

UNIVERSITÀ DI URBINO

INFORMATICA APPLICATA

PROGRAMMAZIONE PROCEDURALE E LOGICA

Relazione

PROGETTO PER LA SESSIONE INVERNALE 2014/2015

Studente:

Marco TAMAGNO
matricola no: 261985

Studente:

Francesco BELACCA
matricola no: 260492

Professore:

Marco BERNARDO

January 22, 2015

Contents

1	Specifica del Problema	1
2	Analisi del Problema	2
2.1	Input	2
2.2	Output	2
3	Progettazione dell' Algoritmo	3
3.1	Teoria	3
3.2	Funzioni per l'acquisizione:	5
3.3	Funzioni per la verifica delle proprietà:	5
3.4	Funzioni principali:	6
3.5	Input	7
3.6	Output - Acquisizione	8
3.7	Output - stampa	8
3.8	Output - ordine_parziale	8
3.9	Output - ordine_totale	8
3.10	Output - relazione_equivalenza	9
3.11	Output - check_funzione	9
4	Implementazione dell' algoritmo	10
4.1	Libreria	10
4.2	Test	40
4.3	Makefile	42
5	Testing del programma	43
5.1	Test 1:	43
5.2	Test 2:	44
5.3	Test 3:	45
5.4	Test 4:	46
5.5	Test 5:	47
5.6	Test 6:	48
5.7	Test 7:	49
5.8	Test 8:	50
5.9	Test 9:	51
5.10	Test 10:	52
6	Verica del programma	53

1 Specifica del Problema

Write an ANSI C library that manages binary relations by exporting the following functions. The first C function returns a binary relation introduced through the keyboard. The second C function has a binary relation as input parameter and prints it to the screen. The third C function has a binary relation as input parameter and establishes whether it is a partial order relation, printing to the screen which property does not hold in the case that the relation is not such. The fourth C function has a binary relation as input parameter and establishes whether it is a total order relation, printing to the screen which property does not hold in the case that the relation is not such. The fifth C function has a binary relation as input parameter and establishes whether it is an equivalence relation, printing to the screen which property does not hold in the case that the relation is not such. The sixth C function has a binary relation as input parameter and establishes whether it is a mathematical function; if it is not, then the element violating the property will be printed to the screen, otherwise a message will be printed to the screen indicating whether the function is injective, surjective, or bijective. [The project can be submitted also by first-year students.]

Scrivere una libreria ANSI C che gestisce le relazioni binarie esportando le seguenti funzioni. La prima funzione C restituisce una relazione binaria acquisita da tastiera. La seconda funzione C ha come parametro di ingresso una relazione binaria e la stampa a video. La terza funzione C ha come parametro di ingresso una relazione binaria e stabilisce se essa è una relazione d'ordine parziale, stampando a video quale proprietà non vale nel caso la relazione non sia tale. La quarta funzione C ha come parametro di ingresso una relazione binaria e stabilisce se essa è una relazione d'ordine totale, stampando a video quale proprietà non vale nel caso la relazione non sia tale. La quinta funzione C ha come parametro di ingresso una relazione binaria e stabilisce se essa è una relazione d'equivalenza, stampando a video quale proprietà non vale nel caso la relazione non sia tale. La sesta funzione C ha come parametro di ingresso una relazione binaria e stabilisce se essa è una funzione matematica; se non lo è, allora si stamperà a video quale elemento violi la proprietà, altrimenti si stamperà a video un messaggio che indica se la funzione è iniettiva, suriettiva o biiettiva. [Il progetto può essere consegnato anche da studenti del primo anno.]

2 Analisi del Problema

2.1 Input

1. Per l'acquisizione come input abbiamo una relazione binaria del tipo (a,b) che viene acquisita da tastiera;
2. Come input per le altre 5 funzioni abbiamo una relazione (precedentemente esportata dalla prima).

2.2 Output

1. La prima funzione (Acquisizione) restituisce una funzione binaria acquisita da tastiera;
2. La seconda funzione (Stampa) non restituisce nulla, ma stampa a video la relazione che aveva in ingresso; //
3. La terza funzione "ordine parziale" non restituisce nulla, ma stampa a video se la Relazione binaria acquisita è di ordine parziale o meno e stampa a video le proprietà della funzione, per poter mostrare a schermo quali proprietà non vengono rispettate;
4. La quarta funzione (ordine totale) non restituisce nulla, ma stampa a video se la Relazione binaria acquisita è di ordine totale o meno, stampando a video quale proprietà non vale nel caso la relazione non sia tale;
5. La quinta funzione (relazione equivalenza) non restituisce nulla, ma stampa a video se la relazione binaria acquisita è una relazione di equivalenza o meno e stampa a video le proprietà della funzione, per poter mostrare a schermo quali proprietà non vengono rispettate;
6. la sesta funzione (check funzione) non restituisce nulla, ma stampa a video se la relazione binaria acquisita è una funzione, e in caso contrario stampa a video quale coppia non fa rispettare le proprietà.

3 Progettazione dell' Algoritmo

3.1 Teoria

Per lo sviluppo di questo programma si necessita di alcuni cenni di Teoria degli insiemi quali:

Concetto di Relazione Binaria : In matematica, una relazione binaria definita su di un insieme, anche detta relazione o corrispondenza tra due oggetti, è un elenco di coppie ordinate di elementi appartenenti all'insieme. In modo equivalente, una relazione binaria è un sottoinsieme del prodotto cartesiano di un insieme con se stesso.

Concetto di Relazione d' Ordine Parziale: In matematica, più precisamente in teoria degli ordini, una relazione d'ordine o ordine su di un insieme è una relazione binaria tra elementi appartenenti all'insieme che gode delle seguenti proprietà:

riflessiva

antisimmetrica

transitiva.

Concetto di Relazione d' Ordine Totale: Una relazione d' ordine si dice Totale, quando oltre a essere parziale soddisfa anche la proprietà di Dicotomia (tutti gli elementi devono essere in relazione tra di loro).

Concetto di riflessività : In logica e in matematica, una relazione binaria R in un insieme X è detta riflessiva se ogni elemento di X è in tale relazione con se stesso.

Concetto di transitività: In matematica, una relazione binaria R in un insieme X è transitiva se e solo se per ogni a, b, c appartenenti ad X , se a è in relazione con b e b è in relazione con c , allora a è in relazione con c .

Concetto di simmetricità: In matematica, una relazione binaria R in un insieme X è simmetrica se e solo se, presi due elementi qualsiasi a e b , vale che se a è in relazione con b allora anche b è in relazione con a .

Concetto di funzione: In matematica, una funzione, anche detta applicazione, mappa o trasformazione, è definita dai seguenti oggetti:

Un insieme X detto dominio della funzione. * Un insieme Y detto codominio della funzione. * Una relazione $f : X \rightarrow Y$ che ad ogni elemento dell'insieme X associa uno ed un solo elemento dell'insieme Y ; l'elemento assegnato a x appartenente ad X tramite f viene abitualmente indicato con $f(x)$.

Concetto di Iniettività: Una funzione si dice iniettiva quando a ogni elemento del dominio è assegnato uno e uno solo elemento del codominio.

Concetto di Suriettività: Una funzione si dice suriettiva quando ogni elemento del codominio viene raggiunto da un elemento del dominio.

3.2 Funzioni per l'acquisizione:

acquisizione() : per acquisire la relazione.

3.3 Funzioni per la verifica delle proprietà:

check_iniettivita() : per controllare se l' iniettività è rispettata o meno (0 non c' è, 1 c' è).

check_transitivita() : per controllare se la transitività viene rispettata o meno (0 non c' è, 1 c' è).

check_antisimmetria() : per controllare se l' antisimmetria viene rispettata o meno (0 non c' è, 1 c' è).

check_simmetria() : per controllare se la simmetria viene rispettata o meno (0 non c' è, 1 c' è).

check_riflessivita() : per controllare se la riflessività viene rispettata o meno (0 non c' è, 1 c' è).

check_dicotomia() : per verificare se la dicotomia viene rispettata o meno (0 non c' è, 1 c' è).

check_suriettivita(): verifica se la funzione gode della proprietà di suriettività, in questo caso sarà sempre settata a 1 in quanto tutti gli elementi del codominio (presi come gli elementi dei vari secondi termini digitati durante l' acquisizione) avranno sempre un elemento del dominio associato(dato che non si può acquisire il secondo termine se non se ne acquisisce prima il relativo primo, o arrivare alla funzione check_suriettivita() avendo acquisito solo il primo).

3.4 Funzioni principali:

`ordine_parziale()` : richiama le funzioni delle proprietà e controlla se c' è un ordine parziale(stampa a video se c' è o meno un ordine parziale, e nel caso non c' è stampa quali proprietà non vengono rispettate).

`ordine_totale()`: richiama la funzione `ordine_parziale` e `check_dicotomia` e controlla se c' è un ordine totale(stampa a video se esiste o meno un ordine totale, e nel caso non c' è stampa quali proprietà non vengono rispettate).

`relazione_equivalenza()` : richiama le funzioni delle proprietà e controlla se c' è una relazione d'equivalenza(stampa a video se c' è o meno una relazione d'equivalenza, e nel caso non c' è stampa quali proprietà non vengono rispettate).

`check_funzione()`:verifica se la relazione è una funzione(stampa a video se c' è o non c' è una funzione e nel caso non ci sia dice quale coppia non soddisfa le proprietà).

