

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 12, 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  *****/
9  appartenenti alla Relazione*****/
10
11 struct relBin{
12     /***** Coppia Numerica *****/
13     double *primo_termine ,
14            *secondo_termine ;
15
16     /***** Coppia Qualsiasi*****/
17     char **prima_stringa ,
18           **seconda_stringa ;
19
20     /**** Variabili per salvare se ho acquisito una
21           coppia numerica o no e il numero delle coppie****
22     */
23     int controllo ,
24         dimensione ;
25 };
26
27 /*DICHIARO LE FUNZIONI*/
28 int check_simmetria(struct relBin);
29 int check_riflessivita(struct relBin);
30 int check_transitivita(struct relBin);
31 int check_suriettivita(struct relBin);
32 void check_biiettivita(struct relBin);
33
34 /*****Funzione di acquisizione
35 *****/
36
37 struct relBin acquisizione(struct relBin relazione){
38
39     int acquisizione_finita = 0;
40     int scan = 0;
```

```

36
37 relazione.dimensione = 0;
38 relazione.primo_termine = (double *) malloc(2);
39 relazione.secondo_termine = (double *) malloc(2);
40 relazione.prima_stringa = (char **) malloc(100);
41 relazione.seconda_stringa = (char **) malloc(100);
42
43 while((relazione.controllo < 1) || (relazione.
    controllo > 2) || scan != 1){
44     fflush(stdin);
45     printf("\n_Premi_1_se_vuoi_inmettere_solo_numeri,_2_
        per_altro\n");
46     printf("\n_scelta:_");
47     scan = scanf("%d",&relazione.controllo);
48 }
49
50 /** resetto scan a 0 **/
51 scan=0;
52
53 /*Acquisizione Numerica*/
54
55 if(relazione.controllo == 1){
56     while(acquisizione_finita == 0){
57         relazione.dimensione++;
58         acquisizione_finita = 2;
59
60         /*Acquisisco il primo termine della coppia*/
61
62         printf("\n_Inserisci_il_primo_termine_della_coppia
            _\n");
63         relazione.primo_termine = (double *) realloc(
            relazione.primo_termine, (relazione.dimensione
            +1) * sizeof(double));
64         scan = 0;
65         /*Check del primo termine della coppia*/
66
67         while(scan != 1){
68             printf("_Primo_Termine:_");
69             fflush(stdin);
70             scan = scanf("%lf",&relazione.primo_termine[
                relazione.dimensione - 1]);
71         if(scan == 0)
72             printf("\n_C'e'_un_errore,_reinserire_il_primo_
                termine\n");

```

```

73     }
74
75     /* Acquisisco il secondo termine della coppia */
76     scan = 0;
77     printf("\n Inserisci il secondo termine della
       coppia\n");
78     relazione.secondo_termine = (double *) realloc(
       relazione.secondo_termine, (relazione.
       dimensione+1) * sizeof(double));
79
80     /* Check del secondo termine della coppia */
81
82     while(scan != 1){
83         printf("\n Secondo Termine: ");
84         fflush(stdin);
85         scan = scanf("%lf",&relazione.secondo_termine[
            relazione.dimensione - 1]);
86         if(scan == 0)
87             printf("\n C'è un errore, reinserire il secondo
            termine\n");
88     }
89
90     /* Chiedo all'utente se ci sono altre coppie */
91
92     while(acquisizione_finita < 0 || acquisizione_finita
       > 1 || scan != 1){
93         printf("\n Vuoi acquisire un'altra coppia? immetti
            1 per uscire, 0 per continuare\n");
94         printf("\n scelta: ");
95         fflush(stdin);
96         scan = scanf("%d",&acquisizione_finita);
97     }
98 }
99 }
100
101 /* resetto scan a 0 */
102 scan = 0;
103
104 /* Acquisizione con stringhe */
105 if(relazione.controllo == 2){
106     while(acquisizione_finita == 0){
107         relazione.dimensione++;
108         acquisizione_finita = 2;
109

```

