

Jak vzniká software

Pro SUPŠ HNN vypracoval Vladimír Bureš 2006 – 2008

verze 0.4



Jaký software obsahuje počítač?

- **Software je programové vybavení počítače** bez něj by počítač byl jen mrtvý kus železa. První software který se spustí po spuštění počítače je **BIOS** jenž vytváří základní vrstvu pro vyšší programy jako například je operační systém. BIOS také poskytuje základní funkčnost klávesnice, diskových řadičů, grafické karty a řízení komunikačních portů. Nastavení BIOSu určuje z kterého disku se bude zavádět operační systém. Po úspěšném proběhnutí se postará se o spuštění zavaděče systému (z oblasti MBR na vybraném disku viz. disk).
- **Operační systém** je sada programů (software) umožňujících co nejefektivnější využití hardware počítače. Jádro operačního systému se stará o správu paměti, běh více procesů (**multitasking**), způsob ukládání dat na disky (**souborový systém**) a o komunikaci s hardwarem počítače (**ovladače**). Hlavním úkolem operačního systému je zabezpečit běh a programovou podporu aplikačních programů (viz rozhraní API).
- **Aplikační software** je jakýkoliv program který používáme na svém PC. Aplikace komunikuje s uživatelem pomocí uživatelského prostředí (**UI – user interface**), které je buďto její vlastní nebo zprostředkováno systémem. Programů je několik druhů například Kancelářské programy, databázové aplikace, systémové nástroje (**utility**) a mnoho dalších k nimž se vrátíme později...

Jak software vzniká – nižší programovací jazyky

- **Software je programové vybavení počítače.** Software tvoří programátoři za pomoci programů jenž se nazývají programovací jazyky. Programovací jazyky se dělí na **nižší programovací jazyky** a **vyšší programovací jazyky**.
- **Assembler – nižší programovací jazyk** - je velice blízký strojovému kódu, též se nazývá **jazyk symbolických adres**. Assembler tvoří pouze zástupné symboly, které přímo odpovídají strojovému kódu. V assembleru je možno naprogramovat **velmi rychlý a paměťově nenáročný** kód. Assembler se například používá pro tvorbu jádra operačního systému, nebo pro výpočetně náročné operace.

Například strojovou instrukci procesoru (*Intel 80386*)

10110000 01100001

je mnohem snazší používat ekvivalentní zápis v assembleru:

mov al, 61h

- Zapsaný kód znamená **přesun hexadecimální hodnoty 61** (97 dekadicky) **do registru procesoru pojmenovaného „al“**. Název instrukce „mov“ (zkratka anglického slova move – přesun) je následován seznamem parametrů. Tak vypadá typická instrukce assembleru.

Jak software vzniká – vyšší programovací jazyky

- **Vyšší programovací jazyky** – jsou jazyky jazyky v kterých se programuje aplikační software již pod určitým operačním systémem. Ve vyšších programovacích jazycích již nepracujeme přímo s hardwarovými adresami, jak to bylo u assembleru, ale používáme programových struktur zvoleného programovacího jazyka (například procedury a funkce, nebo proměnné a datové typy)
- **Jazyk BASIC** - V 80. létech našel BASIC široké uplatnění na domácích mikropočítačích. Původně byl zaveden jako jednoduchý nástroj pro výuku programování bez možností složitějších datových struktur jako jsou například objekty. (dnes se používá Visual Basic, který je objektivě orientován)
- **Jazyk PASCAL** - je název programovacího jazyka, určeného hlavně k výuce programování. Návrh programovacího jazyka Pascal pochází ze začátku 70. let. V oblasti PC dosáhla patrně největšího úspěchu implementace **Turbo Pascal** firmy Borland. Objektové rozšíření Pascalu se pak stalo i základem systému **Delphi** téže firmy. Pascal byl již od počátku velmi mocný a zároveň přísný jazyk, připravující programátory na přestup na jazyk **C**
- **JAZYK C** - V současné době je to jeden z nejpopulárnějších a nejvýkonnějších jazyků, zřejmě nejčastější pro psaní systémového softwaru. Ale je velmi rozšířený i pro aplikační software. V dnešní době je populární verze **C++** nebo **C#**. Nejznámější pro Windows je Borland C a Ms Visual C.



```
for i = 1 to 10 step 1  
  a = a + 1  
  b = a * i  
next i
```

```
for i := 1 to 10 do  
begin  
  a := a + 1;  
  b := a * i;  
end;
```

```
for ( i = 1; i < 10; i++ )  
{  
  a++;  
  b := a * i;  
};
```

