Moderné vzdelávanie pre vedomostnú spoločnosť / Projekt je spolufinancovaný zo zdrojov EÚ

Zuzana Krištofová
Eva Uličná

Baltík pre mierne pokročilých

2014

Publikácia bola vydaná a financovaná z prostriedkov ESF v rámci národného projektu Profesijný a kariérový rast pedagogických zamestnancov.
ITMS kód projektu 26120130002
ITMS kód projektu 26140230002

BALTÍK PRE MIERNE POKROČILÝCH

Zuzana Krištofová

Eva Uličná

Bratislava 2014

OBSAH

ÚVOD

Vzdelávací program Baltík – programovanie na základnej škole – úroveň mierne pokročilý, ku ktorému je vytvorený tento učebný zdroj, nadväzuje na vzdelávací program Baltík – programovanie na základnej škole (úroveň začiatočník).

Podmienkou zaradenia účastníka na vzdelávanie je znalosť Baltíka na úrovni začiatočník či už absolvovaním vzdelávacieho programu, alebo overením profesijných kompetencií z tohto programu. Na úvodných hodinách môžu lektori použiť úlohy z dvoch učebných zdrojov k vzdelávaciemu programu Baltík – programovanie na základnej škole – úroveň začiatočník, ktoré sú zverejnené na webovej stránke Metodicko-pedagogického centra:

Baltík – kniha: http://www.mpc-edu.sk/library/files/baltik_kniha_web.pdf

Baltík – pracovné listy: http://www.mpc-edu.sk/library/files/baltik___pracovne_listy_web.pdf

Prezenčná forma vzdelávacieho programu má 40 hodín, ktoré je možné realizovať napr. formou celodenných sústredení (po 8 vyučovacích hodín) alebo formou dopoludňajších alebo popoludňajších stretnutí (po 6 vyučovacích hodín).

Vzdelávací program je určený pre kategórie učiteľ a vychovávateľ a podkategórie učiteľ pre primárne vzdelávanie, učiteľ pre nižšie stredné vzdelávanie a učiteľ pre kontinuálne vzdelávanie.

Požiadavky na ukončenie vzdelávacieho programu pre účastníkov vzdelávania:

- 80 % osobná účasť na prezenčnej forme štúdia,
- vypracovanie metodického listu na vyučovaciu hodinu informatickej výchovy/informatiky,
- vypracovanie vlastného programu (podľa zadania lektora) s použitím odučených nástrojov programovacieho jazyka Baltík – úroveň mierne pokročilý,
- záverečná prezentácia vlastného programu a metodického listu pred lektorom a účastníkmi kontinuálneho vzdelávania.

Požiadavky na ukončenie vzdelávacieho programu pre pedagogických zamestnancov, ktorí sa budú uchádzať o overenie profesijných kompetencií získaných výkonom pedagogickej činnosti alebo sebavzdelávaním (podľa § 35 odsek 6 zákona č. 317/2009 Z. z.):

- vypracovanie metodického listu na vyučovaciu hodinu,
- vypracovanie vlastného programu s použitím všetkých odučených nástrojov programovacieho jazyka Baltík – úroveň mierne pokročilý,

- záverečná prezentácia vlastného programu a metodického listu pred trojčlennou skúšobnou komisiou vymenovanou poskytovateľom aktualizačného vzdelávania.

Za absolvovanie aktualizačného vzdelávania v rozsahu 60 hodín získa absolvent 12 kreditov a 2 kredity za ukončenie vzdelávania záverečnou prezentáciou, čiže spolu 14 kreditov.

Táto publikácia zahŕňa nástroje, ktoré sú obsahom vzdelávacieho programu Baltík – programovanie na základnej škole – úroveň mierne pokročilý. Ku každému nástroju sú buď riešené príklady aj s ich zobrazením v prostredí Baltík, alebo zadania na možné úpravy už riešených príkladov.

Metodika vyučovania programovacieho jazyka Baltík, jej jednotlivých nástrojov bude na vzdelávaní preberaná v nadväznosti na pokročilejšie ovládanie programovacieho jazyka.

1 NÁHODNÉ ČÍSLO

V programe sa môže na prvý pohľad diať náhodne všetko možné. Na náhodnom mieste sa môže objaviť nejaký predmet, samotný predmet môže byť zvolený náhodne, Baltík môže urobiť náhodný počet krokov, otočiť sa náhodným smerom atď. V skutočnosti sa za všetkými týmito situáciami skrýva **náhodné číslo.**

Základ príkazu tvorí prvok **`náhodné číslo`**  :

- samotné použitie prvku – vráti **reálne číslo** v intervale $\mathbf{0 \leq a < 1}$,
- prvok spolu s číslom umiestneným za ním, napr.  – vygeneruje sa číslo **z prvých 20 nezáporných celých čísel, t. j. číslo od 0 po 19**,
- prvok s číselným intervalom, napr.  – vygeneruje sa **číslo od 30 do 45.**

Príklady použitia:

 – na obrazovke sa zobrazí číslo 17

 – na obrazovke sa zobrazí náhodné číslo od 0 do 19

 – kačička sa vyčaruje na súradniciach 6, 4

– kačička sa vyčaruje na náhodných súradniciach kdekoľvek na pracovnej ploche

– kačička sa vyčaruje na náhodné súradnice *X* v rozmedzí od 5 do 10 a *Y* v rozmedzí od 3 do 8

– vyčaruje sa predmet číslo 37

– vyčaruje sa náhodný predmet od 0 do 19

– vyčaruje sa náhodný predmet od 30 do 45

– Baltík sa posunie dopredu o 4 políčka

– Baltík sa posunie dopredu o náhodný počet políčok od 0 do 5

– Baltík sa posunie dopredu o náhodný počet políčok v rozmedzí od 4 do 10

– Baltík sa otočí v smere číslo 2, t. j. na juh

– Baltík sa otočí náhodným smerom

2 JEDNODUCHÝ PODMIENENÝ PRÍKAZ

V bežnom živote sa musíme často rozhodovať. Rozhodnutia robíme na základe situácie, ktorá nastala. Ráno sa obliekame podľa toho, aké má byť počasie. Ak učiteľ učí zaujímavo, žiaci počúvajú, inak úspešne driemu. Rovnako sa rozhodujeme aj v programoch.

Či už sa bavíme o situáciách z bežného života, alebo o programoch, model formulácie podmienky je vždy rovnaký:

Ak (platí podmienka) {urobí sa niečo}

inak {urobí sa niečo iné}

V prípade, že **inak sa neurobí nič,** zvykneme používať iba prvý riadok našej vety – vtedy hovoríme v programovaní o **jednoduchom podmienenom príkaze.** Ak je veta úplná a zahŕňa

aj to, čo sa stane vtedy, ak podmienka nebola splnená, hovoríme o **úplnom podmienenom príkaze.**

V programovaní slovo **ak** vyjadrujeme anglickým **if, inak** anglickým **else.**

Príklad 1

Na spodnom riadku pracovnej plochy Baltíka sa na náhodných súradniciach (ale mimo miesta, kde stojí Baltík) vyčaruje (alebo priradí na obrazovku) 5-krát hríbik. Úlohou Baltíka je prejsť na druhú stranu a cestou hríbiky pozbierať.

Riešenie

Keďže nevieme, kde hríbiky sú, musí sa Baltík pri každom posune na ďalšie políčko rozhodnúť, či pred ním hríbik je alebo nie.

Najskôr si hríbiky vytvoríme:

Teraz potrebujeme, aby sa Baltík 14-krát posunul o jedno políčko dopredu, ale aby pred každým posunom zistil, či je pred ním hríbik.

Náš podmienený príkaz by mohol vyzerať takto:

Ak je pred Baltíkom hríbik, odčaruj ho.

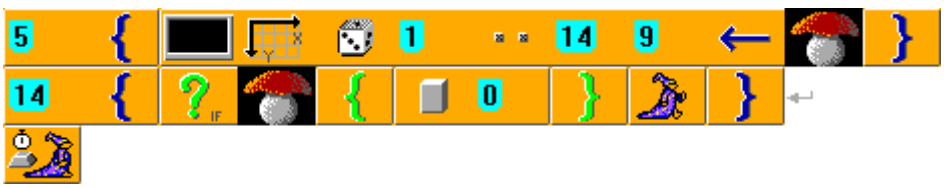
Príkaz **if** nájdeme pod ikonkou  .

Podmienka **ak je pred Baltíkom hríbik:**  .

Za podmienkou nasleduje, čo sa má urobiť, ak táto podmienka platí. V našom prípade na mieste hríbika vyčarujeme predmet číslo 0, čo je vlastne prázdne miesto:

Pozor! Pre príkazy za podmienkou platí, že ak sa na základe podmienky má vykonať iba jeden príkaz, nemusí sa nachádzať v zložených zátvorkách, t. j. v bloku príkazov (ale môže). Pri väčšom počte príkazov je použitie bloku príkazov nevyhnutné.

Po otestovaní políčka prejde Baltík na ďalšie políčko. Toto celé sa musí zopakovať 14-krát:



Príklady ďalších podmienok

 – ak je akýkoľvek predmet pred Baltíkom

 – ak je Baltík na súradniciach 2, 7

 – ak sa myš nachádzala pri stlačení ľavého tlačidla myši na súradniciach 7, 2

Poznámka 1! Prvok ľavého tlačidla myši získame vložením ikonky **`myš`**  do programu. Otvorí sa nám ponuka (pozri obrázok), v ktorej klikneme na tú ikonku, ktorú práve potrebujeme.

Poznámka 2! Ak chceme, aby túto podmienku program vyhodnotil správne, musí jej predchádzať príkaz pre načítanie stlačeného klávesu alebo tlačidla myši. Je potrebné si uvedomiť, že program robí len to, čo musí. Ak mu priamo nepovieme „všímaj si, či niečo bolo stlačené a zapamätaj si, čo to bolo“, tak program nič také neeviduje. Na toto nám budú slúžiť dva podobné príkazy:

 – **`čítaj kláves alebo tlačidlo myši (čakaj na stlačenie)`** – ak vložíme tento príkaz do programu, na tomto mieste sa program zastaví a bude čakať, kým niečo nestlačíme. Po stlačení klávesu alebo tlačidla myši sa pohne ďalej a to, čo bolo stlačené, si pamätá, preto sa na to následne v podmienkach môžeme pýtať.

 – **`čítaj kláves alebo tlačidlo myši (nečakaj na stlačenie)`** – tento príkaz funguje veľmi podobne. Rozdiel je len v tom, že na mieste, kde sa nachádza, sa program nezastaví, ale iba skontroluje, či bolo niečo stlačené, a hneď pokračuje ďalej. Pokiaľ bolo niečo stlačené, zapamätá si to rovnako ako v predchádzajúcom príkaze.

Na obrázku je ukážka použitia:

 – ak sa predmet číslo 63 (slniečko) nachádza na súradniciach myši pri stlačení ľavého tlačidla

 – ak je Baltík otočený na západ

 – ak sa kurzor myši nachádza na súradniciach 8, 5 (k danému prvku sa dostaneme opäť z prvku **myš**)

 – ak sa kurzor myši nachádza na súradniciach Baltíka

 – ak sa číslo predmetu na súradniciach 6, 5 rovná číslu predmetu na súradniciach 8, 5, alebo zjednodušene: ak sú na súradniciach 6, 5 a 8, 5 rovnaké predmety

3 ÚPLNÝ PODMIENENÝ PRÍKAZ

Príklad

Vytvorme program, v ktorom bude Baltík bežať dopredu, ak na ňom bude kurzor myši. V opačnom prípade sa bude točiť na mieste, kým naň opäť nenastavíme kurzor myši.

Riešenie

Našu podmienku formulujeme takto:

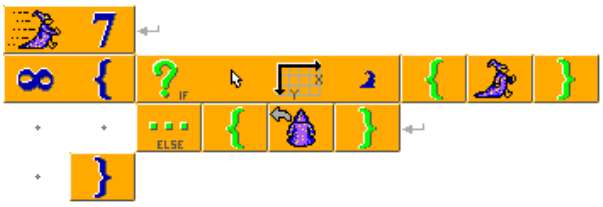
Ak je kurzor myši na Baltíkovi, posuň sa o jedno políčko dopredu, inak sa otoč doľava.

Túto podmienku necháme opakovať donekonečna.

Na riešenie potrebujeme nový príkaz pre slovo **inak = else.** V programovacom jazyku Baltík ho zastupuje prvok: 

.

Za touto ikonkou budú opäť nasledovať príkazy, ktoré v prípade, že je ich viac ako jeden, musíme dať do bloku príkazov:



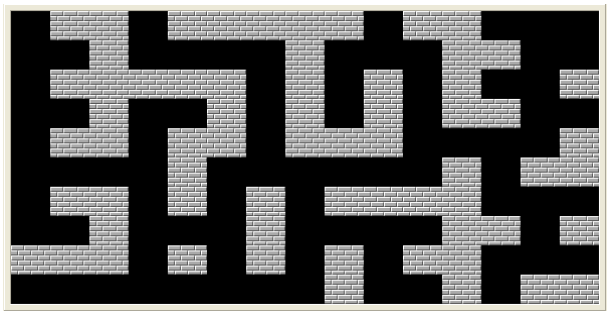
Skúsme teraz našu hru trochu vylepšiť.

Námet č. 1: Na pozadí sa objaví bludisko. Úlohou hráča bude dostať Baltíka do pravého horného rohu obrazovky, ale Baltík pritom nebude schopný prechádzať stenami (pozri obrázok v riešení).

Námet č. 2: Na náhodnom mieste na obrazovke sa objaví sklenená banka s vodou (banka predmetov č. 1). Úlohou hráča bude dostať Baltíka k banke. V momente, keď sa ocitne na banke s vodou, prestane sa pohybovať, objaví sa blahoželanie k úspechu a hra sa skončí. Ešte lepšie bude, keď sa na konci objaví čas, za ktorý sa to hráčovi podarilo.

Riešenie námetu č. 1

K riešeniu si budeme musieť vytvoriť scénu, napr. ako na tomto obrázku:



Na zabezpečenie, aby Baltík nechodil cez steny bludiska, budeme potrebovať nový prvok 

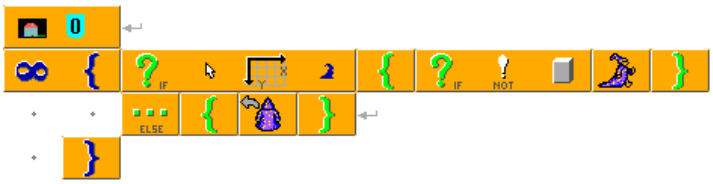
, ktorý sa stará v podmienkach o slovíčko **NIE.**

Porovnajme:

– ak **JE** predmet pred Baltíkom

– ak **NIE JE** predmet pred Baltíkom

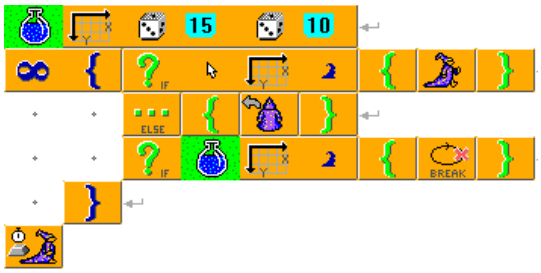
Náš program potom môže vyzerať takto:



Riešenie námetu č. 2

Na začiatku programu musíme vyčarovať na náhodné súradnice banku s vodou. To, čo ešte nevieme, je ukončiť nekonečný cyklus, keď Baltík dorazí na banku s vodou. Postupne sa naučíme takéto situácie riešiť inak, ale zatiaľ využijeme príkaz **`ukončenie cyklu`**  . Ak program vykonáva nejaký cyklus – môže byť aj nekonečný – a narazí v ňom na tento príkaz, vybehne z tohto cyklu von a pokračuje príkazmi za ním.

V našom prípade vždy po podmienke, ktorá rieši Baltíkov pohyb, musí nasledovať ďalšia, ktorá sa pýta, či je na súradniciach Baltíka banka s vodou:

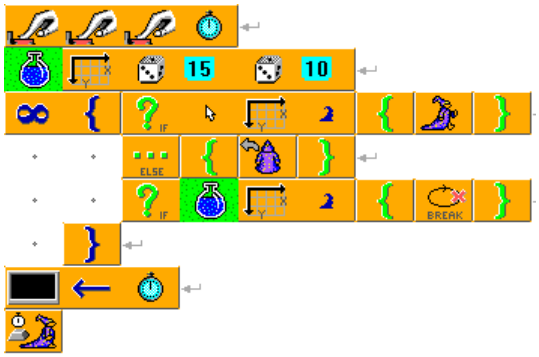


Na meranie času potrebujeme ďalšie dva prvky:

 – **`stlačené tlačidlo`** a  – **`stopky`**.

Kombinácia týchto príkazov  , ktorú vložíme na začiatok programu, spôsobí, že sa **`zastavia, vynulujú a spustia stopky`**. Alebo inak – nastavíme si a spustíme stopky.

Na konci si ich aktuálnu hodnotu necháme vypísať na obrazovku.



4 VNORENÁ PODMIENKA

Niekedy sa stane, že sa budeme rozhodovať medzi viacerými možnosťami.

Príklad z bežného života – rozhodovanie v cukrárni:

ak (budú mať veterník) {kúp 2 veterníky}

inak ak (budú mať krémeše) {kúp 2 krémeše}

inak ak (budú mať laskonky) {kúp 4 laskonky}

inak {kúp, čo chceš}

Táto formulácia znamená: Ak budú v cukrárni veterníky, to, či sú tam ďalšie zákusky, nie je dôležité. Iba ak nebudú, začneme sa zaujímať o krémeše, ak nebudú ani tie, skúšame laskonky a na záver dopĺňame, čo má dotyčný kúpiť, ak nebude platiť ani jedna z predchádzajúcich podmienok. Vždy sa uskutoční len jedna z daných možností. V tom je rozdiel oproti tomu, ak by podmienka bola formulovaná takto:

ak (budú mať veterník) {kúp 2 veterníky}

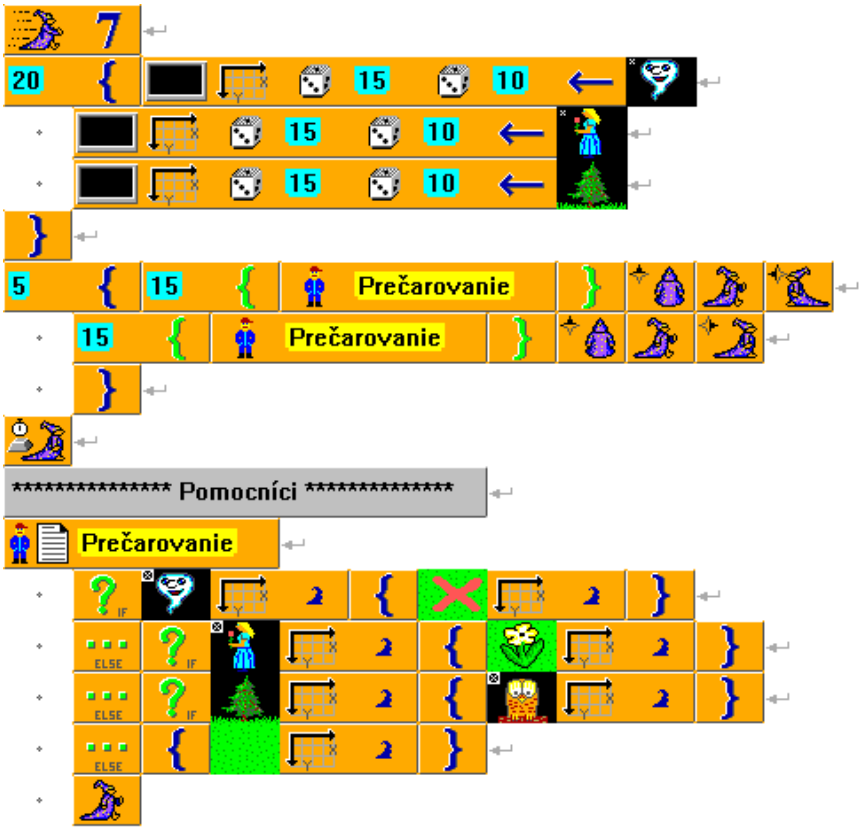
ak (budú mať krémeše) {kúp 2 krémeše}

ak (budú mať laskonky) {kúp 4 laskonky}

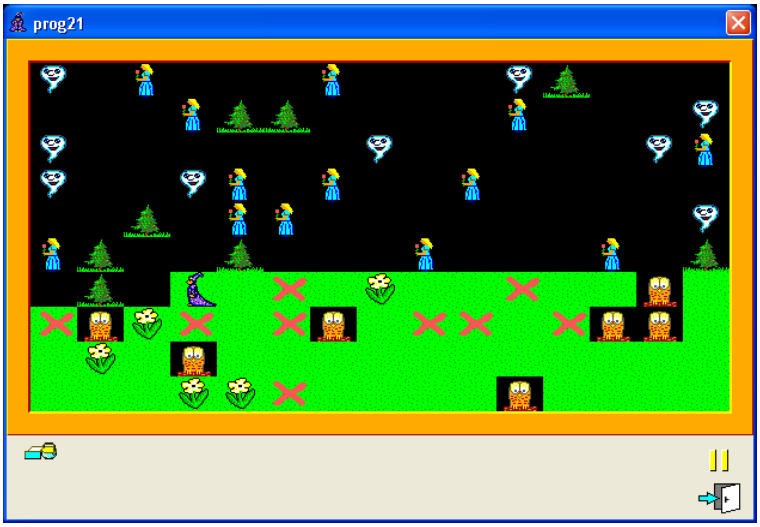
inak {kúp, čo chceš}

Rovnako to bude aj v programe.

Skúste najprv odhadnúť a potom vyskúšať, čo bude robiť nasledujúci program:



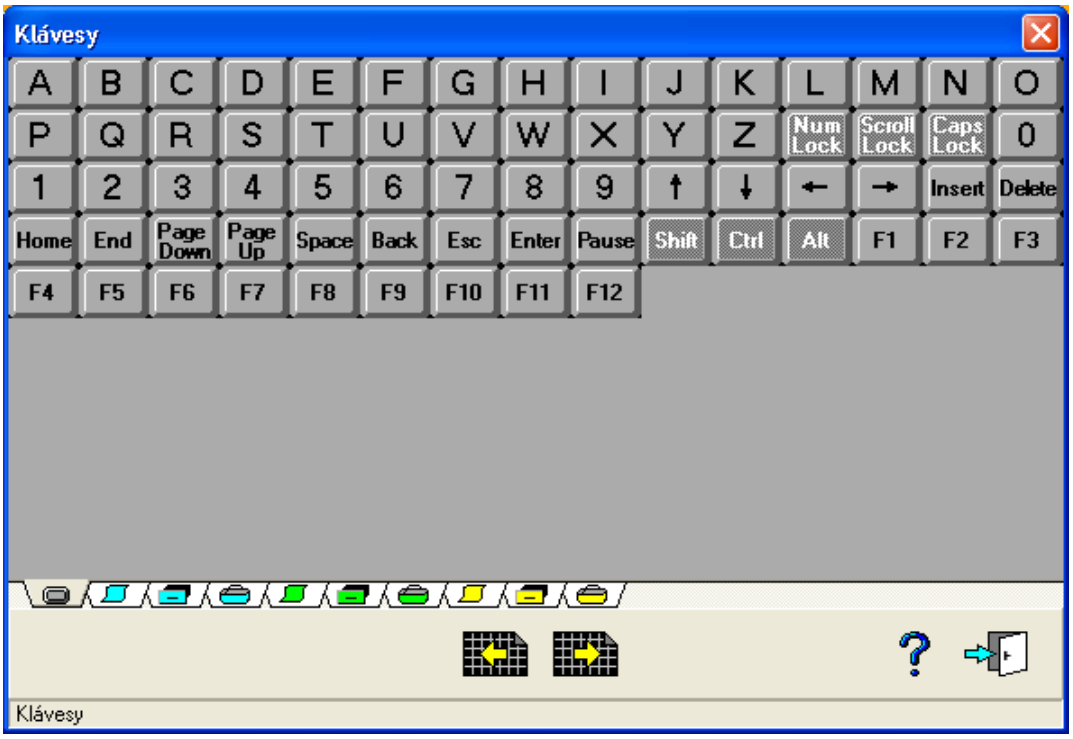
Ak nikde nenastala chyba, v programe sa najprv na náhodných miestach objavia duchovia, princezné a stromčeky. Následne Baltík prejde celú pracovnú plochu a každý z predmetov prečarúva na niečo iné. Ak na danom mieste nie je nič, dá tam zelenú trávu (pozri obrázok).



5 OVLÁDANIE PROGRAMU POMOCOU KLÁVESOV

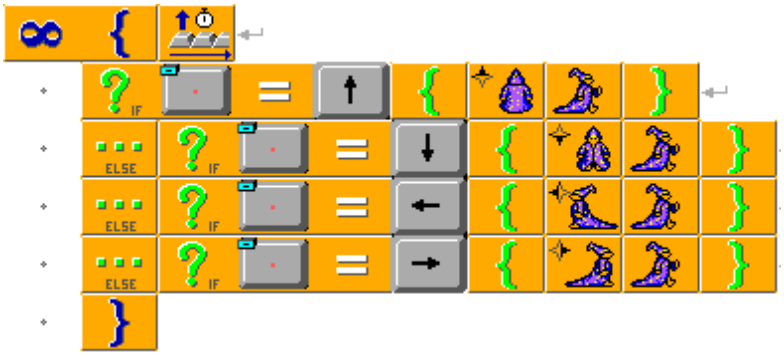
V mnohých hrách používame na ovládanie rôznych postavičiek klávesy. Budeme na to potrebovať:

- 
-  – ikonu **`nejaký kláves`** (jej význam pochopíme o chvíľu)
- ikony pre klávesy – nájdeme ich na paneli nástrojov pod tlačidlom *Klávesy, konštanty, premenné*  . Po jeho stlačení sa otvorí okno. V jeho spodnej časti sa nachádzajú záložky, klikneme ľavým tlačidlom myši na prvú z nich. V tejto záložke máme uložené všetky ikony pre klávesy (pozri obrázok).



