

Sommario

Variabili e tipi di dati	3
Comando di assegnazione	5
Operazioni aritmetiche	7
Operazioni logiche	9
Comandi di Input e Output	12
Conclusione	13

Cosa impareremo in questo modulo:

- Che cosa è un programma?
- Variabili e tipi di dati
- Assegnazione di Comandi
- Operazioni aritmetiche
- Operazioni logiche
- Operazioni di confronto
- Input e Output Comandi

Programma

Come abbiamo visto in precedenza, un programma è un algoritmo che può essere compreso ed eseguito da un computer. Perché ciò avvenga, deve essere scritto in ciò che chiamiamo un linguaggio di programmazione. I linguaggi di programmazione sono costruiti per essere comprensibili all'uomo, che può, a sua volta, scrivere programmi, facilmente traducibili in una forma che può essere letta ed eseguita da un computer.

Ci sono numerosi linguaggi di programmazione per scopi diversi (e preferenze). Possiamo citare come esempi: C, C ++, Java, C #, Ruby, Pascal, Smalltalk, Visual Basic, COBOL, FORTRAN, LISP, JavaScript, ActionScript, Prolog, Luna tra una moltitudine di altri.

Dato che lo scopo di questo corso è l'insegnamento della logica della programmazione e non quello di uno specifico linguaggio, useremo un linguaggio immaginario che contiene le operazioni di base che la stragrande maggioranza dei linguaggi offrono.

Chiamiamo il codice descritto in questo linguaggio fittizio, pseudocodice (pseudo significa falso, che non è reale).

Anche se il nostro non è un modo "ufficiale" per scrivere programmi, i codici scritti in pseudocodice possono essere facilmente convertiti in qualsiasi linguaggio di programmazione vero e proprio, esso è pertanto uno strumento molto utile non solo per la didattica della programmazione, ma anche per aiutare nella progettazione di algoritmi e nello scambio di informazioni, perché in tal modo i programmi saranno scritti in un linguaggio molto vicino al nostro linguaggio naturale.

Useremo parole senza l'accento, perché in questo modo il nostro pseudolinguaggio sarà ancora più vicino a un linguaggio di programmazione.

I programmi scritti in questo linguaggio devono seguire una struttura rigida, di seguito riportata:

```
Programma: <Nome dell'Algoritmo o del Programma>;
```

```
Variabili
```

```
    (dichiarazione delle variabili);
```

```
Inizio
```

```
    (operazioni);
```

```
Fine.
```

Il programma dovrebbe iniziare con una linea "Programma", seguita da due punti e dal nome dell'algoritmo o del programma e terminare con un punto e virgola. In questo linguaggio tutte le istruzioni devono terminare con tale simbolo, che indica la fine dell'istruzione.

Dopo la linea che "battezza" il programma, abbiamo una linea con la parola "Variabili" (senza nessun punto e virgola alla fine), seguito da una o più righe con la dichiarazioni delle variabili (ognuna separata dalla virgola). In questo blocco sono definite le variabili che saranno usate nel codice. Vedremo nei prossimi argomenti cosa sono le variabili e come devono essere dichiarate.

Al blocco di dichiarazione delle variabili segue una linea con la parola "Inizio" (anche essa senza nessun punto e virgola), che delimita il punto in cui il programma inizia effettivamente. Da qui iniziano le operazioni che il computer deve eseguire, in sequenza. Quando il computer esegue il programma eseguirà le operazioni sequenzialmente partendo dalla prima e fino a raggiungere l'ultima (ricordando che ogni istruzione è seguita da un punto e virgola). L'ultima riga del programma deve contenere la parola "Fine" seguita da un ultimo punto, che indica al computer il programma è terminato.

Le parole che fanno parte del linguaggio, comunemente chiamate "parola riservate", sono contrassegnate in grassetto in tutti gli esempi di questo corso.

Variabili e tipi di dati

Abbiamo detto in precedenza che tutte le informazioni utilizzate dal computer sono memorizzate in una memoria che può essere immaginata come un cassetto, dove tutte le informazioni sono memorizzate in un cassetto con un nome. Il processore può, attraverso questo nome, trovare questo cassetto e recuperare le informazioni in esso contenute, o salvare nuove informazioni lì dentro.

