FORTRAN 90
wykład 4 & 5

Janusz Andrzejewski

06/12/11

PLAN

Wyrażenia arytmetyczne

Stałe nazwane

Łańcuchy

Wyrażenia logiczne

Priorytet operacji

Funkcje matematyczne

Wyrażenia arytmetyczne (WA)

Wyrażenia arytmetyczne – mogą zawierać arytmetyczne operatory, operandy, oraz nawiasy. Operandami mogą być:

Stałe liczbowe

Stałe nazwane

Zmienne

Tablice

Odwołania do funkcji (UWAGA! W przypadku występowania kilku funkcji, kolejność wywołania funkcji nie jest zdefiniowana !!!)

Inne wyrażenia arytmetyczne ujęte w nawiasy okrągłe

Uwaga:

operand – argument operatora, np. 3+4=7, operatorem jest dodawanie (+), a operandami 3 oraz 4.Ogólniej, operandem jest to zbiór obiektów na którym wykonywane jest pewne działanie.

WA - operatory

W Fortranie many 5 operatorów arytmetycznych które mogą działać na 3 typach danych arytmetycznych(INTEGER, REAL, COMPLEX)

+ dodawanie lub tożsamość

- odejmowanie lub negacja

/ dzielenie

* mnożenie

** potęgowanie

UWAGA! Wszystkie operatory mogą działać na dowolnym typie arytmetycznym

Typ wyniku dla operacji: + - / *

Typ wyniku dla operacji A .op. B, gdzie .op. oznacza jedno z działań: +(dodawanie), -(odejmowanie), /(dzielenie), *(mnożenie)

Typ A	Typ B	Wartość użyta jako A	Wartość użyta jako B	Typ wyniku
Int.	Int.	A	B	Int
Int.	Real	real(A, kind(B))	B	Real
Int.	Compl.	cmplx(A,0,kind(B))	B	Comp.
Real	Int.	A	real(B, kind(A))	Real
Real	Real	A	B	Real
Real	Compl.	cmplx(a,0,kind(b))	B	Compl.
Compl.	Int.	A	cmplx(B,0,kind(A))	Compl.
Compl.	Real	A	cmplx(B,0,kind(A))	Compl.
Comp.	Comp.	A	B	Comp.

Real(X, kind(rodzaj)) – funkcja wewnętrzna zamieniająca liczbę X na liczbę rzeczywistą o rodzaju TYP

cmplx(x, y, kind(TYP)) – funkcja wewnętrzna zwracającą liczbę zespoloną rodzaju TYP o części rzeczywistej x i urojonej y

Typ wyniku dla operacji: ** (potęgowanie)

Typ wyniku dla operacji AB**

Typ A	Typ B	Wartość użyta jako A	Wartość użyta jako B	Typ wyniku
Int.	Int.	A	B	Int
Int.	Real	real(A, kind(B))	B	Real
Int.	Compl.	cmplx(A,0,kind(B))	B	Comp.
Real	Int.	A	B	Real
Real	Real	A	B	Real
Real	Compl.	cmplx(a,0,kind(b))	B	Compl.
Compl.	Int.	A	B	Compl.
Compl.	Real	A	cmplx(B,0,kind(A))	Compl.
Comp.	Comp.	A	B	Comp.

WA – kolejność działań

Kolejność działań (od najwyższego do najniższego priorytetu):

Wyrażenia w nawiasach

Wywołania funkcji

Potęgowanie

Mnożenie i dzielenie

Dodawanie i odejmowanie

W każdej z tych grup, działania są wykonywane od lewej do prawej np: A/B/C jest równoważne (A/B)/C. Dla potęgowania jest wyjątek, działania są wykonywane od prawej do lewej, tzn A**B**C jest równoważne A**(B**C)

Jeśli wartość wyrażenie da się określić wcześniej, nie wszystkie składniki muszą być obliczane np: X*FUNC(G) – jeśli X=0 to wartość funkcji FUNC nie musi zostać obliczona

WA – kolejność działań cd.

