

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

<b>1</b>	<b>Specifica del Problema</b>	<b>1</b>
<b>2</b>	<b>Analisi del Problema</b>	<b>2</b>
2.1	Input . . . . .	2
2.2	Output . . . . .	2
<b>3</b>	<b>Progettazione dell' Algoritmo</b>	<b>3</b>
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
<b>4</b>	<b>Implementazione dell' algoritmo</b>	<b>10</b>
4.1	Libreria . . . . .	10
4.2	Test . . . . .	40
4.3	Makefile . . . . .	42
<b>5</b>	<b>Testing del programma</b>	<b>43</b>
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
<b>6</b>	<b>Verica del programma</b>	<b>53</b>

## 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 /*DICHIO LA 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     riflessivita = check_riflessivita (verifica);
198     antisimmetria = check_antisimmetria (verifica);
199     transitivita = check_transitivita (verifica);
200
201     /****** Controllo se rispetta le propriet per
202                essere una relazione d'ordine parziale******/
203     if (transitivita == 1 && antisimmetria == 1 &&
204         riflessivita == 1){
205         parziale = 1;
206         printf ("\n\nQuindi e' una relazione d'ordine _
207                parziale\n\n");
208     }
209     else{
210         printf ("\n\nNon e' una relazione d'ordine _
211                in quanto non rispetta tutte le propieta '\n");
212         parziale = 0;
213     }
214     if (transitivita == 0)
215     printf ("\n\nmanca la propieta 'di transitivita '\n");
216     ;
217     if (antisimmetria == 0)
218     printf ("\n\nmanca la propieta 'di antisimmetria\n");
219     ;
220     if (riflessivita == 0)
221     printf ("\n\nmanca la propieta 'di riflessivita '\n");
222     ;
223     /****** Fine controllo Ordine Parziale
224                ******/
225
226

```

```

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 TRANSITIVIT 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 TRANSITIVITA 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 a,b,c,d;
603     int numero_elementi;
604     int dicotomia = 0;
605     int dimensione;
606     int riscontro;
607     int secondo_riscontro;
608     a=0;

```

```

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

```

```

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

```

```

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

```

```

725     secondo_riscontro = 0;
726
727     while (d >= 0){
728         if ( (strcmp (verifica.prima_stringa[a],
729                     verifica.prima_stringa[d])) == 0)
730             secondo_riscontro = 1;
731             d--;
732     }
733     if (secondo_riscontro != 1){
734         if ( (strcmp (verifica.prima_stringa[a],
735                     verifica.seconda_stringa[a])) == 0)
736             numero_elementi++;
737     }
738     a++;
739     c = numero_elementi;
740
741     /****** Conto quanti dovrebbero essere gli
742     elementi per avere la dicotomia *****/
743
744     while (numero_elementi > 0){
745         numero_elementi--;
746         c = c + numero_elementi;
747     }
748 }
749
750 }
751
752 /****** Verifico se la dicotomia verificata
753 ******/
754
755 if (dimensione == c)
756     dicotomia = 1;
757
758 if (dicotomia == 1 )
759     printf ("L' e 'dicotomica\n\n");
760
761 else
762     printf ("L' non e 'dicotomica\n\n");
763
764 /****** Fine verifica dicotomia
765 ******/

```

```

764
765     return (dicotomia);
766 }
767
768 /*Funzione di verifica dell'ordine totale*/
769
770
771 void ordine_totale (rel_bin verifica){
772
773     int parziale ,
774         dicotomia;
775
776     dicotomia=2;
777     parziale = ordine_parziale (verifica);
778     if (parziale == 1)
779         dicotomia = check_dicotomia (verifica);
780
781     if (parziale == 0)
782         printf ("\n\nl'ordine non e' totale in quanto non e'
783             'nemmeno parziale");
784
785     if (dicotomia == 0)
786         printf ("\n\nl'ordine non e' totale in quanto non
787             viene rispettata la proprieta' di dicotomia");
788
789     if (dicotomia == 1 && parziale == 1)
790         printf ("\n\nQuindi e' una relazione d'ordine totale
791             ");
792
793     printf ("\n\n... Controllo Ordine Totale _
794         Terminato...\n\n\n");
795 }
796
797 /*Funzione che stabilisce se e'una relazione di
798     equivalenza o meno*/
799
800
801 void relazione_equivalenza (rel_bin verifica){
802
803     int riflessivita;
804     int simmetria;
805     int transitivita;
806
807     riflessivita = check_riflessivita (verifica);
808     simmetria = check_simmetria (verifica);

```



