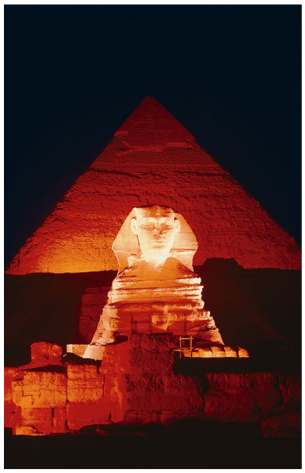
AITECH®
Artificial Intelligence Laboratory
Rok założenia 1990



Krzysztof Michalik

Program dydaktyczny do pakietu Sphinx

Kurs podstawowy

Katowice, 2003

Spis treści:

1. Założenia do przykładowej aplikacji

Pretekstem do prezentacji wybranych mechanizmów języka reprezentacji wiedzy pakietu Sphinx będzie budowa prostej aplikacji z zakresu kardiologii, przy oczywistym założeniu co do faktu, że prezentowana wiedza medyczna nie musi być poprawna, a już na pewno kompletna.

Cel aplikacji jest następujący: ocena ryzyka rozwoju choroby wieńcowej

Założenia merytoryczne:

Miażdżyca tętnic, objawia się zwężeniem tętnic wieńcowych. Na wzrost ryzyka wystąpienia choroby ma wpływ wiele czynników, np. skłonności dziedziczne, stres, używki i inne, które tu pomijamy. Ograniczamy się tylko do czynników mierzalnych (ilościowych), takich jak poziom cholesterolu i ciśnienia krwi.

Czasami za wartość graniczną dla umiarkowanego nadciśnienia tętniczego przyjmuje się 160 mm Hg ciśnienia skurczowego i 95 mm Hg ciśnienia rozkurczowego. Przyjmiemy jednocześnie, że poziom cholesterolu nie powinien przekraczać 210 mg w 100 ml osocza. Jeszcze raz podkreślamy, że naszym celem jest ilustracja sposobu budowy aplikacji, a nie instruktaż medyczny!

Budowę aplikacji rozpoczynamy od zapisania podanej wiedzy w formie reguł, które muszą znaleźć się w bloku reguł, poprzedzonym blokiem faset, w którym definiujemy wszystkie użyte atrybuty i wartości. Wszystkie bloki aplikacji są zawarte w ogólnym bloku ***knowledge base***.

2. Baza reguł przykładu SE

```
knowledge base przykład1
 facets
      ryzyko_rozwoju_choroby_wieńcowej:
            query "czy występują skłonności dziedziczne"
            val oneof{"występuje","nie występuje"};

      ryzyko_miażdzycy_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_miażdzycy_tętnic_wieńcowych="tak",
            skłonności_dziedziczone_do_rozwoju_choroby_wieńcowej="tak";

      ryzyko_miażdzycy_tętnic_wieńcowych ="tak" if
            nadciśnienie_tętnicze="tak",
            podwyższony_poziom_cholesterolu_w_osoczu="tak";

      nadciśnienie_tętnicze="tak" if
            ciśnienie_rozkurczowe_krwi=X,
            X >= 95,
            ciśnienie_skurczowe_krwi=Y,
            Y >= 140;

      podwyższony_poziom_cholesterolu_w_osoczu="tak" if
            poziom_cholesterolu_w_osoczu=X,
            X >= 210;
 end;
end;
```

Tak przygotowana baza reguł (wiedzy) jest już gotową aplikacją, możliwą do uruchomienia w trybie interakcyjnym (opcje systemu PC-Shell: `wnioskowanie | wnioskowanie wstecz`). Wymienione opcje otwierają okno dialogowe „Hipoteza", w którym możemy sformułować problem w formie np. weryfikacji hipotezy:

`ryzyko_rozwoju_choroby_wieńcowej="występuje"` lub żądania znalezienia rozwiązania przez użycie zmiennej jako wartości:

`ryzyko_rozwoju_choroby_wieńcowej=Ocena_ryzyka`

Taki tryb pracy jest na ogół wystarczający na etapie budowy prototypu demonstracyjnego, badawczego lub testowania samych reguł. Nie wystarcza to jednak do osiągnięcia bardziej zaawansowanej funkcjonalności i elastyczności aplikacji lub większej atrakcyjności oraz ergonomii interfejsu użytkownika. Do realizacji m.in. takich cech tworzonych aplikacji wprowadzono blok ***control,*** który zasadniczo jest programem algorytmicznym, sterującym pracą dziedzinowego SE, opracowanego w środowisku systemu PC-Shell.