Kompilator jest zobowiązany do przestrzegania nawiasów
W ramach przekształceń matematycznych które są sobie równoważne,
Kompilator może zmieniać kolejność obliczania wyrażeń

Wyrażenie	Równoważne wyrażenie	Wyrażenie	Zabronione przekształcenie
X+Y	Y+X	I/2	0.5*I
-X+Y	Y-X	X*I/J	X*(I/J)
X*A/Y	X*(A/Y)	(X+Y)+Z	X+(Y+Z)
A/B/C	A/(B*C)	I/J/A	I/(J*A)
X*Y-X*Z	X*(Y-Z)	(X*Y)-(X*Z)	X*(Y-Z)
A/5.0	0.2*A	X*(Y-Z)	X*Y-X*Z

Oznaczenia:

X, Y, Z - dowolne typy numeryczne takie jak: integer, real czy complex
I, J – dowolny typ całkowity
A, B, C – dowolny typ rzeczywisty albo zespolony

WA – niejawna konwersja

W przypadku, gdy w wyrażeniu arytmetycznym mamy doczynienia ze składnikami o różnych typach, zachodzi następująca niejawna konwersja typów

INTEGER => REAL => COMPLEX

czyli konwersja idzie zawsze w kierunku „większych” typów np.

1.0/2 => 1.0/2.0 => 0.5

1.0D0/2 => 1.0D0/2.0D0 => 0.5D0

Powyższa konwersja nie obowiązuje dla operatora potęgowania. Jeśli typ wykładnika jest INTEGER to konwersja typów nie zachodzi tzn.

2.0**3 => 2.0*2.0*2.0 => 8.0

(-2.0)**3 => (-2.0)*(-2.0)*(-2.0) => -8.0 ALE!!!

2.0**3.0 => EXP(3.0*LOG(2.0)) => 8.0 (w tym przypadku podstawa nie może być mniejsza od zera)

WA – niejawna konwersja

Jeśli wykładnik jest typem INTEGER ale ujemnym to wyniki jest odwrotnością potęgowania przy dodatniej potędze:

2.0**(-3) = > 1.0/2.0**3 => 1.0/8.0 => 0.125

Jeśli podstawa jest ma typ COMPLEX, a wykładnik jest typem INTEGER to

(2.0, 3.0)**2 => (2.0, 3.0)*(2.0, 3.0)

W przypadku gdy wykładnik jest typem rzeczywistym lub zespolonym zachodzi następująca konwersja

A**B => EXP(B*LOG(A)), gdzie funkcja wykładnicza EXP oraz logarytmiczna (logarytm naturalny) są także określone dla typów zespolonych. Dla typów zespolonych wartości funkcji EXP oraz LOG są obliczane w sensie wartości głównej.

WA – niejawna konwersja

Konwersja z typu REAL na DOUBLE PRECISION nie może powodować powstawania dodatkowej informacji czyli:

```
REAL R

DOUBLE PRECISION D

R=1.0/3.0

D=R
```

Może spowodować że zmienna D będzie miała wartość 0.3333334326744 08 zamiast spodziewanej 0.3333333333333D0

WA – a typ INTEGER

Uwaga na typ INTEGER

8/3 => 2 -8/3=> -2 2*(-3)=> 1/(2**3)=>1/8=>0

Dzielenie dwóch liczb całkowitych jest liczbą całkowitą

Wyrażenie (-2)**3.0 jest niepoprawne

Inne niespodziewane efekty:

3/4*5.0 => 0*5.0 => 0.0 ALE !!!

