Modelli e strumenti innovativi per lo studio degli algoritmi nella scuola secondaria.

SANTI Paolo
Docente presso I.T.I.S. "Euganeo", Este (PD)
paoloss@ymail.com

All'interno di molti curricoli delle discipline di studio nella scuola italiana sono presenti esplicitamente l'insegnamento e l'utilizzo degli algoritmi. Per lo studio solitamente si impiega uno di due formalismi: diagrammi di flusso o pseudo codifica. Nel testo si pone l'accento sulla equivalenza dei modelli strutturati. Infine viene presentato un programma per calcolatore, sviluppato appositamente, che fornisce un ambiente pratico e didatticamente corretto per la progettazione dei programmi.

1 Introduzione

L'**insegnamento dell'Informatica** fa da molti anni, a giusto titolo, parte dei programmi didattici di molti corsi di studio della Scuola secondaria superiore. La materia trova spazio in varie tipologie di Istituti. Spesso questa disciplina fa parte di quelle definite "di indirizzo", caratterizzanti gli ultimi anni del corso di studi ma, in base alle ultime modifiche ministeriali diventa anche parte del programma didattico dei primi anni degli istituti tecnologici, acquisendo pure il ruolo di materia formativa "di base".

Gli **ambiti in cui gli insegnanti devono operare si sono allargati** richiedendo approcci e strumenti più accessibili e versatili. I docenti, in qualche modo, si adegueranno ma gli strumenti materiali finora utilizzati sono adeguati?

1.1 Informatica dove e come

Nell'accezione comune, si intende "esperto di Informatica" chiunque abbia una padronanza, vera o presunta, nell'uso del computer. Spesso si ritiene "informatica"="trattamento testi" o simili. Certo tutti sappiamo che le applicazioni dei calcolatori in tutti i campi sono enormi e pervasive: non c'è uno spazio in cui oggi non sia presente un qualche sistema informatico, ma non per questo coloro che utilizzano questi sistemi si possono necessariamente definire esperti di informatica o programmatori.

In questa sede ci interessiamo all'ambito che si riferisce allo studio, alla ideazione e alla rappresentazione dei programmi, di cui una applicazione (non necessaria e non l'unica) è la **programmazione dei calcolatori elettronici**. Questa disciplina non riguarda solo una serie di conoscenze e di abilità: è un modo di osservare, di pensare, una mentalità che si forma in anni di studio e di pratica. Per poter veicolare agli alunni le conoscenze in un modo che sia il più

possibile efficace si deve poter ricorrere ad approcci diversi, complementari o paralleli e a tutti gli strumenti, tradizionali o rivoluzionari che siano.

1.2 I linguaggi per programmare

Esistono moltissimi linguaggi di programmazione, più o meno fortunati. In base alle loro caratteristiche peculiari questi rientrano in alcune tipologie principali. Il tipo di approccio, di modalità operative, di gestione delle informazioni, ecc... definiscono i cosiddetti “paradigmi di programmazione” che equivalgono a visioni molto diverse dei dati e delle operazioni possibili. Si possono suddividere in poche categorie principali: imperativi, funzionali, logici/ dichiarativi. La percentuale maggiore in fatto di diffusione è costituita dai **linguaggi di tipo imperativo** ora evoluti nei **linguaggi ad oggetti**: in questi linguaggi “ogni operazione è un ordine” impartito al calcolatore che, chiaramente, non discute.

I linguaggi logici e funzionali sono maggiormente diffusi a livello universitario e di ricerca; a parte qualche sperimentazione non trovano grandissima applicazione nel mercato e neanche nella didattica della scuola secondaria.

I linguaggi più conosciuti sono: Imperativi: ALGOL, BASIC, FORTRAN, PL/I; Strutturati: Ada, C, Pascal; Orientati ad oggetti: C++, C#, Delphi, Java, PHP, Python, Ruby, Visual Basic per citarne solo una piccola parte; Funzionali: Lisp (e derivati); Dichiarativi/Logici: SQL e Prolog.

