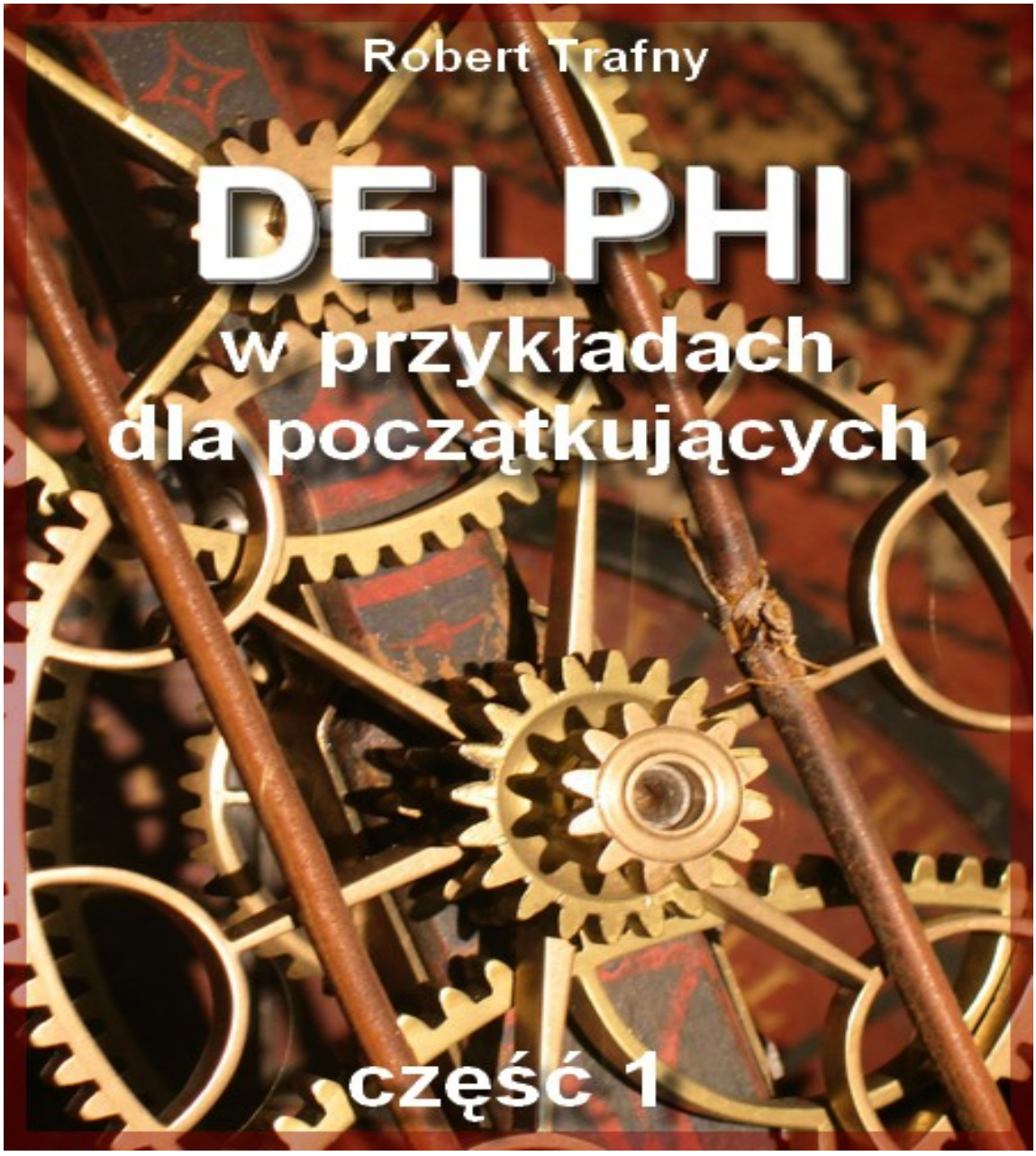
Robert Trafny

Delphi w przykładach

dla początkujących

część 1



edycja 2018

Tytuł: Delphi w przykładach dla początkujących część 1

Autor: Robert Trafny

Skład: Robert Trafny

Projekt okładki: Robert Trafny

Fotografia na okładce pochodzi z serwisu imageafter.com

ISBN: 978-83-950078-4-2

Wydanie I

Wszelkie uwagi proszę kierować na adres: robert-trafny@wp.pl

Spis treści:

Wstęp

Oddajemy do rąk Czytelnika pierwszą część książki poświęconej programowaniu w języku Delphi. Nie należy jej traktować jako podręcznika, ponieważ nią nie jest. Ideą przewodnią całego cyklu jest prezentacja możliwości Delphi na podstawie konkretnych rozwiązań. Poprzez nie Czytelnik zapoznaje się ze sztuką programowania.

Każdy zamieszczony w tej książce program jest szczegółowo opisany i skomentowany. W ten sposób Czytelnik sukcesywnie zapoznaje się z nowymi rozwiązaniami i poleceniami Delphi, które później będzie mógł wykorzystać w swoich projektach.

Do tej pory powstało wiele książek i artykułów na temat podstaw Delphi, także w internecie nie brakuje materiałów na ten temat. Niniejsza książka jest odpowiedzią na powstały niedosyt, jak owe teoretyczne wiadomości wykorzystać w praktyce. Wychodząc naprzeciw początkującym programistom, zawarliśmy w książce projekty ponad dwudziestu różnych programów komputerowych, jak najbardziej stroniąc od powtarzania podstawowej wiedzy na temat programowania, którą bez problemu można pozyskać, chociażby z internetu. Opierając się na tym założeniu, oczekujemy od Czytelnika posiadania podstawowej wiedzy na temat programowania w języku Delphi. Zakładamy więc, iż Czytelnik wie, w jaki sposób utworzyć nowy projekt programu, w jaki sposób umieścić potrzebne komponenty na formie głównej oraz, w jaki sposób zmieniać właściwości komponentów.

Większość zamieszczonych w książce projektów jest w formie surowej, bez estetycznych upiększeń. Naszym celem bowiem było przedstawienie działających rozwiązań, zadbanie o poprawną pracę programów. W ten sposób też każdy programista powinien pracować nad swoim projektem. Najpierw tworzymy surowy szkic programu, zabiegając wyłącznie o to, aby wszystkie jego procedury działały bez zarzutu. Na samym końcu dopiero zaczynamy zajmować się estetyką programu, nadając kolory, style itd. My pominęliśmy ten etap, wychodząc z założe-

nia, iż dla programisty liczyć się będzie tylko działające rozwiązanie. Z uwagi na to, iż o gustach się nie dyskutuje, przyjmujemy, że programista nada zaprezentowanym programom własną szatę graficzną, przystosowaną do indywidualnej wrażliwości estetycznej. Jako przykład poglądowy, w rozdziale pt. „Hot Spot" zamieszczamy w pełni opracowany projekt, także pod względem szaty graficznej, aby przedstawić jedną z propozycji zadbania o wygląd naszego programu.