```

110  /* Acquisisco il primo termine della coppia*/
111
112      printf("\nInserisci il primo termine della coppia \n");
113      printf("\nPrimo Termine: ");
114      relazione.prima_stringa[relazione.dimensione - 1]
          = (char *) malloc(50);
115      scan = scanf("%[^\\n]s", relazione.prima_stringa[
          relazione.dimensione - 1]);
116
117  /* Acquisisco il secondo termine della coppia*/
118
119      printf("\nInserisci il secondo termine della coppia \n");
120      printf("\nSecondo Termine: ");
121      relazione.seconda_stringa[relazione.dimensione -
          1] = (char *) malloc(50);
122      scan = scanf("%[^\\n]s", relazione.seconda_stringa[
          relazione.dimensione - 1]);
123
124  /* riassetto scan a 0*/
125      scan = 0;
126
127  /* Chiedo all'utente se ci sono altre coppie*/
128
129      while(acquisizione_finita < 0 ||
          acquisizione_finita > 1 || scan != 1){
130
131          printf("\nVuoi acquisire un'altra coppia? \n");
132          scan = scanf("%d",&acquisizione_finita);
133      }
134  }
135  }
136
137  printf("\n\n.....Acquisizione Terminata....\n\n");
138  return relazione;
139  }
140
141  /******FUNZIONE DI STAMPA*****
142
143  void stampa(struct relBin stampa){
144

```

```

145  int i = 0;
146
147  printf("\nLa relazione binaria e'");
148  printf("\n\n{");
149
150  /******Stampa per coppie numeriche *****/
151
152      if(stampa.controllo == 1){
153          while(i < stampa.dimensione){
154
155              printf("_(%.2lf,%.2lf)",stampa.primo_termine[i
156                  ],stampa.secondo_termine[i]);
157              if(i+1 != stampa.dimensione)
158                  printf("_;");
159              i++;
160          }
161
162  /******Stampa per coppie non numeriche *****/
163
164      if(stampa.controllo == 2){
165          while(i < stampa.dimensione){
166              printf("(%s,%s)",stampa.prima_stringa[i],
167                  stampa.seconda_stringa[i]);
168              if(i+1 != stampa.dimensione)
169                  printf(";");
170              i++;
171          }
172      }
173
174  /****** Fine Stampa *****/
175
176      printf("}\n");
177      printf("\n\n..._Stampa_Terminata_...\n\n");
178
179  }
180
181  /******FUNZIONE DI VERIFICA DI RELAZIONI D
182      'ORDINE******/
183
184  int ordine_parziale(struct relBin verifica){
185
186      int riflessivita ,

```



```

186     transitivita ,
187     antisimmetria ,
188     parziale ;
189
190     /*STAMPO LE PROPIETA ' DELLA RELAZIONE*/
191
192     printf("\n\nLa relazione:\n\n");
193
194     /****** Chiamo le funzioni per poter stabilire le
        propriet ******/
195
196     riflessivita = check_riflessivita(verifica);
197     antisimmetria = check_antisimmetria(verifica);
198     transitivita = check_transitivita(verifica);
199
200     /****** Controllo se rispetta le propriet per
        essere una relazione d'ordine parziale******/
201
202     if(transitivita == 1 && antisimmetria == 1 &&
        riflessivita == 1){
203         parziale = 1;
204         printf("\n_Quindi e' una relazione d'ordine _
            parziale\n\n");
205     }
206     else{
207
208         printf("\n_Non e' una relazione d'ordine _parziale _
            in quanto non rispetta tutte le _propieta '\n");
209         parziale = 0;
210     }
211     if(transitivita == 0)
212         printf("\n_manca la _propieta ' _di _transitivita '\n")
            ;
213     if(antisimmetria == 0)
214         printf("\n_manca la _propieta ' _di _antisimmetria\n")
            ;
215     if(riflessivita == 0)
216         printf("\n_manca la _propieta ' _di _riflessivita '\n")
            ;
217     /****** Fine controllo Ordine Parziale
        ******/
218
219     printf("\n\n... _Controllo _Ordine _Parziale _
        Terminato _... \n\n\n\n");

```

