Gli invarianti per riflettere sull'iterazione nella scuola secondaria: un'esperienza sul campo

Emanuele Scapin

Istituto Tecnico Tecnologico G.Chilesotti, Thiene, VI
emanuele.scapin@istruzione.it

Abstract. Loop invariants are helpful problem-solving tools that can provide guidelines for algorithm design and program development. They are often presented in university-level introductory courses, but are usually considered too difficult to be addressed in the high school. A major obstacle, in this respect, is the complexity of the mathematical notation commonly used to formalize invariants. In this paper we discuss some practical experiences where high-school students have been working with loop invariants represented less formally, in a pictorial language. In particular, we will consider some of the students' proposals to visualize invariants and the difficulties encountered.

Keywords: didactics, loop, invariant, teaching

1 Introduzione

L'esperianza nell'insegnamento dell'informatica nelle scuole superiori ha portato a considerare la difficoltà di parecchi studenti nell'utilizzare le condizioni delle istruzioni iterative, dette per semplicità cicli. La difficoltà principale degli studenti risulta essere quella di definire la condizione di terminazione soprattutto per l'istruzione iterativa While, che risulta a volte non agevole. In letteratura ci sono parecchi lavori sull'argomento, molti di questi lavori individuano il concetto di *Invariante* come utile ad individuare la condizione di terminazione di un ciclo: in particolare vanno presi in considerazione quelli di D. Ginat [7,8]. L'esperienza di un docente di informatica in merito agl'invarianti di solito si limita agli studi universitari, dove però l'approccio è formale, si veda ad esempio il testo di V. Ambriola e P. Ciancarini [1], ma difficilmente un approccio formale agl'invarianti può essere utile in un ambito scolastico, anche se questo è presso una scuola superiore; altri autori, come ad esempio W.C. Tam [20] oppure O. Astrachan [2], hanno proposto quindi un utilizzo semplificato degl'invarianti (che forse potrebbere essere più facile da utilizzare in ambito scolastico). Questo articolo descrive l'esperienza fatta con due classi a indirizzo informatico di un istituto tecnico tecnologico, le classi erano una terza, quindi al primo anno di specializzazione, e una quinta, al terzo e ultimo anno di specializzazione.

2 Metodo formale

L'introduzione ai fondamenti di un metodo di progettazione razionale degli algoritmi è tipicamente presentato nei corsi universitari: metodi per la progettazione razionale sono stati introdotti da R. Floyd, E.W. Dijkstra [6] e C.A.R. Hoare [14], e integrano e completano il concetto della programmazione strutturata. Un metodo formale di progettazione razionale di programmi iterativi è stato presentato da Montangero e Turini [18] e poi ripreso da Ambriola e Ciancarini [1] in cui un algoritmo A è visto come una funzione di trasformazione di stati $f : S \to S$. Si dice che l'algoritmo A è corretto se e solo se realizza la funzione di trasformazione f, ovvero per ogni elemento x del dominio degli stati di ingresso l'algoritmo termina e calcola effettivamente f(x), elemento del codominio di f. La progettazione ha inizio con la definizione di una *asserzione finale* AF che descrive una proprietà del codominio dell'algoritmo, questo predicato servirà a guidare la definizione del corpo del programma che implementerà l'algoritmo, che consisterà in un ciclo di cui occorrerà individuare l'asserzione *invariante* e quindi il predicato di *guardia* (condizione del ciclo). La progettazione del programma che implementa l'algoritmo con iterazione (ciclo) prevede la definizione degli assegnamenti iniziali delle variabili presenti e della funzione di terminazione, viene conclusa dalla dimostrazione di verifica delle ipotesi del teorema di iterazione, che dimostra la correttezza.
Vale la pena di ricordare l'enunciato del teorema di iterazione finita.

Teorema di iterazione finita: se

1. $\mathrm{INV} \wedge \mathrm{G} \implies \mathrm{wp}[\![\mathrm{C}]\!]\ \mathrm{INV} \wedge \mathrm{Def(G)}$
2. $\forall \mathrm{m} \in \{\mathrm{INV} \wedge \mathrm{G}\} : \mathrm{t(m)}{>}0 \wedge \mathrm{t}(\mathbb{C}[\![\mathrm{C}]\!]_{\mathrm{r\ m}}) < \mathrm{t(m)}$

allora $\mathrm{INV} \wedge \mathrm{Def(G)} \implies \mathrm{wp}[\![\text{while G to C}]\!]\ \mathrm{INV} \wedge \text{not G}$

dove C è un comando (istruzione o sequenza di istruzioni da eseguire), G è un'espressione booleana, m è uno stato di memoria, r un ambiente; $\mathbb{C}[\![\mathrm{C}]\!]_{\mathrm{r\ m}}$ è la semantica del comando C nello stato di memoria m e nell'ambiente; t(m) è la *funzione di terminazione*, ed è sempre positiva (non deve diventare nulla al termine del ciclo, un'unica condizione è che sia monotona decrescente).

