LANGUAGE COMPARISON

	Fortran (95+)	Ada	C	Pascal	Julia

comments	`! line`
	`NO block comments`
		`– line`
		`NO block comments`
			`// line`
			`/* block */`
				`// line`
				`(* block *)`
				`{ block }`
					`# line`
					`#= block =#`

basic data types	`real`
	`integer`
	`character`
	`logical (.true. .false.)`
		`float`
		`integer`
		`character`
		`boolean (true, false)`
			`float, double`
			`int`
			`char`
			`bool (C99)`
				`real`
				`integer`
				`char`
				`boolean (TRUE, FALSE)`
					`Float16, Float32, Float64`
					`Int8, Int16, Int32, Int64`
					`unicode`
					`boolean`

variable declaration + initial value	`T :: id` `T :: id`	`id : T;` `id : T := initV;`	`T id;` `T id = initV;`	`id : T;` `id : T = initV;`	`dynamic declaration` `id = initV`

block statements S, S1, S2, etc.	`statement1`
	`statement2`
	`...`
	`statementn`
		`statement1;`
		`statement2;`
		`...`
		`statementn;`
			`{`
			`  statement1;`
			`  statement2;`
			`  ...`
			`  statementn;`
			`}`
				`begin`
				`  statement1;`
				`  statement2;`
				`  ...`
				`  statementn;`
				`end`
					`statement1`
					`statement2`
					`...`
					`statementn`

if-else	**`if`** `(B)` **`then`**
	`  S1`
	`<else`
	`  S2>`
	`end if`
		`if` `B` **`then`**
		`  S1`
		`<else`
		`  S2>`
		`end if;`
			`if` `(B)`
			`  S1`
			`<else`
			`  S2>`
				`if` `B` **`then`**
				`  S1`
				`<else`
				`  S2>`
					`if` `B`
					`  S1`
					`<else`
					`  S2>`
					`end`

nested if	**`if`** `(B1)` **`then`** `S1` **`elseif`** `(B2)` **`then`** `S2` **`…`** **`elseif`** `(Bn)` **`then`** `Sn` **`<else`** `Sn+1>` **`end if`**	**`if`** `B1` **`then`** `S1` **`elsif`** `B2` **`then`** `S2` **`…`** **`elsif`** `Bn` **`then`** `Sn` **`<else`** `Sn+1>` **`end if;`**	**`if`** `(B1) S1` **`else if`** `(B2) S2` **`…`** **`else if`** `(Bn) Sn` **`<else`** `Sn+1>`	**`if`** `B1` **`then`** `S1` **`else if`** `B2` **`then`** `S2` **`…`** **`else if`** `Bn` **`then`** `Sn` **`<else`** `Sn+1>`	**`if`** `B1 S1` **`elseif`** `B2 S2` **`…`** **`elseif`** `Bn Sn` **`<else`** `Sn+1>` **`end`**

case	**`select case`** `(X)`
	`  `**`case`** `(V1) S1`
	`  `**`case`** `(V2) S2`
	`  …`
	`  `**`case`** `(Vn) Sn`
	`  `**`<case default`** `Sn+1>`
	`end select`
		`case` `X` **`of`**
		`  `**`when`** `V1 => S1;`
		`  `**`when`** `V2 => S2;`
		`  …`
		`  `**`when`** `Vn => Sn;`
		`  `**`<when others`** `=> Sn+1;>`
		`end case;`
			`switch` `(X){`
			`  `**`case`** `V1 : S1;` **`<break;>`**
			`  `**`case`** `V2 : S2;` **`<break;>`**
			`  …`
			`  `**`case`** `Vn : Sn;` **`<break;>`**
			`  <`**`default`** `: Sn+1;>`
			`}`
				`case` `(X)` **`of`**
				`  V1 : S1;`
				`  V2 : S2;`
				`  …`
				`  Vn : Vn;`
				`  <`**`otherwise:`** `Sn+1>`
				`end;`
					`n/a`

case expression examples	`X = int, char, boolean exp`
	`Examples of V`
	`  1`
	`  2:10`
	`  11:`
	`  1,3,5,7:9`
		`X = discrete variable`
		`Examples of V`
		`  5`
		`  5|8|23`
		`  100..125`
		`  50|60|70..75|80`
			`X = discrete variable`
			`Examples of V`
			`  3`
			`  'b'`
				`X = int, char, boolean, enum`
				`Examples of V`
				`  3`
				`  1..5`
				`  'a','e','i','o','u'`	

for	**`do`** `I = X1, X2 [,X3]` `  S` **`end`**	**`for`** `I` **`in`** `X1..X2` **`loop`** `  S` **`end loop;`**	**`for`** `(I=X; B; I=EXP)` `S`	**`for`** `I := X1 [`**`down`**`]`**`to`** `X2` **`do`** `S`	**`for`** `I = X1:X2` `  S` **`end`**
while	**`do while`** `(B)` `  S` **`end do`**	**`while`** `B` **`loop`** `  S` **`end loop;`**	**`while`** `(B)` `S`	**`while`** `B` **`do`** `S`	**`while`** `B` `  S` **`end`**

repeat/until	**`do`**
	`  S`
	`until` `(B)`
		`loop`
		`  S`
		`  `**`exit when`** `B;`
		`end loop;`
			`do` `{`
			`  S`
			`}` **`while`** `(B);`
				`repeat`
				`  S`
				`until` `B;`
					`n/a`

generic loop	**`do`** `  S` **`end do`**	**`loop`** `  S` **`end loop;`**	`n/a`	`n/a`	`n/a`