3.5 Input

Per l' input abbiamo necessità di usare una struttura dati dinamica, nella quale andiamo a salvare la Relazione Binaria dataci dall' utente, il numero delle coppie e il tipo di input (numerico o per stringhe).

L input dovrà essere dotato di diversi controlli, se l' utente sceglie di inserire un input di tipo numerico allora non potrà digitare stringhe e/o caratteri speciali etc.

La scelta di due tipi di input differente dovrà essere data per dare la possibilità all' utente nel caso scelga di fare un' input di tipo numerico di poter effettuare operazioni non legate alle funzioni della libreria, (esempio : l' utente vuole decidere di moltiplicare l' input per due, e vedere se mantiene le proprietà, con un' input di tipo numerico l' utente pu farlo e ci avrebbe un senso, con un' input di tipo stringa meno).

La scelta dell' input di tipo stringa dovrà essere data per aver maggior completezza, una relazione binaria non deve essere forzosamente numerica ma pu essere anche tra cose, oggetti, animali, colori e qualsiasi altra cosa possa venire in mente.

Alle varie funzioni verrà data come input la struttura dati salvata in precedenza dalla funzione Acquisizione, per poterne verificare le varie proprietà.

3.6 Output - Acquisizione

Durante l' acquisizione avremo diversi output video (printf) che guideranno l' utente nell' inserimento dei dati, e che segnaleranno eventuali errori commessi. Finita l' acquisizione dovremo restituire l' indirizzo della struttura, che all' interno quindi conterra' i dati inseriti dall' utente. Abbiamo scelto di fare ci perchè non essendo permesso l' utilizzo di variabili globali, il modo pi semplice di passare i dati inseriti da una funzione all' altra è quello di creare una struttura dinamica. Una volta restituito l' indirizzo della struttura, a seconda della funzione lanciata nel file Test.c si lanceranno le altre 5 funzioni, dato che queste prendono tutte in pasto l' output della prima (cioè l' indirizzo della struttura della relazione binaria) e la utilizzano per verificarne varie proprieta' .

3.7 Output - stampa

La funzione stampa avra' come output la stampa a video della struttura acquisita, con qualche aggiunta grafica(le parentesi e le virgole) per rendere il tutto pi facilmente interpretabile e leggibile.

3.8 Output - ordine_parziale

La funzione ordine_parziale avra' come output la stampa a video del risultato della verifica delle proprieta' di riflessivita' antisimmetria e transitivita' . Nel caso in cui siano tutte verificate si stampera' che la relazione è una relazione di ordine parziale, mentre nel caso in cui non siano verificate si stampera' che non lo è e il perchè (cioè quale proprieta' non è o non sono verificate).

3.9 Output - ordine_totale

La funzione ordine_totale avra' come output la stampa a video del risultato della verifica delle proprieta' necessarie ad avere una relazione d' ordine parziale, e verifichera' poi se anche la dicotomia è valida per la relazione o meno. Nel caso in cui tutto sia positivo, allora si stampera' che la relazione è di ordine totale, mentre se non lo è si stampera' cosa fa in modo che non lo sia.

3.10 Output - relazione_equivalenza

La funzione `relazione_equivalenza` avra' come output la stampa a video del risultato della verifica delle proprieta' di riflessivita' simmetria e transitivita' e nel caso in cui siano tutte positive si stampera' che la relazione è una relazione di equivalenza, mentre nel caso in cui qualcosa non sia verificato si stampera' cio' che impedisce alla relazione di essere una relazione d'equivalenza.

3.11 Output - check_funzione

La funzione `check_funzione` avra' come output la stampa a video della verifica della proprieta' che rende la relazione binaria una funzione, e in caso lo sia anche se questa è suriettiva (che poi spiegheremo essere sempre verificata) e iniettiva, e in caso sia entrambe si stampera' che la relazione binaria oltre ad essere una funzione è una funzione biiettiva.

4 Implementazione dell' algoritmo

4.1 Libreria

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 /******STRUTTURA relBin
6 *****/
7 /****** Creo una struttura dove salvare le coppie
8 ***** appartenenti alla Relazione******/
9
10 typedef struct relBin{
11     /****** Coppia Numerica *****/
12     double *primo_termine ,
13           *secondo_termine ;
14
15     /****** Coppia Qualsiasi*****/
16     char **prima_stringa ,
17          **seconda_stringa ;
18
19     /***** Variabili per salvare se ho acquisito una
20     ***** coppia numerica o no e il numero delle coppie*****
21     */
22     int controllo ,
23       dimensione ;
24 }rel_bin ;
25
26 /*DICHIARO LE FUNZIONI*/
27 rel_bin acquisizione (rel_bin);
28 int check_simmetria (rel_bin);
29 int check_riflessivita (rel_bin);
30 int check_transitivita (rel_bin);
31 int check_suriettivita (rel_bin);
32 void check_biiettivita (rel_bin);
33
34 /******Funzione di acquisizione
35 *****/
36
37 rel_bin acquisizione (rel_bin relazione){
38
39     int acquisizione_finita = 0;
40     int scan = 0;
```

```

36
37
38 relazione.dimensione = 0;
39 relazione.primo_termine = (double *) malloc (2);
40 relazione.secondo_termine = (double *) malloc (2);
41 relazione.prima_stringa = (char **) malloc (100);
42 relazione.seconda_stringa = (char **) malloc (100);
43
44 while (relazione.controllo < 1 || relazione.controllo
      > 2 || scan != 1){
45     fflush (stdin);
46     printf ("\n_Premi_1_se_vuoi_immettere_solo_numeri,_2_
      per_altro\n");
47     printf ("\n_scelta:_");
48     scan = scanf ("%d",&relazione.controllo);
49 }
50
51 /** imposto di nuovo scan a 0 **/
52 scan=0;
53
54 /*Acquisizione Numerica*/
55
56 if (relazione.controllo == 1){
57     while (acquisizione_finita == 0){
58         relazione.dimensione++;
59         acquisizione_finita = 2;
60
61         /*Acquisisco il primo termine della coppia*/
62
63         printf ("\n_Inserisci_il_primo_termine_della_
        coppia_\n");
64         relazione.primo_termine = (double *) realloc (
        relazione.primo_termine, (relazione.dimensione
        +1) * sizeof (double));
65         scan = 0;
66         /*Check del primo termine della coppia*/
67
68         while (scan != 1){
69             printf ("_Primo_Termine:_");
70             fflush (stdin);
71             scan = scanf ("%lf",&relazione.primo_termine[
        relazione.dimensione - 1]);
72         if (scan == 0)

```

```

73     printf ("\nC'e' un errore, reinserire il primo
        termine\n");
74 }
75
76 /*Acquisisco il secondo termine della coppia*/
77     scan = 0;
78     printf ("\nInserisci il secondo termine della
        coppia\n");
79     relazione.secondo_termine = (double *) realloc (
        relazione.secondo_termine, (relazione.
        dimensione+1) * sizeof (double));
80
81 /*Check del secondo termine della coppia*/
82
83     while (scan != 1){
84         printf ("Secondo Termine:");
85         fflush (stdin);
86         scan = scanf ("%lf",&relazione.secondo_termine[
            relazione.dimensione - 1]);
87         if (scan == 0)
88             printf ("\nC'e' un errore, reinserire il secondo
                termine\n");
89     }
90
91 /*Chiedo all'utente se ci sono altre coppie*/
92
93     while (acquisizione_finita < 0 ||
        acquisizione_finita > 1 || scan != 1){
94         printf ("\nVuoi acquisire un'altra coppia?
            immetti 1 per uscire, 0 per continuare\n");
95         printf ("\nscelta:");
96         fflush (stdin);
97         scan = scanf ("%d",&acquisizione_finita);
98     }
99 }
100 }
101
102 /*imposto di nuovo scan a 0*/
103 scan = 0;
104
105 /*Acquisizione con stringhe*/
106 if (relazione.controllo == 2){
107     while (acquisizione_finita == 0){
108         relazione.dimensione++;

```

```

109     acquisizione_finita = 2;
110
111     /* Acquisisco il primo termine della coppia */
112
113     printf ("_Inserisci il primo termine della coppia_\n");
114     printf ("_Primo Termine:_");
115     relazione.prima_stringa[relazione.dimensione - 1]
        = (char *) malloc (50);
116     scan = scanf ("%[^\\n]s", relazione.prima_stringa[
        relazione.dimensione - 1]);
117
118     /* Acquisisco il secondo termine della coppia */
119
120     printf ("_Inserisci il secondo termine della_\n");
121     printf ("_Secondo Termine:_");
122     relazione.seconda_stringa[relazione.dimensione -
        1] = (char *) malloc (50);
123     scan = scanf ("%[^\\n]s", relazione.seconda_stringa
        [relazione.dimensione - 1]);
124
125     /* imposto di nuovo scan a 0 */
126     scan = 0;
127
128     /* Chiedo all'utente se ci sono altre coppie */
129
130     while (acquisizione_finita < 0 ||
        acquisizione_finita > 1 || scan != 1){
131
132         printf ("\nVuoi acquisire un'altra coppia?_\n");
133         scan = scanf ("%d",&acquisizione_finita);
134     }
135 }
136 }
137
138 printf ("\n\n..._Acquisizione Terminata...\n\n");
139 return (relazione);
140 }
141
142 /******FUNZIONE DI STAMPA*****
        *****/
143

```