Nella scuola italiana, in particolare negli Istituti Tecnici Industriali la situazione più frequente è quella dell'uso di uno o due linguaggi di programmazione orientati agli oggetti utilizzati prima per introdurre le basi della programmazione, poi la programmazione strutturata, per passare infine alla programmazione ad oggetti vera e propria. I linguaggi più usati sono (in ordine alfabetico) C, C++, C#, Java, Pascal, Visual Basic.

Un capitolo a parte è riservato alla gestione dei database in cui ora si usa quasi esclusivamente SQL. In ogni caso anche la realizzazione di sistemi costruiti su basi di dati utilizzano poi un linguaggio strutturato o ad oggetti per la realizzazione della logica dell'applicazione.

2 Problemi, programmi, algoritmi

Lo studio della programmazione ha come obiettivo quello di far acquisire agli studenti la capacità di realizzare programmi per risolvere problemi o classi di problemi simili. Il programma è una successione di azioni secondo un certo linguaggio interpretabile da un calcolatore. Nulla obbliga un programma a terminare. Un algoritmo è una applicazione un numero finito di volte di un numero finito di regole. Ci sono programmi che non sono algoritmi (non terminano) e algoritmi che non sono programmi per computer (non possono essere eseguiti dal calcolatore).

Lo **studio dell'Informatica** si svolge prevalentemente nell'intersezione dei due insiemi, i **programmi che terminano e implementano degli algoritmi**. In questo caso spesso usiamo indifferentemente una delle due parole.

3 Modelli di rappresentazione dei programmi

Per realizzare un programma si parte dal problema, si cerca una soluzione generale e la si traduce in istruzioni del calcolatore. Queste operazioni possono essere condotte in modo più o meno rigoroso. Spesso, per i "programmatori dell'ultima ora", produrre un programma si traduce in un percorso pseudo casuale fatto di prove e modifiche estenuanti e dai risultati poco verificabili.

Per poter ottenere software di qualità bisogna porre attenzione al metodo e soprattutto **usare dei modelli adatti per rappresentare gli algoritmi**.

Questi tipi di schematizzazione si pongono a metà strada tra il problema e la codifica del programma e permettono di astrarre più facilmente e trovare una soluzione in modo più rigoroso che non sia "scrivere delle istruzioni finché non si trova la configurazione che funziona (o che sembra funzionare)".

Le modalità utilizzate sono sostanzialmente tre: Flow Chart (o diagramma di flusso), Pseudo codifica, Diagramma gerarchico.

3.1 Rappresentazione mediante Flow Chart

L'ideazione dei diagrammi di flusso avviene negli anni venti per descrivere dei processi industriali. Con l'avvento dell'informatica questo tipo di rappresentazioni vennero usate per **descrivere il "flusso" delle istruzioni** del programma.

I primi linguaggi di programmazione erano linguaggi macchina, sequenze di numeri che rappresentavano operazioni, operandi e condizioni di salto all'interno della memoria.

Il puntatore delle istruzioni poteva essere spostato in qualsiasi altra locazione di memoria producendo a volte dei veri e propri labirinti nel codice. I flow chart si adattarono benissimo a questo tipo di approccio e sono stati usati per molto tempo e proficuamente in tutti quei linguaggi come l'Assembler e il Basic in cui il controllo del programma avviene per mezzo di salti (jump) diretti nella memoria. In questi linguaggi esistono quasi esclusivamente operazioni "elementari" e non ci sono metodi per raggruppare logicamente più istruzioni.

Con l'avvento della programmazione strutturata i Flow Chart sarebbero dovuti diventare presto obsoleti per essere sostituiti a pieno titolo dai "Diagrammi di struttura" ma... non è andata così! I flow chart sono tutt'ora vivi e vegeti e si trovano in quasi tutti i libri di testo scolastici di Informatica.

3.2 Rappresentazione mediante Pseudo codifica

Si tratta di **descrivere il procedimento in modo testuale** utilizzando delle parole chiave molto simili a quelle di un linguaggio di programmazione reale. Non esiste una sola pseudo codifica ma molte... La parola pseudo rappresenta questo grado di libertà. Alcune parti del programma possono essere anche scritte in modo informale se si tratta di procedimenti noti o già risolti.

