyhledávání půlením intervalu potřebujeme asi 20 sekund O(logN).

9. Třídění

Co je třídění

Třídění znamená uspořádání množiny údajů vzestupně nebo sestupně podle zvolené klíčové položky.
Jinými slovy třídění znamená uspořádání množiny hodnot podle velikosti.
Nejde tedy o rozdělování hodnot do tříd, ale o stanovení jejich pořadí.
Přesnější termín pro třídění by mohl být řazení.

Důvod třídění

Hlavní důvod ke třídění - v setříděné množině se lépe vyhledává.

Příklady třídění

- Seznamy se řadí podle abecedy
- Faktury podle data splatnosti
- Závodníci podle výkonů
- …

V programech obvykle třídíme jednotlivé exempláře nějaké datové struktury. V takové struktuře bývá jedna položka označena jako klíč.
Podle této klíčové položky se záznamy třídí. Ostatní položky jsou z hlediska třídění nevýznamné, pouze se společně přemisťují.
My se pro zjednodušení omezíme na třídění celých čísel, která představují klíče tříděných datových záznamů.

Počet tříděných čísel N určuje velikost konkrétní úlohy na třídění.

V závislosti na N budeme vyjadřovat časovou složitost algoritmů. (počet porovnání, počet přesunů)

Není potřeba vymýšlet „rychlé“ algoritmy pro třídění malého počtu dat. Je-li ale N velké, pak už má smysl zamýšlet se nad efektivitou použitého algoritmu.

Vnitřní třídění

O vnitřním třídění mluvíme tehdy, jestliže velikost tříděné množiny umožňuje umístit všechna tříděná data najednou do operační paměti.

Někdy nazýváme vnitřní třídění tříděním polí nebo tříděním na místě.

Data jsou umístěna v datové struktuře pole, využívá se přímého přístupu k jednotlivým prvkům (indexy).

Tříděné hodnoty se v poli vzájemně přemisťují tak dlouho, až vznikne setříděná posloupnost.

Paměťová složitost je tedy N (pole o N prvcích).

Vnější třídění

Tříděná data se nevejdou najednou všechna do operační paměti (je jich příliš mnoho), jsou umístěna v souborech na vnějších paměťových médiích.

Třídění je založeno na čtení a vytváření souborů.

9.1. Přímé metody vnitřního třídění

Charakteristika

Jsou nejjednodušší
Jejich časová složitost je O(N2)
Jsou tedy výhodné pro ne příliš velká N

Deklarace

Pro všechny následující třídící algoritmy předpokládám deklaraci :

```
Const N = potřebná hodnota
Dim I as Integer
Dim P(1 To N) as Integer
```

9.2. Algoritmus třídění přímým výběrem

- Vyber v poli P prvek s nejmenším klíčem
- Vyměň ho s prvkem na první pozici
- Tyto operace opakuj se zbývajícími N-1 prvky, N-2 prvky, …

Časová složitost je $O(N^2)$

Krok algoritmu se provede N krát (N krát se prochází polem hodnot)
V prvním kroku se provede N-1 porovnání
Ve druhém kroku se provede N-2 porovnání
V posledním kroku se provede 1 porovnání
Součet aritmetické řady $(N-1) + (N-2) + \ldots + 1 = ½ N^2$
Tedy celkem se provede $½ N^2$ operací porovnání

Paměťová složitost je N

Je třeba pole velikosti N pro uložení tříděných hodnot plus jedna proměnná pro výměny a dvě proměnné pro indexy (což je vzhledem k velikosti N zanedbatelné)

Poznámka

Všimněte si, že všechny přímé metody rozdělují tříděné pole na dvě části, levou setříděnou a pravou nesetříděnou. Každým „krokem“ algoritmu se levá setříděná část zvětší o jednu hodnotu a pravá nesetříděná o jednu hodnotu zmenší. Tuto vlastnost lze s úspěchem použít při dokazování konečnosti a parciální správnosti těchto algoritmů.

Třídění přímým výběrem – zápis algoritmu

```
Public Sub PrimyVyber(P() As Integer, ByVal N As Integer)
Dim I As Integer
Dim J As Integer
Dim K As Integer
Rem Proměnné pro indexy
Dim X As Integer
Rem Proměnná pro výměny

For I = 1 To N - 1
  K = I
  For J = I + 1 To N
    If P(J) < P(K) Then K = J
    Rem V proměnné K si pamatuje index minima
  Next J
  If K > I Then
    X = P(K)
    P(K) = P(I)
    P(I) = X
  End If
Rem Zkuste si to vypsat
Rem  Vypis P, N
Next I
End Sub
```

9.3. Algoritmus třídění přímou výměnou - bublinkové třídění

V literatuře se setkáte s názvem pro tento algoritmus BubbleSort.
Tato metoda je založena na postupném porovnávání a vyměňování dvou sousedních prvků v poli.
Při troše představivosti vidíme, že nejmenší prvek „probublá“ na začátek.
Opakujeme-li tento postup N krát, získáme v poli setříděnou posloupnost hodnot.

Časová složitost je $O(N^2)$

Existují různá „vylepšení“, která ale nevedou k zásadnímu zrychlení:

- Algoritmus ukončím v okamžiku, kdy při posledním průchodu nedošlo k žádné výměně
- Polem procházím střídavě z obou konců (ShakerSort)

Paměťová složitost je N

Bublinkové třídění – zápis algoritmu

```
Public Sub PrimaVymena(P() As Integer, N As Integer)
Rem Bublinkove trideni
Dim I As Integer
Dim J As Integer
Dim X As Integer

For I = 2 To N
  For J = N To I Step -1
    If P(J - 1) > P(J) Then
      X = P(J - 1)
      P(J - 1) = P(J)
      P(J) = X
    End If
  Next J
  Rem Zkuste si to vypsat
  Rem  Vypis P, N
Next I
End Sub
```

Procedura pro kontrolní výpis hodnot v poli

```
Public Sub Vypis(P() As Integer, ByVal N As Integer)
Dim I As Integer

For I = 1 To N
  Debug.Print P(I);
Next I
Debug.Print
End Sub
```

9.4. Algoritmy vnitřního třídění s časovou náročností $O(N\log_2 N)$

Jsou to algoritmy založené na půlení zpracovávané množiny, mají časovou složitost závislou na logN – myslí se vždy log při základu 2.

Všechny tyto metody spočívají v rozdělení úlohy na dvě nebo více menších úloh, které jsou podobné původní úloze. Dílčí úlohy pak vyřešíme nezávisle na sobě a z jejich výsledků složíme výsledek celkový. Pokud jsou vzniklé dílčí úlohy stále ještě složité, řešíme je obdobně dalším dělením na stále menší úlohy. Dělení končí ve chvíli, kdy k řešení zbývají jednoduché úlohy, které již vyřešíme přímo.

Tato strategie je ve své podstatě rekurzivní, ale lze ji naprogramovat i bez použití rekurze.

9.5. Vyhledávání

Vyhledávání hodnoty v nesetříděném poli

Sekvenčně (od začátku do konce) procházím pole, dokud hodnotu nenajdu
Co se stane, když hledaná hodnota v poli není?
Sekvenčně procházím pole, dokud hodnotu nenajdu nebo dokud nevyčerpám všechny prvky pole (dokud nedojdu na konec pole).

Vyhledávání hodnoty v setříděném poli

Sekvenční vyhledávání:
Průchod polem může skončit buď v okamžiku, když hodnotu nalezneme, nebo když narazíme na hodnotu větší. V tomto případě se hledaná hodnota v poli nenachází.

Časová složitost zatím uvedených algoritmů je lineární, průměrně je třeba ½N porovnání.

Binární vyhledávání:
Je to metoda vyhledávání půlením intervalu. Lze ji použít jen v setříděném poli. (Vzpomeň na hledání v telefonním seznamu.)
V každém kroku zmenšíme prohledávanou množinu na polovinu.
Časová složitost je logN.

Sekvenční hledání v poli – zápis algoritmů

```
Const N = 100
Dim P(1 To N) As Integer
Dim I As Integer
```

Hledej 1

```
Function Hledej1(ByVal X As Integer) As Integer
Rem Funkce Hledej vrátí pozici hledaného čísla X v poli P
Rem Pokud se hledané číslo X v poli P nevyskytuje, vrátí
funkce hodnotu 0

For I = 1 To N
  If P(I) = X Then
    Hledej1 = I
    Exit For
  End If
Next I
End Function
```

Hledej 2

```
Rem Ještě totéž, ale bez použití Exit For
Function Hledej2(ByVal X As Integer) As Integer

For I = 1 To N
  If P(I) = X Then
    Hledej2 = I
  End If
Next I
Rem Dělá totéž jako Hledej1, ale vždy projde pole P až do
konce
End Function
```

Hledej 3

```
Rem Je-li pole setříděné, lze s hledáním skončit už v případě,
když najdeme v poli hodnotu větší než hledané X

Function Hledej3(ByVal X As Integer) As Integer

For I = 1 To N
  If P(I) = X Then
    Hledej3 = I
    Exit For
  Else
    If P(I) > X Then
      Exit For
    End If
  End If
Next I
End Function
```

10. Podprogramy

„Dám kusu programu jméno a mohu jej tímto jménem zavolat na různých místech **„**

Podprogram tvoříme z často se opakující sekvence příkazů nebo relativně samostatné části programu.

Terminologická poznámka : Termín ***podprogram*** je ve VB totožný s termínem ***procedura***

Výhody – důvody tvorby podprogramů :

- Vyhneme se opakovanému zápisu kódu
- Odladíme jen jednou
- Lepší čitelnost programu
- Opětovné použití v jiných programech

Dělení podprogramů :

Událostní procedura – je spojena s některým ovládacím prvkem a nějakou jeho běhovou událostí, její vyvolání je automatické v okamžiku, kdy dojde při běhu programu k odpovídající události
Standardní procedura – je součástí jazyka (Val, Str, Randomize, …)
Obecné procedury – vytváří programátor

Deklarace podprogramu :

Deklarací podprogramu rozumíme určení jeho identifikátoru – jeho jména, kterým budeme později tento podprogram volat a popis jeho činnosti, zápis jeho kódu.
Deklarací podprogramu vlastně zavádíme svůj nový pojmenovaný příkaz nebo funkci.
Deklaraci podprogramu umístíme buď v kódu formuláře, nebo ve standardním modulu, má-li mít obecné použití v celém projektu.

Volání podprogramu :

Voláním podprogramu rozumíme zápis názvu podprogramu v jiné části programu, čímž je vyvoláno vykonání tohoto podprogramu. Z nějakého místa programu je zavolán svým jménem jiný podprogram. Volající program v tu chvíli „počká“, až se vykonají příkazy volaného podprogramu, a pak teprve pokračuje.

Typy podprogramů :

Subrutiny – volání subrutiny je příkaz – něco vykoná

Funkce – volání funkce výraz - něco vykoná a vypočítá (vrátí) hodnotu

10.1. Procedura typu Sub

Deklarace:

```
Sub JmenoProcedury ()
    příkazy procedury
End Sub
```

Volání:

```
    JménoProcedury
```

Deklarace subrutiny s parametry:

```
Sub JmenoProcedury (seznam formálních parametrů)
    příkazy procedury
End Sub
```

Volání:

```
    JménoProcedury seznam skutečných parametrů
```

10.2. Procedura typu Function

Deklarace:

```
Function JménoFunkce (seznam formálních parametrů) As Typ
    příkazy funkce
    přiřazení funkční hodnoty identifikátoru funkce
End Function
```

Volání:

výraz

```
    Prom = JménoFunkce(případné skutečné parametry)
```

Vypočítaný výsledek funkce je vložen do proměnné Prom

```
    Prom = (X * JménoFunkce(případné skutečné parametry)) + 1
```

Vypočítaný výsledek je součástí složitějšího výrazu

```
    If  JménoFunkce(případné skutečné parametry) > X Then
```

Vypočítaný výsledek je součástí složitějšího výrazu

příkaz

```
    JménoFunkce případné skutečné parametry
```

Při volání funkce lze také použít stejnou syntaxi jako při volání procedury typu Sub v případě, že nechceme použít výsledek funkce. Výsledek funkce se tímto způsobem volání „ztratí“. (Tohoto způsobu volání jsme často využívali například při volání MsgBox)

10.3. Parametry

Terminologie :

Formální parametr - je uveden v deklaraci
Skutečný parametr – proměnná, konstanta nebo výraz uvedený při volání procedury

Parametry slouží k modifikaci činnosti procedury.