```

144 void stampa (rel_bin stampa){
145
146     int i = 0;
147
148     printf ("\nLa relazione binaria e'");
149     printf ("\n\n{");
150
151     /******Stampa per coppie numeriche *****/
152
153     if (stampa.controllo == 1){
154         while (i < stampa.dimensione){
155
156             printf ("_(%.2lf,%.2lf)", stampa.primo_termine[
157                 i], stampa.secondo_termine[i]);
158             if (i+1 != stampa.dimensione)
159                 printf ("_");
160             i++;
161         }
162
163     /******Stampa per coppie non numeriche *****/
164
165     if (stampa.controllo == 2){
166         while (i < stampa.dimensione){
167             printf ("_(%s,%s)", stampa.prima_stringa[i],
168                 stampa.seconda_stringa[i]);
169             if (i+1 != stampa.dimensione)
170                 printf (";");
171             i++;
172         }
173     }
174
175     /******Fine Stampa *****/
176
177     printf ("}\n");
178     printf ("\n\n... _Stampa_Terminata...\n\n");
179 }
180
181
182 /******FUNZIONE DI VERIFICA DI RELAZIONI D
'ORDINE*****/
183
184 int ordine_parziale (rel_bin verifica){

```



```

185
186     int riflessivita ,
187         transitivita ,
188         antisimmetria ,
189         parziale ;
190
191     /*STAMPO LE PROPIETA 'DELLA RELAZIONE*/
192
193     printf ("\n\nLa relazione:\n\n");
194
195     /****** Chiamo le funzioni per poter stabilire le
196                propriet ******/
197
198     riflessivita = check_riflessivita (verifica);
199     antisimmetria = check_antisimmetria (verifica);
200     transitivita = check_transitivita (verifica);
201
202     /****** Controllo se rispetta le propriet per
203                essere una relazione d'ordine parziale******/
204
205     if (transitivita == 1 && antisimmetria == 1 &&
206         riflessivita == 1){
207         parziale = 1;
208         printf ("\n\nQuindi e' una relazione d'ordine
209                parziale\n\n");
210     }
211     else{
212
213         printf ("\n\nNon e' una relazione d'ordine parziale
214                in quanto non rispetta tutte le propieta '\n");
215         parziale = 0;
216     }
217
218     if (transitivita == 0)
219         printf ("\n\nmanca la propieta 'di transitivita '\n");
220     ;
221
222     if (antisimmetria == 0)
223         printf ("\n\nmanca la propieta 'di antisimmetria\n");
224     ;
225
226     if (riflessivita == 0)
227         printf ("\n\nmanca la propieta 'di riflessivita '\n");
228     ;
229
230     /****** Fine controllo Ordine Parziale
231                ******/
232
233

```

```

220     printf ("\n\n... \Controllo \Ordine \Parziale \
        Terminato...\n\n\n");
221     return (parziale);
222 }
223
224
225 /******FUNZIONE PER CONTROLLARE LA RIFLESSIVIT
        *****/
226
227 int check_riflessivita (rel_bin verifica){
228
229     int i,
230         j,
231         k,
232         riscontro ,
233         secondo_riscontro ,
234         riflessivita;
235
236     riflessivita = 1;
237     i = 0;
238     j = 0;
239     k = 0;
240     riscontro = 0;
241     secondo_riscontro = 0;
242
243 /* Verifica riflessivit */
244
245 /*Definizione: una relazione per la quale esiste
        almeno un elemento che non e' in relazione con s
        stesso non soddisfa la definizione di riflessivit
        */
246
247     while ( (i < verifica.dimensione) && (k < verifica.
        dimensione)){
248
249 /* Verifica riflessivit per numeri*/
250
251         if (verifica.controllo == 1){
252             riscontro = 0;
253             secondo_riscontro = 0;
254             if (verifica.primo_termine[i] == verifica.
                secondo_termine[i])
255                 riscontro++; /**** Controllo se c' stato un
                    riscontro a,a****/

```

```

256         secondo_riscontro++;
257         if (riscontro != 0){
258             i++;
259             k++;
260         }
261         /**/
262         else{
263             j=0;
264             riscontro = 0;
265             secondo_riscontro = 0;
266
267         /****** Controllo la riflessivit per gli
                elementi del primo insieme
                *****/
268
269         while (j < verifica.dimensione){
270             if (j == i)
271                 j++;
272             else{
273                 if (verifica.primo_termine[i] == verifica.
                    primo_termine[j])
274                 if (verifica.primo_termine[j] ==
                    verifica.secondo_termine[j])
275                     riscontro++;
276
277                 j++;
278             }
279         }
280
281         j = 0;
282
283         /****** Controllo la riflessivit per gli
                elementi del secondo insieme
                *****/
284
285         while (j < verifica.dimensione){
286             if (j == k)
287                 j++;
288             else{
289                 if (verifica.secondo_termine[k] ==
                    verifica.secondo_termine[j])
290                 if (verifica.primo_termine[j] ==
                    verifica.secondo_termine[j])
291                     secondo_riscontro++;

```

```

292
293         j++;
294     }
295 }
296     if (riscontro != 0)
297         i++;
298
299 /***** Se non c'è stato un riscontro di riflessivita
    esco e imposto la riflessivita a 0 *****/
300
301     else{
302         i=verifica.dimensione;
303         riflessivita = 0;
304     }
305
306     if (secondo_riscontro != 0)
307         k++;
308
309     else{
310         k=verifica.dimensione;
311         riflessivita = 0;
312     }
313 }
314
315 }
316
317 /****** VERIFICA RIFLESSIVITA PER STRINGHE
    ******/
318
319 if (verifica.controllo == 2){
320     riscontro = 0;
321     secondo_riscontro = 0;
322     if (strcmp (verifica.prima_stringa[i], verifica.
        seconda_stringa[i]) == 0)
323         riscontro++;
324     secondo_riscontro++;
325     if (riscontro != 0){
326         i++;
327         k++;
328     }
329
330     else{
331         j=0;
332         riscontro = 0;

```

```

333         secondo_riscontro = 0;
334
335  /****** Controllo la riflessivit per gli
elementi del primo insieme
******/
336
337     while (j < verifica.dimensione){
338         if (j == i)
339             j++;
340         else{
341             if (strcmp (verifica.prima_stringa[i],
342                        verifica.prima_stringa[j]) == 0)
343                 if (strcmp (verifica.prima_stringa[j],
344                            verifica.seconda_stringa[j]) == 0)
345                     riscontro++;
346             j++;
347         }
348
349         j = 0;
350
351  /****** Controllo la riflessivit per gli
elementi del secondo insieme
******/
352
353     while (j < verifica.dimensione){
354         if (j == k)
355             j++;
356         else{
357             if (strcmp (verifica.seconda_stringa[k],
358                        verifica.seconda_stringa[j]) == 0)
359                 if (strcmp (verifica.prima_stringa[j],
360                            verifica.seconda_stringa[j]) == 0)
361                     secondo_riscontro++;
362             j++;
363         }
364         if (riscontro != 0)
365             i++;
366
367     else{
368         i=verifica.dimensione;

```

```

369         riflessivita = 0;
370     }
371
372     if (secondo_riscontro != 0)
373         k++;
374
375     else{
376         k=verifica.dimensione;
377         riflessivita = 0;
378     }
379 }
380
381 }
382
383 }
384
385 /****** Controllo se riflessiva
******/
386
387     if (riflessivita == 1)
388         printf ("L' e' riflessiva\n");
389     else
390         printf ("L non e' riflessiva\n");
391
392 /****** Fine riflessivita *****
*/
393
394     return (riflessivita);
395 }
396
397
398
399 /****** FUNZIONE PER CONTROLLARE
LA SIMMETRIA *****
400
401 /****** Definizione: In matematica, una
relazione binaria R in un insieme X **/
402 /****** simmetrica se e solo se, presi due
elementi qualsiasi a e b, vale che **/
403 /****** se a in relazione con b allora anche
b in relazione con a. *****
404
405 int check_simmetria (rel_bin verifica){
406

```

```

407     int i,
408         j,
409         riscontro,
410         simmetria;
411
412     simmetria = 1;
413
414
415     i = 0;
416     j = 0;
417     riscontro = 0;
418
419     /* Check della simmetria per numeri */
420
421     if (verifica.controllo == 1){
422
423         while ( i < verifica.dimensione){
424
425             j = 0;
426             while ( j < verifica.dimensione){
427
428                 if (verifica.primo_termine[i] == verifica.
429                     secondo_termine[j])
430                     if (verifica.primo_termine[j] == verifica.
431                         secondo_termine[i])
432                         riscontro++;
433                 j++;
434             }
435
436             if (riscontro == 0){
437                 j = verifica.dimensione;
438                 i = verifica.dimensione;
439                 simmetria = 0;
440             }
441             riscontro = 0;
442             i++;
443         }
444
445     }
446
447     /* Check della simmetria per stringhe */
448
449     if (verifica.controllo == 2){

```

