unit Algo;
wersja 0.28

IMPLEMENTACJA CZĘŚCI NIESTANDARDOWEGO INTERFEJSU "ALGO" DLA DELPHI 5-7 i Borland Developer Studio i RAD 2010

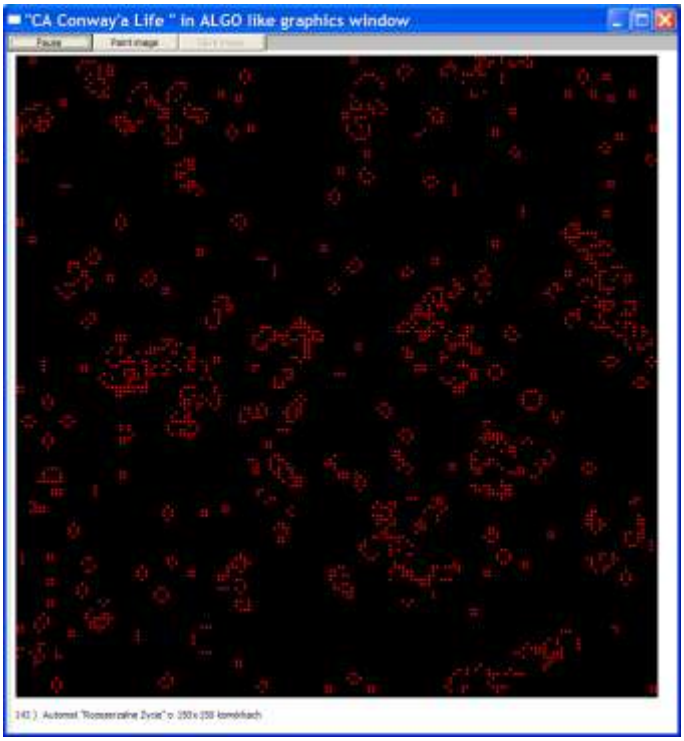
Autor: Wojciech Borkowski,

INSTYTUT STUDIÓW SPOŁECZNYCH UW: BORKOWSK@SAMBA.ISS.UW.EDU.PL
SPOŁECZNA PSYCHOLOGIA INFORMATYKI I KOMUNIKACJI (SPIK) SWPS:
BORKOWSK@SPIK.SWPS.EDU.PL

SKRÓCONY SPIS TREŚCI:

Wprowadzenie

Uczenie programowania a także podstaw symulacji komputerowych studentów o humanistycznej „proweniencji” to bardzo trudne zadanie ☺ Już samo programowanie stanowi wyzwanie dla umysłu nie nawykłego do liniowego, algorytmicznego myślenie, ale dołożenie do tego złożoności współczesnych środowisk programistycznych stawia przed studentem psychologiem, biologiem czy socjologiem poważną barierę. Dla wielu nie do pokonania… Stąd wybór na narzędzie nauczania środowiska ALGO – najprostszego dostępnego dla Windows narzędzi/środowiska do uruchamiania programów w języku Pascal.
Mimo pewnego niedopracowania i kilku wad projektowych ALGO sprawdza się dobrze jako narzędzie do nauczania. Przychodzi jednak moment, gdy przestaje wystarczać – i studentom i prowadzącemu. Pojawia się wtedy problem, jak łagodnie przejść na środowiska dającego większe możliwości nie tracąc dotychczasowego dorobku – i tego w postaci zrobionych programów i tego w postaci nabytych umiejętności.
„Unit ALGO” rozwiązuje ten problem w znacznym stopniu. Pozwala uruchomić w Borland Delphi program napisany w ALGO z użyciem podstawowej grafiki i typowej dla ALGO obsługi myszy i klawiatury. Trzeba jedynie dodać kilku linijek kodu i ewentualnie zrobić kilka innych drobnych modyfikacji wynikających z ograniczeń Object Pascala i *voila* – program działa! Można go teraz uzupełniać o te elementy Pascala, których w ALGO nie zaimplementowano – zbiory, operacja na bitach, zróżnicowane typy danych, a także skorzystać z możliwości Object Pascala – dynamicznych tablic, obiektowości. Istnieje też możliwość rozbudowania aplikacji w sposób typowy dla Delphi przez modyfikacje głównej „formy” wyświetlającej grafikę ALGO i przez dodawanie formularzy tworzonych kreatorem interfejsu użytkownika w środowisku Delphi/BDS czy RAD 2010.



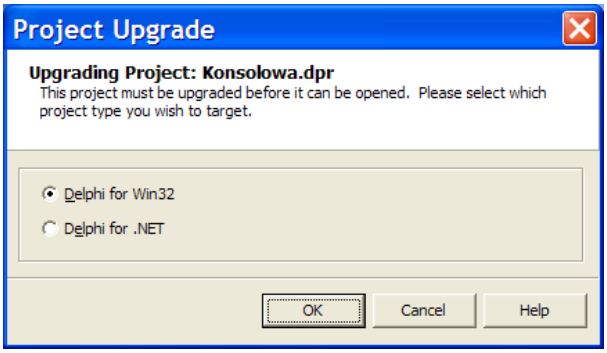
Rysunek 1: Aplikacji Delphi implementująca automat komórkowy „Game of Life” w oknie obsługiwanym przez „Unit ALGO”.