Klikneme na vybraný kláves a vložíme ho do *Pracovnej plochy príkazov* (rovnakým spôsobom ako *predmet* alebo *prvok*).

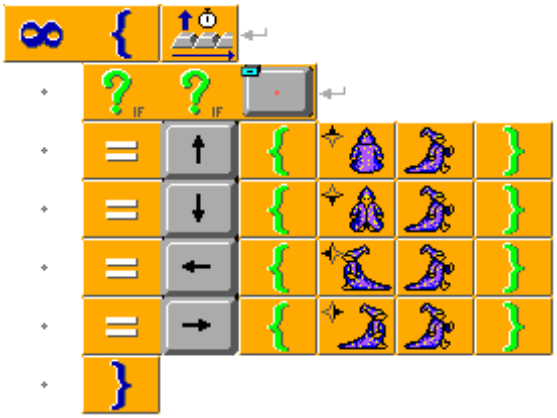
Program, v ktorom ovládame pohyb Baltíka pomocou šípok:



Tento zápis je zbytočne zložitý. Pre prípady, keď sa ten istý objekt vyskytuje v každej podmienke podmieneného príkazu, len má vždy inú hodnotu, sa dá celý podmienený príkaz zapísať jednoduchšie. Touto „vecou“ môže byť napr.:

- kláves (to je náš prípad)  ,
- náhodné číslo  ,
- číslo predmetu  ,
- a ešte mnoho ďalších, o ktorých budeme hovoriť neskôr.

Jednoduchší zápis bude vyzerať takto:

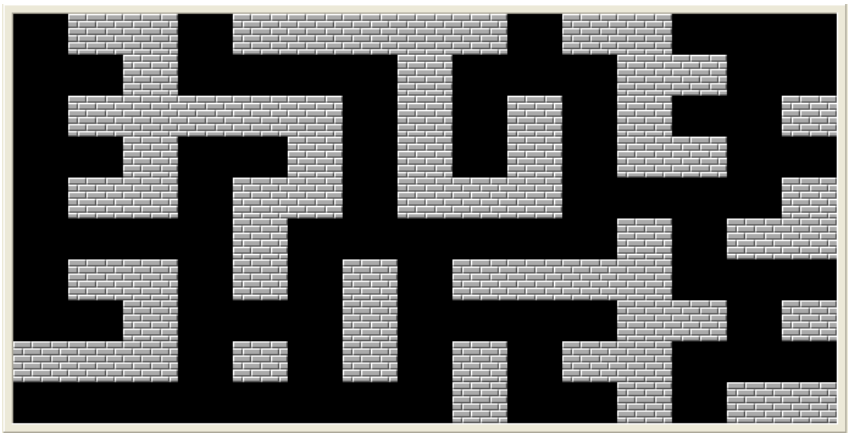


Tomuto druhu vetvenia hovoríme **„switch – case“** alebo **prepínač.**

Prvok  na začiatku zápisu je použitý dvakrát za sebou, aby program pochopil, že chceme použiť iný druh zápisu podmieneného príkazu. Za ním nasleduje prvok („vec“), na základe hodnoty ktorého sa program rozhoduje, čo urobí.

Každý prípad (po anglicky *case*) s inou hodnotou klávesu dávame do nového riadka, ktorý sa začína znamienkom =. Za hodnotou klávesu nasleduje blok príkazov. V ňom sú vložené príkazy, ktoré sa majú vykonať v prípade, že bol stlačený práve tento kláves.

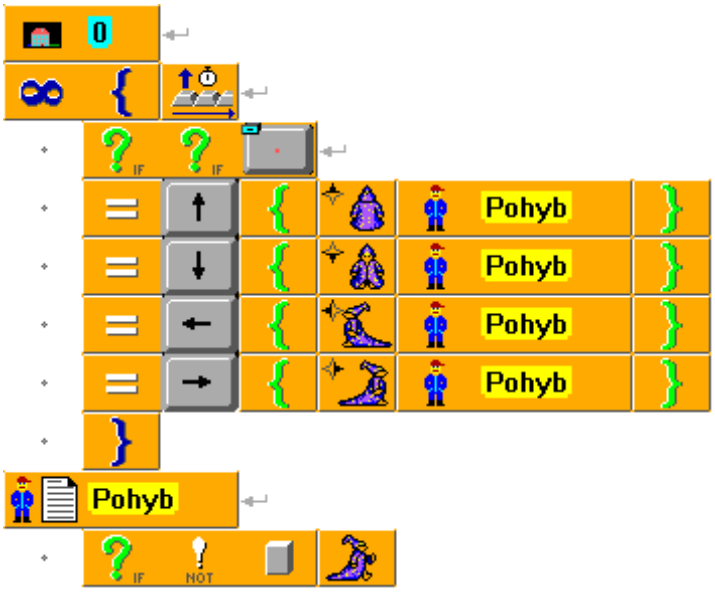
Teraz si program otestujeme. Do pozadia dáme scénu s bludiskom, napr.:



Keď program spustíme, zistíme, že Baltík chodí cez steny. Program hovorí, aby sa po otočení vždy posunul dopredu. V našom prípade mu musíme povedať: „Ak pred tebou nie je žiaden predmet, choď dopredu.“ Týmto podmieneným príkazom musíme nahradiť príkaz **`posuň Baltíka o 1 pole vpred`** pri stlačení každej šípky.

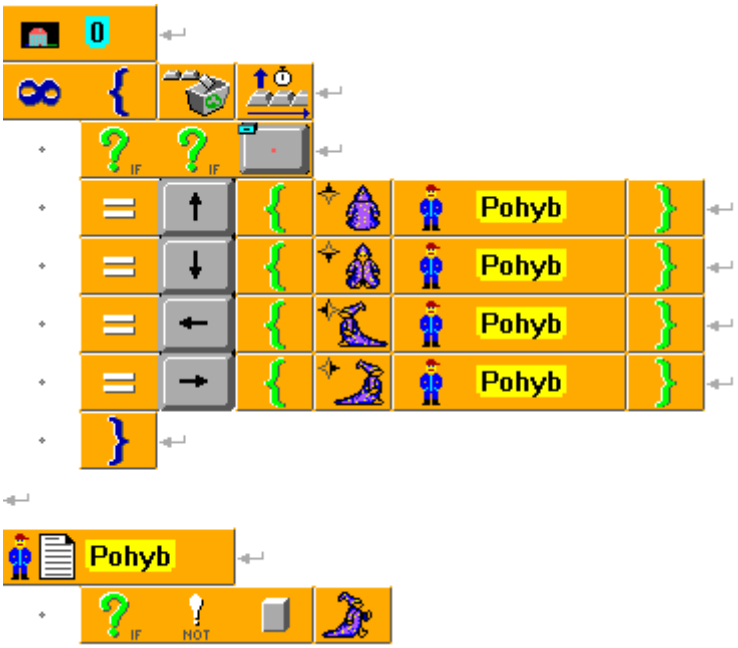
Aby sme program mali naprogramovaný efektívne a nepísali rovnaký zdrojový kód na 4 rôzne miesta, využijeme na tento účel pomocníka.

Riešenie:



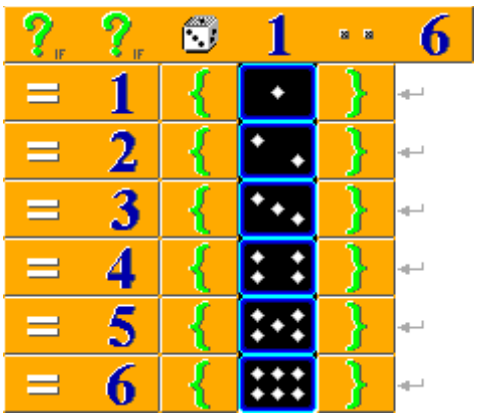
Program je takmer hotový, zostáva len maličkosť. Keď podržíme prst na šípke trochu dlhšie, Baltík pokračuje v pohybe, hoci sme už šípku dávno pustili. Čo s tým?

Pokúsme sa najprv pochopiť, čo sa deje. Keď držíme stlačený kláves dlhšie, je to rovnaké, ako keby sme ho stlačili niekoľkokrát za sebou. Každé stlačenie si program pamätá a poctivo posiela Baltíka daným smerom toľkokrát, koľko stlačení danej šípky napočítal. Ak chceme, aby si program zo svojej pamäte vyprázdnil stlačenia, ktoré nastali v čase presúvania Baltíka, a staral sa len o kláves, ktorý bol stlačený práve v tejto chvíli, použijeme príkaz **`vyprázdni frontu kláves`**  . Umiestnime ho tesne pred prvok  . Príkaz spôsobí, že všetky staré stlačenia program zabudne a bude sa riadiť len tým, čo bolo stlačené až po príkaze pre vyprázdnenie frontu kláves.

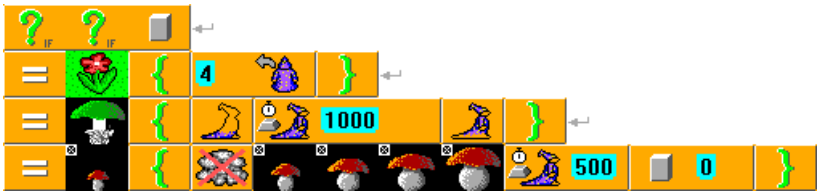


Ďalšie príklady využitia vetvenia **„switch – case“:**

- podľa hodnoty náhodného čísla Baltík vyčaruje pred seba príslušný predmet



- Baltík sa správa podľa toho, aký predmet sa pred ním nachádza.



6 CYKLUS WHILE

Vráťme sa k úlohe, keď sme sa pokúšali dostať k banke s vodou. Ak sme chceli, aby program skončil vo chvíli, keď Baltík dôjde na nádobku, museli sme to riešiť trochu „násilným“ spôsobom.

Toto riešenie je dosť zložité a neprirodzené. Normálne ľuďom nehovoríme: „Rob toto donekonečna, a ak to budeš mať hotové, potom skonči!“ Oveľa prirodzenejšie je: „Pokiaľ toto nebudeš mať hotové, musíš na tom pracovať!“ Podobne je to v programovaní.

V našom programe povieme Baltíkovi:

POKIAĽ nie je banka s vodou na súradniciach Baltíka {podmienený príkaz pre pohyb}

Nekonečno dáme preč a na jeho miesto umiestnime podmienku, na základe ktorej má cyklus prebiehať.

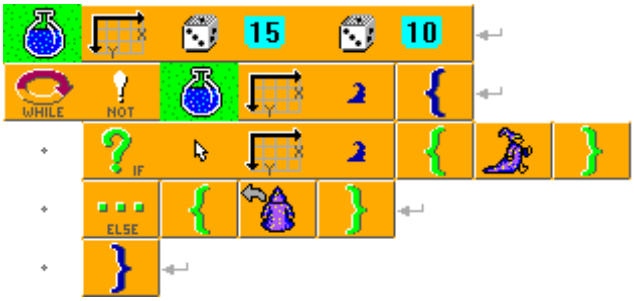
Slovíčko **pokiaľ** sa po anglicky povie **while,** príslušná ikona je 

. Po vložení do programu sa rozbalí ponuka dvoch ikon, je na nich napísané **`while`** a **`do-while`** 

.

Kliknutím ľavým tlačidlom myši vyberieme hornú ikonu. Za ňu dosadíme podmienku:

Celé riešenie:



Tento cyklus nazývame **cyklus s podmienkou.**

Program ho spracuje nasledovne:

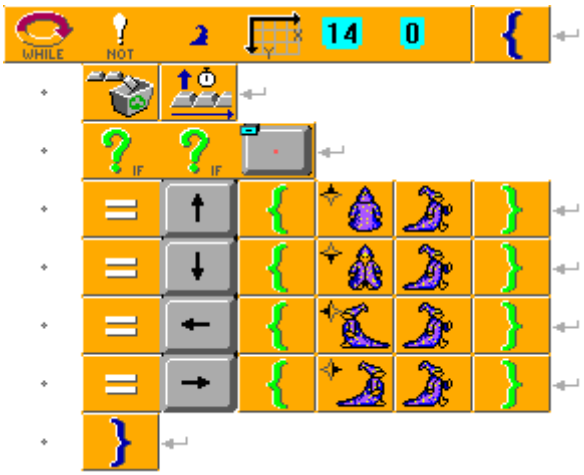
1. Prečíta si podmienku a vyhodnotí, či je pravdivá.
2. Ak áno, príkazy v cykle sa vykonajú a program sa opäť vráti na jeho začiatok, k podmienke (t. j. k bodu 1). Ak nie, príkazy sa nevykonajú a cyklus sa ukončí.

Takto sa program stále pohybuje v danom cykle, až kým nezistí, že podmienka prestala platiť.

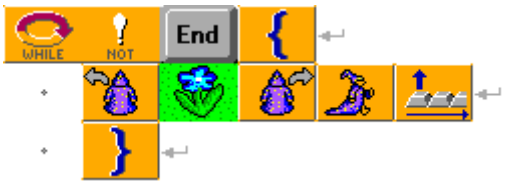
Niekoľko ukážok využitia **cyklu while:**

- Baltík pôjde dopredu, kým nenarazí na nejaký predmet alebo okraj plochy.

- Baltíka môžeme ovládať pomocou šípok, dokiaľ nedorazí na súradnice 14, 0.



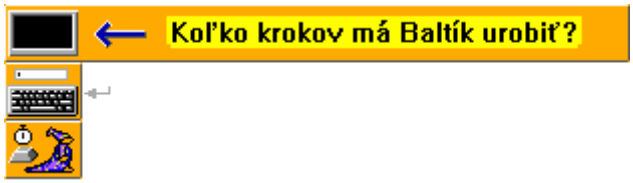
- Dokiaľ nebude stlačený kláves *End*, Baltík bude do susedného radu sadiť kvietky (na stlačenie nebude čakať).



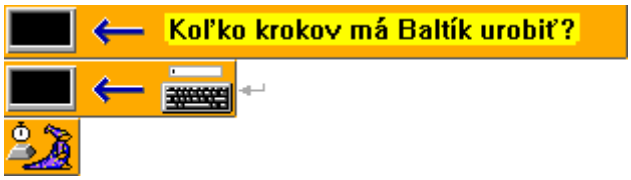
7 VSTUP Z KLÁVESNICE A PREMENNÉ

Vytvorme program, v ktorom bude môcť jeho používateľ napísať, koľko krokov má Baltík urobiť. Číslo zostane zobrazené na obrazovke a zároveň Baltík urobí daný počet krokov.
Na začiatku sa musí objaviť žiadosť o zadanie počtu krokov (nápis sa vypíše do ľavého horného rohu):

Následne potrebujeme získať údaj priamo z klávesnice. Potrebujeme na to príkaz **`čítaj číslo alebo reťazec z klávesnice`**  , ktorý spôsobí, že sa v spustenom programe objaví na obrazovke okienko. Do tohto okienka treba niečo vpísať. Zápis ukončíme klávesom *Enter*. S týmto vstupom z klávesnice môžeme ďalej pracovať. Vyskúšajme si program na nasledujúcom obrázku:



Okienko sa objavilo, číslo sme napísali, ale po stlačení klávesu *Enter* zasa to, čo sme napísali, zmizlo. Ako povedať programu, aby to tam zostalo?
Rovnako ako na obrazovku priraďujeme konkrétne texty, ktoré sa majú vypísať, môžeme programu povedať, aby to, čo načítal, zároveň priradil na obrazovku.



Ako docieliť, aby Baltík urobil toľko krokov, koľko sme zadali z klávesnice?

Tu sa dostávame k jadru problému. Hodnotu z klávesnice potrebujeme použiť viackrát – najskôr ako výpis na obrazovku a potom ako príkaz Baltíkovi, koľko krokov má urobiť. Preto musíme to, čo sme načítali z klávesnice, uložiť niekam, odkiaľ si túto hodnotu budeme brať. Na tento účel máme k dispozícii tzv. **premenné**, ktoré v programe Baltík nazývame **zásuvky**  a **košíky**  .

Čo sú to premenné

Premenné si môžeme predstaviť ako pomenované schránky. Ukladáme do nich hodnoty, ktoré chceme v programe použiť. Keď potrebujeme použiť uloženú hodnotu neskôr, len povieme programu, aby použil tú, ktorá je uložená v danej schránke – premennej.

Rozdelenie premenných

S premennými budeme uskutočňovať rôzne operácie a používať ich v rôznych situáciách. Na to však nevystačíme len s jedným univerzálnym druhom premenných.

Rozdelenie podľa dátového typu

Podľa toho, aký typ dát môžu premenné obsahovať, rozdeľujeme ich na:

 **celočíselné premenné (azúrové) –** ukladáme do nich celé čísla,

 **reálne premenné (zelené) –** ukladáme do nich reálne čísla,

 **reťazcové premenné (žlté) –** ukladáme do nich reťazce (text).

Premenné sú od seba jasne farebne rozlíšené, aby sme hneď videli, k akému typu patria.

Rozdelenie podľa viditeľnosti

Baltík (ale aj mnohé iné programovacie jazyky) rozoznáva dva druhy viditeľnosti premenných:

 **globálne premenné (zásuvky),**

 **lokálne premenné (košíky).**

Globálne premenné (zásuvky)

Globálne premenné sú viditeľné v celom programe. Sú to vyhradené miesta v pamäti počítača, ktoré sú rezervované pre hodnotu, ktorú do premennej vložíme. Keďže globálna premenná má platnosť v celom programe, existuje až do jeho konca a celý tento čas od svojho vzniku zaberá miesto v pamäti.

Lokálne premenné (košíky)

Lokálna premenná sa na rozdiel od globálnej premennej **môže vyskytovať len v pomocníkovi.** Ostatní pomocníci, ani samotný program k nej nemajú prístup.

Lokálna premenná vzniká so spustením pomocníka a zaniká po jeho ukončení. Aby sa pamäť počítača „nezahltila“, je ďaleko šetrnejšie používať lokálne premenné vždy, keď je to možné.

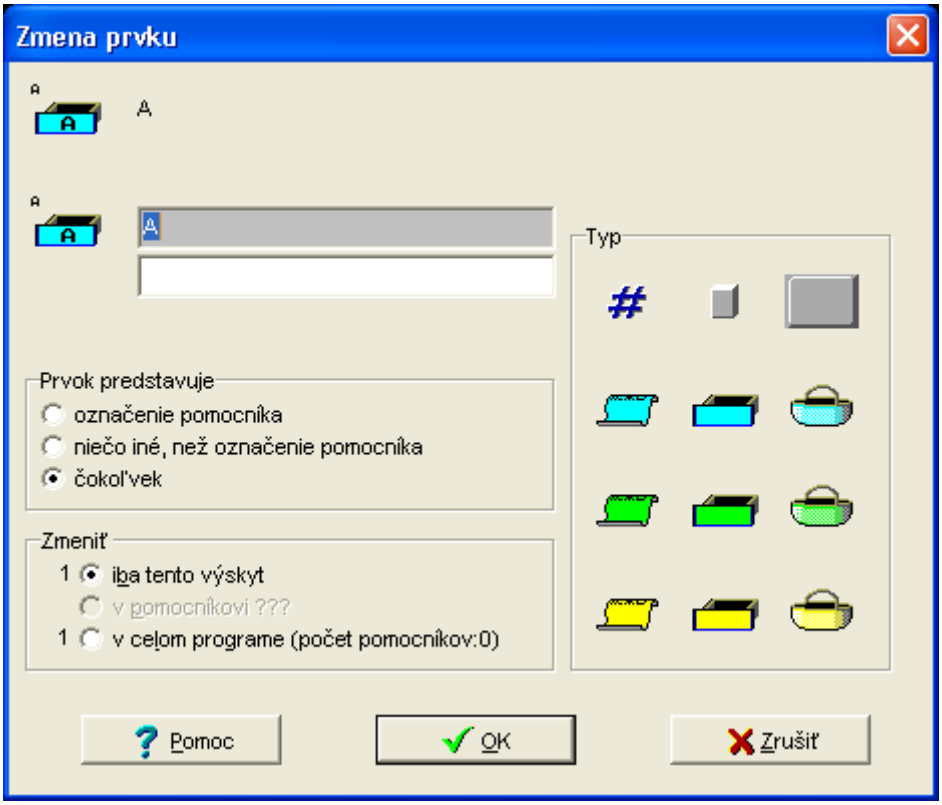
Lokálne premenné si môžeme predstaviť ako pomenované košíky, ktoré si nosí každý pomocník so sebou a nikomu nedovolí, aby sa mu do nich pozeral alebo sa v nich dokonca prehrabával. Môže sa stať, že dvaja pomocníci majú košíky s rovnakými názvami, sú to ale úplne samostatné premenné, ktoré spolu nijako nesúvisia. Oproti tomu existencia dvoch globálnych premenných s rovnakým názvom v jednom programe je vylúčená.

My sa budeme zatiaľ zaoberať použitím globálnych premenných.

K premenným sa dostaneme pomocou tlačidla  . Otvorí sa okno, v ktorom nás budú zaujímať záložky so zásuvkami (pozri obrázok). Sú tri – s modrou, zelenou a žltou zásuvkou – každá pre príslušný dátový typ.

- Vyberme si záložku s modrou zásuvkou. Vidíme pred sebou niekoľko rôznych súborov zásuviek. Prvé dva riadky sú už pomenované zásuvky. Môžu (ale nemusia) sa použiť tam, kde ich obrázky a názvy vystihujú účel, na ktorý ich použijeme. Ak sa presunieme na príslušnú premennú kurzorom myši, jej názov sa ukáže dole v stavovom riadku.
- Ďalších 6 riadkov tvoria klasické zásuvky (premenné), ktoré ešte nie sú pomenované, aj keď majú svoju značku, aby sa medzi sebou rozlíšili. S týmito budeme pracovať najčastejšie.
- Predposledný riadok je určený pre pokročilých programátorov – sú to premenné typu pole.
- Posledný riadok sa používa, ak do zásuvky potrebujeme odložiť hodnoty súradníc.

Vyberme si zásuvku s označením A a vložme ju do programu na miesto, kde ju potrebujeme použiť prvýkrát (pozri obrázok). Otvorí sa dialógové okno, v ktorom sa program pýta na názov premennej, jej počiatočnú hodnotu a ďalšie vlastnosti (pozri obrázok).



Do prvého sivého okienka píšeme názov premennej (ak nič nevpíšeme, program nám navrhuje rovnaký názov, ako je značka premennej).

Biele okienko slúži na zadanie vstupnej hodnoty. Tú môžeme, ale nemusíme zadať.

Pre náš príklad zmeníme iba názov premennej napr. na **krok.** Potvrdíme tlačidlom *OK*. Premennú máme zadefinovanú. Teraz načítanie z klávesnice priradíme do našej zásuvky **krok.** Na obrazovku už nepriradíme načítanie z klávesnice, ale premennú **krok,** v ktorej máme to, čo sa načítalo, odložené (pozri obrázok).

Ak má následne Baltík urobiť príslušný počet krokov – namiesto čísla použijeme našu premennú: 

Dajme všetky príkazy do bloku príkazov a zadajme počet opakovaní (môžeme dať nekonečno). Potom program vyskúšajme.

Ako môžeme vidieť, Baltík síce urobí smerom dopredu zadaný počet krokov, ale otázka sa pri každom ďalšom opakovaní zobrazuje tam, kde skončil výpis posledného čísla. Toto je potrebné upraviť.

Ak nezadáme presné súradnice pre výpis, text sa vypíše tam, kde sa práve nachádza kurzor pre výpis. Kurzor sa na začiatku programu nachádza v ľavom hornom rohu obrazovky a po každom výpise na konci tohto výpisu. To je dôvod, prečo sa ďalšie a ďalšie výpisy zobrazujú jeden za druhým.

Pomôcť sa tomu dá viacerými spôsobmi. Jedným z nich je, že povieme kurzoru – prvok  – na záver cyklu, aby sa presunul na súradnice 0, 0 (je jedno, či zvolíme bodové alebo políčkové).

Aby nám neprekážal predchádzajúci výpis, zmažeme obrazovku (pozri obrázok).

Vylepšime náš pôvodný program tak, aby Baltík dokázal behať po celej obrazovke. Skúsme túto možnosť: striedavo sa bude objavovať žiadosť o počet krokov a zadanie smeru (hodnota 1 – východ, 2 – juh, 3 – západ, 4 – sever).

Postup je podobný ako v pôvodnom programe. Na zmenu smeru využijeme príkaz, v ktorom zadávame smer pomocou čísla  (pozri obrázok).

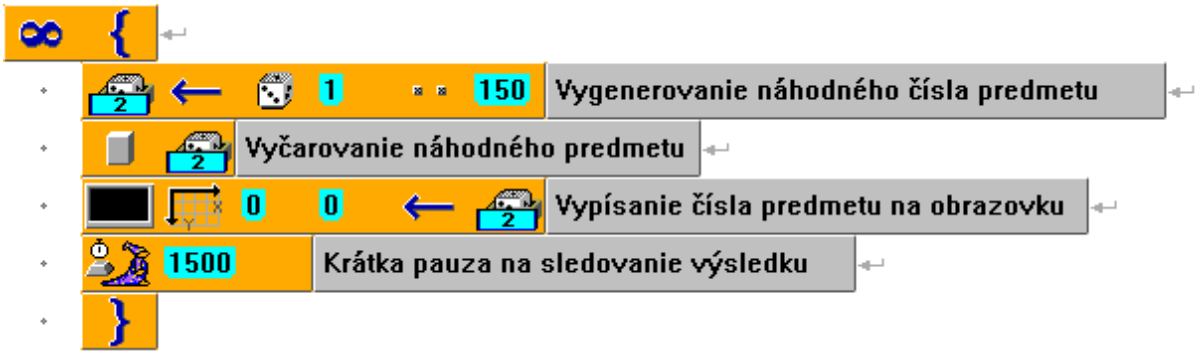
8 PÁR TIPOV PRE VÝPIS NA OBRAZOVKU

Úloha 1

Vytvorme program, v ktorom bude Baltík čarovať pred seba náhodné predmety z banky 0. Za každým čarovaním sa v ľavom hornom rohu ukáže číslo vyčarovaného predmetu.