```

449     while ( i < verifica.dimensione){
450
451         j = 0;
452         while ( j < verifica.dimensione){
453
454             if (strcmp (verifica.prima_stringa[i],verifica
                     .seconda_stringa[j]) == 0 )
455                 if (strcmp (verifica.prima_stringa[j],
                     verifica.seconda_stringa[i]) == 0 )
456                     riscontro++;
457
458             j++;
459         }
460
461         if (riscontro == 0){
462             j = verifica.dimensione;
463             i = verifica.dimensione;
464             simmetria = 0;
465         }
466         riscontro = 0;
467         i++;
468     }
469 }
470 }
471
472 /****** Controllo se la simmetria stata verificata
******/
473
474     if (simmetria == 1)
475         printf ("L'insieme 'simmetrica\n");
476     else
477         printf ("L'insieme 'asimmetrica\n");
478
479 /****** Fine controllo simmetria ******/
480
481     return (simmetria);
482 }
483
484
485
486 /* FUNZIONE PER CONTROLLARE LA TRANSITIVITA' */
487
488 /****** Definizione: In matematica, una relazione
binaria R in un insieme X transitiva se e solo se

```



```

489      per ogni  $a, b, c$  appartenenti ad  $X$ , se  $a$  in
        relazione con  $b$  e  $b$  in relazione con  $c$ ,
        allora
490       $a$  in relazione con  $c$ .*****/
491
492
493  int check_transitivita (rel_bin verifica){
494
495      int i,
496          j,
497          k,
498          transitivita;
499
500  /*IMPOSTO LA TRANSITIVITA INIZIALMENTE COME VERA E
        AZZERO I CONTATORI*/
501      transitivita = 1;
502      i = 0;
503      j = 0;
504      k = 0;
505
506  /*VERIFICA TRANSITIVITA PER NUMERI*/
507
508
509      if (verifica.controllo == 1){
510
511          while (i < verifica.dimensione){
512              j = 0;
513
514              while (j < verifica.dimensione){
515                  k=0;
516
517                  if (verifica.secondo_termine[i] == verifica.
                    primo_termine[j]){
518                      transitivita = 0;
519
520                      while (k < verifica.dimensione){
521                          if (verifica.primo_termine[i] == verifica.
                    primo_termine[k]){
522                              if (verifica.secondo_termine[k]==
                    verifica.secondo_termine[j]){
523                                  transitivita = 1;
524                                  k = verifica.dimensione;
525                              }
526                          }

```

```

527
528         k++;
529     }
530
531     if (transitivita==0){
532         j=verifica.dimensione;
533         i=verifica.dimensione;
534     }
535 }
536
537     j++;
538 }
539
540     i++;
541 }
542 }
543
544
545  /***** VERIFICA TRANSITIVIT PER STRINGHE
        *****/
546
547  if (verifica.controllo == 2){
548
549
550      while (i < verifica.dimensione){
551          j = 0;
552
553          while (j < verifica.dimensione){
554              k=0;
555
556              if (strcmp (verifica.seconda_stringa[i],
557                          verifica.prima_stringa[j]) == 0){
558                  transitivita = 0;
559
560                  while (k < verifica.dimensione){
561                      if (strcmp (verifica.prima_stringa[i],
562                                  verifica.prima_stringa[k]) == 0){
563                          if (strcmp (verifica.seconda_stringa[k],
564                                      verifica.seconda_stringa[j]) == 0){
565                              transitivita = 1;
566                              k = verifica.dimensione;
567                          }
568                      }
569                  }
570              }
571          }
572      }
573  }

```

```

567         k++;
568     }
569
570     if (transitivita==0){
571         j=verifica.dimensione;
572         i=verifica.dimensione;
573     }
574 }
575
576     j++;
577 }
578
579     i++;
580 }
581
582 }
583
584 /****** Controllo se la relazione Transitiva
******/
585
586     if (transitivita == 1)
587         printf ("L'insieme e' transitiva\n");
588
589     else
590         printf ("L'insieme non e' transitiva\n");
591
592 /****** Fine controllo Transittivit *****
*/
593
594     return (transitivita);
595
596 }
597
598 /****** Dicotomia ******/
599
600 int check_dicotomia (rel_bin verifica){
601
602     int i,j,k;
603     int numero_elementi;
604     int dicotomia = 0;
605     int dimensione;
606     int riscontro;
607     int secondo_riscontro;
608     i=0;

```

```

609     j=0;
610     k=i-1;
611     dimensione = verifica.dimensione;
612
613     /****** Dicotomia per numeri *****/
614
615     if (verifica.controllo == 1){
616
617         /****** Conto il numero delle coppie esistenti (
        scarto le coppie uguali) *****/
618
619         while ( i < verifica.dimensione){
620             k = i-1;
621             j = i+1;
622             secondo_riscontro = 0;
623
624             if (i>0){
625                 while ( k >= 0 ){
626                     if (verifica.primo_termine[i] == verifica.
                        primo_termine[k]){
627                         if (verifica.secondo_termine[i] == verifica.
                            secondo_termine[k])
628                             secondo_riscontro = 1;
629                     }
630                     k--;
631                 }
632             }
633
634             if (secondo_riscontro != 1){
635                 while ( j < verifica.dimensione){
636                     if (verifica.primo_termine[i] == verifica.
                        primo_termine[j])
637                         if (verifica.secondo_termine[i] == verifica.
                            secondo_termine[j]){
638                             dimensione--;
639                         }
640                     j++;
641                 }
642             }
643             i++;
644         }
645
646
647         i=0;

```

```

648     j=0;
649     k=0;
650     numero_elementi=0;
651     riscontro = 0;
652  /****** Conto il numero degli elementi
distinti esistenti *****/
653
654     while (i<verifica.dimensione){
655         k=i-1;
656         secondo_riscontro = 0;
657
658         while (k >= 0){
659             if (verifica.primo_termine[i] == verifica.
                primo_termine[k])
660                 secondo_riscontro = 1;
661             k--;
662         }
663         if (secondo_riscontro != 1){
664             if (verifica.primo_termine[i] == verifica.
                secondo_termine[i])
665                 riscontro++;
666
667         }
668         i++;
669     }
670
671     numero_elementi = riscontro;
672
673  /****** Conto quanti dovrebbero essere gli
elementi per avere la dicotomia *****/
674
675     while (numero_elementi > 0){
676         numero_elementi--;
677         riscontro = riscontro + numero_elementi;
678     }
679 }
680
681 /****** VERIFICA DICOTOMICA PER STRINGHE
*****/
682
683     if (verifica.controllo == 2){
684
685  /****** Conto il numero delle coppie esistenti (
scarto le coppie uguali) *****/

```

```

686
687     while ( i < verifica.dimensione){
688         k = i-1;
689         j = i+1;
690         secondo_riscontro = 0;
691     if (i>0){
692         while ( k >= 0 ){
693             if ( (strcmp ( verifica.prima_stringa[i],
694                         verifica.prima_stringa[k])) == 0){
695                 if ( (strcmp ( verifica.seconda_stringa[i],
696                             verifica.seconda_stringa[k])) == 0)
697                     secondo_riscontro = 1;
698             }
699             k--;
700         }
701     if (secondo_riscontro != 1){
702         while ( j < verifica.dimensione){
703             if ( (strcmp ( verifica.prima_stringa[i],
704                         verifica.prima_stringa[j])) == 0)
705                 if ( (strcmp ( verifica.seconda_stringa[i],
706                             verifica.seconda_stringa[j])) == 0){
707                     dimensione--;
708                 }
709             j++;
710         }
711     i++;
712 }
713
714     i=0;
715     k=0;
716     j=0;
717     numero_elementi = 0;
718     /****** Conto il numero degli elementi
719     distinti esistenti *****/
720     while (i<verifica.dimensione){
721         k=i-1;
722         secondo_riscontro = 0;
723
724         while (k >= 0){

```

```

725         if ( (strcmp (verifica.prima_stringa[i],
726                     verifica.prima_stringa[k])) == 0)
727             secondo_riscontro = 1;
728             k--;
729     }
730     if (secondo_riscontro != 1){
731         if ( (strcmp (verifica.prima_stringa[i],
732                     verifica.seconda_stringa[i])) == 0)
733             numero_elementi++;
734     }
735     i++;
736     }
737     riscontro = numero_elementi;
738     /****** Conto quanti dovrebbero essere gli
739     elementi per avere la dicotomia *****/
740     while (numero_elementi > 0){
741     while
742         numero_elementi--;
743         riscontro = riscontro + numero_elementi;
744     }
745     }
746     }
747     }
748     }
749     /****** Verifico se la dicotomia verificata
750     ***** */
751     if (dimensione == riscontro)
752         dicotomia = 1;
753     if (dicotomia == 1 )
754         printf ("...e' dicotomica\n\n");
755     else
756         printf ("...non e' dicotomica\n\n");
757     }
758     }
759     }
760     /****** Fine verifica dicotomia
761     ***** */
762     return (dicotomia);
763 }

```

