

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 26, 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	Scelte di Progetto	5
3.3	Funzioni per l'acquisizione	6
3.4	Funzioni per la verifica delle proprietà:	6
3.5	Funzioni principali:	7
3.6	Input	8
3.7	Output - Acquisizione	9
3.8	Output - stampa	9
3.9	Output - ordine_parziale	9
3.10	Output - ordine_totale	9
3.11	Output - relazione_equivalenza	10
3.12	Output - controllo_funzione	10
4	Implementazione dell'Algoritmo	11
4.1	Libreria	11
4.2	Test	12
4.3	Makefile	15
5	Testing del programma	16
5.1	Test 1:	16
5.2	Test 2:	17
5.3	Test 3:	18
5.4	Test 4:	19
5.5	Test 5:	20
5.6	Test 6:	21
5.7	Test 7:	22
5.8	Test 8:	23
5.9	Test 9:	24
5.10	Test 10:	25
6	Verica del programma	26

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) ; (a_1,b_1) ; (a_2,b_2) ; ... formata da un numero non precedentemente definito di coppie che viene acquisita da tastiera;
2. Come input per le altre 5 funzioni abbiamo una relazione binaria.

2.2 Output

1. Il primo problema (problema dell'acquisizione) restituisce una relazione binaria del tipo (a,b) ; (a_1,b_1) ; (a_2,b_2) ; ... formata da un numero non precedentemente definito.
2. Il secondo problema (problema della stampa) stampa a video la relazione binaria che viene dato in pasto alla funzione;
3. Il terzo problema (problema della verifica dell'ordine parziale) ci richiede di controllare se la relazione binaria data in pasto alla funzione è una relazione d'ordine parziale, e nel caso in cui non lo sia di andare a stampare a video le varie proprietà che non rispetta;
4. Il quarto problema (problema della verifica dell'ordine totale) ci richiede di controllare se la relazione binaria data in pasto al programma è una relazione d'ordine totale, e nel caso in cui non lo sia di andare a stampare a video le varie proprietà che non rispetta;
5. Il quinto problema (problema della verifica dell'ordine di equivalenza) ci richiede di controllare se la relazione binaria data in pasto al programma è una relazione di equivalenza, e nel caso in cui non lo sia di andare a stampare a video le varie proprietà che non rispetta;
6. Il sesto problema (problema della verifica della funzione) ci richiede di controllare se la relazione binaria data in pasto al programma è una funzione, e nel caso in cui non lo sia di andare a stampare a video le varie proprietà che non rispetta, mentre nel caso in cui sia una funzione di controllare se tale funzione rispetti le proprietà di suriettività e iniettività, stampando a video se la funzione è suriettiva, iniettiva o biiettiva;

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: una relazione binaria è un sottoinsieme del prodotto cartesiano di due insiemi (i quali potrebbero pure coincidere, ma ciò non è garantito).

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 con ogni altro elemento presente).

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 .

Un sottoinsieme f di $A \times B$ è una funzione se ad ogni elemento di A viene associato da f al più un elemento di B , dando luogo alla distinzione tra funzioni totali e parziali (a seconda che tutti o solo alcuni degli elementi di A abbiano un corrispondente in B) e lasciando non specificato se tutti gli elementi di B siano i corrispondenti di qualche elemento di A oppure no.

Concetto di Iniettività: ad ogni elemento del codominio corrisponde al più un elemento del dominio, cioè elementi diversi del dominio vengono trasformati in elementi diversi del codominio.

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

3.2 Scelte di Progetto

- Una relazione binaria prende in considerazione due elementi, questi due elementi si potrebbero vedere come due variabili distinte che poi andranno a far parte della stessa struttura, per questo riteniamo opportuno creare una struttura dati che inglobi entrambi gli elementi.
- I due termini potrebbero essere numerici, ma non è detto, quindi per completezza riteniamo opportuno far scegliere all'utente se inserire elementi di tipo numerico, o altro (simboli, lettere etc.) a seconda delle sue necessità.
- A priori, prendendo come input una relazione binaria, non possiamo sapere se tutti gli elementi del primo insieme sono in relazione con almeno un elemento del secondo insieme o se tutti gli elementi del secondo insieme fanno parte di una coppia ordinata, quindi è opportuno chiedere all'utente se ci sono elementi isolati che non fanno parte di nessuna coppia ordinata.

Breve lista delle funzioni da utilizzare:

3.3 Funzioni per l'acquisizione

acquisizione: per acquisire la relazione.

3.4 Funzioni per la verifica delle proprietà:

controllo_iniettività: serve a controllare se l'iniettività è rispettata o meno.

controllo_transitività: serve a controllare se la transitività viene rispettata o meno.

controllo_antisimmetria: serve a controllare se l'antisimmetria viene rispettata o meno.

controllo_simmetria: serve a controllare se la simmetria viene rispettata o meno.

controllo_riflessività: serve a controllare se la riflessività viene rispettata o meno.

controllo_dicotomia: serve a verificare se la dicotomia viene rispettata o meno.

controllo_suriettività: serve a verificare se la suriettività viene rispettata o meno.

3.5 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 `controllo_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 a schermo quali proprietà non vengono rispettate).

`controllo_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).

3.6 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 può farlo e ciò 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 può 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.

3.7 Output - Acquisizione

Durante l'acquisizione avremo diversi output video 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 conterrà i dati inseriti dall'utente. Abbiamo scelto di fare ciò perchè non essendo permesso l'utilizzo di variabili globali, il modo più 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 proprietà.

3.8 Output - stampa

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

3.9 Output - ordine_parziale

La funzione ordine_parziale avrà come output la stampa a video del risultato della verifica delle proprietà di riflessività antisimmetria e transitività. Nel caso in cui siano tutte verificate si stamperà che la relazione è una relazione di ordine parziale, mentre nel caso in cui non siano verificate si stamperà che non lo è e il perchè (cioè quale(o quali) proprietà non è verificata(o non sono verificate)).

3.10 Output - ordine_totale

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

3.11 Output - relazione_equivalenza

La funzione `relazione_equivalenza` avrà come output la stampa a video del risultato della verifica delle proprietà di riflessività simmetria e transitività e nel caso in cui siano tutte positive si stamperà che la relazione è una relazione di equivalenza, mentre nel caso in cui qualcosa non sia verificato si stamperà ciò che impedisce alla relazione di essere una relazione d'equivalenza.

3.12 Output - controllo_funzione

La funzione `controllo_funzione` avrà come output la stampa a video della verifica della proprietà che rende la relazione binaria una funzione, e in caso lo sia, se questa è sia suriettiva e iniettiva, e in caso sia entrambe si stamperà che la relazione binaria oltre ad essere una funzione è una funzione biiettiva.

4 Implementazione dell'Algoritmo

4.1 Libreria (file .h)

```
1
2  /* STRUTTURA relBin */
3  /* Creo una struttura dove salvare le coppie*/
4  /* appartenenti alla Relazione */
5
6  typedef struct relBin
7  {
8  /****** Coppia Numerica *****/
9      double    *primo_termine ,
10               *secondo_termine;
11
12 /****** Coppia Qualsiasi******/
13      char    **prima_stringa ,
14              **seconda_stringa;
15
16 /***** Variabili per salvare se ho acquisito una*/
17 /* coppia numerica o no e il numero delle coppie */
18      int    controllo ,
19             dimensione ,
20             insieme_a ,
21             insieme_b;
22 } rel_bin;
23
24 extern rel_bin acquisizione (rel_bin);
25 extern int controllo_simmetria (rel_bin);
26 extern int controllo_riflessivita (rel_bin);
27 extern int controllo_transitivita (rel_bin);
28 extern int controllo_suriettivita (rel_bin);
29 extern void controllo_biiettivita (rel_bin);
30 extern int controllo_antisimmetria (rel_bin);
31 extern void controllo_funzione (rel_bin);
32 extern void relazione_equivalenza (rel_bin);
33 extern void ordine_totale (rel_bin);
34 extern int ordine_parziale (rel_bin);
35 extern void stampa (rel_bin);
```