Pokusíme se ukázat na jednoduchých příkladech :

Budeme deklarovat subrutinu s názvem Hvězdičky

```
Sub Hvezdicky ()
Debug.Print "**********"
End Sub
```

Její volání se provede příkazem

```
Hvezdicky
```

Každé volání této procedury má za následek vypsání deseti * do ladícího okna.

Nyní deklarujme jinou subrutinu s názvem třeba Hvezdy

```
Sub Hvezdy (byVal Poc as integer)
do while Poc > 0
Debug.Print "*";
Poc = Poc -1
Loop
Debug.Print
End Sub
```

Můžeme ji zavolat příkazem

```
Hvezdy 5
```

Toto volání probíhá tak, že se nejdříve vyhodnotí hodnota skutečného parametru, ta je v tomto případě 5, a tato hodnota se dosadí (přiřadí) do formálního parametru Poc. Pak teprve proběhnou příkazy procedury Hvezdy. Tedy v tomto konkrétním volání procedura Hvezdy vypíše do ladícího okna pět hvězdiček.
Jiné volání

```
Hvezdy X
```

Skutečným parametrem při tomto volání je proměnná (musí být samozřejmě dříve deklarovaná a platná v tomto kontextu). Před vykonáním příkazů procedury Hvezdy se provede přiřazení hodnoty skutečného parametru X do formálního parametru Poc a pak se provedou příkazy procedury Hvezdy.

Ještě jeden příklad procedury se dvěma parametry

```
Sub Znaky (byVal Poc as integer, byVal ZN as String)
do while Poc > 0
Debug.Print ZN;
Poc = Poc -1
Loop
Debug.Print
End Sub
```

A příklady volání této procedury

```
Znaky 5, "@"

Rem musí platit následující deklarace
Dim X As Integer
Dim Y As String
Rem Proměnné X a Y získají nějaké hodnoty
Znaky X, Y
```

Příklad funkce

```
Function Prumer (ByVal X As Integer, ByVal Y As Integer, ByVal Z As Integer) As Double
Dim Pom As Integer
Rem proměnná Pom je lokální proměnná v této proceduře
Pom = X + Y + Z
Prumer = Pom / 3
Rem nesmíme zapomenout na tento příkaz, kterým určíme výslednou hodnotu funkce
End Function
```

Volání funkce

```
P = Prumer(A, B, C)
```

A, B, C a P jsou proměnné odpovídajících typů

Při volání obecné procedury je třeba uvést stejný počet skutečných parametrů a odpovídajících typů ve správném pořadí dle formálních parametrů uvedených v deklaraci procedury.

10.4. Platnost a viditelnost proměnných

V proceduře můžeme deklarovat proměnné, ty jsou pak v této proceduře **lokální**, to znamená, že existují jen v této proceduře, nikdo jiný s nimi nemůže pracovat. Vznikají a zanikají s procedurou. Tedy při každém volání procedury se znovu alokují a inicializují, při skončení běhu procedury zanikají.

Deklarace proměnné v proceduře může být ale uvedena klíčovým slovem **Static**, pak je tato proměnná statická, což znamená, že nezaniká při skončení procedury. Tento způsob deklarace použijeme tehdy, když je potřeba, aby tato lokální proměnná zachovávala svou hodnotu do dalšího volání procedury.
Proměnné deklarované mimo procedury na začátku kódu formuláře jsou **globální**, to znamená, že jsou platné v celém formuláři a mohou je tedy využívat všechny procedury deklarované v tomto formuláři.
Pokud budeme deklarovat v proceduře lokální proměnnou se stejným identifikátorem jako proměnnou globální, nebude v této proceduře globální proměnná vidět. Lokální proměnná může překrýt **viditelnost** globální proměnné.
Pojmy lokální a globální jsou relativní vzhledem k formuláři, modulu nebo celému projektu.
Potřebujeme-li mít globální proměnnou pro celý projekt, je zvykem takovou proměnnou deklarovat ve standardním modulu a její deklaraci uvést klíčovým slovem **Public** místo klíčového slova Dim. Naopak, chceme-li zdůraznit, že proměnná je lokální (soukromá) v daném kontextu, uvádíme její deklaraci klíčovým slovem **Private**.

10.5. Parametry volané hodnotou – vstupní parametry

Parametr volaný hodnotou je v deklaraci procedury je uveden s klíčovým slovem **ByVal**. (Zatím ve všech výše uvedených příkladech bylo popsáno volání parametrů hodnotou.)
Formální parametr se při volání hodnotou chová jako lokální proměnná, které se v okamžiku volání procedury přiřadí hodnota skutečného parametru. Po skončení procedury tedy formální parametr zaniká a nelze již zjistit jeho hodnotu.
Změna hodnoty parametru uvnitř procedury nemá vliv na hodnotu skutečného parametru.
Parametr slouží k předání hodnoty skutečného parametru **do** procedury.
Pokud je parametr některého uživatelsky definovaného typu (zatím např. pole) nesmí být předáván hodnotou.

10.6. Parametry volané odkazem – vstupně/výstupní parametry

Parametr volaný odkazem (referencí) je v deklaraci procedury je uveden s klíčovým slovem **ByRef**. **Není-li uveden způsob volání parametru, pak je standardně použit tento způsob volání odkazem**.
Při tomto způsobu volání se do procedury nepřenáší hodnota skutečného parametru, ale **odkaz** na něj.
Důsledkem je, že každá změna hodnoty formálního parametru se projeví i na skutečném parametru - pracuje se s tímtéž paměťovým místem.
Používá se tehdy, má-li parametr sloužit k předání hodnoty z procedury ven nebo je-li účelem procedury skutečný parametr změnit.
Tento způsob volání musíme použít i tehdy, je-li parametr „velký“ - např. pole.

11. Ladění kódu

11.1. Tři typy chyb

Syntaktická chyba

prostředí VB upozorňuje programátora na některé syntaktické chyby již při zápisu kódu syntakticky nesprávný program nelze spustit

Běhová chyba

způsobí zastavení programu neočekávanou chybou (špatná vstupní data, špatné jméno otevíraného souboru, dělení nulou, …)

Logická chyba

logická chyba programátora, těžko se hledá, program pracuje zdánlivě správně, ale nedává správné výsledky(, čehož si mnohdy ani nevšimneme)

11.2. Ladění aplikace a hledání logických chyb

K ladění aplikace a hledání chyb slouží Debugger, který je součástí prostředí VB. Při ladění se vhodně kombinují následující čtyři základní postupy :

Krokování

Aplikace je spouštěna po jednotlivých krocích (příkazech) pomocí nabídky Debug nebo pomocí odpovídajících klávesových zkratek.
Využívají se tři způsoby krokování :
Step Into (krokování včetně volaných podprogramů), Debug/Step Into F8
Step Over(krokování bez krokování podprogramů), Debug/Step Over Shift+F8
Step Out (návrat z podprogramu na místo jeho volání),Debug/Step Out Ctrl+Shift+F8

Sledování proměnných (watches)

Při krokování můžeme sledovat, jakých hodnot postupně nabývají sledované proměnné. Rychlé sledování umožňuje zjištění aktuální hodnoty proměnné při nastavení myši nad tuto proměnnou.
Jinou možností je vybrat seznam sledovaných proměnných pomocí nabídky Debug/Add Watch a potom sledovat hodnoty proměnných v okně Watches.

Breakpointy

Ve zdrojovém kódu lze nastavit tzv. Breakpointy, tedy označit příkazy, na kterých se má vykonávání aplikace zastavit a přejít do režimu krokování.
Breakpoint nastavíte pomocí nabídky Debug/Toggle Breakpoint nebo klávesou F9. Nejjednodušší způsob nastavení je klik na lištu vedle příslušného řádku. (Obdobně odstranění).

Přímé vstupy

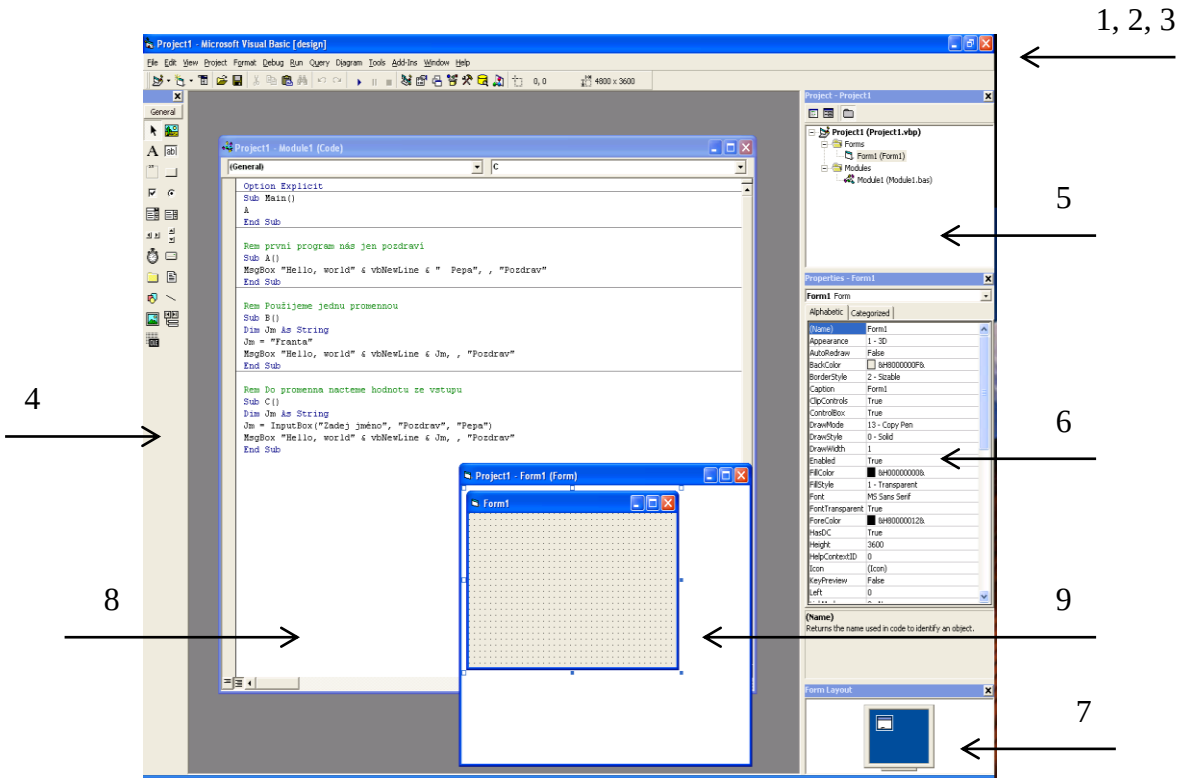
Během ladění aplikace máme také možnost využít speciální okno Immediate Windows – okno přímých vstupů, do kterého lze během krokování psát příkazy programu, kterými lze například změnit aktuální hodnotu proměnné. Příkazy psané do tohoto okna se vykonají vždy ihned po potvrzení klávesou Enter.