3.3 Rappresentazione mediante Diagrammi di struttura

Con l'avvento della programmazione strutturata, per l'approccio alla soluzione dei problemi si è introdotta la metodologia top-down, dal generale al particolare. Si tratta di un procedimento che parte dall'obiettivo principale e

passa a scomporlo in parti sempre più piccole rappresentando questa **scomposizione con un albero rovesciato** con la “radice” verso l'alto e i nodi terminali, le foglie, verso il basso.

Si possono utilizzare i diagrammi di struttura non solo per analizzare un problema ma anche per rappresentare un programma: la radice è costituita dal programma; a ogni livello di raggruppamento si scende di un livello nell'albero.

Si tratta di un metodo tuttora valido e forse più efficace degli altri due ma non ha preso piede essendo oggi pressoché inutilizzato.

3.4 Quale modello scegliere

Quello della scelta del modello di rappresentazione è una questione per molti controversa... alcuni testi e alcuni insegnanti propongono il diagramma di flusso, altri la pseudo codifica, altri... nessuno. Incuriosisce come a volte il dibattito trovi posizioni tanto nette e contrastanti.

Per quello che è il sentire più comune il diagramma di flusso sembra essere più facilmente comprensibile dal maggior numero di alunni, anche in situazioni problematiche.

Per contro la pseudo codifica è sì più difficoltosa ma più rigorosa e dà meno adito ad errori.

3.5 Un approccio unitario

I tre modelli citati hanno lo stesso scopo: rappresentare il programma in modo grafico o testuale più generico rispetto al linguaggio di implementazione. Se ogni modello deve corrispondere al programma finale è transitivamente stabilito che **sono equivalenti tra di loro**.

Per quanto riguarda la pseudo codifica la corrispondenza con il programma è pressoché diretta. Lo stesso dicasi per il diagramma di struttura dal momento che è stato sviluppato assieme ai linguaggi strutturati proprio per schematizzarne i programmi ed è in stretta analogia con essi. Il problema maggiore è rappresentato dal diagramma di flusso. L'incongruenza deriva dal fatto che le istruzioni nei linguaggi strutturati hanno una loro atomicità. Le istruzioni composte come ad esempio IF “contengono” anche le istruzioni che esse controllano; non ha senso considerare IF senza il suo contenuto.

Il diagramma di flusso invece nella sua forma originale rappresenta elementi separati che devono essere uniti da frecce. Il blocco di decisione (che serve a implementare tanto la selezione che i cicli) non “contiene” nessun altro blocco; l'azione intrapresa è il passaggio in un altro punto del programma che nei primi linguaggi imperativi veniva svolto dall'istruzione GOTO del BASIC o JUMP in Assembler. Con l'avvento dei linguaggi strutturati la numerazione delle istruzioni non serve e i salti non si usano più. L'uso dei flow chart tradizionali porta quindi a una difficoltà: utilizzando i “blocchi” e le “frecce”, con le regole sintattiche previste dai diagrammi, è possibile costruire dei “programmi” che non si possono tradurre in un linguaggio strutturato. Un grafico come quello in Fig. 1 non è realizzabile con i costrutti tipici di un linguaggio moderno in quanto nella “freccia di ritorno” dell'istruzione DO – WHILE non si possono inserire altre istruzioni.