Affinché le informazioni contenute nel "cassetto" possano essere modificato dal processore, ad essa sarà associato il nome di una variabile. Ogni variabile ha un nome che consente al processore di utilizzare le informazioni ad essa associate. Tuttavia, è necessario riservare lo spazio ad essa destinato, prima del suo uso, per questo motivo dobbiamo dichiarare tutte le variabili che verranno utilizzate prima dell'inizio del programma. Perciò esiste il blocco "**Variabili**" nel pseudocodice.

```
Variabili
```

```
    (dichiarazione delle variabili);
```

Una dichiarazione di variabile ha il seguente formato:

```
var <nome_variabile>: <tipo>;
```

LOGICA DI PROGRAMMAZIONE - MODULO 1

In questa dichiarazione <nome_variabile> può essere una sequenza di una o più lettere non accentate (maiuscole e minuscole), numeri e simboli “_” (chiamato di sottolineatura), con la condizione che essa inizi con una lettera o il simbolo “_”. Inoltre, non possono essere utilizzate parole riservate del linguaggio come nomi di variabili. Esempi di nomi di variabili validi sono: somma, Valore_totale, io, tasso_di_conversione, subTotale1, _risultato o a1234k_x_b. Esempi di nomi non validi sono: 12variabile, -risultato, \$altri&caratteri, Inizio, Variabili, 1, 123 e 123a.

Dopo il nome della variabile seguito dal carattere due punti viene il tipo della variabile. Una variabile può solo memorizzare dati che hanno le stesse caratteristiche indicate dal tipo con cui è stata definita. Così, se una variabile è definita per ricevere un valore numerico non dovrebbe essere utilizzata per memorizzare, ad esempio, un testo. Per questo motivo dobbiamo anche indicare quali sono le informazioni che intendiamo memorizzare in una variabile prima di utilizzarla.

Per questo corso si ritiene che i tipi validi sono:

Tipo	Descrizione ed esempio di valori assegnabili
Intero	Numeri interi. Ad es.: 1, 100, 1056
Reale	Numero decimale. Ad es.: 1.00, 3.145, 100.32
Testo	Testo contenente lettere, cifre o simboli. Ad es.: “Prova 1, 2, 3”, “Questo è un esempio di testo”, “43”
Carattere	Un solo carattere, comprendendo lettere o simboli. Ad es.: ‘1’, ‘a’, ‘!’
Logico	Valore logico, VERO o FALSO

Si noti che, nel caso di variabili di tipo testo, il valore viene visualizzato tra virgolette doppie (“”), mentre nel caso di caratteri è presentato con virgolette singole (‘’). Questo metodo è seguito da diversi linguaggi di programmazione per evitare che un valore di testo o carattere sia confuso con il nome di una variabile (che similmente può essere formato da lettere, numeri e alcuni simboli).

Alcuni esempi di dichiarazioni di variabili:

```
var contatore: intero;  
var tasso_di_conversione: reale;  
var domanda: testo;  
var lettera: carattere;  
var carta_valida: logico;
```

Possiamo dichiarare più variabili nella stessa istruzione, ma tutte dello stesso tipo. Per fare questo abbiamo separare i nomi delle variabili da virgole prima di dichiararne il tipo. Un esempio di questo tipo di istruzione è:

```
var tasso_di_conversione, valore_in_euro, valore_in_dollari: reale;
```

Questo è un esempio di una dichiarazione di variabili che possiamo usare in un programma di conversione di valuta, in cui abbiamo riservato spazio di memoria per la quantità di euro da convertire, il valore del tasso di conversione e l'importo convertito in dollari.

Dovendo trattare valori di valuta, che hanno tutte sia una parte intera sia una parte decimale, il tipo di dato più appropriato per memorizzare questi valori è reale, perché questo tipo di variabile supporta i decimali. È importante definire adeguatamente i tipi di variabili secondo le informazioni che intendiamo utilizzare nei nostri programmi.

Una buona pratica di programmazione è quella di utilizzare nomi mnemonici per le variabili che identifichino le informazioni che intendiamo memorizzare. Quando cercheremo di leggere e comprendere un programma, sarà molto più facile identificare ciò che rappresentano queste variabili. Tutto diventa più chiaro quando si ricorda l'analogia delle variabili con i cassette. Un'etichetta che dà nome ad un cassetto, deve esprimere con chiarezza il contenuto del cassetto, senza doverlo aprirlo. Quindi, tornando all'esempio di programma di conversione di valuta, è molto più facile comprendere che cosa viene salvato nelle variabili `tasso_di_conversione` e `valore_in_dollari`, di quanto lo sarebbe stato se avessimo chiamato le stesse `AA1` e `_b45`.