Jak software vzniká – programové struktury

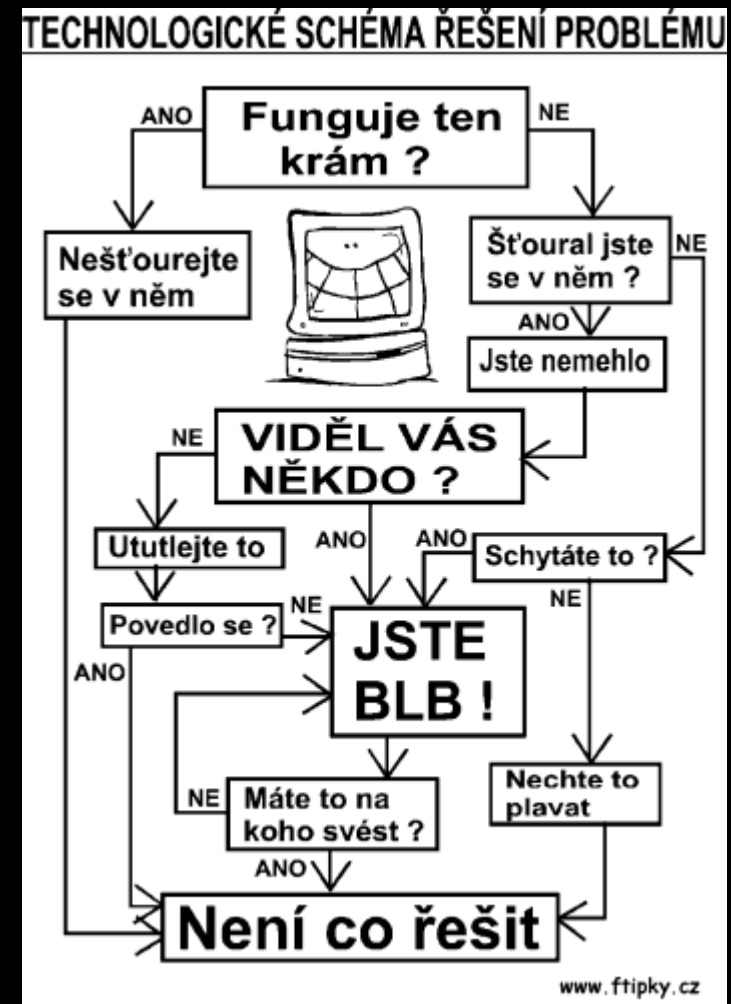
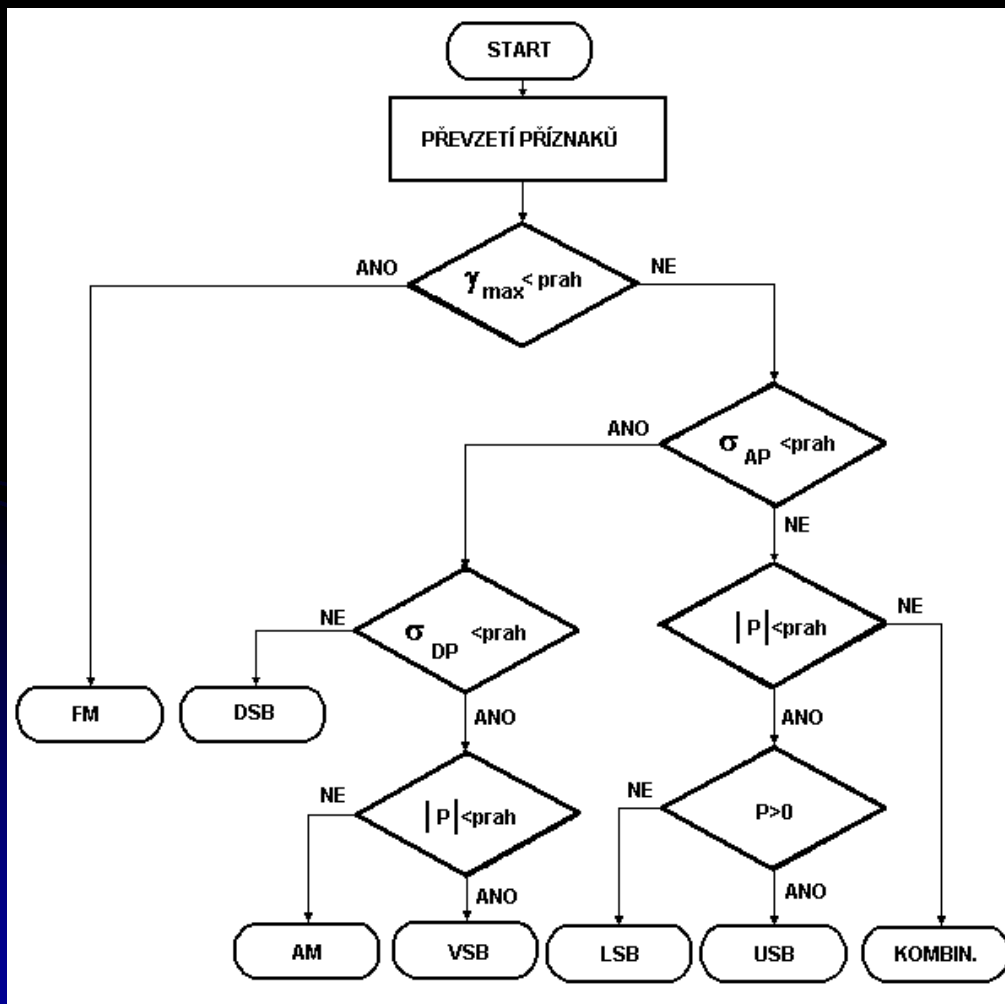
- **Vyšší programovací jazyky** - používají programové struktury, které programátorovi ulehčují jeho práci a zpřehledňují zdrojový kód programu. **Základní programové struktury jsou:**
- **Proměnná** - může nabývat hodnot, podle svého datového typu například `a := 'ahoj'` je textová proměnná `a`, která je textového typu `String`.
- **Podmínka** - položí dotaz a umožní větvení programu na dvě varianty – výsledek dotazu je **kladný** a výsledek dotazu je **záporný** např. `IF a = 'ahoj' THEN kód splněno ELSE kód nesplněno`.
- **Cykly** - opakuje stále danou část kódu dokud není splněna podmínka na ukončení cyklu. Při tvorbě cyklu je třeba dávat pozor, aby taková situace mohla nastat a nevznikl nekonečný cyklus. Ukázka cyklu `WHILE a < 10 DO a := a + 1; // opakuje dokud proměnná a není 10`.
- **Procedura** - umožňuje vytvořit část kódu, kterou lze kdykoliv v případě potřeby vyvolat. Procedura nahradila starší datovou strukturu nazvanou **podprogram**. Procedura může mít vstupní parametry, které ji programátor může předat.
- **Funkce** - podobné jako procedura, ale může mít kromě vstupních parametrů, také výstupní parametr – neboli výsledek; například `a := VypocitejTretiMocninu(10)`
- **Objekty** - objekt je základem objektově orientovaného programování. Objekt může mít své potomky, kteří kromě svých vlastností dědí také vlastnosti svého rodiče. **Například:** objekt letadlo: `tLetadlo = type(tObjekt)`; Pokud potřebujeme něco upřesnit vytvoříme třídu která dědí vlastnosti a dále je rozšiřuje `tCessna = type(tLetadlo)`