```

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

```

```

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

```

```

874     termini_uguali_dopo=0;
875     termini_uguali_prima=0;
876     while (i < verifica.dimensione){
877         k=verifica.dimensione-1;
878         termini_uguali_dopo=termini_uguali_prima;
879         while (k > i){
880             if ( (strcmp (verifica.prima_stringa[i],verifica
881                 .prima_stringa[k])) == 0){
882                 if ( (strcmp (verifica.seconda_stringa[i],
883                     verifica.seconda_stringa[k])) != 0){
884                     errore=1;
885                     printf ("\n_Nel_%d_elemento_c'e_un_errore_
886                         che_impedisce_alla_relazione_binaria\n",k
887                             +1);
888                     printf ("_di_essere_una_funzione\n");
889                     k=i;
890                     i=verifica.dimensione;
891                 }
892             }
893             else
894                 termini_uguali_dopo++;
895         }
896         k--;
897     }
898     if (errore == 0 && termini_uguali_dopo ==
899         termini_uguali_prima)
900         termini_diversi++;
901     termini_uguali_prima = termini_uguali_dopo;
902     i++;
903 }
904 if (errore == 0 && (termini_diversi == (verifica.
905     dimensione - termini_uguali_prima))){
906     printf ("\n_La_relazione_binaria_e_una_funzione\n");
907     check_biiettivita (verifica);
908 }
909 else
910     printf ("\n_La_relazione_binaria_non_e_una_funzione\
911         n");
912 }
913 }
914 printf ("\n\n.....Controllo_Funzione_Terminato...\n
915     \n\n\n");
916 }
917 }

```

```

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

```

```

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

```

```

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

```

```

1024 return (suriettivita);
1025 }
1026
1027 /******FUNZIONE PER IL CHECK DELLA
BIETTIVITA '******/
1028
1029 void check_biettivita (rel_bin verifica){
1030
1031     int    surriettivita ,
1032           iniettivita;
1033
1034     surriettivita = check_suriettivita (verifica);
1035     iniettivita = check_iniettivita (verifica);
1036
1037
1038     if ( surriettivita == 1 && iniettivita == 1)
1039         printf ("\nla_funzione_e'biettiva");
1040     else
1041         printf ("\nla_funzione_non_e'biettiva");
1042     return;
1043 }
1044
1045
1046 int check_antisimmetria (rel_bin verifica){
1047
1048     int i,
1049         j,
1050         riscontro ,
1051         antisimmetria;
1052
1053     antisimmetria = 1;
1054
1055
1056     i = 0;
1057     j = 0;
1058     riscontro = 0;
1059
1060 /*Check della antisimmetria per numeri*/
1061
1062     if (verifica.controllo == 1){
1063
1064         while ( i < verifica.dimensione){
1065
1066             j = 0;

```

```

1067     while ( j < verifica.dimensione){
1068
1069         if (verifica.primo_termine[i] == verifica.
            secondo_termine[j])
1070         if (verifica.primo_termine[j] == verifica.
            secondo_termine[i])
1071         if (verifica.primo_termine[i] == verifica.
            primo_termine[j])
1072             riscontro++;
1073         j++;
1074     }
1075
1076     if (riscontro == 0){
1077         j = verifica.dimensione;
1078         i = verifica.dimensione;
1079         antisimmetria = 0;
1080     }
1081     riscontro = 0;
1082     i++;
1083 }
1084
1085 }
1086
1087 /* Check della antisimmetria per stringhe*/
1088
1089 if (verifica.controllo == 2){
1090
1091     while ( i < verifica.dimensione){
1092
1093         j = 0;
1094         while ( j < verifica.dimensione){
1095
1096             if (strcmp (verifica.prima_stringa[i], verifica
                .seconda_stringa[j]) == 0 )
1097             if (strcmp (verifica.prima_stringa[j],
                verifica.seconda_stringa[i]) == 0 )
1098             if (strcmp (verifica.prima_stringa[j],
                verifica.prima_stringa[i]) == 0 )
1099                 riscontro++;
1100
1101             j++;
1102         }
1103
1104         if (riscontro == 0){

```



