Informatyka 1

Wykład VII

Dokumentacja programu, moduły programowe, typy tablicowe

Robert Muszyński
ZPCiR ICT PWr

Zagadnienia: reguły stylu programowania, komentarze „marginesowe”, „blokowe”, moduły programowe w Pascalu, deklaracja typu tablicowego, zasady posługiwania się tablicami, tablice jako argumenty procedur i funkcji, przeszukiwanie tablic.

– Skład FoilTEX –

Reguły stylu programowania

(a) poprawny podział programu na podprogramy, zgodnie ze strukturą opracowywanego zagadnienia,

Reguły stylu programowania

(a) poprawny podział programu na podprogramy, zgodnie ze strukturą opracowywanego zagadnienia,

(b) umiejętny dobór nazw tak, by wyjaśniały rolę zmiennych, stałych, procedur itp., i sugerowały czego te obiekty tyczą,

Reguły stylu programowania

(a) poprawny podział programu na podprogramy, zgodnie ze strukturą opracowywanego zagadnienia,

(b) umiejętny dobór nazw tak, by wyjaśniały rolę zmiennych, stałych, procedur itp., i sugerowały czego te obiekty tyczą,

(c) postać graficzna programu (oddzielenia pionowe i poziome, wcięcia) tak zaplanowane, aby ukazać jego strukturę i zwiększyć czytelność,

Reguły stylu programowania

(a) poprawny podział programu na podprogramy, zgodnie ze strukturą opracowywanego zagadnienia,

(b) umiejętny dobór nazw tak, by wyjaśniały rolę zmiennych, stałych, procedur itp., i sugerowały czego te obiekty tyczą,

(c) postać graficzna programu (oddzielenia pionowe i poziome, wcięcia) tak zaplanowane, aby ukazać jego strukturę i zwiększyć czytelność,

(d) dokumentacja programowa (komentarze), dostarczająca informacji do dalszej pracy nad programem, wyjaśniająca wykorzystane rozwiązania, itp.,

Reguły stylu programowania

(a) poprawny podział programu na podprogramy, zgodnie ze strukturą opracowywanego zagadnienia,

(b) umiejętny dobór nazw tak, by wyjaśniały rolę zmiennych, stałych, procedur itp., i sugerowały czego te obiekty tyczą,

(c) postać graficzna programu (oddzielenia pionowe i poziome, wcięcia) tak zaplanowane, aby ukazać jego strukturę i zwiększyć czytelność,

(d) dokumentacja programowa (komentarze), dostarczająca informacji do dalszej pracy nad programem, wyjaśniająca wykorzystane rozwiązania, itp.,

(e) asercje, pozwalające zweryfikować poprawność wywołań procedur i funkcji,

Reguły stylu programowania

(a) poprawny podział programu na podprogramy, zgodnie ze strukturą opracowywanego zagadnienia,

(b) umiejętny dobór nazw tak, by wyjaśniały rolę zmiennych, stałych, procedur itp., i sugerowały czego te obiekty tyczą,

(c) postać graficzna programu (oddzielenia pionowe i poziome, wcięcia) tak zaplanowane, aby ukazać jego strukturę i zwiększyć czytelność,

(d) dokumentacja programowa (komentarze), dostarczająca informacji do dalszej pracy nad programem, wyjaśniająca wykorzystane rozwiązania, itp.,

(e) asercje, pozwalające zweryfikować poprawność wywołań procedur i funkcji,

(f) zasada lokalności, która mówi, że elementy powiązane ze sobą powinny wystepować razem, a niezwiązane — oddzielnie.

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;
  KoniecDanych = 99;
  GrupaDanych = 20;
  XZeroMin = 5;
  XZeroMax = 10;
  Sprawdz = TRUE;
VAR
  sygnal: INTEGER;
  bylplus: BOOLEAN;
  nsygn: INTEGER;
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;
  GrupaDanych = 20;
  XZeroMin = 5;
  XZeroMax = 10;
  Sprawdz = TRUE;
VAR
  sygnal: INTEGER;
  bylplus: BOOLEAN;
  nsygn: INTEGER;
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;       { sygnalizuje koniec pracy prog  }
  GrupaDanych = 20;
  XZeroMin = 5;
  XZeroMax = 10;
  Sprawdz = TRUE;
VAR
  sygnal: INTEGER;
  bylplus: BOOLEAN;
  nsygn: INTEGER;
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;       { sygnalizuje koniec pracy prog  }
  GrupaDanych = 20;        { wielk.grupy danych do zliczania}
  XZeroMin = 5;
  XZeroMax = 10;
  Sprawdz = TRUE;
VAR
  sygnal: INTEGER;
  bylplus: BOOLEAN;
  nsygn: INTEGER;
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;       { sygnalizuje koniec pracy prog  }
  GrupaDanych = 20;        { wielk.grupy danych do zliczania}
  XZeroMin = 5;            { minimalna ilosc przeciec zera  }
  XZeroMax = 10;
  Sprawdz = TRUE;
VAR
  sygnal: INTEGER;
  bylplus: BOOLEAN;
  nsygn: INTEGER;
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;       { sygnalizuje koniec pracy prog  }
  GrupaDanych = 20;        { wielk.grupy danych do zliczania}
  XZeroMin = 5;            { minimalna ilosc przeciec zera  }
  XZeroMax = 10;           { maksymalna ilosc przeciec zera }
  Sprawdz = TRUE;

VAR
  sygnal: INTEGER;
  bylplus: BOOLEAN;
  nsygn: INTEGER;
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;       { sygnalizuje koniec pracy prog  }
  GrupaDanych = 20;        { wielk.grupy danych do zliczania}
  XZeroMin = 5;            { minimalna ilosc przeciec zera  }
  XZeroMax = 10;           { maksymalna ilosc przeciec zera }
  Sprawdz = TRUE;          { czy sprawdz.poprawn.wart.sygn. }
VAR
  sygnal: INTEGER;
  bylplus: BOOLEAN;
  nsygn: INTEGER;
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;       { sygnalizuje koniec pracy prog  }
  GrupaDanych = 20;        { wielk.grupy danych do zliczania}
  XZeroMin = 5;            { minimalna ilosc przeciec zera  }
  XZeroMax = 10;           { maksymalna ilosc przeciec zera }
  Sprawdz = TRUE;          { czy sprawdz.poprawn.wart.sygn. }
VAR
  sygnal: INTEGER;         { pobrana wartosc sygnalu }
  bylplus: BOOLEAN;
  nsygn: INTEGER;
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;       { sygnalizuje koniec pracy prog  }
  GrupaDanych = 20;        { wielk.grupy danych do zliczania}
  XZeroMin = 5;            { minimalna ilosc przeciec zera  }
  XZeroMax = 10;           { maksymalna ilosc przeciec zera }
  Sprawdz = TRUE;          { czy sprawdz.poprawn.wart.sygn. }
VAR
  sygnal: INTEGER;         { pobrana wartosc sygnalu }
  bylplus: BOOLEAN;        { czy poprzedni sygnal byl dodat.}
  nsygn: INTEGER;
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;       { sygnalizuje koniec pracy prog  }
  GrupaDanych = 20;        { wielk.grupy danych do zliczania}
  XZeroMin = 5;            { minimalna ilosc przeciec zera  }
  XZeroMax = 10;           { maksymalna ilosc przeciec zera }
  Sprawdz = TRUE;          { czy sprawdz.poprawn.wart.sygn. }
VAR
  sygnal: INTEGER;         { pobrana wartosc sygnalu }
  bylplus: BOOLEAN;        { czy poprzedni sygnal byl dodat.}
  nsygn: INTEGER;          { licznik wczytanych wart. sygn. }
  nxzero: INTEGER;
```

Dokumentacja programu

- komentarze „marginesowe”

```
CONST
  SygnalAmp = 10;          { amplituda sygnalu wejsciowego  }
  KoniecDanych = 99;       { sygnalizuje koniec pracy prog  }
  GrupaDanych = 20;        { wielk.grupy danych do zliczania}
  XZeroMin = 5;            { minimalna ilosc przeciec zera  }
  XZeroMax = 10;           { maksymalna ilosc przeciec zera }
  Sprawdz = TRUE;          { czy sprawdz.poprawn.wart.sygn. }
VAR
  sygnal: INTEGER;         { pobrana wartosc sygnalu }
  bylplus: BOOLEAN;        { czy poprzedni sygnal byl dodat.}
  nsygn: INTEGER;          { licznik wczytanych wart. sygn. }
  nxzero: INTEGER;         { licznik przeciec zera w grupie }
```