Książka podzielona jest na dwa główne działy: Gry i zabawy oraz Programy użytkowe. Nie chcemy w naszym cyklu ograniczać Delphi tylko do programów użytkowych. Programowanie daje znacznie większe możliwości i nie chcemy mentalności początkującego programisty zamykać w tym ograniczonym wyobrażeniu. Zgodnie z tym założeniem w następnych częściach pojawią się kolejne działy, jak chociażby budowa robota czy androida, które odsłonią pełną ofertę możliwości, jaką daje programowanie w języku Delphi. Nabywając w ten sposób szerszego spojrzenia na temat programowania, staniemy się profesjonalnymi programistami, którzy nie będą bali się wyzwań. Tylko na tej podstawie możliwe są wszelkie innowacje i rewolucje technologiczne. Ktoś, kto boi się poszukiwać i z góry zakłada, że to się nie uda, nigdy nie stanie się autorem czegoś przełomowego. Jako przykład – choć może z nieco innej dziedziny – przywołamy Alberta Einsteina, który zapytany o to, w jaki sposób wpadł na pomysł swojej teorii względności, odpowiedział krótko, że nie bał się myśleć. Nie bójmy się zatem i my. Zastanawiajmy się, poszukujmy i pytajmy, jeżeli czegoś nie wiemy, bo tylko w ten sposób możemy się rozwijać, także jako programiści.

Mam nadzieję, że niniejsza książka będzie inspiracją dla Czytelników i zachętą do rozwijania własnych projektów. Tego wszystkim życzę z całego serca.

Robert Trafny

Gry i zabawy

(...)

Powtórz litery

Przedstawiona tu zabawa w założeniach podobna jest do tej, którą prezentowaliśmy poprzednio. W obu chodzi o to, aby powtórzyć ciąg znaków, jakie program dla nas wygeneruje. Pomimo tego podobieństwa, użycie liter jako znaków do odgadnięcia implikuje użycie bardziej skomplikowanej formy kodu. Uwarunkowane jest to naturą tychże znaków. W przypadku poprzedniej zabawy znakami tymi były liczby, więc program – jako maszyna matematyczna – nie miał problemu z ich obróbką. Wystarczyło wylosować jedną liczbę pięcioznakową, aby spełnić nasze oczekiwanie. W sytuacji, gdy mamy do czynienia z literami (a więc znakami niematematycznymi), program nie jest w stanie sam od siebie niczego wygenerować. Musimy mu z poziomu kodu zaimplementować jakiś zbiór znaków (w tym przypadku liter), aby mógł na nim dokonywać działań. Poprzez swoje funkcje Delphi zaczyna traktować te znaki w sposób matematyczny, pamiętając jednak, że są to znaki typu string.

Oto peny kod zabawy:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
```

```
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: Tobject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  WYNIK: String;              //zmienna przechowująca litery do odgadnięcia
  Poprawne: Integer;          //zmienna przechowująca ilość poprawnych odpowiedzi
  Niepoprawne: Integer;       //zmienna przechowująca ilość niepoprawnych odpowiedzi

implementation

{$R *.dfm}


                //przy zamykaniu programu oceń uzyskane wyniki

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
 begin       //jeżeli uzyskałeś więcej złych odpowiedzi lub tyle samo co dobrych wtedy...
    if (Poprawne<Niepoprawne) or (Poprawne=Niepoprawne) then
      begin
        ShowMessage('Musisz jeszcze trochę potrenować');   //wyświetl komunikat
      end                                                  //bez średnika
        else                                             //w przeciwnym wypadku...
          ShowMessage('Gratuluję! Uzyskałeś dobry wynik');       //wyświetl komunikat
 end;
```

```
                                    //gdy naciśniesz przycisk START

procedure TForm1.Button1Click(Sender: TObject);
 const
   Tablica: array [1..23] of String =('a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','r','s',
                                      't','u','w','y','z');                    //tablica liter

 var
  Litera1: String;        //zmienna przechowująca pierwszą literę
  Litera2: String;        //zmienna przechowująca drugą literę
  Litera3: String;        //zmienna przechowująca trzecią literę
  Litera4: String;        //zmienna przechowująca czwartą literę
  Litera5: String;        //zmienna przechowująca piątą literę
  Litera6: String;        //zmienna przechowująca szóstą literę

 begin
      Edit1.Clear;                 //wyczyść pole edycyjne (Do podawania Odpowiedzi)
      Randomize;                             //uruchom maszynę losującą
   Litera1:= Tablica[Random(23)+1];      //wylosuj 1 literę
   Litera2:= Tablica[Random(23)+1];      //wylosuj 2 literę
   Litera3:= Tablica[Random(23)+1];      //wylosuj 3 literę
   Litera4:= Tablica[Random(23)+1];      //wylosuj 4 literę
   Litera5:= Tablica[Random(23)+1];      //wylosuj 5 literę
   Litera6:= Tablica[Random(23)+1];      //wylosuj 6 literę
      WYNIK:= (Litera1+Litera2+Litera3+Litera4+Litera5+Litera6);
                                  //zapisz wylosowane litery do zmiennej globalnej

  Label2.Caption:=WYNIK;                //wyświetl liczbę do powtórzenia
  Label2.Update;                        //uaktualnij komponent label
     Button2.Enabled:=false;            //zablokuj przycisk Sprawdź
     Sleep(3000);                       //czekaj 3 sekundy
  Label2.Caption:='powtórz';            //zamiast liczby wyświetl napis 'powtórz'
      Button2.Enabled:=true;            //odblokuj przycisk Sprawdź
  Edit1.SetFocus;                       //ustaw focus w Polu do Odpowiedzi
 end;


                          //gdy naciśniesz przycisk Sprawdź

procedure TForm1.Button2Click(Sender: TObject);
 var
  Odp: String;              //zmienna przechowująca naszą odpowiedź

 begin
   Odp:= Edit1.Text;        //odczytaj podaną odpowiedź i przypisz ją do zmiennej

  if Odp=WYNIK then                     //jeżeli odpowiedź jest poprawna to ...
    begin
```

```
      ShowMessage('Bardzo dobrze!');  //wyświetl komunikat o poprawnej odpowiedzi
       Label2.Caption:= '------';      //wyczyść działanie
      Poprawne:=Poprawne + 1;          //dodaj punkt do poprawnych odpowiedzi
    end                                //bez średnika

      else                            //w przeciwnym wypadku ...

        begin
          ShowMessage('Niestety źle');       //wyświetl komunikat o złej odpowiedzi
     Niepoprawne:=Niepoprawne + 1;  //dodaj punkt do niepoprawnych odpowiedzi
    end;

  Edit1.Clear;                          //wyczyść pole do podawania Odpowiedzi)
  Edit1.SetFocus;                         //ustaw focus w Polu do Odpowiedzi

  Label6.Caption:= IntToStr(Poprawne);     //wyświetl ilość poprawnych odpowiedzi
  Label7.Caption:= IntToStr(Niepoprawne);  //wyświetl ilość niepoprawnych odpowiedzi
 end;
end.
```

Komentarz:

1. Wstęp

Jak widzimy, komponenty użyte w tej zabawie pozostają te same, co w poprzedniej. Tutaj nic się nie zmienia. Zmianie ulegnie tylko część kodu obsługująca działanie programu. Przede wszystkim nie wprowadzamy procedury **Edit1KeyPress** ograniczającej rodzaj wpisywanych znaków w pole edycyjne przeznaczone do podania odpowiedzi. Musieliśmy usunąć to ograniczenie, aby można było wpisywać litery. Pozostałe rozwiązania pozostają niezmienione, aż do procedury obsługującej przycisk START.

2. *Gdy naciśniesz przycisk START*

Jak już wspominaliśmy, program jako maszyna matematyczna nie zna liter, musimy je dopiero zadeklarować. Wykorzystujemy do tego celu dwa nowe dla nas elementy: stałą i tablicę. Do

tej pory nauczyliśmy się stosować zmienne jako miejsce przechowywania danych, które często się zmieniają. Tutaj sytuację mamy inną: z góry wiemy, jakie litery będą używane. Będzie to zatem stały element.

Konkretne litery, po wylosowaniu w dalszej części procedury zostaną przypisane do zmiennych Litera, w których to program będzie je przechowywał podobnie, jak robił to wcześniej z liczbami. Sekcja ta jest zatem bazą, z której program będzie czerpał litery. Pozostaje jeszcze kwestia ich implementacji. Moglibyśmy to zrobić analogicznie jak w przypadku zmiennych, czyli każdą z liter deklarując osobno, na przykład tak:

```
const
  Znak1:= 'a';
  Znak2:= 'b';
  Znak3:= 'c';
      (...)
  Znak23:= 'z';
```

Powyższe rozwiązanie jak widzimy, generuje znaczny przyrost kodu i stratę naszego czasu. O wiele szybciej i prościej cały ten zbiór możemy zawrzeć w tablicy stałych:

```
const
  Tablica: array [1..23] of String =('a','b','c','d','e','f','g','h',
    'i','j','k','l','m','n','o','p','r','s','t','u','w','y','z');          //tablica liter
```

Zamiast znacznika **var** (od angielskiej nazwy Zmienna) używamy tu znacznika **const**, czyli stała. W dalszej kolejności wprowadzamy nazwę tablicy. Nazwa może być dowolna, pamiętajmy jednak, że musi być ona jednoczłonowa. My przyjęliśmy prostą nazwę, czyli „Tablica”. Teraz musimy zadeklarować z ilu i z jakich elementów będzie się składała oraz po znaku równości, w nawiasie wymieniamy je, każdy element umieszczając w apostrofach i oddzielając go od innych elementów za pomocą przecinka. W naszym przypadku mamy 23 elementy (litery) typu string.

Chcąc skorzystać w procedurze z tak przygotowanego

zbioru, używamy nazwy tablicy i numeru elementu, który chcemy użyć, przykładowo, chcąc użyć litery „c” odwołujemy się do niej w następujący sposób: Tablica[3]. Litera „c” jest trzecim elementem tego zbioru, więc i taką matematyczną nazwę będzie w tym przypadku posiadała.

Aby móc używać liter z naszej tablicy w tworzonej przez nas procedurze, musimy zadeklarować sześć zmiennych typu string (Litera1...6) do ich przechowywania. Ilość zmiennych uzależniona jest od ilości liter, jakie chcemy powtarzać. Przyjęliśmy, że powtórzenie sześciu liter z pamięci jest wystarczająco trudnym zadaniem, stąd też tyle zmiennych przygotowujemy.

W rozpoczynającej się procedurze następuje wylosowanie jednego z elementów tablicy liter i przypisanie go do zmiennej Litera:

```
Litera1:= Tablica[Random(23)+1].
```

Każda ze zmiennych Litera przechowuje jedną literę do powtórzenia, wylosowaną z tego samego zbioru, czyli Tablica. Istnieje więc możliwość, że wśród tych liter niektóre mogą się powtarzać.