3. Blok *control* i jego rola

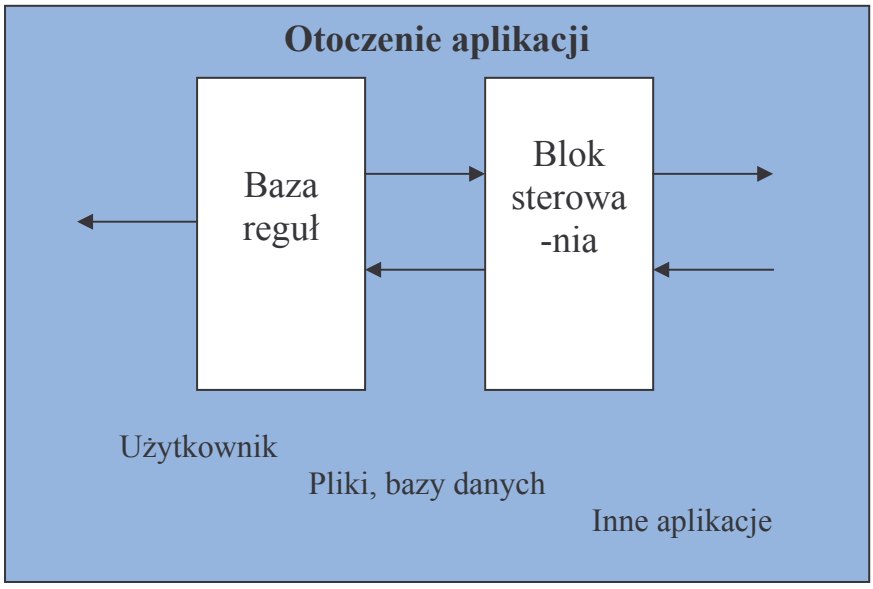
Blok ***control*** jest ostatnim – jednocześnie opcjonalnym – blokiem struktury opisu bazy wiedzy. Zatem w strukturze kodu aplikacji blok ten jest umieszczony w następujący sposób:

```
knowledge base
   ...
   facets
      ...
   end;
   rules
      ...
   end;
   ...
   control
      program
   end;
end;
```

Z formalnego punktu widzenia odróżnia się od bloku faset i reguł, mających charakter deklaratywny, podejściem algorytmicznym do opisu problemu, ma zatem charakter imperatywny. W sensie praktycznym, jego wykorzystanie przypomina programowanie w konwencjonalnych językach takich jak C lub Pascal. Język ten dostarcza bowiem podobnych typów, struktur danych i instrukcji jak wspomniane znane języki. Jednakże jego celem nie było opracowanie kolejnego języka programowania lub stworzenia „konkurencji" dla tych dobrze znanych, lecz dostarczenie użytkownikowi narzędzia, które realizuje przynajmniej następujące cele:

- umożliwia efektywny opis problemów, dla których podejście algorytmiczne jest bardziej odpowiednie,
- zapewnia automatyczne sterowanie pracą SE,
- dostarcza specjalnych instrukcji i typów danych niezbędnych z punktu widzenia specyfiki technologii SE,
- dostarcza narzędzi do budowy przyjaznych interfejsów,
- daje możliwość aplikacji otwartych, tzn. mogących względnie łatwo wymieniać informacje z otoczeniem, w szczególności z innymi systemami,
- zapewnia możliwość budowy aplikacji hybrydowych.