```
CONST
  StopnieNaRadiany = 0.017453292;
  SumaKatow = 180.0;
```

```
CONST
  StopnieNaRadiany = 0.017453292;          { dla przeliczen }
  SumaKatow = 180.0;
```

```
CONST
  StopnieNaRadiany = 0.017453292;          { dla przeliczen }
  SumaKatow = 180.0;                       { ...w trojkacie }
```

```
CONST
  StopnieNaRadiany = 0.017453292;          { dla przeliczen }
  SumaKatow = 180.0;                       { ...w trojkacie }
```

```
READ(sygnal);
nsygn := 1;
bylplus := (sygnal = ABS(sygnal));
```

```
CONST
  StopnieNaRadiany = 0.017453292;          { dla przeliczen }
  SumaKatow = 180.0;                       { ...w trojkacie }
```

```
READ(sygnal);              {pierwsza wart. dla porown. w petli}
nsygn := 1;
bylplus := (sygnal = ABS(sygnal));
```

```
CONST
  StopnieNaRadiany = 0.017453292;          { dla przeliczen }
  SumaKatow = 180.0;                       { ...w trojkacie }
```

```
READ(sygnal);              {pierwsza wart. dla porown. w petli}
nsygn := 1;
bylplus := (sygnal = ABS(sygnal));    { tylko wtedy sygn + }
```

```
CONST
  StopnieNaRadiany = 0.017453292;          { dla przeliczen }
  SumaKatow = 180.0;                       { ...w trojkacie }
```

```
READ(sygnal);            {pierwsza wart. dla porown. w petli}
nsygn := 1;
bylplus := (sygnal = ABS(sygnal));    { tylko wtedy sygn + }
```

```
IF ((sygnal > 0) AND (NOT bylplus)) OR
   ((sygnal < 0) AND bylplus) THEN
BEGIN
  nxzero := nxzero + 1;
  bylplus := (sygnal = ABS(sygnal));
END;
```

```
CONST
  StopnieNaRadiany = 0.017453292;          { dla przeliczen }
  SumaKatow = 180.0;                       { ...w trojkacie }
```

```
READ(sygnal);              {pierwsza wart. dla porown. w petli}
nsygn := 1;
bylplus := (sygnal = ABS(sygnal));    { tylko wtedy sygn + }
```

```
IF ((sygnal > 0) AND (NOT bylplus)) OR
   ((sygnal < 0) AND bylplus) THEN
BEGIN                    {wykryte przeciec. zera - zapamietac!}
  nxzero := nxzero + 1;
  bylplus := (sygnal = ABS(sygnal));
END;
```

```
CONST
  StopnieNaRadiany = 0.017453292;          { dla przeliczen }
  SumaKatow = 180.0;                       { ...w trojkacie }
```

```
READ(sygnal);              {pierwsza wart. dla porown. w petli}
nsygn := 1;
bylplus := (sygnal = ABS(sygnal));    { tylko wtedy sygn + }
```

```
IF ((sygnal > 0) AND (NOT bylplus)) OR
   ((sygnal < 0) AND bylplus) THEN
BEGIN                    {wykryte przeciec. zera - zapamietac!}
  nxzero := nxzero + 1;
  bylplus := (sygnal = ABS(sygnal));  { tylko wtedy sygn + }
END;
```

- komentarze „in-line”

```
IF (b*b - 4.0*a*c) >= 0.0 THEN
BEGIN


  sqdelta := SQRT(b*b - 4.0*a*c);
  x1 := (-b - sqdelta) / (2.0 * a);
  x2 := (-b + sqdelta) / (2.0 * a);
  {...}
```

- komentarze „in-line”

```
IF (b*b - 4.0*a*c) >= 0.0 THEN
BEGIN
  (* wiemy ze istnieja dwa pierwiastki rzeczywiste,  *)
  (* pozostaje je wyliczyc i wyswietlic na terminalu *)
  sqdelta := SQRT(b*b - 4.0*a*c);
  x1 := (-b - sqdelta) / (2.0 * a);
  x2 := (-b + sqdelta) / (2.0 * a);
  {...}
```

- komentarze „in-line”

```
IF (b*b - 4.0*a*c) >= 0.0 THEN
BEGIN
  (* wiemy ze istnieja dwa pierwiastki rzeczywiste,  *)
  (* pozostaje je wyliczyc i wyswietlic na terminalu *)
  sqdelta := SQRT(b*b - 4.0*a*c);
  x1 := (-b - sqdelta) / (2.0 * a);
  x2 := (-b + sqdelta) / (2.0 * a);
  {...}
```

- komentarze „blokowe”

```
{   +------+------+------+------+------+------+   }
{   |   UWAGA: ponizszy fragment programu zostal   |   }
{   |   przeniesiony spod Turbo Pascala i nie ma   |   }
{   |   pewnosci co do jego poprawnego dzialania.  |   }
{   |   Uzywaj tylko na wlasna odpowiedzialnosc!   |   }
{   +------+------+------+------+------+------+   }
```

- komentarze opisujące działanie procedur i funkcji

```
PROCEDURE DodajDni( Dzien1,Miesiac1,Rok1,NDni: INTEGER;
                    VAR Dzien2,Miesiac2,Rok2: INTEGER);
```

- komentarze opisujące działanie procedur i funkcji

```
PROCEDURE DodajDni( Dzien1,Miesiac1,Rok1,NDni: INTEGER;
                    VAR Dzien2,Miesiac2,Rok2: INTEGER);
(*  Procedura dodaje okreslona ilosc dni do podanej     *
 *  daty i oblicza nowa date                            *
```

- komentarze opisujące działanie procedur i funkcji

```
PROCEDURE DodajDni( Dzien1,Miesiac1,Rok1,NDni: INTEGER;
                VAR Dzien2,Miesiac2,Rok2: INTEGER);
(*  Procedura dodaje okreslona ilosc dni do podanej     *
 *  daty i oblicza nowa date                            *
 *  Parametry:                                          *
 *       Dzien1   - dzien daty wyjsciowej (1-31)        *
 *       Miesiac1 - miesiac daty wyjsciowej (1-12)      *
 *       Rok1     - rok daty wyjsciowej (pelny,np.1996) *
 *       NDni     - liczba dni do dodania (+ lub -)     *
 *       Dzien2, Miesiac2, Rok2 - nowo wyliczona data   *
```

- komentarze opisujące działanie procedur i funkcji

```
PROCEDURE DodajDni( Dzien1,Miesiac1,Rok1,NDni: INTEGER;
                VAR Dzien2,Miesiac2,Rok2: INTEGER);
(*  Procedura dodaje okreslona ilosc dni do podanej     *
 *  daty i oblicza nowa date                            *
 *  Parametry:                                          *
 *       Dzien1   - dzien daty wyjsciowej (1-31)        *
 *       Miesiac1 - miesiac daty wyjsciowej (1-12)      *
 *       Rok1     - rok daty wyjsciowej (pelny,np.1996) *
 *       NDni     - liczba dni do dodania (+ lub -)     *
 *       Dzien2, Miesiac2, Rok2 - nowo wyliczona data   *
 *  PRE: Dzien1,Miesiac1,Rok1 musza okreslac poprawna   *
 *       date nie wczesniejsza niz 1 stycznia 1970      *
```

- komentarze opisujące działanie procedur i funkcji

```
PROCEDURE DodajDni( Dzien1,Miesiac1,Rok1,NDni: INTEGER;
                VAR Dzien2,Miesiac2,Rok2: INTEGER);
(*  Procedura dodaje okreslona ilosc dni do podanej     *
 *  daty i oblicza nowa date                            *
 *  Parametry:                                          *
 *       Dzien1   - dzien daty wyjsciowej (1-31)        *
 *       Miesiac1 - miesiac daty wyjsciowej (1-12)      *
 *       Rok1     - rok daty wyjsciowej (pelny,np.1996) *
 *       NDni     - liczba dni do dodania (+ lub -)     *
 *       Dzien2, Miesiac2, Rok2 - nowo wyliczona data   *
 *  PRE: Dzien1,Miesiac1,Rok1 musza okreslac poprawna   *
 *       date nie wczesniejsza niz 1 stycznia 1970      *
 * POST: Dzien2,Miesiac2,Rok2 otrzymuja date            *
 *       pozniejsza (lub wczesniejsza gdy NDni < 0)     *
 *       o NDni od Dzien1,Miesiac1,Rok1                 *)
```

Moduły programowe — Sun Pascal

- dołączanie plików poleceniem „include”

