Budujemy aplikację w systemie ekspertowym

Krzysztof Michalik

W tym artykule poznamy zasady tworzenia aplikacji ekspertowych oraz podstawy języka reprezentacji wiedzy używanego w dołączonym na płycie systemie ekspertowym PC-Shell, wchodzącym w skład pakietu Sphinx. Stworzymy prostą aplikację dla Windows realizującą podobne zadanie, jak wiele prawdziwych systemów diagnostycznych, a mianowicie ocenę ryzyka rozwoju choroby wieńcowej (oczywiście znacznie upraszczając rzeczywiste dane medyczne).

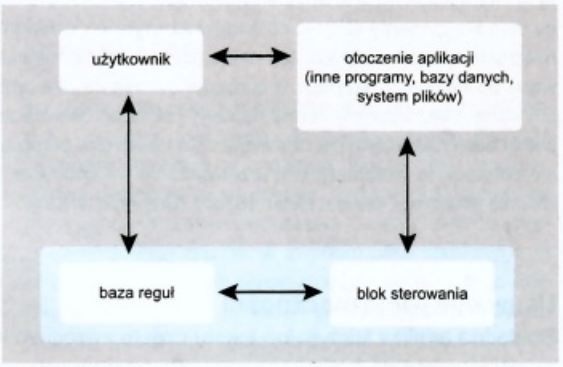
Dane wejściowe, czyli wiedza

Danymi wejściowymi dla systemu ekspertowego są odpowiednio przetworzone informacje z badanej dziedziny wiedzy – w naszym przypadku będą to dane medyczne dotyczące czynników związanych z występowaniem choroby wieńcowej. Na wzrost ryzyka wystąpienia i rozwoju tej choroby ma wpływ wiele czynników, jak np. skłonności dziedziczne, stres, wysokie ciśnienie krwi, poziom cholesterolu i wiele innych. W naszej aplikacji uwzględnimy przede wszystkim czynniki bezpośrednio mierzalne, takie jak poziom cholesterolu i ciśnienia krwi. Przyjmiemy uproszczony model diagnozowania: jeśli w kolejnych pomiarach wartości ciśnienia tętniczego przekroczą 160/90, to uznamy to za nadciśnienie, natomiast za dopuszczalny poziom cholesterolu uznamy 210 mg w 100 ml osocza. W rzeczywistości diagnozowanie zagrożenia chorobą wymaga znacznie więcej danych, wspartych fachowymi ocenami lekarzy, ale dla naszej prostej aplikacji takie uproszczone diagnozy w zupełności wystarczą.

Określenie reguł

Budowę aplikacji rozpoczynamy od zapisania wiedzy w formie reguł. Uruchamiamy program PC-Shell, klikamy menu *Edycja* i w oknie generatora nowej bazy zaznaczamy pola *Blok Faset*, *Blok Reguł* i *do Pliku*, po czym podajemy nazwę pliku dla naszej bazy (bez rozszerzenia) i klikamy *OK*. Pojawi się okno edytora, w którym będziemy mogli wprowadzać kod naszego programu. Edytor jest wyposażony w bardzo przydatną funkcję sprawdzania składni – wystarczy w menu *Plik* wybrać *Uruchom translację* lub kliknąć przycisk z błyskawicą na pasku narzędzi.

Autor założył firmę AITECH Artificial Intelligence Laboratory, którą do dziś kieruje. Jest również pracownikiem naukowym Akademii Ekonomicznej w Katowicach oraz Wyższej Szkoły Bankowej w Poznaniu. Zawodowo zajmuje się sztuczną inteligencją i inżynierią oprogramowania.
Kontakt z autorem: *pomoc@aitech.com.pl*

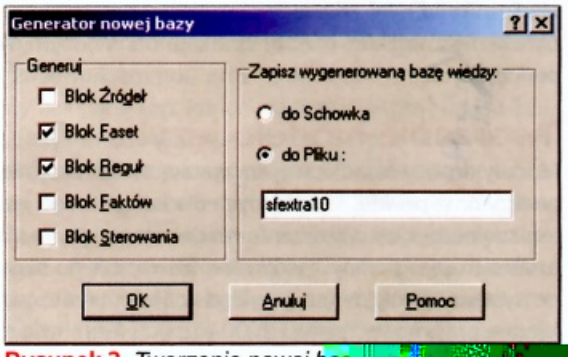


Rysunek 1. *Architektura aplikacji tworzonych w PC-Shell*

Baza wiedzy jest umieszczana w bloku instrukcji `knowledge base nazwa...end`, w ramach którego najpierw zapisujemy definicje wszystkich używanych zmiennych (blok `facets...end`), a następnie same reguły (blok `rules...end`). W ramach reguł określamy związki logiczne między pojęciami i wartościami zdefiniowanymi w bloku `facets...end`. Składnia definicji i reguł jest następująca: jeśli chcemy określić regułę, według której uzyskanie odpowiedzi *tak* dla zmiennych `warunek1` i `warunek2` ma zwracać na temat wyniku `wynik` informację o treści *występuje*, to odpowiedni blok reguł może wyglądać następująco:

```
wynik = "występuje" if warunek1 = "tak", ↩
    warunek2 = "tak";
```