Jak software vzniká – Životní cyklus aplikace

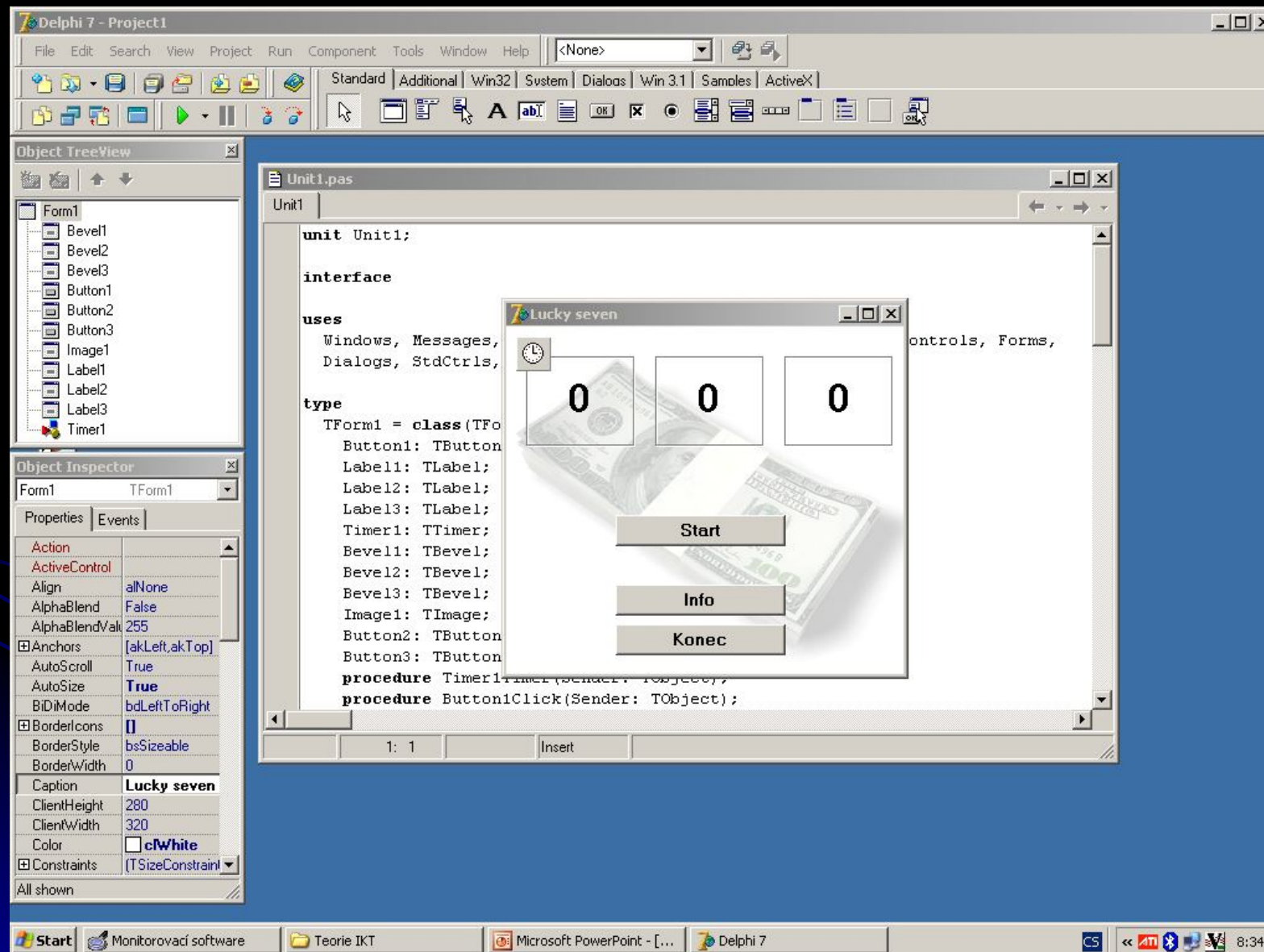
- **Zadání a analýza problému** – V případě software na zakázku probíhá konzultace se zákazníkem, sepsání smlouvy a určení předběžné ceny. V případě software který se bude prodávat (**komerční software**) a nebo šířit zdarma (**open source** nebo **freeware**) musí proběhnout průzkum co bude od programu vyžadováno koncovou skupinou uživatelů.
- **Návrh datových struktur** – práce pro programátora analytika, který vezme v potaz potřeby uživatelů a dostupné technické možnosti a prostředky. Zhotoví **datové struktury** a navrhne způsob, kterým bude software fungovat. **Správný návrh je nejdůležitější část tvorby programu**, protože právě na tomto bude celý projekt postaven. V této fázi pomůže pro zpřehlednění problému použít například **vývojový diagram**. Během návrhu a analýzy programu je nutná konzultace se zákazníkem a koncovými uživateli.
- **Implementace** – práce pro programátora, který na základě analýzy problému a vytvořených datových struktur (**databáze a Interface programu**) napíše program.
- **Testování** – Během tvorby programu a po jeho dokončení přechází program do fáze testování, které nejprve provádí programátor, tester a později i koncový uživatel ve zkušebním provozu. Většina programů projde **Alfa** verzí (**program je nehotov a obsahuje mnoho chyb**) a **Beta** verzí (**program je hotov, ale stále obsahuje chyby**). V případě že jsou chyby odstraněny vydá se konečná – výsledná verze.
- **Nasazení do provozu** – program se začne využívat v ostrém provozu. V případě komerčního - začne se prodávat.
- **Aktualizace** – Ostrý provoz zpravidla objeví další chyby, které odstraňují další aktualizace. Aktualizace zároveň rozšiřují funkce programu a nebo upravují program podle nových požadavků. Čísla aktualizací se číslují např. 1.2
- **Vydání nové verze** – při větších změnách se obvykle vydává nová verze, někdy je nová verze přepsána celá od základu, v případě že změny nelze na aplikaci podle původního návrhu provést a nebo pokud je program důsledkem špatného návrhu a nebo častých změn již nestabilní a neefektivní. Čísla verzí se číslují například 2.0.