Riešenie

Riešenie sa zdá na pohľad jednoduché a mohlo by vyzerať napr. takto:



Ak program spustíme, zistíme, že sa občas objavujú v ľavom hornom rohu divné čísla, ktoré žiaden predmet z banky 0 nemá, napr. 445 a pod. Je to preto, lebo pred predmetom 44 mal predchádzajúci náhodný predmet trojciferné číslo končiace číslom 5. To sa nevymazalo, keďže číslo 44 je kratšie a nedokázalo celé trojciferné číslo prekryť.

Riešenie vymazať celú obrazovku zmaže skutočne všetko a to sa nám nemusí vždy hodiť. Lepšie bude povedať programu, aby daný **výpis** robil tzv. **animovane.** Urobíme to tak, že za vypisovaný text dáme prvok **priehľadnosť**  a za ním číslo 2 alebo vyššie, napr. takto:

Animovaný výpis existuje na obrazovke len dovtedy, kým neprikážeme programu, aby na obrazovku priradil ďalší animovaný výpis s rovnakým číslom priehľadnosti. Predchádzajúci animovaný výpis sa okamžite zmaže.

Úloha 2

Vytvorme program, ktorý bude vypisovať na obrazovku príklady na sčítanie aj s ich výsledkom. Sčítance budú náhodné čísla od 0 do 50.

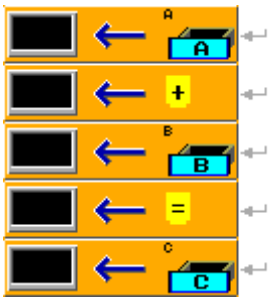
Riešenie

Vytvorme si 3 premenné A, B, C. Do prvých dvoch dáme náhodné čísla, do tretej súčet A a B.

Program teraz musí vypísať na obrazovku napr. 34 + 21 = 55. Znamienka + a = nemôžeme pri výpise na obrazovku použiť z prvkov  a  , pretože sú určené v programe na matematické operácie. Musíme použiť klasické žlté literály, do ktorých tieto znamienka napíšeme ako text. Napr. takto:

V tejto forme program nefunguje správne. Pri výpise na obrazovku miešame hrušky s jablkami. Program vie vypísať naraz niekoľko vecí za sebou, ale musia byť rovnakého dátového typu, napr. len reťazce alebo len čísla. To v tomto prípade nie je splnené.

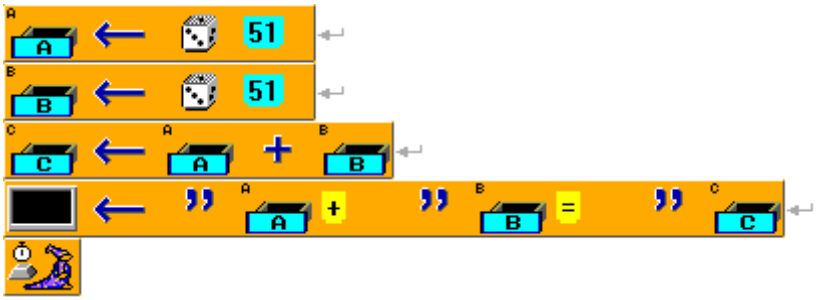
Zložité riešenie by mohlo byť takéto:



Jednoduchšie bude povedať programu, aby čísla z premenných A, B a C vnímal ako reťazce.

Dosiahneme to vložením príkazu **prevod na reťazec**  pred každé číslo.

Riešenie:



9 NIEKTORÉ MATEMATICKÉ OPERÁCIE

So zásuvkami (premennými) môžeme robiť rôzne operácie. Môžeme do nich priraďovať rôzne hodnoty, výsledky výpočtov, zmenšovať aj zväčšovať ich hodnotu. K tomu potrebujeme tri **priraďovacie operátory:**

 – operátor na priradenie novej hodnoty (doterajšia hodnota sa zabudne)

 – operátor na zväčšenie doterajšej hodnoty

 – operátor na zmenšenie doterajšej hodnoty

Niekoľko príkladov použitia:

 – hodnota premennej sa zväčší o 1

 – hodnota premennej sa zväčší o 4

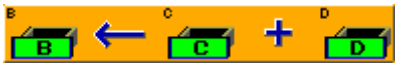
 – hodnota premennej sa zmenší o 1

 – hodnota premennej sa zmenší o 3, 4

 – hodnota premennej A sa zväčší o hodnotu premennej E

 – hodnota premennej sa zväčší o reťazec „kg“

 – do premennej C sa priradí rozdiel premenných B a A

 – do premennej B sa priradí súčet premenných C a D

 – do premennej D sa priradí súčin čísla 22 a hodnoty v premennej A

 – do premennej A sa priradí **celočíselný** podiel hodnôt premenných C a E – keďže premenná A je **celočíselná premenná,** nie je iná možnosť

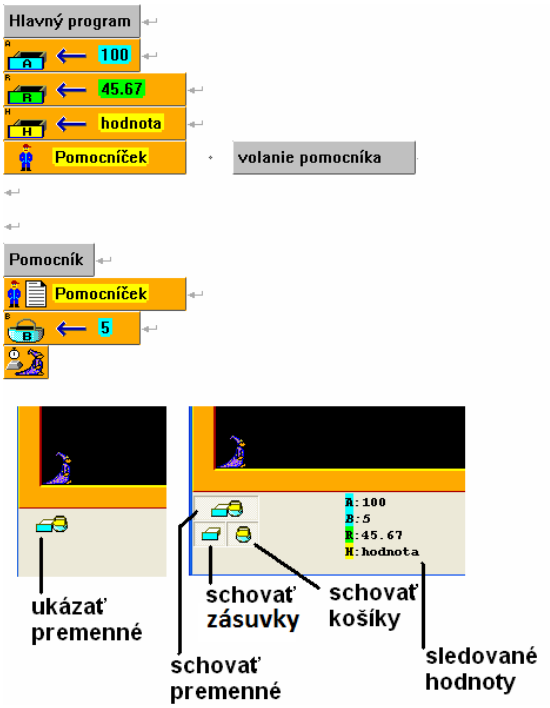
Príklad: Ak C = 7 a E = 3, do A sa priradí celočíselný výsledok podielu 7/3, t. j. 2

 – do premennej B sa priradí **reálny** podiel hodnôt premenných C a E – keďže premenná B je **reálna premenná**

Príklad: Ak C = 7 a E = 3, do B sa priradí reálny výsledok podielu 7/3 zodpovedajúci presnosti počítania programu, t. j. 2,33333333333333.

Sledovanie hodnôt premenných počas priebehu programu

Keď vytvoríme program, v ktorom používame premenné, máme možnosť počas jeho priebehu sledovať, ako sa menia ich hodnoty. Umožňuje nám to odhaliť rôzne chyby v programe.
Ak chceme sledovať počas priebehu programu hodnoty premenných a ich zmeny, klikneme **v spustenom programe** na ikonku v ľavom dolnom rohu okna v jeho sivej časti (pozri obrázok). V sivom obdĺžniku pod programom sa budú vypisovať všetky premenné a ich aktuálne hodnoty.



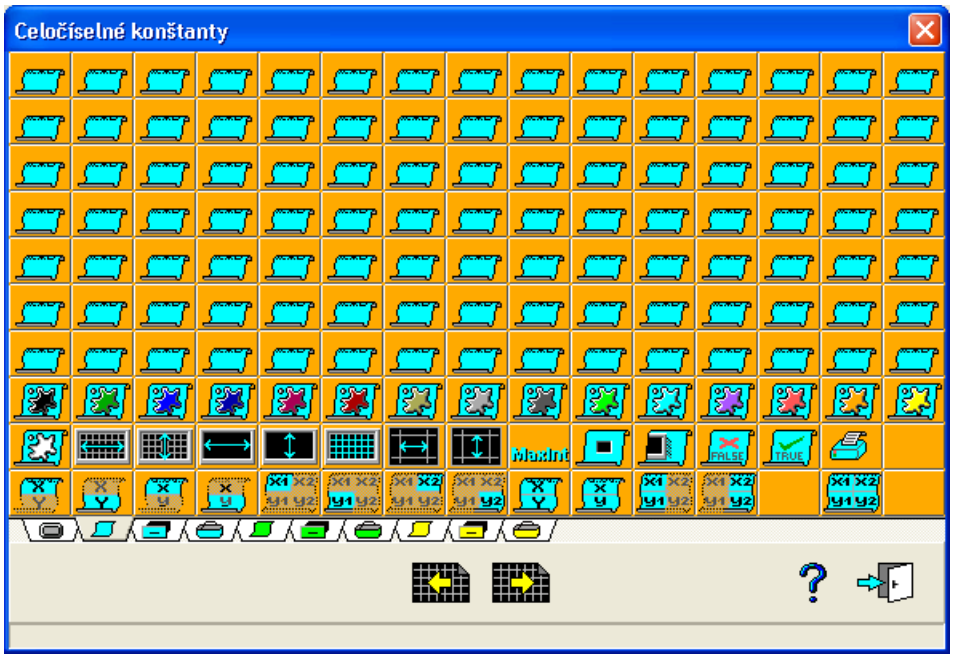
10 KONŠTANTY

V našich programoch bežne používame prvok literál. Vo veľkých programoch by sa literály mali používať čo najmenej a iba v prípade, že literál použijeme v danom kontexte iba raz. Ak sa niektorý literál s rovnakým zmyslom a hodnotou opakuje v programe viackrát, je potrebné uložiť si ho na jedno miesto. Na toto by nám mohla slúžiť premenná, ale v tomto

prípade by to nebolo vhodné. Premenná sa volá premennou preto, že môže svoju hodnotu v priebehu programu meniť. My sa zaoberáme **údajmi, ktoré majú byť v celom programe stále, čiže konštantné.**

Na tento účel máme špeciálny nástroj – **konštanty.**

Konštanty nájdeme pod tlačidlom  . Majú tvar skrúteného papiera (pozri obrázok).



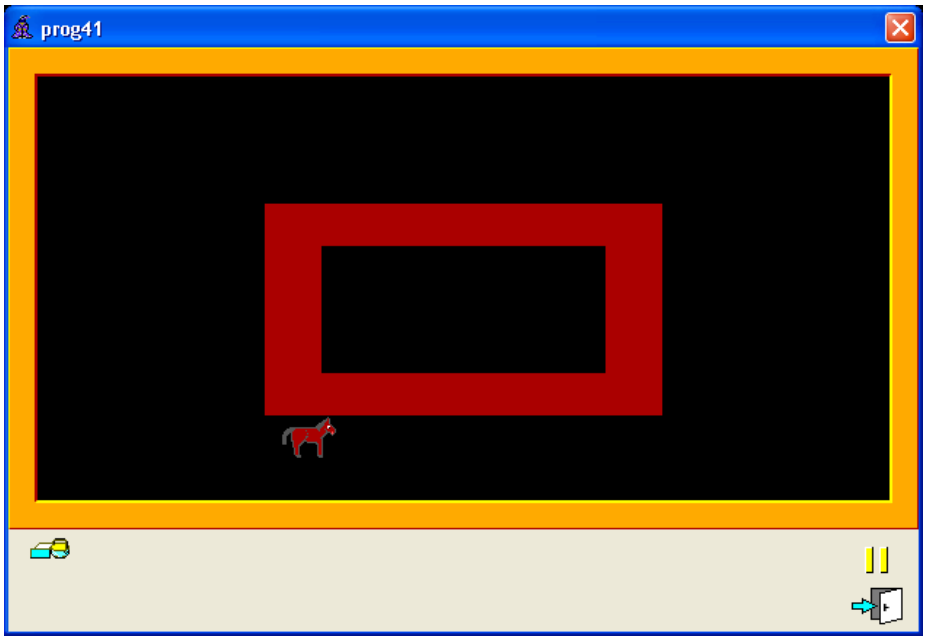
Konštanty – podobne ako premenné – môžu byť celočíselné (azúrovomodré), reálne (zelené) a reťazcové (žlté). Každý dátový typ konštánt má svoju vlastnú záložku so svojou farbou. V rámci jednej záložky sú konštanty tiež rozdelené:

- na poslednom riadku sa nachádzajú konštanty, ktoré využijeme najmä pri práci so súradnicami,
- na predposlednom riadku (u celočíselných konštánt sú to dokonca dva predposledné riadky) sa nachádzajú preddefinované, tzv. **systémové konštanty,**
- zvyšok tvoria **menné konštanty,** ktoré môžeme vytvoriť, pomenovať a vložiť do nich svoje vlastné hodnoty.

Konštanty vkladáme do programu v momente ich prvého použitia.

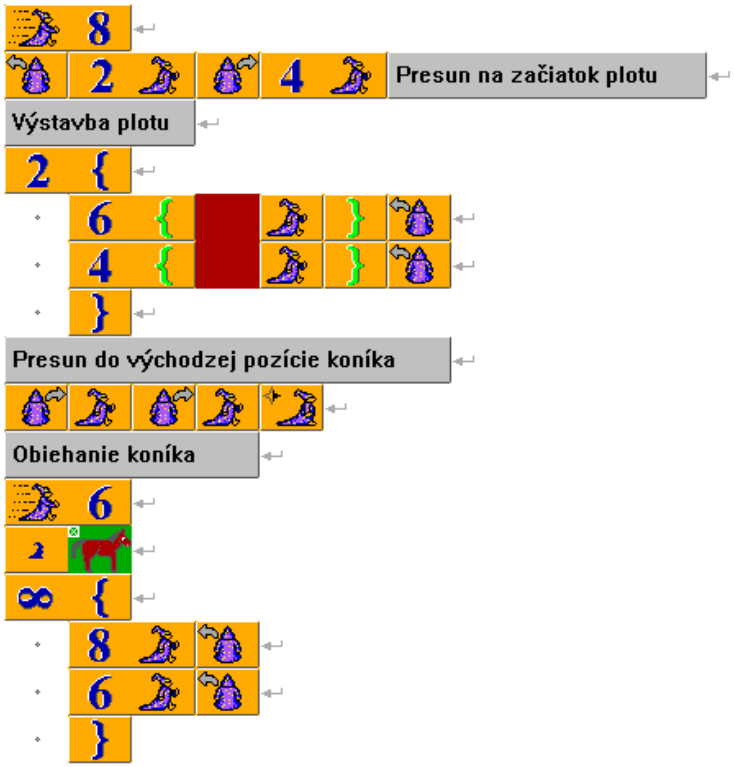
Úloha

Vytvorme program, v ktorom si Baltík najprv vytvorí v strede plot (pozri obrázok), okolo ktorého bude obiehať zmenený na koníka.

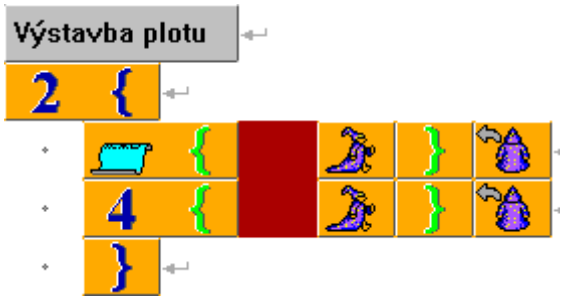


Riešenie

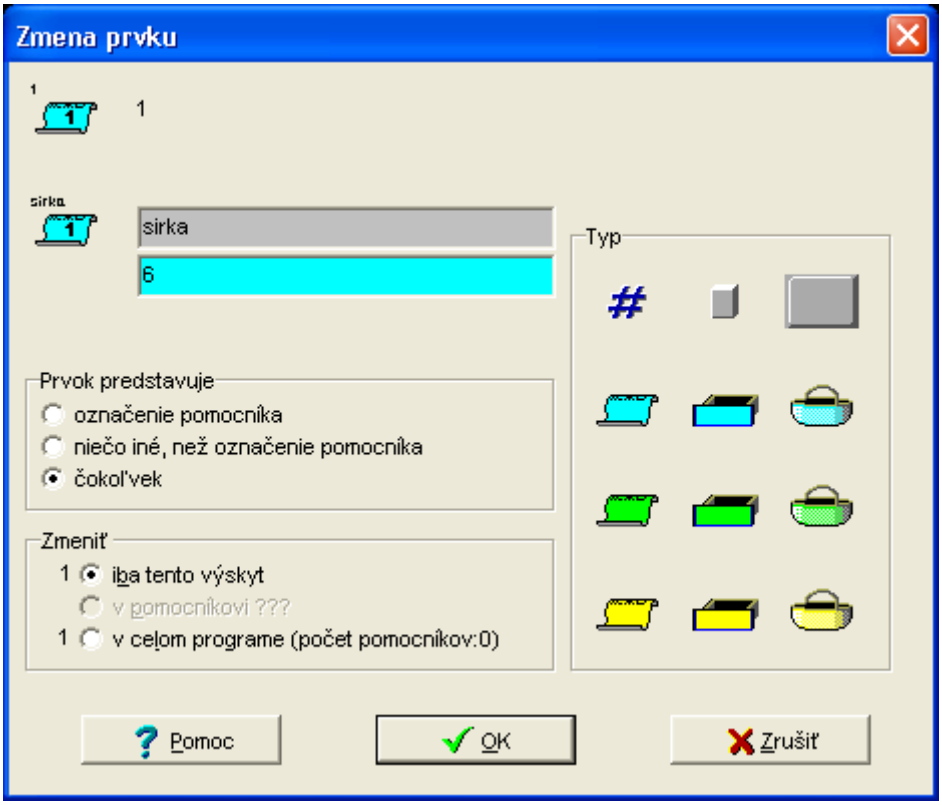
Doteraz by naše riešenie vyzeralo pravdepodobne takto:



Ak by sme sa rozhodli zmeniť rozmer plota, museli by sme prepočítavať a meniť aj počet krokov, koľko má koník prejsť po jeho jednotlivých stranách. Tých je vždy o dva viac, ako je šírka a výška plota. Ak na to zabudneme, nebude už program fungovať správne.
Upravme program tak, aby stačilo zmeniť šírku a výšku plota a počet krokov koníka sa automaticky prispôsobí.
Vytvorme si konštanty s názvami **šírka** a **výška.** Ak chceme vytvoriť konštantu **šírka,** ktorú použijeme v časti programu **Výstavba plota,** miesto čísla 6, vložíme z ponuky celočíselných konštánt jednu z menných konštánt priamo na príslušné miesto v programe.

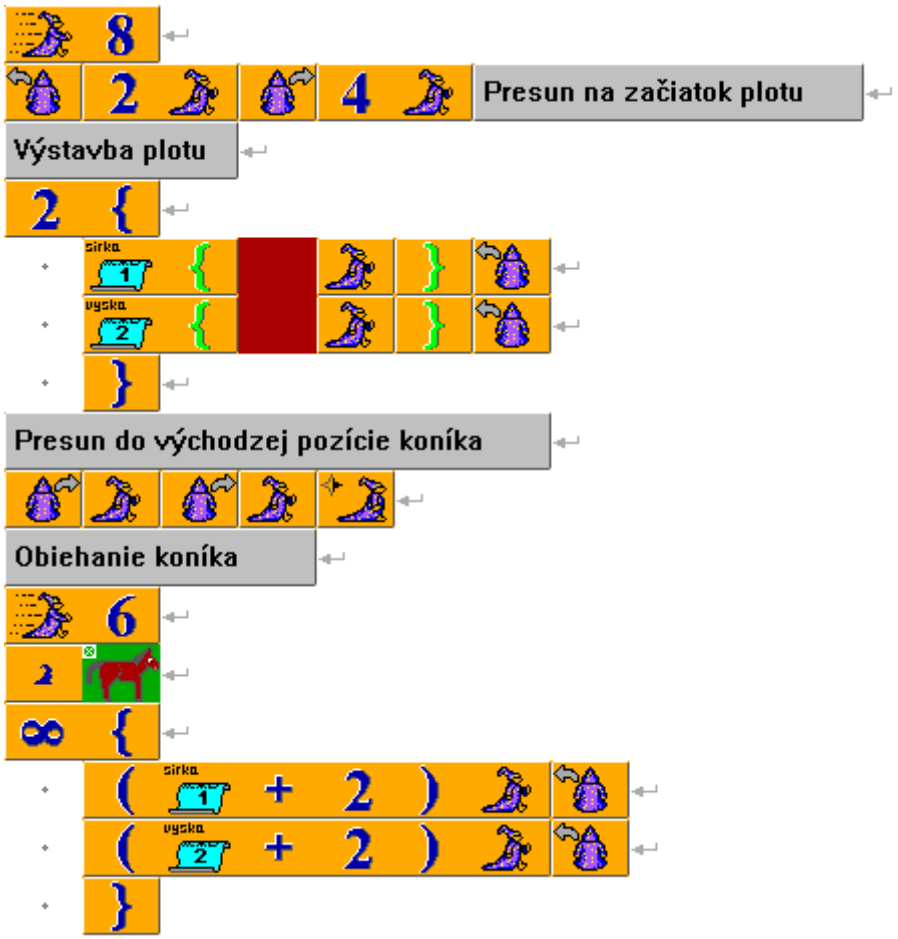


Otvorí sa dialógové okno, v ktorom podobne ako pri premenných zadávame do prvého okienka názov konštanty a v tomto prípade povinne do druhého okienka hodnotu (pozri obrázok).



To isté urobíme pre výšku. Potom už len odvodíme počet krokov Baltíka (koníka) od daných konštánt.

Riešenie:



Ak sa rozhodneme zmeniť šírku a výšku, stačí, keď sa posunieme kurzorom myši na príslušnú konštantu a stlačíme kláves F2 (alebo klikneme na ňu pravým tlačidlom myši a v lokálnom menu vyberieme voľbu *Upraviť*). Otvorí sa nám opäť to isté dialógové okno, v ktorom prepíšeme hodnotu konštanty.

V prípade **systémových konštánt** je použitie jednoduchšie. Tieto konštanty už svoju hodnotu majú priradenú a môžeme ich v programe využívať všade tam, kde je to potrebné. Hodnotu, ktorú skrývajú systémové konštanty, zistíme tak, že sa na ne presunieme kurzorom myši (či už v tabuľke konštánt, alebo priamo v programe). Ich názov aj hodnota sa ukážu v stavovom riadku.

Medzi systémovými konštantami máme množstvo hodnôt, ktoré by sme si pamätali dosť ťažko:

- **celočíselné konštanty –** šírka a výška Baltíkovej pracovnej plochy v pixeloch  , šírka a výška jedného políčka v pixeloch  , čísla farieb  ,
- **reálne konštanty –** π  , prevod radiánov na stupne a naopak  ,
- **reťazcové konštanty –** koniec riadka  , medzera  , prázdny reťazec  atď.

Ukážka využitia:

1. program

2. program

3. program

To isté môžeme vyskúšať aj s konštantou pre medzeru.

11 OBLASTI

Oblasť je obdĺžniková alebo štvorcová časť pracovnej plochy Baltíka, na ktorú sa dá v programe odvolávať a používať ju. Je presne určená pomocou bodových súradníc svojho ľavého horného a pravého dolného rohu.

Do oblasti môžeme vkladať obrázky, banky, predmety, text, geometrické tvary a môžeme dosiahnuť aj to, že sa tieto objekty veľkosťou prispôsobia veľkosti oblasti.

Ich ďalšie využitie je pri zadávaní podmienok – môžeme zistiť, či sa v oblasti niečo nachádza, napr. Baltík, predmet, ukazovateľ myši alebo či bolo v oblasti stlačené tlačidlo myši.

Úloha

Vráťme sa k ľubovoľnému programu z predchádzajúcich kapitol. Tento program budeme chcieť upraviť tak, aby sa na jeho začiatku objavil nápis ŠTART (pozri obrázok). Program sa spustí ďalej až vo chvíli, keď klikneme myšou kdekoľvek na tento nápis a jeho tesné okolie.

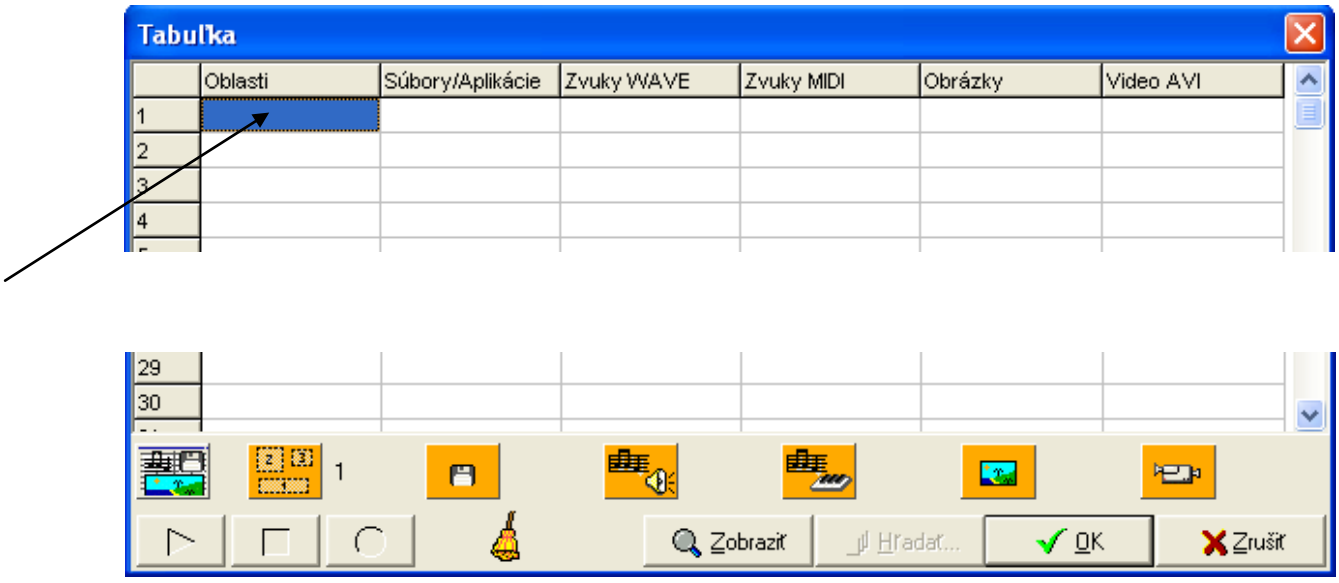
Riešenie

Pripravme si najprv scénu, ktorá bude obsahovať tento nápis, a uložme ju. Môžeme ju zároveň načítať na začiatok nášho programu a Baltíka urobiť neviditeľným, aby nám v nej „nezavadzal“.