Il metodo formale tende a derivare la stesura del programma partendo dall'identificazione del predicato, (obiettivo del programma), cioè dall'asserzione finale AF, che ne descrive gli stati finali. La padronanza di tale metodo non può che avvenire per tentativi ed errori, ovvero euristicamente.

2.1 Un semplice esempio: Esponenziale

Presentiamo un semplice esempio di applicazione del metodo proposto, il caso dell'elevamento a potenza, dati due valori B>0 ed $E \geq 0$ calcolare il valore di B^E. Questo è un classico problema proposto agli studenti delle scuole superiori nel primo anno di studio dell'informatica e della programmazione. La definizione

dell'algoritmo, e quindi dell'obiettivo del programma, può essere presentato con la seguente asserzione finale $AF \equiv (Z = B^E)$.
L'obiettivo però sarebbe presentare questa asserzione in una forma che possa identificare la guardia, ovvero la condizione del ciclo, e l'invariante. Se pensiamo che l'algoritmo consiste nell'enumerare le potenze di B incrementando il valore di una variabile y fino ad ottenere un valore uguale a E, possiamo trasformare l'asserzione in questa espressione equivalente: $AF \equiv (Z = B^Y \wedge Y = E)$, da cui si può ricavare, indebolendo la congiunzione per l'omissione dell'operando $Y = E$, le seguenti: $INV = (Z = B^Y)$, $GUARDIA = (Y \neq E)$, dove INV indica l'invariante e GUARDIA la condizione dell'istruzione iterativa, ovvero il ciclo. Nel corpo del ciclo while si deve includere l'assegnamento y = y + 1; al fine di poter incrementare la variabile y in modo tale che possa raggiungere il valore Y=E definito nell'asserzione finale AF. Allo stesso modo, per mantenere verificato l'invariante, nel corpo del ciclo va aggiunto l'assegnamento z = z * b; per un calcolo progressivo delle potenze di B fino alla terminazione con il calcolo della potenza desiderata. La funzione di terminazione sarà quindi Y=E ovvero E-Y=0.

Il programma completo sarà quindi nella forma

```
void main()
{
        int b, e, z, y;
        scanf("%d", &b);
        scanf("%d", &e);
        z = 1;
        y = 0;

        while /*INV*/ (y != E) /*terminazione: E-Y*/{
                y = y + 1;
                z = z * b;
        }
        /*AF*/
        printf("%d", z);
}
```

Si può quindi dimostrare la correttezza della proposta fatta rispetto al teorema di iterazione finita che durante questa trattazione non verrà presentato.
Questo metodo formale però, per la oggettiva difficoltà, non può essere utilizzato in una scuola superiore, nasce quindi la necessità di individuare delle strategie più semplici.

3 Metodo semplificato

L'uso degli invarianti come strategia per spiegare come definire un ciclo e la sua condizione di terminazione è stato studiato da alcuni docenti: W.C. Tam [20] e