12. Událostmi řízené programování

Program ve VB se chová tak, že se po jeho spuštění nejčastěji zobrazí úvodní okno (formulář) a dále se neděje nic. Program čeká, až nastane událost, na kterou „umí“ reagovat. Reaguje na tuto událost provedením nějaké akce a znovu čeká na další událost.
Událostí se myslí například klik myší na tlačítko na formuláři, pohyb myší po formuláři, zápis znaku z klávesnice do textového pole, …
Reakcí na událost je provedení kódu programu, který ale musíme předem naprogramovat.
Návrh programu ve VB tedy mimo jiné spočívá v tom, že navrhujeme vzhled formulářů a programujeme reakce na události, které mohou nastat v souvislosti s ovládacími prvky umístěnými na formulářích.

12.1. Vývojové prostředí VB

Pokusíme o návrh jednoduchých projektů s využitím základních ovládacích prvků, které umístíme na formulář a navrhneme procedury událostí pro tyto ovládací prvky.
Nejdříve si připomeňme vývojové prostředí Visual Basicu.

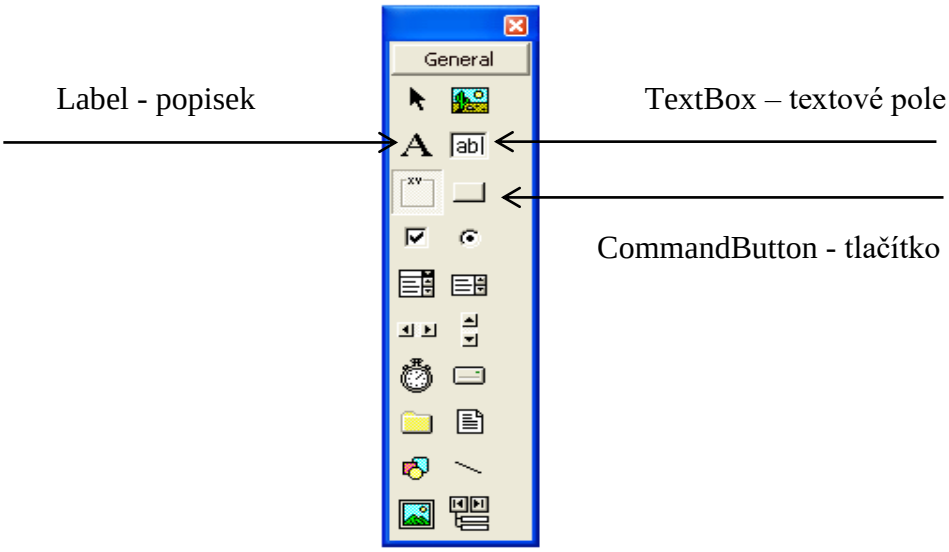


1. Titulkový pruh
2. Nabídkovou lištu (Menu bar)
3. Panel nástrojů (Toolbar)
4. Souprava nástrojů (Toolbox)
5. Průzkumník projektu (Project Explorer)
6. Okno vlastností ovládacích prvků (Properties Window)
7. Okno rozvržení formuláře (Form Layout)
8. Okno kódu (Code Window)
9. Okno návrhu formuláře (Form Designer)

Pokud některé z oken nevidíte, máte k dispozici příkaz k jeho zobrazení v nabídce ***Wiew*** z nabídkové lišty.

12.2. Souprava nástrojů (Toolbox)

V tomto okně je sada nástrojů, které se používají při rozmisťování ovládacích prvků na formulář.
Při návrhu aplikace používáme předdefinované objekty (ovládací prvky), které umisťujeme na formulář a měníme (nastavujeme) jejich vlastnosti, využíváme jejich metody a (programujeme) reakce na události s nimi spojené.



12.3. Vlastnosti (Properties) ovládacích prvků

Nejběžnější vlastnosti :

Name – identifikátor ovládacího prvku
Caption - popisek, titulek
Top, Left - vzdálenost od horního nebo levého okraje
Width - šířka
Height - výška
BackColor - barva pozadí (tlačítku je třeba nejprve nastavit styl na Graphical)
Font
Enabled
Visibled
…

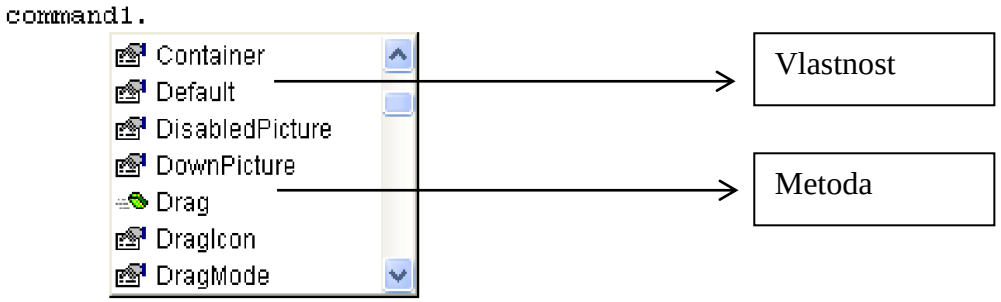
Nastavení vlastností :

- Při zakreslování na formulář (v režimu design)
- Nastavení vlastností v okně vlastností Properties Window
- V programovém kódu za běhu programu name**.**vlastnost = hodnota
  Př. CB1.Left = CB1.Left + 250
  Př. CB1.Top = int(rnd()*(Form1.ScaleHeight-CB1.Height))
  Př. Label1.Caption = “Nějaký nápis”

12.4. Metody

Ovládací prvky jsou opatřeny různými metodami. Metoda je schopnost objektu (vykonat nějakou činnost).
Volání metody – obdobně jako kteroukoli jinou proceduru voláme jejím jménem v kódu programu. (Má-li metoda parametry, budou při zápisu kódu nabízeny.)
Př. CB1.Move 100,100,1500,1500
Př. CB1.SetFocus

Vlastnosti ani metody jednotlivých typů ovládacích prvků si nemusíme pamatovat. Při zápisu kódu jsou nám nabízeny. Metody a vlastnosti mají odlišné ikony.

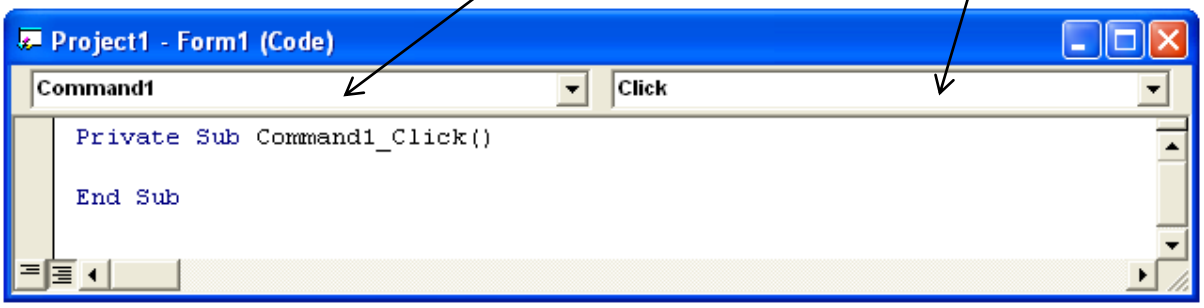


12.5. Události

Ovládací prvky dovedou reagovat na události (Klik myší, stisk klávesy, …) Způsob, jakým bude konkrétní ovládací prvek reagovat na danou událost je třeba naprogramovat - to je vaším úkolem

Programování událostních procedur :

V okně kódu zvolíme **ovládací prvek** (levý rozbalovací seznam) a **událost** (pravý rozbalovací seznam) - tím se automaticky vytvoří šablona odpovídající událostní procedury. Tělo této procedury naprogramujte.



Při zápisu vlastností, metod a událostních procedur důsledně využívejte vestavěných nápověd prostředí VB !!

(Usnadníte si tím život.)

12.6. Základní ovládací prvky

CommandButton - Tlačítko

Standardní identifikátor - Command1

Pokud se rozhodnete identifikátor změnit, musíte tak učinit v Properties Windows, kde nastavíte vlastnost (name) odpovídajícímu prvku. Vlastnost (name) měňte ihned po umístění ovládacího prvku na formulář ještě před vytvářením událostních procedur. Při pozdějších změnách byste si pravděpodobně v kódu vyrobili těžko odstranitelný chaos a program by nepracoval dle vašich představ. (Můžete si rozmyslet, co jsem touto dobře míněnou radou měla na mysli.)

Dohodněme se, že identifikátor tlačítka budeme začínat znaky **CB**

Např. CBOk, CBKonec, CBStart, CB1, CB2

Obvyklá událost spojená s tlačítkem je klik myší.

Label – Štítek, Popisek

Standardní identifikátor - Label1
Slouží k vkládání popisků na formulář, není ovládací prvek v pravém smyslu.
Identifikátor budeme začínat znaky **LB**
Např. LBJmeno, LBPrijmeni, LB1, LB2

TextBox – Textové pole

Standardní identifikátor - Text1
Slouží k zadávání dat uživatelem prostřednictvím klávesnice
Identifikátor budeme začínat znaky **TX**
Např. TXJmeno, TXHeslo, TX1
Nemá vlastnost Caption
Má vlastnost **Text** typu String
Další vlastnosti jsou např.
PasswordChar
SelText - obsahuje vybraný text
SelStart - počáteční pozice vybraného textu
SelLength - délka vybraného textu
MultiLine
ScrollBars
Obvyklé události spojené s TextBoxem jsou Change, GotFocus, LostFocus, KeyPress
K události KeyPress dojde při stisku klávesy v aktivním TextBoxu. V obslužné proceduře této události je parametr KeyAscii, který obsahuje ASCII kód zadaného znaku.

Form - Formulář

Standardní identifikátor - Form1
Doporučuji identifikátor formuláře neměnit (alespoň v prvních příkladech)
Některé události - **Load**, **Activate**, Unload, Resize
Některé zajímavé metody – Hide, Show, Print, Cls, …

12.7. Fokus – zaměření

Fokus je schopnost objektu přijímat uživatelský vstup prostřednictvím klávesnice nebo myši. Obsahuje-li aktivní formulář více prvků, jeden z nich má nastaven fokus - je zvýrazněn. (O získání zaměření má smysl mluvit jen v případě běžící aplikace.)
Události spojené se získáním a ztrátou zaměření :
GotFocus - při získání zaměření
LostFocus - při ztrátě zaměření

Jak nastavit fokus na objekt :

- Vybrat objekt při běhu aplikace (Tab, myš)
- Použít při běhu aplikace přístupovou klávesu (je-li nastavena)
- Zavolat v kódu programu metodu tohoto objektu SetFocus

Objekt může obdržet fokus jen má-li vlastnosti Enabled a Visible nastaveny na True
Některé objekty nemají schopnost získat fokus - např. Label

Pořadí, v jakém prvky získávají fokus stiskem TAB většinou odpovídá pořadí vzniku prvku při návrhu formuláře. Potřebujeme-li toto pořadí změnit, lze to udělat nastavením vlastnosti TabIndex jednotlivých ovládacích prvků. Při změně TabIndex některého prvku se ostatní automaticky přečíslují. Je-li vlastnost Enabled = False, prvek se přeskakuje.
Ovládací prvek bude při stisku klávesy Tab přeskakován i v případě, že má nastavenu vlastnost TabStop = False

13. Pole ovládacích prvků

13.1. Použití

Potřebuji více prvků téhož typu, které se chovají podobně

13.2. Vytvoření

- pojmenuji všechny stejně, rozliším je pomocí vlastnosti Index
- kopírováním - objeví se dotaz, zda si přejete vytvořit pole prvků

13.3. Výhody

- událostní procedury programuji jen jednou, rozliším indexem, kterého z prvků se událost týká, kterým z prvků byla vyvolána
- pomocí For cyklu mohu postupně zpracovat všechny prvky pole
- Výhoda indexování se projeví ve složitějších úlohách, kdy použijete nejen více podobných prvků, ale i procedury psané pro tyto prvky budou komplikovanější.
- Pozor - pokud indexy prvků netvoří souvislou řadu, nelze dost dobře použít For cyklus. Máme ale možnost použít jiný cyklus, který postupně projde všemi prvky kolekce, je jím **For Each** cyklus.