```

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

```

```

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

```

```

293     }
294 }
295     if(riscontro != 0)
296         i++;
297
298 /**** Se non c'è stato un riscontro di riflessivit
esco e setto la riflessivit a 0 *****/
299
300     else{
301         i=verifica.dimensione;
302         riflessivita = 0;
303     }
304
305     if(secondo_riscontro != 0)
306         k++;
307
308     else{
309         k=verifica.dimensione;
310         riflessivita = 0;
311     }
312 }
313
314 }
315
316 /****** VERIFICA RIFLESSIVIT PER STRINGHE
******/
317
318 if(verifica.controllo == 2){
319     riscontro = 0;
320     secondo_riscontro = 0;
321     if(strcmp(verifica.prima_stringa[i], verifica.
322         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  /***** Controllo la riflessivit per gli
      elementi del primo insieme
      *****/
335
336      while(j < verifica.dimensione){
337          if(j == i)
338              j++;
339          else{
340              if(strcmp(verifica.prima_stringa[i], verifica
                          .prima_stringa[j]) == 0)
341                  if(strcmp(verifica.prima_stringa[j],
                              verifica.seconda_stringa[j]) == 0)
342                      riscontro++;
343
344              j++;
345          }
346      }
347
348      j = 0;
349
350  /***** Controllo la riflessivit per gli
      elementi del secondo insieme
      *****/
351
352      while(j < verifica.dimensione){
353          if(j == k)
354              j++;
355          else{
356              if(strcmp(verifica.seconda_stringa[k],
                          verifica.seconda_stringa[j]) == 0)
357                  if(strcmp(verifica.prima_stringa[j],
                              verifica.seconda_stringa[j]) == 0)
358                      secondo_riscontro++;
359
360              j++;
361          }
362      }
363      if(riscontro != 0)
364          i++;
365
366      else{
367          i=verifica.dimensione;
368          riflessivita = 0;
369      }

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

```

610     d=a-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( a < verifica.dimensione){
620             d = a-1;
621             b = a+1;
622             secondo_riscontro = 0;
623
624             if(a>0){
625                 while ( d >= 0 ){
626                     if(verifica.primo_termine[a] == verifica.
                        primo_termine[d]){
627                         if(verifica.secondo_termine[a] == verifica.
                           secondo_termine[d])
628                             secondo_riscontro = 1;
629                     }
630                     d--;
631                 }
632             }
633
634             if(secondo_riscontro != 1){
635                 while ( b < verifica.dimensione){
636                     if(verifica.primo_termine[a] == verifica.
                        primo_termine[b])
637                         if(verifica.secondo_termine[a] == verifica.
                           secondo_termine[b]){
638                             dimensione--;
639                         }
640                     b++;
641                 }
642             }
643             a++;
644         }
645
646
647         a=0;
648         b=0;

```

```

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

```

```

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

```

```

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

```

```

765 }
766
767 /*Funzione di verifica dell'ordine totale*/
768
769
770 void ordine_totale (struct relBin verifica){
771
772     int parziale ,
773         dicotomia;
774
775     dicotomia=2;
776     parziale = ordine_parziale (verifica);
777     if(parziale == 1)
778     dicotomia = check_dicotomia (verifica);
779
780     if(parziale == 0)
781         printf("\n\nl'ordine non e' totale in quanto non e'
            'nemmeno parziale");
782
783     if(dicotomia == 0)
784         printf("\n\nl'ordine non e' totale in quanto non
            viene rispettata la proprieta' di dicotomia");
785
786     if(dicotomia == 1 && parziale == 1)
787         printf("\n\nQuindi e' una relazione d'ordine totale
            ");
788
789     printf("\n\n.....Controllo Ordine Totale Terminato
            ....\n\n\n");
790 }
791
792 /*Funzione che stabilisce se e' una relazione di
    equivalenza o meno*/
793
794 void relazione_equivalenza(struct relBin verifica){
795
796     int riflessivita;
797     int simmetria;
798     int transitivita;
799
800     riflessivita = check_riflessivita(verifica);
801     simmetria = check_simmetria(verifica);
802     transitivita = check_transitivita(verifica);
803