Kompletny kod bazy reguł dla naszej przykładowej aplikacji widzimy na Listingu 1. Do pobierania danych używamy operatora zapytania `query`, zadającego użytkownikowi podane pytanie, oraz dwóch operatorów ograniczających zakres dopuszczal-



Rysunek 2. *Tworzenie nowej ba*

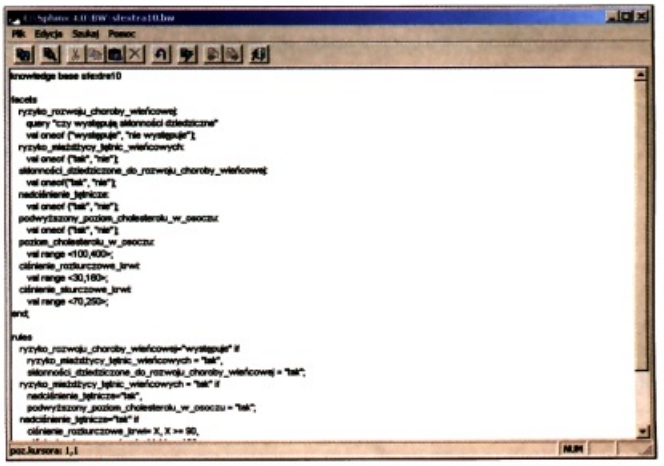
Listing 1. *Baza wiedzy naszej aplikacji*

```
knowledge base sfextra10

facets
   ryzyko_rozwoju_choroby_wieńcowej:
      query "czy występują skłonności dziedziczne"
      val oneof {"występuje", "nie występuje"};
   ryzyko_miażdżycy_tętnic_wieńcowych:
      val oneof {"tak", "nie"};
   skłonności_dziedziczone_do_rozwoju_choroby_wieńcowej:
      val oneof{"tak", "nie");
   nadciśnienie_tętnicze:
      val oneof {"tak", "nie"};
   podwyższony_poziom_cholesterolu_w_osoczu:
      val oneof {"tak", "nie"};
   poziom_cholesterolu_w_osoczu:
      val range <100,400>;
   ciśnienie_rozkurczowe_krwi:
      val range <30,160>;
   ciśnienie_skurczowe_krwi:
      val range <70,250>;
end;

rules
   ryzyko_rozwoju_choroby_wieńcowej="występuje" if
      ryzyko_miażdżycy_tętnic_wieńcowych = "tak",
      skłonności_dziedziczone_do_rozwoju_choroby_←
         wieńcowej  = "tak";
   ryzyko_miażdżycy_tętnic_wieńcowych = "tak" if
      nadciśnienie_tętnicze="tak",
      podwyższony_poziom_cholesterolu_w_osoczu = "tak";
   nadciśnienie_tętnicze="tak" if
      ciśnienie_rozkurczowe_krwi= X, X >= 90,
      ciśnienie_skurczowe_krwi = Y, Y >= 160;
   podwyższony_poziom_cholesterolu_w_osoczu = "tak" if
      poziom_cholesterolu_w_osoczu = X, X >= 210;
end;

end;
```

nych odpowiedzi: `val oneof {...}`, akceptującego wyłącznie jedną z możliwości podanych w nawiasach, oraz `val range <...>`, akceptującego tylko wartości liczbowe z podanego zakresu. Dodatkowo korzystamy ze zmiennych `x` i `y`, których wartości program pobierze od użytkownika za pomocą odpowiedniego okna dialogowego.