```
PROGRAM arytmetyka(INPUT,OUTPUT);

FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    WRITELN('Funkcja: Silnia, blad:
                          ujemny argument: ',N)
  ELSE IF N=0 THEN
    Silnia := 1
  ELSE Silnia := N * Silnia(N-1)
END; {Silnia}

VAR X: INTEGER;
BEGIN
  WRITE('Podaj argument dla funkcji Silnia: ');
  READLN(X);
  WRITELN('Silnia(', X:1, ') = ', Silnia(X):1);
END.
```

Moduły programowe — Sun Pascal

- dołączanie plików poleceniem „include”

```
PROGRAM arytmetyka(INPUT,OUTPUT);

FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    WRITELN('Funkcja: Silnia, blad:
                          ujemny argument: ',N)
  ELSE IF N=0 THEN
    Silnia := 1
  ELSE Silnia := N * Silnia(N-1)
END; {Silnia}

VAR X: INTEGER;
BEGIN
  WRITE('Podaj argument dla funkcji Silnia: ');
  READLN(X);
  WRITELN('Silnia(', X:1, ') = ', Silnia(X):1);
END.
```

⟹ **plik: silnia.inc**

Moduły programowe — Sun Pascal

- dołączanie plików poleceniem „include”

```
PROGRAM arytmetyka(INPUT,OUTPUT);

FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    WRITELN('Funkcja: Silnia, blad:
                          ujemny argument: ',N)
  ELSE IF N=0 THEN
    Silnia := 1
  ELSE Silnia := N * Silnia(N-1)
END; {Silnia}

VAR X: INTEGER;
BEGIN
  WRITE('Podaj argument dla funkcji Silnia: ');
  READLN(X);
  WRITELN('Silnia(', X:1, ') = ', Silnia(X):1);
END.
```

⟹ **plik: silnia.inc**

⟹ **plik: main.p**

plik: silnia.inc

```
FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    WRITELN('Funkcja: Silnia, blad:
                          ujemny argument: ',N)
  ELSE IF N=0 THEN
    Silnia := 1
  ELSE Silnia := N * Silnia(N-1)
END; {Silnia}
```

plik: silnia.inc

```
FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    WRITELN('Funkcja: Silnia, blad:
                          ujemny argument: ',N)
  ELSE IF N=0 THEN
    Silnia := 1
  ELSE Silnia := N * Silnia(N-1)
END; {Silnia}
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);


VAR X: INTEGER;
BEGIN
  WRITE('Podaj argument dla funkcji Silnia: ');
  READLN(X);
  WRITELN('Silnia(', X:1, ') = ', Silnia(X):1);
END.
```

plik: silnia.inc

```
FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    WRITELN('Funkcja: Silnia, blad:
                          ujemny argument: ',N)
  ELSE IF N=0 THEN
    Silnia := 1
  ELSE Silnia := N * Silnia(N-1)
END; {Silnia}
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);

#include "silnia.inc"

VAR X: INTEGER;
BEGIN
  WRITE('Podaj argument dla funkcji Silnia: ');
  READLN(X);
  WRITELN('Silnia(', X:1, ') = ', Silnia(X):1);
END.
```

plik: silnia.inc

```
FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    WRITELN('Funkcja: Silnia, blad:
                          ujemny argument: ',N)
  ELSE IF N=0 THEN
    Silnia := 1
  ELSE Silnia := N * Silnia(N-1)
END; {Silnia}
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);

#include "silnia.inc"

VAR X: INTEGER;
BEGIN
  WRITE('Podaj argument dla funkcji Silnia: ');
  READLN(X);
  WRITELN('Silnia(', X:1, ') = ', Silnia(X):1);
END.
```

Kompilacja

`diablo 21:pc -L main.p` ⟸ tworzy `a.out`

- tworzenie modułów

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    ...                        { jak poprzednio }
END; {Silnia}
```

- tworzenie modułów

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    ...                          { jak poprzednio }
END; {Silnia}
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);


VAR X: INTEGER;
BEGIN
  ...                            { jak poprzednio }
END.
```

- tworzenie modułów

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    ...                          { jak poprzednio }
END; {Silnia}
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);

FUNCTION Silnia(N: INTEGER): INTEGER; EXTERN;

VAR X: INTEGER;
BEGIN
  ...                            { jak poprzednio }
END.
```

- tworzenie modułów

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    ...                          { jak poprzednio }
END; {Silnia}
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);

FUNCTION Silnia(N: INTEGER): INTEGER; EXTERN;

VAR X: INTEGER;
BEGIN
  ...                            { jak poprzednio }
END.
```

Kompilacja (lub jak na stronie następnej)

`diablo 21:pc -L -c modul.p` ⟸ tworzy `modul.o`

- tworzenie modułów

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
BEGIN
  IF N<0 THEN
    ...                          { jak poprzednio }
END; {Silnia}
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);

FUNCTION Silnia(N: INTEGER): INTEGER; EXTERN;

VAR X: INTEGER;
BEGIN
  ...                            { jak poprzednio }
END.
```

Kompilacja (lub jak na stronie następnej)

```
diablo 21:pc -L -c modul.p        ⟸ tworzy modul.o
diablo 22:pc -L main.p modul.o    ⟸ tworzy a.out
```

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
    ...                          { jak poprzednio }
```

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
    ...                          { jak poprzednio }
```

plik: modul.h

```
FUNCTION Silnia( N: INTEGER ): INTEGER; EXTERN;
```

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
    ...                          { jak poprzednio }
```

plik: modul.h

```
FUNCTION Silnia( N: INTEGER ): INTEGER; EXTERN;
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);


VAR X: INTEGER;
BEGIN
  ...                            { jak poprzednio }
END.
```

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
    ...                          { jak poprzednio }
```

plik: modul.h

```
FUNCTION Silnia( N: INTEGER ): INTEGER; EXTERN;
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);

#include "modul.h"

VAR X: INTEGER;
BEGIN
  ...                            { jak poprzednio }
END.
```

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
    ...                          { jak poprzednio }
```

plik: modul.h

```
FUNCTION Silnia( N: INTEGER ): INTEGER; EXTERN;
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);

#include "modul.h"

VAR X: INTEGER;
BEGIN
  ...                            { jak poprzednio }
END.
```

Kompilacja (lub jak na stronie poprzedniej)

`diablo 21:pc -L -c modul.p` ⟸ tworzy `modul.o`

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
    ...                          { jak poprzednio }
```

plik: modul.h

```
FUNCTION Silnia( N: INTEGER ): INTEGER; EXTERN;
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);

#include "modul.h"

VAR X: INTEGER;
BEGIN
  ...                            { jak poprzednio }
END.
```

Kompilacja (lub jak na stronie poprzedniej)

```
diablo 21:pc -L -c modul.p      ⟸ tworzy modul.o
diablo 22:pc -L -c main.p       ⟸ tworzy main.o
```

plik: modul.p

```
MODULE pomocniczy;

FUNCTION Silnia( N: INTEGER ): INTEGER;
    ...                          { jak poprzednio }
```

plik: modul.h

```
FUNCTION Silnia( N: INTEGER ): INTEGER; EXTERN;
```

plik: main.p

```
PROGRAM arytmetyka(INPUT,OUTPUT);

#include "modul.h"

VAR X: INTEGER;
BEGIN
  ...                            { jak poprzednio }
END.
```

Kompilacja (lub jak na stronie poprzedniej)

```
diablo 21:pc -L -c modul.p        ⟸ tworzy modul.o
diablo 22:pc -L -c main.p         ⟸ tworzy main.o
diablo 23:pc -L main.o modul.o    ⟸ tworzy a.out
```

Moduły programowe — zakres zmiennych, funkcji i procedur

- zmienne publiczne i prywatne

plik: main.p

```
PROGRAM zakresy(OUTPUT);
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj; EXTERN;
BEGIN
   witaj; witaj; witaj;
   WRITELN('A w programie:       publ = ',
                   publ:1,', pryw = ',pryw:1);
END.
```