Comando di assegnazione

Iniziamo qui a vedere le operazioni che possono essere eseguite all'interno di un programma per computer. I primi comandi che studieremo sono le istruzioni di assegnazione.

Abbiamo visto nel precedente paragrafo quali sono e a cosa servono le variabili, le entità che servono a memorizzare i dati utilizzati dal computer in un programma. Abbiamo anche visto che dobbiamo dichiarare queste variabili prima di utilizzarle e come possiamo fare questa operazione. Ora dobbiamo cominciare ad utilizzarle. Il primo comando che studieremo è l'istruzione di assegnazione, finalizzata ad attribuire (definire) un valore ad una variabile.

Nella sua forma più semplice l'istruzione di assegnazione ha la seguente struttura:

```
<nome_variabile> ← <valore>;
```

Dove `<nome_variabile>` indica il nome di una variabile precedentemente dichiarata e `<valore>` il valore che intendiamo assegnare a questa variabile. Tale valore deve essere dello stesso tipo della variabile affinché il comando sia valido. Quando il computer esegue questo comando, il valore viene memorizzato nella variabile che si trova nel lato sinistro dell'assegnazione. Osserviamo il primo esempio di programma qui di seguito:

```
1 Programma: PrimoEsempio;
2 Variabili
3   var variabileEsempio : intero;
4 Inizio
5   variabileEsempio ← 3;
...
n Fine.
```

Nella riga 5 il processore è incaricato di assegnare il valore costante 3 nella variabile `variabileEsempio`. Questo sarà il valore della variabile per tutto il resto della esecuzione del codice o fino al momento in cui si

assegnerà un nuovo valore ad essa. In seguito vedremo come questo valore memorizzato può essere utilizzato in altre operazioni. Si noti che il numero all'inizio di ogni riga non esiste nel programma, è stato inserito solo allo scopo di poter identificare le righe nella spiegazione sopra riportata.

Un altro punto importante è osservare che la dichiarazione della variabile e l'istruzione di assegnazione non sono allineati con i comandi Variabili ed Inizio. La differenza serve ad evidenziare che questi comandi sono all'interno di questi blocchi.

Questa è un'altra tecnica importante che facilita notevolmente la lettura del codice e riceve il nome di indentazione.

Esempi di valori assegnazioni per gli altri tipi di variabili sono le seguenti:

```
tasso_di_conversione ← 1.53;

lettera ← 'T';

domanda ← "Come ti chiami?";

carta_valida ← FALSO;
```

Un altro modo di utilizzare l'istruzione di assegnazione è la seguente:

```
altraVariabile ← variabileEsempio;
```

Questo comando consente di creare una copia del valore memorizzato nella variabile <nome_della_variabile_di_origine> alla variabile <nome_della_variabile_di_destinazione>, vale a dire dopo l'esecuzione dell'assegnazione le due variabili conterranno lo stesso valore. Quindi, se inseriamo nel programma PrimoEsempio la riga numero 6 del codice con il seguente comando:

```
altraVariabile ← variabileEsempio;
```

Dopo l'esecuzione di questa riga, sia il variabileEsempio sia altraVariabile conterranno il valore 3.

Ricordate che affinché il programma sia corretto dopo questo cambiamento deve essere dichiarato la variabile altraVariabile nella sezione Variabili e dello stesso tipo della variabile variabileEsempio. Questo ci porta al seguente programma:

```
1 Programma: PrimoEsempio;
2 Variabili
3   var variabileEsempio, altraVariabile : intero;
4 Inizio
5   variabileEsempio ← 3;
6   altraVariabile ← variabileEsempio;
...
n Fine.
```

Infine, il modo più generale per le istruzioni di assegnazione è:

`<nome_variabale_di_destinazione> ← <espressione>;`

Un'espressione può essere una serie di operazioni aritmetiche (addizione, sottrazione, moltiplicazione, etc.) o operazioni logiche (AND, OR, NOT etc.) che restituiscono un valore che verrà, quindi, memorizzato in `<nome_variabale_di_destinazione>`. Siccome le espressioni aritmetiche restituiscono valori numerici esse potranno essere memorizzate solo in variabili intere o reali. Nel caso di espressioni logiche i cui risultati sono vero o falso, le variabili di destinazione possono solo essere di tipo logico. Vedremo come definire e utilizzare espressioni in un prossimo argomento.