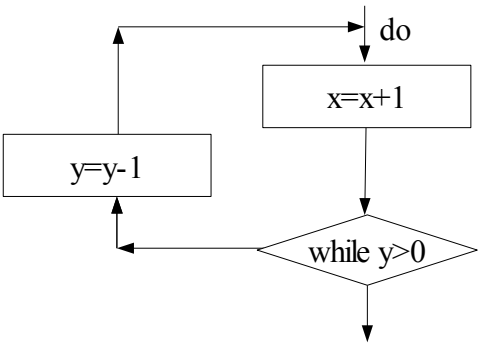


Fig. 1: Diagramma di flusso errato per ciclo do-while

3.6 Sintassi dei linguaggi e Flow Chart strutturati

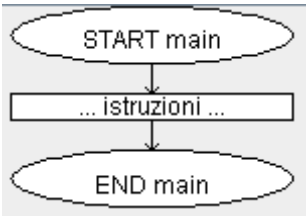
I **linguaggi strutturati** sono definiti in base a **regole grammaticali ricorsive** che permettono di generare insiemi infiniti di espressioni. Queste regole vengono scritte secondo dei formalismi di cui il più conosciuto è BNF (Backus Normal Form o Backus Naur Form) introdotto per definire la sintassi di Algol 60 [Mitchell 2003]. Un modo alternativo usato spesso per descrivere le regole grammaticali è per mezzo delle carte sintattiche riportate in molti manuali come [Bowles 1982].

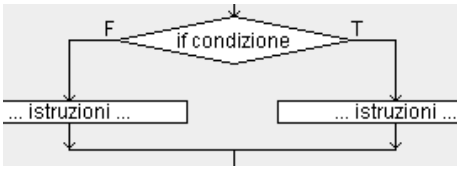
In termini "pratici" possiamo dire che ogni linguaggio imperativo moderno prevede la possibilità che ad ogni istruzione venga sostituito un blocco di istruzioni. Nel Pascal ciò avviene racchiudendo il gruppo di istruzioni tra BEGIN e END, in C e derivati (C++, Java) si racchiude il blocco tra parentesi graffe. Ciò permette di includere istruzioni dentro istruzioni come un gioco di scatole cinesi.

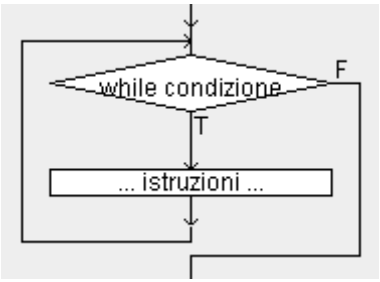
Nel Flow Chart propriamente detto questo approccio non è possibile: è necessario introdurre dei criteri di costruzione dei diagrammi di flusso che tengano conto della natura ricorsiva della sintassi dei linguaggi di programmazione attuali. Possiamo definirlo **Diagramma di Flusso Strutturato**.

Ottenere ciò è possibile utilizzando dei blocchi di codice "precostituiti" in cui le istruzioni composte (if/while/do-while) hanno un unico punto di ingresso e uscita e possono contenere al loro interno le istruzioni controllate. La **composizione avviene per inclusione** e non c'è possibilità di inserire frecce o altri elementi dove capita. È possibile una equivalenza diretta tra codice (o pseudo codice) e diagramma di flusso "strutturato" come esemplificato in Tab.1 in cui sono mostrate alcune istruzioni in pseudo codifica e la rappresentazione corrispondente.

La regola di composizione è semplice: al posto della scritta o del blocco "...istruzioni..." è possibile inserire un qualsiasi tipo e numero di istruzioni.

Pseudo codifica	Blocco equivalente
programma principale	
PROGRAM main ...istruzioni... END main	
selezione a due vie	

IF condizione
...istruzioni...
ELSE
...istruzioni...
END IF
	

ciclo precondizionale	
WHILE condizione ...istruzioni... END WHILE	

Tab. 1 - Corrispondenza tra alcune istruzioni e blocchi utilizzati nel flow chart strutturato

3.7 Didattica e strumenti per la progettazione degli algoritmi

Utilizzando il flow chart strutturato mettiamo d'accordo linguaggi standard, pseudo codifica e rappresentazione grafica: questi altro non sono che interpretazioni di un modello astratto unico in cui si risolve la questione sostanziale.

Oggi molti testi di informatica, anche a livello introduttivo, riportano correttamente sia flow chart che pseudo codice. Tra i docenti alcuni adottano più o meno a lungo un modello progettuale, altri cominciano direttamente con il linguaggio di implementazione pensando che sia tutta una perdita di tempo. Gli studenti normalmente sposano questo secondo punto di vista pensando che la cosa migliore sia sedersi al terminale e “buttare giù” del codice. In realtà **l'uso di formalismi di progetto è un investimento e garanzia di maggiore qualità** del software. A livello professionale l'argomento è fuori discussione; nessun

progetto di buon livello può prescindere da una progettazione dettagliata in tutte le sue fasi. Ci sono ambiti in cui tutti concordano sulla utilità dei linguaggi di progetto. Si pensi ad UML delle classi e ai diagrammi E/R nell'implementazione dei data base.