Moduły programowe — zakres zmiennych, funkcji i procedur

- zmienne publiczne i prywatne

plik: main.p

```
PROGRAM zakresy(OUTPUT);
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj; EXTERN;
BEGIN
   witaj; witaj; witaj;
   WRITELN('A w programie:       publ = ',
                   publ:1,', pryw = ',pryw:1);
END.
```

Moduły programowe — zakres zmiennych, funkcji i procedur

- zmienne publiczne i prywatne

plik: main.p

```
PROGRAM zakresy(OUTPUT);
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj; EXTERN;
BEGIN
   witaj; witaj; witaj;
   WRITELN('A w programie:       publ = ',
                   publ:1,', pryw = ',pryw:1);
END.
```

Moduły programowe — zakres zmiennych, funkcji i procedur

- zmienne publiczne i prywatne

plik: main.p

```
PROGRAM zakresy(OUTPUT);
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj; EXTERN;
BEGIN
   witaj; witaj; witaj;
   WRITELN('A w programie:       publ = ',
                   publ:1,', pryw = ',pryw:1);
END.
```

Moduły programowe — zakres zmiennych, funkcji i procedur

- zmienne publiczne i prywatne

plik: main.p

```
PROGRAM zakresy(OUTPUT);
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj; EXTERN;
BEGIN
   witaj; witaj; witaj;
   WRITELN('A w programie:       publ = ',
                   publ:1,', pryw = ',pryw:1);
END.
```

plik: modul.p

```
MODULE witaj;
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj;
BEGIN
   publ := publ + 1; pryw := pryw + 1;
   WRITELN('Oto jestem w module. publ = ',
                   publ:1,', pryw = ',pryw:1);
END; {witaj}
```

Moduły programowe — zakres zmiennych, funkcji i procedur

- zmienne publiczne i prywatne

plik: main.p

```
PROGRAM zakresy(OUTPUT);
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj; EXTERN;
BEGIN
   witaj; witaj; witaj;
   WRITELN('A w programie:       publ = ',
                   publ:1,', pryw = ',pryw:1);
END.
```

plik: modul.p

```
MODULE witaj;
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj;
BEGIN
   publ := publ + 1; pryw := pryw + 1;
   WRITELN('Oto jestem w module. publ = ',
                   publ:1,', pryw = ',pryw:1);
END; {witaj}
```

Moduły programowe — zakres zmiennych, funkcji i procedur

- zmienne publiczne i prywatne

plik: main.p

```
PROGRAM zakresy(OUTPUT);
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj; EXTERN;
BEGIN
   witaj; witaj; witaj;
   WRITELN('A w programie:       publ = ',
                   publ:1,', pryw = ',pryw:1);
END.
```

plik: modul.p

```
MODULE witaj;
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj;
BEGIN
   publ := publ + 1; pryw := pryw + 1;
   WRITELN('Oto jestem w module. publ = ',
                   publ:1,', pryw = ',pryw:1);
END; {witaj}
```

Moduły programowe — zakres zmiennych, funkcji i procedur

- zmienne publiczne i prywatne

plik: main.p

```
PROGRAM zakresy(OUTPUT);
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj; EXTERN;
BEGIN
   witaj; witaj; witaj;
   WRITELN('A w programie:       publ = ',
                   publ:1,', pryw = ',pryw:1);
END.
```

plik: modul.p

```
MODULE witaj;
PUBLIC  VAR publ: INTEGER;
PRIVATE VAR pryw: INTEGER;
PROCEDURE witaj;
BEGIN
   publ := publ + 1; pryw := pryw + 1;
   WRITELN('Oto jestem w module. publ = ',
                   publ:1,', pryw = ',pryw:1);
END; {witaj}
```

Przykład uruchomienia

```
diablo 21: a.out
Oto jestem w module. publ = 1, pryw = 1
Oto jestem w module. publ = 2, pryw = 2
Oto jestem w module. publ = 3, pryw = 3
A w programie:       publ = 3, pryw = 0
```

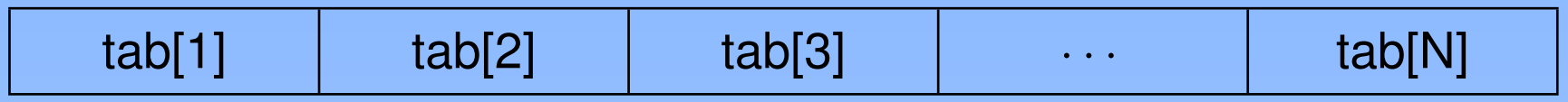
Typy tablicowe

- Tablica jednowymiarowa N elementowa

tab[1]	tab[2]	tab[3]	···	tab[N]

Typy tablicowe

- Tablica jednowymiarowa N elementowa



tab[i] $\Leftrightarrow$ prosta zmienna

Typy tablicowe

- Tablica jednowymiarowa N elementowa

tab[1]	tab[2]	tab[3]	$\cdots$	tab[N]

tab[i] $\Leftrightarrow$ prosta zmienna

- Tablica dwuwymiarowa N$\times$M elementowa

tab[1,1]	tab[1,2]	tab[1,3]	$\cdots$	tab[1,M]
tab[2,1]	tab[2,2]	tab[2,3]	$\cdots$	tab[2,M]
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
tab[N,1]	tab[N,2]	tab[N,3]	$\cdots$	tab[N,M]

Typy tablicowe

- Tablica jednowymiarowa N elementowa

tab[1]	tab[2]	tab[3]	$\cdots$	tab[N]

tab[i] $\Leftrightarrow$ prosta zmienna

- Tablica dwuwymiarowa N$\times$M elementowa

tab[1,1]	tab[1,2]	tab[1,3]	$\cdots$	tab[1,M]
tab[2,1]	tab[2,2]	tab[2,3]	$\cdots$	tab[2,M]
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
tab[N,1]	tab[N,2]	tab[N,3]	$\cdots$	tab[N,M]

tab[i,j] $\Leftrightarrow$ prosta zmienna

Typy tablicowe

- Tablica jednowymiarowa N elementowa

tab[1]	tab[2]	tab[3]	$\cdots$	tab[N]

tab[i] $\Leftrightarrow$ prosta zmienna

- Tablica dwuwymiarowa N$\times$M elementowa

tab[1,1]	tab[1,2]	tab[1,3]	$\cdots$	tab[1,M]
tab[2,1]	tab[2,2]	tab[2,3]	$\cdots$	tab[2,M]
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
tab[N,1]	tab[N,2]	tab[N,3]	$\cdots$	tab[N,M]

tab[i,j] $\Leftrightarrow$ prosta zmienna

- Tablice wielowymiarowe

```
CONST
   NStudentow = 200;      {calkowita liczba studentow}
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF INTEGER;
VAR
   Grupa: Zaliczenia;
   stud, min, max, srednia, suma: INTEGER;
(************************************************)
suma := 0;
min := 10;
max := 0;
FOR stud := 1 TO NStudentow DO
BEGIN
  WRITELN('Prosze podac zaliczenie dla studenta ',stud)
  READLN(Grupa[stud]);
  suma := suma + Grupa[stud];
  IF Grupa[stud] > max THEN max := Grupa[stud];
  IF Grupa[stud] < min THEN min := Grupa[stud];
END;
srednia := suma DIV NStudentow;
```

```
CONST
   NStudentow = 200;      {calkowita liczba studentow}
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF INTEGER;
VAR
   Grupa: Zaliczenia;
   stud, min, max, srednia, suma: INTEGER;
(*************************************************)
suma := 0;
min := 10;
max := 0;
FOR stud := 1 TO NStudentow DO
BEGIN
  WRITELN('Prosze podac zaliczenie dla studenta ',stud)
  READLN(Grupa[stud]);
  suma := suma + Grupa[stud];
  IF Grupa[stud] > max THEN max := Grupa[stud];
  IF Grupa[stud] < min THEN min := Grupa[stud];
END;
srednia := suma DIV NStudentow;
```

```
CONST
   NStudentow = 200;      {calkowita liczba studentow}
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF INTEGER;
VAR
   Grupa: Zaliczenia;
   stud, min, max, srednia, suma: INTEGER;
(************************************************)
suma := 0;
min := 10;
max := 0;
FOR stud := 1 TO NStudentow DO
BEGIN
  WRITELN('Prosze podac zaliczenie dla studenta ',stud)
  READLN(Grupa[stud]);
  suma := suma + Grupa[stud];
  IF Grupa[stud] > max THEN max := Grupa[stud];
  IF Grupa[stud] < min THEN min := Grupa[stud];
END;
srednia := suma DIV NStudentow;
```

```
CONST
   NStudentow = 200;      {calkowita liczba studentow}
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF INTEGER;
VAR
   Grupa: Zaliczenia;
   stud, min, max, srednia, suma: INTEGER;
(*********************************************)
suma := 0;
min := 10;
max := 0;
FOR stud := 1 TO NStudentow DO
BEGIN
  WRITELN('Prosze podac zaliczenie dla studenta ',stud)
  READLN(Grupa[stud]);
  suma := suma + Grupa[stud];
  IF Grupa[stud] > max THEN max := Grupa[stud];
  IF Grupa[stud] < min THEN min := Grupa[stud];
END;
srednia := suma DIV NStudentow;
```