4.2 Libreria (file .c)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "Progetto.h"
5
6
7 rel_bin acquisizione (rel_bin);
8 int controllo_simmetria (rel_bin);
9 int controllo_riflessivita (rel_bin);
10 int controllo_transitivita (rel_bin);
11 int controllo_suriettivita (rel_bin);
12 void controllo_biiettivita (rel_bin);
13 int controllo_antisimmetria (rel_bin);
14 void controllo_funzione (rel_bin);
15 void relazione_equivalenza (rel_bin);
16 void ordine_totale (rel_bin);
17 int ordine_parziale (rel_bin);
18 void stampa (rel_bin);
19
20
21 /******Funzione di acquisizione
22 ******/
23 rel_bin acquisizione (rel_bin relazione)
24 {
25     int acquisizione_finita = 0,
26         risultato_lettura = 0,
27         i;
28
29     char temporaneo,
30         carattere_non_letto;
31
32     relazione.dimensione = 0;
33     relazione.primo_termine = (double *) malloc (2);
34     relazione.secondo_termine = (double *) malloc (2);
35     relazione.prima_stringa = (char **) malloc (2);
36     relazione.seconda_stringa = (char **) malloc (2);
37
38     do
39     {
40         printf ("\n_Premi_1_se_vuoi_immettere_solo_
           numeri,_2_per_altro\n");
```

```

41     printf ("\n_scelta:_");
42
43     risultato_lettura = scanf ("%d",
44                                 &relazione.
45                                 controllo);
46     if (relazione.controllo < 1 || relazione.
47         controllo > 2 || risultato_lettura != 1)
48         do
49             carattere_non_letto = getchar();
50             while (carattere_non_letto != '\n');
51     }
52     while (relazione.controllo < 1 || relazione.
53         controllo > 2 || risultato_lettura != 1);
54
55     /** imposto di nuovo risultato_lettura a 0 **/
56     risultato_lettura=0;
57
58     /*Acquisizione Numerica*/
59
60     if (relazione.controllo == 1)
61     {
62         while (acquisizione_finita == 0)
63         {
64             relazione.dimensione++;
65             acquisizione_finita = 2;
66
67             /*Acquisisco il primo termine della coppia
68             */
69
70             printf ("\n_Inserisci_il_primo_termine_
71                 della_coppia_\n");
72             relazione.primo_termine = (double *)
73                 realloc (relazione.primo_termine, (
74                     relazione.dimensione+1) * sizeof (
75                     double));
76             risultato_lettura = 0;
77             /*controllo del primo termine della coppia
78             */
79
80             do
81             {
82                 printf ("_Primo_Termine:_");
83                 risultato_lettura = scanf ("%lf",

```

```

75                                     &relazione.
                                     primo_termine
                                     [
                                     relazione
                                     .
                                     dimensione
                                     - 1]);
76     if (risultato_lettura != 1)
77         do
78             carattere_non_letto = getchar
79                 ();
80             while (carattere_non_letto != '\n'
81                 );
82         if (risultato_lettura == 0)
83             printf ("\nC'e' un errore, _
84                 reinserire il primo termine\n");
85             ;
86     }
87     while (risultato_lettura != 1);
88
89     /*Acquisisco il secondo termine della
90     coppia*/
91     risultato_lettura = 0;
92     printf ("\nInserisci il secondo termine _
93     della coppia\n");
94     relazione.secondo_termine = (double *)
95         realloc (relazione.secondo_termine, (
96         relazione.dimensione+1) * sizeof (
97         double));
98
99     /*controllo del secondo termine della
100     coppia*/
101     do
102     {
103         printf (" _Secondo Termine:_");
104         risultato_lettura = scanf ("%lf",
105                                     &relazione.
106                                     secondo_termine
107                                     [
108                                     relazione
109                                     .
110                                     dimensione

```



```

98         - 1]);
99         if (risultato_lettura != 1)
100             do
101                 carattere_non_letto = getchar
102                     ();
103                 while (carattere_non_letto != '\n'
104                     );
105             if (risultato_lettura == 0)
106                 printf ("\nC'e' un errore, _
107                     reinserire il secondo termine\n
108                     ");
109         }
110     while (risultato_lettura != 1);
111     /*Chiedo all'utente se ci sono altre
112        coppie*/
113     do
114     {
115         printf ("\nVuoi acquisire un'altra _
116             coppia? immetti _1_ per _uscire, _0_ per
117             _continuare\n");
118         printf ("\nscelta: _");
119         risultato_lettura = scanf ("%d",
120             &
121             acquisizione_finita
122             );
123         if (acquisizione_finita < 0 ||
124             acquisizione_finita > 1 ||
125             risultato_lettura != 1)
126             do
127                 carattere_non_letto = getchar
128                     ();
129                 while (carattere_non_letto != '\n'
130                     );
131             }
132         while (acquisizione_finita < 0 ||
133             acquisizione_finita > 1 ||
134             risultato_lettura != 1);
135     }
136 }
137

```

```

126      /*imposto di nuovo risultato_lettura a 0*/
127      risultato_lettura = 0;
128
129      /*Acquisizione con stringhe*/
130      if (relazione.controllo == 2)
131      {
132          while (acquisizione_finita == 0)
133          {
134              i=0;
135              temporaneo = 'a';
136              relazione.dimensione++;
137              acquisizione_finita = 2;
138
139              relazione.prima_stringa = (char **)
                  realloc (relazione.prima_stringa, (
                  relazione.dimensione+1) * sizeof (char
                  *));
140
141              /*Acquisisco il primo termine della coppia
                  */
142              relazione.prima_stringa[relazione.
                  dimensione - 1] = (char *) malloc (2);
143              fflush(stdin);
144              printf ("\n Inserisci il primo termine _
                  della coppia _\n");
145              printf (" _ Primo Termine: _");
146              while (temporaneo != '\n')
147              {
148                  temporaneo = getc (stdin);
149                  relazione.prima_stringa [relazione.
                  dimensione - 1] = (char*) realloc (
                  relazione.prima_stringa [relazione.
                  dimensione - 1],
150                      (i+1) * sizeof (char*));
151                  relazione.prima_stringa [relazione.
                  dimensione - 1] [i] = temporaneo;
152                  i++;
153              }
154
155              /*Imposto ora il carattere di terminazione
                  a \0 dato che adesso \n*/
156
157              relazione.prima_stringa [relazione.
                  dimensione - 1] [i - 1] = '\0';

```

```

158
159      /*Acquisisco il secondo termine della
        coppia*/
160
161      printf ("\n_Inserisci il secondo termine_
        della coppia_\n");
162      printf (" _Secondo Termine:_");
163      relazione.seconda_stringa = (double **)
        realloc (relazione.seconda_stringa, (
        relazione.dimensione+1) * sizeof (
        double*));
164      relazione.seconda_stringa [relazione.
        dimensione - 1] = (char *) malloc (2);
165      fflush (stdin);
166      temporaneo='a';
167      i=0;
168      while (temporaneo != '\n')
169      {
170          temporaneo = getc (stdin);
171          relazione.seconda_stringa [relazione.
        dimensione - 1] = (char*) realloc (
        relazione.seconda_stringa [relazione
        .dimensione-1],
172              (i+1) * sizeof (char*));
173          relazione.seconda_stringa [relazione.
        dimensione - 1] [i] = temporaneo;
174          i++;
175      }
176
177      /*Imposto ora il carattere di terminazione
        a \0 dato che adesso \n*/
178      relazione.seconda_stringa [relazione.
        dimensione - 1] [i - 1] = '\0';
179
180      /*Chiedo all'utente se ci sono altre
        coppie*/
181
182      while (acquisizione_finita < 0 ||
        acquisizione_finita > 1 ||
        risultato_lettura != 1)
183      {
184
185          printf ("\n_Vuoi acquisire un'altra_
        coppia?_immetti_1_per_uscire, _0_per

```

