NOTE SUI LINGUAGGI DI PROGRAMMAZIONE

	DIAGRAMMA BLOCCHI	VISUAL BASIC	PASCAL	C++
Definizione variabili		**DIM** *nome_variabil*e **AS** tipo_dato	**VAR** nomeVar : tipodato[lunghezza];	tipo_dato *nome_variabile* = valore_iniziale **INT** I = 0
Tipo variabili		**STRING** alfanumerico **SINGLE** (reale fino 6 cifre) **DOUBLE** (reale fino 14 cifre) **BYTE** (0 -128) **INTEGER** (-32768 +32768) **LONG** numeri interi	**CHAR** **STRING** alfanumerico **REAL** **BYTE** **INTEGER**	**CHAR** – singolo carattere per definire una stringa **CHAR** nome_stringa[lunghezza] char S[15] definisce una stringa di nome S lunga 15 caratteri **FLOAT** – reale singola precisione **DOUBLE** – reale doppia precisione **SHORT INT** – intero -128 +127 **INT** – intero -32768 +32768 **LONG INT** – intero
Dichiarazioni VETTORI		**DIM** *nomeVettore* (*numero_elem*) **AS** *tipodato* oppure **DIM** *nome_matrice* (*indice_minimo* **TO** *indice_massimo)* **AS** *tipo* DIM nomi (50) as string DIM nomi (1 TO 50) as string	nome_variabile: **ARRAY** [1 ..n] **OF** tipo_var	Tipo nome_var [num_elem] INT vett[10] vettore di 10 interi da vett[0] a vett[9] INT vett[]={1,2,3,4,5} vettore di 5 elementi inizializzato in fase di dichiarazione INT matrice[2][3]; matrice 2x3

Operatori aritmetici		+
		-
		*
		/
		\ (divisione intera)
		MOD (modulo)
		x MOD y da resto di x \ y
		
		= (per assegnare risultato)
		^ elevamento a potenza
			+
			-
			*
			/
			DIV (divisione intera)
			MOD (modulo)
			x MOD y da resto di x \ y
			
			:= (per assegnare risultato)
			^ elevamento a potenza
				+
				-
				*
				/
				
				% (modulo)
				x%y da resto di x/y
				
				= (per assegnare risultato)
				
				Particolarità di C++, sono corrette le seguenti istruzioni:
				
				a = 2 + (b = 5);
				che è equivalente a:
				b = 5;
				a = 2 + b;
				è anche possibile
				
				a = b = c = 5;
				che assegna 5 alle variabili a, b, e c

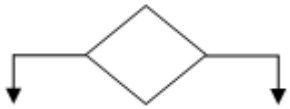
Operazioni in forma concisa
 	
Sono una particolarità di C++
				Operatori di assegnamento composti (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)
				
				a += y; equivale a= a + y;
				
				i+=1 significa i=i+1 uguale a i++
				
				a -= 5; equivale a = a - 5;
				
				a /= b; equivale a = a / b;
				
				prezzo *= numero + 1;
				equivalente
				prezzo = prezzo * (numero + 1);

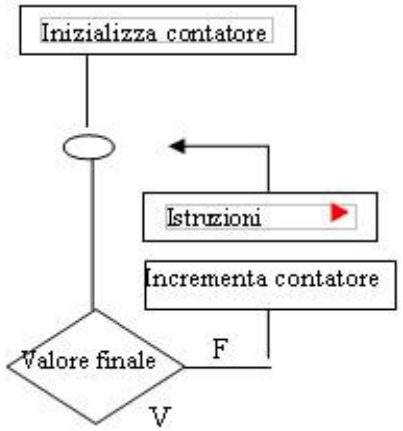
Incremento e decremento		X = x+1	X = x+1;	Incremento (++) e decremento (--) aumentano o diminuiscono di 1 il valore di una variabile e sono equivalenti a **`+= 1 e -= 1`** rispettivamente.
				Sono equivalenti
				a++; a += 1; a = a+1;
				Si possono usare sia come *prefissi* (**++a**) che come *postfissi* (**a++**).
				in espressioni semplici
				a++ è uguale a **++a**
				in altri casi in cui viene utilizzato il risultato dell'operazione nella valutazione di un'altra espressione essi assumono un significato diverso.
				Se l'operatore di incremento viene usato come *prefisso* (**++a**) l'espressione viene valutata usando il valore già incrementato;
				se l'operatore di incremento viene usato come *postfisso* (**a++**) l'espressione viene valutata usando il valore non incrementato.
				Esempio:
				`B = 3;`
				`A = ++B;`
				`// A è 4, B è 4`
				