Forse l'atteggiamento refrattario si spiega con una sorta di pigrizia, il desiderio di evitare “del lavoro in più”; o forse **gli strumenti che si hanno a disposizione non aiutano**.

Mentre la formulazione del problema è puro testo e la scrittura del programma definitivo si avvale di linguaggi e ambienti di programmazione standard di tipo commerciale o scientifico, perfettamente adatti allo scopo, **la fase di modellazione dell'algoritmo rimane ancora poco supportata**.

Qualunque sia il formalismo descrittivo utilizzato, gli strumenti di lavoro finora utilizzati sono quasi sempre gli stessi: lavagna+gesso, carta e penna o, al più, un programma di video scrittura o di grafica vettoriale. L'uso di questi software al posto della carta e della penna ha il chiaro vantaggio di rendere più semplice la memorizzazione e la successiva modifica dei documenti ma non fornisce nessuna funzionalità davvero innovativa rispetto ai metodi “vintage”. Spesso anzi l'uso di queste suite tipo “office” si traduce solo in una ulteriore perdita di tempo, sottratto alla vera soluzione del problema. A tutt'oggi, in pieno sviluppo delle Tecnologie dell'Informazione e Comunicazione applicate alla didattica di quali nuovi strumenti ci siamo dotati per migliorare lo studio e l'apprendimento della programmazione? È giusto che gli informatici non abbiano strumenti informatici adeguati allo studio e della materia? Sembra una contraddizione non da poco.

4 AlgoBuild ambiente didattico per la Progettazione di Programmi

Dall'esperienza quotidiana di lavoro, in classe e in laboratorio, si sono palesati alcuni bisogni:

- necessità di unificazione degli strumenti di modellazione;
- esigenza di interoperabilità tra i vari modelli;
- bisogno di un ambiente più funzionale e moderno che riduca piuttosto che aumentare le operazioni manuali.

Da tutto ciò è nato l'idea che ha dato origine a un programma applicativo dedicato: AlgoBuild.

4.1 Identificazione dei requisiti del software

Nella progettazione del programma si sono tenute in considerazione le seguenti esigenze:

- requisiti utente
  - l'utente deve poter utilizzare elementi (blocchi) logici predefiniti e non deve essere obbligato a preoccuparsi dell'impostazione grafica;
  - non deve essere possibile costruire diagrammi graficamente errati;
  - l'operatività deve essere più semplice e veloce possibile, le azioni devono essere ridotte al minimo necessario;

- deve essere possibile facilmente inserire, copiare e riutilizzare frammenti di codice;

- requisiti hardware/software
  - il software deve essere portabile tra sistemi operativi diversi;
  - le esigenze di memoria e CPU debbono essere compatibili con dei PC di fascia medio/bassa;
  - deve essere possibile utilizzare il programma anche in ambienti per la didattica multimediale utilizzando ad esempio la Lavagna Interattiva Multimediale (LIM).

4.2 Sviluppo del programma e prime sperimentazioni

All'inizio del 2011 è iniziato lo sviluppo del software, distribuito in forma di prototipo a fine febbraio.

Attualmente non è stata ancora rilasciata la prima versione definitiva e il programma è oggetto di sperimentazione informale in alcune classi di due Istituti della provincia di Padova.

A parte alcuni problemi, immancabili nei primi rilasci di un nuovo software, il prodotto ha incontrato pareri favorevoli da parte di docenti e studenti.

4.3 Caratteristiche del programma AlgoBuild

Nella Fig. 2 è possibile vedere un schermata del programma AlgoBuild con un diagramma di flusso di esempio e il riquadro con la pseudo codifica a destra. In basso si può notare un frammento dell'output prodotto dall'esecuzione del programma.