5.0*3/4 => 15.0/4 => 15.0/4.0 => 3.75

Niektóre wyrażenie mogą być niepoprawnie zapisane jak

4/-3.0**-1 - niepoprawnie

4/(-3.0)**(-1) - poprawnie zapisane

WA – końcowe uwagi

Niektóre operacje arytmetyczne są zabronione ponieważ nie są poprawnie zdefiniowane:

Dzielenie przez zero

Podnoszenie ujemnej wartości do rzeczywistej potęgi

Podnoszenie zera do ujemnej potęgi

- standard nie definiuje co się wtedy będzie działo

Inne błędy związane z wyrażeniami arytmetycznymi to

Overflow np. kiedy chcemy dodać dwie bardzo duże liczby

Underflow np. Kiedy chcemy dodać dwie bardzo małe liczby

WSZYSTKIE OPEARNDY W CHWILI OBLICZANIA WYRAŻENIA MUSZĄ MIEĆ ZDEFINIOWANĄ WARTOŚĆ

Stałe nazwane

Stałe nazwane (ang. Named constants) są wygodnym sposobem na zapamiętywanie pewnych szczególnych wartości pod postacią zmiennych. Takie zmienne charakteryzują się tym, że w czasie programu nie można ich zmienić.

```
REAL PI, TWOPI, HALFPI, RTOD
PARAMETER(PI=3.14159265, TWOPI=2*PI)
PARAMETER(HALFPI=PI/2.0, RTOD=180.0/PI)
```

Innym ważnym zastosowaniem instrukcji PARAMETER jest „inteligentne” wykorzystanie przy deklaracji tablic.

Stałe nazwane

W instrukcji PARAMETER wartości stałych nie mogą być określone poprzez zmienne, wywołanie funkcji czy też odwołanie się do elementów tablicy. Stałe nazwane mogą natomiast być określane poprzez wyrażenia w których występują

- Stałe liczbowe (jak 2, 2.0, 2.0D0 itp)
- Stałe nazwane już wcześniej zdefiniowane
- Operatory: dodawania, odejmowania, mnożenia oraz potęgowanie pod warunkiem że wykładnik jest całkowity

Instrukcja PARAMETER może być przed instrukcją IMPLICIT lub po tej instrukcji

Przy obliczaniu wyrażeń arytmetycznych stosują się ogólne reguły języka omówione wcześniej (kolejność działań, konwersja typów).

Stałe nazwane cd.

W przypadku gdy wartość stałej jest określana w wyniku obliczania wartości wyrażenia, najpierw liczona jest wartość owego wyrażenia zgodnie z regułami podanymi powyżej i w precyzji zgodnej z regułami, a następnie wykonywana jest odpowiednia konwersja typów

Ta sama reguła tyczy się też nadawaniu wartości zwykłym zmiennym

```
REAL a
DOUBLE PRECISON b, c
B=1.0/3.0; A=1.0/3.0; C=1.0d0/3.0d0)
Write(*,*)a, b, c
```

Łańcuchy

Typ znakowy, potocznie zwany „łańcuchami” (ang. string) został wprowadzony w celu umożliwienia łatwego przetwarzania tekstów. Wartości typu znakowego reprezentują ciągi (łańcuchy) znaków, których długość jest z góry określona i stała w programie.

```
CHARACTER*5  NAZWA5, HELP*40
```

Stałe znakowe – są to ciągi znaków ograniczone apostrofami np:
'ALA' 'KOT' 'kotek '

Jeśli chcemy w łańcuchu umieścić apostrof, piszemy dwa obok siebie np: 'it''s' - reprezentuje łańcuch składający się z 4 znaków, ma postać it's

Stała nazwaną znakową można zadeklarować

```
CHARATER*12 TEKST

PARAMETER( TEKST='Zwykly tekst')
```

Łańcuchy

Bardziej eleganckim sposobem deklaracji nazwanych stałych znakowych jest

```
CHARACTER *(*) TEKST

PARAMETER(TEKST='A teraz nie muszę liczyc dlugosci :)')
```

Dzięki takiemu zabiegowi, nie trzeba określać długości stałej znakowej – będzie ona wyznaczona automatycznie

Wyrażenia łańcuchowe

Jedynym możliwym wyrażeniem łańcuchowym jest operacja łączenia dwu ciągów znakowym za operatora konkatenacji **//** np:

'XYZ'//'VW' => 'XYZVW'

W przypadku więcej niż jednego operatora konkatenacji, kolejność działań jest od lewej do prawej tzn.