Wydaje się, że w tym miejscu ważne jest przedstawienie wzajemnych relacji pomiędzy bazą reguł i blokiem sterowania (***control***) oraz światem zewnętrznym. Należy podkreślić, że informacje (fakty) z bazy wiedzy mogą być przekazywane do zmiennych w bloku sterowania, dane zaś zawarte w tym bloku (np. jako wartości zmiennych) mogą być (po automatycznej konwersji do postaci faktów) wprowadzone do bazy wiedzy. Mamy zatem możliwość realizacji pełnego, dwukierunkowego przepływu danych (wiedzy) pomiędzy tymi blokami, a wszystko za sprawą odpowiednich instrukcji języka bloku sterowania. Jeśli do tego dodamy, że język ten zawiera instrukcje do komunikacji z otoczeniem (pliki, bazy danych, inne aplikacje, interfejs użytkownika), to uzyskujemy pełną otwartość, co wyraża się m.in. w tym, że fakty z bazy reguł mogą być przekazane - z pośrednictwem bloku sterowania – do otoczenia aplikacji i odwrotnie. Schematycznie zilustrowano to na rys. 1.



Rys. 1. Schemat przepływu danych/wiedzy w systemie PC-Shell

Należy podkreślić, że strzałki wychodzące z bazy reguł do otoczenia oznaczają tu wymianę danych pomiędzy użytkownikiem a bazą reguł, realizowaną automatycznie przez standardowy interfejs systemowy, w trybie pracy interakcyjnej (tj. bez użycia programu sterującego z bloku ***control***).

4. Okno aplikacji dla Windows – instrukcja *createAppWindow*

Budowę aplikacji rozpoczynamy od otwarcia okna aplikacji w systemie Windows. Dzięki temu zakryte będzie systemowe okno i menu systemu PC-Shell. Standardowo jako tytuł okna pojawia się napis „Aplikacja systemu PC-Shell". Do tego celu służy instrukcja o następującej postaci:

```
createAppWindow;
```

5. Ustalenie tytułu okna – instrukcja *setAppWinTitle*

Dzięki tej instrukcji mamy możliwość ustalić tytuł okna aplikacji, utworzonego w poprzednim kroku. W ten sposób standardowy tytuł „Aplikacja systemu PC-Shell" możemy zastąpić nazwą wybraną przez nas, w tym przypadku „Ryzyko choroby wieńcowej". W tym celu dodajemy instrukcję o postaci:

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

6. Dodanie winiety aplikacji – instrukcja *vignette*

W bardzo szybki i łatwy sposób możemy dodać winietę aplikacji w postaci szarego okna dialogowego z trzema polami tekstowymi, których treść definiuje twórca aplikacji. Wystarczy użyć jednej instrukcji ***vignette*****,** z trzema argumentami ustalającymi treść wspomnianych trzech pól winiety. Choć argumenty mogą być zmiennymi typu ***char***, w naszym przypadku wystarczy użycie stałych tekstowych, np. o postaci:

```
vignette(”Ryzyko choroby wieńcowej”,
         ”Przykładowa aplikacja do oceny ryzyka wystąpienia choroby
          wieńcowej”, ”© 2003 AITECH”);
```

7. Automatyczne uruchomienie wnioskowania – instrukcja *goal*

Jak już wspomniano, bazę reguł można uruchomić interakcyjnie, z poziomu menu systemu PC-Shell. Takie rozwiązanie ma jednak wiele wad, przede wszystkim zabiera czas i wymaga ingerencji użytkownika dla rozpoczęcia procesu wnioskowania i potwierdzenia hipotezy lub znalezienia innych rozwiązań – ogólnie rozwiązania problemu. Jeśli jesteśmy w stanie „z góry" przewidzieć jakie cele (hipotezy) będziemy potwierdzać lub jakich rozwiązań

poszukujemy, wygodniej jest użyć instrukcji ***goal***, która automatycznie uruchamia proces wnioskowania (w uproszczeniu rozwiązywania) problemu. Jej argumentem jest cel (hipoteza), którą na poziomie menu systemowego wpisywaliśmy ręcznie. Ogólnie schemat tej instrukcji wygląda następująco (zob. dokumentacja systemu PC-Shell):