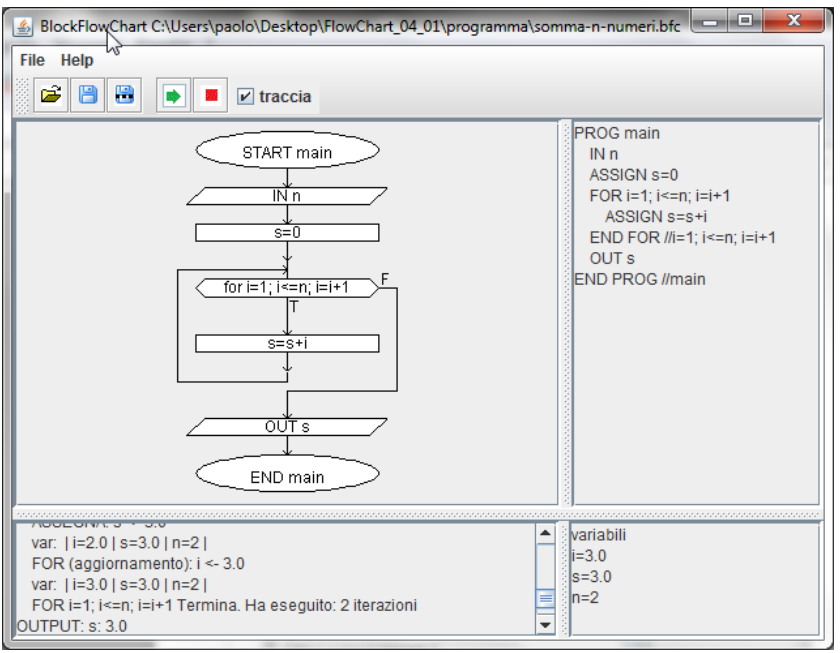


Fig. 2: Finestra principale del programma

Le caratteristiche del programma corrispondono in modo adeguato ai requisiti utente e hardware/software definiti.

In più è stata aggiunta una funzionalità non prevista cioè la possibilità di eseguire il test del programma utilizzando una sintassi semplificata stile C – Java. La pressione di un bottone fa avanzare l'esecuzione passo-passo del programma evidenziando le istruzioni da eseguire, leggendo input, producendo output, visualizzando il valore delle variabili e la traccia di esecuzione delle istruzioni.

4.4 Esempio d'uso

Esempio di realizzazione di un programma: calcolo del fattoriale.

Si legge un numero x che deve essere positivo; in una variabile f si memorizzano i prodotti di tutti i numeri da 1 a x; si visualizza il risultato f. La realizzazione pratica è illustrata in forma di sequenza in Fig. 3.