Jak software vzniká – Vývojový diagram

- **Vývojový diagram** – zobrazí logickou strukturu programu a tím zpřehlední řešení problému



Ukázkový program: hra Lucky seven



Ukázková hra: Lucky seven – Form1

- **Formulář** – vytvořte v **Borland Delphi** nový projekt a uložte (menu **File / Save All**). Poté na něj **Form1** vložte 3 komponenty **tLabel** a 3 tlačítka **tButton** a jeden **tTimer**



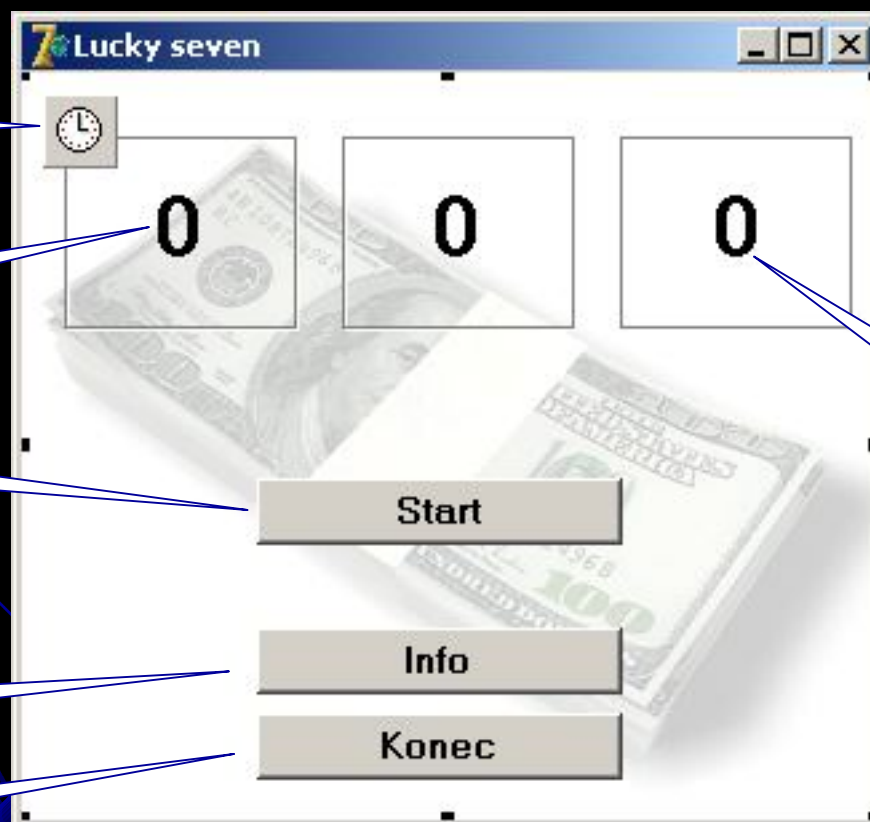
Timer1

Label1

Button1

Button2

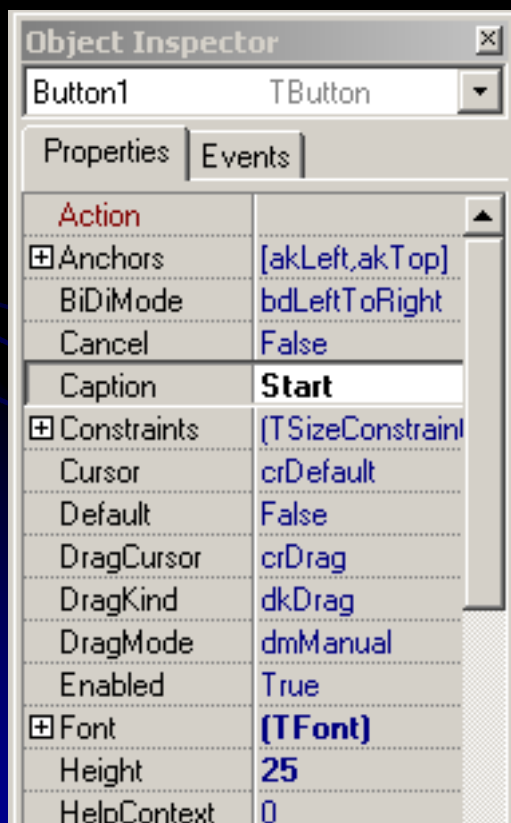
Button3



Label3

Ukázková hra: Lucky seven – Nastavení vlastnosti

- **Button1** nastavte v Object Inspector vlastnost **Caption** na hodnotu **Start**
- **Button2** nastavte v Object Inspector vlastnost **Caption** na hodnotu **Info**
- **Button3** nastavte v Object Inspector vlastnost **Caption** na hodnotu **Konec**
- **Timer1** nastavte v Object Inspector vlastnost **Enabled** na hodnotu **False**
- **Label1** až **Label3** nastavte v Object Inspector vlastnost **Caption** na hodnotu **0**