13.4. Název

- všechny prvky pole mají stejnou vlastnost name, rozlišují se indexem
- např. CB(0).Caption, CB(I).Top, CB(X+1).Left

13.5. Vytváření ovládacích prvků za běhu programu

- Jednotlivé přidávané prvky jsou prvky nějakého pole ovládacích prvků
- Alespoň jeden z prvků musí být vytvořen v návrhu formuláře „ručně“ a musí mu být nastavena vlastnost Index
- Ostatní prvky pole lze vytvářet dynamicky za běhu aplikace příkazem Load

Příkaz Load

Příkazem Load lze vytvořit prvek, který ještě neexistuje (pozor na indexy). Po vytvoření příkazem Load má prvek stejné vlastnosti (Caption, Top, Left, …) jako prvek, podle kterého

vznikl (to je ten, který jste vytvořili ručně) - je tedy třeba jej nově umístit, jinak bude „schován“ pod prvkem, který je jeho vzorem.
Pozor - po vytvoření má vytvořený prvek vlastnost Visible = False.
Př.

```
Load Label(2)          Předpokládám, že Label(1) je vytvořen v návrhu
For I = 1 To 5         Předpokládám, že CB(0) je vytvořen v návrhu
  Load CB(I)
  CB(I).Left= CB(I-1).Left + CB(I-1).Width
  CB(I).Visible = True
Next I
```

Příkaz Unload

Prvky, které byly dynamicky vytvořeny, lze zase odstranit příkazem Unload. Nelze odstranit prvky, které byly vytvořeny „ručně“ v návrhu.
Př.

```
Unload Label(2)
```

14. Další ovládací prvky

14.1. Timer - časovač

Používá se v případě, kdy potřebujete provádět nějaké akce (příkazy) pravidelně po uplynutí určitého časového intervalu.
Na formulář umístíme Timer ze základní sady nástrojů (z Toolboxu). Na běžící aplikaci není vidět.
Vlastnosti:
Má-li Timer nastavenu vlastnost **Enabled** na True, v pravidelných intervalech „tiká“
K nastavení délky intervalu slouží vlastnost **Interval**, hodnota intervali tikání se udává milisekundách, tedy hodnota 1000 je přibližně 1 sekunda (Pozor, nastavíte-li Interval = 0, timer nebude tikat).
Událost, která nastává po uplynutí intervalu má název Timer. (Ovládací prvek Timer **má jedinou událost – Timer**.)

14.2. ScrollBar - posuvná lišta

Použije se tehdy, když potřebujeme mít možnost nechat uživateli zadávat celá čísla z nějakého intervalu.
Na formulář umístíme vertikální nebo horizontální ScrollBar ze základní sady nástrojů (z Toolboxu)
Nejdůležitější vlastnosti :

- Min, Max - rozsah intervalu
- LargeChange - změna při klikání na lištách
- SmallChange - změna při klikání na šipkách
- Value - hodnota

Nejtypičtější událost ScrollBaru je Change.

14.3. ListBox - seznam

a

14.4. ComboBox – pole se seznamem

Zobrazují seznam položek, z nichž si uživatel může vybrat

ComboBox slučuje funkce ListBoxu a TextBoxu, ComboBox obsahuje textové pole, do něhož lze zadat hodnoty, které nejsou uvedeny v seznamu. ComboBox, více šetří místo na formuláři.

Přesáhne-li počet položek ListBoxu velikost ovládacího prvku na formuláři, automaticky se zobrazí posuvník (neplést se ScrollBarem).

Vlastnosti

List - obsah Boxu - všechny položky = indexované pole řetězců box.list(index)
Sorted - True/False - automatické setřídění položek Boxu
Text – vybraná položka
ListIndex – nastavuje nebo vrací index vybrané položky (Je přístupná pouze při běhu) Není-li vybrána žádná položka, hodnota je rovna –1
NewIndex - index poslední přidané položky (hodí se u tříděného seznamu)
ListCount – počet položek
Další obvyklé vlastnosti Name, Enabled, …

Události

Click – vybere položku
doporučení – současně s Boxem použít příkazové tlačítko, Click na toto tlačítko provede s vybranou položkou Boxu příslušnou akci
A další ...

Přidání položky do seznamu

Za běhu programu metoda AddItem
syntaxe - box.AddItem řetězec[,index]
je-li index vynechán přidá se nová položka na konec, indexuje se od 0
Při návrhu – vyplnit vlastnost List, po zapsání položky stisk CTRL+ENTER
(Je-li vlastnost Sorted = True nepoužívat metodu AddItem s indexem !!)

Odstranění položky ze seznamu

metoda RemoveItem box.RemoveItem index
metoda Clear box.Clear odstraní všechny položky

14.5. CheckBox ☐ - zaškrtávací políčko

Jednotlivá zaškrtávací políčka na formuláři jsou na sobě nezávislá, možnost vybrat libovolnou kombinaci zaškrtnutí.
Nejdůležitější vlastnost:
Value

0	VBUnchecked
1	VBChecked
2	(VBGrayed)

obvyklé vlastnosti (caption – popisek, enabled, tabindex, …)
Události:
Click přepíná ✓

DoubleClick je bráno jako 2x Click
Událost Click je vyvolána i tehdy, má-li tento prvek fokus a je stisknut mezerník nebo je-li před některým znakem v Caption znak & →klávesová zkratka ALT+znak

14.6. OptionButton ○ - přepínač

Všechny přepínače umístěné přímo na formuláři tvoří jednu skupinu – lze vybrat jen jednu z možností – jen jeden OptionButton má hodnotu True – vybráním jiného se ostatní přepnou na False
Chceme-li další nezávislou skupinu, nutno umístit tyto přepínače do rámu – prvek Frame nebo do pole s brázkem PictureBox.
Vlastnost **Value** (hodnota True / False)
Změna hodnoty skupiny přepínačů :

- Click
- Tab výběr skupiny, pak kurzorovými klávesami
- Nastavením Value v programu
- Klávesová zkratka

15. Práce s více formuláři

Všechny programy, které jsme zatím vytvořili, měly pouze jeden formulář Někdy je užitečné přidat do programu jeden nebo více dalších formulářů

- Samostatná úvodní obrazovka
- Dialogy – získávání vstupních údajů nebo zobrazení výstupních dat
- Informace o programu, návod

Každý formulář je samostatný objekt, má své vlastní ovládací prvky, vlastnosti, procedury událostí. První formulář se implicitně jmenuje Form1 další formuláře mají implicitní jména Form2, Form3, …

15.1. Vložení nového formuláře

Nabídka Project – Add Form nebo příslušná ikona na nástrojové liště, případně klik pravým tlačítkem myši v okně Project Exploreru.
Objeví se dialog, který si vyžádá zadání typu nového formuláře. Vkládáme buď prázdný formulář, formulář částečně hotový, určený k nějakému účelu nebo existující formulář z jiného projektu – (nezapomeňte jej ihned uložit pod jiným jménem, případné úpravy by vám jinak „pokazily“ původní projekt, ze kterého formulář pochází).

15.2. Spouštěcí formulář projektu

Formulář, který se zavádí jako první po spuštění projektu (např. úvodní obrazovka)
Standardně je to formulář Form1
Určení spouštěcího formuláře:
V nabídce Project – Properties na kartě General určíte Startup Object
Po spuštění projektu nastanou události –

Form_Load spouštěcího formuláře
Form_Activate spouštěcího formuláře

Spouštěcí formulář je tedy zobrazen na monitoru.
Ostatní formuláře nejsou zavedeny do paměti a nejsou tedy samozřejmě ani vidět na monitoru. Jejich zavedení do paměti a zobrazení musíme naprogramovat.

15.3. Příkazy pro formuláře

Zavedení formuláře do paměti

Příkaz Load jméno_formuláře
Př. `Load Form2`
Po zavedení formuláře do paměti již můžeme používat jeho vlastnosti a metody, ale ještě není na monitoru vidět

Zobrazení zavedeného formuláře metodou show

Volání metody formuláře *Jméno_formuláře.Show [mód]*
Př. `Form2.Show`
Stejného efektu dosáhneme nastavením vlastnosti Visible zavedeného formuláře na True
Př. `Form2.Visible = True`
Zavoláte-li metodu Show bez předchozího zavedení formuláře příkazem Load, zadaný formulář bude automaticky zaveden do paměti a zobrazen.

Modální a nemodální formuláře

Je-li v projektu zobrazeno současně více formulářů, můžeme dovolit uživateli libovolně se přepínat mezi jednotlivými formuláři, nebo pevně stanovíme pořadí, v jakém s formuláři může pracovat.

Nemodální formulář – uživatel se z něj může přepnout do jiného formuláře
Jméno_formuláře.Show 0
Modální formulář – po jeho zobrazení s ním musí uživatel pracovat, nemůže se přepnout do jiného formuláře, dokud tento formulář neuzavře
Jméno_formuláře.Show 1
Implicitní hodnota parametru mód je 0, nemodální formulář lze tedy zobrazit příkazem
Jméno_formuláře.Show

Skrytí formuláře

Volání metody formuláře *Jméno_formuláře.Hide*
Př. `Form2.Hide`
Stejného efektu dosáhneme nastavením vlastnosti Visible na False
Př. `Form2.Visible = False`
Skrytý formulář není pro uživatele viditelný, v paměti ale zůstává

Odstranění formuláře z paměti

Volání metody formuláře *Unload jméno_formuláře*
Př. `Unload Form2`

Viditelnost proměnných

Globální proměnné a obecné procedury deklarujte ve standardním modulu jako Public, mohou je pak využívat všechny formuláře
S proměnnými a procedurami deklarovanými ve formuláři pracujte pouze v daném formuláři.

15.4. MDI formuláře

(Multiple Document Interface)

Jde o vztah rodičovského (nadřízeného) formuláře a formulářů synovských (podřízených).
Znáte např. z MS Wordu
Rodičovský formulář vytvoříme příkazem Add MDI Form z nabídky Project
Synovský formulář vytvoříme nejdříve jako standardní formulář příkazem Add Form a pak do jeho vlastnosti MDIChild přiřadíme hodnotu True
Rodičovský formulář je v projektu jeden, synovských může být více

Chování synovských formulářů za běhu programu :

Synovské formuláře se zobrazují uvnitř okna rodičovského formuláře Po minimalizaci se synovský formulář zobrazí jako malý titulek uvnitř rodičovského formuláře. Titulek maximalizovaného synovského formuláře se zobrazí v titulku rodičovského formuláře.

16. Řešené příklady

Pro začátek budeme pracovat pouze ve standardním modulu a své prográmky spouštět ve spouštěcí proceduře Main. Případně budeme programovat pouze událost kliknutí na tlačítko na formuláři.