goal(*problem* **);**

W naszym przykładzie będzie to problem złożony z dwójki <A,W> (atrybut-wartość), gdzie atrybut został nazwany: ryzyko_rozwoju_choroby_wieńcowej, a wartość określimy zmienną, za którą podstawi się poszukiwane rozwiązanie, np."występuje". Ostatecznie ten fragment bloku sterowania będzie wyglądał następująco:

goal(”ryzyko_rozwoju_choroby_wieńcowej = Ocena” **);**

gdzie ”Ocena” jest identyfikatorem (nazwą) zmiennej, rozpoczynającym się od dużej litery. Ta zmienna musi być zadeklarowana – w tym przypadku – jako typ znakowy, bowiem rozwiązania są tekstami. Zatem na tym etapie, uwzględniając poprzednio omówione instrukcje, nasz blok sterowania powinien wyglądać następująco:

```
control

   char Ocena;

   createAppWindow;
   setAppWinTitle(”Ryzyko choroby wieńcowej”);
   vignette(”Ryzyko choroby wieńcowej”,
         ”Przykładowa aplikacja do oceny ryzyka wystąpienia choroby
          wieńcowej”, ”© 2003 AITECH”);
   goal( ”ryzyko_rozwoju_choroby_wieńcowej = Ocena” );
end;
```

Teraz dla rozwiązania problemu wystarczy wybrać opcję (na poziomie menu systemu PC-Shell): `Wnioskowanie | Wykonanie programu.`

Uruchomienie wnioskowania i podanie podczas konsultacji danych, których wartości są takie, że uniemożliwiają potwierdzenie hipotezy (celu, problemu), tym samym znalezienie jakichkolwiek rozwiązań spowoduje wygenerowanie w oknie rozwiązań systemowego komunikatu „Hipoteza niepotwierdzona”. Zależnie od merytorycznej konstrukcji bazy wiedzy (głównie reguł), trochę na zasadzie domkniętego świata (ang. Closed-world assumption), możemy w pewnych sytuacjach uznać, że jeśli dana hipoteza nie jest potwierdzona to prawdziwa jest teza jej przeciwstawna. Tak jest w naszym przykładzie. I w tej sytuacji, z punktu widzenia użytkownika aplikacji, byłoby korzystniej, gdyby zamiast systemowego komunikatu pojawiałby się inny tekst np. „Brak podstaw do uznania, że występuje ryzyko choroby wieńcowej” lub „ryzyko_rozwoju_choroby_wieńcowej = ”nie występuje”. Do tego celu służy instrukcja ***setSysText***, która w tym przypadku może przyjąć następującą postać:

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

Tę samą instrukcję, lecz z parametrem *problem* można użyć do zmiany tekstu pojawiającego się automatycznie w polu *Problem* okien konsultacji i rozwiązania. Domyślnie jest to najczęściej dwójka <A, W> o treści zgodnej z tą użytą jako argument instrukcji ***goal***. W naszym przykładzie możemy zamienić ustawienia domyślne w następujący sposób:

```
setSysText( problem, ”Ocena ryzyka wystąpienia miażdżycy” );
```

8. Automatyczne uruchomienie aplikacji – instrukcja *run*

Jak Państwo z pewnością zauważyli, nadal pozostaje pewna niedogodność w postaci konieczności wybierania opcji `Wnioskowanie | Wykonanie programu` za każdym razem, gdy chcemy ponownie uruchomić proces wnioskowania. Ten problem rozwiązuje instrukcja ***run***, która może być zapisana w dowolnym miejscu programu w bloku sterowania, w szczególności na jego początku. Ta instrukcja (można ją również traktować deklaratywnie) nie ma argumentów (zob. dokumentacja). Teraz już nasz kod wygląda następująco:

```
control

   char Ocena;

   run;
   createAppWindow;
   setAppWinTitle(”Ryzyko choroby wieńcowej”);
   vignette(”Ryzyko choroby wieńcowej”,
         ”Przykładowa aplikacja do oceny ryzyka wystąpienia choroby
          wieńcowej”, ”© 2003 AITECH”);
   goal( ”ryzyko_rozwoju_choroby_wieńcowej = Ocena” );

end;
```

Dzięki temu dla uruchomienia naszej bazy wiedzy i rozwiązania problemu wystarczy jedynie ją załadować (czyli otworzyć odpowiedni plik) wybierając opcję `Plik | Otwarcie`, co było nieodzowne również w poprzedniej sytuacji.