Tak przygotowana baza wiedzy jest już sama w sobie gotową aplikacją. Aby ją uruchomić, zapisujemy naszą bazę wiedzy, zamykamy okno edytora, z menu *Plik* wybieramy *Otwórz* i wskazujemy plik naszej bazy. Teraz wybieramy *Wnioskowanie -> Do tyłu*, co spowoduje otworzenie okna dialogowego *Hipoteza*. W tym oknie formułujemy hipotezę, która ma być sprawdzona na podstawie informacji zawartych w bazie wiedzy – w naszym przypadku chcemy się dowiedzieć, czy dla danego pacjenta występuje ryzyko choroby wieńcowej, więc wpisujemy:

```
ryzyko_rozwoju_choroby_wieńcowej="występuje"
```



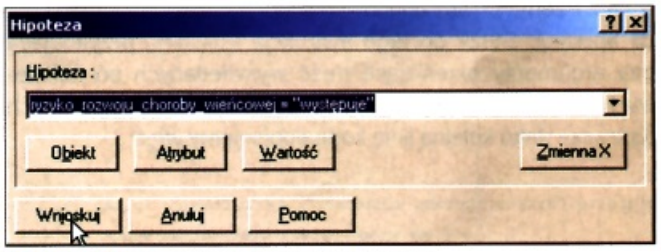
Rysunek 3. *Okno edytora bazy wiedzy*

Zamiast wpisywać hipotezę możemy wybrać oba jej elementy z list dostępnych po naciśnięciu przycisków *Atrybut* i *Wartość*. Klikamy *Wnioskuj*. Teraz program będzie nas prosił o podawanie wartości zmiennych użytych w bazie reguł. Zauważmy, że dane są przetwarzane sekwencyjnie i system zadaje nam tylko tyle pytań, ile jest mu potrzebne do weryfikacji hipotezy – jeśli już po podaniu pierwszej wartości będzie jasne, że ryzyka nie ma, to wynik zostanie wyświetlony bez dalszych pytań. Wynik jest wyświetlany jako komunikat o treści *hipoteza potwierdzona* lub *hipoteza niepotwierdzona*. Taki tryb pracy na ogół wystarcza na etapie budowy programu czy testowania reguł, ale nie jest zbyt wygodny. Dodanie bardziej zaawansowanych funkcji czy chociażby obsługi interfejsu użytkownika wymaga dołączenia do programu bloku sterowania `control...end`, umożliwiającego algorytmiczne sterowanie pracą samego systemu regułowego.

Blok sterowania

Blok sterowania `control...end` jest trzecim (po `facets` i `rules`) blokiem w kodzie bazy wiedzy. Od pozostałych dwóch bloków różni się tym, że jest nieobowiązkowy i ma charakter proceduralny, czyli umożliwia algorytmiczny zapis przepływu sterowania w programie. W bloku sterowania używany jest wyspecjalizowany język programowania wysokiego poziomu, stworzony specjalnie dla potrzeb programu PC-Shell w celu ułatwienia obsługi specyficznych wymagań aplikacji w nim tworzonych. Poza funkcjami sterującymi, język ten dostarcza instrukcji i typów danych niezbędnych w systemach ekspertowych, umożliwia stworzenie interfejsu użytkownika, pozwala integrować tworzoną aplikację z innymi programami i daje możliwość efektywnego opisywania wiedzy o charakterze algorytmicznym.

Ciekawą cechą architektury aplikacji tworzonych w PC-Shell jest dwukierunkowy przepływ wiedzy pomiędzy bazą re-



Rysunek 4. *Okno formułowania hipotezy*

Listing 2. *Blok sterujący naszej aplikacji*

```
control

   char Ocena;

   run;
   createAppWindow;
   setAppWinTitle("Ryzyko choroby wieńcowej");
   setSysText(notConfirmed,"Brak podstaw do uznania, że ↩
       występuje ryzyko choroby wieńcowej");
   setSysText(problem,"Ocena ryzyka wystąpienia choroby
                       wieńcowej");
   vignette("Ryzyko choroby wieńcowej", "Przykładowa ↩
       aplikacja do oceny ryzyka wystąpienia choroby ↩
       wieńcowej", "AITECH");
   menu "Diagnoza"
      1. "Konsultacja"
      2. "Wyjście"
   case 1:
      delNewFacts;
      goal( "ryzyko_rozwoju_choroby_wieńcowej = Ocena" );
   case 2:
      exit;
   end;

end;
```

guł, blokiem sterowania i otoczeniem aplikacji. Oznacza to, że fakty z bazy wiedzy mogą być przekazywane do zmiennych w bloku sterowania, poddane odpowiedniej modyfikacji na podstawie komunikacji z otoczeniem (na przykład z innym programem) i ponownie zapisane w bazie wiedzy. Uproszczoną ilustrację tej architektury przedstawia Rysunek 1.