'AB' // 'CD' // 'EF' => ('AB' // 'CD') // 'EF'

Wycinki znakowe

Fortran umożliwia przetwarzanie wycinków łańcuchów tekstowych, przez tworzenie nazw wycinków. Składnia:

nazwa(w1:w2)

nazwa – jest nazwą zmiennej lub elementem tablicy typu CHARACTER

w1 oraz w2 – są wyrażeniami typu całkowitego, których wartość wskazują odpowiednio lewy i prawy koniec wycinka. Wartości w1 oraz w2 muszą spełniać warunek 1<= w1 <=w2<=L, gdzie L jest długością zmiennej lub elementu tablicy. Jeśli w1 nie występuje zakłada się że ma wartość jeden, jeśli w2 nie występuje to zakłada się że ma wartość L, natomiast jeśli oba wyrażenia nie występują to zakłada się wartości odpowiednio 1 i L

NAZWA='KOWALSKI'

NAZWA(2:5) => 'OWAL'

= a łańcuchy

Instrukcja podstawienia dla typu łańcuchowego ma postać

zmienna_CHAR = wyrażenie_lancuchowe

Jest jedno istotne ograniczenie: instrukcja typu

STRING(1:N) = STRING(10:)

jest poprawna, pod warunkiem ze N jest nie większe niż 9.

```
CHARACTER AUTOR*30, SHORT*5, EXPAND*10
AUTHOR='Shakespeare, William'
SHORT=AUTHOR
EXPAND=SHORT
```

- Jeśli wyrażenie jest dłuższe niż zmienna, wtedy znaki od prawej strony są obcinane
- Jeśli wyrażenie jest krótsze niż zmienna, wtedy z prawej strony dodawane są spacje

= a łańcuchy

```
CHARACTER AUTOR*30, SHORT*5, EXPAND*10
AUTHOR='Shakespeare, William'
SHORT=AUTHOR
EXPAND=SHORT
```

Wynikiem będzie :

'Shake' - to wartość zmiennej SHORT

'Shake ' - to wartość zmiennej EXPAND

Uporządkowanie znaków

Standard Fortranu wymaga, by wszystkie jego implementacje spełniały następujące warunki uporządkowania poszczególnych typów znaków:

- 0 < 1 < 2 < … < 9
- A< B < C < … < Z
- 9 < A lub Z <0 tzn. wszystkie cyfry poprzedzają litery lub następują po nich
- Spacja < A lub spacja < 0

Funkcje operujące na typie znakowym

Funkcja i jej argumety	Typ argumentu	Typ wyniku	Opis
ACHAR(ival)	INT	CHAR	Zwraca znak o numerze ival w tablicy ASCII
CHAR(ival)	INT	CHAR	Zwraca znak o numerze ival w implementacji kompilatora
IACHAR(char)	CHAR	INT	Zwraca numer znaku w tablicy ASCII
ICHAR(char)	CHAR	INT	Zwraca numer znaku w implementacji kompilatora
INDEX(str1, str2, back)	CHAR, LOG	INT	(*)
LEN(str1)	CHAR	INT	Długość łańcucha str1 (zadeklarowana)
LEN_TRIM(str1)	CHAR	INT	Długość bez znaków spacji na końcu łańcucha
TRIM(str1)	CHAR	CHAR	Obcina spacje z końca łańcucha