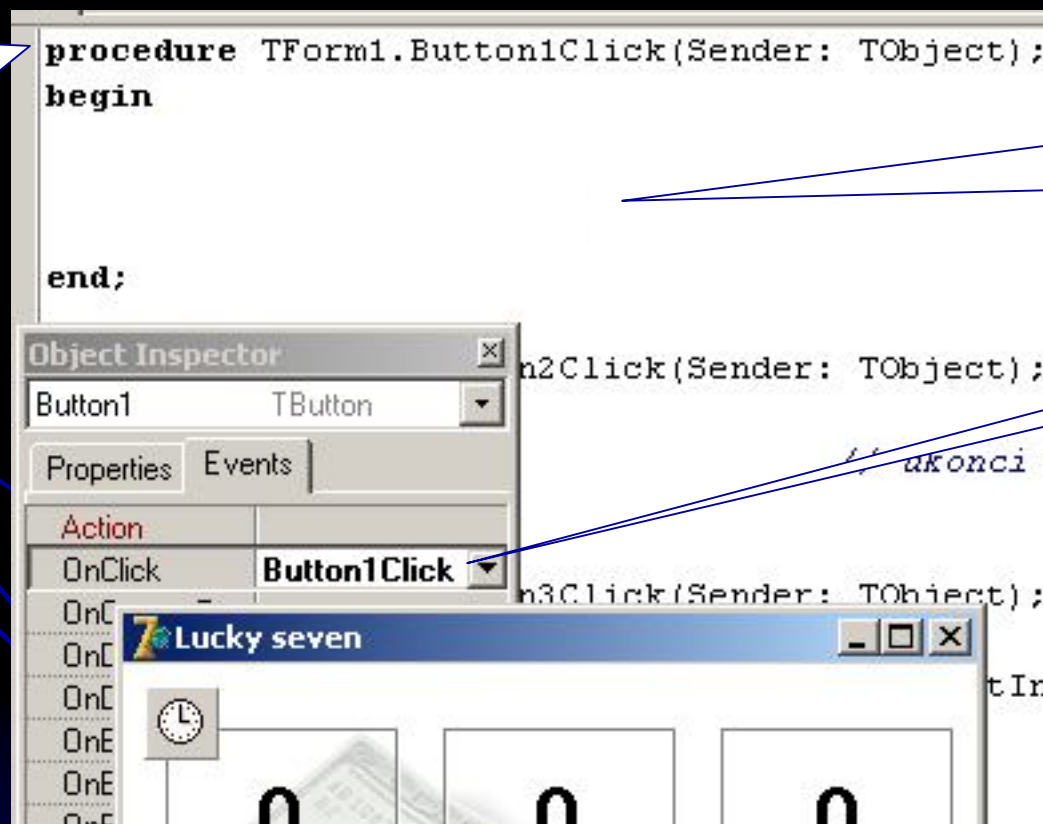
Vlastnost

Vybraný
objekt

Ukázková hra: Lucky seven – Nastavení události

- Vyberte **Button1** a v Object Inspector v událostech **dvakrát klikněte** na událost **OnClick**
- Vytvoří se **Procedure TForm1.Button1Click(Sender: TObject);** - do které napíšete kód co se provede když na tlačítko Button1 kliknete.

Procedura
tlačítka
Button1



Kód Procedury
se píše mezi
begin a end

Událost

Ukázková hra: Lucky seven – Pomocná proměnná i

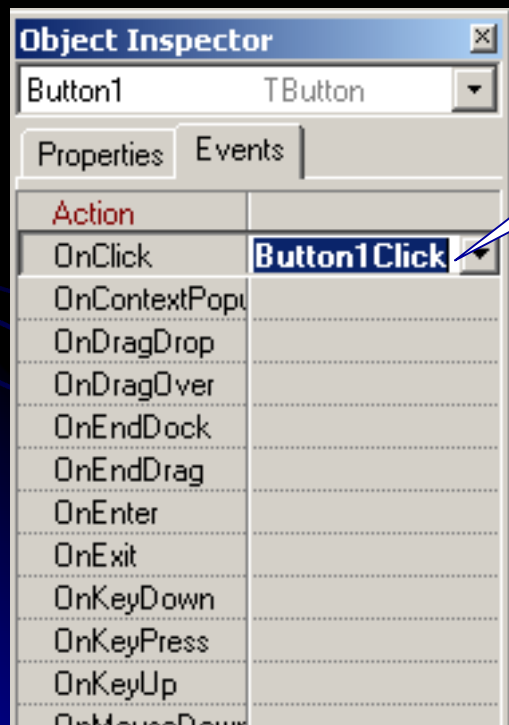
- **Proměnná i** - bude celočíselného typu **Integer**. Využije se později v Timer1 pro ukončení generování náhodných čísel.
- **Proměnnou je vždy třeba deklarovat**. Aby byla společná pro všechny procedury ve **Form1** vložíme ji do **Private**.

```
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Timer1: TTimer;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
  private
    { Private declarations }
    i: integer;
  public
    { Public declarations }
  end;
```

Deklarace
proměnné i v
private

Ukázková hra: Lucky seven – Obsluha tlačítka Button1

- Dvakrát klikněte na tlačítko **Button1** a nebo v Object Inspectoru na událost **Button1 onClick**
- V proceduře **onClick** zadejte tento kód, který **vynuluje proměnnou i**, nastaví **Interval Timer1** na 100 milisekund a spustí jej.