```
WYNIK:= (Litera1+Litera2+Litera3+Litera4+Litera5+Litera6);
```

Po przypisaniu liter do zmiennych następuje połączenie ich w jeden ciąg znaków i zapisanie do zmiennej globalnej WYNIK. Działanie to wymaga od nas, aby i my podali odpowiedź, wpisując jedna litera za drugą, bez oddzielania ich za pomocą spacji. Warto zwrócić uwagę na fakt, że litery jako znaki tekstowe nie wymagają od zmiennej WYNIK żadnej konwersji.

Dalsza część procedury, jak i dalszy kod pozostają bez zmian w stosunku do pierwowzoru, więc pomijamy ją w tym komentarzu. Zwracamy tylko uwagę, aby zmienną WYNIK nie konwertować przy sprawdzaniu odpowiedzi, gdyż przecież jest ona w tym projekcie zmienną typu string, a więc tego samego typu, co zmienna Odp.

W trzecim przykładzie zabawy z tej grupy napiszemy program łączący obie wcześniejsze zabawy, czyli będzie losował dla nas zarówno litery, jak i liczby.

Hot Spot

Na zakończenie działu poświęconego grom i zabawom, prezentujemy w pełni wykończony projekt, aby pokazać, jak odpowiednie zadbanie o estetykę interfejsu wpływa na atrakcyjność programu. Jako przykład wybraliśmy symulator automatu do gier. Będzie on oparty w większości na tych rozwiązaniach, które poznaliśmy do tej pory, choć nie brakuje też i nowych elementów.

Założenia i opis programu:

Naszym głównym założeniem jest chęć wykonania typowego automatu do gier, który będzie obracał bębnami i przyznawał punkty za trafiony układ. Wzorując się na oryginale, nie zabraknie także możliwości zmieniania stawki, za jaką w danej chwili chcemy zagrać oraz liczników obrazujących, ile otrzymaliśmy punktów za układ i jaki jest stan naszego konta.

Oczywiste jest, że nie zrobimy automatu na prawdziwe pieniądze, ale – nawiązując do prawdziwych automatów – będziemy posiadać przycisk, który będzie zasilać nasze konto w grze. Zachowując ten realizm, będziemy mieli rozeznanie, jak łatwo można stracić pieniądze, grając w prawdziwe automaty do gier.

Jako że jesteśmy twórcami naszego symulatora, nie musimy ograniczać się tylko do zgodności z oryginałem, możemy wprowadzić swoje dodatkowe założenia. Najważniejszym z nich będzie możliwość grania na debecie: Kiedy skończą się nam fundusze na koncie, będziemy mogli dalej grać, zaciągając pożyczkę. Stan naszego konta będzie wtedy ujemny do czasu, aż trafimy układ i program odliczy sobie przyznaną pożyczkę. Aby nie popaść w nadmierny hazard, założymy blokadę, abyśmy mogli zadłużać się tylko do kwoty 200 złotych. Powyżej tej

granicy program uniemożliwi dalszą grę.

Wprowadzamy także bonusy za ustawienie niektórych symboli w określonym miejscu, mianowicie, będą nimi pola narożne oraz środkowe. Za każde takie trafienie będą przyznawane punkty, lecz nie o ściśle wyznaczonej wartości, tylko o wartości zmiennej, którą program przydzieli. Chcemy także, aby po umieszczeniu tych symboli we wszystkich tych miejscach naraz, program zachował się w sposób niezwykły, mianowicie, chcemy aby odwrócił nasz los o 180 stopni w ten sposób, że gdy będziemy mieli debet, anuluje go nam, natomiast w przypadku posiadania przez nas dodatniego stanu konta, po prostu wyzeruje go, wyświetlając napis „Bankrut”.

Aby ułatwić sobie zadanie, rezygnujemy z obrazków owców widniejących na typowym automacie. Zamiast nich wyświetlać będziemy liczby, od zera do dziewięciu. Przyjmujemy, że elementem odpowiedzialnym za uzyskanie bonusu będzie cyfra zero. Wybieramy także najtrudniejszą wersję gry, czyli pięć bębnów. Za układy będziemy uważać te grupy liczb (minimum trzy), które są obok siebie w linii prostej: poziomej, pionowej lub ukośnej; są tego samego typu i mają swój początek w pierwszym lub ostatnim bębnie.

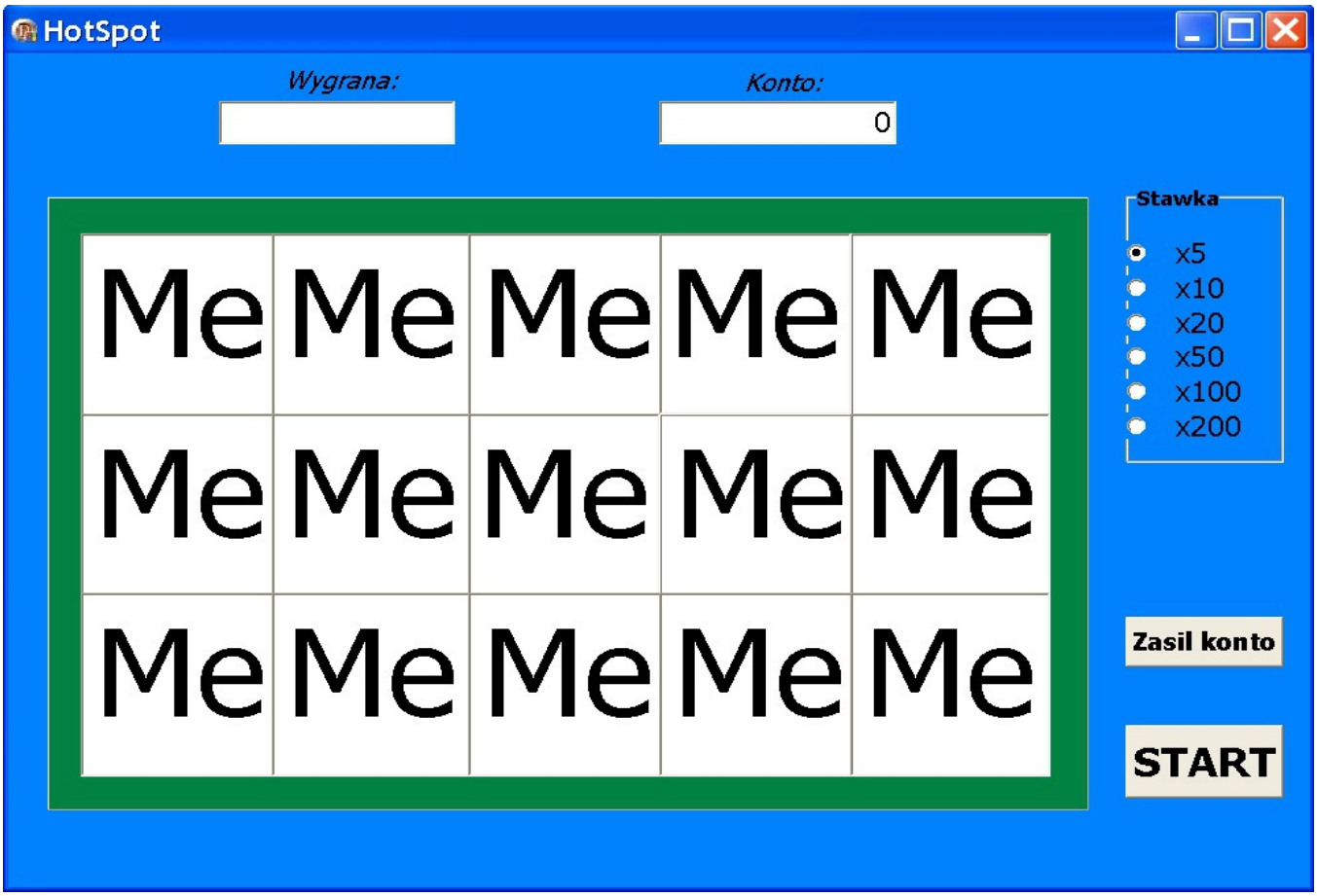
Zasada działania programu jest wbrew pozorom bardzo prosta. Po naciśnięciu przycisku „START” program losuje i wyświetla liczby na poszczególnych polach bębnów, a następnie sprawdza, czy ułożyły się jakieś układy. Jeżeli powstał układ lub bonus, przydziela za niego punkty pomnożone przez wybraną przez nas stawkę i wyświetla ją w polu wygranych. Jednocześnie, uzyskaną liczbę punktów dodaje także do naszego konta. Dla ułatwienia odczytywania układów program będzie zaznaczał układy na czerwono, a bonusy na zielono.

Potrzebne komponenty:

W tym projekcie używamy dużo komponentów. Wszystkie wstawiamy na formę główną. Oto wykaz potrzebnych komponentów:

Button1 – przycisk „Zasil konto”,
Button2 – przycisk „START”,
Panel1 – Ramka, na której znajdują się komponenty Memo,
Memo1..15 – Pola bębnów (3x5),
Edit1 – Pole do wyświetlania liczby punktów,
Edit2 – Pole do wyświetlania stanu konta,
Label1 – etykieta wyświetlająca napis „Wygrana”,
Label2 – etykieta wyświetlająca napis „Konto”,
RadioGroup1 – ramka do grupowania kontrolek RadioButton,
RadioButton1..6 – kontrolki do wybierania stawki.

Poszczególne komponenty rozmieszczamy i kształtujemy według poniższego wzoru:



Rozkład komponentów Memo jest następujący:

Memo: 1 2 3 4 5
6 7 8 9 10
11 12 13 14 15

A oto pełny kod programu:

(...)

Programy użytkowe

Część poświęconą programom użytkowym rozpoczynamy od kilku programów zajmujących się czasem. Na początek przedstawimy program, który na nasze życzenie będzie informował w ludzkim języku, która jest aktualnie godzina, jaki jest dzisiaj dzień oraz, który to dzień i miesiąc roku. Na bazie rozwiązań wykorzystanych w tym projekcie, utworzymy kolejny program, który będzie informował nas, czyje są dzisiaj imieniny. Idąc za ciosem, w trzecim projekcie napiszemy prosty program w formie przypominajki. Będzie on kontrolował datę i informował nas o zbliżających się zdarzeniach, które wcześniej ustaliliśmy.

Centrum czasowe

Założenia i opis programu:

Program składa się z trzech oddzielnych rozwiązań i procedur, które mogą funkcjonować zupełnie niezależnie od siebie. Umieszczamy je we wspólnym projekcie, ponieważ należą do wspólnej grupy tematycznej, stąd nazwa: Centrum czasowe. Nic nie stoi na przeszkodzie, aby w skład tego projektu nie włączyć

kolejnego projektu pod nazwą „Czyje są dzisiaj imieniny”. Wystarczy tylko umieścić na formie kolejny przycisk i przenieść kod, który go obsługuje.

Centrum czasowe, które prezentujemy, jest częścią większego projektu o nazwie „Budowa androida”, stąd też w rozwiązaniu tym informacje przekazywane przez program są wypowiadane ludzką mową. W zależności od naszych założeń możemy te słowne komunikaty zamienić na ich wersje tekstowe, stosując znaną nam już formę komunikatu, czyli ShowMessage.

Zanim zajmiemy się kodem, musimy przygotować odpowiednie pliki dźwiękowe, z których program będzie korzystał. Warto większość z nich umieścić we wspólnym katalogu o nazwie WSPÓLNE. Zauważmy, że część plików będziemy mieli już przygotowanych, gdyż stworzyliśmy je na potrzeby zabawy pt. „Policz, ile to jest – wersja audio”. Możemy je teraz wykorzystać.

Jakie pliki będziemy potrzebować? Dla opcji „Która jest godzina” potrzebujemy liczb, które je określają. Możemy je przygotować w dwojaki sposób: Pierwsza możliwość to przygotować oddzielne pliki dla każdej minuty i godziny dnia, czyli na przykład 20.01, 20.02, 20.03 itd. Jak łatwo policzyć, musielibyśmy przygotować 1440 plików (24 godziny x 60 minut). Możemy jednak uprościć sobie zadanie, rozbijając aktualną godzinę na dwa osobne pliki: godziny oraz minuty, i w tej formie przygotować pliki dźwiękowe. Mamy więc 24 pliki godzinowe: pierwsza, druga, trzecia... oraz 15 plików minutowych, które rozbiliśmy na kolejne dwa osobne pliki: dziesiątki i minuty. Dzięki takiemu rozwiązaniu nie musimy przygotowywać 60 plików dla każdej minuty godziny, czyli 0, 1, 2, 3...58, 59. W tym rozwiązaniu stosujemy następujące pliki:

dziesiątki: 10, 20, 30, 40, 50,
minuty: 0, 1, 2, 3...9.

Z tych trzech części program będzie składał godzinę według następującego schematu:

(Jest godzina) (plik godzinowy) (plik dziesiątek) (plik minutowy),

czyli na przykład: *(Jest godzina) (dwudziesta) (pięćdziesiąt) (osiem).*

Powyższe rozwiązanie wymaga od nas większego zaangażowania programistycznego, musimy stworzyć więcej instrukcji do wykonania, ale dzięki temu oszczędzamy znacznie więcej czasu na przygotowywaniu plików dźwiękowych. Przygotowanie półtora tysiąca plików dźwiękowych to praca na kilka dni.

Oprócz liczb dla opcji „Która jest godzina" potrzebujemy jeszcze początku wypowiedzi, czyli „Jest godzina...". W tym miejscu wprowadzamy oczekiwanie, aby program miał przygotowanych kilka wariantów tej wypowiedzi i za każdym razem losował jedną z nich. Program nie musi przecież za każdym razem mówić (aż do znudzenia) „Jest godzina...", może także użyć wypowiedzi wariantywnych, na przykład: „Teraz jest godzina...", „Teraz mamy...", „Jest...", „Na moim zegarku jest godzina..." itp. Nada to nieco żywości programowi. Przygotowane pliki umieszczamy w katalogu głównym programu lub jakimkolwiek innym.

Opcja „Dzisiaj jest" będzie wymagała od nas przygotowania 365 osobnych plików, dla każdego dnia roku, choć – podobnie jak to zrobiliśmy dla opcji „Która jest godzina" – możemy zaoszczędzić czas, stosując tamto rozwiązanie. W takim wypadku musielibyśmy przygotować następujące pliki:

(pierwszy, drugi, trzeci..., dziewiąty),
(dziesiąty, dwudziesty, trzydziesty)
oraz 12 plików z nazwami miesięcy.

Odpowiednio skonstruowany kod będzie składał odpowiedź z tych czterech fragmentów: (Dzisiaj jest) (plik dziesiątek) (plik jednostek) (plik miesiąca).

Dla opcji „Jaki dzisiaj dzień" potrzebujemy tylko siedmiu plików z nazwami dni. W plikach tych podobnie jak to uczyniliśmy wcześniej, zawrzemy od razu początek wypowiedzi, czyli – w pierwszym przypadku – np. „Dzisiaj jest dwudziesty pierwszy dzień października", a w drugim: „Dzisiaj mamy wtorek". Zauważmy, że nazwy plików z dniami tygodnia muszą odpowiadać nazwom z tablicy „Dni", czyli „Ni", „Po", „Wt" itd. W takiej formie

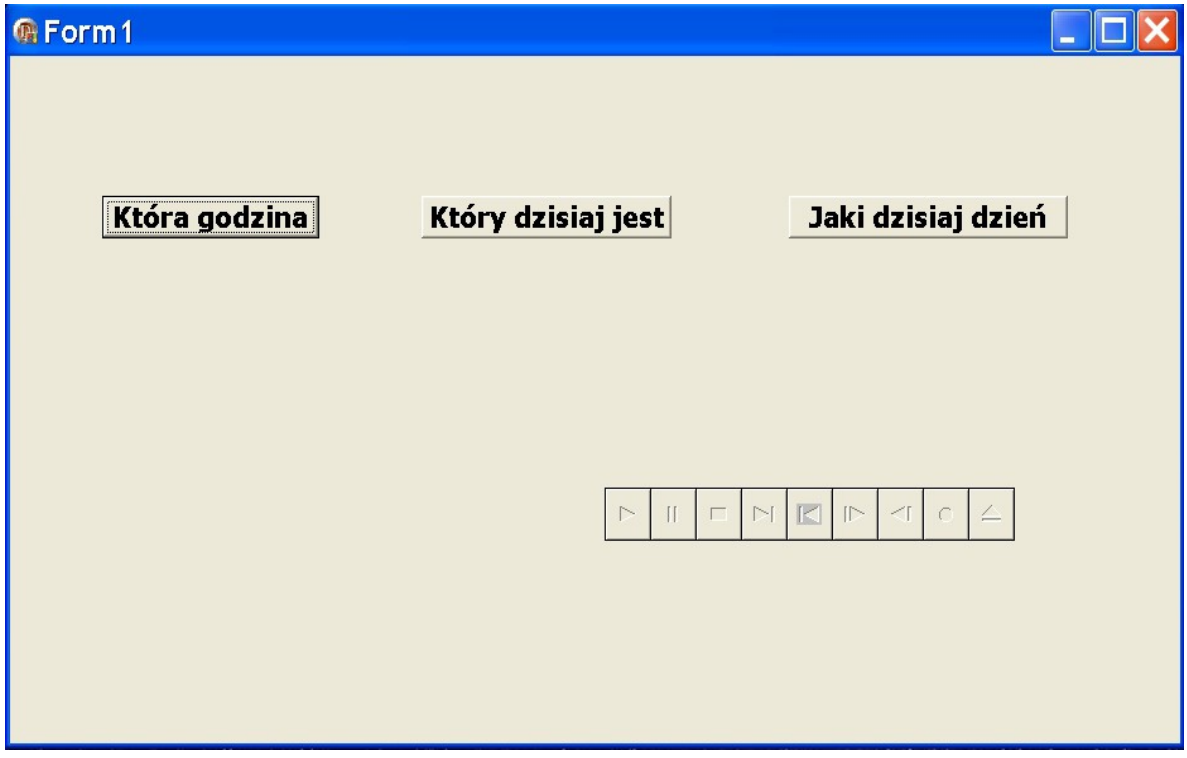
są wybierane do odtworzenia, więc i taką nazwę muszą posiadać.

W przypadku miesięcy, nazwy plików muszą być ich liczbowymi odpowiednikami, czyli dla dnia „dziesiąty stycznia” będzie to nazwa 1001, a dla „trzynasty listopada” – 1311. Zwracamy uwagę, że dla wypowiedzi typu „pierwszy”, „drugi”, „trzeci” stosujemy nazwy liczbowe „1”, „2”, „3”, więc nie mogą znaleźć się one razem z wypowiedziami typu „jeden”, „dwa”, „trzy”, gdyż mają one te same nazwy. Musimy zatem umieścić je w katalogu WSPÓLNE, ale w jakimś innym jego podkatalogu.

Potrzebne komponenty:

Button1 – przycisk „Która godzina”,
Button2 – przycisk „Który dzisiaj jest”,
Button3 – przycisk „Jaki dzisiaj dzień”,
MediaPlayer1 – odtwarzacz multimedialny.

Powyższe komponenty wstawiamy na formę główną naszego projektu i nadajemy im nazwy według rysunku poniżej:



A oto pełny kod programu:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, MmSystem, StdCtrls, MPlayer;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    MediaPlayer1: TMediaPlayer;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: Tobject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}


                              //przycisk Która godzina
procedure TForm1.Button1Click(Sender: TObject);
 var
    Czas: String;             //zmienna przechowująca aktualną godzinę
      godzina1: String;       //zmienna przechowująca pierwszą cyfrę godziny
      godzina2: String;       //zmienna przechowująca drugą cyfrę godziny
    minuta1: String;          //zmienna przechowująca pierwszą cyfrę minuty
    minuta2: String;          //zmienna przechowująca drugą cyfrę minuty
      Sciezka1: String;       //zmienna przechowująca ścieżkę dostępu do 'Jest godzina...'
      Sciezka2: String;       //zmienna przechowująca ścieżkę dostępu do liczb dziesiątek
      Sciezka3: String;       //zmienna przechowująca ścieżkę dostępu do liczb jednostek
    Plik: Integer;            //zmienna przechowująca plik 'Jest godzina...'
```

```
begin
  Czas:= TimeToStr(Time);           //odczytaj aktualną godzinę

  godzina1:= Czas[1];               //odczytaj pierwszą cyfrę godziny
   if godzina1=IntToStr(0) then     //jeżeli pierwsza cyfra to zero, to...
      godzina1:='';                 //...zmienna godzina1 nie ma wartości

  godzina2:= Czas[2];               //odczytaj drugą cyfrę godziny

  minuta1:= Czas[4];                //odczytaj pierwszą cyfrę minuty
   if minuta1=IntToStr(0) then      //jeżeli pierwsza cyfra to zero, to...
      minuta1:='';                  //...zmienna minuta1 nie ma wartości

  minuta2:= Czas[5];                //odczytaj drugą cyfrę minuty

    Randomize;
    Plik:= Random(8);               //losuj plik Jest godzina...
 Sciezka1:='C:\Aplikacje\Centrum czasowe\Która godzina\'+IntToStr(Plik)+'.wav';
                                    //ścieżka dostępu do Jest godzina...
 Sciezka2:= 'c:\Gry\WSPOLNE\Liczby i znaki\Dziesiątki\' +godzina1+godzina2+ '.wav';
                                    //ścieżka dostępu do liczb dziesiątek
 Sciezka3:= 'c:\Gry\WSPOLNE\Liczby i znaki\'+minuta1+minuta2+'.wav';
                                    //ścieżka dostępu do liczb jednostek
   PlaySound(PChar(Sciezka1),0,SND_FILENAME);     //odtwórz Jest godzina...
   PlaySound(PChar(Sciezka2),0,SND_FILENAME);     //odtwórz pełną godzinę
   PlaySound(PChar(Sciezka3),0,SND_FILENAME);     //odtwórz minuty
end;

                          //przycisk Który dzisiaj jest
procedure TForm1.Button2Click(Sender: TObject);
 var
  Data: String;                     //zmienna przechowująca aktualną datę
   dzien1: String;                  //zmienna przechowująca pierwszą cyfrę dnia
   dzien2: String;                  //zmienna przechowująca drugą cyfrę dnia
  miesiac1: String;                 //zmienna przechowująca pierwszą cyfrę miesiąca
  miesiac2: String;                 //zmienna przechowująca drugą cyfrę miesiąca

 begin
  Data:= (DateToStr(Date));         //odczytaj aktualną datę

  dzien1:= Data[9];                 //odczytaj pierwszą cyfrę dnia
  dzien2:= Data[10];                //odczytaj drugą cyfrę dnia
  miesiac1:= Data[6];               //odczytaj pierwszą cyfrę miesiąca
  miesiac2:= Data[7];               //odczytaj drugą cyfrę miesiąca

  MediaPlayer1.FileName:='C:\Aplikacje\Centrum czasowe\Który dzisiaj jest\'+
     dzien1+dzien2+miesiac1+miesiac2+'.mp3';
```

```
   MediaPlayer1.Open;            //otwórz plik
   MediaPlayer1.Play;            //odtwórz plik
end;


                          //przycisk Jaki dzisiaj dzień
procedure TForm1.Button3Click(Sender: TObject);
const
  Dni: array[1..7] of String= (('Ni '),('Po '),('Wt '),('Śr '),('Cz '),('Pi '),('So '));
                                                    //tablica z nazwami dni
var
  Data: String;                 //zmienna przechowująca aktualny dzień tygodnia
  dzien1: String;               //zmienna przechowująca pierwszą literę dnia
  dzien2: String;               //zmienna przechowująca drugą literę dnia

begin
   Data:=Dni[DayOfWeek(Now)];      //odczytaj aktualny dzień tygodnia z tablicy

  dzien1:=Data[1];                  //odczytaj pierwszą literę dnia z tablicy
  dzien2:=Data[2];                  //odczytaj drugą literę dnia z tablicy
 MediaPlayer1.FileName:='C:\Aplikacje\Centrum czasowe\Jaki dzisiaj dzień\'+
dzien1+dzien2+'.mp3';
  MediaPlayer1.Open;               //otwórz plik z nazwą dnia
  MediaPlayer1.Play;               //odtwórz plik z nazwą dnia
end;

end.
```