16.1. Pozdravy

```
Rem první program nás jen pozdraví
Sub A()
MsgBox "Hello, world" & vbNewLine & "  Pepa", , "Pozdrav"
End Sub
```

```
Rem Použijeme jednu proměnnou
Sub B()
Dim Jm As String
Jm = "Franta"
MsgBox "Hello, world" & vbNewLine & Jm, , "Pozdrav"
End Sub
```

```
Rem Do proměnné načteme hodnotu ze vstupu
Sub C()
Dim Jm As String
Jm = InputBox("Zadej jméno", "Pozdrav", "Pepa")
MsgBox "Hello, world" & vbNewLine & Jm, , "Pozdrav"
End Sub
```

16.2. Jednoduché výpočty

```
Sub Bazen()
Rem Výpočet objemu a povrchu bazénu
Dim A As Integer
Dim B As Integer
Dim C As Integer
Dim Objem As Integer
Dim Povrch As Integer

Rem Funkce Val převádí znakový řetězec na číslo
Rem Pokud znakový řetězec neobsahuje zápis čísla,
Rem výsledkem funkce bude 0
A = Val(InputBox("Zadej šíøku bazénu ", "1. rozměr bazénu "))
B = Val(InputBox("Zadej délku bazénu ", "2. rozměr bazénu "))
```

```
C = Val(InputBox("Zadej hloubku bazénu ", "3. rozměr bazénu "))

Objem = A * B * C
Povrch = 2 * C * A + 2 * C * B + A * B

MsgBox "Bazén s rozměry " & A & "*" & B & "*" & C & vbNewLine & "Objem ... " & _ Objem & vbNewLine & "Povrch ... " & Povrch, , "Výpočet objemu a povrchu bazénu"
End Sub
```

```
Sub Hodiny()
Rem převod údaje v hodinách, minutách a sekundách na sekundy
Dim H As Integer
Dim M As Integer
Dim S As Integer
Dim Sekund As Integer

H = Val(InputBox("Zadej počet hodin ", "H O D I N Y "))
M = Val(InputBox("Zadej počet minut ", "M I N U T Y "))
S = Val(InputBox("Zadej počet sekund ", "V T E Ř I N Y "))

Sekund = H * 3600 + M * 60 + S

MsgBox H & " Hod " & M & " Min " & S & " Sec je" & vbNewLine & Sekund & " Sec", , _ "Převod na sekundy"
End Sub
```

```
Sub HodinyNaopak()
Rem převod údaje v sekundách na hodiny, minuty a sekundy
Dim H As Integer
Dim M As Integer
Dim S As Integer
Dim SekundyCelkem As Integer
Dim Zb As Integer

SekundyCelkem = Val(InputBox("Zadej údaj v sekundách ", "V T E Ř I N Y"))

H = SekundyCelkem \ 3600
Zb = SekundyCelkem Mod 3600
M = Zb \ 60
S = Zb Mod 60

MsgBox SekundyCelkem & " >>> " & H & ":" & M & ":" & S, , "Sekundy na hodiny, _ minuty a sekundy"
End Sub
```

16.3. Podmínky

```
Sub Nejvetsi ()
Rem Najdi největší ze tří čísel
Dim X As Integer
Dim Y As Integer
Dim Z As Integer
Dim Nej As Integer

Rem proměnné X,Y a Z získají nějaké hodnoty
Rem např. načtením ze vstupu

If  X > Y Then
  Nej = X
Else
  Nej = Y
End If
Rem V prom. Nej je hodnota většího z X a Y

If Z > Nej Then Nej = Z
MsgBox "Největší z čísel " _
& X & " " & Y & " " & Z & "  je " & Nej
End Sub
```

```
Sub Obrazce()
Rem Určeni obrazce
Dim rozmerA As Integer
Dim rozmerB As Integer

rozmerA = Val(InputBox("Zadej sirku jako X."))
rozmerB = Val(InputBox("Zadej delku jako Y."))

If rozmerA = rozmerB Then
  MsgBox "To je čtverec"
Else
  MsgBox "To je obdélnik"
End If
End Sub
```

```
Sub Podil()
Rem Přečti ze vstupu dvě čísla X a Y,
Rem spočítej hodnotu výrazu 1/XY
Dim X As Integer
Dim Y As Integer
Dim Titulek As String

Titulek = "***  1/XY  ***"
```

```
X = Val(InputBox("Zadej první číslo ", Titulek))
Y = Val(InputBox("Zadej druhé číslo ", Titulek))
If X = 0 Or Y = 0 Then
  MsgBox "Nulou dělit nelze", vbCritical, Titulek
Else
  MsgBox "Výsledek pro hodnoty X=" & X & " Y=" & Y & _
vbNewLine & " je " & 1 / (X * Y), vbInformation, Titulek
End If
End Sub
```

```
Sub Trojuhelnik()
Rem Zadej ze vstupu délky tří úseček
Rem Rozhodni, jestli mohou být stranami trojúhelníka
Rem Součet každých dvou stran trojúhelníka musí být větší
Rem než strana třetí
Dim A As Integer
Dim B As Integer
Dim C As Integer

A = Val(InputBox("Zadej délku první úsečky ", "Trojúhelník"))
B = Val(InputBox("Zadej délku druhé úsečky ", "Trojúhelník"))
C = Val(InputBox("Zadej délku třetí úsečky ", "Trojúhelník"))
If A + B > C And B + C > A And A + C > B Then
  MsgBox "Úsečky mohou být stranami trojúhelníka", _
vbInformation, "Trojúhelník"
Else
  MsgBox "Úsečky nemohou být stranami trojúhelníka", _
vbCritical, "Trojúhelník"
End If
End Sub
```

```
Sub Mereni()
Rem Zadej hodnoty třikrát opakovaného fyzikálního měření
Rem a předpokládanou hodnotu
Rem Zjisti, zda byla alespoň jednou naměřena předpokládaná
Rem hodnota
Dim Predp As Single
Dim M1 As Single
Dim M2 As Single
Dim M3 As Single

Predp = Val(InputBox("Zadej očekávanou hodnotu ", "Měření"))
M1 = Val(InputBox("Zadej hodnotu prvního měření ", "Měření"))
M2 = Val(InputBox("Zadej hodnotu druhého měření ", "Měření"))
M3 = Val(InputBox("Zadej hodnotu třetího měření ", "Měření"))

If M1 = Predp Or M2 = Predp Or M3 = Predp Then
  MsgBox "Alespoň jednou jste se trefili do předpokládané _
```

```
hodnoty", , "Měření"
Else
  MsgBox "Ani jednou nebyla naměřena předpokládaná hodnota", _
vbCritical, "Měření"
End If

Rem ještě můžeme zjistit, KOLIKRÁT byla naměřena předpokládaná
hodnota
Dim Poc As Integer
If M1 = Predp Then Poc = Poc + 1
If M2 = Predp Then Poc = Poc + 1
If M3 = Predp Then Poc = Poc + 1
MsgBox "Předpokládaná hodnota byla naměřena " & Poc & " _
krát", , "Měření"

Rem co kdybychom chtěli zjistit, kolikrát byla naměřená
Rem hodnota v toleranci 10% od předpokládané hodnoty
Dim Mez1 As Single
Dim Mez2 As Single
Dim Pocet As Integer
Mez1 = Predp * 0.9
Mez2 = Predp * 1.1
If M1 >= Mez1 And M1 <= Mez2 Then Pocet = Pocet + 1
If M2 >= Mez1 And M2 <= Mez2 Then Pocet = Pocet + 1
If M3 >= Mez1 And M3 <= Mez2 Then Pocet = Pocet + 1
MsgBox "S tolerancí 10% ..... " & Pocet & " měření", , _
"Měření"
End Sub
```

16.4. Vnořování podmínek

```
Rem Rovnoměrný pohyb
Rem Uživatel si vybere, kterou z veličin dráha, rychlost nebo
Rem čas, si přeje počítat
Rem Vyžádáme si zadání hodnot zbývajících dvou veličin
Rem a požadovanou veličinu vypočítáme

Sub A()
Dim Odpoved As Integer
Dim Rychlost As Single
Dim Cas As Single
Dim Draha As Single

Odpoved = Val(InputBox("Zadej,co budeme počítat" & vbNewLine &
"rychlost (1), čas (2), dráhu (3)"))
```

```
If Odpoved = 1 Then
    Draha = Val(InputBox("Zadej dráhu:"))
    Cas = Val(InputBox("Zadej čas"))
    If Cas > 0 Then
      Rychlost = Draha / Cas
      MsgBox "Rychlost je: " & Rychlost
    Else
      MsgBox "Nelze dělit nulou"
    End If
Else
    If Odpoved = 2 Then
        Draha = Val(InputBox("Zadej dráhu:"))
        Rychlost = Val(InputBox("Zadej rychlost:"))
        If Rychlost > 0 Then
          Cas = Draha / Rychlost
          MsgBox "Čas je: " & Cas
        Else
          MsgBox "Nelze dělit nulou"
        End If
    Else
        If Odpoved = 3 Then
            Rychlost = Val(InputBox("Zadej rychlost:"))
            Cas = Val(InputBox("Zadej čas"))
            Draha = Rychlost * Cas
            MsgBox "Dráha je: " & Draha
        Else
            MsgBox "Špatné zadání"
        End If
    End If
End If
End Sub
```

```
Sub B()
Rem Totéž ale trochu jinak
Dim Odpoved As Integer
Dim Rychlost As Single
Dim Cas As Single
Dim Draha As Single

Odpoved = Val(InputBox("Zadej,co budeme počítat" & vbNewLine &
"rychlost (1), čas (2), dráhu (3)"))

If Odpoved = 1 Then
    Draha = Val(InputBox("Zadej dráhu:"))
    Cas = Val(InputBox("Zadej čas"))
    If Cas > 0 Then
      Rychlost = Draha / Cas
      MsgBox "Rychlost je: " & Rychlost
```

```
    Else
      MsgBox "Nelze dělit nulou"
    End If
End If

If Odpoved = 2 Then
  Draha = Val(InputBox("Zadej dráhu:"))
  Rychlost = Val(InputBox("Zadej rychlost:"))
  If Rychlost > 0 Then
    Cas = Draha / Rychlost
    MsgBox "Čas je: " & Cas
  Else
    MsgBox "Nelze dělit nulou"
  End If
End If

If Odpoved = 3 Then
  Rychlost = Val(InputBox("Zadej rychlost:"))
  Cas = Val(InputBox("Zadej čas"))
  Draha = Rychlost * Cas
  MsgBox "Dráha je: " & Draha
End If

If Odpoved <> 1 And Odpoved <> 2 And Odpoved <> 3 Then
  MsgBox "Špatné zadání"
End If

End Sub
```

```
Sub C()
Rem A ještě jinak
Dim s As Integer
Dim t As Integer
Dim v As Integer
Dim Z As String

Z = (InputBox("Co chcete počítat?" & vbNewLine & "rychlost
(v), čas (t), dráhu (s)"))
Z = LCase(Z)
Rem Funkce Lcase převádí v řetězci všechna písmena na velká
If Z = "t" Then
  s = Val(InputBox("Zadej dráhu"))
  v = Val(InputBox("Zadej rychlost"))
  If v > 0 Then
    t = s / v
    MsgBox "čas je " & t
  Else
    MsgBox "Nulou dělit nelze"
```

```
  End If
Else

  If Z = "s" Then
    t = Val(InputBox("Zadej čas"))
    v = Val(InputBox("Zadej rychlost"))
    s = v * t
    MsgBox "dráha je " & s
  Else

    If Z = "v" Then
      s = Val(InputBox("zadej dráhu"))
      t = Val(InputBox("zadej čas"))
      If t > 0 Then
        v = s / t
        MsgBox "rychlost je " & v
      Else
        MsgBox "Nulou dělit nelze"
      End If
    Else
      MsgBox "zadej správně"
    End If
  End If
End If

End Sub
```