Typy tablicowe — zgodność typów

```
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF INTEGER;
```

Typy tablicowe — zgodność typów

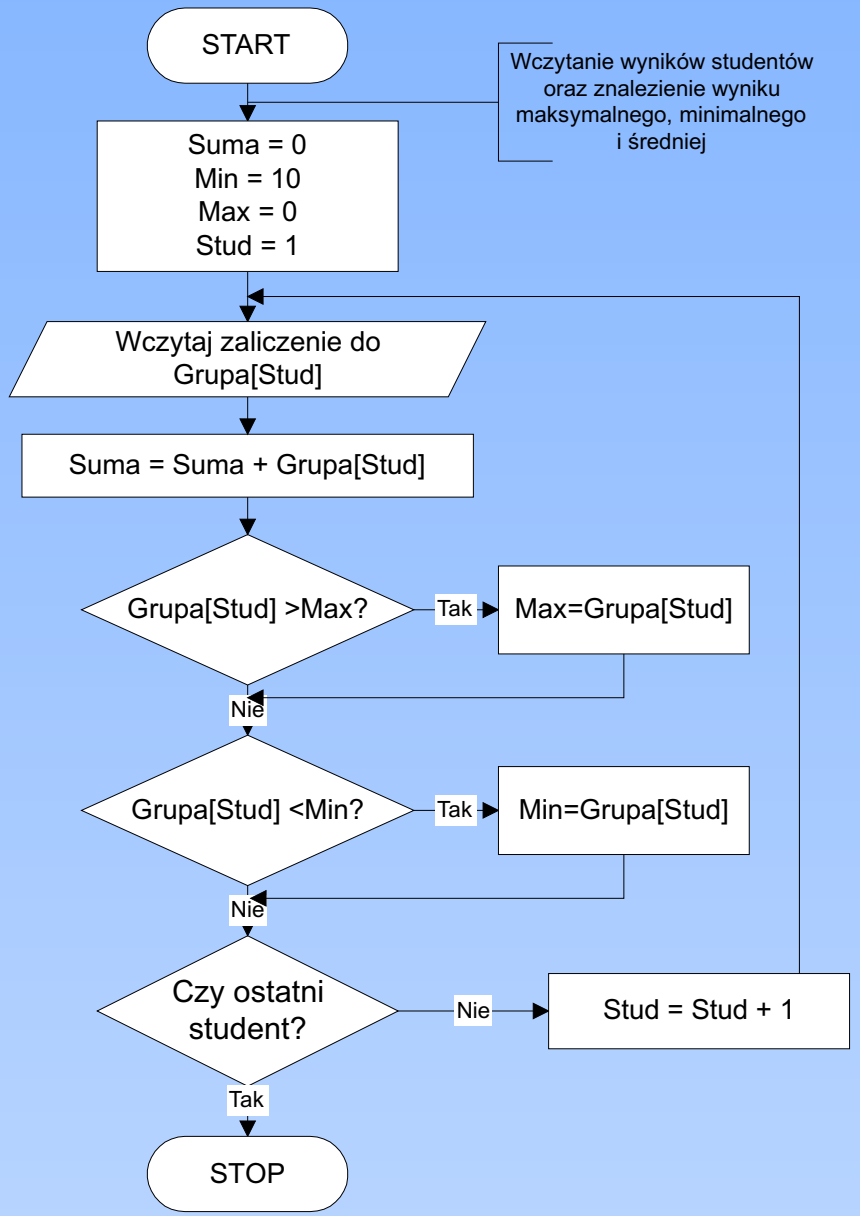
```
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF INTEGER;
VAR
   Grupa1, Grupa2: Zaliczenia;
   Grupa3:         Zaliczenia;
```

Typy tablicowe — zgodność typów

```
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF INTEGER;
VAR
   Grupa1, Grupa2: Zaliczenia;
   Grupa3:         Zaliczenia;
   Grupa4, Grupa5: ARRAY [1..NStudentow] OF INTEGER;
```

Typy tablicowe — zgodność typów

```
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF INTEGER;
VAR
   Grupa1, Grupa2: Zaliczenia;
   Grupa3:         Zaliczenia;
   Grupa4, Grupa5: ARRAY [1..NStudentow] OF INTEGER;
   Grupa6:         ARRAY [1..NStudentow] OF INTEGER;
```


START
Wczytanie wyników studentów oraz znalezienie wyniku maksymalnego, minimalnego i średniej
Suma = 0
Min = 10
Max = 0
Stud = 1
Wczytaj zaliczenie do Grupa[Stud]
Suma = Suma + Grupa[Stud]
Grupa[Stud] >Max?
Tak
Max=Grupa[Stud]
Nie
Grupa[Stud] <Min?
Tak
Min=Grupa[Stud]
Nie
Czy ostatni student?
Nie
Stud = Stud + 1
Tak
STOP

Tablica dwuwymiarowa — przykład

Cwiczenie $\longrightarrow$

NrStudenta $\downarrow$

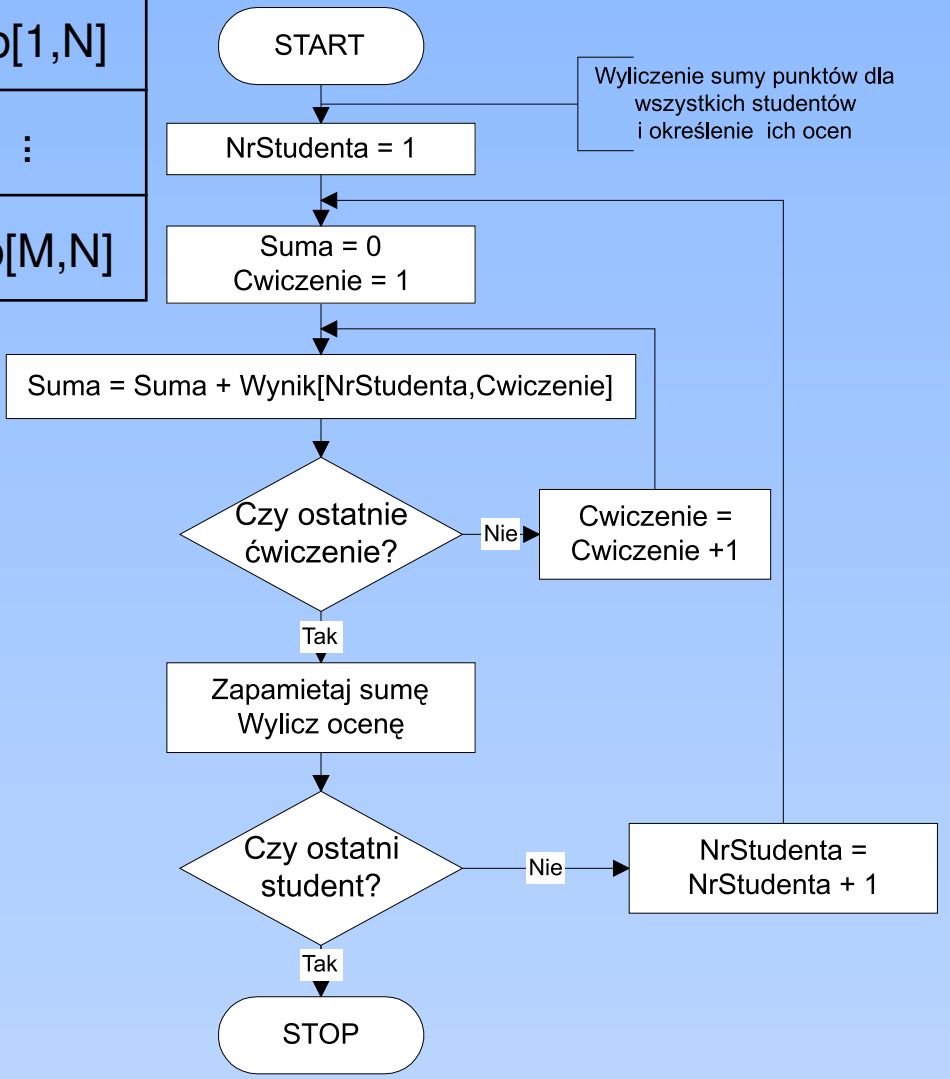
tab[1,1]	$\cdots$	tab[1,N]
$\vdots$	$\ddots$	$\vdots$
tab[M,1]	$\cdots$	tab[M,N]