				`B = 3;`
				`A = B++;`
				`// A è 3, B è 4`

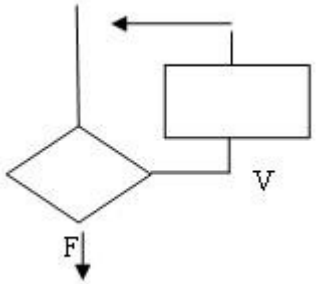
Operatori relazionali e logici		>= <=	>= <=	>= <=

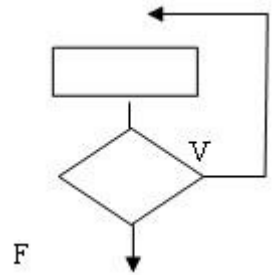
		= uguale <> diverso **NOT** not **AND** and **OR** or	= uguale <> diverso **NOT** not **AND** and **OR** or	== uguale **!=** diverso **!** not **&&** and || or operatori relazionali vengono eseguiti dopo operatori aritmetici && ha precedenza su ||

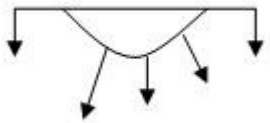
Espressioni condizionali				cond? E2 : E3
				se cond è vero dammi il valore E2
				altrimenti E3
				
				z= (a>b)?a:b
				se a>b
				  z=a
				altrimenti
				  z=b

Istruzioni controllo SELEZIONE		**IF** *condizione* **THEN**
		  istruzione
		  .....
		ELSE
		  istruzione
		  .....
		END IF
			IF *condizione* **THEN**
			  istruzione1
			ELSE
			  istruzione2
			
			(quando c'è una unica istruzione)
			
			IF *condizione*
			THEN BEGIN
			  istruzione
			  .....
			ELSE BEGIN
			  istruzione
			  .....
			END;
			(quando ci sono più istruzioni)
				IF(condizione)
				  istruzione1;
				ELSE
				  istruzione2;
				
				<u>NOTA: il ; finale è obbligatorio</u>
				se vanno eseguite più istruzioni devono essere racchiuse tra parentesi graffe
				IF(espressione)
				  {
				  istruzione_a;
				  istruzione_b;
				  istruzione_c;
				  istruzione_d;
				}
				ELSE
				  istruzione2;

Salti incondizionati			**GOTO** labelA;	**GOTO** labelA; trasferisce l'esecuzione all'etichetta labelA che è così definita nel programma labelA:
Cicli determinati		**FOR** *contatore = iniziale* **TO** *finale* **STEP** *passo* *Istruzioni* ..... **NEXT**	**FOR** *contatore := iniziale* **TO** *finale* ***DO BEGIN*** *Istruzioni* ..... **END:**	**FOR** (espres1;espres2;espres3) istruzione; Esegui a partire da valore iniziale espres1 fino a espres2 incrementando variabile for (i=1;i<=n;i++) istruzione; equivale a for i=1 to n incrementando i=i+1 esegui istruzione; NOTA se ci sono più istruzioni usare parentesi graffe

Cicli per vero
 	
TEST all'inizio
		**DO WHILE** condizione
		istruzione
		…..
		LOOP
			WHILE condizione **DO**
			istruzione
			
			oppure
			
			WHILE condizione **DO**
			BEGIN
			istruzione
			…...
			END;
				WHILE (condizione)
				istruzione;
				
				NOTA: se ci sono più istruzioni si usano parentesi graffe
				Si rimane nel ciclo finchè espres è VERA

Cicli per vero
 	
TEST alla fine
		**DO**
		
		…..
		LOOP WHILE condizione
				DO
				{
				blocco istruzioni;
				}
				WHILE (condizione);
				
				Si esce quando espres è FALSA

Selezione multipla		**SELECT CASE** *variabile_di_controllo* **CASE** *valore1* *Istruzione1* **CASE** *valore2* *Istruzione2* **CASE** *valore3* *Istruzione3* ….. **END CASE**	**CASE** *variabile OF* *valore1: Istruzione1;* *valore2: Istruzione2;* ….. **END**	**SWITCH** (scelta) **CASE** costante1: istruzione/blocco; **CASE** costante2: …..

Funzioni		**Public Function** *nome_funzione (Parametri* **As** *tipo)* **As** *tipo*
		…..
		….
		….
		End Function
			Function *nome_funzione (Parametri) : tipo*
			definizione costanti
			variabili
			BEGIN
			istruzioni
			….
			….
			nome_funzione:= valore:
			End;
				(tipo risultato) NOME_Funzione (elencoparametri)
				{
				…
				….**RETURN** (espressione);
				}