9. Dodanie prostego menu – instrukcja *menu*

Obecna postać naszej aplikacji ma dość istotną wadę, a mianowicie nie można powtarzać konsultacji wielokrotnie, bez ponownego uruchamiania całej aplikacji. Problem ten można rozwiązać na wiele sposobów, np. przez wprowadzenie pętli programowej, testującej określony warunek końca pracy z systemem. Innym rozwiązaniem może być użycie instrukcji ***menu*** (szczegóły składniowe zob. Dokumentacja), która ponadto umożliwia dodanie innych opcji (wariantów) pracy aplikacji, np. rozwiązywania innych, wybranych problemów.

Po użyciu instrukcji menu aplikację można zmodyfikować w następujący sposób:

```
control

   char Ocena;

   run;
   createAppWindow;
   setAppWinTitle(”Ryzyko choroby wieńcowej”);
   vignette(”Ryzyko choroby wieńcowej”,
         ”Przykładowa aplikacja do oceny ryzyka wystąpienia choroby
          wieńcowej”, ”© 2003 AITECH”);
   menu ”Diagnoza”
      1. ”Konsultacja”
      2. ”Wyjście”
      case 1:
         goal( ”ryzyko_rozwoju_choroby_wieńcowej = Ocena” );
         delNewFacts;
      case 2:
         exit;
   end;
end;
```

W podanym przykładzie pojawiły się dwie inne instrukcje: *delNewFacts oraz exit.* Pierwsza z nich usuwa nowe fakty, które zostają automatycznie wprowadzane do bazy wiedzy podczas odpowiedzi na pytania generowane przez system podczas konsultacji. Bez jej zastosowania system zadawałby pytania jedynie podczas pierwszej konsultacji, następne wykorzystywałyby wcześniejsze odpowiedzi. Instrukcja *exit* powoduje bezwarunkowe opuszczenie menu i rozpoczęcie wykonywania dalszych instrukcji o ile byłyby takie.

10. Dodanie uproszczonego arkusza do 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 na etapie budowy prototypu i wstępnego testowania, jednakże ten sposób wprowadzania danych nie jest dobrym rozwiązaniem z punktu widzenia ergonomii interfejsu. Dotyczy to m.in. przypadku konieczności dostarczenia większej liczby danych, zwłaszcza wtedy, gdy określony podzbiór tych danych będzie stały, tzn. niezbędny do przeprowadzenia każdej konsultacji i możliwe jest przewidzenie z góry tego podzbioru. W takim przypadku, dużo wygodniejszym i bardziej przejrzystym z punktu widzenia interfejsu rozwiązaniem jest zastosowanie prostej tabeli (arkusza) do wprowadzania danych. Do tego celu można wykorzystać instrukcję ***nsheetBox***,

która służy do utworzenia prostego arkusza kalkulacyjnego w formie tabeli o jednej kolumnie opisowej i jednej do wprowadzania danych liczbowych.

Napisanie odpowiedniego kodu programu należy poprzedzić określeniem danych, które będą wprowadzane za pomocą tego arkusza. W naszym przykładzie będą to następujące informacje:

- wysokość ciśnienia skurczowego krwi,
- wysokość ciśnienia rozkurczowego krwi,
- ogólny poziom cholesterolu w osoczu.

Instrukcja ***nsheetBox*** wymaga dodatkowo zadeklarowania trzech tablic będących jej argumentami. Pierwsza, o nazwie *Dane* jest typu ***float*** i przechowuje wprowadzane wartości (2. kolumna arkusza), druga o nazwie *NazDan* określa nazwę danych (1. kolumna arkusza), trzecia zaś ustala napisy, które pojawią się w oknie z arkuszem, będące kolejno tytułem okna oraz nazwami kolumn.