```



```

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

```

```

840         if(verifica.secondo_termine[i] != verifica.
            secondo_termine[k]){
841             errore=1;
842             printf("\n_Nel_%d_elemento_c'e'_un_errore_
                che_impedisce_alla_relazione_binaria\n",k
                +1);
843             printf("_di_essere_una_funzione\n");
844             k=i;
845             i=verifica.dimensione;
846         }
847         if(verifica.secondo_termine[i] == verifica.
            secondo_termine[k])
848             termini_uguali_dopo++;
849     }
850     k--;
851 }
852 if(errore == 0 && termini_uguali_dopo ==
    termini_uguali_prima)
853     termini_diversi++;
854
855     termini_uguali_prima = termini_uguali_dopo;
856     i++;
857 }
858 if(errore == 0 && (termini_diversi == (verifica.
    dimensione - termini_uguali_prima))){
859     printf("\n_La_relazione_binaria_e'_una_funzione\n");
860     check_biiettivita(verifica);
861 }
862 else
863     printf("\n_La_relazione_binaria_non_e'_una_funzione\
        n");
864 }
865
866 /****** Controllo se c' una funzione per stringhe
    (le stringhe sono considerate come costanti di
    diverso valore) *****/
867
868 if(verifica.controllo == 2){
869
870     i=0;
871     errore=0;
872     termini_diversi=0;
873     termini_uguali_dopo=0;
874     termini_uguali_prima=0;

```

```

875 while(i < verifica.dimensione){
876     k=verifica.dimensione-1;
877     termini_uguali_dopo=termini_uguali_prima;
878     while(k > i){
879         if((strcmp(verifica.prima_stringa[i],verifica.
            prima_stringa[k])) == 0){
880             if((strcmp(verifica.seconda_stringa[i],
                verifica.seconda_stringa[k])) != 0){
881                 errore=1;
882                 printf("\n_Nel_%d_elemento_c'e'_un_errore_
                    che_impedisce_alla_relazione_binaria\n",k
                        +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     }
892     if(errore == 0 && termini_uguali_dopo ==
        termini_uguali_prima)
893         termini_diversi++;
894
895     termini_uguali_prima = termini_uguali_dopo;
896     i++;
897 }
898 if(errore == 0 && (termini_diversi == (verifica.
        dimensione - termini_uguali_prima)){
899     printf("\n_La_relazione_binaria_e'_una_funzione\n");
900     check_biiettivita(verifica);
901 }
902 else
903     printf("\n_La_relazione_binaria_non_e'_una_funzione\
        n");
904 }
905
906 printf("\n\n....._Controllo_Funzione_Terminato...\n\n
        n\n\n");
907
908 }
909

```

```

910  /*****FUNZIONE PER IL CHECK DELL'INIETTIVITA
      *****/
911
912  int check_iniettivita(struct relBin verifica){
913
914      int i;
915      int k;
916      int termini_diversi;
917      int termini_uguali_prima;
918      int termini_uguali_dopo;
919      int errore;
920      int iniettivita;
921
922      iniettivita = 0;
923
924      if(verifica.controllo == 1){
925
926          i=0;
927          errore=0;
928          termini_diversi=0;
929          termini_uguali_dopo=0;
930          termini_uguali_prima=0;
931
932          while(i < verifica.dimensione){
933
934              k=verifica.dimensione-1;
935              termini_uguali_dopo=termini_uguali_prima;
936              while(k > i){
937
938                  if(verifica.secondo_termine[i] == verifica.
                      secondo_termine[k]){
939
940                      if(verifica.primo_termine[i] != verifica.
                          primo_termine[k]){
941
942                          errore=1;
943                          printf("\n_Nel_%d_elemento_c'e'_un_errore_
                              che_impedisce_alla_funzione\n",k+1);
944                          printf("_di_essere_iniettiva\n");
945                          k=i;
946                          i=verifica.dimensione;
947                      }
948                      if(verifica.primo_termine[i] == verifica.
                          primo_termine[k])