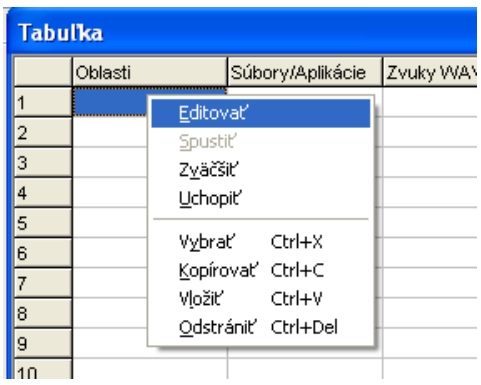
Ďalej potrebujeme vložiť podmienku, ktorá zabezpečí, že sa program nepohne ďalej, kým neklikneme niekde **do oblasti** nápisu.
Oblasť definujeme pomocou *Editora oblastí* alebo *Tabuľky oblastí*.

Vytvorenie oblasti pomocou *Editora oblastí*

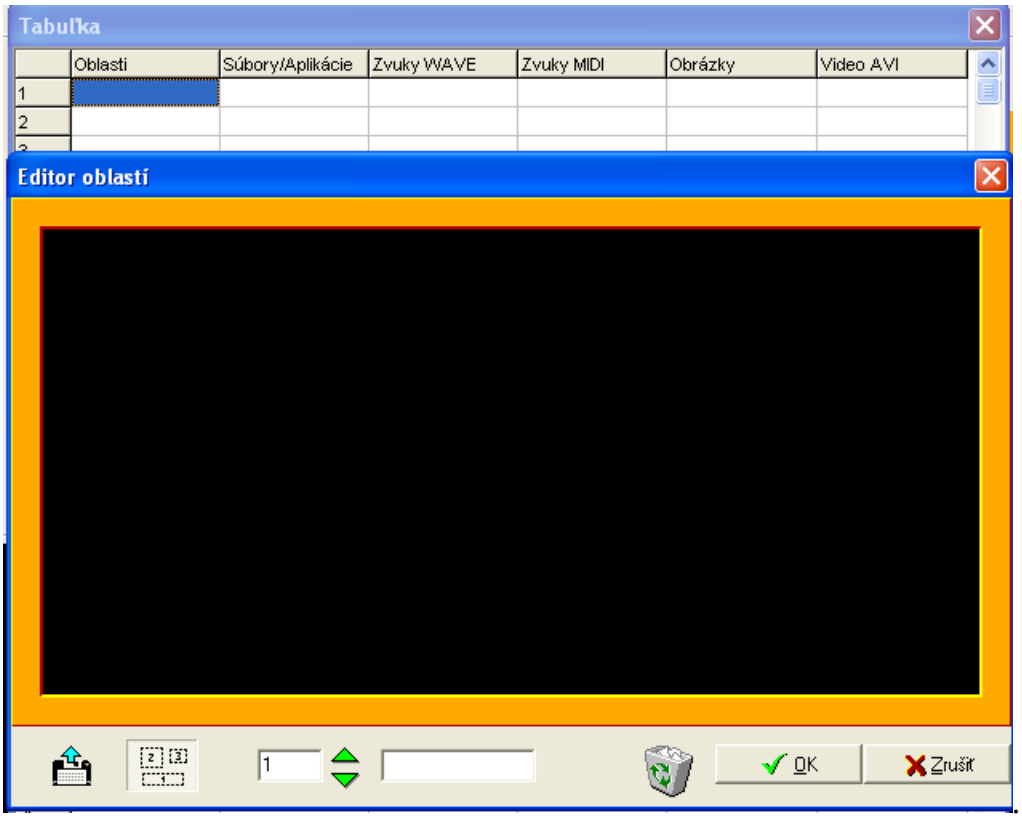
Stlačíme tlačidlo *Tabuľka*, nachádza sa nad panelom príkazov: 
Po stlačení sa objaví *Tabuľka súborov a oblastí*:



Táto tabuľka uľahčuje prácu so súbormi a s oblasťami. Každý stĺpec je vyhradený pre niečo iné – to, na čo je určený, poznáme z jeho hlavičky. Pre oblasti je vyhradený prvý stĺpec. Každá bunka daného stĺpca slúži na zadefinovanie jednej oblasti. Ak nemáme vyznačený prvý riadok v prvom stĺpci, klikneme myšou na danú bunku.
Ak chceme zadávať oblasť pomocou *Editora oblastí*, klikneme pravým tlačidlom myši do označenej bunky a z miestnej ponuky vyberieme možnosť *Editovať*.

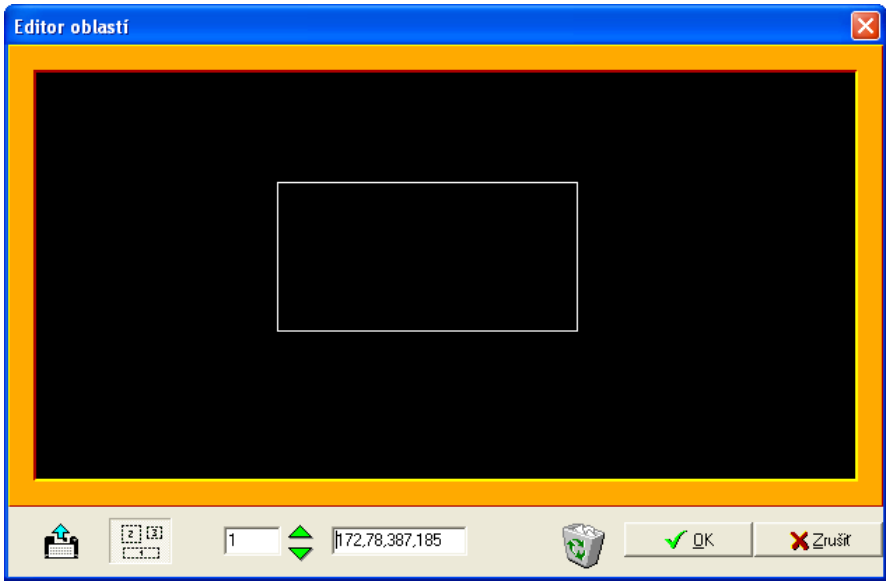


Objaví sa okno *Editora oblastí* (pozri obrázok). V tomto okne vyznačíme oblasť pracovnej plochy, ktorú si chceme zapamätať na ďalšie použitie.



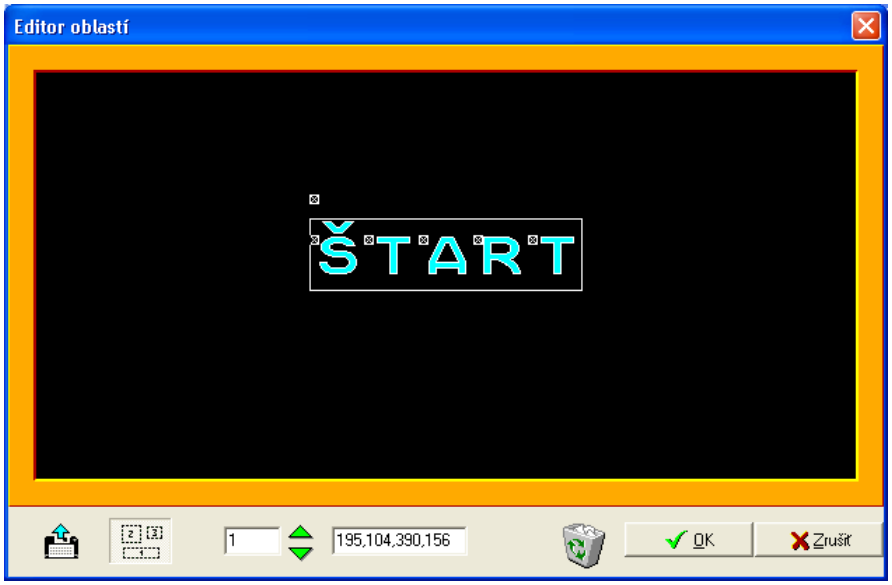
Pomocou myši metódou drž a ťahaj volíme umiestnenie a veľkosť oblasti. Okolo vyznačenej oblasti sa vytvorí biely obdĺžnik.

V dolnej časti vidíme dve okná. V prvom sa ukazuje číslo oblasti, ktorú tvoríme, a v druhom bodové súradnice, ktorými je ohraničená. Prvé dve sú súradnice ľavého horného rohu a druhé dve pravého dolného rohu.



Aby sme vedeli, kde je nápis ŠTART, musíme si scénu načítať do pozadia a podľa nej danú oblasť vyznačiť.

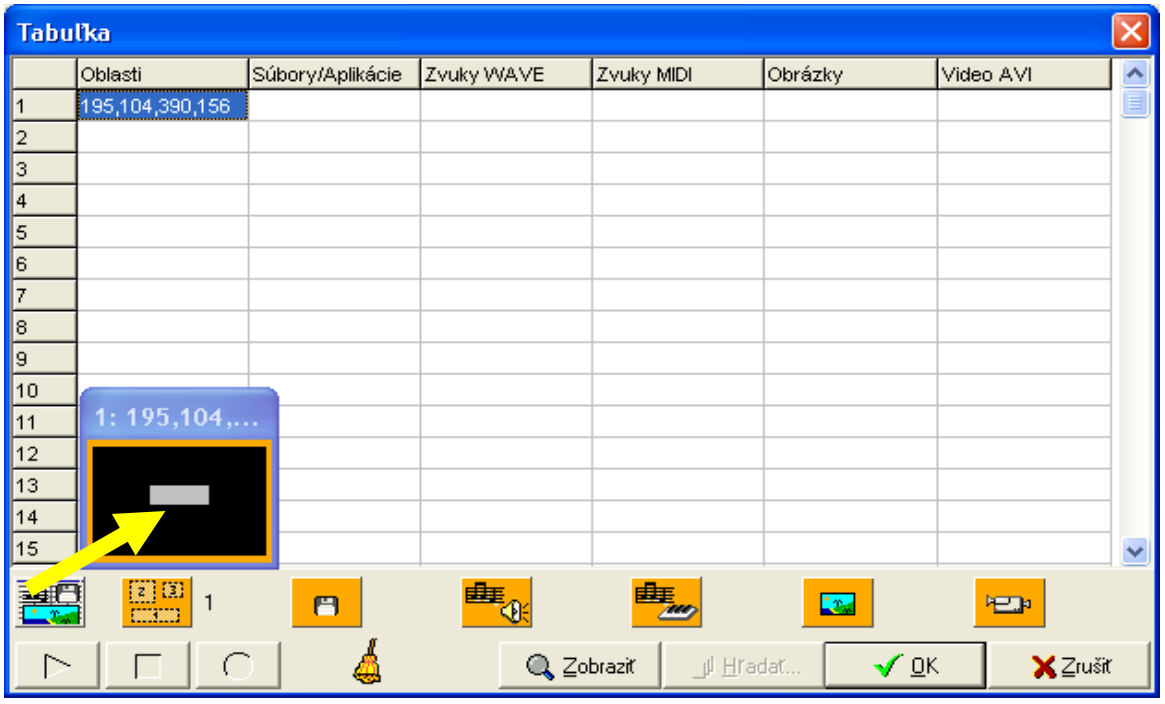
Úplne vľavo sa nachádza ikona diskety, ktorá slúži na načítanie banky, scény alebo ľubovoľného obrázka (formát musí byť BMP) do pozadia. Týmto spôsobom si načítame našu scénu. Teraz dokážeme vyznačiť oblasť tak, aby obsahovala celý nápis ŠTART (pozri obrázok).



Ak nie sme s oblasťou spokojní, vymažeme ju kliknutím na ikonku kôš.

Ak vyznačená oblasť vyhovuje potrebám programu, potvrdíme vytvorenie **aktívnej oblasti** tlačidlom OK.

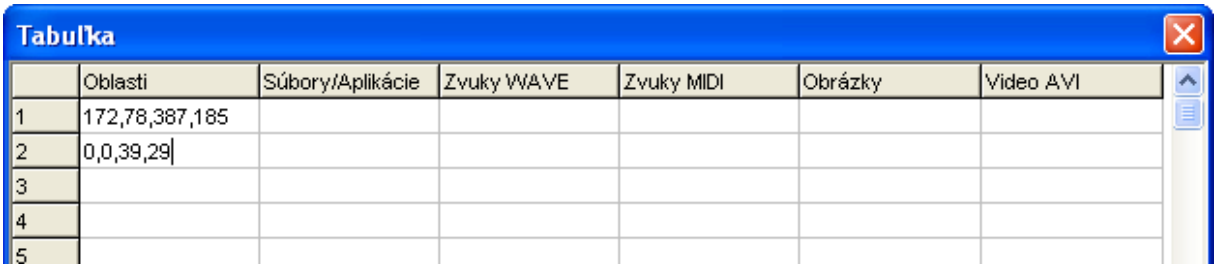
Po jeho stlačení sa opäť zobrazí okno *Tabuľka súborov a oblastí*.



Vo zvolenej bunke vidíme súradnice oblasti (v našom prípade ľavý horný roh 195, 104 a pravý dolný roh 390, 156). Žltou šípkou je označený náhľad na zvolenú oblasť č. 1 – vidíme zmenšenú pracovnú plochu (čierna) a zvolenú oblasť (sivá). Ak chceme mať väčší náhľad, klikneme do pracovnej plochy (čierna) ľavým tlačidlom myši. Opätovným stlačením vrátime späť menší náhľad.

Zadávanie oblastí priamo v *Tabuľke súborov a oblastí*

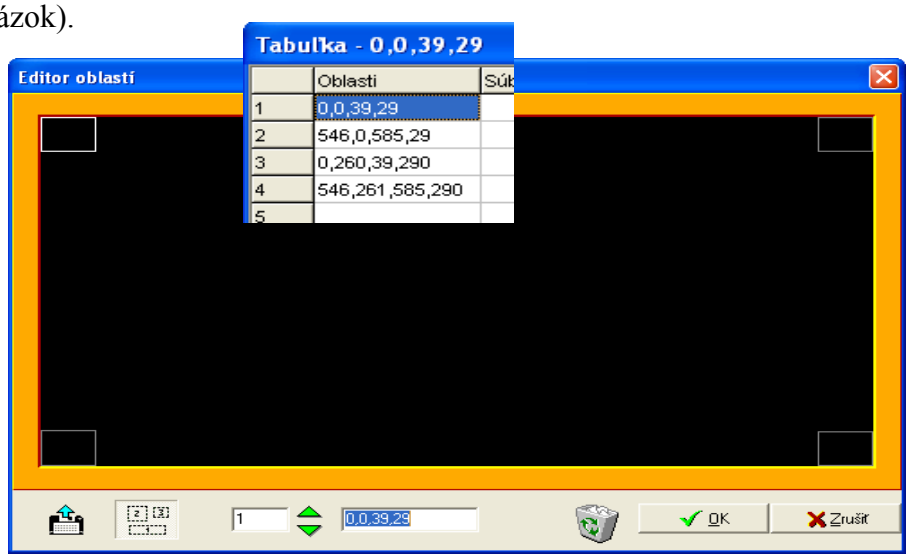
Tento spôsob používame v prípade, že súradnice vieme, resp. chceme vypočítať presne. Prepneme sa do *Tabuľky súborov a oblastí*, nastavíme sa do bunky, do ktorej chceme oblasť zadefinovať tak, aby zmenila farbu na modrú. Potom do nej klikneme ešte raz a dopíšeme súradnice, ktoré má daná oblasť mať. Súradnice sú oddelené čiarkou.



Editovanie vytvorenej oblasti

Ak sa rozhodneme, že už vytvorenú oblasť chceme akýmkoľvek spôsobom upraviť, buď sa prepneme opäť cez lokálnu ponuku do *Editora oblastí*, kde našu oblasť upravíme do želaného stavu, alebo jej súradnice prepíšeme priamo v tabuľke.

Poznámka! Ak máme v našom programe viac oblastí, budú v *Editore oblastí* viditeľné všetky naraz. Tá, s ktorou práve pracujeme, bude vždy vyznačená výraznejšie. Ak budeme chcieť upraviť inú, prepneme sa do nej zelenými šípkami umiestnenými pod pracovnou plochou editora (pozri obrázok).



Ako prenesieme oblasť do programu?

Musíme byť nastavení v tabuľke na bunke, v ktorej je daná oblasť. Následne uchopíme prvok **oblasť**  (v *Tabuľke súborov a oblastí* v dolnej časti pod stĺpcom vyhradeným pre oblasti) a vložíme ho do programu tam, kde potrebujeme. Jeho ikonka bude vyzerať nasledovne: 

Kým k tomuto prvku nepripojíme niečo, s čím dohromady bude dávať zmysluplný príkaz, bude program zobrazovať na tejto ikonke výkričník.

Ako ďalej pracovať s oblasťou v programe?

Oblasť môžeme využívať dvomi základnými spôsobmi:

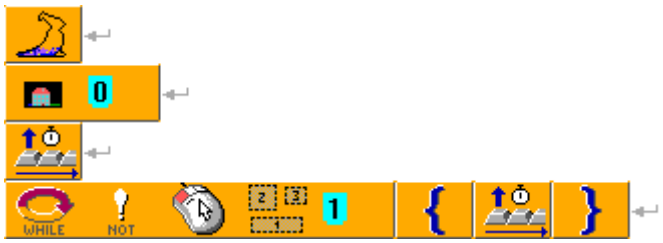
1. môžeme do nej niečo vložiť,
2. môžeme sa v podmienke pýtať, či sa v nej niečo nachádza, napr. kurzor myši pri stlačení ľavého tlačidla a pod.

Nás v tejto chvíli zaujíma druhá možnosť, pretože sa chceme pýtať, či sme neklikli do oblasti nášho nápisu ŠTART.

To, čo potrebujeme do programu vložiť, by mohlo byť formulované takto:

```
čítaj kláves s čakaním na stlačenie;
kým (nebude ľavé tlačidlo myši v oblasti 1) {čítaj kláves s čakaním na stlačenie}
```

Riešenie:



Z daného programu vidíme, že prvok oblasti umiestňujeme tam, kde sme predtým vkladali súradnice. Rozdiel je v tom, že pri zadaných súradniciach sme takto určili len jeden bod alebo políčko, kým týmto spôsobom môžeme naopak určovať akúkoľvek ohraničenú oblasť pracovnej plochy Baltíka.

12 PRÁCA S OBRÁZKAMI, VKLADANIE OBRÁZKOV DO PROGRAMU, RESP. DO OBLASTÍ

Čo sa týka grafiky našich programov, doteraz sme si museli vystačiť s predmetmi v bankách. Používali sme tie, ktoré sa v bankách nachádzali, alebo sme si vytvorili vlastné. Teraz si ukážeme, ako do programu vložiť externé obrázky.

Formát obrázka

Ak bude obrázok vytvorený v inom programe ako v Baltíkovom editore *Paint*, musí mať formát BMP. Iné formáty, žiaľ, Baltík nepodporuje.

Spôsoby, ako dostať obrázok do programu:

1. Prostredníctvom Baltíkovho editora *Paint*. Aj tu sú dve možnosti:
    a) obrázok si nakreslíme v čistej banke, pričom si ju nastavíme tak, aby nebola podelená na jednotlivé políčka, ale pokreslíme ju ako celistvý obrázok a uložíme medzi banky k danému programu,
    b) obrázok si nakreslíme v inom programe, uložíme ho ako BMP a do banky ho v editore *Paint* importujeme.

    Banku potom načítame ako celok do programu.
2. Prostredníctvom *Tabuľky súborov a oblastí*. Obrázok si nakreslíme napr. v Skicári, uložíme ho ako BMP a cez túto tabuľku ho zaradíme do zoznamu súborov pridaných k nášmu programu. Potom odkaz naň vložíme do nášho programu podobne, ako to robíme pri oblastiach.

1. a) Nakreslenie vlastnej banky

Otvorme si nový program s názvom **uloha2.bpr** a v ňom si vytvorme ľubovoľný obrázok v novej banke, ktorú uložme pod názvom **uloha2.b14** do toho istého priečinka ako náš program. Otvorme banky a presuňme sa na banku 14.

V dolnej časti sa nachádza oranžové tlačidlo s názvom banky. Stlačíme ho 

Okno s bankami sa zatvorí a na kurzore zostane visieť ikonka pre načítanie banky do programu, ktorú umiestnime do programu:

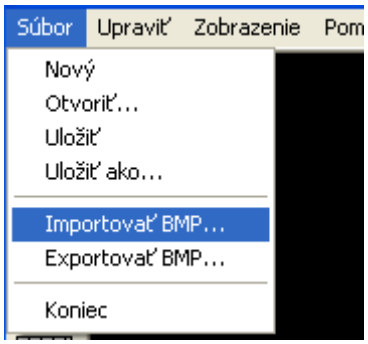
Po spustení programu sa celá banka načíta do programu.

Pozor! Medzi načítaním banky a načítaním scény je jeden rozdiel. Kým súbor scéna obsahuje len čísla predmetov s ich polohami na pracovnej ploche, banka sa načítava skutočne ako obrázok. Preto je rovnako ako iné obrázky náročnejšia na pamäť počítača a jeho rýchlosť.

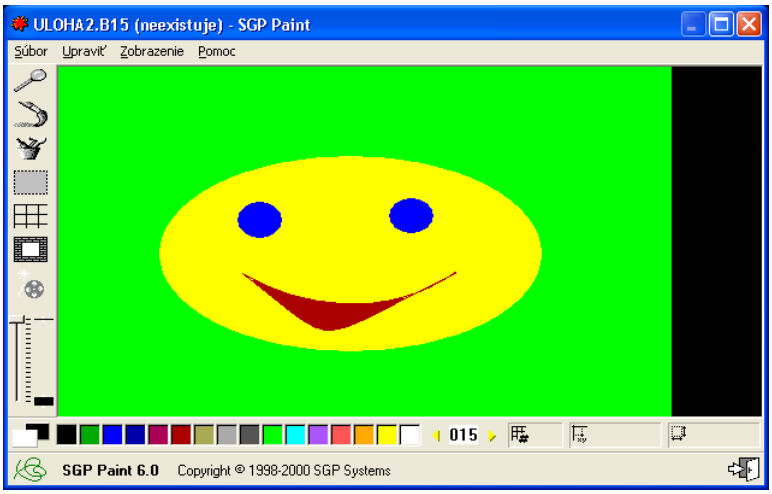
1. b) Importovanie obrázka do banky

Tento spôsob je podobný predchádzajúcemu, ale umožňuje nám vytvoriť si obrázok v inom programe. Podmienka je, že obrázok **musí byť vo formáte BMP.**
Samotný názov ani umiestnenie obrázka nie je rozhodujúce, pretože po importovaní do banky pôvodný súbor program využívať nebude.

Ak máme obrázok pripravený, prepneme sa v našom programe do editora *Paint* a nastavíme sa do prázdnej banky. V hlavnej ponuke vyberieme možnosť ***Súbor – Importovať BMP.***



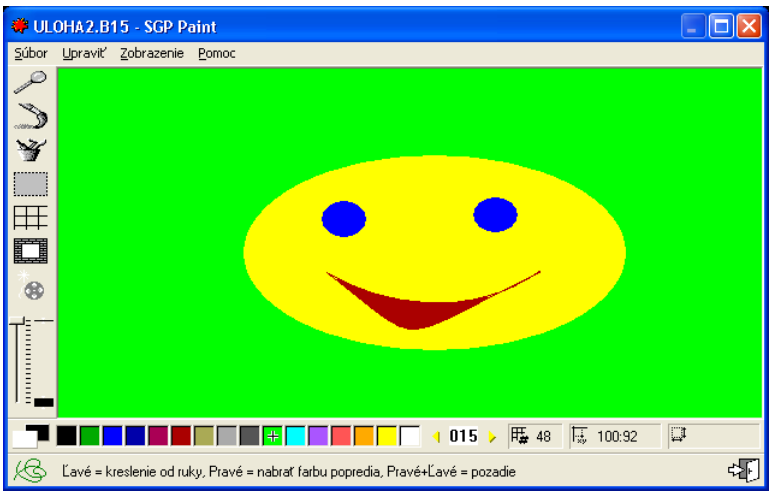
Otvorí sa okno pre výber súboru, v ktorom si nájdeme práve ten s naším obrázkom. Po potvrdení sa obrázok zobrazí v banke.



Problém je zrejmý na prvý pohľad. Ak náš obrázok nemal presne tie rozmery, ktoré má pracovná plocha Baltíka, t. j. 585 x 290 pixelov, obrázok „nesadne“ presne do banky. Ak je menší, ostane v nej kus prázdneho miesta, ak je väčší, usekne sa z neho.

Preto je potrebné nastaviť si ho pred importovaním na správny rozsah.

Obrázok potom vyplní celú plochu banky.