```

1105         j = verifica.dimensione;
1106         i = verifica.dimensione;
1107         antisimmetria = 0;
1108     }
1109     riscontro = 0;
1110     i++;
1111 }
1112
1113 }
1114
1115 /****** Controllo se la simmetria stata verificata
******/
1116
1117     if (antisimmetria == 1)
1118         printf ("_e'antisimmetrica\n");
1119     else
1120         printf ("_non_e'antisimmetrica\n");
1121
1122 /****** Fine controllo simmetria ******/
1123
1124     return (antisimmetria);
1125 }

```

## 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:
```

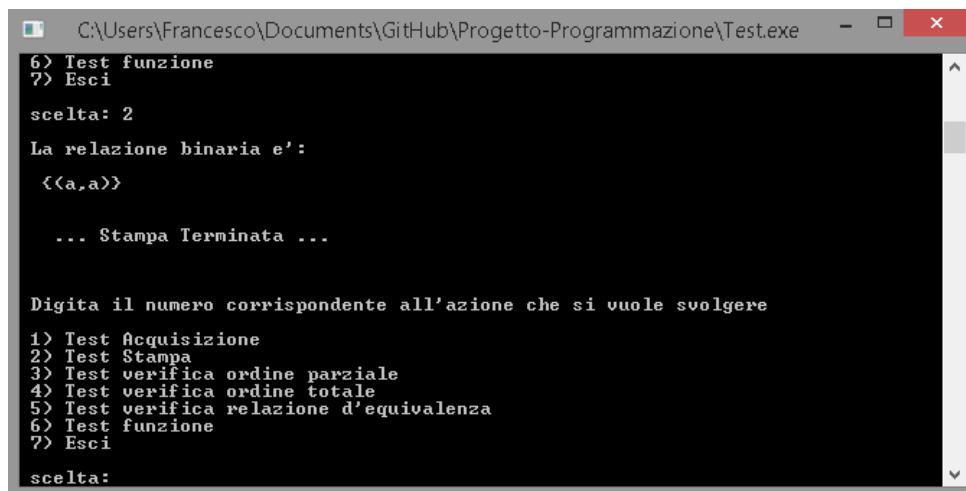
## 5.6 Test 6:

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

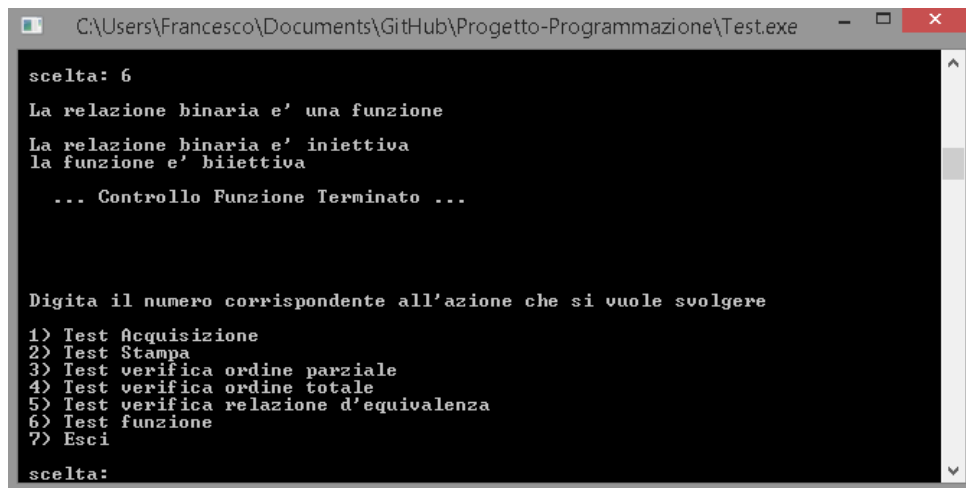
La relazione binaria e':