LANGUAGE COMPARISON

	Fortran (95+)	Ada	C	Pascal	Julia
exit loop	`exit`	`exit` `exit when B;`	`break`	`break`	`break`
next cycle of loop	`cycle`	`n/a has to be done using goto`	`continue`	`continue`	`continue`
arrays (size a..b)	`T, dimension(a:b) :: A`	`A : array(a..b) of T;`	`T A[b];`	`A : array[a..b] of T;`	`A = array(T,a,b)`
strings (length b)	`character (len=b) :: Str`	`Str : string(1..b);`	`char Str[b];`	`type Str = packed array[1..b] of char;`	`Str = "xyz"`
multi-dimensional arrays (numbers) r=rows, c=columns	`T, dimension(r,c) :: x`	`x : array(1..r,1..c) of T;`	`int x[r][c];`	`x : array[1..r,1..c] of T;`	`x = Array{T}(undef, r, c)`

function	`function add(a,b) result(r)`
	`  integer, intent(in) :: a,b`
	`  integer :: r`
	`  r = a + b`
	`end function add`
		`function add(a: integer; b: integer) return integer is`
		`begin`
		`  return a + b;`
		`end add;`
			`int add(int a, int b)`
			`{`
			`  return a + b;`
			`}`
				`function add(a, b: integer):integer;`
				`begin`
				`  add := a + b;`
				`end;`
					`function add(a,b)`
					`  return a + b`
					`end`

procedure
(no return)
	`subroutine printSum(a,b)`
	`  integer, intent(in) :: a,b`
	`  print, a+b`
	`end subroutine printSum`
		`procedure printSum(a: in integer; b: in integer) is`
		`begin`
		`  put(a+b);`
		`end printSum;`
			`void printSum(int a, int b)`
			`{`
			`  printf("%d", a+b);`
			`}`
				`procedure printSum(a, b: integer);`
				`begin`
				`  writeln(a+b);`
				`end;`
					`function printSum(a,b)`
					`  println(a+b)`
					`end`

procedure
(value return)
	`subroutine squared(x,sq)`
	`  real, intent(in) :: x`
	`  real, intent(out) :: sq`
	`  sq = x * x`
	`end subroutine squared`
		`procedure squared(x: in float; sq: out float) is`
		`begin`
		`  sq := x * x;`
		`end squared;`
			`void squared(float x, float *sq)`
			`{`
			`  *sq = x * x;`
			`}`
				`procedure squared(x: real; var sq: real);`
				`begin`
				`  sq := x * x;`
				`end;`
					`function squared(x)`
					`  sq = x * x`
					`  return sq`
					`end`

recursion	`recursive function fact(n)`
	`result (f)`
	`  integer, intent(in) :: n`
	`  integer :: f`
	`  if (n == 0) then`
	`  f = 1`
	`  else`
	`  f = n * fact(n-1)`
	`  end if`
	`end function fact`
		`function fact(n: integer) return Long_Integer is`
		`begin`
		`  if n = 0 then`
		`  return 1;`
		`  else`
		`  return Long_Integer(n) * fact(n-1);`
		`  end if;`
		`end fact;`
			`int fact(int n)`
			`{`
			`  if (n == 0)`
			`  return 1;`
			`  else`
			`  return n * fact(n-1);`
			`}`
				`function fact(n: integer): integer;`
				`begin`
				`  if (n = 0) then`
				`  fact := 1;`
				`  else`
				`  fact := n * fact(n-1);`
				`end;`
					`function fact(n)`
					`  if n == 0`
					`  return 1`
					`  else`
					`  return n * fact(n-1)`
					`  end`
					`end`

standard input	`read(*,*) V` `read(*,format) V`	`get(V)`	`scanf(format, &V);`	`read(V);` `readln(V);`	`s = readline()` `V = parse(T, chomp(s))`
standard output	`write(*,*) V` `write(*,format) V`	`put(V)`	`printf(format, V);`	`write(V);` `writeln(V);`	`print(V)` `println(V)`

KEY									
	S = block statement	B = boolean (logical) expression	T = datatype	< > = optional terms					
	I = integer variable identifier	X = a scalar expression	EXP = expression of any type	V = variable identifier					