16.5. Příkaz Select Case

```
Sub D()
Rem A ještě jednou totéž pomocí příkazu Case
Dim sel As String
Dim x1 As Double
Dim x2 As String
Dim v As Integer
Dim t As Integer
Dim s As Integer

sel = InputBox("Co budeme počítat? [s, v, t]")
sel = LCase(sel)
Select Case sel
Case "s"
t = Val(InputBox("Zadej čas t? [h]"))
v = Val(InputBox("Zadej rychlost v? [km/h]"))
x1 = v * t
x2 = "km"
```

```
Case "v"
t = Val(InputBox("Zadej čas t? [h]"))
s = Val(InputBox("Zadej dráhu s? [km]"))
If t > 0 Then
  x1 = s / t
  x2 = "km/h"
Else
  x2 = "Nulou dělit nelze"
End If

Case "t"
s = Val(InputBox("Zadej drahu s? [km]"))
v = Val(InputBox("Zadej rychlost v? [km/h]"))
If v > 0 Then
  x1 = s / v
  x2 = "h"
Else
  x2 = "Nulou dělit nelze"
End If

Case Else
  x2 = "špatně zadaná volba"
End Select

If x1 = 0 Then
  MsgBox x2
Else
  MsgBox "Výsledek je " & x1 & " " & x2
End If
End Sub
```

```
Sub JesteCase()
Dim Mes As Integer
Mes = Val(InputBox("Zadej číslo měsíce"))
Select Case Mes
Case 1 To 2
  MsgBox "Zima"
Case 3 To 5
  MsgBox "Jaro !!!"
Case 6
  MsgBox "Léto"
Case 7, 8
  MsgBox "P R Á Z D N I N Y"
Case 9 To 11
  MsgBox "Podzim"
Case 12
  MsgBox "Vánoce"
```

```
Case Else
  MsgBox "Špatné číslo měsíce"
End Select
End Sub
```

16.6. Cykly

```
Sub Cykly1()
Dim Cislo As Integer
Dim Soucin As Long

Rem čti ze vstupu čísla, poslední zadané číslo je 0,
Rem spočítej součet přečtených čísel
Do
  Cislo = Val(InputBox("Zadej číslo ", "Počítám součet", 0))
  Sum = Sum + Cislo
Loop While Cislo <> 0
MsgBox Sum, , "Součet čísel"
End Sub
```

```
Sub Cykly2()
Dim Cislo As Integer
Dim Soucin As Long

Rem čti ze vstupu čísla, poslední zadané číslo je 0,
Rem spočítej součin přečtených čísel
Rem Když to udělám obdobně jako v předchozí úloze,
Rem tak to asi nepůjde! Proč???
Do
  Cislo = Val(InputBox("Zadej číslo ", "Počítám součin", 0))
  Soucin = Soucin * Cislo
Loop While Cislo <> 0
MsgBox Sum, , "Součin čísel"

Rem Protože součin bude vždy 0. Víš proč ???
Rem Má to dva důvody, oba je napravíme v dalším řešení
End Sub
```

```
Sub Cykly3()
Dim Cislo As Integer
Dim Soucin As Long

Soucin = 1
Cislo = Val(InputBox("Zadej číslo ", "Počítám součin", 0))
Do While Cislo <> 0
Soucin = Soucin * Cislo
```

```
Cislo = Val(InputBox("Zadej číslo ", "Počítám součin", 0))
Loop
Rem Už víš ty dva důvody ??
End Sub
```

```
Sub Cykly4()
Rem čti ze vstupu čísla, poslední zadané číslo je 0,
Rem spočítej počet zadaných čísel

Dim Cislo As Integer
Dim Pocitadlo As Integer
Do
  Cislo = Val(InputBox("Zadej číslo ", _
"Počítám počet zadaných čísel", 0))
  Pocitadlo = Pocitadlo + 1
Loop While Cislo <> 0
MsgBox Pocitadlo, , "Počet čísel"
End Sub
```

```
Sub Cykly5()
Rem čti ze vstupu čísla, poslední zadané číslo je 0,
Rem spočítej aritmetický průměr přečtených čísel

Dim Sum As Integer
Dim Cislo As Integer
Dim Pocitadlo As Integer
Sum = 0
Pocitadlo = 0
Do
  Cislo = Val(InputBox("Zadej číslo ", "Počítám průměr", 0))
  Sum = Sum + Cislo
  Pocitadlo = Pocitadlo + 1
Loop While Cislo <> 0
MsgBox Sum / (Pocitadlo - 1), , "Průměr čísel"
Rem Víš proč dělím součet hodnotou proměnné Počitadlo - 1 ???
Rem Napiš to tak, aby se průměr vypočítal jako Sum/Pocitadlo
End Sub
```

Náměty na další procvičení :
Čti ze vstupu čísla, poslední zadané číslo je 0, spočítej počet kladných členù posloupnosti

Všechny předchozí úlohy, ale nečtu čísla tak dlouho dokud nepřečtu číslo 0, ale vím předem, kolik čísel se bude číst.

16.7. Cykly - Házení kostkou

```
Sub Hazeni1()
```

```
Dim Hod As Integer
Rem Házej kostkou tak dlouho, dokud nepadne 6
Do
Hod = Int(Rnd * 6 + 1)
Rem Použili jsme generátor náhodných čísel
Rem Funkce Rnd generuje náhodnou hodnotu  -
Rem desetinné číslo z intervalu <0, 1)
Rem funkce Int dává celou část z desetinného čísla
MsgBox Hod
Loop While Hod < 6
End Sub
```

```
Sub Hazeni2()
Dim Hod As Integer
Dim Poc As Integer
Rem Házej kostkou tak dlouho, dokud nepadne 6,
Rem spočítej, na kolikátý pokus se to podařilo

Poc = 0
Do
Poc = Poc + 1
Hod = Int(Rnd * 6 + 1)
MsgBox Hod, , "Pokus " & Poc
Loop While Hod < 6
MsgBox "Šestku hodil na " & Poc & ". pokus", , _
"Člověče, nezlob se"
End Sub
```

```
Sub Hazeni3()
Dim Hod As Integer
Dim Poc As Integer
Dim Poc6 As Integer
Rem Hoď desetkrát kostkou. Spočítej, kolikrát padla 6

Poc = 0
Poc6 = 0
Do
Poc = Poc + 1
Hod = Int(Rnd * 6 + 1)
MsgBox Hod, , "Pokus " & Poc
If Hod = 6 Then
  Poc6 = Poc6 + 1
End If
Loop While Poc < 10
MsgBox "Šestku hodil " & Poc6 & " krat", , _
"Člověče, nezlob se"
End Sub
```

```
Sub Hazeni4()
Dim Hod As Integer
Dim Cesta As Integer
Dim Postoupil As Integer
Dim Poc As Integer

Cesta = Val(InputBox("Zadej délku cesty"))
Rem Zadá počet políček hrací desky
Rem Házej kostkou a postupuj o hozený počet políček po hrací
Rem desce dokud neprojdeš celou cestu

Do
  Hod = Int(Rnd * 6) + 1
  Poc = Poc + 1
  Postoupil = Postoupil + Hod
  MsgBox "Hodil " & Hod & vbNewLine & "Postoupil o " & _
Postoupil & " políček"
Loop While Postoupil < Cesta
MsgBox "Zadanou cestu prošel na " & Poc & " pokusů"
End Sub
```

```
Sub Hazeni5()
Dim Hod As Integer
Dim Cesta As Integer
Dim Postoupil As Integer
Dim Poc As Integer

Cesta = Val(InputBox("Zadej délku cesty"))
Rem Totéž jako Hazeni4, ale zadanou cestu musí projít přesně,
Rem hody, kterými by délku cesty překročil, se nepoužijí

Do
  Hod = Int(Rnd * 6) + 1
  Poc = Poc + 1
  If Postoupil + Hod <= Cesta Then
    Postoupil = Postoupil + Hod
  End If
  MsgBox "Hodil " & Hod & vbNewLine & "Postoupil o " & _
Postoupil & " políček"
Loop While Postoupil < Cesta
MsgBox "Zadanou cestu prošel na " & Poc & " pokusů"
End Sub
```

```
Sub Hazeni6()
Dim Hod As Integer
Dim Poc As Integer

Rem Házej kostkou!
```

```
Rem Házení skončí když padne 6 nebo když vyčerpáš 5 pokusù

Do
  Poc = Poc + 1
  Hod = Int(Rnd * 6 + 1)
  MsgBox Hod, , "Pokus " & Poc
Loop While Hod < 6 And Poc < 5
If Hod = 6 Then
  MsgBox "Padla 6", , "Bingo"
Else
  MsgBox "Vyčerpány pokusy, 6 nepadla", , "Tak nic"
End If
End Sub
```

16.8. Cykly - Hledání minima

```
Sub Hledani1()
Rem Hledá nejmenší hodnotu v posloupnosti zadaných čísel
Const Konec = 50
Rem příkaz pro deklaraci pojmenované konstanty s hodnotou 50
Dim Cis As Integer
Dim Min As Integer
Cis = Val(InputBox("Zadej číslo "))
Min = Cis
Do While Cis <> Konec
  Cis = Val(InputBox("Zadej číslo "))
  If Cis < Min Then
    Min = Cis
  End If
Loop
MsgBox Min, , "Minimum"
End Sub
```

Co se stane, když Konec bude menší než všechny hodnoty posloupnosti ?
.. Min bude vždy = Konec .. načtení až po zpracování ..
Vyřeší následující příklad

```
Sub Hledani2()
Const Konec = 50
Dim Cis As Integer
Dim Min As Integer
Cis = Val(InputBox("Zadej číslo "))
Min = Cis
Do While Cis <> Konec
  If Cis < Min Then
    Min = Cis
```

```
  End If
  Cis = Val(InputBox("Zadej číslo "))
Loop
MsgBox Min, , "Minimum"
End Sub
```

```
Sub Hledani3()
Rem S využitím znalosti vlastností posloupnosti čísel vkládám
Rem do Min na začátku "velkou hodnotu"

Const Konec = 50
Const VelkaHodnota = 32000
Dim Cis As Integer
Dim Min As Integer

Min = VelkaHodnota
Do
  Cis = Val(InputBox("Zadej číslo "))
  If Cis < Min Then
    Min = Cis
  End If
Loop While Cis <> Konec
MsgBox Min, , "Minimum"
End Sub
```

```
Sub Hledani4()
Rem Urči počet výskytů minimální hodnoty

Const Konec = 30
Dim Cis As Integer
Dim Min As Integer
Dim Poc As Integer

Min = Konec
Do
  Cis = Val(InputBox("Zadej teplotu!"))
  If Cis = Min Then Poc = Poc + 1
  If Cis < Min Then
    Min = Cis
    Poc = 0
  End If
Loop While Cis <> Konec
MsgBox Min & vbNewLine & Poc & " krát", , "Minimum"
End Sub
```

Další varianty

Urči pořadí minima v posloupnosti

Urči počet výskytù minima v posloupnosti

16.9. Ještě cykly

```
Sub A()
Rem Čti posloupnost čísel tak dlouho, dokud je rostoucí
Rem Musím si pamatovat dvě po sobě jdoucí čísla

Dim Cislo As Integer
Dim Predch As Integer

Cislo = Val(InputBox("Zadej číslo "))

Do
  Predch = Cislo
  Cislo = Val(InputBox("Zadej číslo "))
Loop While Cislo > Predch

End Sub
```

```
Sub B()
Rem Přečti ze vstupu celé číslo, urči jeho ciferný součet
Rem Řeš bez použití řetězcových proměnných a funkcí

Dim C As Long
Dim CifSum As Integer
Dim Pom As Integer

C = Val(InputBox("Zadej číslo "))

Rem Číslo C si musím někam zapamatovat,
Rem protože se v následujícím cyklu nakonec vynuluje
Rem a nemohli bychom jej použít v odpovědi
Pom = C

Do While C > 0
  CifSum = CifSum + (C Mod 10)
  Rem Operátor Mod dává zbytek po celočíselném dělení
  C = C \ 10
  Rem Operátor \ - celočíselné dělení
Loop

C = Pom: Rem teď to zapamatované číslo zase použiji

MsgBox "Ciferný součet čísla " & C & " je " & CifSum
```

```
End Sub
```