Specyfika aplikacji w Delphi

Filozofia tworzenia aplikacji w **Delphi** jest bardzo odmienna od klasycznego programowania proceduralnego, jakie stosuje się w standardowym Pascalu. Aplikacja jest zbudowana nie wokół algorytmu, lecz wokół tworzonego w kreatorze graficznego interfejsu użytkownika. Jest to zupełnie odmienny paradygmat programowania – sterowane zdarzeniami programowanie obiektowe, a właściwie jego specjalna odmiana – „programowanie komponentowe" albo RAD (*Rapid Application Development*)
Główny program Pascalowy, umieszczony w pliku o obowiązkowym rozszerzeniu DPR (*Delphi Project*) jest domyślnie przed użytkownikiem ukryty i modyfikowany jedynie przez środowisko Delphi (IDE – *Integrated Development Environment*). Programista Delphi ma pracować jedynie na plikach źródłowych modułów (unit), które zwykle związane są z formularzami i zarządzane przez kreatora – jedyne zadanie człowieka to wpisywanie kodu w ciała zdefiniowanych automatycznie metod obiektów interfejsu.
Można jednak używać Delphi jako zwykłego kompilatora ObjectPascala, a nawet Pascala niemal standardowego tworząc tzw. *aplikację konsolową*. Wystarczy zrobić plik o rozszerzeniu DPR podobny jak poniższy:

```
Program KonsolaDemo;
{$APPTYPE CONSOLE}
begin
writeln( 'Witam z konsoli tekstowej - naciśnij ENTER' );
readln;
end.
```

Tekst {$APPTYPE KONSOLE} gwarantuje, że program będzie aplikacją konsolową – tzn. będzie miał dostęp do konsoli – albo specjalnie otwartej, albo tej, z której plik EXE został uruchomiony. Podwójne kliknięcie w plik o rozszerzeniu DPR powoduje jego wczytanie do zainstalowanego środowiska obsługującego jezyk **ObjectPascal/Delphi**. Przy pierwszym wczytywaniu operator może zostać poproszony o wybór typu, aplikacji jaki zostanie utworzony. Gdy odpowiedni zestaw plików pomocniczych już istnieje pytanie się nie pojawi.



Rysunek 2: Pytanie o rodzaj aplikacji do jakiej będzie upgradowany projekt zawierający jedynie plik DPR lub plik DPR i pliki źródłowe modułów, ale nie zawierający zestawu plików koniecznych do działania środowiska (IDE).

Niestety jest to „ślepy zaułek" – normalna aplikacja konsolowa jest niezgodna z komponentami GUI (*Graphics User Interface*), i nie może być w „tą stronę" rozbudowywana. Co gorsza firma **Borland** tworząc **Delphi** nie zapewniła też zgodności na poziomie źródeł ze swoim wcześniejszym produktem **Turbo/Borland Pascalem** i aplikacje z tego środowiska

używające tzw. crt – czyli rozbudowanego dostępu do ekranu tekstowego, oraz grafiki BGI nie dają się uruchomić w środowisku Delphi[1].
Główny powód tej niezgodności to zawłaszczanie przez interfejs użytkownika całego sterowania w programie. Plik DPR typowej aplikacji używającej GUI wygląda podobnie do poniższego…

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {FormularzGlowny} ,
  Unit2 in 'Unit2.pas' {MojPaint} ;



{$R *.res}



Begin

  Application.Initialize;
  Application.CreateForm
          (TFormularzGlowny,FormularzGlowny);
  Application.CreateForm
          (TMojPaint,MojPaint);

..Application.Run;

end.
```

Deklaracje użytych modułów. „Forms” to niezbędny moduł definiujący podstawy API (*Aplication Programing Interface*) dla obsługi formularzy. Pozostałe widoczne tu moduły to definiowane dla programisty przez kreatora GUI kody obsługujące poszczególne formularze

Dyrektywa nakazująca szukać plików zasobów definiujących wygląd formularzy w tym samym katalogu co odpowiednie pliki źródłowe modułów

Inicjalizacja obiektu aplikacji odpowiedzialnego za sterowanie GUI, oraz zdefiniowanych w kreatorze formularzy

Przekazanie sterowania do GUI. Od tego momentu aplikacja reaguje jedynie na zdarzenia interfejsu. Jakikolwiek kod umieszczony za tym wywołaniem wykona się dopiero po zamknięciu głównego formularza GUI, czyli na koniec działania aplikacji.

Konieczność wywołania `Application.Run` przekreśla właściwie wszelkie szanse na eleganckie zaprogramowanie współpracy aplikacji konsolowej z GUI w ramach aplikacji jednowątkowej. Jedynym rozwiązaniem jest stworzenie drugiego wątku. Optymalnie byłoby, gdyby taki nowy wątek przejął wywołanie feralnej metody Run. Wtedy jedyną dużą modyfikacją w kodzie źródłowym przystosowująca aplikacje konsolową do współpracy z

[1] Choć sprawa nie jest całkiem przegrana – istnieją produkty zapewniające lepiej lub gorzej taką wsteczną kompatybilność, np. unit crt32 czy różne wersje graph32 (jedna autora tego tekstu)

GUI byłoby dodanie deklaracji użycia odpowiedniego „wrappera”[2] w sekcji `uses` – np. `uses TurboPascal;` albo `uses Algo`;
Niestety to tylko marzenie. Z przyczyn głęboko ukrytych w implementacji bibliotek Delphi takie rozwiązanie powoduje zawieszenie aplikacji – tylko główny wątek aplikacji może obsługiwać GUI. Nakłada to istotne ograniczenia na elegancję rozwiązań łączących klasyczne programowanie proceduralne z programowaniem zdarzeniowym Delphi.
Inne ograniczenia wynikają z nieregularności składni Pascala. Procedury biblioteczne `write`, `writeln`, `read`, `readln` są w Pascalu na specjalnych prawach. Nie dość, że mogą przyjmować zmienną liczbę parametrów prostych typów, to mogą mieć jeszcze wyróżniony, pierwszy parametr plikowy, a `write`, `writeln` mogą akceptować jeszcze parametry formatujące oddzielane dwukropkiem[3], co ma jedyny odpowiednik w procedurze `str` konwertującej zmienną liczbową na łańcuch tekstowy.

Założenia implementacji modułu ALGO

Unit Algo ma za zadanie w miarę wiernie odwzorować interfejs programistyczny **ALGO** na procedury i metody obiektów delfiowego API – głównie chodzi tu o funkcje prymitywów graficznych, obsługę zdarzeń (`IsEvent` i `Event`) i kilka innych często używanych, a specyficznych dla **ALGO** lub działających w nim odmiennie niż w **ObjectPascalu**. Moduł opakowuje definicję i przygotowanie głównej formy GUI zawierającej komponenty graficzne do rysowania, oraz przyciski do podstawowej obsługi aplikacji oraz pole edycyjne przyjmujące znaki z klawiatury kierowane do tego okna. Moduł przejmuje też na siebie inicjację głównego obiektu aplikacji, wywołanie `Application.Run` oraz w odpowiednim momencie uruchomienie wątku wykonującego kod „proceduralny” pierwotnej aplikacji **ALGO**
W module zaimplementowany jest zestaw procedur `Write`/`Writeln`. Korzystając z dostępnego w Delphi mechanizmu przeciążania funkcji i typu `variant` na który automatycznie konwertują się typy proste deklaracje te „przykrywają” definicje systemowe `Write`/`Writeln` aż do 6 parametrów. Dzięki temu „pisanie” odbywa się tak jak w **ALGO** na okno graficzne, a jednocześnie dla zachowania zgodności kopia tekstu trafia na konsolę, jeśli aplikacja została skompilowana jako konsolowa.
Ponieważ niemożliwa jest reimplementacja przez programistę procedur z parametrami formatującymi oddzielanymi dwukropkiem zdefiniowano dodatkową funkcję `Format` która pełni tę rolę.
Moduł nie ma natomiast własnego zestawu procedur `read`/`readln`. Jeśli zachodzi konieczność ich użycia do odczytania danych z klawiatury, to działają one na konsoli tekstowej. To rozwiązanie powoduje, że niekiedy zawartość konsoli może nie być zsynchronizowana z zawartością okna graficznego (okno graficzne wygląda jakby było „w tyle” za treścią konsoli). Można ten problem rozwiązać wywołując przez `read` funkcję synchronizującą `AlgoSync`.

Różne wersje modułu ALGO

Kolejne wersje Delphi różnią się między sobą szczegółowymi właściwościami działania bibliotek. Powoduje to konieczność dostarczania wariantów kodu dla różnych zastosowań. Najniższa obsługiwana wersja to **Delphi 5** najwyższa przetestowana to **Rapid Application Development Studio 2010**

[2] „Wrapper” to biblioteka zamieniająca inteligentnie odwołania jednego interfejsu programistycznego na inny. Np. odwołania do grafiki BGI na odwołania do grafiki Windows, albo odwołania do procedur czasu wykonania ALGO na odwołania do biblioteki czasu wykonania ObjectPascala.
[3] Jak w wywołaniu `write(Pi:10:7)`

Dopasowanie można uzyskać modyfikując odpowiednio poniższą deklarację znajdującą się w początkowej części pliku *Algo.pas*:

```
interface
const DELPHI_VERSION=6; {7,8...warunkowa kompilacja w
zależności od wersji Delphi}
```

ale w przypadku wersji 5 nie posiadającej warunkowej kompilacji zależnej od wartości stałej trzeba użyć specjalnej wersji modułu, kopiując pliki *Algo.pas* i *Algo.dfm* z katalogu **Delphi 5** do katalogu instalacyjnego pakietu.
Zestawienie wykrytych różnic implementacyjnych w przetestowanych wersjach kompilatorów zawiera poniższa tabela.

Kompilator	**Konieczne pliki**	**Wartość** DELPHI_VERSION	**Działanie zapisu obrazka**	**Obracanie napisów**
Delphi 5	*Delphi5/Algo.pas* *Delphi5/Algo.dfm*	5, ale raczej bez znaczenia	+/-	-
Delphi 6	*Algo.pas* *Algo.dfm*	6	+/-	-
Delphi 7	*Algo.pas* *Algo.dfm*	7	- ???	-
BDS 2006	*Algo.pas*	8	+/-	+

Uruchamianie aplikacji ALGO w Delphi

W dalszej części tekstu znajdują się szczegółowe informacje na temat przystosowania programu napisanego w ALGO do kompilacji w Delphi, możliwości jego rozbudowy o bardziej złożone API oraz użycia poszczególnych procedur i funkcji modułu.

Szybki start

Program dla ALGO należy zapisać pod nazwą z rozszerzeniem *dpr* (*Delphi project*) i umieścić w katalogu razem z plikami *Algo.pas* i ewentualnie *Algo.dfm*. Plik *dfm* nie jest potrzebny, gdy kompilator Delphi jest w wersji wyższej niż 7 - ale trzeba wtedy zmienić definicje `DELPHI_VERSION` w pliku *Algo.pas* na wartość większą niż 7.

Po pierwszej linii programu w pliku *dpr*, „brzmiącej" zwykle jakoś tak:

```
Program NazwaProgramu;
```

Trzeba umieścić linię:

```
uses Algo in 'Algo.pas';
```

Jeśli aplikacja ma używać procedury `read`, albo udostępniać dane tekstowe do kopiowania przez schowek to trzeba dodać jeszcze:

```
{$APPTYPE CONSOLE}
```

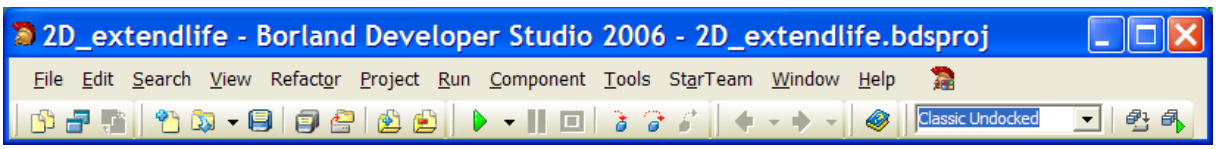
Następnie pierwotny program główny, czyli kod znajdujący się pomiędzy „`end` z kropką", a odpowiadającym mu `begin`, należy przekształcić w procedurę (nazwaną np. Main), a w nowym programie głównym umieścić jedynie wywołanie procedury uruchamiającej aplikacje:

```
begin
RunAsALGO(Main, 'Nazwa aplikacji')
end.
```

Jako pierwszy parametr należy podać procedurę reprezentującą dawny program główny, a jako drugi łańcuch tekstowy będący nazwą dla okna.
Można podać też wymagane rozmiary użytkowe okna grafiki jako dwa kolejne parametry. Np.:

```
RunAsALGO(Main,'Moja aplikacja',800,600)
```

Teraz program można skompilować i uruchomić. Najprościej jednym zielonym przyciskiem…

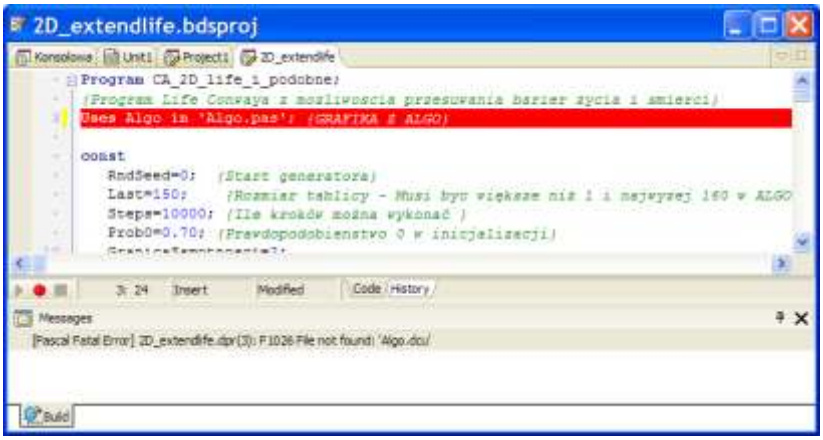


Rysunek 3: Belka głównego menu BDS2006 z widocznymi ikonami skrótów do ważniejszych akcji. Naciśnięcie zielonego trójkąta powoduje kompilacje i uruchomienie programu. W przypadku wcześniejszych wersji Delphi wygląda to analogicznie.

Jeśli plik `dpr` znajduje się w tym samym katalogu, co pliki `Algo.pas` i `Algo.dfm`, a sam program jest raczej prosty to powinno pojawić się okno analogiczne jak na Rysunek 1.
Po naciśnięciu „Start" pierwszego z lewej przycisku w oknie graficznym, procedura `Main` (czy inna podana jako pierwszy parametr wywołania procedury `RunAsAlgo`) zostanie uruchomiona w osobnym wątku.
Będzie on działał do momentu zakończenia sposobem przewidzianym w kodzie pochodzącym z ALGO. Można ten wątek wstrzymać i wznowić przez ponowne naciskanie pierwszego przycisku a całą aplikację zakończyć w dowolnym momencie zamykając okno w dowolny przyjęty w Windows sposób (Alt-F4, kliknięcie w 'X' itd.)
Działanie procedur graficznych jest domyślnie skierowane na komponent Delphi o nazwie `PaintBox` ponieważ gwarantuje to szybkie i niezawodne rysowanie. Wciskając drugi przycisk okna graficznego można skierować na alternatywny komponent `Image` okna graficznego. Rysowanie jest wtedy kilkukrotnie wolniejsze, a co więcej może być rozsynchronizowane z przebiegiem programu, można za to zapisać zawartość graficzną bezpośrednio do pliku[4]
Po zakończeniu wątku proceduralnego można go wystartować ponownie bez zamykania okna aplikacji.

Typowe błędy pierwszego uruchamiania

Jeśli naciśniecie zielonego trójkąta nie powoduje pojawienia się okna aplikacji analogicznego do tego z Rysunek 1 to najprawdopodobniej nastąpił błąd kompilacji, którego rodzaj można stwierdzić przeglądając podokno komunikatów znajdujące się zazwyczaj poniżej podokna edycyjnego w głównym oknie projektu:



Rysunek 4: Podokno edycji z uwidocznionym błędem kompilacji opisanym w znajdującym się poniżej oknie komunikatu. W tym wypadku, choć komunikat jest nieco inny, nie udało się znaleźć pliku `Algo.pas`, więc nie został wyprodukowany plik `Algo.dcu` – skompilowana wersja modułu.

Użycie znaków narodowych (polskich liter) w identyfikatorach

Kompilator ALGO pozwala na użycie znaków narodowych w identyfikatorach Pascala, czyli w definiowanych przez programistę nazwach procedur, funkcji, zmiennych i typów. Także nazwa programu po instrukcji `Program` oraz nazwa pliku mogą zawierać polskie litery.

- Nazwy plików zawierające znaki narodowe powodują błędy kompilacji we wszystkich wersjach kompilatorów **Borlanda**/**Embarcadero** – trzeba je zamienić na łacińskie odpowiedniki.

[4] Niestety w nowszych wersjach Delphi zmieniono gdzieś głębokie szczegóły implementacyjne i rysowanie „na obrazek" niemal nigdy nie działa poprawnie ☹ Trzeba by jakoś zmienić implementacje, ale nie wiem jak i nie bardzo mam czas to tropić…

- Jedak od wersji **BDS 2006** identyfikatory w kodzie mogą zawierać znaki narodowe
- W Delphi 5 i 6 (i najprawdopodobniej też w 7) wszystkie znaki narodowe w identyfikatorach należy zamienić na łacińskie odpowiedniki zwracając uwagę by nie doprowadzić do „zlania się" pierwotnie różnoimiennych identyfikatorów w jeden.

Brak dostępu do plików źródłowych

Najbardziej prawdopodobnym błędem może być niedostępność plików źródłowych modułu Algo. Pakiet nie zawiera wersji skompilowanej modułu w postaci pliku *Algo.dcu*, ponieważ każda wersja Delphi ma nieco inny format tych plików. Plik taki powstaje natomiast automatycznie przez kompilacje *Algo.pas*, pod warunkiem, że kompilator może go odnaleźć.
Żeby nie umieszczać kopi plików dla każdego kolejnego projektu można posłużyć się rozwiązaniem zastosowanym dla przykładowych projektów dostarczanych w pakiecie „**Unit_ALGO**". Plik projektu (*dpr*) jest umieszczony w podkatalogu katalogu instalacyjnego pakietu i w dyrektywie **uses** podana jest informacja, że pliku *Algo.pas* należy szukać w katalogu nadrzędnym:

```
uses Algo in '../Algo.pas';
```

Użycie nie zaimplementowanych procedur

Moduł implementuje znaczący podzbiór procedur API **ALGO**, ale nie wszystkie. Nie są zaimplementowane żadne elementy grafiki żółwiowej, nie ma też żadnych procedur „robota". Procedura `PlaySound` występuje na razie w postaci atrapy, która zamiast odgrywania dźwięku z pliku *WAV* produkuje „trzy dzwonki".

Niezgodne wyjście konsolowe lub plikowe (write/writeln)

Moduł implementuje `write/writeln` jako zestaw przeciążonych[5] procedur o różnej liczbie parametrów typu `variant`:

```
procedure Writeln(p1,p2,p3,p4,p5,p6:variant);overload;
```

Największa zaimplementowana liczba parametrów to 6, więc jeśli w kodzie ALGO pojawi się wywołanie o większej liczbie parametrów to nie zostanie poprawnie zrozumiane. Należy takie wywołanie `write/writeln` podzielić na kilka wywołań, z których każde nie będzie miało więcej niż 6 parametrów.
Drugi problem, który może się pojawiać to użycie formatowania wyników za pomocą dwukropków. Np.:

```
Var a:integer;
      x:real;
…
writeln(a:3);
writeln(x:10:7);
```

Ponieważ nie ma możliwości reimplementacji przez programistę używającego Delphi takiego nietypowego przekazywania parametrów, w moduł dostarcza zastępczo dwie przeciążone

[5] O funkcjach przeciążonych mówimy wtedy, gdy tej samej nazwy funkcji można użyć z różnymi zestawami parametrów. Kompilator rozpoznaje właściwą funkcję porównując nazwę i parametry formalne w deklaracjach z nazwą funkcji i parametrami aktualnymi jej wywołania.

funkcje o nazwie `format`. Akceptują one parametry typu `integer` lub `real` i odpowiednio 1 lub 2 parametry formatujące, a zwracają łańcuch tekstowy zawierający odpowiednio sformatowaną tekstową reprezentacje podanego parametru liczbowego. Powyższy przykład należałoby przekształcić tak:

```
Var a:integer;
       x:real;
…
writeln(format(a,3));
writeln(format(x,10,7));
```

Pełna lista wykrytych błędów i przypadków niekompatybilności

- Ze względu na brak parametrów otwartych w Delphi procedury `write[ln]` akceptują jedynie do 6 parametrów. Wywołania z większą liczbą parametrów trzeba podzielić.
- Brak jest reimplementacji `Read`/`Readln`, więc procedury te mogą nie działać synchronicznie w stosunku do okna graficznego. Można to zachowanie poprawić wywołując bezpośrednio przed `Read`/`Readln` procedurę `AlgoSync`;
- Funkcje `readln`/`read` czytają treść tylko z konsoli tekstowej nie pozostawiając echa na ekranie graficznym i nie przechodząc tam do następnej linii.
- W wersji Delphi 5,6 i 7 nie działa nachylenie tekstu, a w żadnej wersji nie działa atrybut wytłuszczania tekstu
- Rysowanie w trybie „Image" umożliwiającym bezpośredni zapis do pliku graficznego jest bardzo powolne, a ponadto często potrafi pomijać fragmenty rysunku. Wynika to z nieodkrytego błędu współdziałania obu wątków programu i okazało się niezwykle trudne do usunięcia
- Nie zaimplementowano `PlaySound` - zamiast tego jest prosty sygnał dźwiękowy

Historia poprawek do wersji 0.24 w porządku odwrotnym:

0.285:

- Jakieś drobne poprawki techniczne.

0.281:

- „Etykieta autorska" przenosząca na stronę www.iss.uw.edu.pl/borkowski

0.28:

- Zablokowanie wywołania formy konfiguracyjnej, gdy wątek główny działa i nie jest zawieszony

0.272:

- .....walka z synchronizacją :((((Bez skutku, nadal TPicture rysuje się kawałkami

0.271:

- Poprawienie kompatybilnosci z Delphi 7 (WindowHandle zamiast GetOwnerWindow)
- i przywrócenie zaginionego wywołania Randomize w inicjlizacji modułu

0.27:

- Poprawione zarządzanie wywoływaniem formularza konfiguracyjnego
- Główna forma jest chowana, a wątek zawieszany jeśli konfiguracja zostanie wywołana bez jawnego zawieszenia wątku głównego
- Funkcja dająca dostęp do surowego uchwytu okna
- Dodanie ikonki z kwadracikami do domyślnego DFMa
- Dodanie przykładów z oknem konfiguracyjnym

0.25-26:

- Funkcja zmiany rozmiaru ekranu
- Zabezpieczenie zawieszania wątku w czasie wywoływania setupu
- Dodanie automatycznej numeracji zrzucanych obrazków

0.24:

- Przygotowano specjalną wersję kodu modułu dla **Delphi 5.0**
- Dodano możliwość restartu głównego wątku
- Dodano możliwość podpięcia dodatkowej formy zrobionej w kreatorze, a z nią praktycznie całej aplikacji Delphi, pod warunkiem ręcznej inicjalizacji form w programie głównym.

0.23:

- Dodano wywołanie `AlgoSync`; w procedurze `Delay`
- Dodano komunikat o zakończeniu wątku aplikacyjnego
- Dodano procedurę `Randomize(seed)`, której nie ma w **Delphi**
- Dodano funkcje `format(v:number,f,f):string` zastępującą nieprzykrywalne konstrukcje formatujące liczby w wywołaniach `writeln`.

0.22:

- Dodano wywołanie procedury `Randomize` w inicjacji wątku głównego dla poprawienia zgodności z tym, co robi **ALGO**
- Usunięcie błędu w wariantowej kompilacji dla wersji 6 i 7 (z plikiem dfm) i wyższych (bez pliku)

Detaliczny opis procedur modułu

Poniżej znajduje się kompletna lista procedur modułu z podziałem na kategorię i informacjami o użyciu każdej z nich. Część tekstu została bezpośrednio oparta na zawartości helpu ALGO jako jedynej dokumentacji interfejsu tego kompilatora.

Podstawowe Procedury graficzne

Procedure Line(x1, y1, x2, y2: Integer);

Rysuje odcinek linii prostej łączący punkt o współrzędnych (x1, y1) z punktem o współrzędnych (x2, y2). Przemieszcza także kursor graficzny do punktu o współrzędnych (x2, y2).

Procedure LineTo(x, y: Integer);

Rysuje odcinek linii prostej od aktualnej pozycji kursora graficznego do punktu o współrzędnych (x, y). Przemieszcza także kursor graficzny do punktu o współrzędnych (x, y).

Procedure MoveTo(x, y: Integer);

Przemieszcza kursor graficzny do punktu o współrzędnych (x, y).

Procedure Coordinates(Var x, y: Integer);

Zwraca informację o współrzędnych kursora graficznego.

Procedure Pen(n, r, g, b: Integer);

Ustala kolor i grubość rysowanych linii. Parametry `r, g, b` – to nasycenie kolorami czerwonym, zielonym i niebieskim, a parametr n - grubość.

Procedure Brush(k, r, g, b: Integer);

Ustala kolor i styl wypełnienia. Parametry r, g, b – to nasycenie kolorami czerwonym, zielonym i niebieskim. Jeśli k=1 to figury są zamalowywane wybranym kolorem pędzla, jeśli k=0 to kolor jest przezroczysty.

Procedure TextColor(r, g, b: Integer);

Określa kolor dla wypisywanych tekstów procedurami `Write` i `WriteLn`. Parametry r, g, b – to nasycenie kolorami czerwonym, zielonym i niebieskim.

Procedure Rectangle(x1, y1, x2, y2: Integer);

Rysuje prostokąt, którego przeciwległe wierzchołki mają współrzędne (x1, y1) i (x2, y2). Przesuwa także kursor graficzny do punktu o współrzędnych (x2, y2).

Procedure Ellipse(x1, y1, x2, y2: Integer);

Rysuje elipsę. Parametry określają współrzędne dwóch przeciwległych wierzchołków prostokąta opisanego na elipsie. Współrzędne kursora graficznego nie ulegają zmianie.

Procedure Point(x, y: Integer);

Zaznacza punkt o współrzędnych (x, y) w kolorze pisaka. Przemieszcza także kursor graficzny do punktu o współrzędnych (x, y).

Procedure Fill(x, y: Integer);

Wypełnia zadanym kolorem pędzla wnętrze obszaru obejmującego punkt o współrzędnych (x, y)

Procedure Font(rozmiar, kierunek, grubosc: Integer);

Wybiera rozmiar (6..72), kierunek (0..359) i grubość (1..1000) wypisywanych tekstów procedurami `Write` i `WriteLn`. Argumentami procedury Czcionka mogą być dowolne wyrażenia całkowite. W **Unit_ALGO** nie jest w pełni zaimplementowana ze względu na brak odpowiednich właściwości komponentu **Delphi** obudowującego czcionkę API **Windows**

Procedure Clear;

Wyczyszczenie ekranu graficznego i usytuowanie kursora graficznego w lewym górnym rogu okna wyników.
Oprócz wyczyszczenia okna, procedura wykonuje następujące czynności:
- ustawia czarny pisak;
- ustawia czarny kolor tekstu;
- ustawia przezroczysty kolor wypełnienia;
- ustawia czcionkę (8,0,400).

Inne niezbędne procedury API ALGO

Procedure Date(Var rok, miesiac, dzien: integer);

Procedura wylicza aktualną datę, czyli rok (1900..2099), miesiąc (1..12), dzień i dzień (1..31).

Procedure Time(Var godzina, minuta, sekunda: Integer);

Procedura wylicza aktualny czas, czyli godzinę (0..23), minuty (0..59) i sekundy (0..59).

Procedure Delay(ms: Integer);

Wstrzymanie wykonywania programu na okres ms milisekund.

Procedure PlaySound(atr: Integer; plik: string);

Odtwarza pliki dźwiękowe typu *.wav.
Pierwszym parametrem jest wyrażenie całkowite atr.
0 - wstrzymuje wykonywania programu na czas odtwarzania;
1 - odtwarza plik bez wstrzymania programu;
2 - jeśli w momencie wywołania jest odtwarzany inny dźwięk to zostanie przerwany;
3 - odtwarzanie w pętli bez końca;
4 - wstrzymanie odtwarzania pliku.
Drugim parametrem jest napis określający nazwę pliku typu *.wav. W ogólnym przypadku nazwa określa pełną ścieżkę dostępu do pliku - dysk, folder i jego nazwę.
NA RAZIE NIE ZAIMPLEMENTOWANA W **Unit_ALGO**!

Random i sparametryzowane Randomize

W Delphi występuje zgodna ze standardem Pascala funcja `Random` w postaci bezparametrowej i w postaci parametrowej, natomiast funkcja `Randomize` – inicjacja generatora liczba pseudolosowych występuje tylko w postaci bezparametrowej. W Algo istnieje postać z parametrem pozwalająca ustalić „ziarno” generatora tzn. zainicjować generator dowolną liczbą całkowitą. Stąd **Unit_Algo** zawiera dwie definicje przesłaniające wersję domyślną, i definiującą wersję sparametryzowaną:

```
procedure Randomize;overload;
procedure Randomize(seed:integer);overload;
```

UWAGA:

- Nie ma gwarancji, że generatory liczb pseudolosowych ALGO i Delphi będą produkować taki sam ciąg liczb po zainicjowaniu tym samym „ziarnem”!
- Generator ALGO ma błąd powodujący, że wersja bezparametrowa zwraca niekiedy liczbę 1.0. Bezparametrowe Random w Delphi nigdy nie zwraca 1, choć może zwrócić liczbę, która po wykonaniu dowolnej operacji zmiennoprzecinkowej wymagającej zaokrąglenia da wynik taki jakby była równa 1.

Obsługa wejście-wyjście okna graficznego

Function IsEvent: Boolean;

Wynikiem funkcji jest wartość logiczna prawda, jeśli od ostatniego wywołania procedury Zdarzenie zaszło jakieś zdarzenie (naciśnięcie klawisza klawiatury lub lewego przycisku myszy w obrębie okna wyników), w przeciwnym przypadku fałsz.

Procedure Event(Var k, x, y: Integer);

Za zdarzenie uważa się naciśnięcie klawisza na klawiaturze lub wciśnięcie lewego przycisku myszki w obrębie okna wyników. Jeśli w momencie wywołania procedury zdarzenia jeszcze nie było, to program oczekuje zdarzenia.
Wywołanie procedury Zdarzenie powoduje przypisanie zmiennym k, x, y wartości:

- k=1, x=kod, y=0 - naciśnięto klawisz sterujący nie mający reprezentacji ASCII (np. Home, F5);
- k=1, x=kod, y=1 - naciśnięto klawisz sterujący ASCII (np. Enter, Tab);
- k=1, x=kod, y=2 - naciśnięto klawisz ASCII o kodzie >31 (np. t, H, O);
- k=2, x, y=współrzędne kursora myszy - wciśnięto lewy przycisk myszy;
- k=3, x, y=współrzędne kursora myszy - mysz przemieszcza się z wciśniętym lewym przyciskiem.

Procedura Write/Writeln dla okna i dla plików

Procedura ta jest zaimplementowana w postaci zestawu przeciążonych procedur o różnej liczbie parametrów typu `variant`. Co prawda wersje z parametrem plikowym wydają się zbędne, ale obecność deklaracji dla konsoli ukrywa też w Delphi domyślne `write` i `writeln` dla plików.

```
procedure Writeln;overload;
procedure Write(p1:variant);overload;
procedure Writeln(p1:variant);overload;
procedure Write(p1,p2:variant);overload;
procedure Writeln(p1,p2:variant);overload;
procedure Write(p1,p2,p3:variant);overload;
procedure Writeln(p1,p2,p3:variant);overload;
procedure Write(p1,p2,p3,p4:variant);overload;
procedure Writeln(p1,p2,p3,p4:variant);overload;
procedure Write(p1,p2,p3,p4,p5:variant);overload;
procedure Writeln(p1,p2,p3,p4,p5:variant);overload;
procedure Write(p1,p2,p3,p4,p5,p6:variant);overload;
procedure Writeln(p1,p2,p3,p4,p5,p6:variant);overload;

procedure Writeln(var f:text);overload;
procedure Write(var f:text;p1:variant);overload;
procedure Writeln(var f:text;p1:variant);overload;
procedure Write(var f:text;p1,p2:variant);overload;
procedure Writeln(var f:text;p1,p2:variant);overload;
```

```
procedure Write(var f:text;p1,p2,p3:variant);overload;
procedure Writeln(var f:text;p1,p2,p3:variant);overload;
procedure Write(var f:text;p1,p2,p3,p4:variant);overload;
procedure Writeln(var f:text;p1,p2,p3,p4:variant);overload;
procedure Write(var f:text;p1,p2,p3,p4,p5:variant);overload;
procedure Writeln(var f:text;p1,p2,p3,p4,p5:variant);overload;
procedure Write(var f:text;p1,p2,p3,p4,p5,p6:variant);overload;
procedure Writeln(var f:text;p1,p2,p3,p4,p5,p6:variant);overload;
```

Można pominąć zbędne w tym miejscu użycie wrappera posługując się składnią bezpośrednio odwołującą się do źródła deklaracji:

```
System.writeln(plik,”Coś do pliku”);
```

Funkcja Format

Funkcja formatująca wartości `real` i `integer` dla `write`/`writeln` zastępująca pascalową składnię write(x:c:p), której nie da się przeimplementować. Jej deklaracja w kodzie źródłowym modułu wygląda następująco:

```
function Format(v:real;c:integer;ap:integer):string;overload;
function Format(v:integer;c:integer):string;overload;
```

Funkcje te akceptują parametry typu `integer` lub `real` i odpowiednio 1 lub 2 parametry formatujące, a zwracają łańcuch tekstowy zawierający odpowiednio sformatowaną tekstową reprezentacje podanego parametru liczbowego. Na przykład poniższy kod:

```
Var a:integer;
       x:real;
…
writeln(a:3);
writeln(x:10:7);
```

należałoby przekształcić tak:

```
Var a:integer;
       x:real;
…
writeln(format(a,3));
writeln(format(x,10,7));
```

Interfejs spinający kod proceduralny ALGO z GUI aplikacji Delphi

Procedure AlgoSync;

Zapewnia uaktualnienie okna graficznego przed spodziewana przerwą, np. wywołaniem `read`/`readln`. Nie trzeba jej wywoływać przed `Delay` ani przed `Event` ponieważ takie wywołanie już się we wnętrzu tych procedur znajduje

Function GetMyForm:pointer;

Funkcja ta pozwalająca otrzymać uchwyt do głównej formy aplikacji zdefiniowanej we wnętrzu modułu i ukrytej przed kodem użytkownika modułu. Wynik trzeba zrzutować na typ `TForm` i można wykonać na nim wszystkie operacje udostępnione dla tego typu przez API Delphi

Procedura rejestrowania aplikacji ALGO

Kod modułu musi otrzymać kilka niezbędnych informacji, żeby mógł zainicjować współpracę kodu proceduralnego ALGO z kodem GUI Delphi. Są to:

- Punkt wejścia wątku aplikacyjnego, czyli procedura zrobiona z programu głównego ALGO
- Nazwa dla okna aplikacji, która będzie stanowić część napisu w belce okna.
- Rozmiary komponentu graficznego, na który będzie trafiać rezultat działania funkcji graficznych i `Write` i `WriteLn`

Definicja ta wygląda tak:

```
type MainProcedure=procedure;
Procedure RunAsALGO(Main:MainProcedure;
                    AppName:string='Application';
                    Width:integer=800;
                    Height:integer=800
                    );
```

Poza pierwszym parametrem pozostałe mają wartości domyślne, więc można je w wywołaniu funkcji pominąć.

„Czysta" aplikacja przerobiona z kodu ALGO zawiera wywołanie procedury `RunAsAlgo` jako jedyny kod w programie głównym (porównaj rozdział „Szybki start").

UWAGA: Ponieważ we wnętrzu procedury `RunAsAlgo` znajduje się wywołanie `Application.Run`, każdy kod umieszczony w programie pod tym wywołaniu zostanie wykonany dopiero po zamknięciu głównego formularza GUI, albo nawet nie zostanie wykony nigdy!

Procedura dopinania dodatkowego formularza (InsertSecondaryForm)

Procedura `InsertSecondaryForm` pozwala rozbudować algopodobną aplikacje o normalne formy GUI ObjectPascala. Pozwala na zarejestrowanie jednej formy, ale nic nie stoi na przeszkodzie, żeby forma ta otwierała kolejne! Żeby zadziałała trzeba ją wywołać **PRZED** wywołaniem `RunAsALGO`.

```
Procedure InsertSecondaryForm(Form:pointer;
                              ButtName:string;
                              RunFirst:boolean=true
                              );
```

- Pierwszy parametr przyjmuje wskaźnik do zainicjowanej formy. Jest beztypowy, żeby umożliwić kompilacje głównego programu bez dodawania modułów biblioteki Delphi do klauzuli `uses` programu. Podanie błędnego wskaźnika spowoduje oczywiście „wywrócenie się" programu w czasie wykonania.
- Drugi parametr jest nazwą przycisku, który pojawi się na głównej formie i będzie służył do uruchamiania zarejestrowanej formy.
- Trzeci parametr informuje moduł czy zarejestrowana forma ma zostać uruchomiona i pokazana na początku programu (`true`), czy jedynie na żądanie (`false`) po naciśnięciu przycisku.

Dodawanie form Delphi do aplikacji ALGO wymaga już sporej znajomości programowania komponentowego. Przykład znajduje się z pakiecie, a dalsze informacje w rozdziale „Rozbudowa aplikacji o inne formy GUI".

Rozbudowa aplikacji o inne formy GUI

Do uruchomionego projektu używającego Unit_Algo można dokładać kolejne formy za pomocą kreatora GUI zawartego w Delphi. Powoduje to automatyczne tworzenie i modyfikację odpowiednich plików kodu modułów ObjectPascala (**.PAS*) oraz zasobów definiujących wygląd formularzy (**.DFM*). Jedną z takich form można podpiąć do głównej formy aplikacji za pomocą funkcji `InsertSecondaryForm` a pozostałe do siebie wzajemnie.
Podpięcie form wymaga jednak ich inicjalizacji. Normalnie dba o to kreator kodu dopisując odpowiednie fragmenty do zarządzanego przez siebie pliku projektu (**.DPR*). W przypadku pliku *DPR* skonstruowanego ręcznie z aplikacji ALGO kreator jednak się „gubi" i trzeba to zadanie wykonać za niego. Kompletny kod tak działającej aplikacji znajduje się w katalogu *Examples/AttitudeWithSetup*.

Modyfikacje sekcji uses

W sekcji `uses` trzeba dopisać wszystkie używane moduły, w razie potrzeby podając ścieżki dostępu do plików źródłowych. Np.:

```
uses
  Algo in '../../Algo.pas',
  Forms, {Konieczne do inicjacji zewnętrznej Formy: "SetupForm"}
  AttitSetup in 'AttitSetup.pas'; {SetupForm <--Ustaw parametry pracy}
```

`Forms` jest najważniejszym modułem bibliotecznym Delphi definiującym API do obiektu `Application` i do klasy `TForm`. Jego umieszczenie jest niezbędne, żeby można było wywołać kod inicjujący dla zdefiniowanych w modułach form.
`AttitSetup` jest nazwą modułu z formą zdefiniowaną z użyciem kreatora – w tym wypadku służącą do umożliwienia operatorowi programu zmiany parametrów wykonania.
Moduł dla każdej innej formy kreatora musi także zostać umieszczony na liście klauzuli `uses`!!!

Modyfikacje programu głównego

W programie głównym przed wywołaniem procedury `RunAsAlgo` trzeba umieścić kod inicjujący wszystkie używane w programie formy:

```
begin
Application.CreateForm(TSetupForm, SetupForm);
```

Inicjacja formy zdefiniowanej w module AttitSetup. Nazwa typu formy `TSetupForm` i nazwa zmiennej będącej uchwytem formy `SetupForm` są zdefiniowane w sekcji interfejsu modułu.

```
{Application.CreateForm(TOthForm, OthForm);}
```

Kolejne formy, jeśli istnieją też muszą być zainicjowane w analogiczny sposób

```
InsertSecondaryForm(SetupForm,'Setup form',true);
```

Wywołanie procedury rejestrującej dodatkową formę GUI. Można to zrobić tylko dla jednej z form. Pozostałe muszą być dostępne za jej pośrednictwem!

```
RunAsALGO(Main,'Attitude move');
end.
```

Normalne uruchomienie kodu aplikacji algopochodnej

Pełny spis treści