Interfejs użytkownika

Budowę interfejsu użytkownika dla naszej aplikacji rozpoczynamy od otwarcia okna aplikacji w systemie Windows. W tym celu w kodzie naszej aplikacji tworzymy blok `control...end` i wpisujemy w nim linię:

```
createAppWindow;
```

Instrukcja ta utworzy nowe okno o standardowym tytule *Aplikacja systemu PC-Shell*. Teraz zmienimy ten tytuł za pomocą funkcji `setAppWinTitle`:

```
setAppWinTitle("Ryzyko choroby wieńcowej");
```

W bardzo szybki i łatwy sposób możemy dodać ekran powitalny aplikacji (tzw. *splashscreen*), będący oknem dialogowym z trzema polami tekstowymi, których treść definiuje twórca aplikacji. Służy do tego instrukcja `vignette`, przyjmująca trzy argumenty określające treść wyświetlanych pól tekstowych. Napisy możemy podać bezpośrednio lub w zmiennych typu `char`. Jako kolejną linię kodu dopisujemy więc:

```
vignette("Ryzyko choroby wieńcowej", "Przykładowa aplikacja
                    do oceny ryzyka wystąpienia choroby
                    wieńcowej", "AITECH");
```

Automatyczne uruchomienie wnioskowania

Wiemy już, że bazę reguł można uruchamiać z menu systemu PC-Shell, ale nie jest to rozwiązanie zbyt wygodne dla użytkownika. Jeśli z góry wiemy, jakie hipotezy będziemy potwierdzać i jakich rozwiązań poszukujemy, to znacznie wygodniej jest umieścić w bloku sterowania instrukcję `goal`, która automatycznie uruchamia proces wnioskowania stosując wnioskowanie wstecz (czyli robi dokładnie to, co do tej pory robiliśmy ręcznie). Jako argument funkcja przyjmuje hipotezę, która ma być sprawdzona – będzie to dokładnie ta sama hipoteza, którą dotychczas podawaliśmy ręcznie w pierwszym oknie dialogowym wnioskowania. Dodajemy zatem kolejną linię do instrukcji bloku sterowania:

```
goal("ryzyko_rozwoju_choroby_wieńcowej = Ocena");
```

gdzie `Ocena` jest nazwą zmiennej znakowej, którą musimy uprzednio zadeklarować poleceniem `char Ocena`. Teraz do rozwiązania problemu wystarczy z menu *Wnioskowanie* wybrać opcję *Wykonanie programu* lub nacisnąć klawisz [*F9*].

Nasza aplikacja już działa, ale zwracany przez nią wynik ma postać niezbyt zrozumiałego komunikatu *hipoteza potwierdzona* lub *hipoteza niepotwierdzona*, a rozwiązywany problem jest przedstawiany w równie mało mówiącej postaci. Możemy to zmienić za pomocą instrukcji `setSysText`, która jako argumenty przyjmuje typ i treść komunikatu. Możemy w ten sposób zmodyfikować dwa komunikaty: opis zadania (jako typ podajemy `problem`) oraz wiadomość o niepotwierdzonej hipotezie (jako typ podajemy `notConfirmed`). Przed instrukcją `goal` w naszym kodzie wstawiamy zatem następujące dwie linie:

```
setSysText(notConfirmed,"Brak podstaw do uznania, że występuje
                    ryzyko choroby wieńcowej");

setSysText(problem,"Ocena ryzyka wystąpienia choroby wień-
cowej");
```

Automatyczne uruchomienie aplikacji

Korzystanie z naszego programu jest nadal dość niewygodne, gdyż za każdym razem po załadowaniu go do PC-Shella trzeba go ręcznie uruchomić z menu *Wnioskowanie* lub klawiszem [*F9*]. Problem ten rozwiązuje bezargumentowa instrukcja `run`, która może być zapisana w dowolnym miejscu bloku sterowania – dla większej czytelności kodu dopiszemy

Rysunek 5. *Wynik wnioskowania*

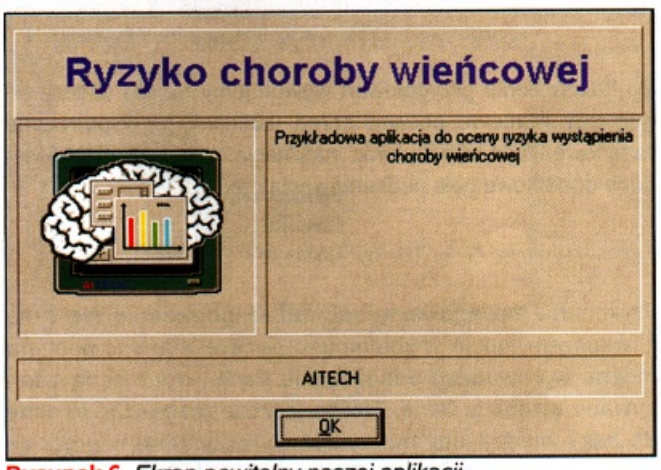
Listing 3. *Przykładowa obsługa arkusza danych*

```
createAppWindow;
// otwarcie arkusza
openSheet("Dane wejściowe", "krew.vts");
menu "Menu"
   1. "Pokaż arkusz"
   2. "Konsultacja"
   3. "Wyjście"
  case 1:
   // wyświetlenie arkusza do wprowadzania danych
      showSheet("Dane wejściowe", 1 );
   case 2:
      float Dane[3];
      getSheetValue("Dane wejściowe","",1,1, Dane[0]);
      getSheetValue("Dane wejściowe","",2,1, Dane[1]);
      getSheetValue("Dane wejściowe","",3,1, Dane[2]);
      // w tym miejscu uruchomilibyśmy wnioskowanie
   case 3:
      exit;
end;
// zamknięcie arkusza
closeSheet("Dane wejściowe");
```