Událost
onClick

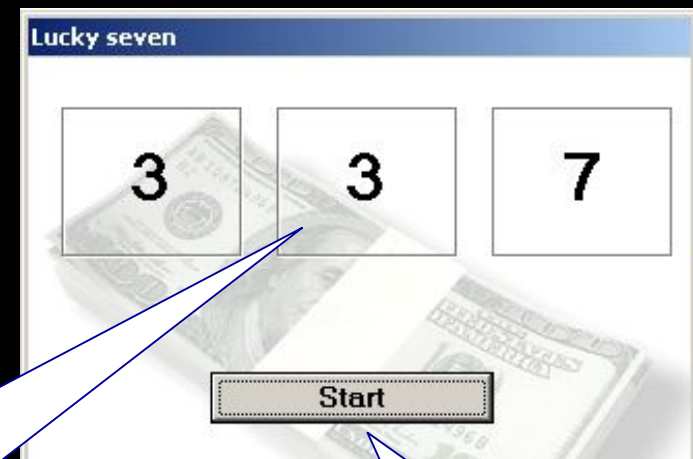
Kód se provede
při stisku
Button1

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    i:=0;  
    Timer1.Interval := 100;  
    Timer1.Enabled := true;  
end;
```

Ukázková hra: Lucky seven – Generování náh. čísla

- Stejně jako událost tlačítka **onClick** vytvořte u časovače **Timer1** událost **onTimer** a do procedury: `procedure TForm1.Timer1Timer(Sender: TObject);` zadejte následující kód...
- **Randomize;** - inicializuje generátor náhodných čísel
- **Label1.Caption :=** - nastaví vlastnost **Caption** na daný textový řetězec
- **IntToStr()** - převede číselnou hodnotu na textový řetězec
- **Random(7)** - vygeneruje náhodné celé číslo od 0 do 6
- **+ 1** - zvýší vygenerované číslo o 1

```
procedure TForm1.Timer1Timer(Sender: TObject);  
begin  
    randomize;  
    Label1.Caption := IntToStr(random(7) + 1);  
    Label2.Caption := IntToStr(random(7) + 1);  
    Label3.Caption := IntToStr(random(7) + 1);  
end;
```



Zde se bude každých 100 milisekund zobrazovat náhodné číslo mezi 1 až 7

Spustí Timer

Ukázková hra: Lucky seven – Detekování výhry

- Číslo se bude generovat 3 krát a poté se zjistí jestli je kombinace 7 7 7. Pokud ano, zobrazí se windows dialog !!!!! Výhra !!!!!
- Na konec procedure TForm1.Timer1Timer(Sender: TObject); přidejte kód:
- `i := i + 1;` - zvýší pomocnou proměnnou i o jedna
- `if i > 3 then` – co se stane když timer proběhne 3 krát
- `Timer1.Enabled := false;` – vypne Timer1

Dialog co se zobrazí při výhře



```
i := i + 1;  
  
if i > 3 then  
begin  
    Timer1.Enabled := false;  
    if (Label1.Caption = '7') and  
       (Label2.Caption = '7') and  
       (Label3.Caption = '7') then  
  
        MessageDlg(' !!!!! Výhra!!!! ', mtInformation, [mbOk], 0);  
end;
```

Podmínka co kontroluje zda-li Label1, Label2 a Label3 mají hodnoty 777. pokud ano, tak je výhra...

Ukázková hra: Lucky seven – Tlačítka Info a Konec

- Nyní již stačí přidat funkčnost tlačítku Button2 – Info a Button3 - Konec
- Vytvořte událost **onClick** tlačítka Button2 a událost **OnClick** tlačítka Button3. Do vytvořených procedur запиšte následující kód, který obslouží tlačítka Info a Konec.
- **form1.close;** - ukončí program, tím že zavře jeho hlavní formulář
- **MessageDlg()** – zobrazí hlášku systému stejně jako při výhře, v níž bude nápovědný text


Procedura události onClick

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    MessageDlg('Nápověda: vyhrává 7 7 7 !!!', mtInformation, [mbOK],0);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
    form1.Close;
end;
```

Konec programu

Ukázková hra: Lucky seven – Kompilace programu

- Hotový program je třeba **překompilovat** (přeložit zdrojový kód do binární podoby) do výsledného EXE souboru (project1.exe).
- Před kompilací program zkontroluje **debugger** (překlad: odvšivovač ☺), který **vyhledá chyby v kódu**. Pokud je v kódu chyba, tak na ni upozorní červeným řádkem a zaství překlad, dokud není chyba odstraněna.
- **Start překladu** – klávesa F9 a nebo toto tlačítko 

```
randomize;  
Label1.Caption := IntToStr(random(7) + 1);  
Label2.Caption := IntToStr(random(7) + 1);  
Label3.Caption := IntToStr(random(7) + 1);  
  
i := i + 1;
```

Chyba překladu, příkaz musí být ukončen středníkem ;

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
  i:=0;  
  Timer1.Interval := abc;  
  Timer1.Enabled := true;  
end;
```

Chyba překladu, je vyžadována celočíselná hodnota

Ukázková hra: Lucky seven – Program je spuštěn

- Pokud proběhne správně překlad, tak se spustí výsledný program a v adresáři kde je uložen se vytvoří spustitelný EXE soubor.
- Gratuluji právě jste naprogramovali hru Lucky seven !