Komentarz:

1.Przycisk „Która godzina”

```
var
  Czas: String;           //zmienna przechowująca aktualną godzinę
    godzina1: String;     //zmienna przechowująca pierwszą cyfrę godziny
    godzina2: String;     //zmienna przechowująca drugą cyfrę godziny
  minuta1: String;        //zmienna przechowująca pierwszą cyfrę minuty
  minuta2: String;        //zmienna przechowująca drugą cyfrę minuty
    Sciezka1: String;     //zmienna przechowująca ścieżkę dostępu do 'Jest godzina...'
    Sciezka2: String;     //zmienna przechowująca ścieżkę dostępu do liczb dziesiątek
    Sciezka3: String;     //zmienna przechowująca ścieżkę dostępu do liczb jednostek
  Plik: Integer;          //zmienna przechowująca plik 'Jest godzina...'
```

Do wypowiadania informacji o aktualnej godzinie wykorzystamy funkcję PlaySound, gdyż będziemy łączyć ze sobą kilka

różnych elementów wypowiedzi według następującego schematu: (Jest godzina) (pełna godzina) (minuty). Nie zapominajmy o tym, że chcąc użyć odtwarzacza PlaySound, musimy wpierw wpisać w sekcji **uses** (na samym początku kodu) moduł odpowiedzialny za to, czyli MmSystem.

Użycie funkcji PlaySound implikuje użycie wielu zmiennych: musimy mieć trzy zmienne do przechowywania ścieżki dostępu do określonych plików (Sciezka1, 2, 3) oraz zmienną „Plik" do losowania wypowiedzi wariantywnej, jedną z wielu dla początku wypowiedzi, czyli „Jest godzina..."

Musimy także przygotować cztery zmienne do przechowywania poszczególnych cyfr godziny i minut, które zostaną przez program zdekodowane.

Analiza czasu w Delphi:

Aby odczytać aktualną godzinę, zastosujemy jedno z możliwych rozwiązań:

```
Czas:= TimeToStr(Time);          //odczytaj aktualną godzinę
```

Jak widzimy powyżej, jest to rodzaj konwersji czasu do postaci tekstowej, czyli string, zapisanej do zmiennej „Czas". Wyświetla ona godzinę w następującym formacie: gg:mm:ss, czyli dwie cyfry godziny, dwukropek, dwie cyfry minut, dwukropek i dwie cyfry sekund, razem 8 znaków. Chcąc teraz odwołać się do tych znaków, stosujemy zwyczajowy zabieg, czyli: nazwa zmiennej i numer znaku. W ten sposób do zmiennych „godzina1,2" i „minuta1,2" zapisujemy poszczególne cyfry:

```
godzina1:= Czas[1];              //odczytaj pierwszą cyfrę godziny
  if godzina1=IntToStr(0) then   //jeżeli pierwsza cyfra to zero, to...
    godzina1:='';                //...zmienna godzina1 nie ma wartości

godzina2:= Czas[2];              //odczytaj drugą cyfrę godziny

minuta1:= Czas[4];               //odczytaj pierwszą cyfrę minuty
  if minuta1=IntToStr(0) then    //jeżeli pierwsza cyfra to zero, to...
```

```
minuta1:='';                    //...zmienna minuta1 nie ma wartości

minuta2:= Czas[5];              //odczytaj drugą cyfrę minuty
```

Przygotowując ścieżkę dostępu dla funkcji PlaySound, uruchamiamy maszynę losującą (Randomize), która wylosuje początek wypowiedzi. W naszym katalogu zostało przygotowanych siedem wariantów tej wypowiedzi, więc z tego przedziału program będzie losował.