D. Ginat [7,8] indicano negli Invarianti una strategia per agevolare lo studente nella stesura di un programma con presenza di cicli; O. Astrachan [2], R.J. Back [3] e L. Mannila [16] suggeriscono invece uno studio con l'utilizzo di immagini, per aiutare lo studente ad individuare delle strategie di risoluzione; A.I. Bočkin [4] ha scelto altre strategie più tradizionali per spiegare il funzionamento dei cicli.
Tale metodo ha lo scopo di individuare alcuni punti chiave per attivare il ragionamento dello studente. Lo studente già conosce ampiamente dallo studio della matematica cosa si intende per elevamento a potenza, per cui il problema sarà di formalizzarlo in una sequenza di passi in modo da poter codificare un programma. Per lo studente l'obiettivo $Z = B^E$ è chiaro, il problema può essere come fare il calcolo con sole moltiplicazioni. Bisogna inanzitutto far capire che l'obiettivo B^E è raggiungibile con una serie di prodotti, infatti $B^E = B \times B \times ... \times B$ dove B si moltiplica a se stesso per E volte. Sì può quindi individuare una parte dell'invariante nella forma di B^Y dove Y è un valore che all'inizio avrà valore 1 e alla fine arriverà a valore E.
Ad ogni ciclo si calcola quindi un valore $Z = B^Y$, che sarà un risultato parziale per ottenere, alla fine delle iterazioni, il valore desiderato, questo dovrebbe indurre lo studente a capire che una istruzione z = z * b all'interno del ciclo sia necessaria, in quanto z ad ogni iterazione è il valore del prodotto ottenuto durante il ciclo precedente. Il valore Y per contare i prodotti verrà incrementato ogni volta di 1, per cui ad ogni iterazione avremo $Y = Y + 1$, questo dovrebbe indurre lo studente a capire che una istruzione y = y + 1 all'interno del ciclo sia necessaria. Naturalmente l'incremento di Y dovrà cessare quando avrà contato E iterazioni, ovvero il prodotto di $Z \times B$ sarà stato eseguito E volte.
L'obiettivo a questo punto è proprio quello di individuare la condizione del ciclo, tenendo conto che lo si raggiunge quando $Y = E$ lo studente può intuire che la serie di cicli continuerà fino al raggiungimento di tale condizione, per cui il valore della condizione o guardia del ciclo può venir individuata da $not(Y = E)$, ovvero $Y \neq E$, per cui l'asserzione finale risulterà $AF = (Z = B^Y \wedge Y = E)$, per ottenere tale risultato si deve quindi individiare una caratteristica del processo di calcolo che non cambia mai, ovvero l'invariante, nella forma $INV = (Z = B^Y \wedge Y \leq E)$ dove la condizione di terminazione, la guardia, del ciclo è individuata da $GUARDIA = (Y \neq E)$, oppure per dare un senso al concetto di incremento di Y dalla $GUARDIA = (Y < E)$, che è analoga.
Il programma che si ottiene è comunque uguale a quello presentato al punto precedente.

3.1 Metodo semplificato con l'utilizzo di immagini

Per far capire meglio l'uso degli invarianti nello studio degli algoritmi con iterazioni O. Astrachan [2] ha proposto un metodo che fa uso di schemi o diagrammi, per facilitare l'apprendimento dello studente.
In questo caso gli stessi concetti visti precedentemente con il metodo formale e poi con quello semplificato possono essere espressi in forma grafica per facilitare

l'apprendimento dello studente.
Si può pensare di rendere in forma grafica il concetto di sequenza di valori B, per rendere l'idea della successione di moltiplicazioni per B, contando il numero di volte - tramite y - che questo avviene.
Nella figura 1.a oltre a indicare la sequenza di valori B, il cui prodotto darà il risultato desiderato, abbiamo il valore y che conta il numero di volte che il prodotto viene effettuato, e che viene incrementato ad ogni iterazione, inoltre è indicato il valore z che all'inizio è 1 e ad ogni iterazione assume il valore $z = z \times B$.

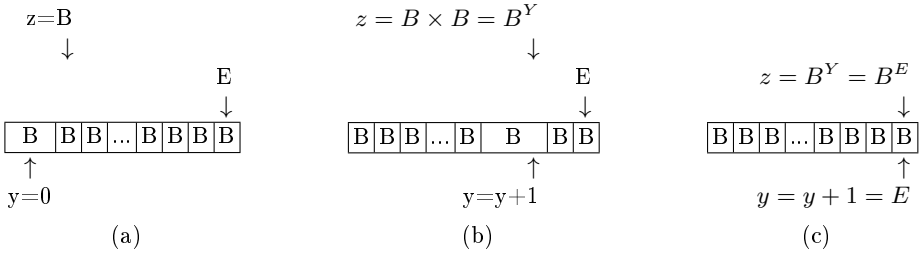

Figura 1

Nella figura 1.b il ciclo esegue ogni volta due operazioni ovvero z=z*B e y=y+1. La sequenza di azioni $z = z \times B$ ripetute rende però più evidente l'obiettivo di $z = B^Y$. Il limite E ci fa capire che questo è il valore che deve raggiungere y, raggiunto il quale, con $y = E$, si ha la terminazione delle iterazioni. Da questi presupposti per lo studente dovrebbe essere più semplice capire che l'asserzione finale sarà del tipo $(Z = B^Y \quad and \quad Y = E)$ si vede pure che l'invariante può essere quindi $(Z = B^Y \quad and \quad Y < E)$, la guardia, ovvero la condizione del ciclo, sarà quindi $(Y \neq E)$, o meglio $(Y < E)$.