```

764
765  /*Funzione di verifica dell'ordine totale*/
766
767
768  void ordine_totale (rel_bin verifica){
769
770      int parziale ,
771          dicotomia;
772
773      dicotomia=2;
774      parziale = ordine_parziale (verifica);
775      if (parziale == 1)
776          dicotomia = check_dicotomia (verifica);
777
778      if (parziale == 0)
779          printf ("\n\nl'ordine non e' totale in quanto non e'
              'nemmeno parziale");
780
781      if (dicotomia == 0)
782          printf ("\n\nl'ordine non e' totale in quanto non
              viene rispettata la proprieta' di dicotomia");
783
784      if (dicotomia == 1 && parziale == 1)
785          printf ("\n\nQuindi e' una relazione d'ordine totale
              ");
786
787      printf ("\n\n... Controllo Ordine Totale
              Terminato...\n\n\n");
788  }
789
790  /*Funzione che stabilisce se e' una relazione di
      equivalenza o meno*/
791
792  void relazione_equivalenza (rel_bin verifica){
793
794      int riflessivita;
795      int simmetria;
796      int transitivita;
797
798      riflessivita = check_riflessivita (verifica);
799      simmetria = check_simmetria (verifica);
800      transitivita = check_transitivita (verifica);
801

```



```

802     if (riflessivita == 1 && simmetria == 1 &&
          transitivita == 1)
803     printf ("\n_Quindi_e 'una_relazione_di_equivalenza\n"
            );
804
805     if (riflessivita == 0)
806     printf ("\n_Quindi_non_e 'una_relazione_di_
            equivalenza_perche 'non_riflessiva\n");
807
808     if (simmetria == 0)
809     printf ("\n_Quindi_non_e 'una_relazione_di_
            equivalenza_perche 'non_simmetrica\n");
810
811     if (transitivita == 0)
812     printf ("\n_Quindi_non_e 'una_relazione_di_
            equivalenza_perche 'non_transitiva\n");
813 }
814
815 /*Funzione che stabilisce se la relazione binaria
      acquisita e'una funzione matematica*/
816
817 void check_funzione (rel_bin verifica){
818
819     int i;
820     int k;
821     int termini_diversi;
822     int termini_uguali_prima;
823     int termini_uguali_dopo;
824     int errore;
825
826     if (verifica.controllo == 1){
827
828         i=0;
829         errore=0;
830         termini_diversi=0;
831         termini_uguali_dopo=0;
832         termini_uguali_prima=0;
833         while (i < verifica.dimensione){
834             k=verifica.dimensione-1;
835             termini_uguali_dopo=termini_uguali_prima;
836             while (k > i){
837                 if (verifica.primo_termine[i] == verifica.
                    primo_termine[k]){

```

```

838         if (verifica.secondo_termine[i] != verifica.
            secondo_termine[k]){
839             errore=1;
840             printf ("\nNel %d elemento c'è un errore
                che impedisce alla relazione binaria\n",k
                +1);
841             printf ("\ndi essere una funzione\n");
842             k=i;
843             i=verifica.dimensione;
844         }
845         if (verifica.secondo_termine[i] == verifica.
            secondo_termine[k])
846             termini_uguali_dopo++;
847     }
848     k--;
849 }
850 if (errore == 0 && termini_uguali_dopo ==
    termini_uguali_prima)
851     termini_diversi++;
852
853     termini_uguali_prima = termini_uguali_dopo;
854     i++;
855 }
856 if (errore == 0 && (termini_diversi == (verifica.
    dimensione - termini_uguali_prima))){
857     printf ("\nLa relazione binaria è una funzione\n");
858     check_biiettivita (verifica);
859 }
860 else
861     printf ("\nLa relazione binaria non è una funzione\n
        n");
862 }
863
864 /****** Controllo se c'è una funzione per stringhe
        (le stringhe sono considerate come costanti di
        diverso valore) *****/
865
866 if (verifica.controllo == 2){
867
868     i=0;
869     errore=0;
870     termini_diversi=0;
871     termini_uguali_dopo=0;
872     termini_uguali_prima=0;

```

```

873 while (i < verifica.dimensione){
874     k=verifica.dimensione-1;
875     termini_uguali_dopo=termini_uguali_prima;
876     while (k > i){
877         if ( (strcmp (verifica.prima_stringa[i],verifica
878             .prima_stringa[k])) == 0){
879             errore=1;
880             printf ("\n_Nel_%d_elemento_c'e' un_errore_
881                 che_impedisce_alla_relazione_binaria\n",k
882                 +1);
883             printf ("_di_essere_una_funzione\n");
884             k=i;
885             i=verifica.dimensione;
886         }
887         else
888             termini_uguali_dopo++;
889     }
890     k--;
891     if (errore == 0 && termini_uguali_dopo ==
892         termini_uguali_prima)
893         termini_diversi++;
894     termini_uguali_prima = termini_uguali_dopo;
895     i++;
896     if (errore == 0 && (termini_diversi == (verifica.
897         dimensione - termini_uguali_prima)){
898         printf ("\n_La_relazione_binaria_e'una_funzione\n");
899         check_biiettivita (verifica);
900     }
901     else
902         printf ("\n_La_relazione_binaria_non_e'una_funzione\n
903         n");
904     }
905     printf ("\n\n.....Controllo_Funzione_Terminato...\n
906     \n\n\n");
907 }

```

```

908  /*****FUNZIONE PER IL CHECK DELL'INIETTIVITA
      *****/
909
910  int check_iniettivita (rel_bin verifica){
911
912      int i;
913      int k;
914      int termini_diversi;
915      int termini_uguali_prima;
916      int termini_uguali_dopo;
917      int errore;
918      int iniettivita;
919
920      iniettivita = 0;
921
922      if (verifica.controllo == 1){
923
924          i=0;
925          errore=0;
926          termini_diversi=0;
927          termini_uguali_dopo=0;
928          termini_uguali_prima=0;
929
930          while (i < verifica.dimensione){
931
932              k=verifica.dimensione-1;
933              termini_uguali_dopo=termini_uguali_prima;
934              while (k > i){
935
936                  if (verifica.secondo_termine[i] == verifica.
                      secondo_termine[k]){
937
938                      if (verifica.primo_termine[i] != verifica.
                          primo_termine[k]){
939
940                          errore=1;
941                          printf ("\n_Nel_%d elemento c'e' un_errore_
                                  che_impedisce_alla_funzione\n",k+1);
942                          printf ("_di_essere_iniettiva\n");
943                          k=i;
944                          i=verifica.dimensione;
945                      }
946                      if (verifica.primo_termine[i] == verifica.
                          primo_termine[k])

```

```

947         termini_uguali_dopo++;
948     }
949     k--;
950 }
951 if (errore == 0 && termini_uguali_dopo ==
    termini_uguali_prima)
952     termini_diversi++;
953
954     termini_uguali_prima = termini_uguali_dopo;
955     i++;
956 }
957 if (errore == 0 && (termini_diversi == (verifica.
    dimensione - termini_uguali_prima))) {
958     printf ("\nLa funzione e' iniettiva\n");
959     iniettivita = 1;
960 }
961 else
962     printf ("\nLa funzione non e' iniettiva\n");
963
964
965 }
966
967 /****** Controllo iniettivita' per stringhe
******/
968
969 if (verifica.controllo == 2){
970
971     i=0;
972     errore=0;
973     termini_diversi=0;
974     termini_uguali_dopo=0;
975     termini_uguali_prima=0;
976
977     while (i < verifica.dimensione){
978         k=verifica.dimensione-1;
979         termini_uguali_dopo=termini_uguali_prima;
980         while (k > i){
981             if ( (strcmp (verifica.seconda_stringa[i],
                verifica.seconda_stringa[k])) == 0){
982                 if ( (strcmp (verifica.prima_stringa[i],
                    verifica.prima_stringa[k])) != 0){
983                     errore=1;
984                     printf ("\nNel %d elemento c'e' un errore
                        che impedisce alla funzione\n", k+1);

```

```

985         printf ("_di_essere_iniettiva\n");
986         k=i;
987         i=verifica.dimensione;
988     }
989     if ( (strcmp (verifica.prima_stringa[i],
990                 verifica.prima_stringa[k])) == 0)
991         termini_uguali_dopo++;
992 }
993     k--;
994 }
995     if (errore == 0 && termini_uguali_dopo ==
996         termini_uguali_prima)
997         termini_diversi++;
998     termini_uguali_prima = termini_uguali_dopo;
999     i++;
1000 }
1001     if (errore == 0 && (termini_diversi == (verifica.
1002         dimensione - termini_uguali_prima))){
1003         printf ("\n_La_funzione_e' iniettiva");
1004         iniettivita = 1;
1005     }
1006     else
1007         printf ("\n_La_funzione_non_e' iniettiva");
1008 }
1009 return (iniettivita);
1010 }
1011
1012 /******FUNZIONE PER IL CHECK DELLA
1013 SURIETTIVITA'******/
1014
1015 int check_suriettivita (rel_bin verifica){
1016     /****** La suriettivit sempre verificata in quanto
1017     il dominio e il codominio ******/
1018     /* sono entrambi i rispettivi x,y acquisiti, quindi
1019     non ho elementi y non associati a x */
1020     int suriettivita;
1021     suriettivita = 1;
1022     return (suriettivita);
1023 }

```

```

1023
1024  /******FUNZIONE PER IL CHECK DELLA
      BIIETTIVITA'******/
1025
1026  void check_biiettivita (rel_bin verifica){
1027
1028      int    surriettivita ,
1029            iniettivita ;
1030
1031      surriettivita = check_surriettivita (verifica);
1032      iniettivita = check_iniettivita (verifica);
1033
1034
1035      if ( surriettivita == 1 && iniettivita == 1)
1036          printf ("\nla_funzione_e'biiettiva");
1037      else
1038          printf ("\nla_funzione_non_e'biiettiva");
1039      return;
1040  }
1041
1042
1043  int check_antisimmetria (rel_bin verifica){
1044
1045      int i,
1046          j,
1047          riscontro ,
1048          antisimmetria;
1049
1050      antisimmetria = 1;
1051
1052
1053      i = 0;
1054      j = 0;
1055      riscontro = 0;
1056
1057  /*Check della antisimmetria per numeri*/
1058
1059      if (verifica.controllo == 1){
1060
1061          while ( i < verifica.dimensione){
1062
1063              j = 0;
1064              while ( j < verifica.dimensione){
1065

```