```

```

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

```

```

987         printf(" _di_essere_iniettiva\n");
988         k=i;
989         i=verifica.dimensione;
990     }
991     if((strcmp(verifica.prima_stringa[i],verifica.
        prima_stringa[k])) == 0)
992         termini_uguali_dopo++;
993 }
994
995     k--;
996 }
997     if(errore == 0 && termini_uguali_dopo ==
        termini_uguali_prima)
998         termini_diversi++;
999
1000     termini_uguali_prima = termini_uguali_dopo;
1001     i++;
1002 }
1003     if(errore == 0 && (termini_diversi == (verifica.
        dimensione - termini_uguali_prima))){
1004         printf("\n_La_funzione_e'_iniettiva");
1005         iniettivita = 1;
1006     }
1007     else
1008         printf("\n_La_funzione_non_e'_iniettiva");
1009 }
1010
1011 return(iniettivita);
1012 }
1013
1014 /******FUNZIONE PER IL CHECK DELLA
SURIETTIVITA'******/
1015
1016 int check_suriettivita(struct relBin verifica){
1017
1018     /****** La suriettivit sempre verificata in quanto
il dominio e il codominio ******/
1019     /** sono entrambi i rispettivi x,y acquisiti, quindi
non ho elementi y non associati a x **/
1020     int suriettivita;
1021
1022     suriettivita = 1;
1023     return(suriettivita);
1024 }

```

```

1025
1026  /******FUNZIONE PER IL CHECK DELLA
      BIIETTIVITA'******/
1027
1028  void check_biiettivita(struct relBin verifica){
1029
1030      int    surriettivita ,
1031            iniettivita ;
1032
1033      surriettivita = check_suriettivita(verifica);
1034      iniettivita = check_iniettivita(verifica);
1035
1036
1037      if( surriettivita == 1 && iniettivita == 1)
1038          printf("\nla funzione e' biiettiva");
1039      else
1040          printf("\nla funzione non e' biiettiva");
1041      return;
1042  }
1043
1044
1045  int check_antisimmetria(struct relBin verifica){
1046
1047      int i ,
1048          j ,
1049          riscontro ,
1050          antisimmetria ;
1051
1052      antisimmetria = 1;
1053
1054
1055      i = 0;
1056      j = 0;
1057      riscontro = 0;
1058
1059      /* Check della antisimmetria per numeri*/
1060
1061      if(verifica.controllo == 1){
1062
1063          while( i < verifica.dimensione){
1064
1065              j = 0;
1066              while( j < verifica.dimensione){
1067

```

```

1068         if( verifica.primo_termine[i] == verifica .
           secondo_termine[j])
1069         if( verifica.primo_termine[j] == verifica .
           secondo_termine[i])
1070         if( verifica.primo_termine[i] == verifica .
           primo_termine[j])
1071             riscontro++;
1072         j++;
1073     }
1074
1075     if(riscontro == 0){
1076         j = verifica.dimensione;
1077         i = verifica.dimensione;
1078         antisimmetria = 0;
1079     }
1080     riscontro = 0;
1081     i++;
1082 }
1083
1084 }
1085
1086 /* Check della antisimmetria per stringhe*/
1087
1088 if(verifica.controllo == 2){
1089
1090     while( i < verifica.dimensione){
1091
1092         j = 0;
1093         while( j < verifica.dimensione){
1094
1095             if(strcmp( verifica.prima_stringa[i], verifica .
              seconda_stringa[j]) == 0 )
1096             if(strcmp( verifica.prima_stringa[j], verifica
              .seconda_stringa[i]) == 0 )
1097             if(strcmp( verifica.prima_stringa[j],
              verifica.prima_stringa[i]) == 0 )
1098                 riscontro++;
1099
1100             j++;
1101         }
1102
1103         if(riscontro == 0){
1104             j = verifica.dimensione;
1105             i = verifica.dimensione;

```



```

1106         antisimmetria = 0;
1107     }
1108     riscontro = 0;
1109     i++;
1110 }
1111
1112 }
1113
1114 /****** Controllo se la simmetria stata verificata
******/
1115
1116     if(antisimmetria == 1)
1117         printf(" _ _ _ _e' _antisimmetrica\n");
1118     else
1119         printf(" _non _e' _antisimmetrica\n");
1120
1121 /****** Fine controllo simmetria ******/
1122
1123     return(antisimmetria);
1124 }