4 Un'esperinza pratica

Sono state tentate delle prove pratiche di utilizzo degli invarianti per l'analisi di algoritmi con iterazioni con alcune classi di un istituto tecnico tecnologico ad indirizzo informatica, per vedere anche la differenza di risposta a seconda dell'età e quindi dell'esperianza maturata, lo stesso problema è stato sottoposto ad una classe quinta (quindi terzo e ultimo anno di specializzazione) e a una classe terza (primo anno di specializzazione). In entrambe le classi è stato proposto il problema della somma dei primi N numeri naturali.

4.1 Esperienza con una classe con competenze

Il problema della somma dei primi N numeri naturali è conosciuto e gli studenti sanno algoritmicamente come risolverlo, ma alla richiesta di esprimere i concetti di asserzione finale, invariante e guardia (condizione) dimostrano fin da subito alcune difficoltà. La poca dimestichezza con aspetti formali non permette agli studenti di definire i predicati, per questo si tenta di raggiungere l'obiettivo aiutandosi con degli schemi così come suggerito da O. Astrachan [2].
Presentiamo le due proposte ritenute più significative.

Primo schema

Il primo schema proposto è del tipo

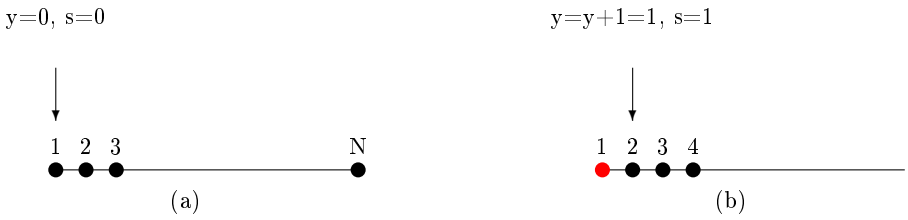

Figura 2

dove abbiamo la sequenza di valori da sommare, ad ogni iterazione si incrementa y, fino alla penultima iterazione, quanto y raggiunge il valore di N-1 (y=N-1),

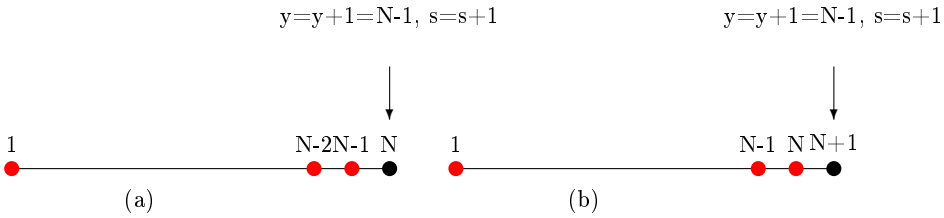

Figura 3

la conclusione si ottiene al passaggio finale (ultima iterazione) dove y=N e s=1+2+3+...+N, che era l'obiettivo. La condizione di terminazione sarà quindi

individuata dal raggiungimento del valore y=N, per cui, durante il processo di iterazioni il valore necessariamente sarà $y \leq N$, la negazione di questo predicato permette di ottenere $y > N$ che sarà una parte dell'asserzione finale, che quindi potrebbe essere $s = 1 + 2 + 3 + ... + N \wedge y > N$, ovvero, tenendo contro dell'incremento della variabile y al fine di individuare il valore da sommare si ottiene $s = s + y \wedge y > N$. L'invariante potrebbe essere quindi del tipo $s = s + y \wedge y \leq N$.

Secondo schema

La seconda proposta emersa fa anch'essa uso di schemi per presentare la successione di numeri da selezionare ad ogni iterazione. L'idea di fondo è che ci sono N numeri naturali da 1 a N, e questi valori vanno selezionati uno alla volta ad ogni iterazione, il valore y permette di contare il numero di cicli e quindi di stabilire a quale ciclo si è al momento attuale, il valore s indicherà la somma ottenuta, all'inizio s=0.