(*) -przeszukuje ciąg znaków str1 w celu znalezienie wzorca str2 (0 = nie znaleziono)
Argument back jest opcjonalny, i jeśli ma wartość .TRUE. wtedy ciąg znaków str1 Przeszukiwany jest od końca zamiast od początku

Funkcje operujące na typie znakowym ciąg dalszy

Funkcja i jej argumety	Typ argumentu	Typ wyniku	Opis
ADJUSTL(CHAR)	CHAR	CHAR	Zwraca łańcuch o takiej samej długości jak argument ale przesuwając spacje z lewej na prawą stronę łańcucha
ADJUSTR(CHAR)	CHAR	CHAR	Zwraca łańcuch o takiej samej długości jak argument ale przesuwa spacje z prawej na lewą stronę łańcucha
SCAN(str, set, back)	CHAR	INT	Skanuje łańcuch str aby znaleźć jakiś znak ze zbioru znaków set i zwraca jego pozucję. Zero jeśli nie ma znaku ze zbioru set w str.
VERIFY(str, set, back)	CHAR	INT	Zwraca zero, jeśli każdy ze znaków str jest w zbiorze set, lub pozycje w str która nie występuje w zbiorze set

Przykłady:

Przykład 1;

```
CHARACTER :: out1, out2
INTEGER  :: nrZnaku = 65
 out1=CHAR(nrZnaku)
 out2=ACHAR(nrZnaku)       ! Odpowiedz – znak 'A'
```

Przykład 2;

```
CHARACTER :: znak='A'
INTEGER :: wyn1, wyn2
wyn1=ICHAR(znak)
  wyn2=IACHAR(znak)     ! Odpowiedz - 65
```

UWAGA:

Używaj zawsze funkcji ACHAR oraz IACHAR gdyż ich wynik nie zależy od implementacji czy też typu komputera

Przykłady 2

```
CHARACTER(LEN=20) :: str1
INTEGER :: wyn1, wyn2
Str1='ABC XYZ'
  wyn1=LEN(str1)    ! wyn1=20
  wyn2=LEN_TRIM(str1)  ! wyn2=7

CHARACTER(LEN=20) :: str1='THIS IS A TEST!'
CHARACTER(LEN=20) :: str2='TEST'
CHARACTER(LEN=20) :: str3='IS'
INTEGER :: wyn1, wyn2, wyn3
wyn1=INDEX(str1, str2)  ! wyn1=11
wyn2=INDEX(str1, str3)   ! wyn2=3
wyn3=INDEX(str1, str3, .TRUE.)  ! wyn3=6
```

Tabela kodów ASCII

0	nul	32		64	@	96	`
1	soh	33	!	65	A	97	a
2	stx	34	"	66	B	98	b
3	etx	35	#	67	C	99	c
4	eot	36	$	68	D	100	d
5	enq	37	%	69	E	101	e
6	ack	38	&	70	F	102	f
7	bel	39	'	71	G	103	g
8	bs	40	(	72	H	104	h
9	ht	41	)	73	I	105	i
10	lf	42	*	74	J	106	j
11	vt	43	+	75	K	107	k
12	ff	44	,	76	L	108	l
13	cr	45	-	77	M	109	m
14	so	46	.	78	N	110	n
15	si	47	/	79	O	111	o
16	dle	48	0	80	P	112	p
17	dc1	49	1	81	Q	113	q
18	dc2	50	2	82	R	114	r
19	dc3	51	3	83	S	115	s
20	dc4	52	4	84	T	116	t
21	nak	53	5	85	U	117	u
22	syn	54	6	86	V	118	v
23	etb	55	7	87	W	119	w
24	can	56	8	88	X	120	x
25	em	57	9	89	Y	121	y
26	sub	58	:	90	Z	122	z
27	esc	59	;	91	[	123	{
28	fs	60	<	92	\	124	|
29	gs	61	=	93	]	125	}
30	rs	62	>	94	^	126	~
31	us	63	?1	95	_	127	del