Tablica dwuwymiarowa — przykład

Cwiczenie →

NrStudenta ↓

tab[1,1]	· · ·	tab[1,N]
⋮	⋱	⋮
tab[M,1]	· · ·	tab[M,N]



```
CONST
   NStudentow = 200;      {calkowita ilosc studentow }
   NCwiczen = 8;          {ilosc cw.do wykon.w semest}
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF 2..5;
   WynikiCwiczen = ARRAY [1..NStudentow,0..NCwiczen]
                         OF INTEGER;
VAR
   Zal1,Zal2,Zal3: Zaliczenia;
   Wyn1,Wyn2,Wyn3: WynikiCwiczen;
(*********************************************)
FOR stud := 1 TO NStudentow DO
BEGIN
  Suma := 0;
  FOR cwicz := 1 TO NCwiczen DO
    Suma := Suma + Wyn1[stud,cwicz];
  Wyn1[stud,0] := Suma;
  Zal1[stud] := 2 + (3*Suma) DIV (NCwiczen*MaxPunkt);
END;
```

```
CONST
   NStudentow = 200;      {calkowita ilosc studentow }
   NCwiczen = 8;          {ilosc cw.do wykon.w semest}
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF 2..5;
   WynikiCwiczen = ARRAY [1..NStudentow,0..NCwiczen]
                         OF INTEGER;
VAR
   Zal1,Zal2,Zal3: Zaliczenia;
   Wyn1,Wyn2,Wyn3: WynikiCwiczen;
(*********************************************)
FOR stud := 1 TO NStudentow DO
BEGIN
  Suma := 0;
  FOR cwicz := 1 TO NCwiczen DO
    Suma := Suma + Wyn1[stud,cwicz];
  Wyn1[stud,0] := Suma;
  Zal1[stud] := 2 + (3*Suma) DIV (NCwiczen*MaxPunkt);
END;
```

```
CONST
   NStudentow = 200;      {calkowita ilosc studentow }
   NCwiczen = 8;          {ilosc cw.do wykon.w semest}
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF 2..5;
   WynikiCwiczen = ARRAY [1..NStudentow,0..NCwiczen]
                         OF INTEGER;
VAR
   Zal1,Zal2,Zal3: Zaliczenia;
   Wyn1,Wyn2,Wyn3: WynikiCwiczen;
(*************************************************)
FOR stud := 1 TO NStudentow DO
BEGIN
  Suma := 0;
  FOR cwicz := 1 TO NCwiczen DO
    Suma := Suma + Wyn1[stud,cwicz];
  Wyn1[stud,0] := Suma;
  Zal1[stud] := 2 + (3*Suma) DIV (NCwiczen*MaxPunkt);
END;
```

```
CONST
   NStudentow = 200;      {calkowita ilosc studentow }
   NCwiczen = 8;          {ilosc cw.do wykon.w semest}
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF 2..5;
   WynikiCwiczen = ARRAY [1..NStudentow,0..NCwiczen]
                         OF INTEGER;
VAR
   Zal1,Zal2,Zal3: Zaliczenia;
   Wyn1,Wyn2,Wyn3: WynikiCwiczen;
(*********************************************)
FOR stud := 1 TO NStudentow DO
BEGIN
  Suma := 0;
  FOR cwicz := 1 TO NCwiczen DO
    Suma := Suma + Wyn1[stud,cwicz];
  Wyn1[stud,0] := Suma;
  Zal1[stud] := 2 + (3*Suma) DIV (NCwiczen*MaxPunkt);
END;
```

```
CONST
   NStudentow = 200;      {calkowita ilosc studentow }
   NCwiczen = 8;          {ilosc cw.do wykon.w semest}
TYPE
   Zaliczenia = ARRAY [1..NStudentow] OF 2..5;
   WynikiCwiczen = ARRAY [1..NStudentow,0..NCwiczen]
                         OF INTEGER;
VAR
   Zal1,Zal2,Zal3: Zaliczenia;
   Wyn1,Wyn2,Wyn3: WynikiCwiczen;
(*************************************************)
FOR stud := 1 TO NStudentow DO
BEGIN
  Suma := 0;
  FOR cwicz := 1 TO NCwiczen DO
    Suma := Suma + Wyn1[stud,cwicz];
  Wyn1[stud,0] := Suma;
  Zal1[stud] := 2 + (3*Suma) DIV (NCwiczen*MaxPunkt);
END;
```

Posługiwanie się tablicami

- wykorzystanie pętli `FOR-TO-DO`

Posługiwanie się tablicami

- wykorzystanie pętli `FOR-TO-DO`
- liczba elementów tablicy z góry ustalona

Posługiwanie się tablicami

- wykorzystanie pętli `FOR-TO-DO`
- liczba elementów tablicy z góry ustalona
- przekroczenie rozmiaru tablicy

Posługiwanie się tablicami

- wykorzystanie pętli `FOR-TO-DO`
- liczba elementów tablicy z góry ustalona
- przekroczenie rozmiaru tablicy
- możliwość podstawiania całych tablic

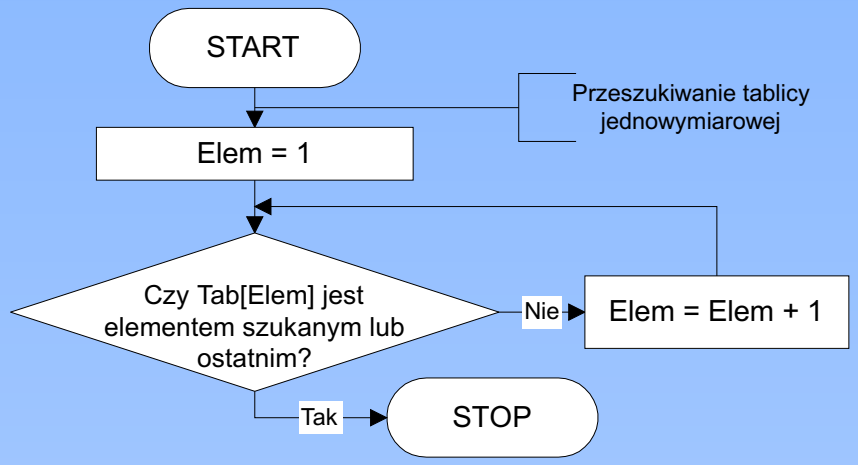
Posługiwanie się tablicami

- wykorzystanie pętli `FOR-TO-DO`
- liczba elementów tablicy z góry ustalona
- przekroczenie rozmiaru tablicy
- możliwość podstawiania całych tablic
- tablice jako argumenty procedur i funkcji (ale nie wartości funkcji!!!)