Druhý problém je so zachovaním farebnosti. Ak sme týmto spôsobom importovali fotku alebo obrázok s vyšším rozsahom farieb, budeme výsledkom sklamaní. Na takéto obrázky bude najvhodnejší posledný spôsob vkladania do programu.
Ďalší postup je totožný s postupom v bode **1.a).** Banku uložíme a prenesieme jej načítanie do programu rovnako ako v predchádzajúcom postupe.

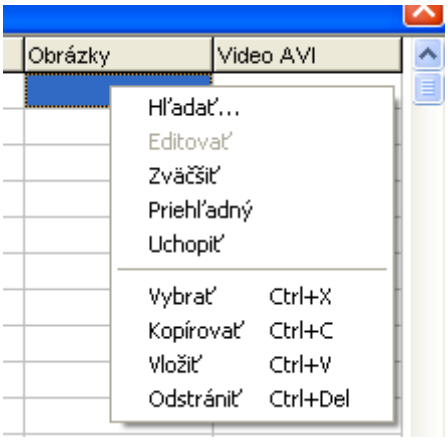
2. Vkladanie obrázkov zo súborov

Tento spôsob bude využívať pôvodné súbory obrázkov typu BMP. Ich názov aj umiestnenie budú dôležité, pretože ich budeme musieť prenášať všade s naším programom.
Na začiatok si vytvorme program s názvom **uloha3.bpr** a do toho istého priečinka si vytvorme aspoň dva obrázky. Pre ich názvy bude platiť podmienka, že sa **musia začínať rovnako, ako názov programu**, t. j. napr. **uloha31.bmp, uloha3a.bmp, uloha3smejko.bmp** a pod. Tak môžeme k nášmu programu vytvoriť ľubovoľný počet obrázkov s rôznymi názvami, len ich začiatok bude vždy zhodný.
Toto pravidlo nie je nevyhnutné dodržať pre samotné fungovanie načítania takéhoto súboru do programu, ale **v momente, keď ukladáme náš program ako BZIP, Baltík zbalí do tohto balíčka aj takéto súbory.**
Pripravíme si dva obrázky a pokračujeme s názvami **uloha3a.bmp, uloha3b.bmp** ďalej.

Pridanie obrázkov do zoznamu súborov programu

Najprv si potrebujeme vložiť naše obrázky do databázy súborov a oblastí nášho programu. Dostaneme sa do nej cez *Tabuľku súborov a oblastí*. V nej si budeme všímať predposledný stĺpec s hlavičkou **Obrázky** (pozri obrázok).

Nastavíme sa do prvej bunky v stĺpci tak, aby bola modrá, a pravým tlačidlom myši si otvoríme lokálnu ponuku. Z nej vyberieme možnosť *Hľadať*.



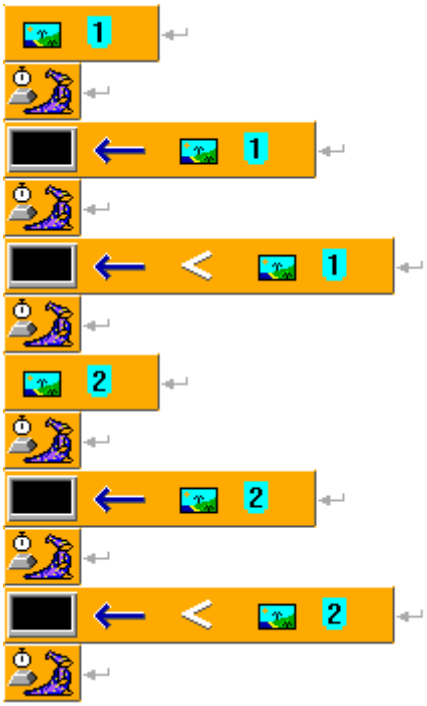
Otvorí sa okno na vyhľadanie súboru, v ktorom nájdeme náš obrázok **uloha3a.bmp,** náš výber potvrdíme tlačidlom *Otvoriť*. Názov súboru sa objaví v našej bunke (ak sa súbor nachádzal v inom priečinku ako náš program, tak sa tam objaví aj cesta k nemu).
Následne sa prepneme o bunku nižšie a urobíme to isté pre súbor **uloha3b.bmp.**
Výsledok by mal vyzerať takto:

Obrázok vložíme do programu rovnako, ako sme vkladali oblasť. Nastavíme sa na bunku obrázka, ktorý chceme vložiť do programu tak, aby bunka bola modrá. Pod stĺpcom sa nachádza tlačidlo pre načítanie obrázka do programu: 

.
Po jeho stlačení sa tabuľka zavrie a na kurzore ostane visieť ikonka, ktorá čaká na svoje umiestnenie v programe.

Vyskúšajme si niekoľko spôsobov, ako prikázať programu, aby náš obrázok vložil na pozadie.

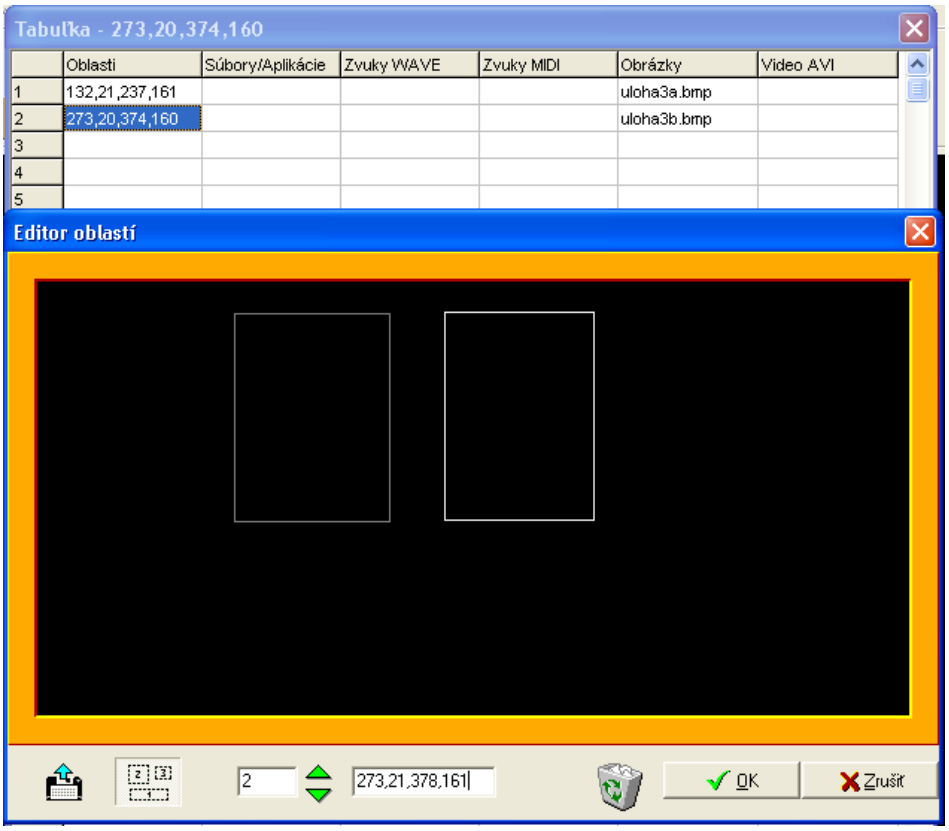
Program môžeme zapísať takto a otestovať:



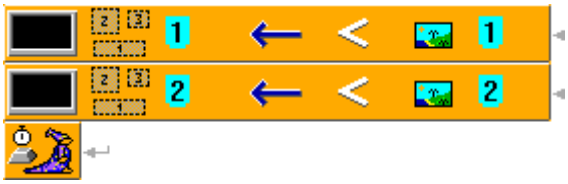
Ako vidíme, pri prvom aj druhom obrázku sú príkazy **zobraz obrázok** (1. a 7. riadok programu) a príkaz **na obrazovku priraď obrázok** (3. a 9. riadok programu) úplne rovnocenné. Zaujímavý je príkaz **na obrazovku priraď obrázok v prispôsobenej veľkosti** (5. a 11. riadok programu), ktorý spôsobí, že sa obrázok natiahne alebo naopak zmenší na rozmer obrazovky. Ak jeho pôvodný pomer strán bol iný, bude viac či menej zdeformovaný.

Vkladanie obrázka do oblasti

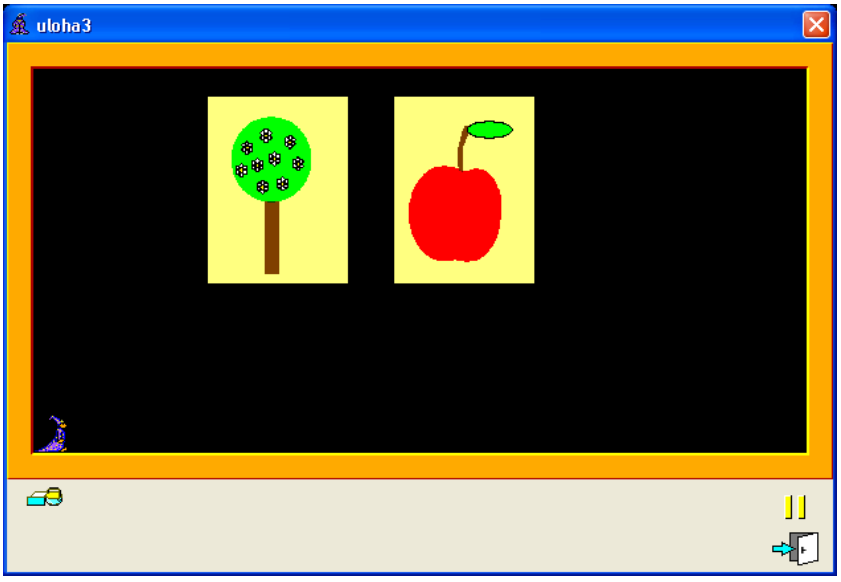
Teraz chceme obrázok vložiť na iné miesta pracovnej plochy, nielen do ľavého horného rohu. Nachystajme si v programe dve oblasti (ak sa nám bude chcieť trošku počítať, tak súradnice tej druhej si môžeme odvodiť od prvej) podľa obrázka:



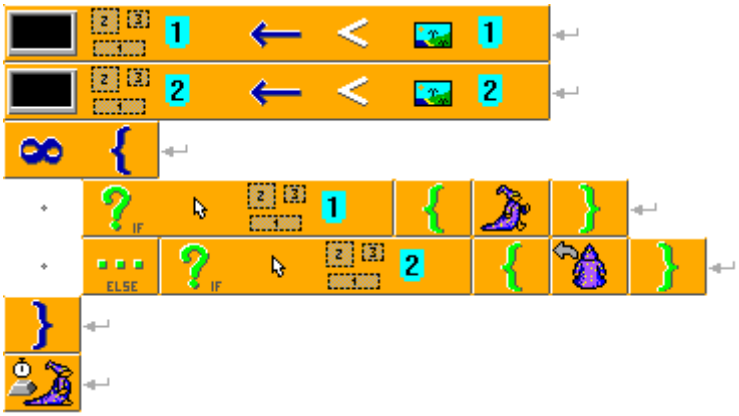
Následne vyskúšajme tieto príkazy:



Výsledok:



S programom sa dá rôzne pohrať:



Na záver si program uložíme ako **uloha3.bzip.** Ak sme naše súbory obrázkov správne umiestnili aj pomenovali, budú pribalené do balíčka.

13 ZVUKY

Aké typy zvukových súborov môžeme v Baltíkovi použiť?

Na prehrávanie zvukov môžeme použiť iba súbory s príponou .WAV.

Hotové súbory by mali byť rovnako ako obrázky uložené v tom istom priečinku ako program a začiatok ich názvu by mal byť totožný s názvom programu. Do tabuľky oblastí a súborov ich vkladáme rovnako ako obrázky, len použijeme stĺpec s hlavičkou **Zvuky WAVE.**

Zaujímavou vlastnosťou je to, že za názvom súboru môžu byť čiarkami oddelené dve čísla: čas začiatku prehrávania v milisekundách a čas konca prehrávania v milisekundách. Napr.: zápis v tabuľke **hudba.wav,120000,180000** zabezpečí, že zo súboru **hudba.wav** sa prehrá tretia minúta (t. j. od 120. do 180. sekundy) zaznamenanej nahrávky.

Zvuky v súboroch .wav môžeme nielen prehrávať, ale môžeme si cez mikrofón alebo zvukový vstup nahrať aj vlastné súbory.

Nahrávanie vlastných zvukových súborov

Postup je nasledovný:

1. Označme si prázdnu bunku v stĺpci **Zvuky WAVE –** bunka zmení farbu na modrú.
2. V dolnej časti tabuľky sa nachádza *ovládací panel zvukov*  . Kedykoľvek aktivujeme niektorú bunku z tohto stĺpca, ožije na ovládacom paneli

zvukov vedľa tlačidla *Prehrať súbor*  aj tlačidlo *Nahrať súbor* WAVE  . Medzi nimi sa nachádza zatiaľ neaktívne tlačidlo pre zastavenie prehrávaného súboru  .

3. Nahrávanie spustíme stlačením tlačidla *Nahrať súbor* WAVE.  . Po jeho stlačení začne počítač nahrávať.

   Počas nahrávania sa v dolnej časti tabuľky objaví mikrofón a beží čas:

4. Nahrávanie ukončíme stlačením tlačidla *Zastaviť prehrávanie alebo nahrávanie*  . Po jeho stlačení sa otvorí dialógové okno *Uložiť nový zvuk* WAVE. Zadáme názov súboru, ktorý sa potom zapíše do aktívnej bunky. Je dôležité, aby pred nahrávaním zvukov bol náš program už pomenovaný a určené jeho umiestnenie v počítači. V opačnom prípade sa bude do buniek zapisovať tzv. **absolútna adresa** zvukového súboru, t. j. jeho cesta až od disku, na ktorom sa nachádza. To nám môže spôsobiť problémy pri prenose na iný počítač. Naopak, ak máme program už dopredu pomenovaný a prípadne aj uložený a zvuk umiestnime do rovnakého priečinka, bude sa v bunke nachádzať len názov nášho súboru.

Zvuky MIDI

Do stĺpca s hlavičkou **Zvuky MIDI** zadávame názvy súborov s príponou .mid alebo .rmi. V Baltíkovi sa súbory typu MIDI nedajú nahrávať a tento stĺpček tabuľky využijeme rovnako ako stĺpček pre súbory WAVE v prípade, že máme k dispozícii nejaké hotové súbory daného typu. Použitie je rovnaké ako v predchádzajúcom prípade.

Video AVI

V Baltíkovi je možné použiť aj videá, ak sa končia príponou AVI. Spoľahlivosť ich spustenia nie je príliš veľká. Môže za to opäť „vek“ Baltíka a často je nutné do počítača doinštalovať ešte nejaké kodeky.

Prehrávanie multimediálnych súborov:

Multimediálne súbory prehráme stlačením tlačidla *Prehrať súbor*  .

Po spustení prehrávania ožije tlačidlo *Zastaviť prehrávanie alebo nahrávanie*  , jeho stlačením môžeme prehrávanie dlhého súboru predčasne zastaviť.

Postup na použitie multimédií v programe:

1. Pripravme si súbor s príponou .wav (môžu to byť aj súbory typu .mid, .rmi alebo .avi) so zvukom (hudbou alebo videoanimáciou).
2. Do tabuľky súborov vložíme názov pripraveného súboru. Postup je rovnaký ako pri vkladaní obrázka, t. j. nastavíme sa v tabuľke na bunku s daným súborom a pomocou oranžovej ikonky pod príslušným stĺpcom  prenesieme do programu príkaz na jeho spustenie.

Príklady:

Predchádzajúce príkazy vykonajú požadovanú akciu (prehraj) s obsahom súboru, ktorého názov je v tabuľke súborov v prvom riadku príslušného stĺpca.
Výnimkou je posledný príkaz na prehratie skladby z CD, ktorý sa na *Tabuľku* neodvoláva. Prvok **Prehraj skladbu z CD** je na príkazovom paneli medzi príkazmi pre animáciu a maskovacími komentármi. Jeho použitie je však takmer totožné s použitím ostatných príkazov na prehrávanie zvukov.

Doba prehrávania

Zvuk, hudbu alebo videoanimáciu ukončíme spustením iného súboru alebo zadaním čísla -1 za príslušný príkaz. Keďže v danom okamihu sa môže prehrávať iba jeden súbor daného typu, netreba dopĺňať číslo zastavovaného súboru.

Často je potrebné počkať na dokončenie prehrávania preto, že po ňom chceme spustiť ďalšiu skladbu, alebo preto, že s koncom programu sa zároveň končí prehrávanie všetkých skladieb a my chceme nechať prehrávanie pred koncom programu dôjsť do konca.
Ak má program počkať na dokončenie prehrávania, zadáme za prvok charakterizujúci typ prehrávaného súboru číslo -2.

Počkaj, kým dohrá zvuk WAVE.

Počkaj, kým dohrá hudba MIDI.

V ľubovoľnom okamihu môžeme prehrávanie zvuku WAVE, hudby MIDI alebo videa AVI pozastaviť, prípadne ju znova spustiť zadaním parametra -3.

Pozastav/spusti zvuk WAVE.

Pozastav/spusti hudbu MIDI.

Ukladanie programu

Rovnako ako v prípade použitia scén, vlastných bánk a obrázkov, aj v prípade použitia zvukových súborov je najideálnejšie program ukladať vo formáte **BZIP.** Ak sme dodržali pravidlo, že všetky zvukové súbory majú začiatok názvu totožný s názvom programu, pribalia sa do nášho programu BZIP.

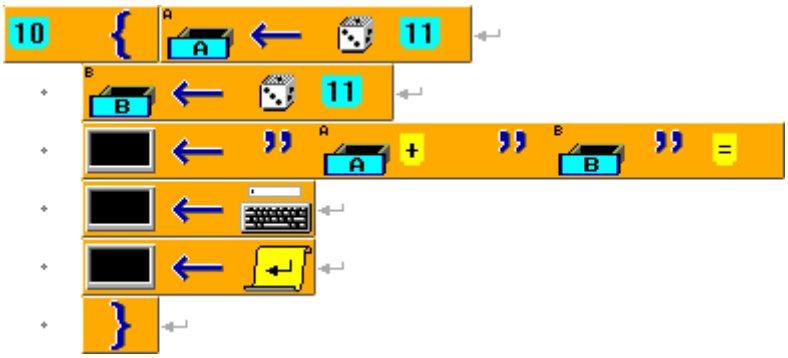
14 CYKLUS S PEVNÝM POČTOM OPAKOVANÍ

V programovaní používame dva druhy cyklov:

- cyklus s podmienkou na začiatku (prípadne na konci),
- cyklus s pevným počtom opakovaní.

V cykle s podmienkou nevieme dopredu, koľkokrát prebehne, závisí to od toho, ako dlho bude platiť podmienka.
V cykle s pevným počtom opakovaní je na jeho začiatku jednoznačne dané, koľkokrát prebehne.
V určitej forme sme cyklus s pevným počtom opakovaní už používali.

V tomto prípade Baltík 14-krát vyčaruje kvietok a posunie sa dopredu.

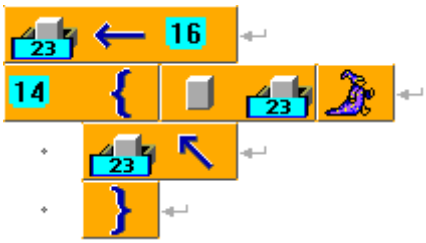


V tomto programe sa 10-krát vypíše na obrazovku príklad na sčítanie do desať, ku ktorému bude možné dopísať výsledok.

V oboch prípadoch jasne vidíme, že cyklus sa zopakuje v pevne určenom počte, v prvom prípade 14-krát, v druhom 10-krát, t. j. môžeme hovoriť o cykle s pevným počtom opakovaní. Oba prípady majú spoločnú ešte jednu vec – **v cykle sa počas jeho opakovania nič nemení v závislosti od toho, koľkýkrát cyklus prebieha**.

Čo v prípade, ak by sme chceli dosiahnuť, aby Baltík nečaroval 14-krát kvietok, ale postupne predmety číslo 16 – 29?

S našimi doterajšími vedomosťami by naše riešenie vyzeralo asi takto:



V iných programovacích jazykoch takáto forma zápisu cyklu s pevným počtom opakovaní neexistuje a ani v Baltíkovi nebudeme toto riešenie uprednostňovať. Zhrnutie:

- tam, kde sa cyklus s pevným počtom opakovaní nemení v závislosti od toho, koľkýkrát prebieha, budeme používať v Baltíkovi doterajšiu formu zápisu,
- tam, kde sa cyklus s pevným počtom opakovaní mení v závislosti od toho, koľkýkrát prebieha, budeme používať novú formu zápisu, ktorú si teraz ukážeme.

Cyklus for

Cyklus **for** („for“ po slovensky znamená „pre“) určuje svoj počet opakovaní pomocou premennej – hovoríme jej **riadiaca premenná.** V hlavičke cyklu vždy uvedieme názov premennej, jej počiatočnú a koncovú hodnotu a o koľko má pri každom ďalšom zopakovaní cyklu zmeniť svoju hodnotu. Potom sa môžeme na túto premennú v cykle odvolávať.

Na vytvorenie hlavičky cyklu **for** budeme potrebovať nasledujúce prvky:

 – prvok pre kľúčové slovo **for,**

 – logické zátvorky,

 – čiarka oddeľujúca jednotlivé parametre v hlavičke.

Pozrime sa, ako bude teraz vyzerať celý cyklus. Ako príklad nám poslúži práve program, v ktorom má Baltík vyčarovať v dolnej časti plochy predmety 16 – 29:

Prvý riadok tvorí hlavička cyklu, v druhom je vyjadrené, čo sa má v cykle a v akom počte urobiť.

Hlavička cyklu musí vždy ako prvé obsahovať kľúčové slovo **for.** V zátvorke za ním nasledujú vždy 4 parametre:

1. riadiaca premenná, v našom prípade je jej názov **Predmet,**
2. jej počiatočná hodnota (t. j. hodnota, ktorú bude mať, keď cyklus prebehne prvýkrát) – v našom prípade je to **16,**
3. jej koncová hodnota (t. j. hodnota, ktorú bude mať, keď cyklus prebehne naposledy) – v našom prípade je to **29,**
4. krok (t. j. hodnota, o ktorú sa má premenná zmeniť pri každom zopakovaní cyklu) – v našom prípade sa bude **zväčšovať o 1.**

Všimnime si, že v nasledujúcom bloku príkazov sa riadiaca premenná hneď využíva. Pri každom opakovaní cyklu sa vyčaruje predmet s takým číslom, aká je práve hodnota riadiacej premennej, takže cyklus bude prebiehať asi takto:

1-krát (**Premenná = 16**): vyčaruje sa predmet č. 16; Baltík sa posunie

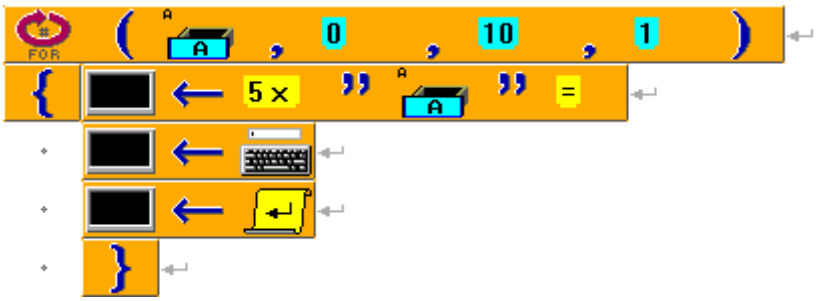
2-krát (**Premenná = 17**): vyčaruje sa predmet č. 17; Baltík sa posunie

3-krát (**Premenná = 18**): vyčaruje sa predmet č. 18; Baltík sa posunie

....

14-krát (**Premenná = 29**): vyčaruje sa predmet č. 29; Baltík sa posunie

Poďme sa pozrieť na druhý program, ktorý sme si na začiatku ukázali. Skúsme ho zmeniť tak, aby sa v danom cykle vypísali príklady na násobenie čísla 5 číslami od 0 po 10.



Riadiaca premenná je v tomto prípade **A** a cyklus bude prebiehať takto:

1. cyklus (**A = 0**): vypíše sa príklad 5 x 0 =; program si vyžiada výsledok
2. cyklus (**A = 1**): vypíše sa príklad 5 x 1 =; program si vyžiada výsledok
3. cyklus (**A = 2**): vypíše sa príklad 5 x 2 =; program si vyžiada výsledok

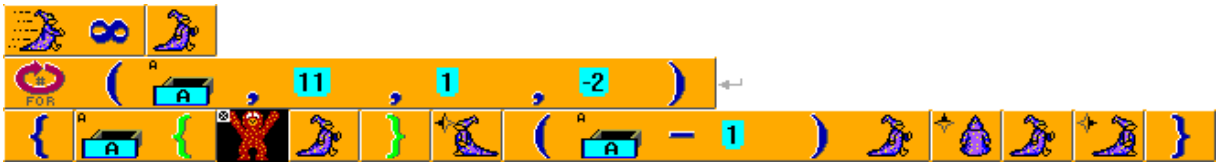
   ....

11. cyklus (**A = 10**): vypíše sa príklad 5 x 10 =; program si vyžiada výsledok

Riadiaca premenná môže v cykle aj klesať. Príklad programu, v ktorom by Baltík postavil pyramídu z opíc:

- riešenie bez cyklu **for:**

- riešenie pomocou cyklu **for:**



Prvýkrát bude mať riadiaca premenná hodnotu 11, čo spôsobí, že sa v bloku príkazov urobí to isté, čo v prvom riadku opíc v predošlom riešení.