W przypadku ścieżek dostępu dla wypowiadania godziny i minut łączymy ze sobą zmienne „godzina1" i „godzina2", podobnie „minuta1" i „minuta2". Rozwiązanie to wymusza na nas przygotowanie 24 plików dźwiękowych dla pełnych godzin oraz 60 plików dla każdej minuty z osobna. Jeżeli nie chcemy tracić czasu na przygotowywanie tylu próbek dźwiękowych, możemy zmodyfikować kod, dostosowując go do rozwiązań omawianych we wstępie.

2. Przycisk „Który dzisiaj jest"

Podobnie jak w poprzednim przypadku zastosowaliśmy bezpośrednią konwersję czasu, w tym przypadku daty do postaci tekstowej string. Będzie ona miała następującą postać: rrrr:-mm:dd, razem dziesięć znaków. Do poszczególnych informacji odwołujemy się podobnie jak poprzednio:

```
Data:= (DateToStr(Date));     //odczytaj aktualną datę

dzien1:= Data[9];             //odczytaj pierwszą cyfrę dnia
dzien2:= Data[10];            //odczytaj drugą cyfrę dnia
miesiac1:= Data[6];           //odczytaj pierwszą cyfrę miesiąca
miesiac2:= Data[7];           //odczytaj drugą cyfrę miesiąca
```

Pamiętajmy, że nazwy plików dźwiękowych muszą odpowiadać tym, zawartym w ścieżce dostępu dla odtwarzacza Media Player.

Mając już odczytane i przypisane cyfry miesiąca i dnia, łąc-

zymy je bezpośrednio w ścieżce dostępu i odtwarzamy właściwy plik.

3. Przycisk „Jaki dzisiaj dzień”

Do odczytania aktualnej nazwy dnia wykorzystujemy jedną z funkcji Delphi, mianowicie:

```
Data:=Dni[DayOfWeek(Now)];      //odczytaj aktualny dzień tygodnia z tablicy
```

Wynik tej funkcji zapisujemy w stworzonej przez nas zmiennej „Data”. Aby móc wykorzystać tę funkcję, musimy na początku zadeklarować tablicę stałych, z których program będzie czerpał nazwy dni:

```
Dni: array[1..7] of String= (('Ni '),('Po '),('Wt '),('Śr '),('Cz '),('Pi '),('So '));
```

My, w naszym projekcie ograniczyliśmy się do podania dwóch pierwszych liter z uwagi na to, iż naszym celem nie jest ich wyświetlanie. Są to tylko nazwy umowne, dzięki którym program będzie wiedział, jaki plik dźwiękowy ma odtworzyć.
Oczywiste chyba jest, że pliki te muszą mieć takie same nazwy, a więc „Ni”, „Po”, „Wt” itd.

Po odczytaniu pierwszej i drugiej litery dnia i umieszczeniu w zmiennych „dzień1” i „dzień2” następuje ich połączenie w ścieżce dostępu MediaPlayer i odtworzenie odpowiadającego nazwie pliku dźwiękowego.

Uzupełnienie

Zarządzaj słowem

Jako uzupełnienie tematów zawartych w tej książce, prezentujemy pakiet poleceń programistycznych związanych z przetwarzaniem tekstu. Znajdują one swoje miejsce w jednym programie jako jego poszczególne opcje.

Założenia i opis programu:

Zamysł zbudowania takiego Centrum tekstowego zrodził się z chęci ukazania na konkretnych przykładach, działania poszczególnych poleceń. Centrum zawiera dwadzieścia opcji wpływających na wpisane przez nas słowo lub zdanie. Dzięki nim możemy na żywo przeglądać wpływ danego polecenia na wygląd tekstu.

W większości przypadków wprowadzamy założenie, aby istniała możliwość włączania i wyłączania danej opcji, czyli aby po ponownym naciśnięciu przycisku, słowo wróciło do pierwotnej formy. Stosując takie rozwiązanie, Czytelnik ma okazję zapoznać się z kolejnymi poleceniami i metodami, dzięki którym jego wiedza jeszcze bardziej się poszerzy.

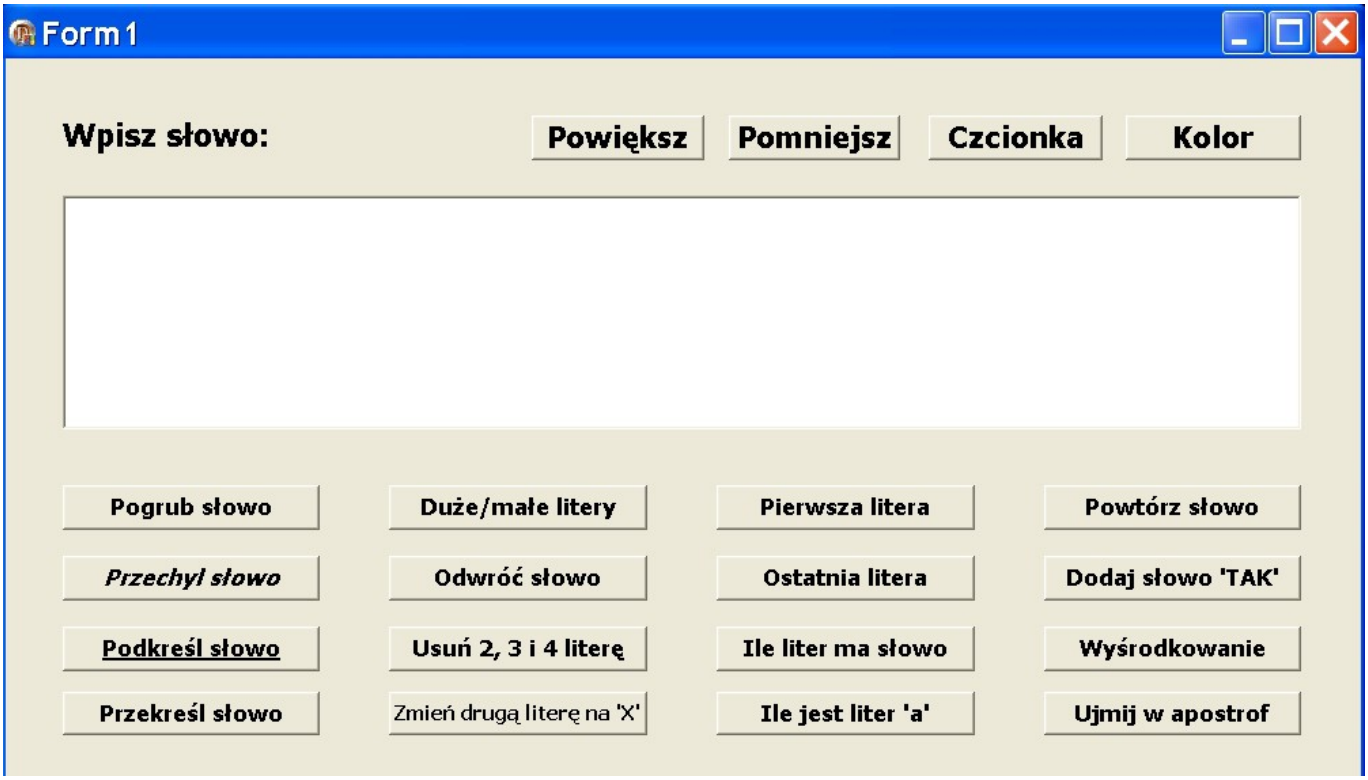
Chcąc uzyskać właściwe polecenie realizujące dane zadanie, należy skorzystać z komentarzy, zawartych w kodzie. Nie musimy przecież używać całej wykorzystywanej przez nas konstrukcji. Centrum ma być tylko inspiracją, skarbnicą wiedzy, z której czerpiemy właściwe polecenia. Jeżeli na przykład chcemy podkreślić słowo, to przecież nie musimy zamieszczać całej konstrukcji, jeśli nie jest nam ona potrzebna. Wykorzystamy wtedy tylko jedno, właściwe polecenie:

```
Memo1.Font.Style:=  [fsUnderLine];  //podkreśl słowo
```

Każda z opcji zmieniająca właściwość tekstu jest niezależna od pozostałych, stąd też poszczególne polecenia możemy wykorzystywać dowolnie w swoich projektach.