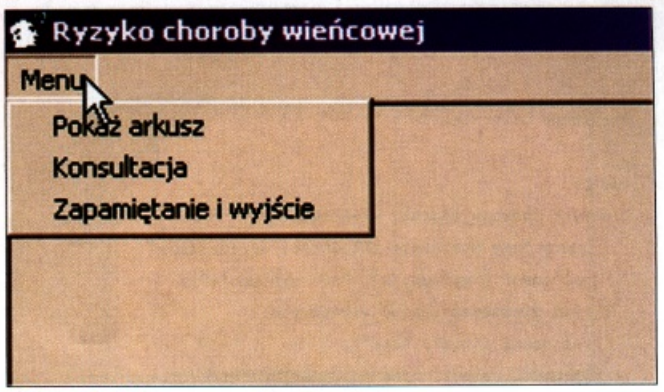
ją na początku naszego programu. Teraz nasz program będzie się automatycznie uruchamiał po otwarciu jego pliku z menu *Plik*.

Dodajemy menu

Obecna postać naszej aplikacji ma dość istotną wadę: nie można wykonać kilku konsultacji bez ponownego uruchamiania programu. Najprostszym rozwiązaniem będzie w tym przypadku dodanie menu zawierającego opcję *Konsultacja*, której wybranie spowoduje wyczyszczenie dotychczasowych wyników i uruchomienie wnioskowania od nowa. Do tworzenia menu służy instrukcja `menu...end`, stanowiąca pewien rodzaj instrukcji warunkowej. Jako parametr podajemy nazwę menu, a następnie podajemy kolejne pozycje menu, numerując je od jedynki. Teraz możemy określać działanie programu po wybraniu poszczególnych opcji, umieszczając odpowiednie polecenia po instrukcjach `case numer_opcji:` . Dodamy tylko dwie opcje: *Konsultacja* i *Wyjście*. Wybranie pozycji *Konsultacja*



Rysunek 6. *Ekran powitalny naszej aplikacji*



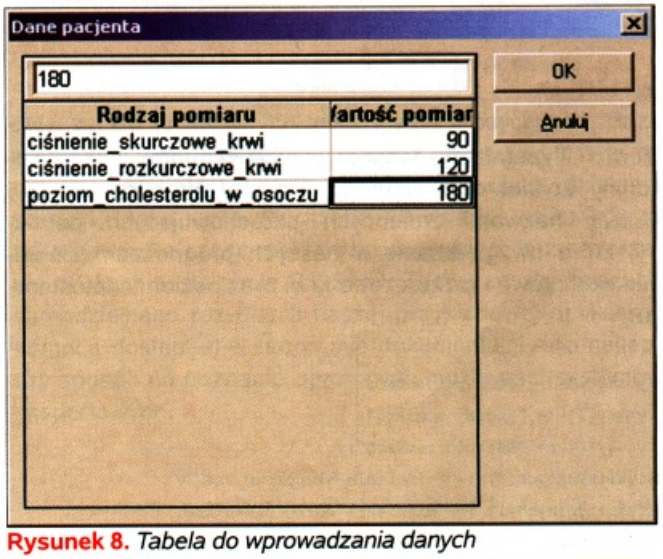
Rysunek 7. *Menu naszej aplikacji*

spowoduje wykasowanie wszelkich zapamiętanych informacji o wartościach zmiennych (polecenie `delNewFacts`) i uruchomienie wnioskowania za pomocą znanej już nam instrukcji `goal`. Wybranie opcji *Wyjście* powoduje wywołanie wbudowanej funkcji `exit`, kończącej pracę programu. Na Listingu 2 widzimy nasz blok danych wzbogacony o wszystkie omówione dotychczas elementy, włącznie z menu.