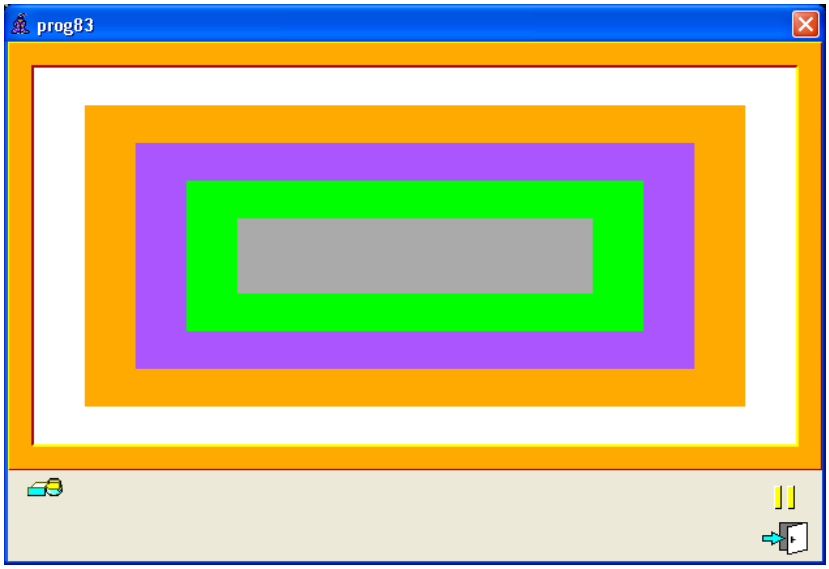
Druhýkrát bude mať riadiaca premenná hodnotu 9, to zodpovedá druhému riadku opíc atď.

Počiatočná hodnota riadiacej premennej je vyššia ako koncová, preto sa jej hodnota musí meniť o záporné číslo.

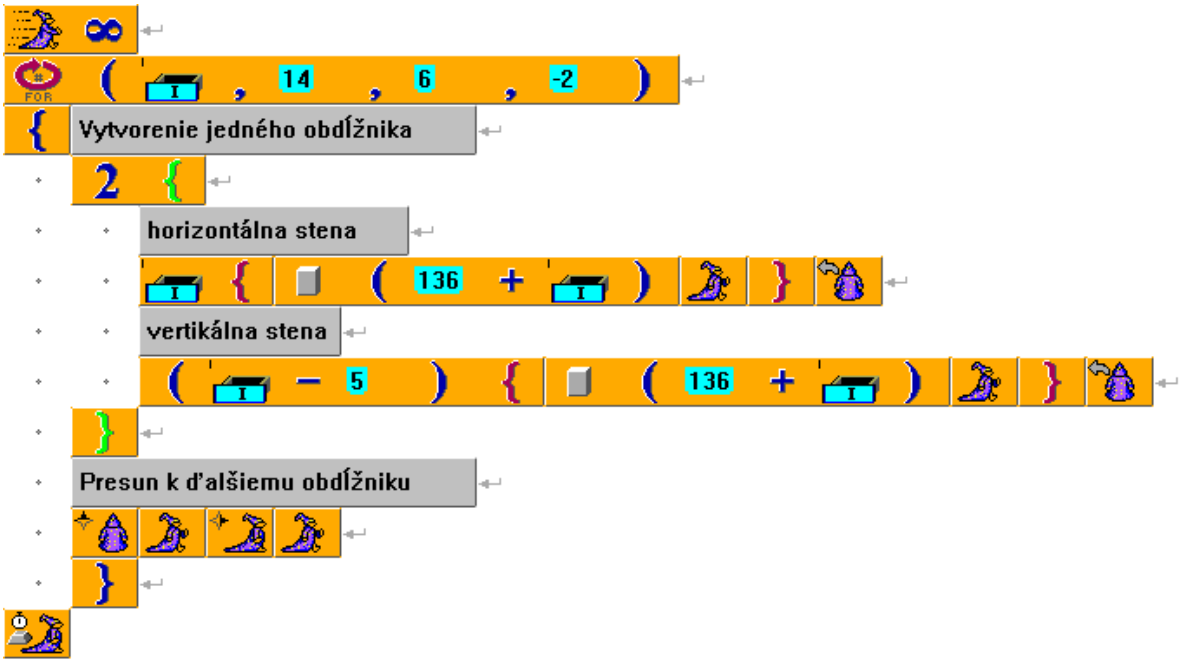
Ďalšie príklady využitia cyklu **for:**

Príklad č. 1

Program, v ktorom sa vytvoria do seba vložené obdĺžniky niekoľkých farieb (pozri obrázok):

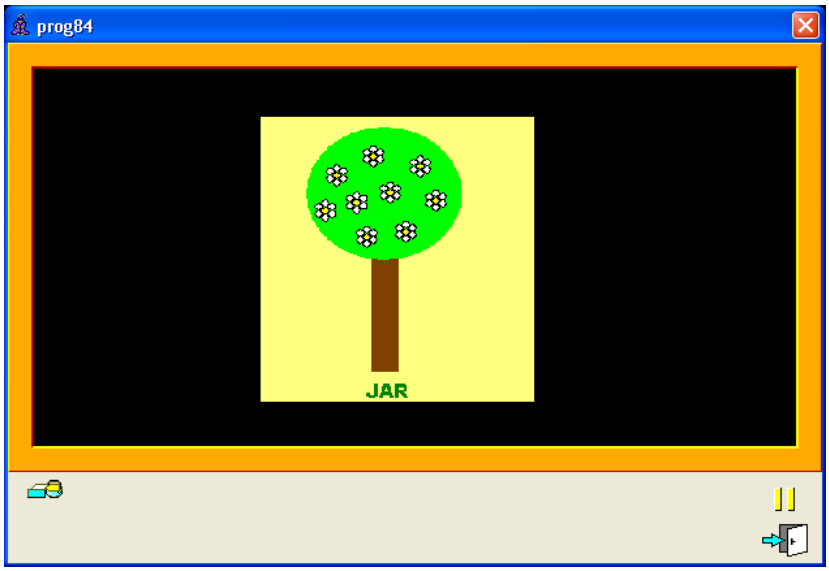


Možné riešenie:

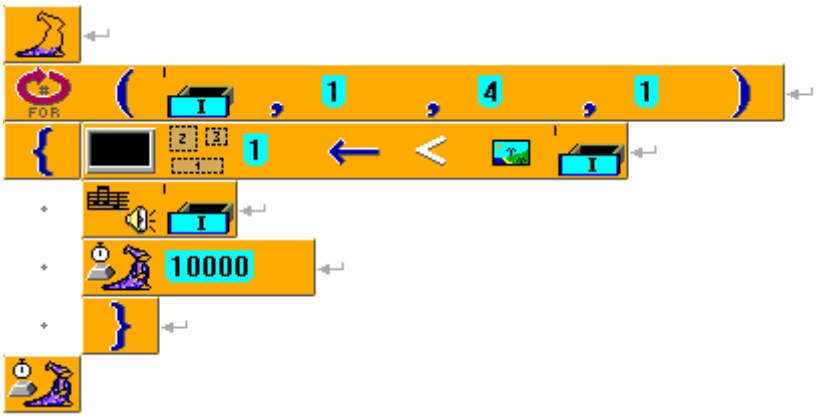


Príklad č. 2

Program, v ktorom sa postupne na 10 sekúnd zobrazia obrázky symbolizujúce jar, leto, jeseň a zimu. Ku každému ročnému obdobiu zároveň zaznie aj iná skladba.



Možné riešenie:



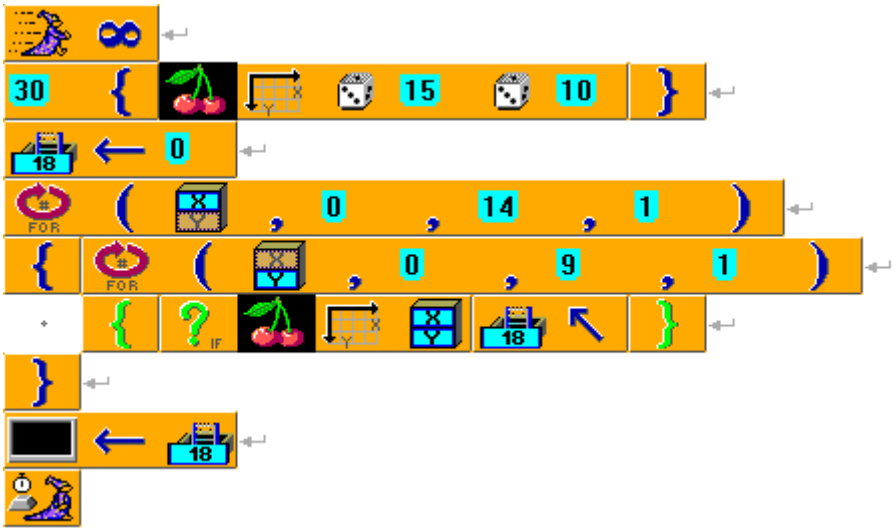
Podobne by sme mohli za sebou načítavať aj scény, oblasti a pod.

Cyklus **for** môže v sebe obsahovať ďalší cyklus **for** a niekedy je takéto riešenie veľmi efektívne.

Príkladom je program, v ktorom sa vyčaruje napr. 30-krát náhodne na Baltíkovu plochu čerešnička. V skutočnosti sa pravdepodobne niektoré čerešničky prekryli (zobrazili na tie isté súradnice), po skončení nemôžeme presne vedieť, koľko sa ich na ploche nachádza. Ak by sme chceli vytvoriť program, v ktorom bude Baltík pomocou šípok chodiť a zbierať

čerešničky a tento program by mal skončiť vo chvíli, keď ich všetky pozbiera, musíme najprv zistiť, koľko čerešní tam skutočne máme.

Riešenie by mohlo vyzerať takto:



Riadiace premenné sú v tomto prípade **X** a **Y** a cyklus bude prebiehať takto:

1. **X = 0:**
    1.1 **Y = 0:** ak je na súradniciach **0, 0** čerešňa, **Počítadlo** zväčší o 1
    1.2 **Y = 1:** ak je na súradniciach **0, 1** čerešňa, **Počítadlo** zväčší o 1
    1.3 **Y = 2:** ak je na súradniciach **0, 2** čerešňa, **Počítadlo** zväčší o 1
    ....
    1.10 **Y = 9:** ak je na súradniciach **0, 9** čerešňa, **Počítadlo** zväčší o 1
2. **X = 1:**
    2.1 **Y = 0:** ak je na súradniciach **1, 0** čerešňa, **Počítadlo** zväčší o 1
    2.2 **Y = 1:** ak je na súradniciach **1, 1** čerešňa, **Počítadlo** zväčší o 1
    2.3 **Y = 2:** ak je na súradniciach **1, 2** čerešňa, **Počítadlo** zväčší o 1
    ....
    2.10 **Y = 9:** ak je na súradniciach **1, 9** čerešňa, **Počítadlo** zväčší o 1

....

15. **X = 14**
    15.1 **Y = 0:** ak je na súradniciach **14, 0** čerešňa, **Počítadlo** zväčší o 1
    15.2 **Y = 1:** ak je na súradniciach **14, 1** čerešňa, **Počítadlo** zväčší o 1
    15.3 **Y = 2:** ak je na súradniciach **14, 2** čerešňa, **Počítadlo** zväčší o 1
    ....
    15.10 **Y = 9:** ak je na súradniciach **14, 9** čerešňa, **Počítadlo** zväčší o 1

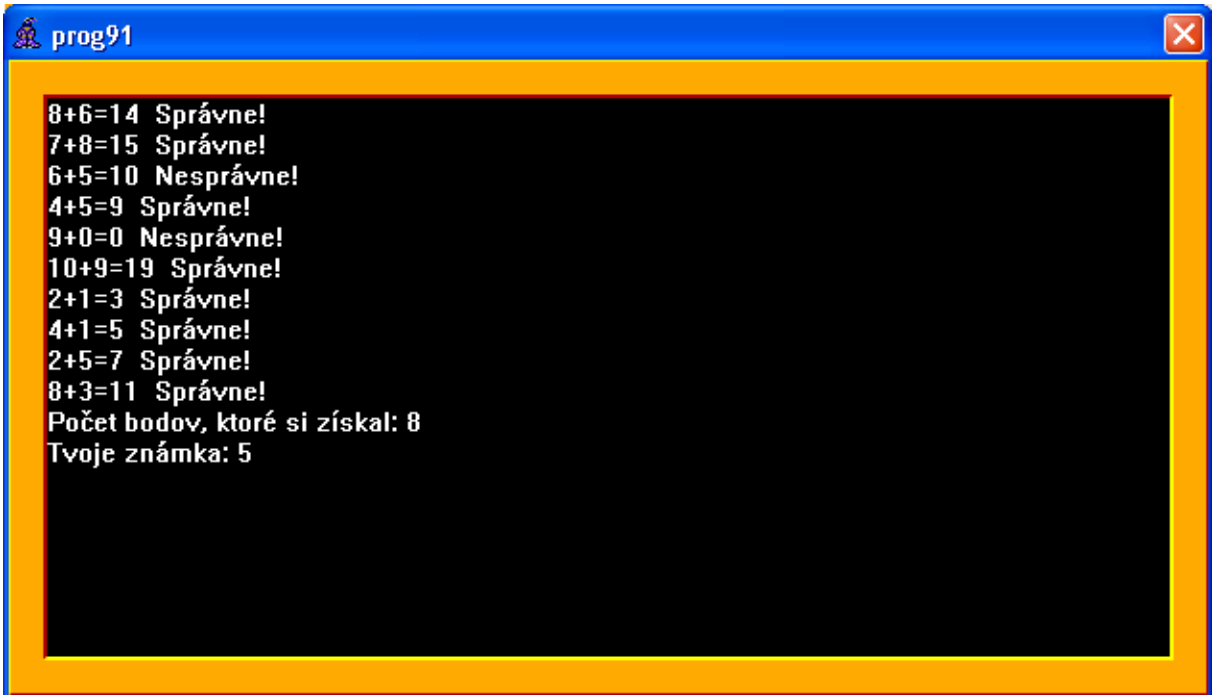
Vnorenie týchto dvoch cyklov spôsobuje, že sa postupne prezrú všetky stĺpce pracovnej plochy. Najprv stĺpec s *X*-ovou súradnicou 0, v ktorom sú prejdené všetky políčka postupne podľa ich *Y*-ových súradníc, potom stĺpec s *X*-ovou súradnicou 1 atď. až po posledný stĺpec.

15 PREMENNÉ V PODMIENKACH

Úloha

Vytvorme program – test z matematiky. V programe sa postupne objaví 10 náhodných príkladov na sčítanie do 20, ktoré má užívateľ úspešne vyriešiť. Aby sme to nemali príliš ťažké, nebudeme uvažovať nad príkladmi, kde jeden zo sčítancov je väčší ako 10, napr. 13 + 5 =, ale oba sčítance budú mať maximálnu hodnotu 10.

Každý príklad program hneď vyhodnotí a napíše, či bol správny alebo nesprávny. Po 10 príkladoch program vypíše, koľko príkladov sme vypočítali správne, prípadne nás ohodnotí aj známkou.

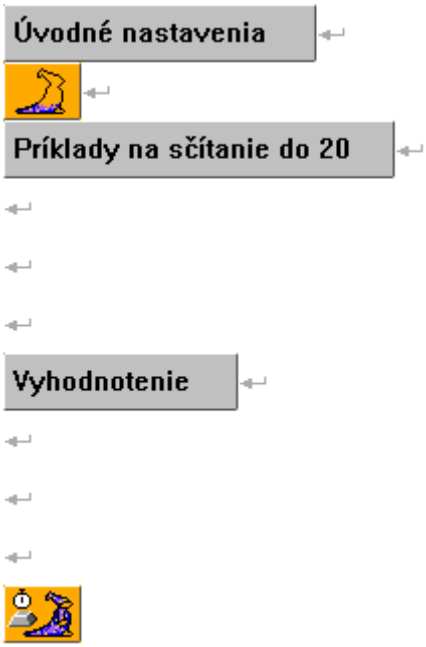


Riešenie

Na začiatku programu urobíme úvodné nastavenia – Baltíka urobíme neviditeľným, keďže v tomto programe nie je potrebný.

Hlavná časť programu bude obsahovať samotné zadávanie príkladov a ich výpočet.

Na konci programu bude jeho vyhodnotenie.

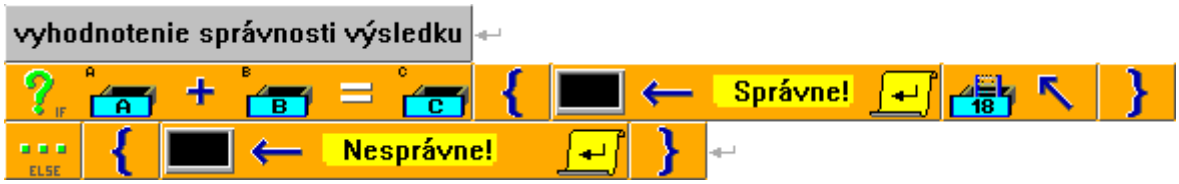


Čo musí obsahovať časť *Príklady na sčítanie do 20*:

1. Vygenerujú sa dva náhodné sčítance, ktoré si uložíme do dvoch premenných s názvom **A** a **B.**
2. Vypíšu sa na obrazovku v tvare A + B =.

3. Program si vyžiada od užívateľa výsledok – vstup z klávesnice, ktorý si následne uloží do ďalšej premennej s názvom **C**.
4. Jej hodnotu vypíše na obrazovku ako výsledok príkladu, t. j. na obrazovke sa zobrazí tvar A + B = C.

5. Program vyhodnotí, či bol výsledok správny. Ak sa A + B = C, za príklad napíše „Správne!“ a do premennej, nazvime ju napr. **Počítadlo,** priráta 1 bod. V opačnom prípade (inak) len vypíše za príklad „Nesprávne!“



Pred text „Správne!“ aj „Nesprávne!“ sme vložili niekoľko medzier, aby nebol prilepený rovno na príklade. Za text sme doplnili ešte konštantu pre presun kurzora na ďalší riadok, aby sa ďalší príklad začínal v novom riadku.

Celý tento proces sa bude 10-krát opakovať.

A teraz sa pozrime na časť *Vyhodnotenie:*

1. Na obrazovku sa vypíše, koľko bodov sme získali, t. j. hodnota premennej **Počítadlo** spolu s príslušným komentárom.

2. Ak chceme, aby program vypísal aj známku, musíme ju odvodiť z hodnoty v premennej **Počítadlo**, napr. takto:
   ak **Počítadlo**
   = 10 – 9 {„Známka: 1“}

= 8 – 7 {„Známka: 2“}

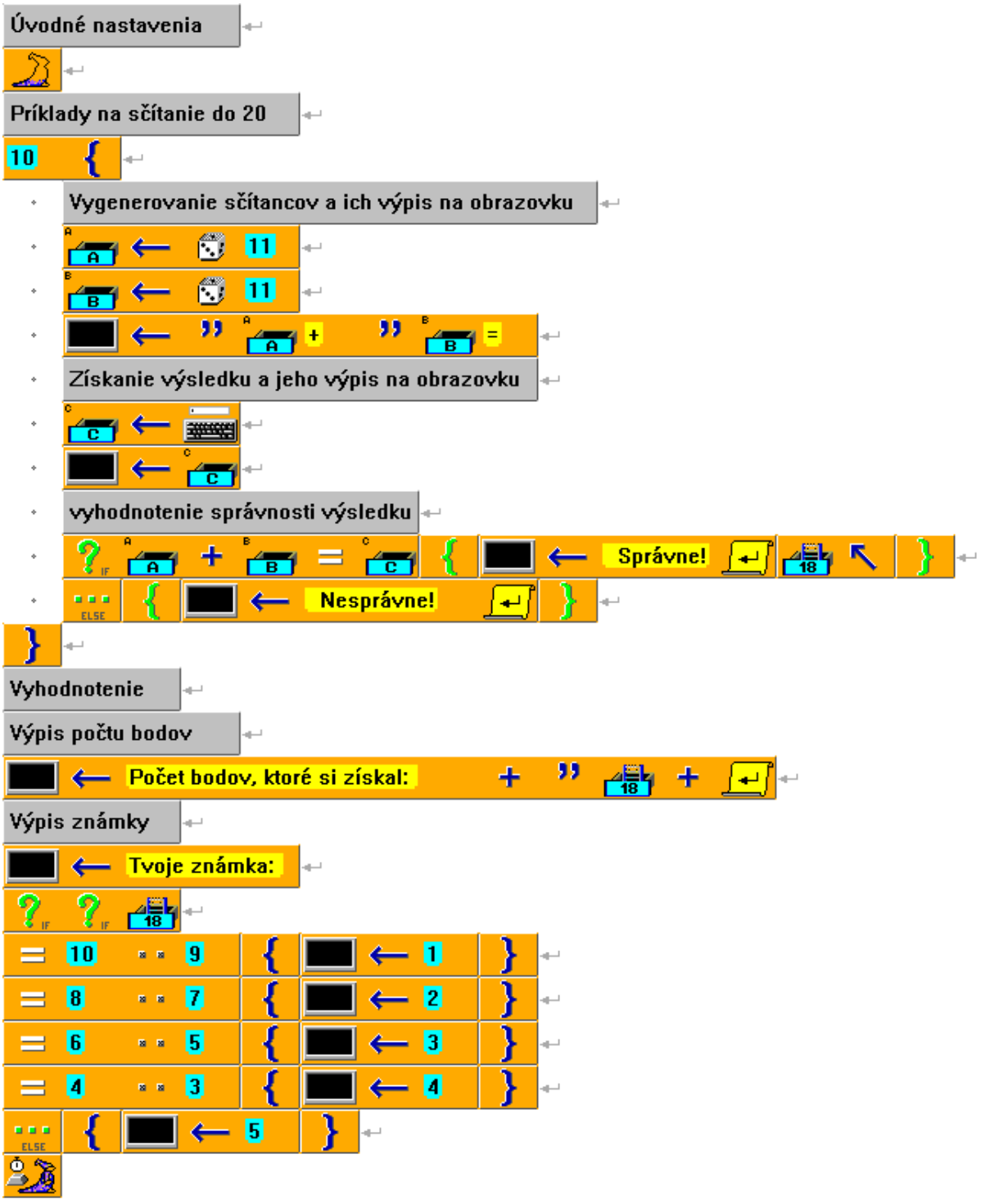
= 6 – 5 {„Známka: 3“}

= 4 – 3 {„Známka: 4“}

inak {„Známka: 5“}

Ako vidíme z našej ukážky, interval zapisujeme v Baltíkovi pomocou prvku 

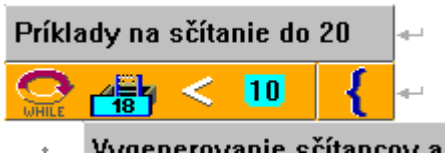
Celý náš program potom bude vyzerať nasledovne:



Skúsme si teraz program trošku obmeniť. Naším cieľom bude zadávať príklady dovtedy, kým užívateľ nevypočíta 10 z nich správne. Známku vynecháme, ale program spočíta, na koľko pokusov sa to užívateľovi podarí.

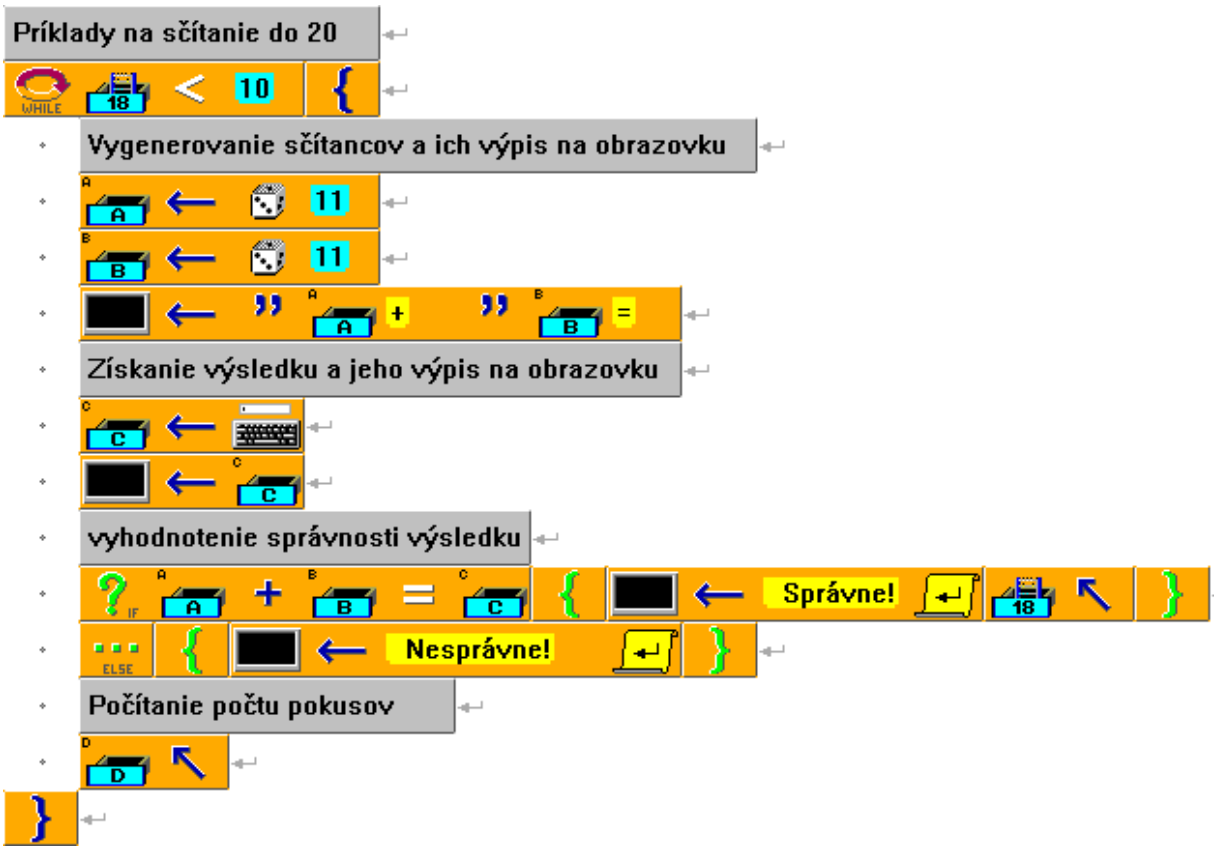
Hlavná časť programu, t. j. časť *Príklady na sčítanie do 20,* sa zmení len minimálne.