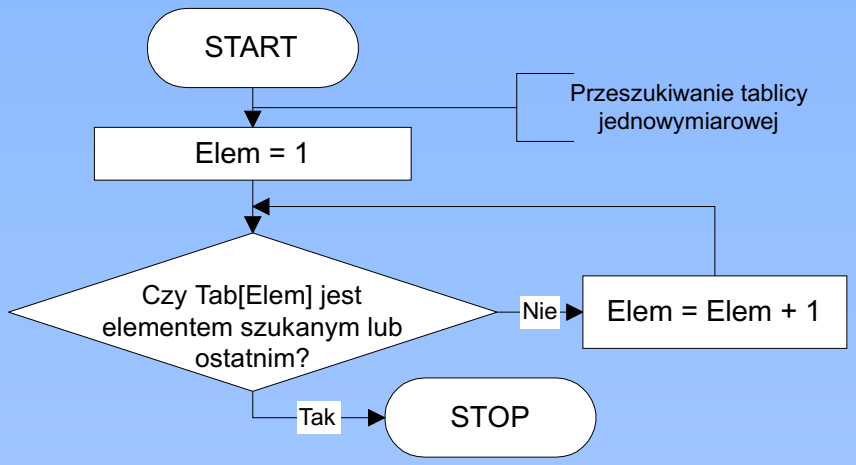
Posługiwanie się tablicami

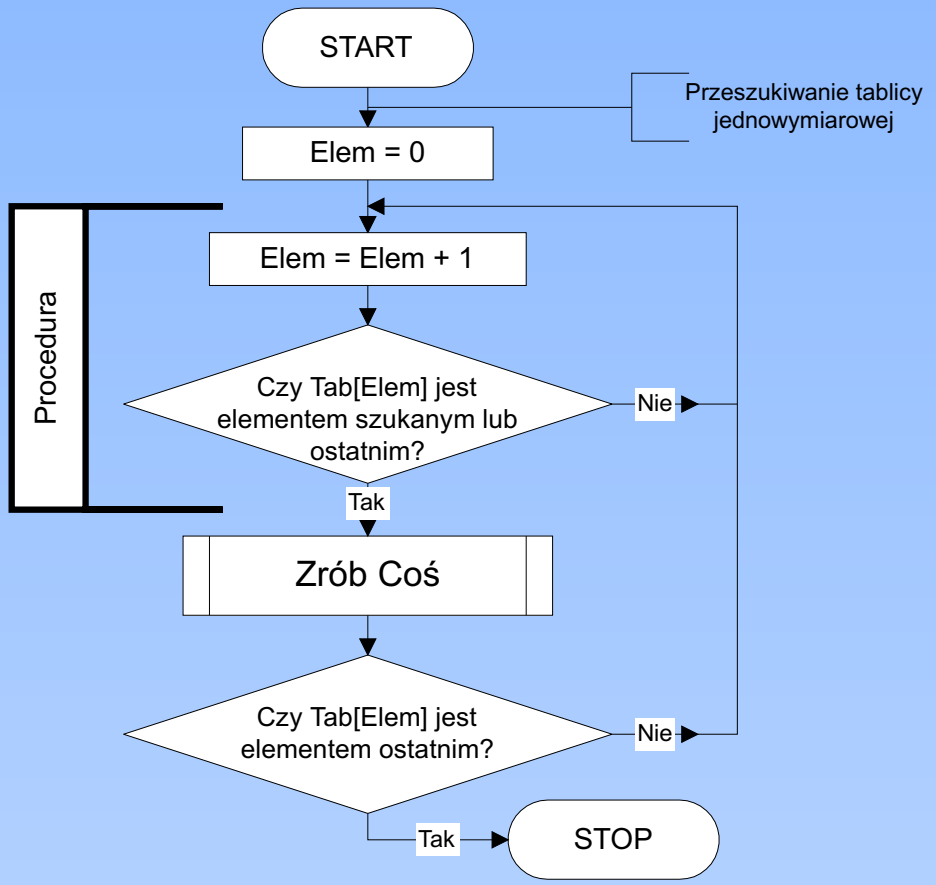
- wykorzystanie pętli `FOR-TO-DO`
- liczba elementów tablicy z góry ustalona
- przekroczenie rozmiaru tablicy
- możliwość podstawiania całych tablic
- tablice jako argumenty procedur i funkcji (ale nie wartości funkcji!!!)
- zgodność typów

Przeszukiwanie tablic



Przeszukiwanie tablic





```
CONST NWartosci = 30;
TYPE Wartosci = ARRAY[1..NWartosci] OF INTEGER;
(*************************************************************)
FUNCTION ZnajdzNastepna(tab: Wartosci; w,i: INTEGER): INTEGER;
(*    Wyszukuje w tablicy podana wartosc w zaczynajac od     *)
(*    pozycji i + 1, o ile jest wewnatrz tablicy. Zwraca     *)
(*    znaleziony indeks lub -1 gdy nie zostal znaleziony.    *)
VAR Znaleziona: BOOLEAN;
BEGIN
  Znaleziona := FALSE; i := i + 1;
  WHILE (i>=1) AND (i<=NWartosci) AND NOT Znaleziona DO
    IF (tab[i] = w)
      THEN Znaleziona := TRUE
      ELSE i := i + 1;
  IF Znaleziona
    THEN ZnajdzNastepna := i
    ELSE ZnajdzNastepna := -1
END; {ZnajdzNastepna}
(*************************************************)
BEGIN
  i := 0; Szukana := ... (* Inicjacja wartosci *)
  REPEAT
    i := ZnajdzNastepna(tablica,Szukana,i);
    IF i<>-1 THEN
      WRITELN('Szukana wartosc znaleziona na pozycji ',i:0);
  UNTIL i=-1;
END.
```

```
CONST NWartosci = 30;
TYPE Wartosci = ARRAY[1..NWartosci] OF INTEGER;
(*************************************************************)
FUNCTION ZnajdzNastepna(tab: Wartosci; w,i: INTEGER): INTEGER;
(*    Wyszukuje w tablicy podana wartosc w zaczynajac od     *)
(*    pozycji i + 1, o ile jest wewnatrz tablicy. Zwraca     *)
(*    znaleziony indeks lub -1 gdy nie zostal znaleziony.    *)
VAR Znaleziona: BOOLEAN;
BEGIN
  Znaleziona := FALSE; i := i + 1;
  WHILE (i>=1) AND (i<=NWartosci) AND NOT Znaleziona DO
    IF (tab[i] = w)
      THEN Znaleziona := TRUE
      ELSE i := i + 1;
  IF Znaleziona
    THEN ZnajdzNastepna := i
    ELSE ZnajdzNastepna := -1
END; {ZnajdzNastepna}
(*************************************************)
BEGIN
  i := 0; Szukana := ... (* Inicjacja wartosci *)
  REPEAT
    i := ZnajdzNastepna(tablica,Szukana,i);
    IF i<>-1 THEN
      WRITELN('Szukana wartosc znaleziona na pozycji ',i:0);
  UNTIL i=-1;
END.
```

```
CONST NWartosci = 30;
TYPE Wartosci = ARRAY[1..NWartosci] OF INTEGER;
(*************************************************************)
FUNCTION ZnajdzNastepna(tab: Wartosci; w,i: INTEGER): INTEGER;
(*    Wyszukuje w tablicy podana wartosc w zaczynajac od     *)
(*    pozycji i + 1, o ile jest wewnatrz tablicy. Zwraca     *)
(*    znaleziony indeks lub -1 gdy nie zostal znaleziony.    *)
VAR Znaleziona: BOOLEAN;
BEGIN
  Znaleziona := FALSE; i := i + 1;
  WHILE (i>=1) AND (i<=NWartosci) AND NOT Znaleziona DO
    IF (tab[i] = w)
      THEN Znaleziona := TRUE
      ELSE i := i + 1;
  IF Znaleziona
    THEN ZnajdzNastepna := i
    ELSE ZnajdzNastepna := -1
END; {ZnajdzNastepna}
(*************************************************)
BEGIN
  i := 0; Szukana := ... (* Inicjacja wartosci *)
  REPEAT
    i := ZnajdzNastepna(tablica,Szukana,i);
    IF i<>-1 THEN
      WRITELN('Szukana wartosc znaleziona na pozycji ',i:0);
  UNTIL i=-1;
END.
```

Wykorzystanie tablic

```
CONST MaxString = 25;
      MaxElement = 100;
TYPE Menu = (Koniec,Dodaj,Wczytaj,Zapisz);
     String = PACKED ARRAY[1..MaxString] OF CHAR;
     TabNazwisk = ARRAY[1..MaxElemenent] OF String;
     TabLiczb = ARRAY[1..MaxElemenent] OF INTEGER;
VAR IloscOsob                          : INTEGER;    {w bazie}
    dane                               : TEXT;
    NazwaPliku                         : String;
    KoniecPracy                        : BOOLEAN;
    WyborMenu                          : Menu;
    ImieiNazwisko, ImieiNazwisko_Ojca,          {skladowe}
    ImieiNazwisko_Matki                : TabNazwisk; {tablice}
    NumerStatystyczny, DzienUrodzenia,              {bazy}
    MiesiacUrodzenia, RokUrodzenia     : TabLiczb;    {danych}
                                                          CDN.
```