```

        _continuare\n");
186         risultato_lettura = scanf ("%d",&
            acquisizione_finita);
187     }
188 }
189 }
190
191 relazione.insieme_b = -1;
192 risultato_lettura = 0;
193
194 printf ("\n_Ci_sono_elementi_del_secondo_insieme\n
        _che_non_fanno_parte_di_nessuna_coppia_ordinata
        ?");
195 printf ("\n_1_si_2_no\n\n_scelta:_");
196 while ((relazione.insieme_b < 0) || (relazione.
        insieme_b > 2) || risultato_lettura != 1)
197 {
198     fflush (stdin);
199     risultato_lettura = scanf("%d",&relazione.
        insieme_b);
200 }
201
202 relazione.insieme_a = -1;
203 risultato_lettura = 0;
204
205 printf ("\n_Ci_sono_elementi_del_primo_insieme\n
        _che_non_fanno_parte_di_nessuna_coppia_ordinata?
        ");
206 printf ("\n_1_si_2_no\n\n_scelta:_");
207 while ((relazione.insieme_a < 0) || (relazione.
        insieme_a > 2) || risultato_lettura != 1)
208 {
209     fflush (stdin);
210     risultato_lettura = scanf("%d",&relazione.
        insieme_a);
211
212 }
213
214 printf ("\n\n.....Acquisizione_Terminata....\n\n
        ");
215 return (relazione);
216 }
217

```

```

218  /*****FUNZIONE DI STAMPA
      *****/
219
220  void stampa (rel_bin stampa)
221  {
222
223      int i = 0;
224
225      printf ("\nLa relazione binaria e'");
226      printf ("\n\n{");
227
228      /*****Stampa per coppie numeriche *****/
229
230      if (stampa.controllo == 1)
231      {
232          while (i < stampa.dimensione)
233          {
234
235              printf ("(%.2lf,%.2lf)", stampa.
                  primo_termine[i], stampa.secondo_termine
                  [i]);
236              if (i+1 != stampa.dimensione)
237                  printf (" ";)
238              i++;
239          }
240      }
241
242      /*****Stampa per coppie non numeriche *****/
243      */
244
245      if (stampa.controllo == 2)
246      {
247          while (i < stampa.dimensione)
248          {
249              printf ("(%s,%s)", stampa.prima_stringa[i],
                  stampa.seconda_stringa[i]);
250              if (i+1 != stampa.dimensione)
251                  printf (" ");
252              i++;
253          }
254      }
255

```

```

256      /* ***** Fine Stampa ***** */
257      */
258      printf ("}\n");
259      printf ("\n\n... _Stampa_Terminata_...\n\n");
260
261  }
262
263  /* *****FUNZIONE DI VERIFICA DI RELAZIONI D
'ORDINE******/
264
265  int ordine_parziale (rel_bin verifica)
266  {
267
268      int riflessivita ,
269          transitivita ,
270          antisimmetria ,
271          parziale;
272
273      /*STAMPO LE PROPIETA 'DELLA RELAZIONE*/
274
275      printf ("\n\nLa relazione:\n\n");
276
277      /****** Chiamo le funzioni per poter stabilire
le propriet ******/
278
279      riflessivita = controllo_riflessivita (verifica);
280      antisimmetria = controllo_antisimmetria (verifica)
281      ;
282      transitivita = controllo_transitivita (verifica);
283
284      /****** Controllo se rispetta le propriet
per essere una relazione d'ordine parziale
******/
285
286      if (transitivita == 1 && antisimmetria == 1 &&
287          riflessivita == 1)
288      {
289          parziale = 1;
290          printf ("\n_Quindi_e'una_relazione_d'ordine_
291              parziale\n\n");
292      }
293      else
294      {

```

```

292
293         printf ("\nNon e' una relazione d'ordine
                parziale in quanto non rispetta tutte le
                proprieta'\n");
294     parziale = 0;
295 }
296 if (transitivita == 0)
297     printf ("\nmanca la proprieta' di transitivita
                '\n");
298 if (antisimmetria == 0)
299     printf ("\nmanca la proprieta' di antisimmetria
                '\n");
300 if (riflessivita == 0)
301     printf ("\nmanca la proprieta' di riflessivita
                '\n");
302 /****** Fine controllo Ordine Parziale
                *****/
303
304     printf ("\n\n... Controllo Ordine Parziale
                Terminato...\n\n\n");
305     return (parziale);
306 }
307
308
309 /******FUNZIONE PER CONTROLLARE LA RIFLESSIVITA
                *****/
310
311 int controllo_riflessivita (rel_bin verifica)
312 {
313
314     int i,
315         j,
316         k,
317         riscontro,
318         secondo_riscontro,
319         riflessivita;
320
321     riflessivita = 1;
322     i = 0;
323     j = 0;
324     k = 0;
325     riscontro = 0;
326     secondo_riscontro = 0;
327

```

```

328      /* Verifica riflessivit */
329
330      /* Definizione: una relazione per la quale esiste
          almeno un elemento che non e' in relazione con
          s stesso non soddisfa la definizione di
          riflessivit */
331
332      while ( (i < verifica.dimensione) && (k < verifica
          .dimensione))
333      {
334
335          /* Verifica riflessivit per numeri */
336
337          if (verifica.controllo == 1)
338          {
339              riscontro = 0;
340              secondo_riscontro = 0;
341              if (verifica.primo_termine[i] == verifica.
                  secondo_termine[i])
342                  riscontro++; /**** Controllo se c'
                               stato un riscontro a,a*** */
343              secondo_riscontro++;
344              if (riscontro != 0)
345              {
346                  i++;
347                  k++;
348              }
349              /**/
350              else
351              {
352                  j=0;
353                  riscontro = 0;
354                  secondo_riscontro = 0;
355
356                  /****** Controllo la
                               riflessivit per gli elementi del
                               primo insieme
                               *****/
357
358                  while (j < verifica.dimensione)
359                  {
360                      if (j == i)
361                          j++;
362                      else

```



```

363         {
364             if (verifica.primo_termine[i]
                == verifica.primo_termine[j]
                )
365                 if (verifica.primo_termine
                    [j] == verifica.
                    secondo_termine[j])
366                     riscontro++;
367
368             j++;
369         }
370     }
371
372     j = 0;
373
374     /****** Controllo la
        riflessivit per gli elementi del
        secondo insieme
        *****/
375
376     while (j < verifica.dimensione)
377     {
378         if (j == k)
379             j++;
380         else
381         {
382             if (verifica.secondo_termine[k]
                == verifica.
                secondo_termine[j])
383                 if (verifica.primo_termine
                    [j] == verifica.
                    secondo_termine[j])
384                     secondo_riscontro++;
385
386             j++;
387         }
388     }
389     if (riscontro != 0)
390         i++;
391
392     /**** Se non c' stato un riscontro di
        riflessivit esco e imposto la
        riflessivit a 0 *****/
393

```

```

394         else
395         {
396             i=verifica.dimensione;
397             riflessivita = 0;
398         }
399
400         if (secondo_riscontro != 0)
401             k++;
402
403         else
404         {
405             k=verifica.dimensione;
406             riflessivita = 0;
407         }
408     }
409
410 }
411
412 /****** VERIFICA RIFLESSIVIT PER
413 STRINGHE *****/
414
415 if (verifica.controllo == 2)
416 {
417     riscontro = 0;
418     secondo_riscontro = 0;
419     if (strcmp (verifica.prima_stringa[i],
420               verifica.seconda_stringa[i]) == 0)
421         riscontro++;
422     secondo_riscontro++;
423     if (riscontro != 0)
424     {
425         i++;
426         k++;
427     }
428
429     else
430     {
431         j=0;
432         riscontro = 0;
433         secondo_riscontro = 0;
434
435         /****** Controllo la
436         riflessivita per gli elementi del
437         primo insieme

```

```

434 *****/
435 while (j < verifica.dimensione)
436 {
437     if (j == i)
438         j++;
439     else
440     {
441         if (strcmp (verifica.
                     prima_stringa[i], verifica.
                     prima_stringa[j]) == 0)
442             if (strcmp (verifica.
                         prima_stringa[j],
                         verifica.
                         seconda_stringa[j]) ==
                         0)
443                 riscontro++;
444
445         j++;
446     }
447 }
448
449 j = 0;
450
451 /***** Controllo la
         riflessivit  per gli elementi del
         secondo insieme *****/
452
453 while (j < verifica.dimensione)
454 {
455     if (j == k)
456         j++;
457     else
458     {
459         if (strcmp (verifica.
                     seconda_stringa[k], verifica.
                     .seconda_stringa[j]) == 0)
460             if (strcmp (verifica.
                         prima_stringa[j],
                         verifica.
                         seconda_stringa[j]) ==
                         0)
461                 secondo_riscontro++;

```