16.10. Cykly- Eukleides

```
Sub Eukleides()
Rem Eukleidův algoritmus nalezení největšího společného
Rem dělitele dvou přirozených čísel

Dim A As Integer
Dim B As Integer
Dim AA As Integer
Dim BB As Integer
Dim NSD As Integer
Dim Pom As Integer

A = Val(InputBox("Zadej 1. číslo"))
B = Val(InputBox("Zadej 2. číslo"))
AA = A
BB = B
Rem Zadaná čísla si musím zapamatovat do pomocných proměnných
AA a BB

Rem Zjisti největší společný dělitel čísel A a B
Do While A <> B
  If A < B Then
    Rem nejdříve zařídíme, aby v proměnné A byla větší z
hodnot
    Rem Následující tři příkazy provedou výměnu hodnot
    Rem proměnných A a B pomocí třetí proměnné Pom
    Pom = A
    A = B
    B = Pom
  End If
  A = A - B
Loop
NSD = A
A = AA
B = BB
Rem Použiji zpět zapamatovaná čísla
MsgBox "NSD čísel " & A & " a " & B & " je " & NSD
End Sub
```

```
Sub EukleidesRychleji()
Dim A As Integer
Dim B As Integer
```

```
Dim AA As Integer
Dim BB As Integer
Dim NSD As Integer
Dim Pom As Integer

A = Val(InputBox("Zadej 1. číslo"))
B = Val(InputBox("Zadej 2. číslo"))
AA = A
BB = B
Rem Zadaná čísla si musím zapamatovat do pomocných proměnných
AA a BB
Do While A <> B
  If A < B Then
    Rem nejdříve zařídíme, aby v proměnné A byla větší z
hodnot
    Rem Následující tři příkazy provedou výměnu hodnot
    Rem proměnných A a B pomocí třetí proměnné Pom
    Pom = A
    A = B
    B = Pom
  End If
  A = A Mod B
  If A = 0 Then
    A = B
  End If
Loop
NSD = A
A = AA
B = BB
Rem Použiji zpět zapamatovaná čísla
MsgBox "NSD čísel " & A & " a " & B & " je " & NSD
End Sub
```

16.11. For cyklus

```
Dim I As Integer
Rem Globální deklarace pro všechny úlohy
```

```
Sub Uloha1()
Rem For cyklus proběhne 10krát,
Rem vypisuje hodnotu řídící proměnné

For I = 1 To 10
  Debug.Print I;
Next I
```

```
Debug.Print
End Sub
```

```
Sub Uloha2()
Rem For cyklus s krokem 2

For I = 1 To 10 Step 2
  Debug.Print I;
Next I
Debug.Print
End Sub
```

```
Sub Uloha3()
Rem A co teď ?

For I = 10 To 1 Step -1
  Debug.Print I;
Next I
Debug.Print
End Sub
```

```
Sub Uloha4()
Rem Přečti ze vstupu počet žáků ve třídě,
Rem Zadej jejich známky z matematiky a vypočítej
Rem průměrnou známku ve třídě

Dim PocZaku As Integer
Dim Zn As Integer
Dim S As Integer
Dim Prum As Single

PocZaku = Val(InputBox("Zadej počet žáků ve třídě"))

For I = 1 To PocZaku
  Zn = Val(InputBox("Zadej znamku " & I & ". žáka"))
  S = S + Zn
Next I

Prum = Round(S / PocZaku, 2)
MsgBox "Průměrná známka ve třídě : " & Prum
End Sub
```

```
Sub Uloha5()
Rem Totéž jako Uloha4, ale kontroluje známky

Dim PocZaku As Integer
Dim Zn As Integer
Dim S As Integer
```

```
Dim Prum As Single

PocZaku = Val(InputBox("Zadej počet žáků ve třídě"))

For I = 1 To PocZaku
  Do
    Zn = Val(InputBox("Zadej znamku " & I & ". žáka"))
  Loop While Zn < 1 Or Zn > 5
  S = S + Zn
Next I

Prum = Round(S / PocZaku, 2)
MsgBox "Průměrná známka ve třídě : " & Prum
End Sub
```

```
Sub Uloha6()
Rem a celá škola

Dim PocZaku As Integer
Dim Zn As Integer
Dim S As Integer
Dim Prum As Single

Dim PocTrid As Integer
Dim J As Integer

PocTrid = Val(InputBox("Zadej počet tříd ve škole"))

For J = 1 To PocTrid

PocZaku = Val(InputBox("Zadej počet žáků v " & J & ". třídě"))
S = 0
For I = 1 To PocZaku
  Do
    Zn = Val(InputBox("Zadej znamku " & I & ". žáka"))
  Loop While Zn < 1 Or Zn > 5
  S = S + Zn
Next I

Prum = Round(S / PocZaku, 2)
MsgBox "Průměrná známka v " & J & ". třídě : " & Prum
Next J
End Sub
```

```
Rem Ještě si můžete spočítat průměr za celou školu

Sub Uloha7()
Rem hoď Nkrát kostkou, spočítej, kolikrát padla 6
```

```
Const N = 20

Dim H As Integer
Dim Poc6 As Integer

Randomize

For I = 1 To N
  H = Int(Rnd * 6) + 1
  Debug.Print H;
  If H = 6 Then Poc6 = Poc6 + 1
Next I
MsgBox "Šestka padla " & Poc6 & " krát"
Debug.Print 'to jen pro odřádkování
End Sub
```

```
Sub Nasobilka(ByVal X As Integer)

For I = 1 To 10
  Debug.Print X * I; Tab(I * 5);
Next I
Debug.Print

End Sub
```

```
Sub CelaNasobilka()
Dim J As Integer

For J = 1 To 10
  Nasobilka J
Next J

End Sub
```

```
Sub NasobilkaJinak(ByVal X As Integer)

For I = X To 10 * X Step X
  Debug.Print I;
Next I
Debug.Print
Rem Ale tady obtížněji udělám pěknou tabulku, zkus to vyřešit
End Sub
```

```
Sub CelaNasobilkaJinak()
Rem Volá proceduru NasobilkaJinak
Dim J As Integer
```

```
For J = 1 To 10
  NasobilkaJinak J
Next J

End Sub
```

16.12. Pole a For cyklus, subrutiny a funkce

```
Dim P(1 To 10) As Integer
Dim I As Integer
Rem Globální deklarace pro všechny úlohy
```

```
Sub Uloha1()
Rem Vyplň pole náhodnými hodnotami
For I = 1 To 10
  P(I) = Int(Rnd * 10)
Next I
End Sub
```

```
Sub Uloha2()
Rem Vypiš hodnoty pole
For I = 1 To 10
  Debug.Print P(I);
Next I
Debug.Print
End Sub
```

```
Sub Uloha3()
Rem sečti hodnoty všech prvků pole
Dim Sum As Integer
For I = 1 To 10
  Sum = Sum + P(I)
Next I
Debug.Print "Součet prvků pole : "; Sum
End Sub
```

```
Sub Uloha4()
Rem Vypíše hodnoty pole v opačném pořadí
For I = 10 To 1 Step -1
  Debug.Print P(I);
Next I
Debug.Print
End Sub
```

```
Sub Uloha5()
Rem Zjistí kolikrát se v poli vyskytuje zadaná hodnota
```

```
Dim H As Integer
Dim Poc As Integer

H = Val(InputBox("Zadej hledané číslo "))

For I = 1 To 10
  If P(I) = H Then
    Poc = Poc + 1
  End If
Next I

MsgBox "Hodnota " & H & " se v poli vyskytuje " & Poc & "
krát"
End Sub
```

```
Sub Uloha6(ByVal H As Integer)
Rem stejně jako předchozí úloha, ale hledané číslo je dáno
parametrem
Dim Poc As Integer

For I = 1 To 10
  If P(I) = H Then
    Poc = Poc + 1
  End If
Next I

MsgBox "Hodnota " & H & " se v poli vyskytuje " & Poc & "
krát"

End Sub
```

```
Function Uloha7(ByVal H As Integer) As Integer
Rem zase totéž, ale funkce
Rem výsledkem funkce je počet hledaných hodnot v poli

Dim Poc As Integer

For I = 1 To 10
  If P(I) = H Then
    Poc = Poc + 1
  End If
Next I

Uloha7 = Poc
End Function
```

```
Sub Uloha8()
```

```
Rem zjistí, kolik hodnot v poli je menších než hodnota
posledního prvku pole
Dim Poc As Integer

For I = 1 To 9
  If P(I) < P(10) Then
    Poc = Poc + 1
  End If
Next I

MsgBox Poc & " hodnot menších než poslední hodnota"
End Sub
```

```
Function Uloha9() As Integer
Rem totéž jako Uloha8, ale je to funkce,
Rem jejím výsledkem je počet hodnot menších než poslední
hodnota

Dim Poc As Integer

For I = 1 To 9
  If P(I) < P(10) Then
    Poc = Poc + 1
  End If
Next I

Uloha9 = Poc
End Function
```

*16.13. Pole **

```
Rem Kód formuláře
Option Explicit
Option Base 1
Const N = 10
Dim A(N) As Integer
Dim I As Integer
Dim J As Integer
Rem Globální deklarace pro celý formulář
```

```
Private Sub Vypis()
For I = 1 To N
  Print A(I); Tab(4 * I);
Next I
Print
Rem Vypíše obsah pole do řádku na formulář
```

```
End Sub
```

```
Private Sub Prosta1()
Rem Vyplní pole náhodnými hodnotami tak, aby se žádná hodnota
neopakovala
Dim bylo As Boolean
A(1) = Int(Rnd * 10) + 1
Rem První položku pole vyplním libovolným číslem
Rem U dalších položek pole musím kontrolovat,
Rem aby hodnota byla různá od již vyplněných položek
For I = 2 To N
  Do
    A(I) = Int(Rnd * 10) + 1
    Rem Předpokládám, že se podařilo do položky A(I)
    Rem vygenerovat hodnotu,
    Rem která ještě nebyla v předchozích položkách
    bylo = False
    Rem Důvěřuj, ale prověřuj
    For J = 1 To I - 1
      If A(I) = A(J) Then
        bylo = True
      End If
    Next J
  Loop While bylo
Next I

Rem Pozor, pole má 10 položek, různých náhodných hodnot
Rem generovaných předpisem Int(Rnd * 10) + 1
Rem je 10 - tak to vyjde !!!
Rem Vyzkoušej generovat předpisem Int(Rnd * 20) + 1
Rem nebo třeba Int(Rnd * 5) + 1
End Sub
```

```
Private Sub Prosta2()
Rem Trochu rychlejší algoritmus než v předchozí proceduře
Rem Pokud zjistím, že se hodnota opakuje,
Rem končím s prohledáváním a ihned zkouším jinou hodnotu
A(1) = Int(Rnd * 10) + 1
For I = 2 To N
  Do
    A(I) = Int(Rnd * 10) + 1
    J = 1
    Do While (J < I) And (A(I) <> A(J))
        J = J + 1
    Loop
  Loop Until I = J
Next I
End Sub
```

```
Private Sub Command1_Click()
Prosta1
Vypis
End Sub
```

```
Private Sub Command2_Click()
Prosta2
Vypis
End Sub
```

```
Private Sub Command3_Click()
Rem Statistika házení kostkou
Rem Zjisti, kolikrát padla 1, 2, 3, .. pri N hodech kostkou
Dim Hody(1 To 6) As Integer
Dim Hod As Integer
Const N = 1200
Rem Je-li generátor v pořádku, mělo by každé hodnoty padnout
asi 20
Rem Čím větší bude počet hodů, tím budou výsledky přesnější
Dim I As Integer
For I = 1 To N
  Hod = Int(Rnd * 6) + 1
  Hody(Hod) = Hody(Hod) + 1
Next I
For I = 1 To 6
  Print Hody(I);
Next I
Print
End Sub
```

```
Private Sub Command4_Click()
Rem Zjistí nejmenší hodnotu v poli
Dim Min As Integer
Min = A(1)
For I = 1 To N
  If A(I) < Min Then
    Min = A(I)
  End If
Next
Print "Nejmenší hodnota v poli: " & Min
End Sub
```

```
Private Sub Command5_Click()
Rem Zjistí nejmenší hodnotu v poli a určí, na které pozici
Rem v poli se tato nejmenší hodnota nachází
Rem Co když je nejmenší hodnota v poli vícekrát ?
Dim Min As Integer
Dim Poz As Integer
```

```
Min = A(1)
Poz = 1
For I = 1 To N
  If A(I) < Min Then
    Min = A(I)
    Poz = I
  End If
Next
Print "Nejmenší hodnota: " & Min, "Je na pozici " & Poz
End Sub
```

```
Private Sub Command6_Click()
Rem Zjistí nejmenší hodnotu v poli a vymění ji s hodnotou na
Rem první pozici v poli
Dim Min As Integer
Dim Poz As Integer
Dim Pom As Integer

Min = A(1)
Poz = 1
For I = 1 To N
  If A(I) < Min Then
    Min = A(I)
    Poz = I
  End If
Next
Rem V promenne Poz si pamatuje pozici minima,
Rem pokud je pozice ruzna od 1 - minimum je na jine pozici nez
na prvni
Rem prohazuje hodnoty pomocí promenne Pom
If Poz <> 1 Then
  Pom = A(1)
  A(1) = A(Poz)
  A(Poz) = Pom
End If
Vypis
End Sub
```

```
Private Sub Form_Load()
Randomize
End Sub
```