```

1066         if (verifica.primo_termine[i] == verifica.
            secondo_termine[j])
1067         if (verifica.primo_termine[j] == verifica.
            secondo_termine[i])
1068         if (verifica.primo_termine[i] == verifica.
            primo_termine[j])
1069             riscontro++;
1070         j++;
1071     }
1072
1073     if (riscontro == 0){
1074         j = verifica.dimensione;
1075         i = verifica.dimensione;
1076         antisimmetria = 0;
1077     }
1078     riscontro = 0;
1079     i++;
1080 }
1081
1082 }
1083
1084 /* Check della antisimmetria per stringhe */
1085
1086 if (verifica.controllo == 2){
1087
1088     while ( i < verifica.dimensione){
1089
1090         j = 0;
1091         while ( j < verifica.dimensione){
1092
1093             if (strcmp (verifica.prima_stringa[i], verifica
                .seconda_stringa[j]) == 0 )
1094             if (strcmp (verifica.prima_stringa[j],
                verifica.seconda_stringa[i]) == 0 )
1095             if (strcmp (verifica.prima_stringa[j],
                verifica.prima_stringa[i]) == 0 )
1096                 riscontro++;
1097
1098             j++;
1099         }
1100
1101         if (riscontro == 0){
1102             j = verifica.dimensione;
1103             i = verifica.dimensione;

```



```

1104         antisimmetria = 0;
1105     }
1106     riscontro = 0;
1107     i++;
1108 }
1109
1110 }
1111
1112 /****** Controllo se la simmetria stata verificata
******/
1113
1114     if (antisimmetria == 1)
1115         printf ("_e'_antisimmetrica\n");
1116     else
1117         printf ("_non_e'_antisimmetrica\n");
1118
1119 /****** Fine controllo simmetria ******/
1120
1121     return (antisimmetria);
1122 }

```

4.2 Test

```
1 #include<stdio.h>
2 #include"librerie/Progetto.h"
3
4 int main(void){
5     struct relBin RelazioneBinaria;
6     int scelta;
7     int scan;
8     int test_terminati;
9     scan = 0;
10    test_terminati = 0;
11    printf("\n_Programma_per_effettuare_i_Test_sulla_
        libreria\n");
12
13
14    printf("\n\n_Digita_il_numero_corrispondente_all_
        azione_che_si_vuole_svolgere\n");
15    printf("\n_1)_Test_Acquisizione\n_2)_Esci\n");
16
17
18    while((scelta < 1) || (scelta > 2) || scan != 1){
19        printf("\n_scelta:_");
20        fflush(stdin);
21        scan = scanf("%d",&scelta);
22    }
23    if(scelta == 1)
24        RelazioneBinaria = acquisizione(RelazioneBinaria);
25
26    if(scelta == 2){
27        printf("\n\n..... Test_terminati ..... \n\n");
28        test_terminati = 1;
29    }
30
31    scelta = -1;
32    while(scelta != 7 && test_terminati != 1){
33        printf("\n\n_Digita_il_numero_corrispondente_all_
        azione_che_si_vuole_svolgere\n");
34        printf("\n_1)_Test_Acquisizione\n_2)_Test_Stampa\n_
        3)_Test_verifica_ordine_parziale\n_4)_Test_
        verifica_ordine_totale");
35        printf("\n_5)_Test_verifica_relazione_d'equivalenza\
        n_6)_Test_funzione\n_7)_Esci\n");
36        scelta = -1;
```

```

37  while((scelta < 1) || (scelta > 7) || scan != 1){
38      printf("\n_scelta:_");
39      fflush(stdin);
40      scan = scanf("%d",&scelta);
41  }
42
43
44  if(scelta == 1)
45      RelazioneBinaria = acquisizione(RelazioneBinaria);
46  if(scelta == 2)
47      stampa(RelazioneBinaria);
48  if(scelta == 3)
49      ordine_parziale(RelazioneBinaria);
50  if(scelta == 4)
51      ordine_totale(RelazioneBinaria);
52  if(scelta == 5)
53      relazione_equivalenza(RelazioneBinaria);
54  if(scelta == 6)
55      check_funzione(RelazioneBinaria);
56  if(scelta == 7){
57      printf("\n\n.....Test_terminati.....\n\n");
58      test_terminati = 1;
59  }
60  }
61  return(0);
62
63  }

```

4.3 Makefile

```
Test.exe: Test.c Makefile
gcc -ansi -Wall -O Test.c -o Test.exe
pulisci:
rm -f Test.o
pulisci_tutto:
rm -f Test.exe Test.o
```

5 Testing del programma

5.1 Test 1:

Test di Relazione d'ordine Totale.

Inputs: (a,a)(a,b)(b,b)

Outputs: checkriflessività : 1, checksimmetria : 0, checktransitività : 1
checkdicotomia : 1, la relazione è una relazione d'ordine totale in quanto è
rispetta anche la proprietà di Dicotomia.

```
6> Test funzione
7> Esci

scelta: 2

La relazione binaria e':
<(a,a);(a,b);(b,b) >

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci

scelta: _
```

```
La relazione:
e' riflessiva
e' asimmetrica
e' transitiva

Quindi e' una relazione d'ordine parziale

... Controllo Ordine Parziale Terminato ...

e' dicotomica

Quindi e' una relazione d'ordine totale
... Controllo Ordine Totale Terminato ...
```

5.2 Test 2:

Test di Relazione d'ordine Parziale.

Inputs:(a,a)(b,b)(a,b)(c,c)

Outputs:checkriflessività : 1, checksimmetria : 0, checktransitività : 1 la relazione è una relazione d'ordine parziale in quanto rispetta le proprietà.

```
6> Test funzione
7> Esci

scelta: 2

La relazione binaria e':
<(a,a);(a,b);(b,b);(c,c) >

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci

scelta:
```

```
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci

scelta: 3

La relazione:
e' riflessiva
e' asimmetrica
e' transitiva

Quindi e' una relazione d'ordine parziale

... Controllo Ordine Parziale Terminato ...
```

5.3 Test 3:

Test di Relazione d'ordine non Parziale.

Inputs:(a,a)(b,b)(c,c)(d,d)(e,e)(a,b)(b,c)

Outputs:checkriflessività : 1, checksimmetria : 0, checktransitività : 0 la relazione non è una relazione d'ordine parziale in quanto non rispetta le proprietà.

```
6> Test funzione
7> Esci
scelta: 2
La relazione binaria e':
<(a,a);(b,b);(c,c);(d,d);(e,e);(a,b);(b,c) >

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere
1) Test Acquisizione
2) Test Stampa
3) Test verifica ordine parziale
4) Test verifica ordine totale
5) Test verifica relazione d'equivalenza
6) Test funzione
7) Esci
scelta:
```

```
6> Test funzione
7> Esci
scelta: 3
La relazione:
e' riflessiva
e' asimmetrica
non e' transitiva
Non e' una relazione d'ordine parziale in quanto non rispetta tutte le proprietà
manca la proprietà di transitività

... Controllo Ordine Parziale Terminato ...

Digita il numero corrispondente all'azione che si vuole svolgere
```

5.4 Test 4:

Test di Relazione d'equivalenza.

Inputs:(a,a)(a,b)(b,a)(b,b)

Outputs:checkriflessività : 1, checksimmetria : 1, checktransitività : 1 checkdicotomia : 0, la relazione è una relazione d'equivalenza in quanto rispetta le proprietà.

```
6> test funzione
7> Esci

scelta: 2

La relazione binaria e':
<(a,a);(a,b);(b,a);(b,b)>

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> test verifica ordine parziale
4> test verifica ordine totale
5> test verifica relazione d'equivalenza
6> test funzione
7> Esci

scelta:
```

```
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> test funzione
7> Esci

scelta: 5
e' riflessiva
e' simmetrica
e' transitiva

Quindi e' una relazione di equivalenza

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> test Stampa
3> test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> test funzione
7> Esci

scelta:
```


5.5 Test 5:

Test di Relazione non d'equivalenza.

Inputs:(a,a)(a,b)(b,c)

Outputs:checkriflessività : 0, checksimmetria : 0, checktransitività : 0 la relazione non è una relazione d'ordine d'equivalenza in quanto non rispetta le proprietà.