<<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:
```

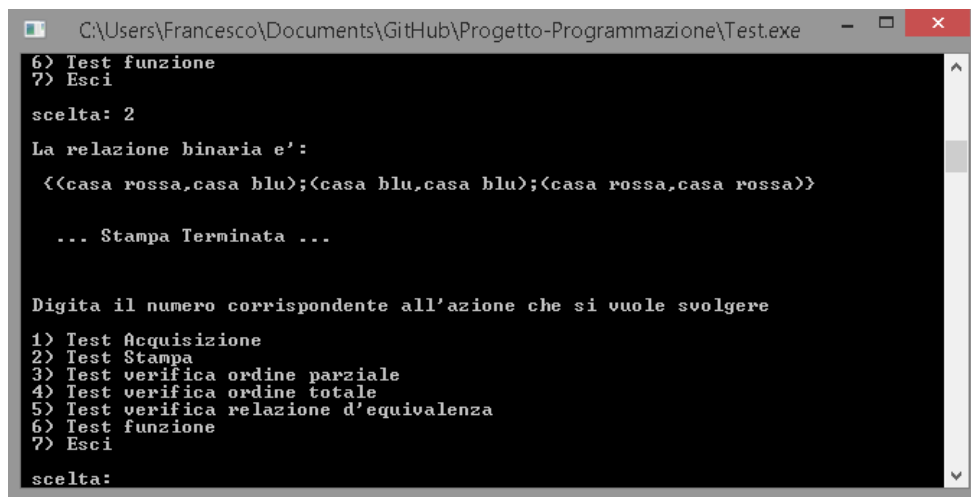
## 5.7 Test 7:

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
?> Esci

scelta: 2

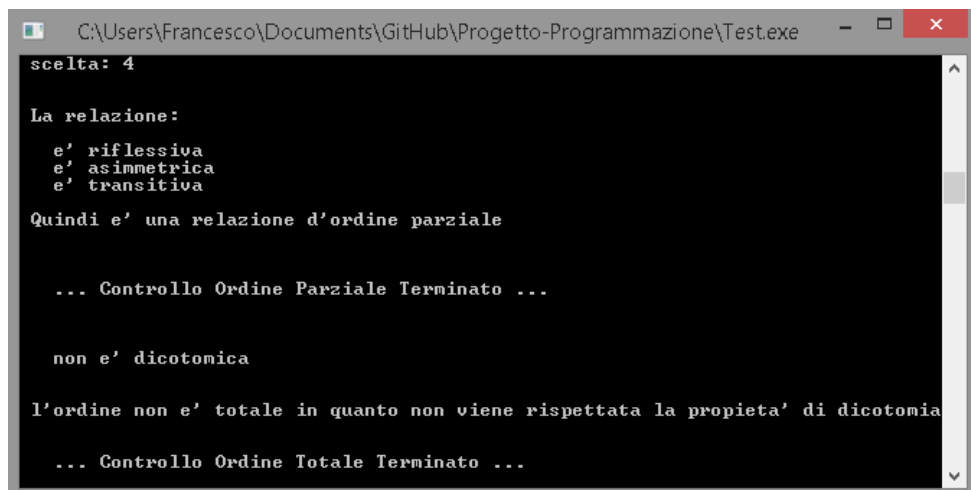
La relazione binaria e':

<(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
?> 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 ...
```