Zatem instrukcja ta wraz z koniecznymi deklaracjami przyjmie następującą postać:

```
float Dane[3];
char Tytuł[3], NazDan[3];

Tytuł[0] := ”DANE PACJENTA”;
Tytuł[1] := ”Rodzaj pomiaru”;
Tytuł[2] := ”Wartość pomiaru”;

NazDan[0] := ”ciśnienie_skurczowe_krwi”;
NazDan[1] := ”ciśnienie_rozkurczowe_krwi”;
NazDan[2] := ” poziom_cholesterolu_w_osoczu”;

nsheetBox( 0, 0, 3, 0, Tytuł, NazDan, Dane );
```

Znaczenie pozostałych argumentów zostało omówione w dokumentacji.

Wykonanie podanego fragmentu kodu spowoduje pojawienie się okna z arkuszem, gdzie w kolumnie o nazwie „Wartość pomiaru” można wprowadzać odpowiednie dane. Będą one zapamiętane w tablicy *Dane*. Powstaje jednak problem jak przekazać te dane do bazy wiedzy tak by mogły być wykorzystane w procesie wnioskowania jako fakty? Do tego celu należy wykorzystać instrukcję *addFact* (szczegółowe omówienie w dokumentacji). W dużym skrócie, trzy argumenty tej instrukcji stanowią odpowiednio: identyfikator obiektu (o ile występuje), identyfikator atrybutu oraz wartość. Wykonanie tej instrukcji powoduje utworzenie faktu (w postaci trójki OAW) i wprowadzenie go do bazy wiedzy, dzięki czemu staje się on „widoczny” dla modułu wnioskującego SE.

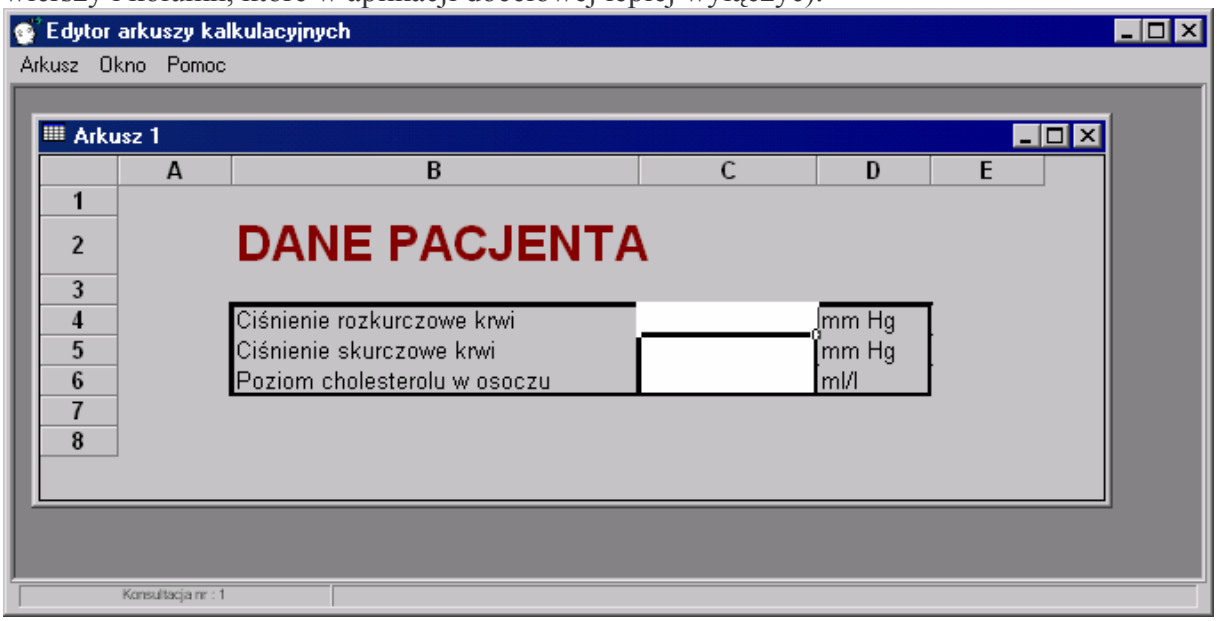
Kod tej procedury może wyglądać następująco:

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

10. Dodanie uproszczonego arkusza do wprowadzania danych

Kolejnym, bardziej elastycznym rozwiązaniem jest wykorzystanie arkuszy kalkulacyjnych zgodnych z arkuszami kalkulacyjnymi systemu Microsoft Excel. Język Sphinx został wyposażony w szereg instrukcji umożliwiających tworzenie, zarządzanie i obsługę arkuszy, dokładny opis znajduje się w *Podręczniku inżyniera wiedzy* Rozdział 9. Poniżej przedstawione są jedynie kroki służące realizacji postawionego zadania – zapewnienia wprowadzania danych do bazy wiedzy systemu ekspertowego.

Chcąc wykorzystać arkusz kalkulacyjny w aplikacji musimy na początek stworzyć jego stronę wizualną. W tym celu możemy wykorzystać aplikację ***arkusze.bw***, która umożliwia tworzenie i edycję arkuszy. Po jej uruchomieniu i utworzeniu nowego arkusza pod prawym przyciskiem myszki dostępne są opcje służące do formatowania zawartości arkusza. Przykładowy arkusz może wyglądać jak na rysunku poniżej (posiada on włączone nagłówki wierszy i kolumn, które w aplikacji docelowej lepiej wyłączyć).



Tak zbudowany arkusz zapisujemy w pliku najlepiej w formacie pierwotnym z rozszerzeniem *vts* lub formacie Excel (*xls*).
Format *xls* (Excel) jest zgodny z wersją Excela 5.0 oraz Excel 95, w przypadku nowszych wersji arkusza mogą wystąpić niezgodności formatów.

Kolejnym etapem jest stworzenie obsługi arkusza w naszej aplikacji. Aby arkusz był stale utworzony w pamięci przed główną pętlą menu dodajemy instrukcję jego otwarcia:

```
openSheet(„Dane wejściowe”, „krew.vts”);
```

gdzie: ”Dane wejściowe” – to nazwa, którą będziemy posługiwać się przy odwoływaniu do arkusza oraz ”krew.vts” to nazwa pliku wzorcowego, gdzie zapisaliśmy wzorzec arkusza – gdy jest podana bez ścieżki dostępu należy umieścić go w tym samym katalogu co baza wiedzy.

Instrukcja ***openSheet*** tworzy arkusz w pamięci lecz nie wizualizuje go. Opcję wizualizacji dodamy w odpowiedniej pozycji menu. **Uwaga** instrukcje arkuszowe wymagają w momencie ich wywołania istnienia głównego okna aplikacji, które nimi zarządza. Dlatego

przed instrukcją ***openSheet*** musi wystąpić instrukcja ***createAppWindow***, która tworzy okno aplikacji, natomiast za instrukcją ***menu*** dodajemy instrukcję ***closeSheet*** zwalniającą arkusz:

```
  createAppWindow;
  openSheet( "Dane wejściowe", "krew.vts"); // otwarcie arkusza
  menu "Menu"
   1. "Pokaż arkusz"
   2. "Konsultacja"
   3. "Wyjście"
  case 1:
   showSheet( "Dane wejściowe", 0 ); // pokazanie na ekranie arkusza do
                                     // wprowadzania
  case 2:
   float Dane[3];
   getSheetValue( "Dane wejściowe","",4,3, Dane[0] );
   getSheetValue( "Dane wejściowe","",5,3, Dane[1] );
   getSheetValue( "Dane wejściowe","",6,3, Dane[2] );
   ...      // uruchomienie wnioskowania
 case 3:
   exit;
 end;
 closeSheet( "Dane wejściowe");    // zamknięcie arkusza
```

Instrukcja, która umożliwia pobranie danych, to instrukcja ***getSheetValue*** (użyta w przykładzie).