```

462
463             j++;
464         }
465     }
466     if (riscontro != 0)
467         i++;
468
469     else
470     {
471         i=verifica.dimensione;
472         riflessivita = 0;
473     }
474
475     if (secondo_riscontro != 0)
476         k++;
477
478     else
479     {
480         k=verifica.dimensione;
481         riflessivita = 0;
482     }
483 }
484
485 }
486
487 }
488
489 /****** Controllo se riflessiva
490 ******/
491
492 if (riflessivita == 1)
493     printf ("___e' riflessiva\n");
494 else
495     printf ("___non e' riflessiva\n");
496
497 /****** Fine riflessivita
498 ******/
499
500 return (riflessivita);
501 }
502

```

```

503  /***** FUNZIONE PER CONTROLLARE
      LA SIMMETRIA *****/
504
505  /***** Definizione: In matematica, una
      relazione binaria  $R$  in un insieme  $X$  **/
506  /***** simmetrica se e solo se, presi due
      elementi qualsiasi  $a$  e  $b$ , vale che **/
507  /***** se  $a$  in relazione con  $b$  allora anche
       $b$  in relazione con  $a$ . *****/
508
509  int controllo_simmetria (rel_bin verifica)
510  {
511
512      int i,
513          j,
514          riscontro,
515          simmetria;
516
517      simmetria = 1;
518
519
520      i = 0;
521      j = 0;
522      riscontro = 0;
523
524      /*controllo della simmetria per numeri*/
525
526      if (verifica.controllo == 1)
527      {
528
529          while ( i < verifica.dimensione)
530          {
531
532              j = 0;
533              while ( j < verifica.dimensione)
534              {
535
536                  if (verifica.primo_termine[i] ==
                      verifica.secondo_termine[j])
537                  if (verifica.primo_termine[j] ==
                      verifica.secondo_termine[i])
538                      riscontro++;
539                  j++;
540              }

```

```

541
542         if (riscontro == 0)
543         {
544             j = verifica.dimensione;
545             i = verifica.dimensione;
546             simmetria = 0;
547         }
548         riscontro = 0;
549         i++;
550     }
551
552 }
553
554 /* controllo della simmetria per stringhe */
555
556 if (verifica.controllo == 2)
557 {
558
559     while ( i < verifica.dimensione)
560     {
561
562         j = 0;
563         while ( j < verifica.dimensione)
564         {
565
566             if (strcmp (verifica.prima_stringa[i],
567                     verifica.seconda_stringa[j]) == 0 )
568                 if (strcmp (verifica.prima_stringa
569                         [j],verifica.seconda_stringa[i
570                         ]) == 0 )
571                     riscontro++;
572
573             j++;
574         }
575
576         if (riscontro == 0)
577         {
578             j = verifica.dimensione;
579             i = verifica.dimensione;
580             simmetria = 0;
581         }
582         riscontro = 0;
583         i++;
584     }
585 }

```

```

582
583     }
584
585     /* ***** Controllo se la simmetria stata
verificata ***** */
586
587     if (simmetria == 1)
588         printf ("L'insieme 'simmetrica\n");
589     else
590         printf ("L'insieme 'asimmetrica\n");
591
592     /* ***** Fine controllo simmetria ***** */
593
594     return (simmetria);
595 }
596
597
598
599 /* FUNZIONE PER CONTROLLARE LA TRANSITIVITA' */
600
601 /* ***** Definizione: In matematica, una relazione
binaria R in un insieme X transitiva se e solo se
602 per ogni a, b, c appartenenti ad X, se a e b in
relazione con b e b e c in relazione con c,
allora
603 a e c in relazione con c. ***** */
604
605
606 int controllo_transitivita (rel_bin verifica)
607 {
608
609     int i,
610         j,
611         k,
612         transitivita;
613
614     /*IMPOSTO LA TRANSITIVITA INIZIALMENTE COME VERA E
AZZERO I CONTATORI*/
615     transitivita = 1;
616     i = 0;
617     j = 0;
618     k = 0;
619
620     /* VERIFICA TRANSITIVITA PER NUMERI */

```

```

621
622
623     if (verifica.controllo == 1)
624     {
625
626         while (i < verifica.dimensione)
627         {
628             j = 0;
629
630             while (j < verifica.dimensione)
631             {
632                 k=0;
633
634                 if (verifica.secondo_termine[i] ==
635                     verifica.primo_termine[j])
636                 {
637                     transitivita = 0;
638
639                     while (k < verifica.dimensione)
640                     {
641                         if (verifica.primo_termine[i]
642                             == verifica.primo_termine[k]
643                             )
644                         {
645                             if (verifica.
646                                 secondo_termine[k]==
647                                 verifica.
648                                 secondo_termine[j])
649                             {
650                                 transitivita = 1;
651                                 k = verifica.
652                                     dimensione;
653                             }
654                         }
655                     }
656                     k++;
657                 }
658             }
659         }
660     }
661     if (transitivita==0)
662     {
663         j=verifica.dimensione;
664         i=verifica.dimensione;
665     }
666 }

```



```

658
659             j++;
660         }
661
662         i++;
663     }
664 }
665
666
667 /****** VERIFICA TRANSITIVIT PER
        STRINGHE *****/
668
669 if (verifica.controllo == 2)
670 {
671
672     while (i < verifica.dimensione)
673     {
674         j = 0;
675
676         while (j < verifica.dimensione)
677         {
678             k=0;
679
680             if (strcmp (verifica.seconda_stringa[i]
681                     ],verifica.prima_stringa[j]) == 0)
682             {
683                 transitivita = 0;
684
685                 while (k < verifica.dimensione)
686                 {
687                     if (strcmp (verifica.
688                             prima_stringa[i],verifica.
689                             prima_stringa[k]) == 0)
690                     {
691                         if (strcmp (verifica.
692                             seconda_stringa[k],
693                             verifica.
694                             seconda_stringa[j]) ==
695                             0)
696                         {
697                             transitivita = 1;
698                             k = verifica.
699                                 dimensione;

```

```

693         }
694     }
695
696     k++;
697 }
698
699     if (transitivita==0)
700     {
701         j=verifica.dimensione;
702         i=verifica.dimensione;
703     }
704 }
705
706     j++;
707 }
708
709     i++;
710 }
711
712 }
713
714 /****** Controllo se la relazione Transitiva
******/
715
716     if (transitivita == 1)
717         printf ("LLE'transitiva\n");
718
719     else
720         printf ("LLEnonLE'transitiva\n");
721
722 /****** Fine controllo Transitivity
******/
723
724     return (transitivita);
725
726 }
727
728 /****** Dicotomia ******/
729
730 int controllo_dicotomia (rel_bin verifica)
731 {
732
733     int i,j,k;
734     int numero_elementi;

```

```

735     int dicotomia = 0;
736     int dimensione;
737     int riscontro;
738     int secondo_riscontro;
739     i=0;
740     j=0;
741     k=i-1;
742     riscontro = 0;
743     dimensione = verifica.dimensione;
744
745     /****** Dicotomia per numeri *****/
746
747     if (verifica.controllo == 1)
748     {
749
750         /****** Conto il numero delle coppie
751         esistenti (scarto le coppie uguali)
752         *****/
753
754         while ( i < verifica.dimensione)
755         {
756             k = i-1;
757             j = i+1;
758             secondo_riscontro = 0;
759
760             if (i>0)
761             {
762                 while ( k >= 0 )
763                 {
764                     if (verifica.primo_termine[i] ==
765                         verifica.primo_termine[k])
766                     {
767                         if (verifica.secondo_termine[i
768                             ] == verifica.
769                             secondo_termine[k])
770                             secondo_riscontro = 1;
771                     }
772                     k--;
773                 }
774             }
775
776             if (secondo_riscontro != 1)
777             {
778                 while ( j < verifica.dimensione)

```