Tabela wprowadzania danych

Do tej pory wszystkie niezbędne systemowi dane były przekazywane przez użytkownika za pomocą odpowiednich okien dialogowych, generowanych automatycznie w procesie wnioskowania. Takie rozwiązanie jest wystarczające do testowania programu, ale ten sposób wprowadzania danych nie jest zbyt wygodny dla użytkownika, zwłaszcza przy większych ilościach danych. Dużo wygodniejszym i bardziej przejrzystym rozwiązaniem jest zastosowanie prostej tabeli (arkusza) do wprowadzania danych. Wykorzystamy do tego celu instrukcję `nsheetBox`, która służy do utworzenia prostej tabeli o dwóch kolumnach. Najpierw musimy zadeklarować trzy tablice, które będą argumentami tej instrukcji: jedną tablicę liczb typu `float`, która będzie przechowywać dane i dwie tablice znakowe (typu `char`), z których jedna będzie zawierać nazwy zmiennych, a druga tytuł okna oraz kolumn tabeli. Tablice te deklarujemy następująco:

```
float Dane[3];
char Tytuły[3], EtykietyDanych[3];
```



Rysunek 8. *Tabela do wprowadzania danych*

Listing 4. *Kompletny kod aplikacji*

```
knowledge base sfextra10

facets
  ryzyko_rozwoju_choroby_wieńcowej:
    query "czy występują skłonności dziedziczne"
    val oneof {"występuje", "nie występuje"};
  ryzyko_miażdżycy_tętnic_wieńcowych:
    val oneof {"tak", "nie"};
  skłonności_dziedziczone_do_rozwoju_choroby_wieńcowej:
    val oneof{"tak", "nie"};
  nadciśnienie_tętnicze:
    val oneof {"tak", "nie"};
  podwyższony_poziom_cholesterolu_w_osoczu:
    val oneof {"tak", "nie"};
  poziom_cholesterolu_w_osoczu:
    val range <100,400>;
  ciśnienie_rozkurczowe_krwi:
    val range <30,160>;
  ciśnienie_skurczowe_krwi:
    val range <70,250>;
end;

rules
  ryzyko_rozwoju_choroby_wieńcowej="występuje" if
    ryzyko_miażdżycy_tętnic_wieńcowych = "tak",
    skłonności_dziedziczone_do_rozwoju_choroby_wieńcowej =
                        "tak";
  ryzyko_miażdżycy_tętnic_wieńcowych = "tak" if
    nadciśnienie_tętnicze="tak",
    podwyższony_poziom_cholesterolu_w_osoczu = "tak";
  nadciśnienie_tętnicze="tak" if
    ciśnienie_rozkurczowe_krwi= X, X >= 90,
    ciśnienie_skurczowe_krwi = Y, Y >= 160;
  podwyższony_poziom_cholesterolu_w_osoczu = "tak" if
    poziom_cholesterolu_w_osoczu = X, X >= 210;
end;
control
  char Ocena, S;
  float Dane[3];
  run;
  setSysText( problem, "Ocena ryzyka wystąpienia ←
    choroby wieńcowej" );
  setSysText( notConfirmed,"Brak podstaw do uznania, że←
  występuje ryzyko choroby wieńcowej" );
  createAppWindow;
  setAppWinTitle("Ryzyko choroby wieńcowej");
  vignette("Ryzyko choroby wieńcowej", "Przykładowa←
      aplikacja do oceny ryzyka wystąpienia choroby←
      wieńcowej", "AITECH");
  openSheet( "Dane wejściowe", "krew.vts");
  getProfile("Krew","CiśnienieRozkurczowe",S,"","krew.ini");
  setSheetValue( "Dane wejściowe","",1,1, S );
  getProfile("Krew","CiśnienieSkurczowe",S,"","krew.ini");
  setSheetValue( "Dane wejściowe","",2,1, S );
  getProfile("Krew","PoziomCholesterolu",S,"","krew.ini");
  setSheetValue( "Dane wejściowe","",3,1, S );
  menu "Menu"
    1. "Pokaż arkusz"
    2. "Konsultacja"
    3. "Zapamiętanie i wyjście"
  case 1:
    showSheet( "Dane wejściowe", 1 );
  case 2:
    delNewFacts;
    getSheetValue( "Dane wejściowe","",1,1, Dane[0] );
    getSheetValue( "Dane wejściowe","",2,1, Dane[1] );
    getSheetValue( "Dane wejściowe","",3,1, Dane[2] );
    addFact( _, ciśnienie_rozkurczowe_krwi, Dane[0] );
    addFact ( _, ciśnienie_skurczowe_krwi, Dane[1] );
    addFact (_ , poziom_cholesterolu_w_osoczu, Dane[2] );
    goal("ryzyko_rozwoju_choroby_wieńcowej = Ocena" );
  case 3:
    getSheetValue( "Dane wejściowe","",1,1, S );
    writeProfile("Krew", CiśnienieRozkurczowe",S, krew.ini");
    getSheetValue( "Dane wejściowe","",2,1, S );
    writeProfile("Krew","CiśnienieSkurczowe",S,"krew.ini");
    getSheetValue( "Dane wejściowe","",3,1, S );
    writeProfile("Krew","PoziomCholesterolu",S,"krew.ini");
    exit;
  end;
end;

end;
```