Potrzebne komponenty:

Projekt zawiera dwadzieścia przycisków, każdy uruchamiający jedną opcję zmiany tekstu, pola Memo, w które wpisujemy słowo lub zdanie do przetworzenia oraz etykiety tekstowej z napisem „Wpisz słowo”. Całość przyjmuje następujący wygląd:



A oto pełny kod programu:

(...)

Przechwyt klawiatury (hook)

Wstęp

Programowanie w języku Delphi umożliwia nie tylko tworzenie programów, ale także sterowanie podzespołami komputera takimi jak monitor, napęd CD\DVD czy porty szeregowe i równoległe. Zwiększa to znacznie możliwości dla naszych projektów, dzięki czemu możemy z nimi wyjść na zewnątrz komputera.

Nieodzownym elementem większości programów jest interfejs i sterowanie. To dzięki nim użytkownik może realizować swoje zamierzenia, wykorzystując do tego program. Przeważnie odbywa się to za pomocą klikania myszką w wirtualny przycisk umieszczony na interfejsie programu. Jest to wygodne rozwiązanie dla zwykłych (graficznych) programów komputerowych. Co jednak zrobić w przypadku, gdybyśmy chcieli stworzyć jakiś mobilny projekt, na przykład poruszającego się robota? Chodzenie za nim i uważanie, aby nas nie rozjechał podczas klikania w jego interfejs, nie wchodzi przecież w rachubę. Musimy stworzyć sterowanie jego funkcjami na innej zasadzie.

Pierwszym, najbardziej naturalnym rozwiązaniem tej kwestii jest klawiatura. Jeżeli użyjemy na dodatek klawiatury bezprzewodowej mamy w pełni mobilną jednostkę. Nie jest to jednak takie proste. Co prawda klawiatura posiada wystarczająco dużo klawiszy, a w połączeniu ze skrótami klawiaturowymi (czyli naciśnięcie kilku klawiszy jednocześnie) ma ich całą masę (kilkaset), jednak każdy z nich ma swoje przeznaczenie. Nie możemy więc ich wykorzystać w sposób bezpośredni.

Narzędzia programistyczne radzą sobie jednak doskonale z tego typu problemami. Klawiatura komunikuje się z komputerem, wysyłając doń komunikat, który klawisz został naciśnięty. Na podstawie uzyskanych informacji komputer wykonuje zadanie przypisane do tego klawisza. Jeżeli powiedzmy, naciśniemy klawisz z literką „a” nic się nie stanie, gdyż system z niego nie korzysta. Wystarczy jednak uruchomić jakikolwiek edytor tekstu

(chociażby WordPad), aby przekonać się, że ten klawisz jest jednak potrzebny.

Chcąc wykorzystać poszczególne klawisze do swoich celów lub zmienić ich przeznaczenie musimy przechwycić komunikaty, które są wysyłane do komputera. Po ich przechwyceniu wydajemy komputerowi własne komendy. Nic więc nie stoi na przeszkodzie, aby komputer po naciśnięciu klawisza „a" wykonał jakąś czynność, odtworzył muzykę lub na przykład, wyłączył się. Powyższy zabieg nazywamy przechwytem klawiatury lub – częściej – z anglojęzycznego oryginału, hookiem: zakładaniem hooka na klawiaturę.

Hook

Przy pomocy hooka możemy przejąć kontrolę nad każdym klawiszem klawiatury oraz utworzyć skróty klawiaturowe z wykorzystaniem klawiszy rozszerzonych, czyli Ctrl, Alt, Shift, Win, a nawet ich kombinacje, jak ctrl+alt+klawisz. Czym zatem jest ów hook? Jest to procedura, jedna z wielu, która odbiera od klawiatury informacje o tym, jaki klawisz lub klawisze zostały naciśnięte i na tej podstawie wykonuje działanie, które dla tego przypadku wcześniej zdefiniowaliśmy.

Hook to nie jest tylko jedna procedura. Jest to raczej ogólna nazwa procesu przechwytywania klawiatury, a konkretnych procedur, które to realizują, jest kilka. Zazwyczaj uzupełniają się one wzajemnie. Przedstawimy trzy oddzielne procedury, które przechwytują klawiaturę. Oto one: Key Press, Short Cut i Hot Key.

Każda z tych procedur króluje w pewnym, swoim tylko zakresie, lecz połączone razem doskonale się uzupełniają. Poniższa tabela przedstawia możliwości poszczególnych procedur:

(…)

Oto link do Centrum Sprzedaży:

http://wiedza-jest-super.blogspot.com/p/centrum-sprzedazy.html

---

Zapraszamy na strony internetowe:

Darmowe pliki do pobrania

www.chomikuj.pl/e-Darmo

e-booki w pełnych wersjach, audio-booki, zdjęcia (także do komercyjnego wykorzystania), prezentacje, czasopisma i wiele innych...

Wiedza jest super!

www.wiedza-jest-super.blogspot.com/

Blog tematyczny o szeroko rozumianej wiedzy: wiedzy zarówno praktycznej, doświadczalnej, naukowej, jak i wiedzy duchowej, religijnej i metafizycznej.

Zmień swoje życie

www.jakzmnienicswojezycie.blox.pl/

Blog poświęcony rozwojowi osobistemu, duchowemu i religijnemu prowadzony przez wieloletniego praktyka.

Samobójca z depresją

www.samobojcazdepresja.blogspot.com

Blog adresowany do osób zmagających się z myślami samobójczymi, depresją i własną niemocą.

Wiara jest super

www.wiara-jest-super.blogspot.com

Blog adresowany do osób, które chcą rozwijać swoją wiarę i zbliżać się do Pana Boga pisany przez wieloletniego Praktyka.

Zabawne obrazki

www.funny-bob.blogspot.com

Strona z zabawnymi obrazkami.

Zapraszamy!

Zapisz się na newsletter.

Jeżeli chciałbyś otrzymywać informacje o moich nowych książkach i publikacjach, proszę o wiadomość zwrotną z wyborem kategorii tematycznej i adresu e-mail, na jaki informacja ma trafiać. Można podać kilka kategorii. Będzie to tylko informacja na e-maila, bez przymusu i żadnych zobowiązań.

W temacie listu wpisz: Zapisz do newslettera

Oto kategorie do wyboru:

1. Wszystkie książki
2. Tylko darmowe książki
3. Książki dla dzieci
4. Sztuki teatralne
5. Poezja
6. Proza
7. Książki popularno-naukowe

8. Poradniki
9. Inne kategorie

e-mail kontaktowy: robert-trafny@wp.pl