PRZEWODNIK PO SAIL'u

Budowa kodu

Blok startowy zawsze rozpoczynamy słowem kluczowym starting.

```
starting
  begin
    //kod programu
    begin

    end;
  end;
```

Znaczenie słów kluczowych:

- Starting - słowo rozpoczynające główny blok;
- Begin - blok rozpoczynający główne instrukcje;
- end - kończy instrukcje

Moduły i kompilacja

Do zarządzania modułami przyda nam się menu File, w którym znajdują się przydatne opcje:

- New module - uruchamia nowy moduł
- Rename module - zmienia nazwę modułu
- Save module - zapisuje moduł
- Remove module - usuwa moduł

W celu kompilacji kodu edytor udostępnia nam szereg opcji w menu Commands z tym związanych, m.in:

- Run - kompilacja i uruchomienie projektu
- Compile - po dogłębnym sprawdzeniu błędów w kodzie kompiluje moduły, które zostały zmienione po ostatniej kompilacji

Komentowanie

Komentarze w kodzie możemy wstawiać (i powinniśmy to robić ze względów estetycznych) w dwa sposoby:
Blokowe (za pomocą znaczników {}):

```
{       jakiś sobie tam komentarz
        I jego kolejna linijka      }
```

Oraz liniowe (za pomocą znaczników //):

```
// obejmuje tylko jedną linię komentarzu
```

Operatory

Arytmetyczne:	
Dodawanie	+
Odejmowanie	-
Mnożenie	*
Dzielenie	/
Dzielenie całkowite	Div
Reszta z dzielenia	Mod
Logiczne:	
zaprzeczenie	Not
Łącznie z	And
Łączenie stringów itp.	&
Lub, alternatywa	Or
Alternat. wykluczająca	XOr
Porównywanie:	
Nierówny	<>
Równy	=
Mniejszy	<
Większy	>
Mniejszy lub równy	<=
Większy lub równy	=>

Zmienne

Definiujemy je przed blokiem startowym.

***Zmienne lokalne** (var):*

```
Var twoja_zmienna;
```

***Zmienne globalne** (export):*

```
Export twoja_zmienna;
```

Możemy też zadeklarować kilka zmiennych jednocześnie:

```
Var zmienna1, zmienna2;
//lub
Export zmienna1, zmienna2;
```

Deklarowanie zmiennej:

```
Zmienna := 120;
```

W SAIL'u nie określamy typu danej zmiennej, gdyż typ zostanie nadany automatycznie uniwersalny. Możemy, więc pracować jednocześnie na wielu typach zmiennych.

Typy zmiennych

Typ zmiennej	Opis
String	~2^31 znaków (2 GB)
Integer	-2147483648.. 2147483648
Boolean	TRUE lub FALSE
List	lista
Real	liczba zmiennoprzecinkowa
Time	minut$sekund liczby rzeczywiste

Tablice

Deklarują wiele elementów tego samego typu. Tworzymy je w nawiasach [ ].

```
Array:=[3, 4, 5];
```

Tablice łączymy znacznikiem ^.

```
Array1:=[3,4,5];
Array2:=[3,4,6];
JointArray:= Array1 ^ Array2;
//w ten sposób tablica JointArray będzie zawierać wartości 3,4,5,3,4,6
```

Zwracanie wyników z pominięciem duplikatów tworzymy za pomocą union.
Aby pokazały się tylko pojedyncze wartości używamy diff.
Odwrotnością do diff jest operator isect, zwracający część wspólną obu tablic.

```
JointArray:= [1, 2, 3] union [1, 2, 4, 5];
// zwróci 1,2, 3, 4, 5
JointArray:= [1, 2, 3, 1] union [];
//zwróci 1, 2, 3
JointArray:= [1, 2, 3] diff [1, 2, 4];
//zwróci 3, 4
JointArray:= [1, 2, 3] isect [1, 2, 4];
//zwróci 1, 2
```

Pozycję danej wartości w tablicy określamy jako *numer indexu.*

Instrukcje warunkowe

Instrukcja if:

```
If (warunek) then
    {blok instrukcji}
  else
    {alternatywny blok instrukcji}
```

Przy najprostszych instrukcjach możemy zrezygnować z bloku else, czyli alternatywy dla poprzedniej opcji. Zastosowanie:

```
If zmienna = 1 then
  Begin
    { jakieś instrukcje }
  End else
  Begin
    { inne instrukcje }
  End;
```

Możemy też używać kilku warunków w alternatywie do siebie np.:

```
If x= 1 then
  // blok instrukcji #1
Else if x=2 then
  //blok instrukcji #2
Else
  //blok alternatywny
```

Instrukcja case:

```
Case (wyrażenie) of
  wartość1:{blok instrukcji #1};
  wartość2:{blok instrukcji #2};
end;
```