Teraz wypełniamy tablicę `Tytuły` tytułami okna, kolumny etykiet oraz kolumny danych, a tablicę `Etykiety Danych` nazwami zmiennych przechowujących pomiary, które uwzględniamy w naszych prognozach (ciśnienie skurczowe i rozkurczowe krwi oraz poziom cholesterolu):

```
Tytuły[0] := "Dane pacjenta";
Tytuły[1] := "Rodzaj pomiaru";
Tytuły[2] := "Wartość pomiaru";
EtykietyDanych[0] := "ciśnienie_skurczowe_krwi";
EtykietyDanych[1] := "ciśnienie_rozkurczowe_krwi";
EtykietyDanych[2] := "poziom_cholesterolu_w_osoczu";
```

Po zakończeniu przygotowań wywołujemy tabelę (cztery argumenty liczbowe podane przed tablicami to współrzędne ekranowe, ilość wierszy oraz informacja, czy ma być wyświetlone dodatkowe pole podsumowania):

```
nsheetBox( 0, 0, 3, 0, Tytuły, EtykietyDanych, Dane);
```

Wykonanie takiego kodu spowoduje pojawienie się okna z arkuszem, gdzie w kolumnie o nazwie *Wartość pomiaru* można wprowadzać odpowiednie dane, które będą zapisywane w tablicy *Dane*. Trzeba jeszcze przekazać te dane do bazy wiedzy, aby mogły być wykorzystane w procesie wnioskowania jako fakty. Posłuży nam do tego instrukcja

addFact, przyjmująca trzy argumenty: identyfikator obiektu (jeśli występuje – w przeciwnym razie podajemy w tym miejscu znak podkreślenia), atrybut oraz wartość. Wykonanie tej instrukcji powoduje utworzenie faktu w postaci trójki obiekt-atrybut-wartość i wprowadzenie go do bazy wiedzy, dzięki czemu staje się on widoczny dla modułu wnioskującego. Do naszej aplikacji dodajemy zatem następujący kod:

```
addFact( _, EtykietyDanych[0], Dane[0] );
addFact( _, EtykietyDanych[1], Dane[1] );
addFact( _, EtykietyDanych[2], Dane[2] );
```

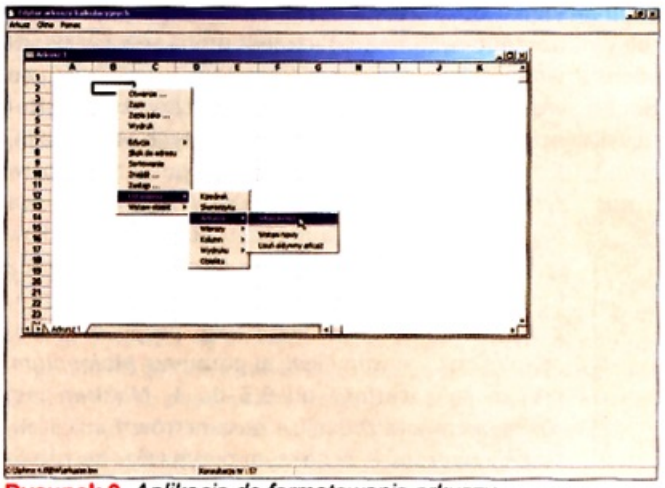
Obsługa arkusza danych

Tabelka danych jest już znacznie wygodniejszym rozwiązaniem od podawania danych w oknie dialogowym, ale nadal wymaga ręcznego wprowadzania informacji. Bardziej elastycznym rozwiązaniem jest wykorzystanie arkuszy kalkulacyjnych w dużym stopniu zgodnych z arkuszami kalkulacyjnymi Microsoft Excel. Chcąc wykorzystać arkusz kalkulacyjny w aplikacji musimy na początek stworzyć jego stronę wizualną. W tym celu możemy wykorzystać gotową aplikację *arkusze.bw*, która umożliwia właśnie tworzenie i edycję arkuszy, które potem mogą być wykorzystywane w naszych programach. Po jej otwarciu wybieramy z menu *Arkusz* opcję *Nowy*. Po utworzeniu nowego arkusza pod prawym przyciskiem myszki dostępne są opcje formatowania jego zawartości, za pomocą których możemy dostosować wygląd i zachowanie arkusza do własnych potrzeb.