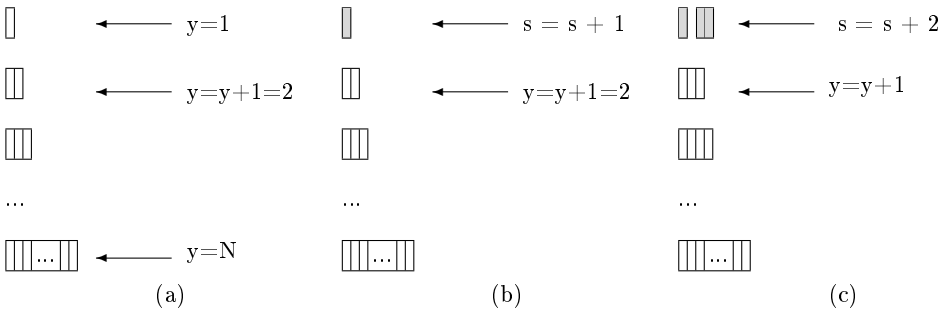

Figura 4

Una volta fatto il primo ciclo il primo valore viene selezionato e sommato. Al passo successivo si seleziona un nuovo valore da sommare alla somma precedentemente calcolata. Alla fine del processo saranno stati eseguiti N cicli, per cui all'ultima iterazione si avrà y = N, e la somma dei numeri da 1 a N, ottenendo quindi s = 1 + 2 + ... + N. Si può quindi capire che ad ogni passo si somma al valore precedentemente ottenuto di s il valore di y , quindi si ha $s = s + y$, dove $y \leq N$. La condizione del ciclo è evidente in quanto si procede fino a che il valore di $y \leq N$.

4.2 Esperienza in una classe senza competenze

I concetti di numero naturale, algoritmo, istruzioni iterative e condizioni di terminazione sono note, mentre il concetto di invariante non è conosciuto dagli stu-

denti. Tra le tre proposte emerse presentiamo le due più semplice da raffigurare con immagini.

Primo schema

La prima proposta fa uso di una rappresentazione simile agli istogrammi.

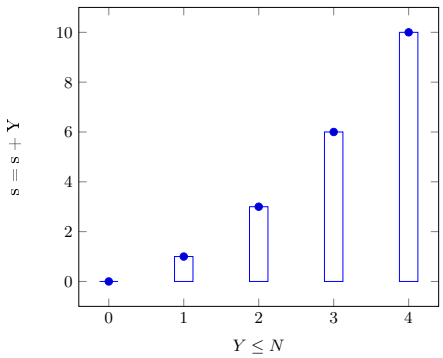

Figura 5

L'idea è quella di esprimere tramite le ascisse i valori crescenti dei numeri naturali da sommare, mentre nelle ordinate si hanno i valori della somma fino a quel momento calcolata. Nelle ascisse abbiamo quindi i valori individuati ad ogni ciclo da $y = y + 1$, che conta quante iterazioni sono state fatte ma soprattutto ci dà il valore da sommare al valore s, che è la somma dei valori fino a quel momento considerati. Lo studente riesce abbastanza agevolmente a capire che il ciclo durerà fino ad avere considerato tutti i primi N valori, ovvero fino a che $Y \leq N$, e quindi riesce a individuare in tale predicato la condizione, o guardia, dell'istruzione iterativa. Se si rappresenta l'asserzione finale nella forma $s = 1 + 2 + 3 + ... + N \wedge y > N \equiv s = 1 + 2 + 3 + ... + N \wedge !(y \leq N)$ lo studente ne riconosce la validità, ma non riesce a fare il successivo passo di astrazione $s = s + Y \wedge Y \leq N$ che dovrebbe portare a riconoscere l'invariante.

Secondo schema

Un secondo schema proposto fa uso di un diagramma a torta. E' evidente che gli studenti hanno maggiore dimestichezza con i grafici.

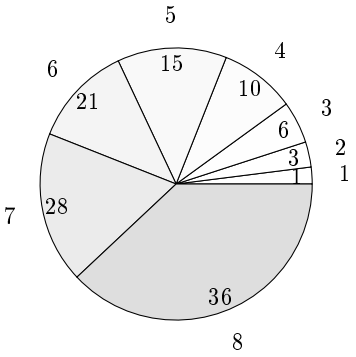

Figura 6

L'idea è di indicare i valori da sommare all'esterno, mentre la somma calcolata nell'area della porzione di cerchio. Questo metodo con grafico a torta mostra subito alcune problematiche:

- non si riesce a esprimere il concetto di incremento del valore da sommare (y) fino al raggiungimento del valore N;
- non si riesce a rendere evidente il valore che individua la somma ottenuta;
- non è chiaro qual è il limite della sequenza delle azioni, ovvero il valore N.

Questa rappresentazione, anche se riesce a esprimere l'idea di somma progressiva dei numeri non chiarisce, non rende evidente, l'obiettivo di rappresentare i concetti di asserzione finale, invariante e guardia (condizione del ciclo), inoltre non permette di astrarre per un numero N qualsiasi.