16.14. Podprogramy s parametry

```
Function Mocnina(ByVal X As Integer, ByVal N As Integer) As Long
Rem Vypočítá N-tou mocninu X
```

```
Rem Předpokládám, že N je nezáporné celé číslo

Dim Pom As Long
Pom = 1
Do While N > 0
  Pom = Pom * X
  N = N - 1
Loop

Mocnina = Pom

End Function
```

```
Sub Mocneni()
Rem Příklady volání funkce Mocnina
Dim C As Integer
Dim D As Integer
Dim V As Long

MsgBox Mocnina(2, 4)

C = Val(InputBox("Zadej číslo "))
V = Mocnina(C, 3)
MsgBox V
End Sub
```

```
Function JeCislice(ByVal Z As String) As Boolean

If Len(Z) = 1 And Z >= "0" And Z <= "9" Then
  JeCislice = True
Else
  JeCislice = False
End If

End Function
```

```
Function DejCislo(ByVal Znak As String) As Integer
If JeCislice(Znak) Then
  DejCislo = Asc(Znak) - Asc("0")
Else
  DejCislo = 10
End If
End Function
```

```
Sub Cisla()
Rem Příklady volání funkce JeCislice
Dim Cislice As Integer
```

```
Rem MsgBox JeCislice("b")
Cislice = DejCislo("B")
Cislice = DejCislo("5")
End Sub
```

```
Function MinZeTri(ByVal A As Integer, ByVal B As Integer,
ByVal C As Integer) As Integer
If A <= B And A <= C Then MinZeTri = A
If B <= A And B <= C Then MinZeTri = B
If C <= B And C <= A Then MinZeTri = C
End Function
```

```
Sub HledejMin()
Dim X As Integer
Dim Y As Integer
Dim Z As Integer
Dim Poc As Integer
Dim Vysledek As Integer

Randomize
Do
  Poc = Poc + 1
  X = Int(Rnd * 100)
  Y = Int(Rnd * 100)
  Z = Int(Rnd * 100)
  Vysledek = MinZeTri(X, Y, Z)
  MsgBox "Nejmenší z čísel " & X & " " & Y & " " & Z & " je "
& Vysledek
Loop While Poc < 10
End Sub
```

```
Sub Uprav(ByRef A As Integer, ByRef B As Integer)
Rem V proměnných A a B zůstanou stejné hodnoty, ale
Rem tak, že v A bude menší z nich a v B větší

Rem parametry musí být volány referencí !!!!

Dim Pom As Integer
Rem pomocná proměnná pro výměnu

If A > B Then
  Pom = A
  A = B
  B = Pom
End If
End Sub
```

```
Sub PokusRef()
```

```
Dim X As Integer
Dim Y As Integer

X = 12
Y = 8

Uprav X, Y
Uprav Y, 6
Rem Odkrokuj a ověř hodnoty proměnných
End Sub
```

```
Function NSD(ByVal A As Integer, ByVal B As Integer) As
Integer
'Zjisti nejvetsi spolecny delitel A a B
Dim Pom As Integer

Do While A <> B
  If A < B Then
    Pom = A: A = B: B = Pom
  End If
  A = A - B
Loop
NSD = A
End Function
```

```
Function NSN(ByVal A As Integer, ByVal B As Integer) As
Integer
'Zjisti nejmensi spolecny nasobek A a B

NSN = A * B / NSD(A, B)
End Function
```

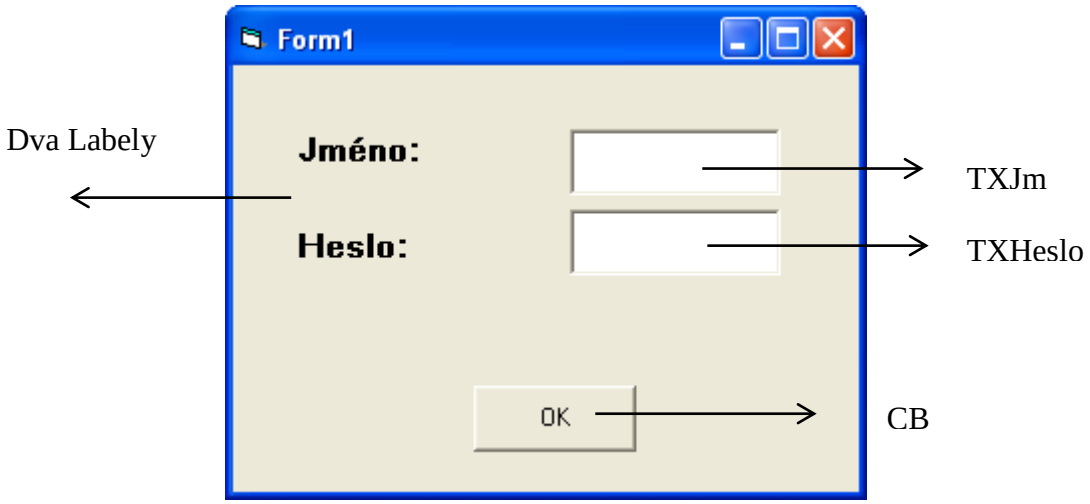
```
Sub Pokus1()
Dim X As Integer
Dim Y As Integer
Dim Pom As Integer

X = InputBox("Zadej 1. cislo ")
Y = InputBox("Zadej 2. cislo ")
Pom = NSD(X, Y)
MsgBox "Je cislo " & Pom, , "Nejvetsi spol delitel cisel " & X
& " a " & Y
End Sub
```

```
Sub Kraceni()
Dim X As Integer
Dim Y As Integer
Dim Pom As Integer
```

```
X = InputBox("Citatel ")
Y = InputBox("Jmenovatel ")
Pom = NSD(X, Y)
MsgBox X & "/" & Y & " = " & X \ Pom & "/" & Y \ Pom, ,
"Kraceni"
End Sub
```

16.15. Přihlášení



Kód formuláře

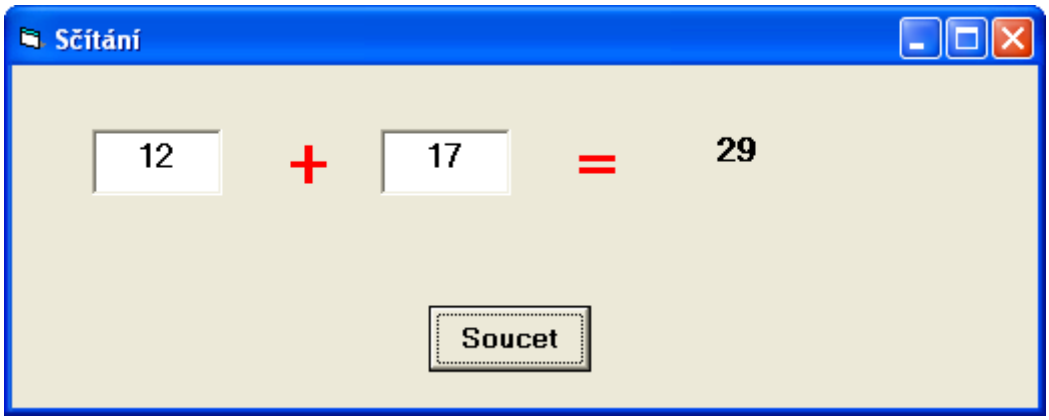
```
Private Sub CB_Click()
Static Poc As Integer
If UCase(TXJm.Text) = "ADMIN" And UCase(TXHeslo.Text) = "AAA"
Then
  MsgBox "Vítám administrátora"
  Poc = 0
Else
  If UCase(TXHeslo.Text) = "BBB" Then
    MsgBox "Vítám uživatele jménem " & TXJm.Text
    Poc = 0
  Else
    MsgBox "Přístup zamítnut"
    Poc = Poc + 1
  End If
End If
If Poc = 3 Then
  MsgBox "Třetí špatné přihlášení"
  CB.Enabled = False
End If
```

```
TXJm.SetFocus
End Sub
```

```
Private Sub TXHeslo_GotFocus()
Rem Při ztrátě fokusu se text v TextBoxu vymaže
TXHeslo.Text = ""
End Sub
```

```
Private Sub TXJm_GotFocus()
Rem Při získání zaměření bude text v textBoxu vybrán
TXJm.SelStart = 0
TXJm.SelLength = Len(TXJm.Text)
End Sub
```

16.16. Počty



Do TextBoxů uživatel zadá čísla. Po kliknutí na tlačítko Součet se do Labelu zapíše jejich součet.

Kód formuláře

```
Option Explicit

Private Sub CB_Click()
Dim A As Integer
Dim B As Integer
Dim Soucet As Integer

A = Val(TXCisloA.Text)
B = Val(TXCisloB.Text)
Soucet = A + B
LBVysl.Caption = Soucet
TXCisloA.SetFocus
End Sub
```

```
Private Sub TXCisloA_GotFocus()
TXCisloA.SelStart = 0
TXCisloA.SelLength = Len(TXCisloA.Text)
End Sub
```

```
Private Sub TXCisloB_GotFocus()
TXCisloB.SelStart = 0
TXCisloB.SelLength = Len(TXCisloB.Text)
End Sub
```

```
Private Sub TXCisloA_Change()
LBVysl.Caption = ""
End Sub
```

```
Private Sub TXCisloB_Change()
LBVysl.Caption = ""
End Sub
```

```
Private Sub TXCisloA_KeyPress(KeyAscii As Integer)
Rem Jiné znaky než číslice se nepřijmou
  If KeyAscii > Asc(" ") And (KeyAscii < Asc("0") Or KeyAscii > Asc("9")) Then
    KeyAscii = 0
  End If
End Sub
```

```
Private Sub TXCisloB_KeyPress(KeyAscii As Integer)
  If KeyAscii > Asc(" ") And (KeyAscii < Asc("0") Or KeyAscii > Asc("9")) Then
    KeyAscii = 0
  End If
End Sub
```

Poznámka : Elegantnější řešení by bylo použít dvouprvkové pole TextBoxů místo TXCisloA a TXCisloB. Procedury událostí spojené s těmito TextBoxy by se pak psaly jen jednou.