Wykorzystanie tablic

```
CONST MaxString = 25;
      MaxElement = 100;
TYPE Menu = (Koniec,Dodaj,Wczytaj,Zapisz);
     String = PACKED ARRAY[1..MaxString] OF CHAR;
     TabNazwisk = ARRAY[1..MaxElemenent] OF String;
     TabLiczb = ARRAY[1..MaxElemenent] OF INTEGER;
VAR IloscOsob                          : INTEGER;    {w bazie}
    dane                               : TEXT;
    NazwaPliku                         : String;
    KoniecPracy                        : BOOLEAN;
    WyborMenu                          : Menu;
    ImieiNazwisko, ImieiNazwisko_Ojca,          {skladowe}
    ImieiNazwisko_Matki                : TabNazwisk; {tablice}
    NumerStatystyczny, DzienUrodzenia,              {bazy}
    MiesiacUrodzenia, RokUrodzenia     : TabLiczb;    {danych}
                                                          CDN.
```

Wykorzystanie tablic

```
CONST MaxString = 25;
      MaxElement = 100;
TYPE Menu = (Koniec,Dodaj,Wczytaj,Zapisz);
     String = PACKED ARRAY[1..MaxString] OF CHAR;
     TabNazwisk = ARRAY[1..MaxElemenent] OF String;
     TabLiczb = ARRAY[1..MaxElemenent] OF INTEGER;
VAR IloscOsob                          : INTEGER;    {w bazie}
    dane                               : TEXT;
    NazwaPliku                         : String;
    KoniecPracy                        : BOOLEAN;
    WyborMenu                          : Menu;
    ImieiNazwisko, ImieiNazwisko_Ojca,          {skladowe}
    ImieiNazwisko_Matki                : TabNazwisk; {tablice}
    NumerStatystyczny, DzienUrodzenia,              {bazy}
    MiesiacUrodzenia, RokUrodzenia     : TabLiczb;    {danych}
                                                          CDN.
```

Wykorzystanie tablic

```
CONST MaxString = 25;
      MaxElement = 100;
TYPE Menu = (Koniec,Dodaj,Wczytaj,Zapisz);
     String = PACKED ARRAY[1..MaxString] OF CHAR;
     TabNazwisk = ARRAY[1..MaxElemenent] OF String;
     TabLiczb = ARRAY[1..MaxElemenent] OF INTEGER;
VAR IloscOsob                          : INTEGER;    {w bazie}
    dane                               : TEXT;
    NazwaPliku                         : String;
    KoniecPracy                        : BOOLEAN;
    WyborMenu                          : Menu;
    ImieiNazwisko, ImieiNazwisko_Ojca,          {skladowe}
    ImieiNazwisko_Matki                : TabNazwisk; {tablice}
    NumerStatystyczny, DzienUrodzenia,             {bazy}
    MiesiacUrodzenia, RokUrodzenia     : TabLiczb;    {danych}

                                                          CDN.
```

```
BEGIN                                                          CD.
  KoniecPracy := FALSE;
  IloscOsob := 0;
  REPEAT
    CASE WyborMenu OF
      Koniec : KoniecPracy := TRUE;
      Dodaj : {...};
      Wczytaj :
      BEGIN
        WRITELN('Podaj nazwe pliku z baza danych:');
        READLN(NazwaPliku)
        RESET(dane,NazwaPliku);
        WczytajPlik(dane, {... ,} IloscOsob);
        CLOSE(dane);
      END;
      Zapisz : {...};
    END; {CASE}
  UNTIL KoniecPracy;
END.
```

```
BEGIN                                                          CD.
  KoniecPracy := FALSE;
  IloscOsob := 0;
  REPEAT
    CASE WyborMenu OF
      Koniec : KoniecPracy := TRUE;
      Dodaj : {...};
      Wczytaj :
      BEGIN
        WRITELN('Podaj nazwe pliku z baza danych:');
        READLN(NazwaPliku)
        RESET(dane,NazwaPliku);
        WczytajPlik(dane, {... ,} IloscOsob);
        CLOSE(dane);
      END;
      Zapisz : {...};
    END; {CASE}
  UNTIL KoniecPracy;
END.
```

```
PROCEDURE WczytajPlik(dane: TEXT; { VAR ...: (tablice); }
                      VAR IloscOsob: INTEGER);
{ Wczytuje zawartosc pliku do struktur w pamieci          }
{ Parametry: (pominiety opis parametrow)                  }
{ PRE: plik dane musi byc otwarty do czytania (RESET)     }
{      zm. IloscOsob okresla dotychczasowa ilosc elemen.  }
{ POST: czyta plik dane do wystapienia NumerOsoby<0       }
{       umieszcza wczytane elementy w tablicach           }
{       umieszcza nowa ilosc elementow w zmien.IloscOsob  }
{       wyswietla komunikat na ekranie                    }

VAR Poprzednio : INTEGER;

                                                     CDN.
```

```
BEGIN                                                              CD.
  Poprzednio := IloscOsob;
  READLN(dane,NumerOsoby);  {pierwszy numer osoby}

  WHILE (NumerOsoby > 0) AND (IloscOsob < MaxElement) DO
    BEGIN
      IloscOsob := IloscOsob + 1;
      NumerStatystyczny[IloscOsob] := NumerOsoby;
      READLN(dane,ImieiNazwisko[IloscOsob]);
      READLN(dane,ImieiNazwisko_Ojca[IloscOsob]);
      READLN(dane,ImieiNazwisko_Matki[IloscOsob]);
      READLN(dane,RokUrodzenia[IloscOsob]);
      READLN(dane,MiesiacUrodzenia[IloscOsob]);
      READLN(dane,DzienUrodzenia[IloscOsob]);

      READLN(dane,NumerOsoby);           {nastepny numer}
    END;

  WRITE('Wczytano ',IloscOsob-Poprzednio,' elementow');
  IF EOF(dane)
    THEN WRITELN('.')
    ELSE WRITELN(' --- PRZED KONCEM PLIKU !!',chr(12));
END; (* WczytajPlik *)
```

```
BEGIN                                                          CD.
  Poprzednio := IloscOsob;
  READLN(dane,NumerOsoby);  {pierwszy numer osoby}

  WHILE (NumerOsoby > 0) AND (IloscOsob < MaxElement) DO
    BEGIN
      IloscOsob := IloscOsob + 1;
      NumerStatystyczny[IloscOsob] := NumerOsoby;
      READLN(dane,ImieiNazwisko[IloscOsob]);
      READLN(dane,ImieiNazwisko_Ojca[IloscOsob]);
      READLN(dane,ImieiNazwisko_Matki[IloscOsob]);
      READLN(dane,RokUrodzenia[IloscOsob]);
      READLN(dane,MiesiacUrodzenia[IloscOsob]);
      READLN(dane,DzienUrodzenia[IloscOsob]);

      READLN(dane,NumerOsoby);           {nastepny numer}
    END;

  WRITE('Wczytano ',IloscOsob-Poprzednio,' elementow');
  IF EOF(dane)
    THEN WRITELN('.')
    ELSE WRITELN(' --- PRZED KONCEM PLIKU !!',chr(12));
END; (* WczytajPlik *)
```

```
BEGIN                                                              CD.
  Poprzednio := IloscOsob;
  READLN(dane,NumerOsoby);  {pierwszy numer osoby}

  WHILE (NumerOsoby > 0) AND (IloscOsob < MaxElement) DO
    BEGIN
      IloscOsob := IloscOsob + 1;
      NumerStatystyczny[IloscOsob] := NumerOsoby;
      READLN(dane,ImieiNazwisko[IloscOsob]);
      READLN(dane,ImieiNazwisko_Ojca[IloscOsob]);
      READLN(dane,ImieiNazwisko_Matki[IloscOsob]);
      READLN(dane,RokUrodzenia[IloscOsob]);
      READLN(dane,MiesiacUrodzenia[IloscOsob]);
      READLN(dane,DzienUrodzenia[IloscOsob]);

      READLN(dane,NumerOsoby);           {nastepny numer}
    END;

  WRITE('Wczytano ',IloscOsob-Poprzednio,' elementow');
  IF EOF(dane)
    THEN WRITELN('.')
    ELSE WRITELN(' --- PRZED KONCEM PLIKU !!',chr(12));
END; (* WczytajPlik *)
```

Indeks