Jest to instrukcja wyboru i używamy jej, gdy mamy do wyboru kilka warunków. Np.:

```
Case (zmienna) of
  1: {jakiś kod który będzie wykonany gdy 1};
  2: {jakiś kod który będzie wykonany gdy 2};
  3: {jakiś kod który będzie wykonany gdy 3};
End;
```

Zakresy:
Służą do określenia przedziałów liczb.

```
Case (zmienna) of
  1..5: // coś tam
  6..10 // coś tam
  Else {inna instrukcja}
End;
```

PRZEWODNIK PO SAIL'u

Pętle

Pętla for:

```
For {początek} to {koniec} do
  {blok instrukcji};
```

Służy m.in. do zliczania z góry określonych wartości. Przykład:

```
For i := 1 to 10 do
  {blok instrukcji}
```

Gdy chcemy zliczać od mniejszej do większej zamiast to dajemy downto, np.:

```
For i:= 10 downto 1 do {blok instrukcji}
```

Pętle możemy stosować też np. z tablicami.

```
For i:=x in array do {blok instrukcji}
```

Pętla while:

```
While (warunek) do
  begin
    {instrukcje}
  end;
```

Służy np. do zliczania z góry nieznanych wartości. Przykład:

```
Var i := 0;
Begin
  While i < 10 do
    Begin
      // jakaś instrukcja
    end;
end;
```

Pętla repeat:

```
Repeat
  // jakaś instrukcja
Until {warunek zakończenia};
```

Pętla repeat działa odwrotnie do pętli while musi być wykonana przynajmniej jeden raz i będzie powtarzana tak długo, aż warunek nie zostanie spełniony.

```
Var i := 0;
Begin
  i := 0;
  Repeat
    // jakaś instrukcja
  Until i := 5;
End;
```

Instrukcje Break i Continue:
Break służy do zatrzymania i wyjścia z pętli, a Continue do przejścia do kolejnej iteracji i sprawdzenia warunku jej zakończenia.

```
Var i := 0;
Begin
  While i < 10 do
    Begin
      // jakaś instrukcja
      If i = 5 then Break;
    end;
end;

// oraz

Var i := 0;
Begin
  While i < 10 do
    Begin
      If i = 5 then
      Begin
      i:=i+1;
      Continue;
      // jakaś instrukcja
    end;
end;
```

Every i Event

Every:
Działa podobnie do pętli, jednak określa co jaki czas kod ma zostać wykonywany.

```
Every minuty$sekundy do
  {blok instrukcji}
```

Opcja enable włącza powtarzanie, opcja disable wyłącza powtarzanie lub całe every (zależnie od miejsca wpisania). Przykład Every:

```
Every 0$30 do
begin
  // instrukcje do wykonania
  enable;
  // wykonuje wybrane instrukcje co 30
sekund
end;
```

W przypadku opcji enable i disable możemy je również oznaczać , aby je globalnie włączać i wyłączać, czyli MARKED i numer every.

```
Every 0$30 MARKED 01 do
begin
  // instrukcje do wykonania
end;

// i gdzieś w kodzie, gdzie nam jest to
potrzebne w celu wyłączenia wstawiamy:
Disable(numer_every);    // np.:
Disable(01);

// a w celu włączenia:
Enable(numer_every);
```

Every możemy też uwarunkować dodając trigger i warunek do spełnienia. W ten sposób będzie działać, aż nie spełni sie warunek.

```
Every 0$30 trigger zmienna = 1 do
begin
  // instrukcje do wykonania
  enable;
  // wykonuje wybrane instrukcje co 30
sekund
    jednak włączona zostanie dopiero, gdy
    zmienna osiągnie wartość 1 podobnie jak
    przy instrukcji if
end;
```

Event:
To nic innego jak określanie zdarzenia, po którym ma nastąpić jakaś akcja.

```
on Event(wartosc1, wartosc2,..., wartoscN) do
  begin
    // instrukcje
  end;
```

Nie podajemy tutaj wartości prędzej zdefiniowanych zmiennych, ani stałych. Jedynie wartości na użytek danego zdarzenia.

```
on Event(a, b) do
  begin
    // instrukcje do wykonania
  end;
```

Funkcje

Ma za zadanie zwrócenie tylko i wyłącznie jednego wyniku.

```
function NazwaFunkcji;
  {blok deklaracji}
  begin
    {zwracanie rezultatu}
    Result:= 'coś tam';
  end;

// i gdzieś w kodzie
Zmienna := NazwaFunkcji;
```

Przekazywanie parametrów:
Przez wartość:
Tworzy jego kopie lokalną.

```
Function nazwa(parametr1, parametr2);
  {blok instrukcji}
```

Przez referencję:
Nazwę parametru poprzedzamy słowem kluczowym var.

```
Function nazwa(var Parametr);
  Begin
    Parametr:=1+2;
  End;
```