```

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      scan = 0;
9
10     printf("\n_Programma_per_effettuare_i_Test_sulla_
        libreria\n");
11
12
13     printf("\n\n_Digita_il_numero_corrispondente_all_
        azione_che_si_vuole_svolgere\n");
14     printf("\n1)_Test_Acquisizione\n2)_Esci\n");
15
16
17     while((scelta < 1) || (scelta > 2) || scan != 1){
18         printf("\n_scelta:_");
19         fflush(stdin);
20         scan = scanf("%d",&scelta);
21     }
22     if(scelta == 1)
23         RelazioneBinaria = acquisizione(RelazioneBinaria);
24     if(scelta == 2){
25         printf("\n\n.....Test_terminati.....\n\n");
26         return(0);
27     }
28     scelta = -1;
29     while(scelta != 7){
30         printf("\n\n_Digita_il_numero_corrispondente_all_
        azione_che_si_vuole_svolgere\n");
31         printf("\n1)_Test_Acquisizione\n2)_Test_Stampa\n
        3)_Test_verifica_ordine_parziale\n4)_Test_
        verifica_ordine_totale");
32         printf("\n5)_Test_verifica_relazione_d'equivalenza\
        n_6)_Test_funzione\n7)_Esci\n");
33         scelta = -1;
34         while((scelta < 1) || (scelta > 7) || scan != 1){
35             printf("\n_scelta:_");
36             fflush(stdin);
```

```

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

```

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:
```

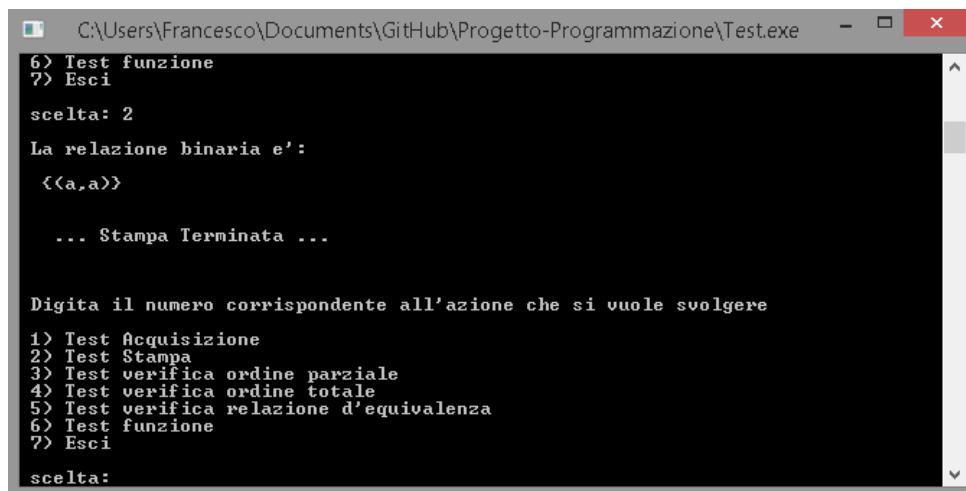
5.6 Test 6:

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

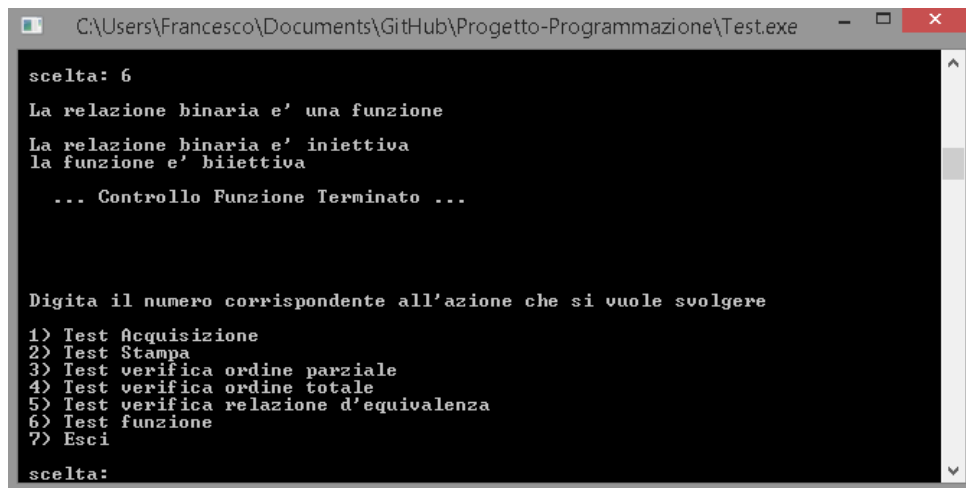
La relazione binaria e':