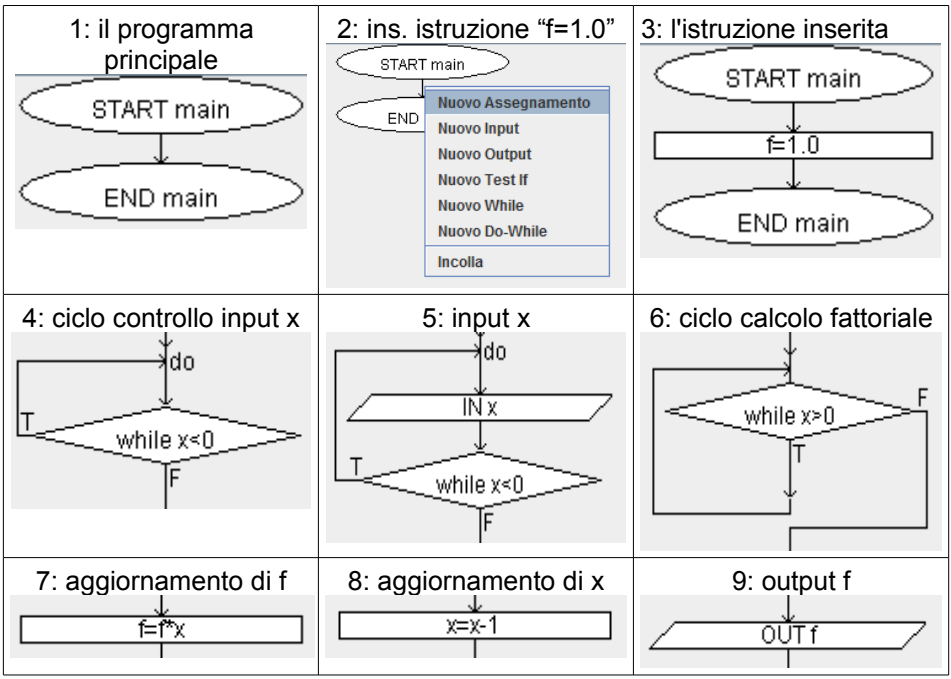


Fig. 3 - Sequenza per la realizzazione del programma per il calcolo del fattoriale

Ogni operazione di inserimento necessita di 2 “clic” più l'inserimento del testo corrispondente. Eventuali aggiunte, cancellazioni, copiature sono altrettanto facili e rapide.

Il diagramma e il pseudo codice risultanti sono riportati in Fig. 4.

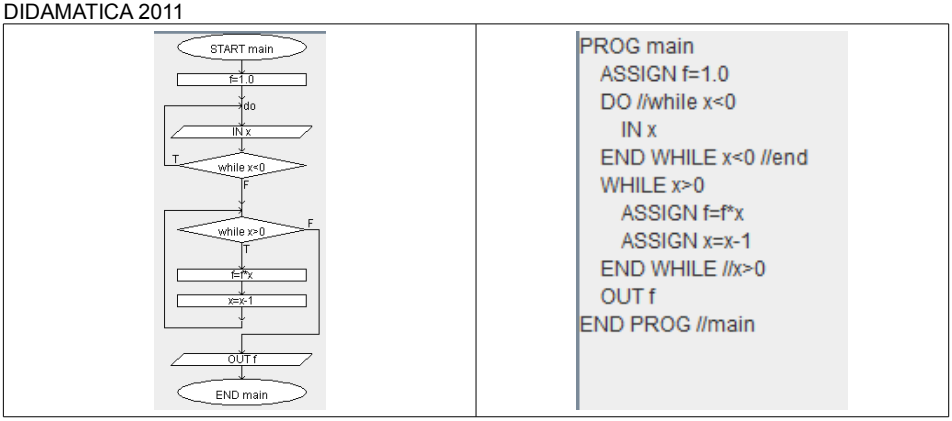


Fig. 4 – Diagramma e pseudo codice del programma per il calcolo del fattoriale

Una volta terminata la composizione del programma è possibile eseguirlo passo-passo con o senza tracciamento delle istruzioni per mezzo di opportuni controlli visibili in Fig. 5.

Fig. 5: I controlli per l'esecuzione del programma

Per approfondimenti, esempi e reperimento del software AlgoBuild si deve fare riferimento al sito dedicato www.algobuild.com.

5 Conclusioni e sviluppi futuri

Da quelle che sono le prime impressioni degli utenti che hanno sperimentato l'ambiente, l'impostazione generale e l'operatività sono state accolte in modo favorevole. Poter pensare solo alla logica del programma senza dover preoccuparsi dell'aspetto grafico toglie un sovraccarico di lavoro non necessario. La struttura dei diagrammi è unificata e quindi più leggibile.

È stato apprezzato anche il fatto di poter utilizzare efficacemente il programma con la lavagna interattiva multimediale (LIM).

Perché il prodotto diventi uno strumento "professionale" in tutti i sensi serve ancora tempo, impegno e lavoro. Gli sviluppi futuri dipendono in gran parte dal gradimento, i consigli e le richieste degli utenti. Se i riscontri continueranno ad essere positivi il programma sarà ulteriormente sviluppato e verranno implementate altre funzionalità che per ora non sono state considerate.

6 Bibliografia

[Bowles 1982] Bowles K. L., Soluzione di problemi con Pascal, Gruppo Editoriale Jackson, Milano 1982.

[Mitchell 2003] Mitchell J. C., Concepts in programming languages, Cambridge University Press, Cambridge UK, 2003.