Operazioni aritmetiche

Le operazioni aritmetiche sono operazioni su valori numerici (cioè, nel linguaggio che stiamo usando su valori interi e reali). Le operazioni elementari possono essere combinate per formare un più espressioni matematiche più complesse.

Gli operatori utilizzeranno qui ci sono `+`, `-`, `*`, `/`, `%`, `^` e `radice()`, essi sono descritti nella tabella che segue:

Operatore	Operazione	Esempio
<code>+</code>	Somma	<code>3 + 4</code>
<code>-</code>	Differenza	<code>6 - 2</code>
<code>*</code>	Moltiplicazione	<code>3 * 2</code>
<code>/</code>	Divisione	<code>5 / 2</code>
<code>%</code>	Resto della divisione	<code>5 % 3</code> , il cui risultato è 2
<code>^</code>	Elevamento a potenza	<code>2 ^ 3</code> , il cui risultato è 8
<code>radice(x)</code>	Radice quadrata del numero x	<code>radice(4)</code> , il cui risultato è 2

Possiamo usare queste operazioni per creare un'espressione matematica. Scriviamo il seguente comando:

```
a ← 3 + 2;
...
```

La variabile avrà il valore 5 dopo l'esecuzione di questo comando. Possiamo anche includere variabili nella espressione matematica:

```
1   a ← 3 + 2;
2   b ← 5 + a;
...
```

Nell'esempio precedente, la variabile b dopo l'istruzione di assegnazione, avrà valore 10 (perché il valore di a è 5, dopo l'esecuzione del comando 1). Un altro punto importante da osservare è che prima viene

valutata l'espressione a destra dell'assegnazione, e, solo dopo il calcolo, il risultato è memorizzato nella variabile a sinistra. Consideriamo adesso il seguente frammento di programma:

```
1    a ← 3 + 2;
2    a ← 2 + a;
...

```

Il programma sopra riportato porterebbe la variabile "a" ad assumere il valore 5 dopo l'esecuzione della riga 1. Poi sulla riga 2 la prima operazione che il computer effettuerebbe, sarebbe quella di "trasformare" le variabili nei valori in esse contenuti, quindi, si otterrebbe lo stesso risultato dell'operazione "2 + 5;". Calcolata l'espressione, solo allora il valore ottenuto sarebbe assegnato alla variabile, che conterrebbe il valore 7. In tal modo possiamo utilizzare la stessa variabile sia sul lato destro sia sul lato sinistro di un'istruzione di assegnazione.

Un'altra cosa che dobbiamo ricordare è quella che noi denominiamo precedenza degli operatori, cioè, l'ordine in cui vengono eseguite le operazioni. Questa regola è simile alla regola applicata in un'espressione matematica. Immaginiamo di avere l'espressione "5 + 2 * 3".

Per la precedenza degli operatori, questa espressione sarà interpretata dal computer come "(5 + 2) * 3", determinando il valore 21, risultato di (7*3). Se non esistesse la regola sopra enunciata, un'altra possibilità sarebbe valutare la stessa espressione come "5 + (2 * 3)", che darebbe il valore 11, da (5 + 6). Pertanto, senza regole chiare, il programma potrebbe darci un valore diversi da quelli che ci aspettiamo, il che non è accettabile in termini computazionali.

Per questo motivo è definito un ordine tra gli operatori, come segue:

Operatori in ordine decrescente di precedenza
() parentesi
radice() e ^
*, / e %
+ e -

Nel caso di operatori con la stessa priorità, il computer esegue prima quelli a sinistra nelle operazioni. In questo modo avremmo i seguenti risultati per le seguenti espressioni:

Espressione	Interpretazione dell'elaboratore	Risultato
3 + 2 * 5	3 + (2 * 5)	13
(3 + 2) * 5	(3 + 2) * 5	25
3 + 2 - 1	(3 + 2) - 1	4

$2 - 3 * 4$	$2 - (3 * 4)$	-10
$3 + 2 * 3 ^ 2$	$3 + (2 * (3 ^ 2))$	21

Si noti che possiamo usare le parentesi, come in " $(3 + 2) * 5$ " per modificare l'ordine con cui il computer potrebbe eseguire le operazioni, quindi le parentesi hanno la precedenza più alta tra gli operatori.