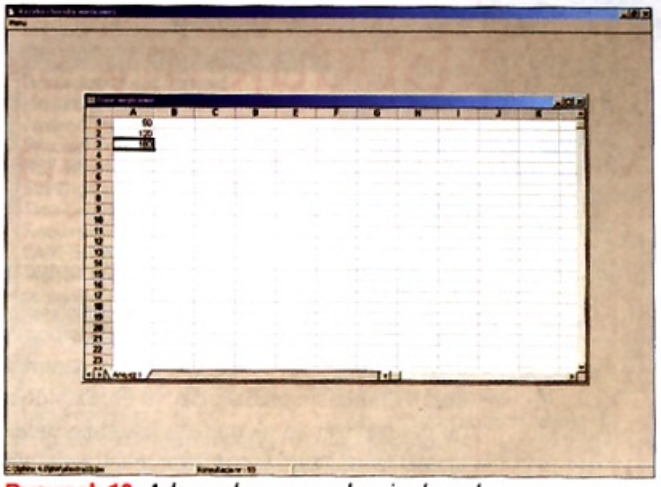
Tak przygotowany arkusz zapisujemy w pliku w macierzystym formacie *.vts* lub w formacie Excela *.xls*. Pora dodać arkusz do naszej aplikacji za pomocą instrukcji openSheet. Musi on zostać załadowany do pamięci przed wejściem w pętlę menu, więc następującą instrukcję dodajemy przed blokiem menu:

```
openSheet("Dane wejściowe", "krew.vts");
```

gdzie pierwszym parametrem jest nazwa arkusza, a drugim nazwa pliku z utworzonym przez nas wzorcem arkusza. Instrukcja openSheet ładuje tylko arkusz do pamięci, ale nie wyświetla go – na to przyjdzie pora później. Trzeba



Rysunek 9. *Aplikacja do formatowania arkuszy*



Rysunek 10. *Arkusz do wprowadzania danych*

pamiętać, że instrukcje operujące na arkuszach wymagają istnienia głównego okna aplikacji, czyli w kodzie powinny być umieszczane dopiero po instrukcji createAppWindow. Gdy użytkownik zażąda wyświetlenia arkusza, wywołujemy funkcję showSheet, która otworzy nowe okno ze wskazanym arkuszem, do którego można teraz wprowadzać wartości zmiennych używanych w aplikacji – w naszym przykładzie będą to odpowiednio ciśnienie skurczowe, ciśnienie rozkurczowe i poziom cholesterolu w pierwszych trzech wierszach pierwszej kolumny arkusza. Podane przez użytkownika wartości pobieramy za pomocą funkcji getSheetValue, przyjmującej jako parametry nazwę arkusza, nazwę skoroszytu, wiersz, kolumnę i zmienną, do której należy zapisać dane ze wskazanej komórki. Po zamknięciu programu (czyli po wyjściu z bloku menu) zamykamy arkusz instrukcją closeSheet . Listing 3 ilustruje szkieletowy kod takiej obsługi arkusza.

Na razie dane wprowadzane do arkusza są jednorazowe – są tracone przy każdym zamknięciu programu. Do zapamiętania i odtworzenia ostatnio wprowadzonych dane służą instrukcje getProfile, writeProfile oraz setSheetValue. Pierwsze dwie wczytują i zapisują dane z i do wskazanego źródła danych (w naszym przypadku pliku tekstowego), natomiast ostatnia zapisuje te dane w arkuszu. Kompletny kod naszej aplikacji wraz z obsługą arkusza danych ilustruje Listing 4.

Co dalej?

Stworzona przez nas aplikacja jest bardzo prosta i nie nadaje się do prawdziwych zastosowań medycznych. Bazy wiedzy systemów eksperckich wykorzystywanych w praktyce są znacznie bardziej złożone, ale zasada działania pozostaje taka sama. Pokazane w tym artykule funkcje to tylko niewielka część możliwości dostępnych przy tworzeniu aplikacji w systemie PC-Shell. Zachęcam do dalszego samodzielnego rozwijania naszej przykładowej aplikacji z pomocą dostępnej w programie dokumentacji – to najlepszy sposób na poznanie ogromnych możliwości systemów ekspertowych. ■