```

774         {
775             if (verifica.primo_termine[i] ==
776                 verifica.primo_termine[j])
777                 if (verifica.secondo_termine[i]
778                     ] == verifica.
779                         secondo_termine[j])
780                     {
781                         dimensione--;
782                     }
783                 j++;
784             }
785         }
786         i++;
787     }
788
789     i=0;
790     j=0;
791     k=0;
792     numero_elementi=0;
793     riscontro = 0;
794     /****** Conto il numero degli
795     elementi distinti esistenti *****
796     */
797
798     while (i<verifica.dimensione)
799     {
800         k=i-1;
801         secondo_riscontro = 0;
802
803         while (k >= 0)
804         {
805             if (verifica.primo_termine[i] ==
806                 verifica.primo_termine[k])
807                 secondo_riscontro = 1;
808             k--;
809         }
810         if (secondo_riscontro != 1)
811         {
812             if (verifica.primo_termine[i] ==
813                 verifica.secondo_termine[i])
814                 riscontro++;
815         }
816     }

```

```

811         i++;
812     }
813
814     numero_elementi = riscontro;
815
816     /****** Conto quanti dovrebbero essere
        gli elementi per avere la dicotomia
        *****/
817
818     while (numero_elementi > 0)
819     {
820         numero_elementi--;
821         riscontro = riscontro + numero_elementi;
822     }
823 }
824
825 /****** VERIFICA DICOTOMICA PER
    STRINGHE *****/
826
827 if (verifica.controllo == 2)
828 {
829
830     /****** Conto il numero delle coppie
        esistenti (scarto le coppie uguali)
        *****/
831
832     while ( i < verifica.dimensione)
833     {
834         k = i-1;
835         j = i+1;
836         secondo_riscontro = 0;
837         if (i>0)
838         {
839             while ( k >= 0 )
840             {
841                 if ( (strcmp (verifica .
                        prima_stringa[i], verifica .
                        prima_stringa[k])) == 0)
842                 {
843                     if ( (strcmp (verifica .
                                seconda_stringa[i], verifica
                                .seconda_stringa[k])) == 0)
844                         secondo_riscontro = 1;
845                 }

```

```

846         k--;
847     }
848 }
849
850     if (secondo_riscontro != 1)
851     {
852         while ( j < verifica.dimensione)
853         {
854             if ( (strcmp (verifica .
                        prima_stringa[i], verifica .
                        prima_stringa[j])) == 0)
855                 if ( (strcmp (verifica .
                        seconda_stringa[i], verifica .
                        seconda_stringa[j])) == 0)
856                     {
857                         dimensione--;
858                     }
859                 j++;
860             }
861         }
862         i++;
863     }
864
865     i=0;
866     k=0;
867     j=0;
868     numero_elementi = 0;
869     /****** Conto il numero degli
870     elementi distinti esistenti *****
871     */
872     while (i<verifica.dimensione)
873     {
874         k=i-1;
875         secondo_riscontro = 0;
876
877         while (k >= 0)
878         {
879             if ( (strcmp (verifica.prima_stringa[i
                        ], verifica.prima_stringa[k])) == 0)
880                 secondo_riscontro = 1;
881             k--;
882         }

```

```

883         if (secondo_riscontro != 1)
884         {
885             if ( (strcmp (verifica.prima_stringa[i]
                        ],verifica.seconda_stringa[i])) ==
                        0)
886                 numero_elementi++;
887
888         }
889         i++;
890     }
891     riscontro = numero_elementi;
892
893     /****** Conto quanti dovrebbero essere
        gli elementi per avere la dicotomia
        *****/
894
895     while (numero_elementi > 0)
896     {
897
898         numero_elementi--;
899         riscontro = riscontro + numero_elementi;
900
901     }
902
903 }
904
905 /****** Verifico se la dicotomia
        verificata *****/
906
907 if (dimensione == riscontro)
908     dicotomia = 1;
909
910 if (dicotomia == 1 )
911     printf ("L' e' dicotomica\n\n");
912
913 else
914     printf ("L' non e' dicotomica\n\n");
915
916 /****** Fine verifica dicotomia
        *****/
917
918 return (dicotomia);
919 }
920

```

```

921  /*Funzione di verifica dell'ordine totale*/
922
923
924  void ordine_totale (rel_bin verifica)
925  {
926
927      int parziale ,
928          dicotomia;
929
930      dicotomia=2;
931      parziale = ordine_parziale (verifica);
932      if (parziale == 1)
933          dicotomia = controllo_dicotomia (verifica);
934
935      if (parziale == 0)
936          printf ("\n_l'ordine_non_e'totale_in_quanto_
937                  non_e'nemmeno_parziale");
938
939      if (dicotomia == 0)
940          printf ("\n_l'ordine_non_e'totale_in_quanto_
941                  non_viene_rispettata_la_propieta'di_
942                  dicotomia");
943
944      if (dicotomia == 1 && parziale == 1)
945          printf ("\n_Quindi_e'una_relazione_d'ordine_
946                  totale");
947
948      printf ("\n\n..._Controllo_Ordine_Totale_
949              Terminato...\n\n\n");
950  }
951
952  /*Funzione che stabilisce se e'una relazione di
953  equivalenza o meno*/
954
955  void relazione_equivalenza (rel_bin verifica)
956  {
957
958      int riflessivita;
959      int simmetria;
960      int transitivita;
961
962      riflessivita = controllo_riflessivita (verifica);
963      simmetria = controllo_simmetria (verifica);
964      transitivita = controllo_transitivita (verifica);

```



```

959
960     if (riflessivita == 1 && simmetria == 1 &&
          transitivita == 1)
961         printf ("\n_Quindi_e'una_relazione_di_
                equivalenza\n");
962
963     if (riflessivita == 0)
964         printf ("\n_Quindi_non_e'una_relazione_di_
                equivalenza_perche'non_riflessiva\n");
965
966     if (simmetria == 0)
967         printf ("\n_Quindi_non_e'una_relazione_di_
                equivalenza_perche'non_simmetrica\n");
968
969     if (transitivita == 0)
970         printf ("\n_Quindi_non_e'una_relazione_di_
                equivalenza_perche'non_transitiva\n");
971 }
972
973 /*Funzione che stabilisce se la relazione binaria
    acquisita e'una funzione matematica*/
974
975 void controllo_funzione (rel_bin verifica)
976 {
977
978     int i;
979     int k;
980     int termini_diversi;
981     int termini_uguali_prima;
982     int termini_uguali_dopo;
983     int errore;
984
985     if (verifica.controllo == 1)
986     {
987
988         i=0;
989         errore=0;
990         termini_diversi=0;
991         termini_uguali_dopo=0;
992         termini_uguali_prima=0;
993         while (i < verifica.dimensione)
994         {
995             k=verifica.dimensione-1;
996             termini_uguali_dopo=termini_uguali_prima;

```

```

997     while (k > i)
998     {
999         if (verifica.primo_termine[i] ==
1000             verifica.primo_termine[k])
1001         {
1002             if (verifica.secondo_termine[i] !=
1003                 verifica.secondo_termine[k])
1004             {
1005                 errore=1;
1006                 printf ("\nNel_%d_elemento_c',
1007                     e'un_errore_che_impedisce_
1008                     alla_relazione_binaria\n",k
1009                     +1);
1010                 printf ("_di_essere_una_
1011                     funzione\n");
1012                 k=i;
1013                 i=verifica.dimensione;
1014             }
1015             if (verifica.secondo_termine[i] ==
1016                 verifica.secondo_termine[k])
1017                 termini_uguali_dopo++;
1018         }
1019         k--;
1020     }
1021     if (errore == 0 && termini_uguali_dopo ==
1022         termini_uguali_prima)
1023         termini_diversi++;
1024     termini_uguali_prima = termini_uguali_dopo
1025         ;
1026     i++;
1027 }
1028 if (errore == 0 && (termini_diversi == (
1029     verifica.dimensione - termini_uguali_prima)
1030 ))
1031 {
1032     if(verifica.insieme_a == 2)
1033         printf ("\nLa_relazione_binaria_e'una
1034             _funzione_totale\n");
1035     else
1036         printf ("\nLa_relazione_binaria_ _una
1037             _funzione_parziale\n");
1038     controllo_biiettivita (verifica);
1039 }

```