5 Discussione

L'esperienza fatta in alcune classi di scuola superiore, dove sono stati proposti degli algoritmi che prevedevano istruzioni iterative, ha evidenziato la capacità di solamente pochi studenti di descrivere i concetti cercati, mentre le attività svolte in gruppo e con il coinvolgimento diretto degli studenti, cosi come suggerito da O. Hazzan e altri [12,13], ha dato stimoli e motivazioni a tutti gli allievi di discutere le varie metodologie proposte. Sia gli studenti dell'ultimo anno di specializzazione, classe quinta, che quelli del primo anno di specializzazione, classe terza, sono stati in grado di proporre metodi che tramite schemi riuscissero a rappresentare le azioni svolte dall'algoritmo, ma solamente gli studenti dell'ultimo anno hanno dimostrato capacità logiche maggiori e infatti sono riusciti a produrre una descrizione grafica, tramite schemi e immagini più chiare e dettagliate, che evidenziava bene la progressione delle azioni del ciclo e sono riusciti a individuare con facilità la condizione del ciclo.

Gli studenti del terzo anno hanno dimostrato più difficoltà nel proporre degli schemi significativi di rappresentazione del problema e della sua evoluzione. La prima delle due proposte, pur facendo uso di uno schema che si rifà agli istogrammi, riesce comunque a far capire allo studente le azioni che il ciclo deve eseguire e riesce anche a far capire la condizione di terminazione del ciclo stesso. La seconda proposta invece non riesce a centrare l'obiettivo per vari motivi: la scelta di utilizzare un diagramma a torta non è efficace; il diagramma a torta non riesce a guidare verso un maggiore grado di astrazione; il diagramma a torta non riesce a evidenziare la condizione di terminazione del ciclo; non è possibile generalizzare il valore N che indica l'ultimo valore da sommare. Per quanto riguarda il concetto di invariante sia gli studenti del quinto anno che quelli del terzo anno non sono riusciti a formalizzare l'invariante per il problema proposto, anche se gli studenti del quinto anno hanno dimostrato una maggiore capacità di capirne la natura. Gli studenti del terzo anno hanno dimostrato minori abilità logiche di astrazione, probabilmente dovute al fatto che nei corsi di Matematica delle scuole secondarie di II grado il Calcolo degli Enunciati non viene più esposto con il necessario approfondimento.

Riferimenti bibliografici

1. Vincenzo Ambriola, Paolo Ciancarini *Progettazione razionale di programmi Pascal.* 1989.
2. Owen Astrachan. *Pictures and Invariants.* 1991.
3. Ralph-Johan Back, Johannes Eriksson, and Linda Mannila. *Teaching the Construction of Correct Programs Using Invariant Based Programming.* 2007.
4. A.I. Bočkin. *Методика преподавания информатики: Учеб. пособие.* 1998.
5. S.P. Davies. *Models and theories of programming strategy. International Journal of Man-Machine Studies.* 1993.
6. E.W. Dijkstra. *A Discipline of Programming.* 1976.
7. David Ginat. *Seeking or Skipping Regularities? Novice Tendencies and the Role of Invariants.* 2003.
8. David Ginat. *Loop invariants and mathematical games.* 1995.
9. David Ginat. *Learning from Wrong and Creative Algorithm Design.* 2008.
10. David Ginat. *Efficiency of algorithms for programming beginners.* 1996.
11. D. Gries. *The Science of Programming.* 1981.
12. Orit Hazzan, Tami Lapidot. *The Practicum in Computer Science Education: Bridging Gaps between Theoretical Knowledge and Actual Performance.* 1999.
13. Orit Hazzan, Tami Lapidot, Noa Ragonis. *Guide to Teaching Computer Science.* 2011.
14. C.A.R. Hoare. *An Axiomatic basis for computer programming.* 1969.
15. Fabrizio Luccio. *La struttura degli algoritmi.* 1982.
16. Linda Mannila. *Invariant Based Programming in Education – An Analysis of Student Difficulties.* 2009.
17. R.E. Mayer. *The psychology of how novices learn computer programming.* 1989.
18. Carlo Montangero, Franco Turini. *Introduzione alla programmazione.* 1987.
19. Arnold Pears et Al. *A Survey of Literature on the Teaching of Introductory Programming.* 2007.
20. Wing C. Tam *Teaching Loop Invariants To Beginners By Examples.* 1992.