```
6> Test funzione
7> Esci

scelta: 2

La relazione binaria e':
<(a,a);(a,b);(b,c)>

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci

scelta:
```

```
7> Esci

scelta: 5
non e' riflessiva
e' asimmetrica
non e' transitiva

Quindi non e' una relazione di equivalenza perche' non riflessiva
Quindi non e' una relazione di equivalenza perche' non simmetrica
Quindi non e' una relazione di equivalenza perche' non transitiva

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci

scelta:
```

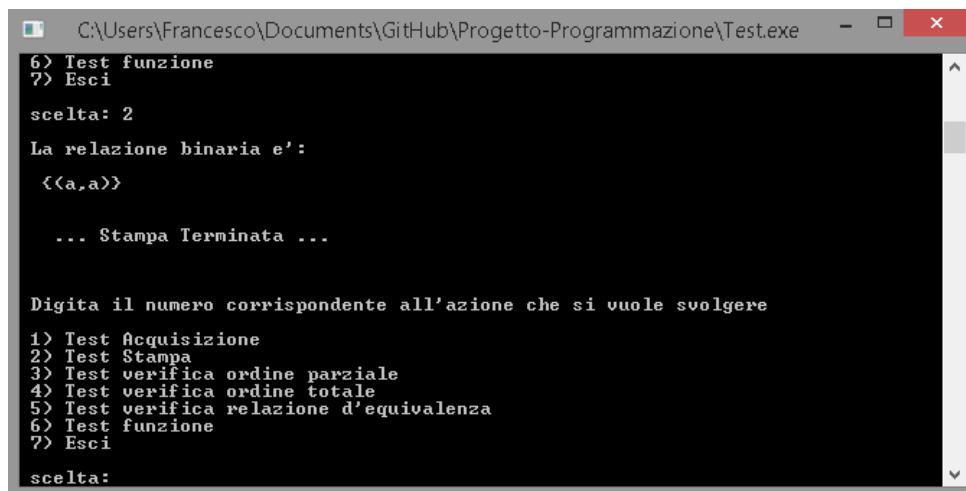
5.6 Test 6:

Test di Funzione.

Inputs:(a,a) Outputs:La relazione binaria è una funzione.

La relazione binaria è iniettiva.

La relazione binaria è biiettiva.



```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

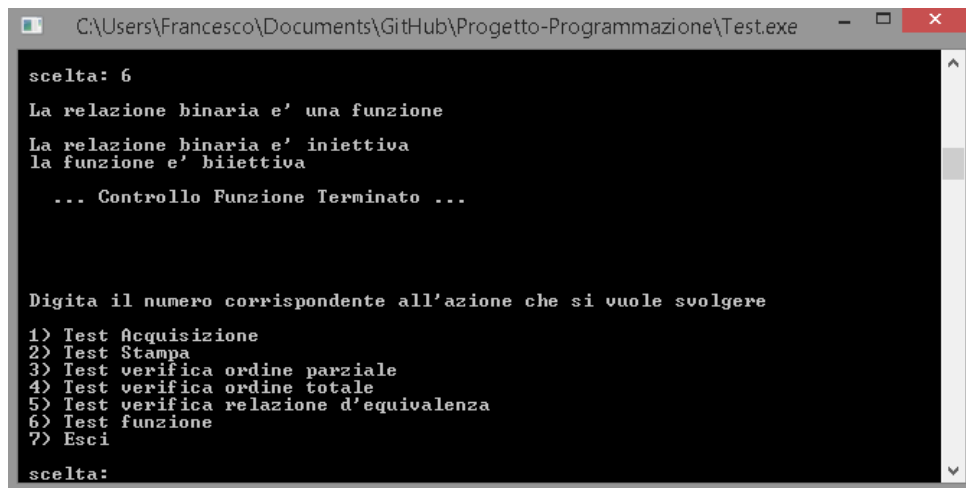
scelta: 2

La relazione binaria e':

  <<a,a>>

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci
scelta:
```



```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe

scelta: 6

La relazione binaria e' una funzione
La relazione binaria e' iniettiva
la funzione e' biiettiva

... Controllo Funzione Terminato ...

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci
scelta:
```

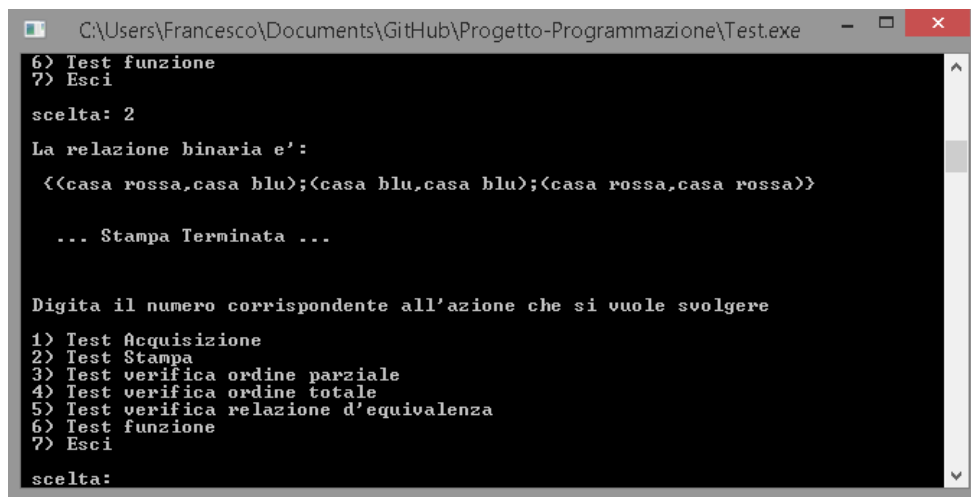
5.7 Test 7:

Test per verificare il controllo degli inputs.

Inputs:(casa rossa,casa blu)(casa blu,casa blu)(casa rossa,casa rossa)

Outputs:check_riflessività : 1,check_simmetria : 1, check_transitività : 1
dicotomia :1 la relazione è una relazione d'ordine totale in quanto rispetta le proprietà.

le funzioni funzionano anche con input contenuti degli spazi.



```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

scelta: 2

La relazione binaria e':

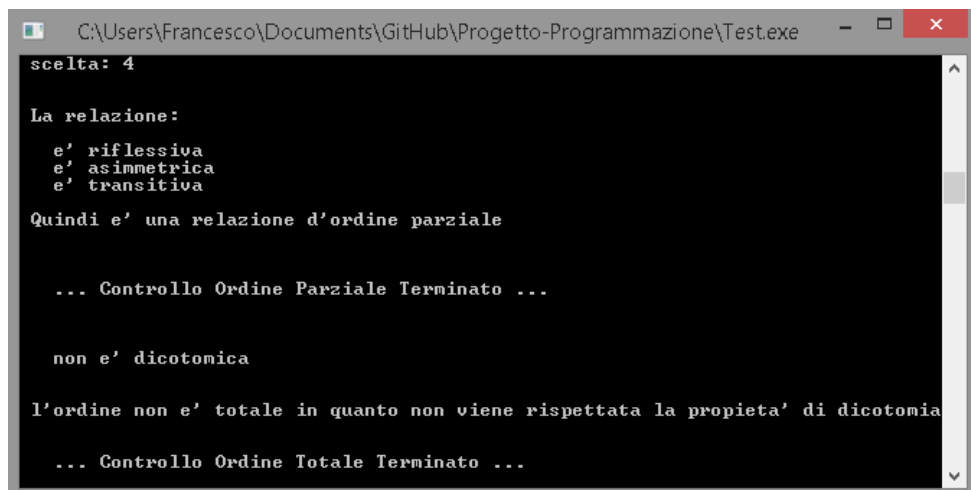
<(casa rossa,casa blu);(casa blu,casa blu);(casa rossa,casa rossa)>

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere

1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci

scelta:
```



```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe

scelta: 4

La relazione:

e' riflessiva
e' asimmetrica
e' transitiva

Quindi e' una relazione d'ordine parziale

... Controllo Ordine Parziale Terminato ...

non e' dicotomica

l'ordine non e' totale in quanto non viene rispettata la proprieta' di dicotomia

... Controllo Ordine Totale Terminato ...
```

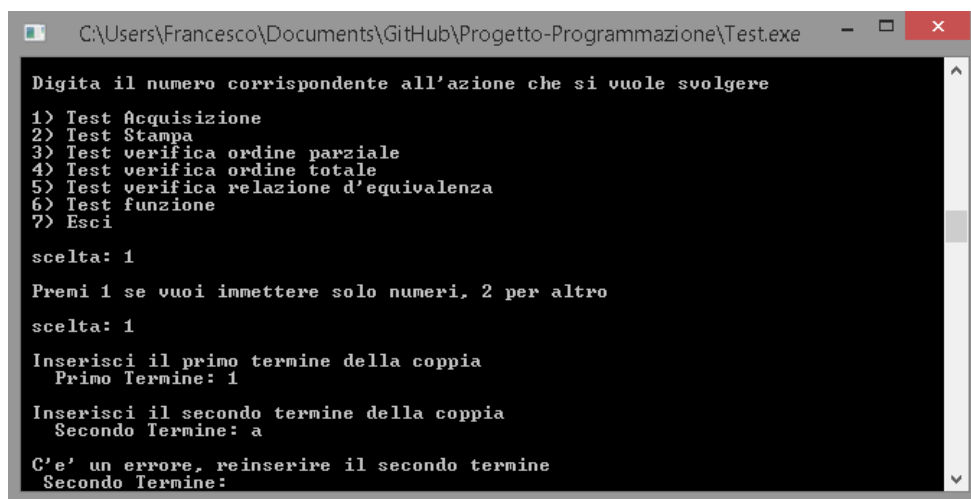
5.8 Test 8:

Test per inserire stringhe in una relazione numerica.

Inputs:(1,a)

Outputs: c' è un errore reinserisci il valore.

stampa errore in quanto si era selezionato di voler immettere un input di tipo numerico.