1. Nebude sa opakovať v pevne určenom počte, ale dovtedy, kým **Počítadlo** bude mať hodnotu menšiu ako 10. Na to je vhodný cyklus **while.**



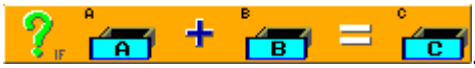
2. Aby sme mohli zistiť, koľko pokusov sme reálne potrebovali, zavedieme si ešte jednu premennú, napr. **D**, ktorej hodnotu budeme zvyšovať o 1 pri každom prebehnutí cyklu.

Celá táto časť bude potom vyzerať nasledovne:



Na záver programu vypíšeme, koľko pokusov sme potrebovali na vypočítanie 10 správnych príkladov.

Ako vidíme, premenné sa dajú v programovaní využívať vo veľkom ako súčasť podmienok. Rôznymi spôsobmi môžeme porovnávať ich hodnoty 

alebo hodnoty výrazov, ktoré pomocou nich vytvoríme 

. Aby tieto porovnania boli možné, používame na ne tzv. **relačné operátory.** Sú to znamienka:

 – je rovný,

 – je menší,

 – je väčší,

 – je menší nanajvýš rovný,

 – je väčší nanajvýš rovný,

 – nie je rovný.

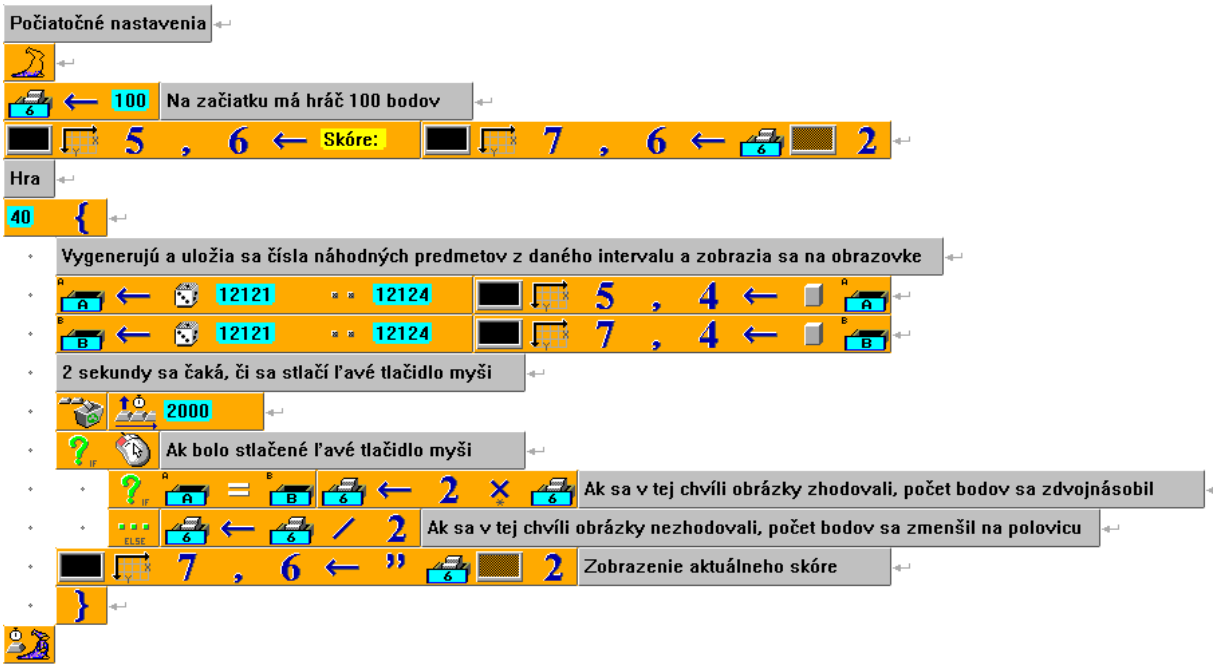
Ďalšie príklady:

Príklad č. 1

Program, v ktorom sa približne v strede obrazovky zobrazujú každé dve sekundy dva predmety vedľa seba. Predmety sú generované náhodne z intervalu predmetov č. 12121 – 12124. Zároveň hráč dostáva hneď na začiatku pridelených 100 bodov.

Po zobrazení obrázkov má hráč 2 sekundy na to, aby klikol ľavým tlačidlom myši. Stlačiť by ho mal v prípade, ak sú oba obrázky zhodné. Ak klikne správne, počet jeho bodov sa zdvojnásobí, ak klikne vtedy, keď sú obrázky rôzne, počet sa zmenší na polovicu. Ak do 2 sekúnd neklikne, nestane sa nič, iba sa vygenerujú nové obrázky. Celkovo má hráč 40 pokusov.

Riešenie:



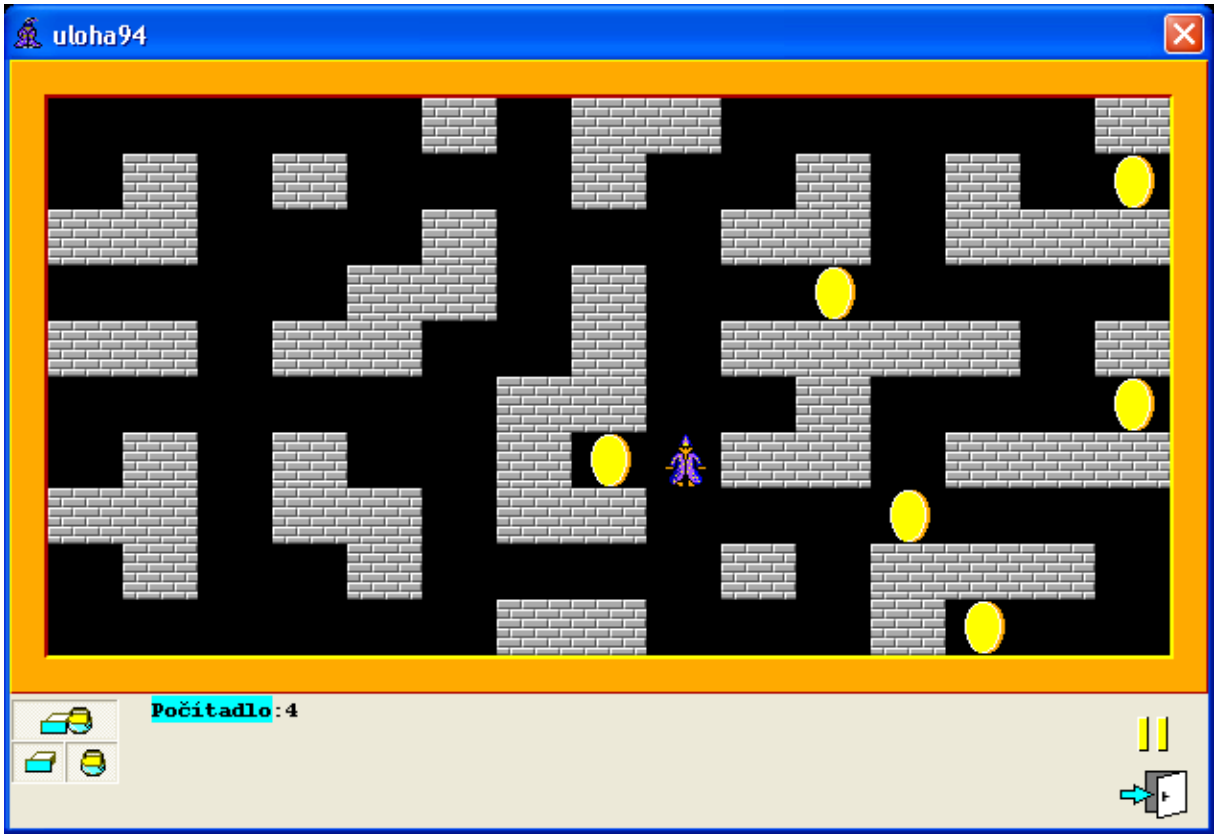
Príklad č. 2

Vráťme sa ku klasickým programom s bludiskami. Premenné nám dávajú ďalšie možnosti na rôzne obmeny týchto programov. Môžeme zbierať body za vyzbierané predmety, dostávať na niektorých miestach bludiska otázky z ľubovoľného predmetu, dostávať tresty vo forme

znižovania rýchlosti Baltíka, ak stúpime na zlé miesto, a naopak, zvyšovať jeho rýchlosť, ak stúpime na dobré miesto atď.

Ukážeme si program, v ktorom má Baltík v bludisku vyzbierať peniaze. Všetkých mincí je 10, a keď sa mu to podarí, objavia sa na náhodnom mieste dvere, do ktorých má vojsť.

Týmto sa ukážka skončí, ale program by samozrejme mohol pokračovať ďalším bludiskom, v ktorom by Baltík zasa zbieral niečo iné, napr. aj za náročnejších podmienok (časový limit alebo iné už spomínané možnosti).



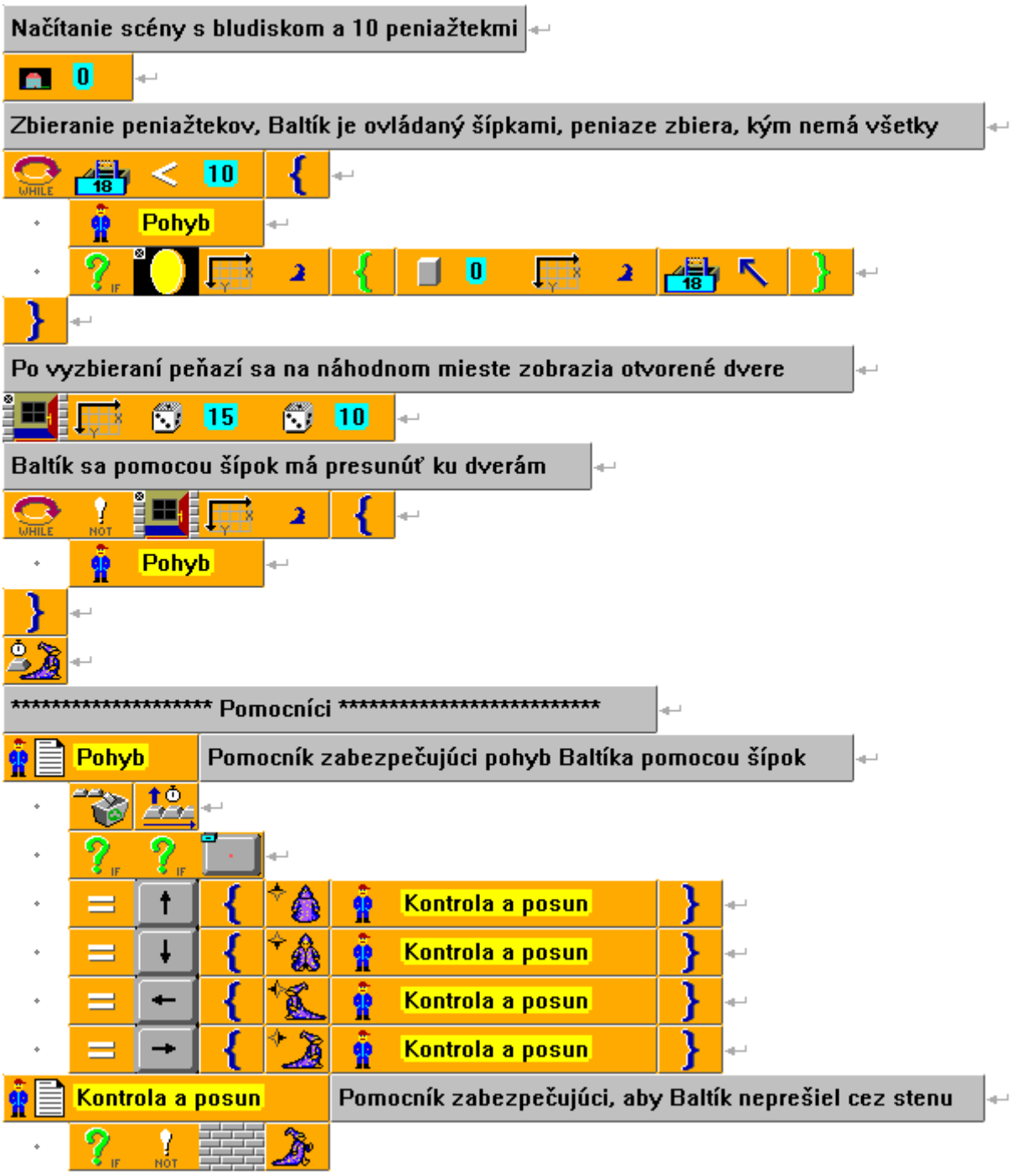
Na obrázku vidíme bludisko, v ktorom Baltík zbiera peniaze. Dole máme zobrazenú aktuálnu hodnotu premennej **Počítadlo,** ktorá ráta počet nazbieraných peňazí.

Na tomto obrázku vidíme situáciu po zozbieraní všetkých mincí.

Riešenie:

V riešení sa budú niektoré časti zdrojového kódu opakovať viackrát, preto využijeme pomocníkov.

Poznámka! Ak by sme sa rozhodli vytvoriť viac „levelov“, ktoré by sa od seba líšili iba druhom a počtom zbieraných predmetov, mohlo by byť aj celé naše terajšie riešenie v ďalšom pomocníkovi, ktorému by sme pri jeho volaní v programe vždy zadali ako parametre predmet, ktorý budeme zbierať, a jeho počet.


Načítanie scény s bludiskom a 10 peniažtekmi
0
Zbieranie peniažtekov, Baltík je ovládaný šípkami, peniaze zbiera, kým nemá všetky
WHILE
18
10
Pohyb
IF
2
0
2
18
Po vyzbieraní peňazí sa na náhodnom mieste zobrazia otvorené dvere
15
10
Baltík sa pomocou šípok má presunúť ku dverám
WHILE
NOT
2
Pohyb
********** Pomocníci **********
Pohyb
Pomocník zabezpečujúci pohyb Baltíka pomocou šípok
IF
IF
Kontrola a posun
Kontrola a posun
Kontrola a posun
Kontrola a posun
Kontrola a posun
Pomocník zabezpečujúci, aby Baltík neprešiel cez stenu
IF
NOT

16 ZLOŽENÉ PODMIENKY

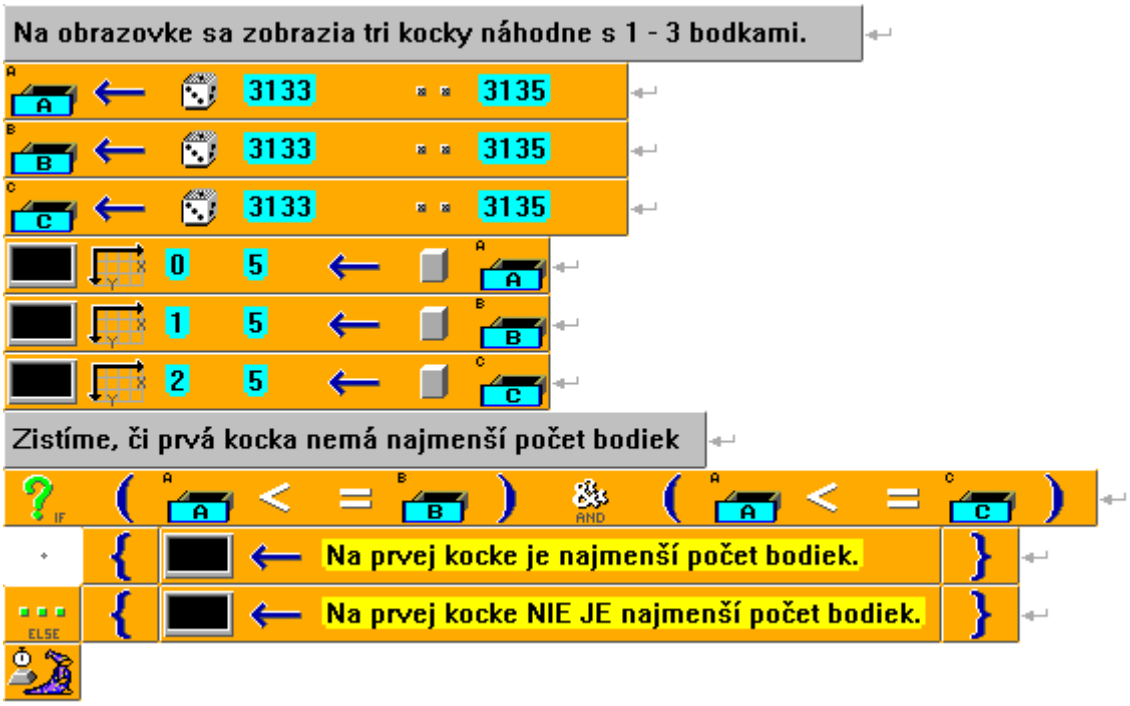
Aj v reálnom živote sa nám stáva, že vykonanie nejakej akcie podmieňujeme splnením viacerých podmienok, resp. jednej z viacerých podmienok. Napríklad povieme:
„Ak si spravíš domácu úlohu a upraceš si izbu, môžeš ísť von.“
Takúto formuláciu volíme v prípade, ak akciu podmieňujeme splnením oboch podmienok. Ak čo len jedna splnená nebude, dotyčný ostáva sedieť doma.
Pozrime sa na druhú formuláciu:
„Ak mi zarecituješ básničku alebo zaspievaš pesničku, dám ti čokoládu.“
Táto formulácia znamená, že stačí, aby dotyčný splnil jednu z oboch podmienok (ale samozrejme môže aj obe) a čokoládu dostane.
Tieto dve formulácie používame aj v programovaní. O tom, či musia byť splnené obe podmienky, alebo stačí len jedna, rozhoduje, či na spojenie jednoduchých podmienok do zloženej použijeme spojku **A** (prípadne **A ZÁROVEŇ**) alebo **ALEBO.** V programovaní používame ich anglické názvy **AND** (a) a **OR** (alebo).

Na vytvorenie zložených podmienok používame tzv. **logické operátory:**

– operátor **AND** (a zároveň) – tzv. **logický súčin**

Tento prvok nahrádza v našom programe slovo **a**, resp. **a zároveň** (anglicky **and**). Ak sú dve podmienky spojené takýmto spôsobom, zložená podmienka bude splnená (t. j. pravdivá) len v prípade, že obe podmienky sú splnené (pravdivé). V prípade, že aspoň jedna z podmienok pravdivá nebude, nebude pravdivá ani celá zložená podmienka.

Napríklad:

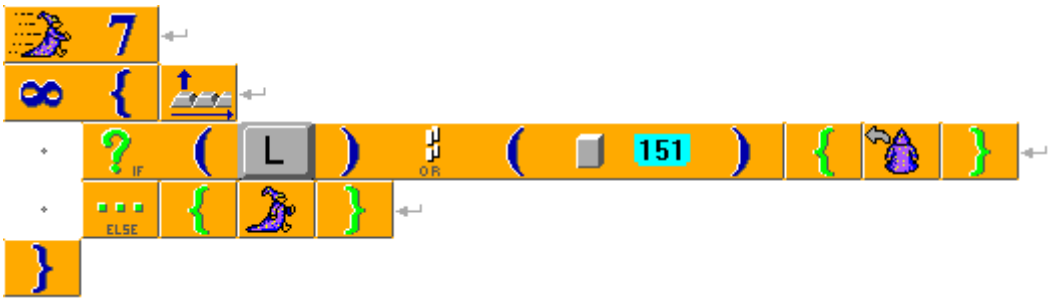


Jednoduché podmienky sme dali do tzv. **logických zátvoriek**  . Nie je to v tomto prípade nevyhnutné, ale v prípade zložitejších podmienok alebo výrazov v podmienkach by mohol program bez zátvoriek prečítať podmienku inak, ako chceme. Navyše tieto zátvorky značne zvyšujú prehľadnosť zápisu podmienky.

 – **operátor OR** (alebo) – tzv. **logický súčet**

Tento prvok nahrádza v programe slovo **alebo** (anglicky **or**). Ak sú dve podmienky spojené týmto spôsobom, výsledná podmienka bude splnená (pravdivá) v prípade, ak bude splnená (pravdivá) aspoň jedna z týchto podmienok. Iba v prípade, že nebude splnená ani jedna z daných podmienok, nebude pravdivá ani celá zložená podmienka.

Napríklad:



V tomto programe bude Baltík bežať rovno dovtedy, kým buď nestlačíme kláves L, alebo nenarazí na okraj obrazovky (predmet 151).

Pravdivostná hodnota zložených podmienok

Ak p1 a p2 sú dve podmienky spojené logickým súčinom alebo súčtom, pričom hodnota 1 znamená, že je podmienka pravdivá (splnená) a hodnota 0, platia pre logický súčet a súčin nasledovné hodnoty:

p1	p2	p1 AND p2	p1 OR p2
1	**1**	**1**	**1**
1	**0**	**0**	**1**
0	**1**	**0**	**1**
0	**0**	**0**	**0**

– operátor **XOR** (výlučné alebo, non ekvivalencia)

Tento operátor je špeciálnym prípadom operátora **OR.** Používa sa vtedy, ak chceme, aby zložená podmienka bola pravdivá len v prípade, že platí **práve jedna** z podmienok, t. j. nesmú byť pravdivé obe súčasne.

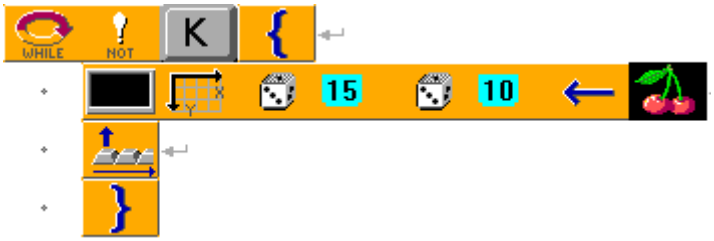
– operátor **NOT** (neplatí) – tzv. **negácia**

Tento prvok poznáme. Nespája dve podmienky dohromady, ale patrí medzi logické operátory. Ak ho vložíme pred podmienku, spôsobí, že sa bude zisťovať platnosť jej opaku, t. j. negácie. Z toho vyplýva, že negácia podmienky musí mať vždy opačnú pravdivostnú hodnotu ako pôvodná podmienka. Ak je podmienka pravdivá, jej negácia je nepravdivá a naopak.

Napríklad:

Baltík bude v tomto programe chodiť rovno dovtedy, kým pred ním **NIE JE** nejaký predmet. Potom sa otočí doľava.

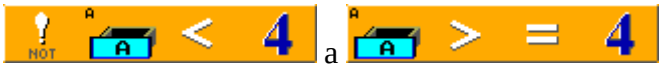
Druhý príklad:



Dokiaľ nebude stlačený kláves K, budú sa na obrazovke na náhodných miestach objavovať čerešničky.

V podmienkach tvorených premennými sa obvykle môžeme rozhodnúť, či ich negáciu zapíšeme pomocou prvku **negácia,** alebo zapíšeme jej opak.

Príklady rovnakej podmienky:



Vytvorenie opaku podmienky **a<4** nie formou negácie, ale zapísaním opačnej podmienky nesie so sebou riziko, že niečo prehliadneme, napr. zabudneme dať aj znamienko = . Naopak, ak pred túto podmienku dáme prvok negácie, máme zaručené, že sme určite vytvorili opak pôvodnej podmienky.

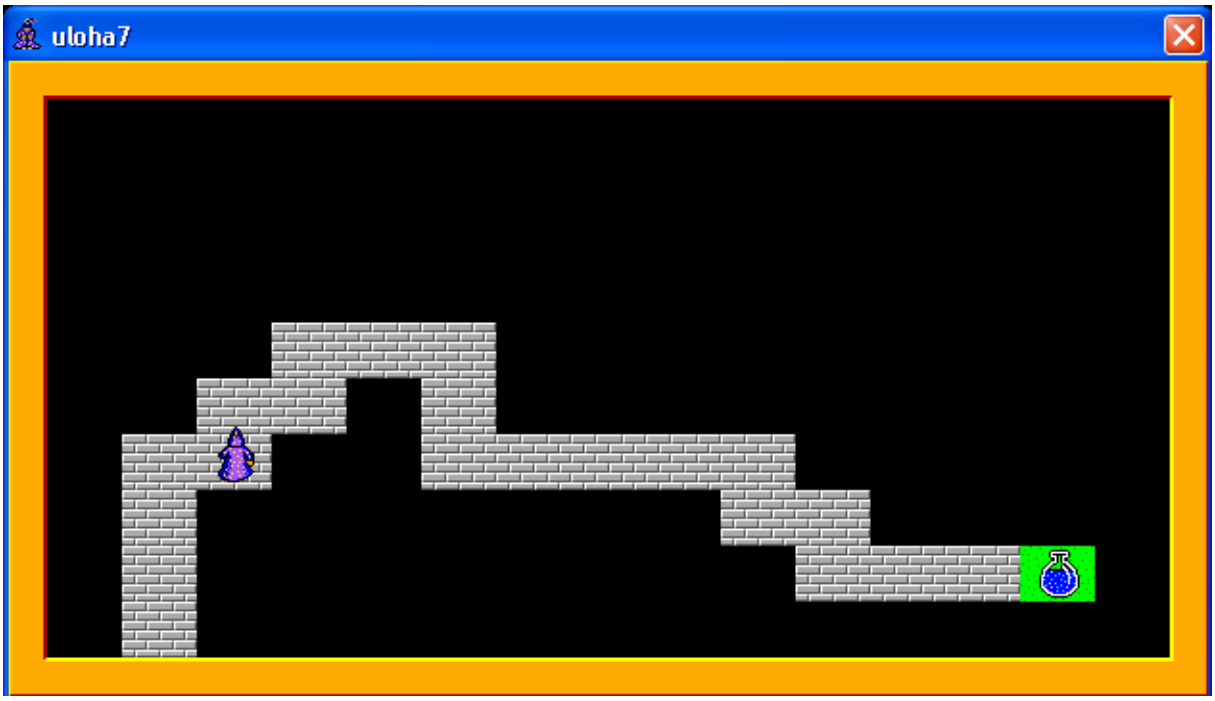
Pri negovaní zloženej podmienky je vytvorenie jej opaku zložitejšie:

zložená podmienka	**jej negácia**	**alternatíva jej negácie**
p1 AND p2	**!(p1 AND p2)**	**!p1 OR !p2**
p1 OR p2	**!(p1 OR p2)**	**!p1 AND !p2**

Pozor! Nesprávna negácia zloženej podmienky, najmä používanie alternatívy jej negácie (pozri 3. stĺpec tabuľky) je jednou z najčastejších logických chýb v programe.