Un ultimo dato che vale la pena ricordare è che la divisione (/) si tradurrà in un valore intero se fatto su valori interi (o memorizzato in una variabile intera) e in un valore reale quando viene effettuata tra operandi reali. Quindi, se abbiamo una variabile intera a e una variabile reale b:

Assegnazione	Valore della variabile dopo l'esecuzione del comando
$a \leftarrow 5 / 2$	$a = 2$
$b \leftarrow 5 / 2$	$b = 2.5$
$a \leftarrow 5.0 / 2.0$	$a = 2$
$b \leftarrow 5.0 / 2.0$	$b = 2.5$

Per comprendere meglio i risultati generati dagli operatori / e %, consideriamo ad esempio la seguente divisione:

$$\begin{array}{r} 11 \mid \underline{4} \\ 3 \quad 2 \end{array}$$

Il risultato dell'operazione $11 \% 4$ è 3, poiché 3 è il resto della divisione di 11 per 4. Facendo l'operazione $11/4$, siccome 11 e 4 sono costanti intere, avremo come risultato 2, se la variabile a cui sarà assegnato il risultato è intera. Se questa variabile è di tipo reale, si otterrà come risultato 2.0. Ma se la divisione viene fatta tra i valori 11.0 e 4.0 (che sono reali), e la variabile che riceve il calcolo sarà di tipo intero, si ottiene il valore di 2. D'altra parte, se la variabile che riceverà il calcolo è di tipo reale, il risultato è 2.75.

Operazioni logiche

Gli operatori logici operano solo su valori logici, chiamati anche booleani (VERO/TRUE o FALSO / FALSE). Così, possono essere eseguite solo su operandi dello stesso tipo.

Gli operatori che si possono utilizzare sono: NOT, AND e OR.

L'operatore NOT inverte il valore dell'operando a destra. Così, la tabella seguente mostra i possibili risultati con questo operatore.

Operazione	Risultato
NOT VERO	FALSO
NOT FALSO	VERO

LOGICA DI PROGRAMMAZIONE - MODULO 1

L'operatore AND restituisce valore vero solo se i due operandi hanno valore vero, come nella seguente tabella:

Operazione	Risultato
VERO AND VERO	VERO
VERO AND FALSO	FALSO
FALSO AND VERO	FALSO
FALSO AND FALSO	FALSO

L'operatore OR restituisce valore vero quando almeno uno dei due operandi ha valore vero, come nella seguente tabella:

Operazione	Risultato
VERO OR VERO	VERO
VERO OR FALSO	VERO
FALSO OR VERO	VERO
FALSO OR FALSO	FALSO

Come gli operatori aritmetici, anche gli operatori logici seguono una regola di precedenza:

Operatori in ordine decrescente di precedenza
() parentesi
NOT
AND
O

Come con gli operatori aritmetici, dobbiamo prima risolvere le operazioni che coinvolgono operatori di precedenza più alta. Così otteniamo i seguenti valori per le seguenti espressioni:

Espressione	Interpretazione dell'elaboratore	Risultato
VERO OR VERO AND FALSO	VERO OR (VERO AND FALSO)	VERO
VERO AND (VERO OR FALSO)	VERO AND (VERO OR FALSO)	VERO
VERO AND (NOT VERO OR FALSO)	VERO AND ((NOT VERO) OR FALSO)	FALSO

FALSO OR FALSO OR VERO	(FALSO OR FALSO) OR VERO	VERO
VERO AND FALSO AND VERO	(VERO AND FALSO) AND VERO	FALSO

Possiamo anche usare le variabili logiche al posto delle espressioni vero o falso, come ad esempio $a \text{ AND } b$. Si riportano di seguito alcuni esempi di utilizzo di espressioni logiche nel corpo di un programma (sempre ricordando che le variabili devono essere dichiarate correttamente).

```

1   a ← VERO;
2   b ← FALSO;
3   risultatoAND ← a AND b;
4   risultatoO ← a OR b;
5   risultatoNOT ← NOT a;
```

Operazioni di confronto

Le operazioni di confronto agiscono su due operandi dello stesso tipo e restituiscono un valore logico. Esse sono utilizzate quando abbiamo bisogno di determinare la relazione tra due operandi. Gli operatori di confronto in questo linguaggio sono:

Operatore	Significato
=	Uguale
<>	Diverso
<	Minore
>	Maggiore
<=	Minore o uguale
>=	Maggiore o uguale

Non è necessario stabilire un ordine di precedenza tra questi operatori, perché non possiamo utilizzare più di un operatore di confronto alla volta. Comunque, il risultato di un'operazione di confronto è un valore logico, pertanto la possiamo utilizzare nel mezzo di un'espressione logica. In questo caso, gli operatori di confronto hanno la precedenza sugli operatori logici.