Wyrażenia logiczne

Wyrażenia w których występują operatory logiczne.
Operator logiczny – operator który działa na typach numerycznych, znakowych oraz logicznych i wynikiem operacji jest typ logiczny. Są dwa typy operatorów logicznych

- **Operatory relacji**
- **Operatory „combinational” (koniunkcji, alternatywy, negacji, tożsamości oraz nierównoważności)**

Operatory relacji – działają na dwu numerycznych lub dwu znakowych operandach, wynikiem działania operatora jest wartość logiczna

Arytmetyczne operatory relacji:

F90 (nowy styl)	F77 (stary styl)	Znaczenie
==	.EQ.	równe
/=	.NE.	różne
>	.GT.	większe niż
>=	.GE.	większe lub równe
<	.LT.	mniejsze
<=	.LE.	mniejsze lub równe

Wyrażenia logiczne

W przypadku porównywania ciągów znakowych – aby mieć pewność Że ciągi znakowe będą porównywane zgodnie z tablicą ASCII, należy Użyć funkcji LLT, LLE, LGT. LGE

Funkcja i argumety	Typ argumentu	Typ wyniku	Opis
LLT(str1, str2)	CHAR	LOG	Prawda, jeśli str1< str2 zgodnie z ASCII
LLE(str1, str2)	CHAR	LOG	Prawda, jeśli str1<=str2 zgodnie z ASCII
LGT(str1, str2)	CHAR	LOG	Prawda, jeśli str1> str2 zgodnie z ASCII
LGE(str1, str2)	CHAR	LOG	Prawda, jeśli str1>=str2 zgodnie z ASCII

Tabele prawdy

Tablica prawdy dla operatorów .AND., .OR., .EQV. oraz .NEQV.

p	q	p .AND. q	p .OR. q	p .EQV. q	p .NEQV. q
.FALSE.	.FALSE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.
.FALSE.	.TRUE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.TRUE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.TRUE.	.TRUE.	.TRUE.	.TRUE.	.TRUE.	.FALSE.

Tabela prawdy dla operatora .NOT.

p	.not. p
.TRUE.	.FALSE.
.FALSE.	.TRUE.

Kolejność działań - od najwyższej

- Wszystkie operacje arytmetyczne
- Wszystkie operacje relacji, obliczenia są wykonywane od lewej do prawej
- Wszystkie operatory negacji – operator .NOT.
- Wszystkie operatory .AND., obliczenia są wykonywane od lewej do prawej
- Wszystkie operatory .OR., obliczenia są wykonywane od lewej do prawej
- Wszystkie operatory .EQV. I .NEQV., obliczenia od lewej do prawej

Funkcje matematyczne

Fortran definiuje tzw. funkcje grupowe oraz konkretna (ang generic and specific names). Nazwą konkretną jest jednoznaczna nazwa pojedynczej funkcji wewnętrznej o ściśle określonym typie argumentu (lub argumentów) i wyniku. Nazwa grupowa – to grupa funkcji (o podobnym sensie matematycznym) z której kompilator w czasie kompilacji wybiera odpowiednia nazwę konkretną np. nazwa główna **LOG** pozwala odwołać się do funkcji

- **ALOG** – argument i wynik typu REAL
- **DLOG** – argument i wynik typu DOUBLE PRECISION
- **CLOG** – argument i wynik typu COMPLEX

Funkcje matematyczne

W F77 jest 5 funkcji matematycznych które są określone dla argumentów zespolonych, wynikiem też jest liczba zespolona

- *=**EXP**(RDX) - funkcja wykładnicza e^x
- *=**LOG**(RDX) – logarytm naturalny
- *=**SIN**(RDX) – sinus, argument w radianach
- *=**COS**(RDX) – cosinus, argument w radianach
- *=**SQRT**(RDX) – pierwiastek kwadratowy