```

1028         else
1029             printf ("\nLa relazione binaria non e' una
                _funzione\n");
1030     }
1031
1032     /****** Controllo se c' e' una funzione per
        stringhe (le stringhe sono considerate come
        costanti di diverso valore) *****/
1033
1034     if (verifica.controllo == 2)
1035     {
1036
1037         i=0;
1038         errore=0;
1039         termini_diversi=0;
1040         termini_uguali_dopo=0;
1041         termini_uguali_prima=0;
1042         while (i < verifica.dimensione)
1043         {
1044             k=verifica.dimensione-1;
1045             termini_uguali_dopo=termini_uguali_prima;
1046             while (k > i)
1047             {
1048                 if ( (strcmp (verifica.prima_stringa[i]
                                ],verifica.prima_stringa[k])) == 0)
1049                 {
1050                     if ( (strcmp (verifica .
                                seconda_stringa[i],verifica .
                                seconda_stringa[k])) != 0)
1051                     {
1052                         errore=1;
1053                         printf ("\nNel %d elemento c'
                                e' un errore che impedisce
                                alla relazione binaria\n",k
                                +1);
1054                         printf ("\ndi essere una
                                funzione\n");
1055                         k=i;
1056                         i=verifica.dimensione;
1057                     }
1058                     else
1059                         termini_uguali_dopo++;
1060                 }
1061                 k--;

```

```

1062     }
1063     if (errore == 0 && termini_uguali_dopo ==
        termini_uguali_prima)
1064         termini_diversi++;
1065
1066     termini_uguali_prima = termini_uguali_dopo
        ;
1067     i++;
1068 }
1069 if (errore == 0 && (termini_diversi == (
    verifica.dimensione - termini_uguali_prima)
    ))
1070 {
1071     if(verifica.insieme_a == 2)
1072         printf ("\nLa relazione binaria e' una
            _funzione_totale\n");
1073     else
1074         printf ("\nLa relazione binaria e' una
            _funzione_parziale\n");
1075     controllo_biiettivita (verifica);
1076 }
1077 else
1078     printf ("\nLa relazione binaria non e' una
        _funzione\n");
1079 }
1080
1081 printf ("\n\n... _Controllo_Funzione_Terminato_
    ... \n\n\n");
1082
1083 }
1084
1085 /******FUNZIONE PER IL controllo DELL'INIETTIVITA
    ******/
1086
1087 int controllo_iniettivita (rel_bin verifica)
1088 {
1089
1090     int i;
1091     int k;
1092     int termini_diversi;
1093     int termini_uguali_prima;
1094     int termini_uguali_dopo;
1095     int errore;
1096     int iniettivita;

```

```

1097
1098     iniettivita = 0;
1099
1100     if (verifica.controllo == 1)
1101     {
1102
1103         i=0;
1104         errore=0;
1105         termini_diversi=0;
1106         termini_uguali_dopo=0;
1107         termini_uguali_prima=0;
1108
1109         while (i < verifica.dimensione)
1110         {
1111
1112             k=verifica.dimensione-1;
1113             termini_uguali_dopo=termini_uguali_prima;
1114             while (k > i)
1115             {
1116
1117                 if (verifica.secondo_termine[i] ==
1118                     verifica.secondo_termine[k])
1119                 {
1120
1121                     if (verifica.primo_termine[i] !=
1122                         verifica.primo_termine[k])
1123                     {
1124
1125                         errore=1;
1126                         printf ("\n_Nel_%d_elemento_c'
1127                             e'un_errore_che_impedisce_
1128                             alla_funzione\n",k+1);
1129                         printf ("_di_essere_iniettiva\
1130                             n");
1131                         k=i;
1132                         i=verifica.dimensione;
1133                     }
1134                     if (verifica.primo_termine[i] ==
1135                         verifica.primo_termine[k])
1136                         termini_uguali_dopo++;
1137                 }
1138             }
1139             k--;
1140         }
1141     }

```

```

1134         if (errore == 0 && termini_uguali_dopo ==
1135             termini_uguali_prima)
1136             termini_diversi++;
1137         termini_uguali_prima = termini_uguali_dopo
1138             ;
1138         i++;
1139     }
1140     if (errore == 0 && (termini_diversi == (
1141         verifica.dimensione - termini_uguali_prima)
1142     ))
1141     {
1142         printf ("\nLa funzione e' iniettiva\n");
1143         iniettivita = 1;
1144     }
1145     else
1146         printf ("\nLa funzione non e' iniettiva\n"
1147             );
1148
1149 }
1150
1151 /****** Controllo iniettivita' per stringhe
1152 ******/
1153
1153     if (verifica.controllo == 2)
1154     {
1155
1156         i=0;
1157         errore=0;
1158         termini_diversi=0;
1159         termini_uguali_dopo=0;
1160         termini_uguali_prima=0;
1161
1162         while (i < verifica.dimensione)
1163         {
1164             k=verifica.dimensione-1;
1165             termini_uguali_dopo=termini_uguali_prima;
1166             while (k > i)
1167             {
1168                 if ( (strcmp (verifica.seconda_stringa
1169                     [i], verifica.seconda_stringa[k]))
1170                     == 0)
1171                 {

```

```

1170         if ( (strcmp (verifica .
                    prima_stringa[i], verifica .
                    prima_stringa[k])) != 0)
1171         {
1172             errore=1;
1173             printf ("\n_Nel_%d_elemento_c'
                    e' un_errore_che_impedisce_
                    alla_funzione\n", k+1);
1174             printf ("_di_essere_iniettiva\
                    n");
1175             k=i;
1176             i=verifica . dimensione;
1177         }
1178         if ( (strcmp (verifica .
                    prima_stringa[i], verifica .
                    prima_stringa[k])) == 0)
                    termini_uguali_dopo++;
1179
1180     }
1181
1182     k--;
1183 }
1184 if (errore == 0 && termini_uguali_dopo ==
    termini_uguali_prima)
1185     termini_diversi++;
1186
1187     termini_uguali_prima = termini_uguali_dopo
        ;
1188     i++;
1189 }
1190 if (errore == 0 && (termini_diversi == (
    verifica . dimensione - termini_uguali_prima)
    ))
1191 {
1192     printf ("\n_La_funzione_e' iniettiva");
1193     iniettivita = 1;
1194 }
1195 else
1196     printf ("\n_La_funzione_non_e' iniettiva");
1197 }
1198
1199 return (iniettivita);
1200 }
1201

```

```

1202  /*****FUNZIONE PER IL controllo DELLA
      SURIETTIVITA *****/
1203
1204  int controllo_suriettivita (rel_bin verifica)
1205  {
1206      int suriettivita;
1207
1208      if (verifica.insieme_b == 2)
1209      {
1210          suriettivita = 1;
1211          printf ("\nla funzione e' suriettiva");
1212      }
1213
1214      else
1215      {
1216          suriettivita = 0;
1217          printf ("\nla funzione non e' suriettiva");
1218      }
1219
1220      return (suriettivita);
1221  }
1222
1223  /*****FUNZIONE PER IL controllo DELLA
      BIETTIVITA *****/
1224
1225  void controllo_biettivita (rel_bin verifica)
1226  {
1227
1228      int    surriettivita ,
1229            iniettivita;
1230
1231      surriettivita = controllo_suriettivita (verifica);
1232      iniettivita = controllo_iniettivita (verifica);
1233
1234
1235      if ( surriettivita == 1 && iniettivita == 1)
1236          printf ("\nla funzione e' biettiva");
1237      else
1238          printf ("\nla funzione non e' biettiva");
1239      return;
1240  }
1241
1242
1243  int controllo_antisimmetria (rel_bin verifica)

```