Un esempio di questo è l'espressione " $5 < 2 \text{ AND } \text{VERO}$ ". Questa espressione è eseguita da computer come " $(5 < 2) \text{ AND } \text{VERO}$ ", risultando in un valore falso (poiché 5 è maggiore di 2, l'espressione " $5 < 2$ " restituisce valore FALSO, quindi " $5 < 2 \text{ AND } \text{VERO}$ " genera " FALSO AND VERO ", che dà FALSO).

Programma: VerificaNumeri;

Variabili

var a, b: intero;

var negativo, positivo, neutro, maggiore: logico;

```
Inizio  
  
  a ← 1;  
  
  b ← 2;  
  
  negativo ← a < 0;  
  
  positivo ← a > 0;  
  
  neutro ← a = 0;  
  
  maggiore ← a > b;  
  
Fine.
```

Comandi di Input e Output

Finora abbiamo visto solo i comandi che ci permettono di memorizzare, recuperare e calcolare valori. Ma tali comandi non permettono l'interazione con il programma. Per questo, abbiamo bisogno di comandi di ingresso e di uscita.

Il primo comando importante che usiamo è il comando WRITE, che stampa/visualizza uno o più valori sul monitor. Così, il programma di seguito causerà la visualizzazione del messaggio "Ciao Mondo!!!" sullo schermo del computer. Si noti che, nel programma, il messaggio è racchiuso tra virgolette perché è un testo.

```
Programma: CiaoMondo;  
  
Variabili  
  
Inizio  
  
  write "Ciao Mondo!!!";  
  
Fine.
```

Possiamo usare lo stesso comando per la stampa di variabili come "WRITE a", o espressioni come "WRITE 3 + 2;".

Inoltre, possiamo stampare più valori, variabili o espressioni nella stessa istruzione WRITE utilizzando le virgole tra questi valori, come nel seguente esempio:

```
Programma: ScriviDiversiValori;  
  
Variabili  
  
Inizio  
  
  write "Ciao Mondo!!!";  
  
  write "Il valore di 3+2 è ", 3 + 2;  
  
Fine.
```

Il risultato di questo programma sarebbe una linea sullo schermo con il messaggio "Ciao mondo!!!" e una seconda linea con il messaggio "Il valore 3 + 2 è 5".

Per poter passare valori al computer si può utilizzare il comando READ.

Questo comando deve essere seguito da una variabile che riceverà un valore passato dalla tastiera (quando l'utente preme il tasto ENTER). Si consideri il seguente esempio:

```
Programma: LeggiScriviNome;  
  
Variabili  
  
    var nome : testo;  
  
Inizio  
  
    write "Per favore, digiti il suo nome ";  
  
    read nome;  
  
    write "Salve ", nome, " come va?";  
  
Fine.
```

Questo programma scrive sullo schermo il messaggio " Per favore, digiti il suo nome " e aspetta che l'utente digiti qualcosa sulla tastiera seguita dal tasto INVIO. Dopo questo, si visualizza un messaggio di saluto e il valore della variabile, che contiene ciò che è stato digitato dalla persona.

Nel caso di variabili numeriche, carattere o logiche, il computer visualizzerà un messaggio di errore e interromperà l'esecuzione del programma, se viene inserito un valore non valido.

E' giusto ricordare che è possibile leggere e scrivere più di una variabile utilizzando lo stesso comando di lettura o scrittura.

Con i comandi e gli operatori illustrati, possiamo già fare alcuni semplici programmi.

Conclusione

In questa sezione si apprende che cosa è un programma e qual è la sua struttura di base. Abbiamo anche imparato il concetto di "variabile" e il loro tipo, esse sono locazioni di memoria finalizzate ad archiviare i dati. In relazione alle variabili, abbiamo visto come assegnare valori ed eseguire le principali tipologie di operazioni: aritmetiche, logiche e di confronto. Infine, abbiamo visto come permettere una semplice interattività con i nostri programmi attraverso i comandi di lettura e scrittura. Pertanto, se uno di questi concetti non vi è chiaro, rivedere il materiale!