<<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:
```

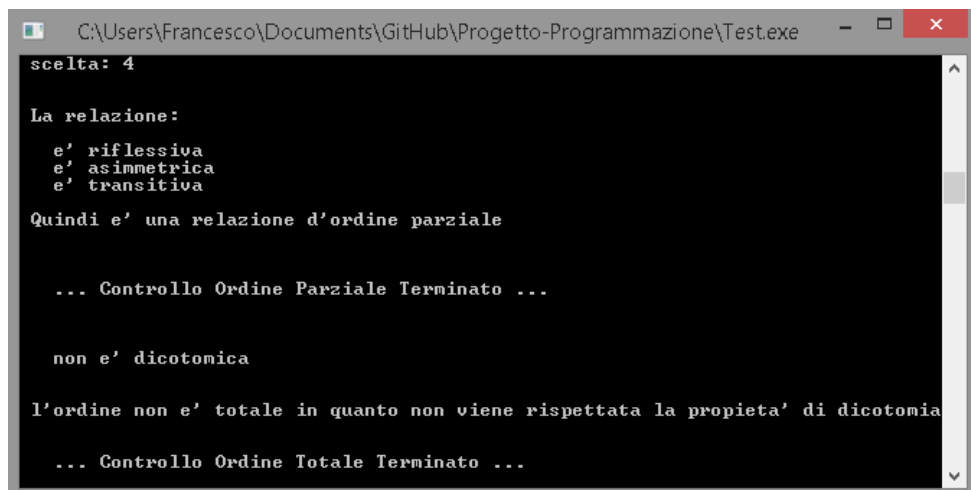
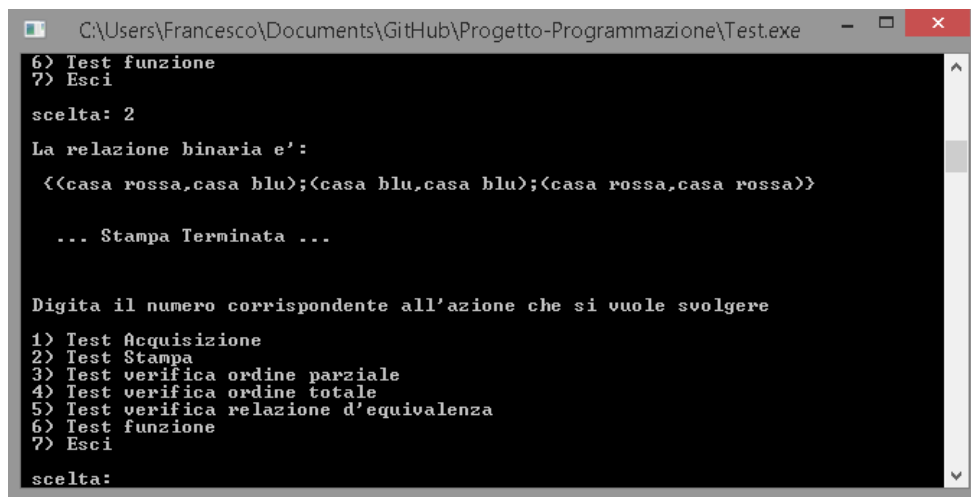
5.7 Test 7:

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.



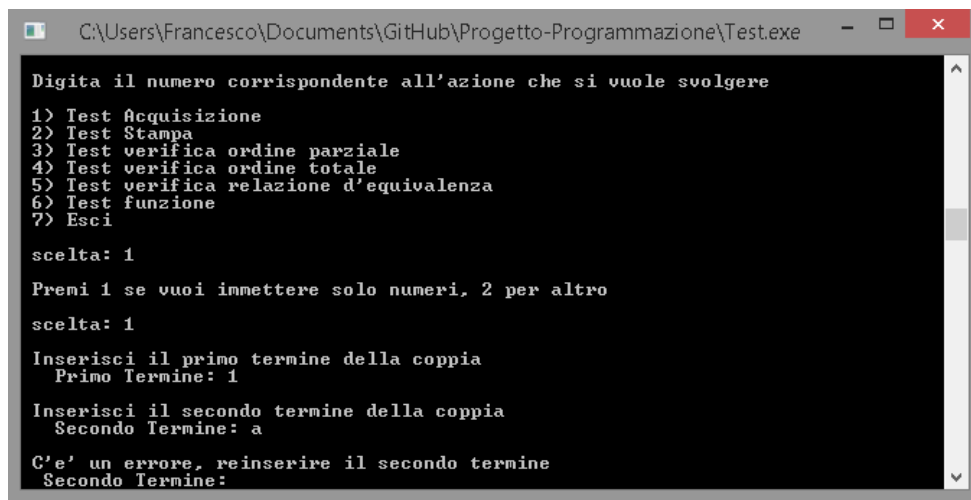
5.8 Test 8:

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:

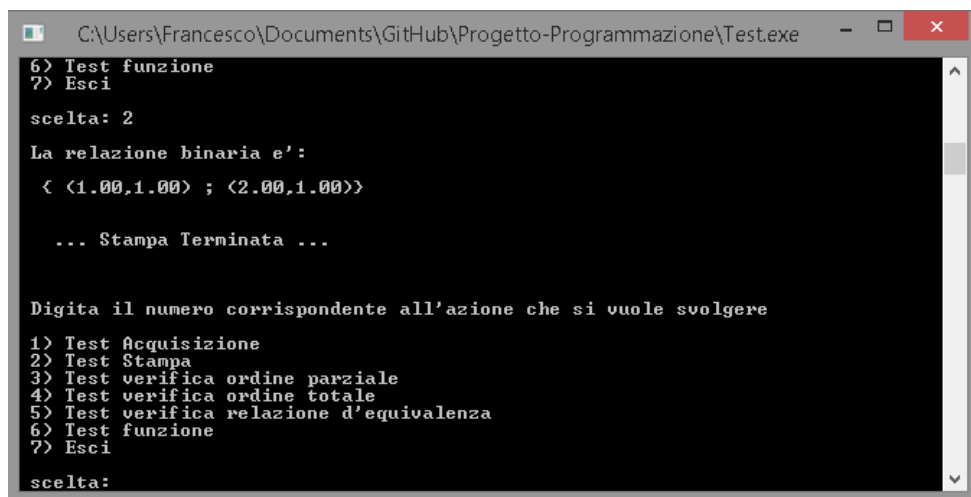
Inputs:(1,1)(2,1)

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



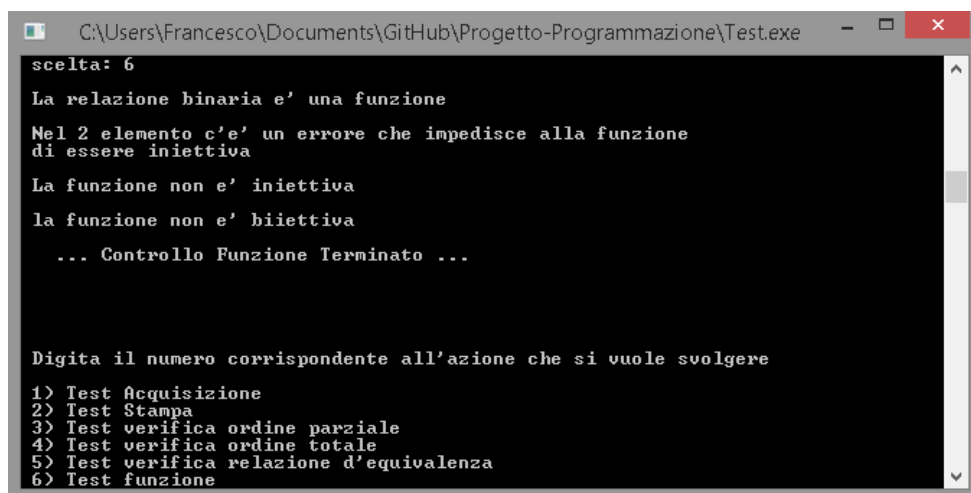
```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

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
?) 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
?) Esci
```

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:

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

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

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:

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