Príklad

Ukážme si program, v ktorom sa vygeneruje náhodne jedna zo štyroch scén. V každej sa nachádza inak zakrútená cestička, na konci ktorej je banka so živou vodou. Baltík má bez našej pomoci prejsť sám po cestičke až k banke.



Riešenie

Aby sme pochopili riešenie, musíme si najprv niečo povedať o hľadaní cesty z bludiska. Existuje všeobecné riešenie, ako hľadať cestu von, resp. ako docieliť, aby Baltík našiel správnu cestu po ľubovoľnej kľukatej cestičke.

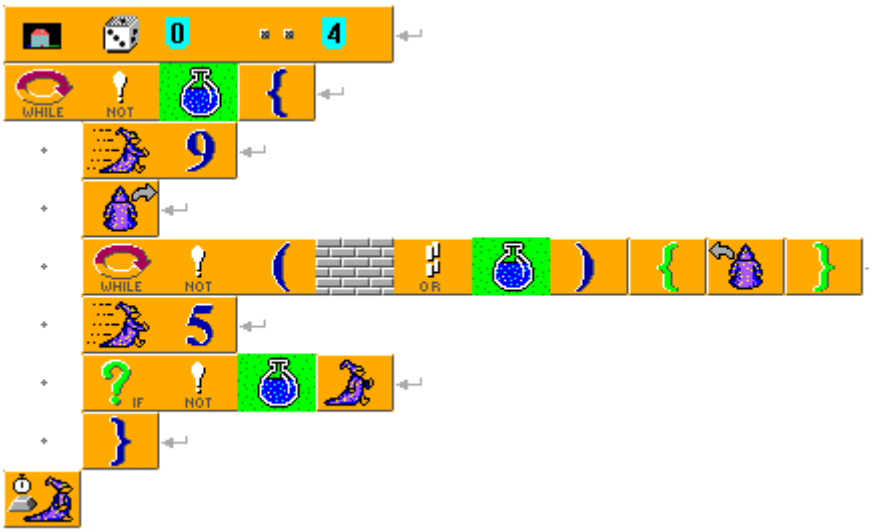
Algoritmus má nasledovný princíp:

1. Baltík sa otočí doprava.
2. Kým nebude pred ním priechod, točí sa doľava.
3. Urobí krok.

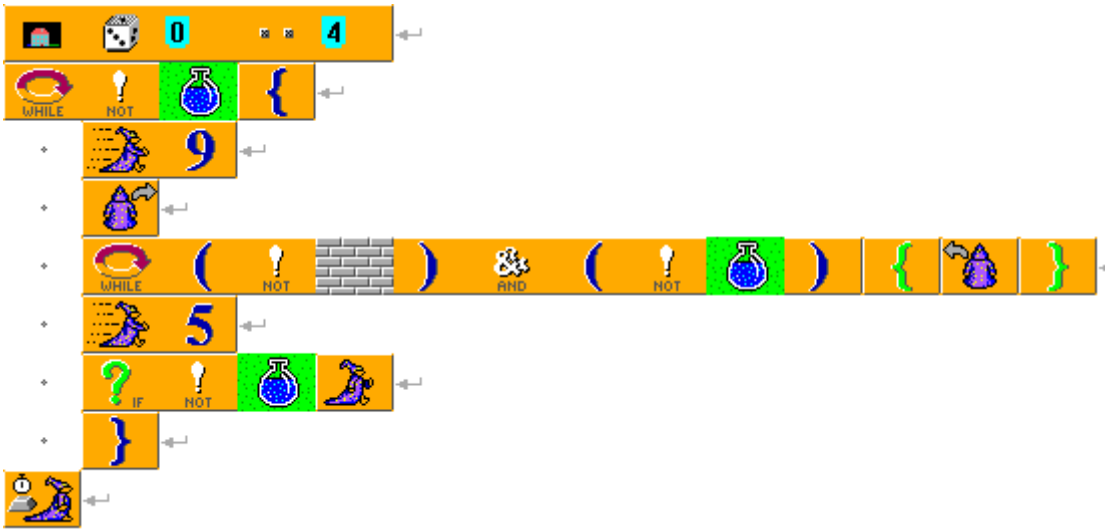
Tento algoritmus môžete uplatniť v akomkoľvek bludisku.

V tomto príklade sa v 2. kroku musí Baltík točiť doľava dovtedy, kým nebude pred ním dlažba cestičky alebo banka so živou vodou.

V 3. kroku sa pohne dopredu vtedy, keď sa pred ním nenachádza banka so živou vodou. Ak by pred ním banka bola, program sa skončí.



Samotnú zloženú podmienku máme v 5. riadku programu. Je to negácia podmienky „pred Baltíkom je dlažba alebo banka s vodou“. Podľa tabuľky negácií zložených podmienok môžeme túto negáciu napísať aj takto:



Keď si vyskúšame oba programy, zistíme, že fungujú úplne rovnako.

Môže sa nám stať, že v jednej podmienke použijeme viac logických operátorov. V tom prípade potrebujeme vedieť, v akom poradí sú vyhodnocované.

V programovaní platí, že bez použitia zátvoriek sa najskôr vyhodnotí operátor **NOT,** potom operátor **AND** a nakoniec operátor **OR.**

Príklad:

Táto podmienka je splnená (pravdivá), ak je pred Baltíkom nejaký predmet **a súčasne** bol stlačený aspoň jeden z klávesov A **alebo** B.

Oproti tomu tá istá podmienka zapísaná bez zátvoriek:

Táto podmienka je splnená práve vtedy, ak platí aspoň jedno z nasledujúcich tvrdení:

pred Baltíkom je predmet **a súčasne** bol stlačený kláves A

alebo

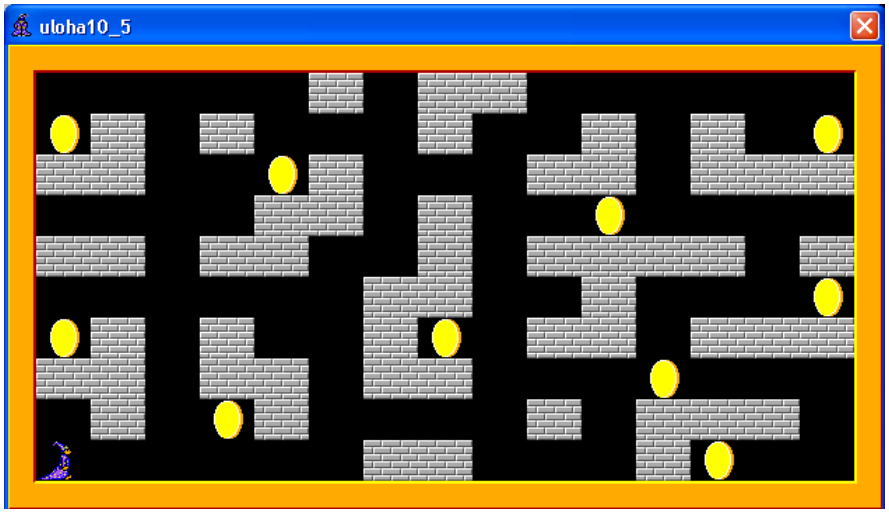
bol stlačený kláves B (v tomto prípade už nezáleží na tom, čo je pred Baltíkom).

Poznámka: Predchádzajúcu podmienku by sme pomocou zátvoriek zapísali takto:

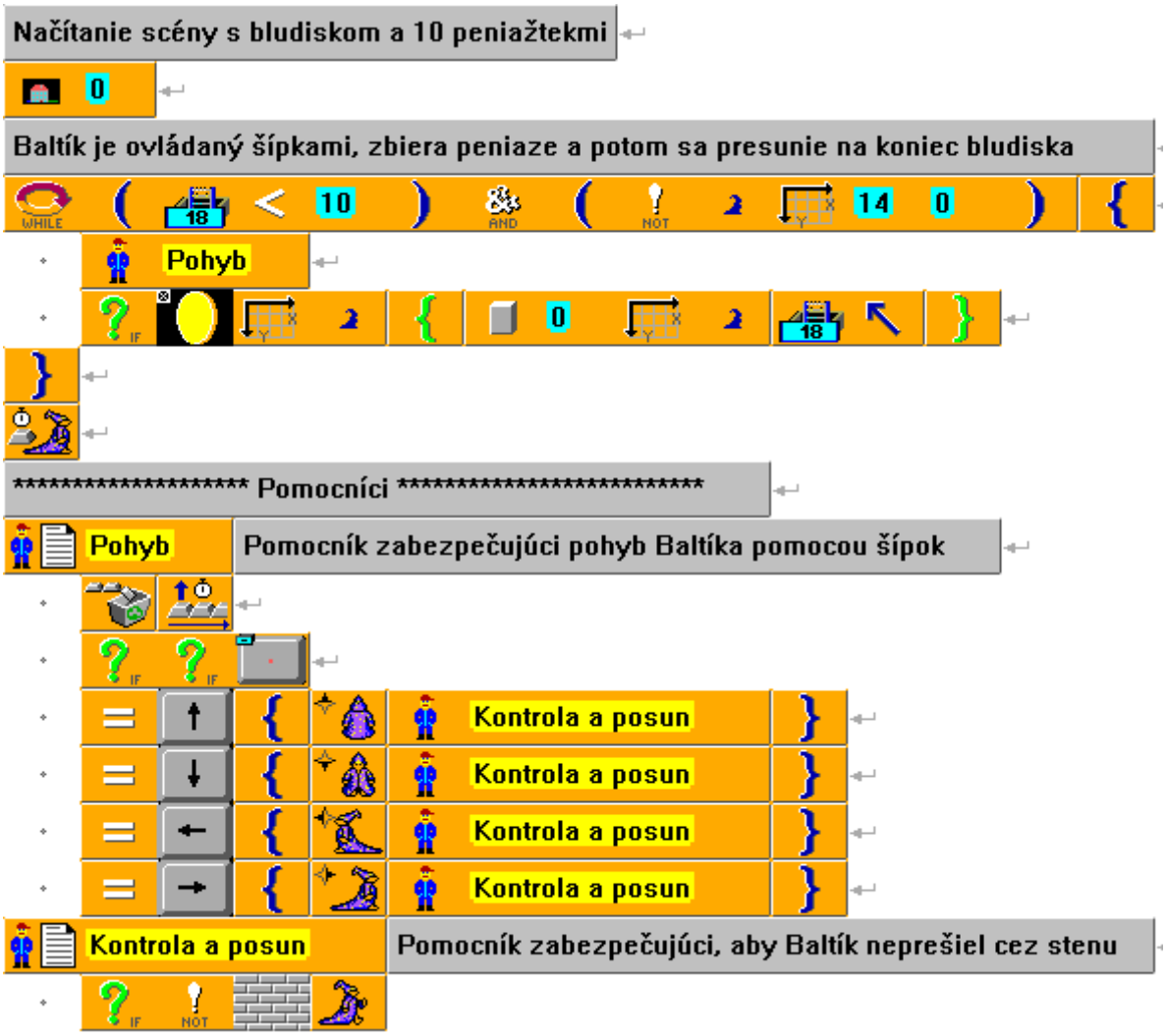
Ak spájame dohromady viac ako dve podmienky, stávajú sa logické zátvorky nevyhnutnou pomôckou pri označení ich správneho zoskupenia.

Príklad:

Vytvorme bludisko s peniažkami, ktoré je potrebné zozbierať. Postup do ďalšieho bludiska, resp. koniec programu nastane až vtedy, keď Baltík zozbiera všetky peniažky a zároveň sa presunie do pravého horného rohu obrazovky, ktorý budeme považovať za východ z bludiska.



Riešenie:


Načítanie scény s bludiskom a 10 peniažtekmi
0
Baltík je ovládaný šípkami, zbiera peniaze a potom sa presunie na koniec bludiska
WHILE
10
AND
NOT
14
0
Pohyb
IF
2
0
2
18
**********Pomocníci**********
Pohyb
Pomocník zabezpečujúci pohyb Baltíka pomocou šípok
IF
IF
Kontrola a posun
Kontrola a posun
Kontrola a posun
Kontrola a posun
Kontrola a posun
Pomocník zabezpečujúci, aby Baltík neprešiel cez stenu
IF
NOT

17 GRAFICKÉ PRÍKAZY

V programoch nemusíme byť závislí iba od obrázkov, ktoré vložíme do programov z bánk predmetov alebo z nejakého súboru. Na niektoré geometrické tvary sú špeciálne príkazy, pomocou ktorých môžeme vytvoriť ľubovoľný obdĺžnik, elipsu, čiaru alebo bod. Ich kombináciou následne môžeme vytvoriť množstvo zaujímavých tvarov a obrázkov, s ktorými sa dá ďalej pracovať.

V Baltíkovi máme tieto grafické príkazy:

 **Elipsa** – slúži na kreslenie elipsy a kruhu.

 **Obdĺžnik** – slúži na kreslenie obdĺžnika a štvorca.

 **Čiara** – slúži na kreslenie čiary.

 **Bod** – slúži na vykreslenie bodu a zisťovanie farby bodu v scéne.

 **Sprej** – môžeme ním sprejovať, používať sprej.

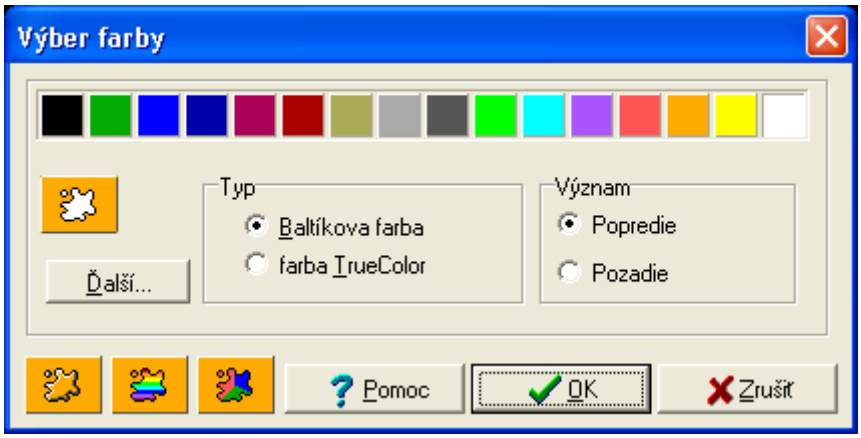
 **Výplň** – slúži na vyplnenie uzavretých obrazcov, napr. elipsy, obdĺžnika atď.

Elipsa a obdĺžnik môžu mať za sebou tri základné parametre, ktorými môžeme nastavovať:

1. hrúbku čiary, ktorá ich tvorí,
2. ich farbu, prípadne farbu ich výplne,
3. ich umiestnenie spolu s ich rozmerom.

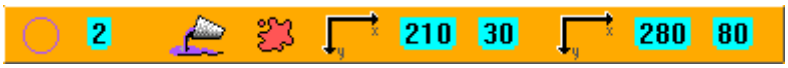
Parametre môžeme nastavovať všetky alebo len niektoré z nich, prípadne ani jeden. Ak použijeme niekoľko parametrov, musíme ich zapisovať v poradí, v akom sme ich vymenovali.

Farbu vkladáme pomocou prvku  . Po jeho vložení do programu sa otvorí dialógové okno, v ktorom vyberieme príslušnú farbu.



Ukážme si niekoľko príkladov.

Elipsa sa vykreslí hrúbkou čiary 2, farba čiary bude červená. Bude sa nachádzať v priestore ohraničenom (opísanom) obdĺžnikom, ktorého ľavý horný roh bude na bodových súradniciach 100, 30 a pravý dolný roh na súradniciach 200, 80.

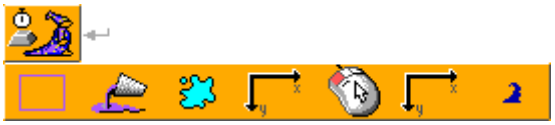


Rozdiel oproti prvému zápisu je v tom, že táto elipsa bude celá vyfarbená na červeno. Parameter hrúbky čiary je v tomto prípade úplne zbytočný a môžeme ho vynechať.

 alebo

V tomto prípade sa vykreslí presne do políčka pred Baltíkom žltý obdĺžnik alebo elipsa. Hrúbka čiary bude samozrejme 1, keďže sme ju nenastavili inak.

V tomto prípade sa vykreslí v danej oblasti elipsa s hrúbkou čiary 4. Keďže sme neurčili jej farbu, bude automaticky biela.



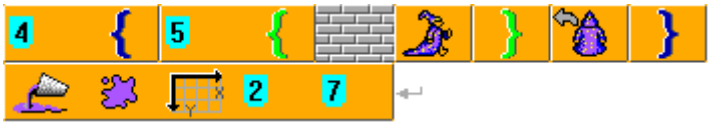
Ak klikneme pred príkazom na vykreslenie obdĺžnika niekde na obrazovku programu, obdĺžnik sa vykreslí v hraniciach od nami určeného bodu po Baltíka.

Prvok **Bod** sa používa veľmi často v podmienkach, kde sa môžeme pýtať na jeho farbu. Skúsme predchádzajúci program pre bledomodrý obdĺžnik doplniť ešte týmto zdrojovým kódom:

Výsledkom bude program, v ktorom Baltík pôjde dopredu vždy, keď sa nastavíme kurzorom myši na modrý obdĺžnik.

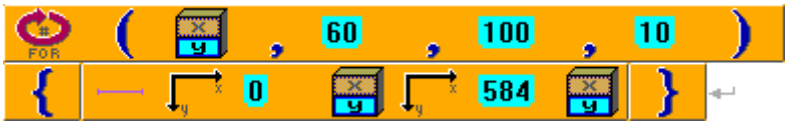
Pre čiaru sú parametre veľmi podobné ako pre elipsu a obdĺžnik, tu nemá zmysel uvažovať o výplni:

Príkaz pre výplň môžeme používať nielen pre elipsu a obdĺžnik, ale aj samostatne. Vyskúšajme si to v nasledujúcom programe:

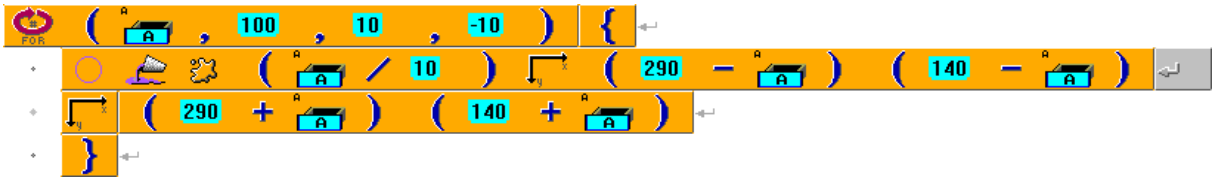


V tomto programe Baltík ohraničí určitú oblasť múrikom. Potom zadáme pri výplni ľubovoľný bod vo vnútri tejto oblasti a fialová farba sa rozleje po celej oblasti, až kým nenarazí na ohraničenie (v tomto prípade múrik).

Ukážme si dva príklady, keď môžeme vytvoriť zaujímavé útvary pomocou grafických príkazov a cyklu **for:**

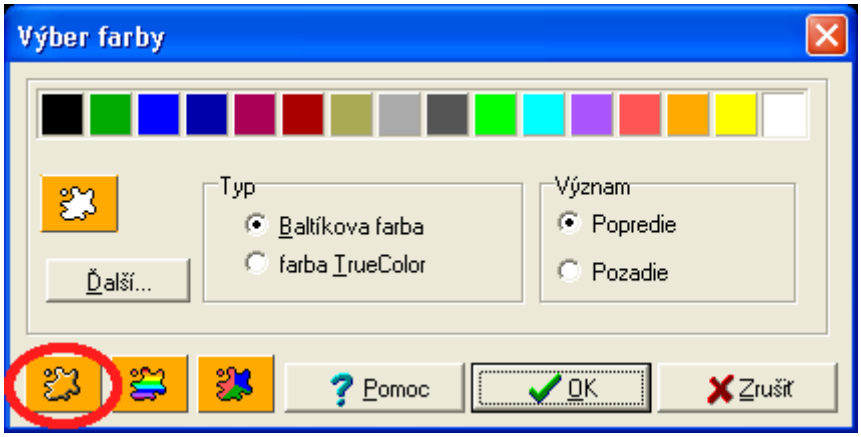


Tento cyklus nám vykreslí notovú osnovu (samozrejme bez husľového kľúča ☺).



Približne do stredu obrazovky sa vykreslí terč. Keďže príkaz pre elipsu bol v tomto prípade príliš dlhý, aby sa zmestil na jeden riadok, rozdelili sme ho „šedým Enter“ – prvok **rozdelenie riadka.** Spôsobí, že príkaz, ktorý takto rozdelíme, sa nepreruší, na rozdiel od klasického „čierneho Enter“, použitím ktorého by príkaz nefungoval správne.

Možno sa zastavíme pri probléme, ako dostať do príkazu na vykreslenie elipsy prvok  .
Nájdeme ho v dialógovom okne pre výber farby:



Za týmto prvkom musíme napísať číslo farby, ktorú chceme vykresliť, pričom základné farby, ktoré sú v Baltíkovi ponúknuté, majú čísla od 0 po 15 (pozri celočíselné konštanty).

ZÁVER

Učebný zdroj Baltík – úroveň mierne pokročilý obsahuje 17 tém venovaných metodike vyučovania programovania na základnej škole na pokročilejšej úrovni.

Je určený predovšetkým ako materiál pre účastníkov vzdelávacieho programu Baltík – programovanie na základnej škole – úroveň mierne pokročilý. Po ukončení vzdelávacieho programu ho absolventi môžu veľmi vhodne využiť na vyučovaní informatickej výchovy alebo informatiky v školách, ako aj v rámci voľnočasových aktivít, na krúžkoch, najmä pri zapájaní sa žiakov do súťaží v programovacom jazyku Baltík, ktoré sa organizujú na školskej, oblastnej a celoslovenskej úrovni. Najlepší žiaci majú možnosť dostať sa aj do medzinárodného kola.

Baltík je programovací jazyk, pomocou ktorého môžeme žiakov naučiť takmer všetky typické programové konštrukcie. Napriek tomu, že je svojou skladbou a metodikou programovania určený už pre žiakov na 1. stupni základných škôl, obsahuje dostatočne silné nástroje aj pre tých, pre ktorých sa programovanie stane viac ako len súčasť informatiky v škole. Okrem toho je jeho veľkou výhodou tvorba príkazov pomocou ikon, čím žiakov odbremeňujeme od zbytočných chýb v preklepoch a „bodkočiarkach“. Naopak, nevýhodou sa stáva problém so spustením Helpu v novších verziách OS Windows, keďže práve ten bol často najlepším pomocníkom pre žiakov, ktorí „prerástli“ svojho učiteľa.

Pre každého učiteľa je radosť, keď stretne žiaka, ktorý chce vedieť viac. A preto je potrebné, aby sme takéhoto žiaka dokázali usmerniť a ukázať mu aj oblasti, ktoré sú už tak trochu nad rámec osnov.

Ani tento materiál neposkytuje úplný výklad všetkých možností, ktoré Baltík prináša. Naznačuje však možnosti, kadiaľ sa môžeme v rámci programovania vydať. Či už s talentovanými žiakmi, alebo pri príprave vlastných edukačných programov.

Materiál bude zverejnený na webovej stránke Metodicko-pedagogického centra Bratislava vo formáte PDF. Dúfame, že prispeje k rozvoju profesijných kompetencií pedagogických zamestnancov najmä na prezenčnej i dištančnej forme vzdelávacieho programu. Materiál je možné a vhodné využívať aj na vyučovaní informatickej výchovy a informatiky na základných školách, resp. v nižších ročníkoch osemročných gymnázií.

ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV

GALBAVÁ, Ľ., KRIŠTOFOVÁ, Z. 2011. *Baltík – pracovné listy* [online]. [cit. 25-11-2013]. Dostupné na internete: <http://www.mpc-edu.sk/library/files/baltik___pracovne_listy_web.pdf>. ISBN: 978-80-8052-373-2.

KRIŠTOFOVÁ, Z., ULIČNÁ, E. 2011. *Baltík – úroveň začiatočník* [online]. [cit. 25-11-2013]. Dostupné na internete: <http://www.mpc-edu.sk/library/files/baltik_kniha_web.pdf>.ISBN: 978-80-8052-363-3.

Názov:	Baltík pre mierne pokročilých
Autori:	Ing. Zuzana Krištofová, Mgr. Eva Uličná
Recenzenti:	Ing. Peter Michálek
	RNDr. Janka Schreiberová
Vydavateľ:	Metodicko-pedagogické centrum v Bratislave
Odborná redaktorka:	Mgr. Terézia Peciarová
Grafická úprava:	Ing. Zuzana Krištofová, Mgr. Eva Uličná
Vydanie:	1.
Rok vydania:	2014
Počet strán:	82
ISBN	978-80-8052-589-7