```

1244 {
1245
1246     int i ,
1247         j ,
1248         riscontro ,
1249         antisimmetria;
1250
1251     antisimmetria = 1;
1252
1253
1254     i = 0;
1255     j = 0;
1256     riscontro = 1;
1257
1258     /* controllo della antisimmetria per numeri*/
1259
1260     if (verifica.controllo == 1)
1261     {
1262
1263         while ( i < verifica.dimensione)
1264         {
1265
1266             j = 0;
1267             while ( j < verifica.dimensione)
1268             {
1269
1270                 if (verifica.primo_termine[i] ==
1271                     verifica.secondo_termine[j])
1272                     if (verifica.primo_termine[j] ==
1273                         verifica.secondo_termine[i])
1274                         if (verifica.primo_termine[i]
1275                             == verifica.primo_termine[j]
1276                             )
1277                             riscontro++;
1278                 else
1279                     riscontro = 0;
1280             j++;
1281         }
1282
1283         if (riscontro == 0)
1284         {
1285             j = verifica.dimensione;
1286             i = verifica.dimensione;
1287             antisimmetria = 0;

```

```

1284         }
1285         i++;
1286     }
1287
1288 }
1289
1290 /* controllo della antisimmetria per stringhe */
1291
1292 if (verifica.controllo == 2)
1293 {
1294
1295     while ( i < verifica.dimensione)
1296     {
1297
1298         j = 0;
1299         while ( j < verifica.dimensione)
1300         {
1301
1302             if (strcmp (verifica.prima_stringa[i],
1303                     verifica.seconda_stringa[j]) == 0 )
1304                 if (strcmp (verifica.prima_stringa
1305                         [j],verifica.seconda_stringa[i
1306                         ]) == 0 )
1307                     if (strcmp (verifica.
1308                             prima_stringa[j],verifica.
1309                             prima_stringa[i]) == 0 )
1310                         riscontro++;
1311
1312             else
1313                 riscontro=0;
1314
1315             j++;
1316         }
1317
1318         if (riscontro == 0)
1319         {
1320             j = verifica.dimensione;
1321             i = verifica.dimensione;
1322             antisimmetria = 0;
1323         }
1324         i++;
1325     }
1326 }

```

```

1323      /****** Controllo se la simmetria stata
           verificata *****/
1324
1325      if (antisimmetria == 1)
1326          printf ("_le'antisimmetrica\n");
1327      else
1328          printf ("_non_le'antisimmetrica\n");
1329
1330      /****** Fine controllo simmetria *****/
1331
1332      return (antisimmetria);
1333  }

```

4.3 Test

```
1 #include<stdio.h>
2 #include"librerie/progetto.h"
3
4 int main (void)
5 {
6     struct relBin RelazioneBinaria;
7     int scelta;
8     int scan;
9     int test_terminati;
10    char carattere_non_letto;
11
12    scan = 0;
13    test_terminati = 0;
14    printf ("\n_Programma_per_effettuare_i_Test_sulla_
        libreria\n");
15
16
17    printf ("\n\n_Digita_il_numero_corrispondente_all'
        azione_che_si_vuole_svolgere\n");
18    printf ("\n1)_Test_Acquisizione\n2)_Esci\n");
19
20    do
21    {
22        printf ("\n_scelta:_");
23        scan = scanf("%d",
24                    &scelta);
25        if ((scelta < 1) || (scelta > 2) || scan != 1)
26            do
27                carattere_non_letto = getchar();
28                while (carattere_non_letto != '\n');
29    }
30    while ((scelta < 1) || (scelta > 2) || scan != 1);
31
32
33    if (scelta == 1)
34        RelazioneBinaria = acquisizione(
            RelazioneBinaria);
35
36    if (scelta == 2)
37    {
38        printf ("\n\n.....Test_terminati.....\n\n");
39        test_terminati = 1;
```

```

40     }
41
42     scelta = -1;
43     while(scelta != 7 && test_terminati != 1)
44     {
45         printf("\n\nDigita il numero corrispondente a
           all'azione che si vuole svolgere\n");
46         printf("\n1) Test Acquisizione\n2) Test
           Stampa\n3) Test verifica ordine parziale\n
           4) Test verifica ordine totale\n");
47         printf("\n5) Test verifica relazione d'
           equivalenza\n6) Test funzione\n7) Esci\n"
           );
48         scelta = -1;
49         do
50         {
51             printf ("\n_scelta:_");
52             scan = scanf("%d",
53                         &scelta);
54             if ((scelta < 1) || (scelta > 7) || scan
               != 1)
55                 do
56                     carattere_non_letto = getchar();
57                     while (carattere_non_letto != '\n');
58         }
59         while ((scelta < 1) || (scelta > 7) || scan !=
               1);
60
61
62         if (scelta == 1)
63             RelazioneBinaria = acquisizione (
               RelazioneBinaria);
64         if (scelta == 2)
65             stampa (RelazioneBinaria);
66         if (scelta == 3)
67             ordine_parziale (RelazioneBinaria);
68         if (scelta == 4)
69             ordine_totale (RelazioneBinaria);
70         if (scelta == 5)
71             relazione_equivalenza (RelazioneBinaria);
72         if (scelta == 6)
73             controllo_funzione (RelazioneBinaria);
74         if (scelta == 7)
75         {

```

```

76             printf ("\n\n..... Test_terminati ..... \n\n
77                 test_terminati = 1;
78             }
79         }
80     return(0);
81
82 }

```

4.4 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: controlloriflessività: 1,controllosimmetria: 0, controllotransitività: 1 controllodicotomia: 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:controlloriflessività: 1,controllosimmetria: 0, controllotransitività:
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:controlloriflessività: 1,controllosimmetria: 0, controllotransitività: 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:controlloriflessività: 1,controllosimmetria: 1, controllotransitività: 1 controllodicotomia: 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:controlloriflessività: 0,controllosimmetria: 0, controllotransitività: 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:
```

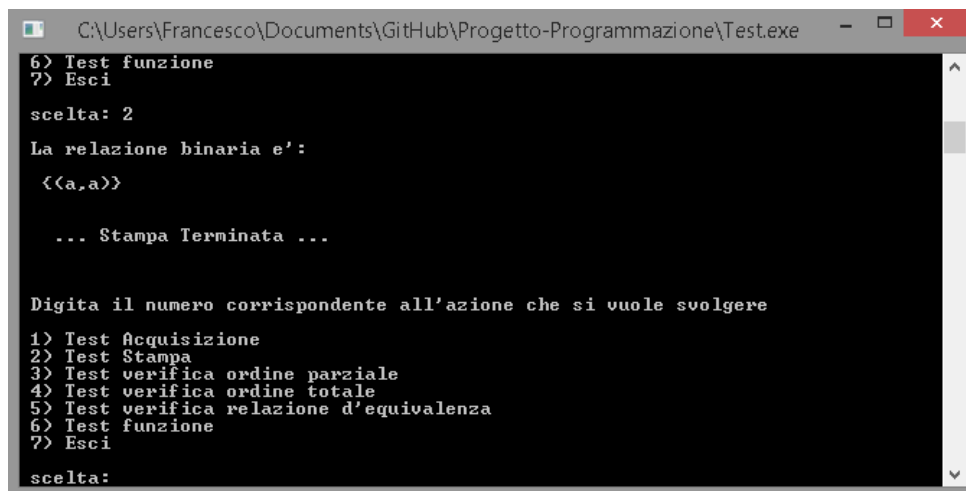
5.6 Test 6:

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

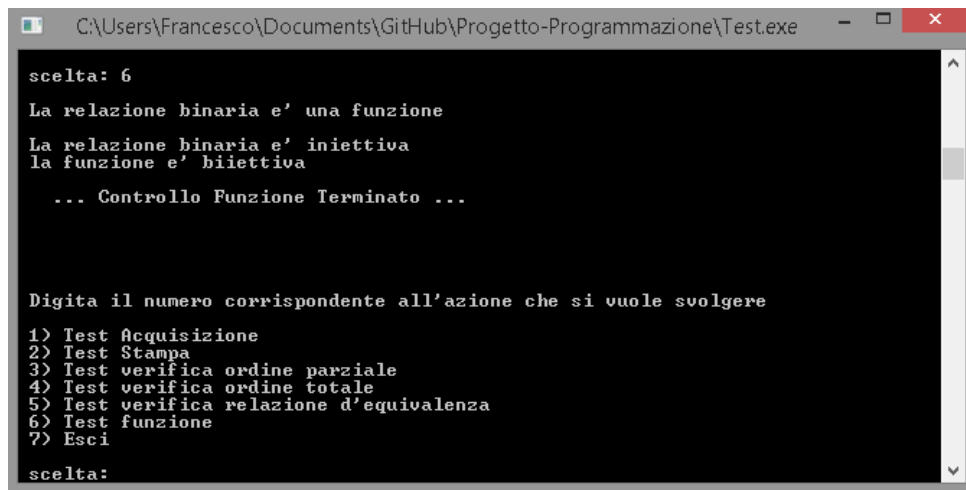
La relazione binaria e':