Oznaczenie: * = - oznacza, że wynik funkcji jest taki sam jak wynik argumentu; IRDX – określa jaki jest typ argumentu

- I => INTEGER
- R => REAL
- D => DOUBLE PRECISION
- X => COMPLEX

Funkcje matematyczne

Funkcje trygonometryczne

- ***=LOG10(RD) –** logarytm dziesiętny
- ***=TAN(**RD) - tangens. Argument w radianach
- *=**ASIN(**RD) – arcus sinus, wynik w zakresie od -PI/2 do +PI/2
- *=**ACOS(**RD) – arcus cosinus, wynik w zakresie od 0 do +PI
- *=**ATAN(**RD) – arcus tangens, wynik w zakresie -PI/2 do PI/2
- *=**ATAN2(**RD, RD) – arcus tangens arg1/arg2, wynik w zakresie -PI do +PI, oba argumenty muszą być różne od zera
- *=**SINH(**RD) – sinus hiperboliczny
- *=**COSH(**RD) - cosinus hiperboliczny
- *=**TANH**(RD) – tangens hiperboliczny

Funkcje matematyczne

Funkcje służące do zmiany typu danych

- I=**INT(**IRDX) – zamiana na typ INTEGER poprzez obcięcie części ułamkowej
- I=**NINT(**RD) – zamiana na typ INTEGER do najbliższej liczby całkowitej
- R=**REAL(**IRDX) – zamiana na typ REAL
- D=**DBLE**(IRDX) – zamiana na typ DOUBLE PRECISION
- X=**CMPLX(**IRDX) – zamiana na typ COMPLEX, w przypadku gdy argument jest typu IRD to część urojona jest zero
- X=**CMPLX**(IRD, IRD) – zamiana na typ COMPLEX

Inne funkcje

- *=**AINT(**RD) – obcina część ułamkową (podobnie jak INT) ale zachowuje typ wyniku (tzn. typ wyniku jest taki sam jak typ argumentu)
- *=**ANINT(**RD) – zaokrągla do najbliższej liczby całkowitej, zachowuje typ wyniku

Funkcje matematyczne

Inne funkcje – dalsza część

- *=**ABS(**IRD) – wartość bezwzględna z liczby
- R=**ABS(**X) – moduł z liczby zespolonej, tj. pierwiastek z sumy kwadratów części rzeczywistej i urojonej
- =**MOD(**IRD, IRD) – oblicza resztę z dzielenia A1/A2, czyli A1-INT(A1/A2)*A2
- *=**SIGN(**IRD, IRD) – funkcja znaku. Jeśli A2 jest ujemne to wynikiem jest -ABS(A1), w przeciwnym przypadku wynikiem jest ABS(A1)
- *=**DIM(**IRD, IRD) - zwraca dodatnią różnice A1 i A2 tzn. jeśli A1>A2 zwraca A1-A2, w przeciwnym przypadku zwraca zero
- D=**DPROD**(R, R) – oblicza iloczyn dwu liczb REAL
- R=**AIMAG**(X) – część urojona liczby zespolonej.
- R=**REAL(**X) – część rzeczywista liczy zespolonej
- X=**CONJG(**X) - sprzężenie zespolone liczby zespolonej

Funkcje matematyczne

Są dwie funkcje, które przyjmują zmienną liczbę argumentów

- *=**MIN(**IRD, IRD, ...) - najmniejsza z liczb
- *=**MAX(**IRD, IRD, ...) - największa z liczb

Funkcje które operują na łańcuchach (o LEN, INDEX, CHAR i ICHAR już było)

- L = **LGE(**C, C) - leksykalnie większy lub równy
- L = **LGT**(C, C) - leksykalnie większy
- L = **LLE(**C, C) – leksykalnie mniejszy lub równy
- L = **LLT(**C, C) – leksykalnie mniejszy

gdzie: L – oznacza typ LOGICAL, C – oznacza typ CHARACTER

Dziękuję za uwagę