Aby zapamiętać i odtworzyć ostatnio wprowadzone dane można wykorzystać instrukcje ***getProfile, writeProfile oraz setSheetValue*** (szerzej o tych instrukcjach – zob. dokumentacja). W takim ujęciu podany fragment kodu należałoby rozszerzyć do podanej niżej postaci:

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

getProfile( "Krew", "CiśnienieRozkurczowe", S, "", "krew.ini");
setSheetValue( "Dane wejściowe","",4,3, S );
getProfile( "Krew", "CiśnienieSkurczowe", S, "", "krew.ini");
setSheetValue( "Dane wejściowe","",5,3, S );
getProfile( "Krew", "PoziomCholesterolu", S, "", "krew.ini");
setSheetValue( "Dane wejściowe","",6,3, S );

menu "Menu"
  1. "Pokaż arkusz"
  2. "Konsultacja"
  3. "Zapamiętanie i wyjście"
case 1:
  showSheet( "Dane wejściowe", 0 );
case 2:
  getSheetValue( "Dane wejściowe","",4,3, Dane[0] );
  getSheetValue( "Dane wejściowe","",5,3, Dane[1] );
  getSheetValue( "Dane wejściowe","",6,3, Dane[2] );
  ...      // instrukcje dotyczące uruchomienia wnioskowania
case 3:
  getSheetValue( "Dane wejściowe","",4,3, S );
  writeProfile( "Krew", "CiśnienieRozkurczowe", S, "krew.ini");
  getSheetValue( "Dane wejściowe","",5,3, S );
  writeProfile( "Krew", "CiśnienieSkurczowe", S, "krew.ini");
```

```
 getSheetValue( "Dane wejściowe","",6,3, S );
 writeProfile( "Krew", "PoziomCholesterolu", S,  "krew.ini");
```

13. Ostateczny kod aplikacji (bazy wiedzy)

```
knowledge base przykład1
 facets
      ryzyko_rozwoju_choroby_wieńcowej:
            query "czy występują skłonności dziedziczne"
            val oneof{"występuje","nie występuje"};

      ryzyko_miażdzycy_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_miażdzycy_tętnic_wieńcowych ="tak",
            skłonności_dziedziczone_do_rozwoju_choroby_wieńcowej="tak";

      ryzyko_miażdzycy_tętnic_wieńcowych ="tak" if
            nadciśnienie_tętnicze="tak",
            podwyższony_poziom_cholesterolu_w_osoczu="tak";

      nadciśnienie_tętnicze="tak" if
            ciśnienie_rozkurczowe_krwi=X,
            X >= 95,
            ciśnienie_skurczowe_krwi=Y,
            Y >= 140;

      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 miażdżycy” );
    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”, ”© 2003 AITECH”);
    openSheet( "Dane wejściowe", "krew.vts");
    getProfile( "Krew", "CiśnienieRozkurczowe", S, "", "krew.ini");
    setSheetValue( "Dane wejściowe","",4,3, S );
    getProfile( "Krew", "CiśnienieSkurczowe", S, "", "krew.ini");
    setSheetValue( "Dane wejściowe","",5,3, S );
    getProfile( "Krew", "PoziomCholesterolu", S, "", "krew.ini");
    setSheetValue( "Dane wejściowe","",6,3, S );

    menu "Menu"
      1. "Pokaż arkusz"
      2. "Konsultacja"
      3. "Zapamiętanie i wyjście"
    case 1:
      showSheet( "Dane wejściowe", 0 );
    case 2:
      getSheetValue( "Dane wejściowe","",4,3, Dane[0] );
      getSheetValue( "Dane wejściowe","",5,3, Dane[1] );
      getSheetValue( "Dane wejściowe","",6,3, 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” );
      delNewFacts;
    case 3:
      getSheetValue( "Dane wejściowe","",4,3, S );
      writeProfile( "Krew", "CiśnienieRozkurczowe", S, "krew.ini");
      getSheetValue( "Dane wejściowe","",5,3, S );
      writeProfile( "Krew", "CiśnienieSkurczowe", S, "krew.ini");
      getSheetValue( "Dane wejściowe","",6,3, S );
      writeProfile( "Krew", "PoziomCholesterolu", S,  "krew.ini");
      exit;
    end;
  end;
end;
```