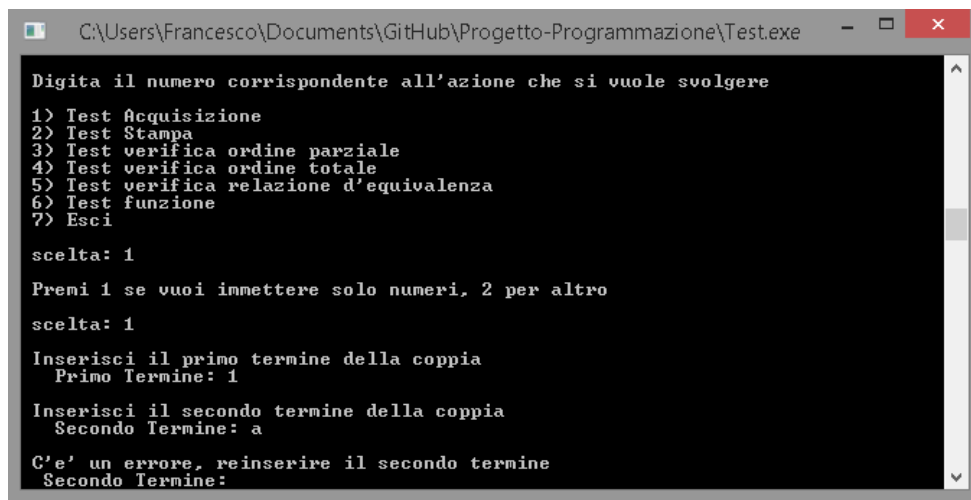
## 5.8 Test 8:

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:

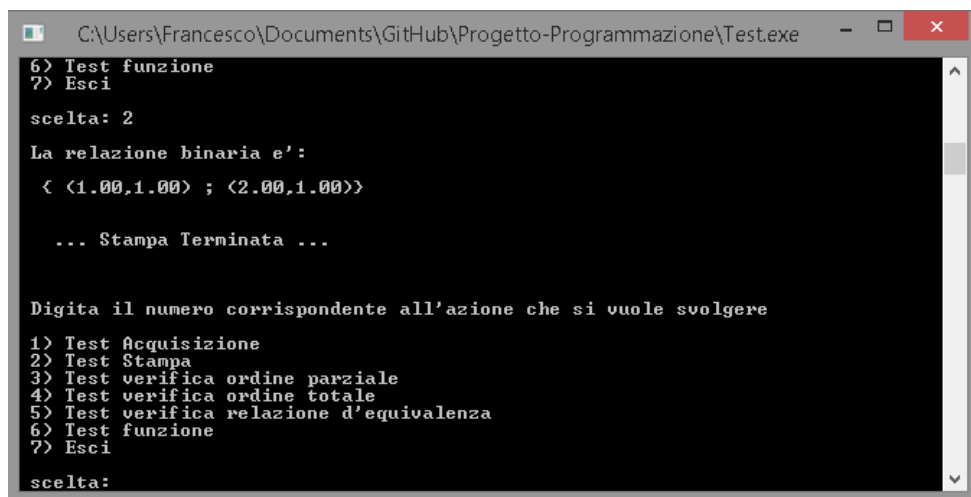
Inputs:(1,1)(2,1)

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

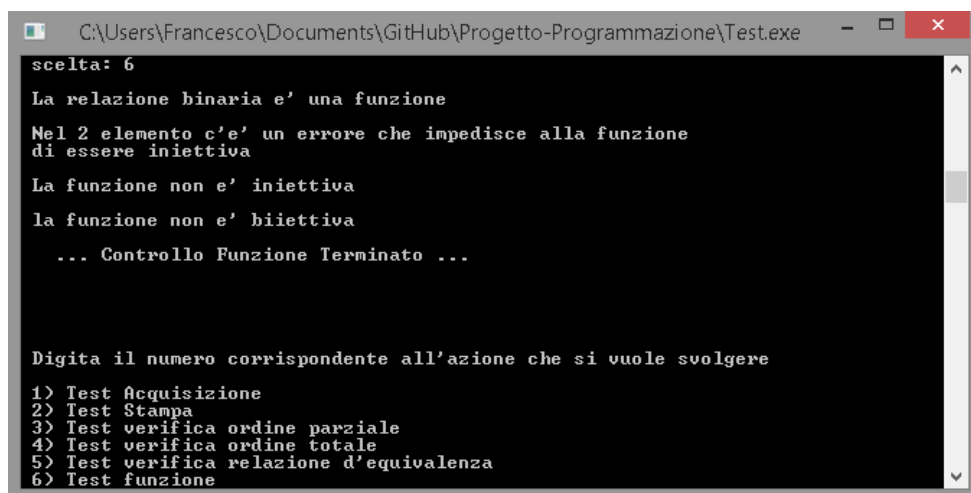
scelta: 2

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:

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