```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci

scelta: 1

Premi 1 se vuoi immettere solo numeri, 2 per altro
scelta: 1

Inserisci il primo termine della coppia
Primo Termine: 1

Inserisci il secondo termine della coppia
Secondo Termine: a

C'e' un errore, reinserire il secondo termine
Secondo Termine:
```

5.9 Test 9:

Test per vedere se una relazione binaria qualunque e' una funzione.

Inputs:(1,2)(1,1)

Outputs: La relazione binaria non è una funzione

Nel 2 elemento c'è un errore che impedisce alla relazione binaria di essere una funzione;

```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

scelta: 2

La relazione binaria e':
< (1.00,1.00) ; (1.00,2.00)>

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci

scelta:
```

```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
scelta: 6

Nel 2 elemento c'e' un errore che impedisce alla relazione binaria
di essere una funzione

La relazione binaria non e' una funzione

... Controllo Funzione Terminato ...

Digita il numero corrispondente all'azione che si vuole svolgere
1> Test Acquisizione
2> Test Stampa
3> Test verifica ordine parziale
4> Test verifica ordine totale
5> Test verifica relazione d'equivalenza
6> Test funzione
7> Esci

scelta:
```

5.10 Test 10:

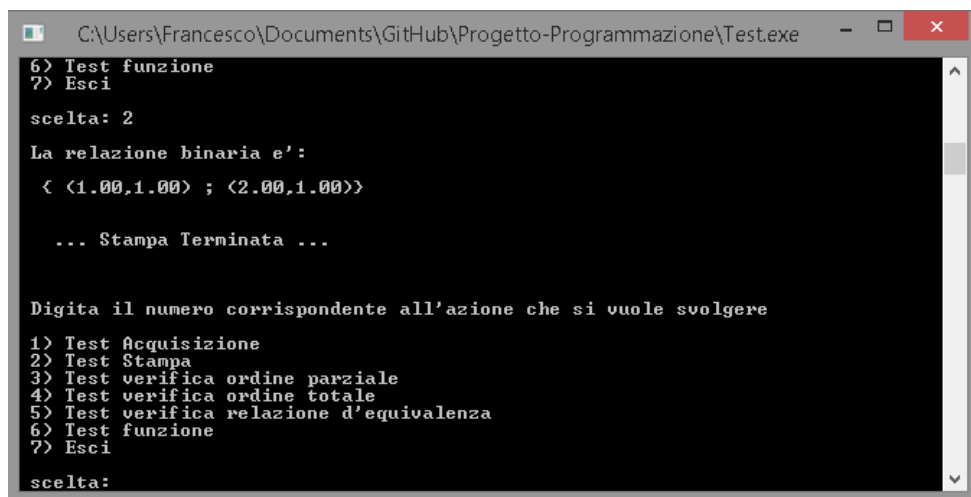
Inputs:(1,1)(2,1)

Outputs: La relazione binaria è una funzione

Nel 2 elemento c'è un errore che impedisce alla funzione di essere iniettiva

La funzione non è iniettiva

La funzione non è biiettiva



```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

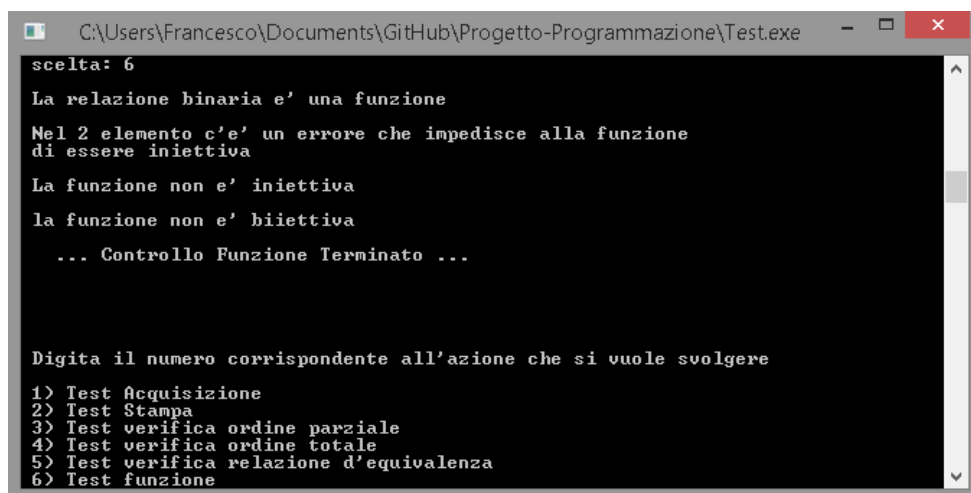
scelta: 2

La relazione binaria e':
{ <1.00,1.00> ; <2.00,1.00> }

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere
1) Test Acquisizione
2) Test Stampa
3) Test verifica ordine parziale
4) Test verifica ordine totale
5) Test verifica relazione d'equivalenza
6) Test funzione
7) Esci

scelta:
```



```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe

scelta: 6

La relazione binaria e' una funzione
Nel 2 elemento c'e' un errore che impedisce alla funzione
di essere iniettiva
La funzione non e' iniettiva
la funzione non e' biiettiva

... Controllo Funzione Terminato ...

Digita il numero corrispondente all'azione che si vuole svolgere
1) Test Acquisizione
2) Test Stampa
3) Test verifica ordine parziale
4) Test verifica ordine totale
5) Test verifica relazione d'equivalenza
6) Test funzione
```

6 Verica del programma

Questa porzione di codice fa in modo che una volta eseguito si abbia nel valore c la sommatoria del numero di elementi distinti inseriti dall'utente.

```
while(numero_elementi>0)
{ numero_elementi - -;
  c = c + numero_elementi;
}
```

La postcondizione è

$$R = (c = \sum_{j=0}^{numero_elementi-1} numero_elementi - j)$$

si pu rendere la tripla vera mettendo preconditione vero in quanto:

-Il predicato

$$P = (numero_elementi > 0 \wedge c = \sum_{j=0}^{numero_elementi-1} numero_elementi - j)$$

e la funzione :

$$tr(numero_elementi) = numero_elementi - 1)$$

soddisfano le ipotesi del teorema dell'invariante di ciclo in quanto:

$$*\{P \wedge numero_elementi > 0\} c = c + numero_elementi; numero_elementi = numero_elementi - -; \{P\}$$

segue da :

$$P_{numero_elementi, numero_elementi-1} \wedge c \quad \sum_{j=0}^{numero_elementi-2} numero_elementi - j$$

e denotato con P' quest'ultimo predicato, da:

$$\begin{aligned} P'_{c,c+numero_elementi} &= (numero_elementi > 0 \wedge c + numero_elementi = \\ &= \sum_{j=0}^{numero_elementi-2} numero_elementi - j) \end{aligned}$$

$$\begin{aligned} P'_{c,c+numero_elementi} &= (numero_elementi > 0 \wedge c = \\ &= \sum_{j=0}^{numero_elementi-1} numero_elementi - j) \end{aligned}$$

$$\begin{aligned} &\text{in quanto denotato con } P'' \text{ quest'ultimo predicato, si ha: } (P \wedge numero_elementi > 1) = \\ &(numero_elementi > 0 \wedge c = \sum_{j=0}^{numero_elementi-1} numero_elementi - j \wedge numero_elementi > 1) \\ &| = P'' \end{aligned}$$

* Il progresso è garantito dal fatto che $tr(numero_elementi)$ decresce di un unità ad ogni iterazione in quanto $numero_elementi$ viene decrementata di un' unità ad ogni iterazione.

* La limitatezza segue da:

$$\begin{aligned} (P \wedge tr(numero_elementi) < 1) &= (numero_elementi > 0 \wedge c = \sum_{j=0}^{numero_elementi-1} numero_elementi - \\ &j \wedge numero_elementi > 1) \\ &\equiv (c = \sum_{j=0}^{numero_elementi-1} numero_elementi - j) \end{aligned}$$

$$\begin{aligned} &| = numero_elementi > numero_elementi - 1 \\ &\text{Poichè:} \end{aligned}$$

$$\begin{aligned}
& (P \wedge \text{numero_elementi} < 1) = (\text{numero_elementi} > 0 \wedge c = (P \wedge \text{numero_elementi} > 1) = \\
& (\text{numero_elementi} > 0 \wedge c = \\
& = \sum_{j=0}^{\text{numero_elementi}-1} \text{numero_elementi} - j \wedge \text{numero_elementi} < 1) \\
& \equiv (\text{numero_elementi} = 1 \wedge c = \\
& = \sum_{j=0}^{\text{numero_elementi}-1} \text{numero_elementi} - j \wedge \text{numero_elementi} < 1)))
\end{aligned}$$

Dal corollario del teorema dell'invariabilit  di ciclo si ha che P pu essere usato solo come preconditione dell'intera istruzione di ripetizione.

-Proseguendo infine a ritroso si ottiene prima:

$$P_{\text{numero_elementi},0} = (0 < = 0 < = \text{numero_elementi} \wedge c = \sum_{j=0}^{0-1} \text{numero_elementi} - j) \text{ (c} = 0)$$

e poi, denotato con P''' quest'ultimo predicato si ha:

$$P'''_{c,0} = (0 = 0) = \text{vero}$$