<<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:
```

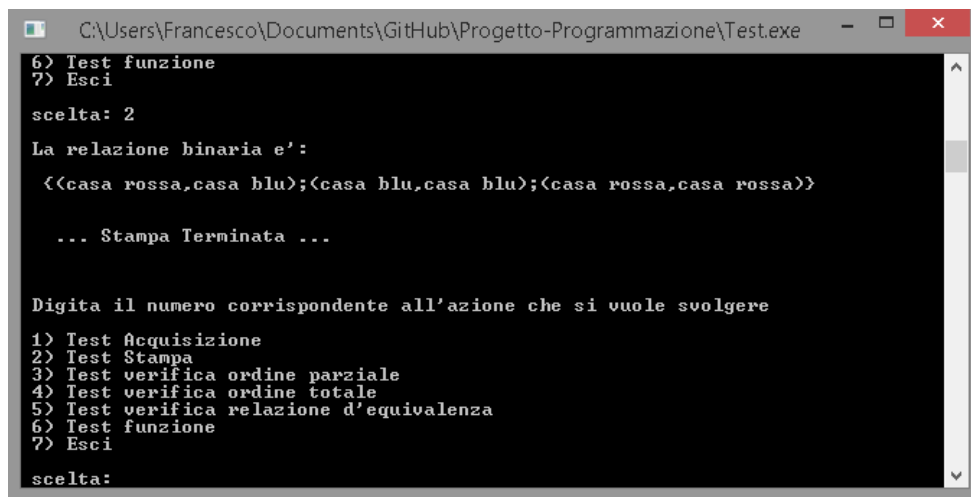
5.7 Test 7:

Test per verificare il controllo degli inputs.

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

Outputs:controllo_riflessività: 1,controllo_simmetria: 1, controllo_transitività:
1 dicotomia: 1 la relazione è una relazione d'ordine totale in quanto rispetta
le proprietà.

le funzioni funzionano anche con input contenuti degli spazi.



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

scelta: 2

La relazione binaria e':

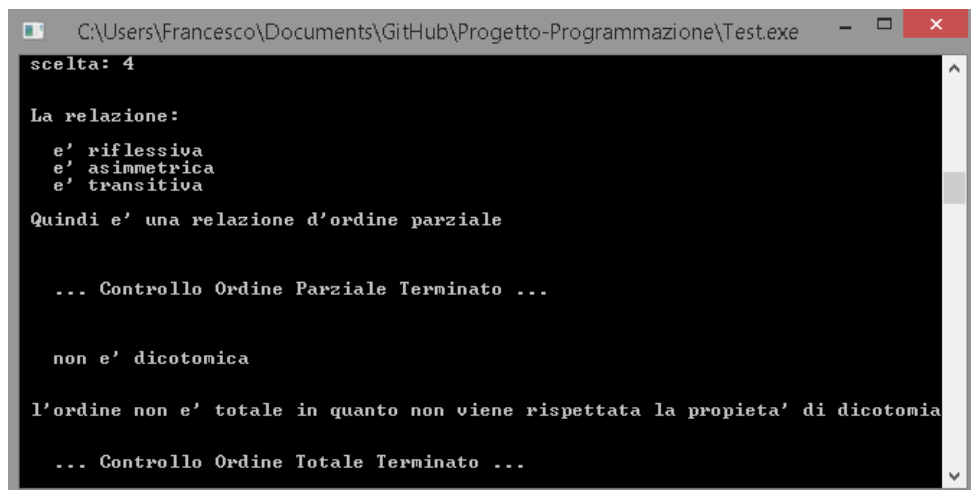
<(casa rossa,casa blu);(casa blu,casa blu);(casa rossa,casa rossa)>

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere

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

scelta:
```



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

scelta: 4

La relazione:

e' riflessiva
e' asimmetrica
e' transitiva

Quindi e' una relazione d'ordine parziale

... Controllo Ordine Parziale Terminato ...

non e' dicotomica

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

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

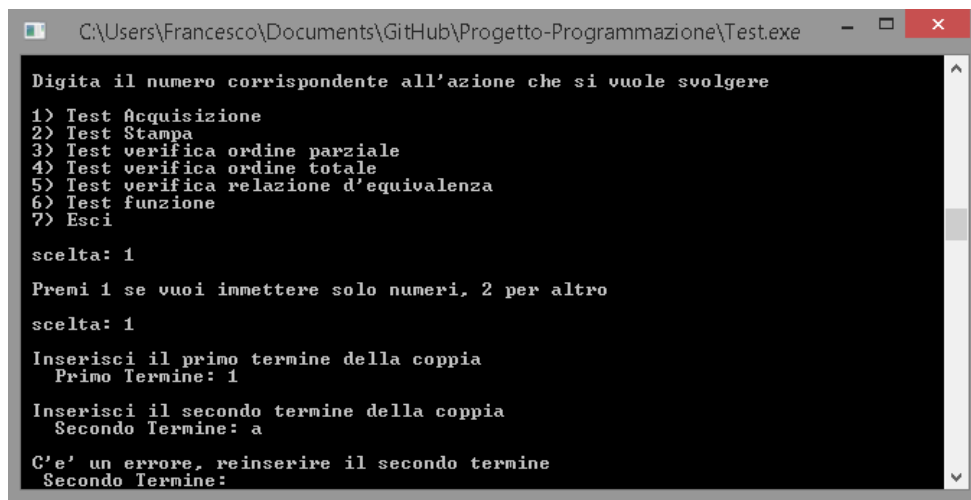
5.8 Test 8:

Test per inserire stringhe in una relazione numerica.

Inputs:(1,a)

Outputs: c'è un errore reinserisci il valore.

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



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

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

scelta: 1

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

Inserisci il primo termine della coppia
Primo Termine: 1

Inserisci il secondo termine della coppia
Secondo Termine: a

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

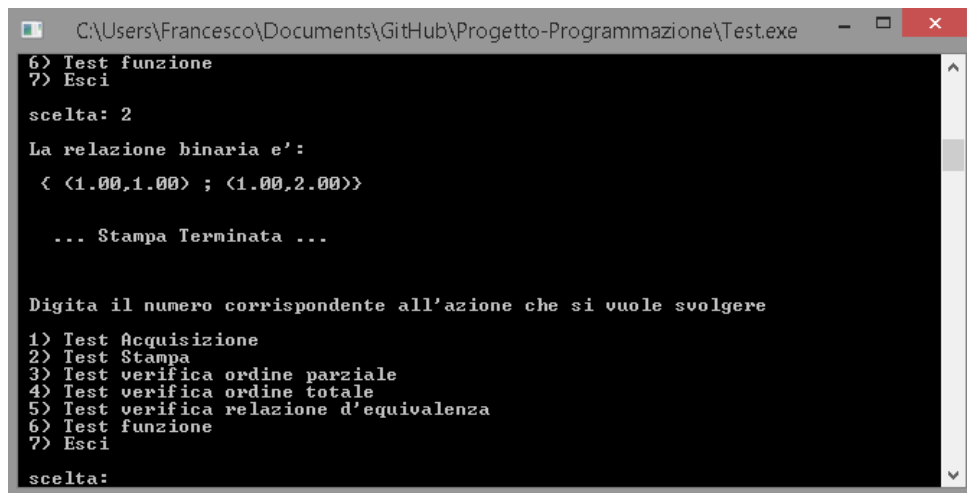
5.9 Test 9:

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

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

Outputs: La relazione binaria non è una funzione

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



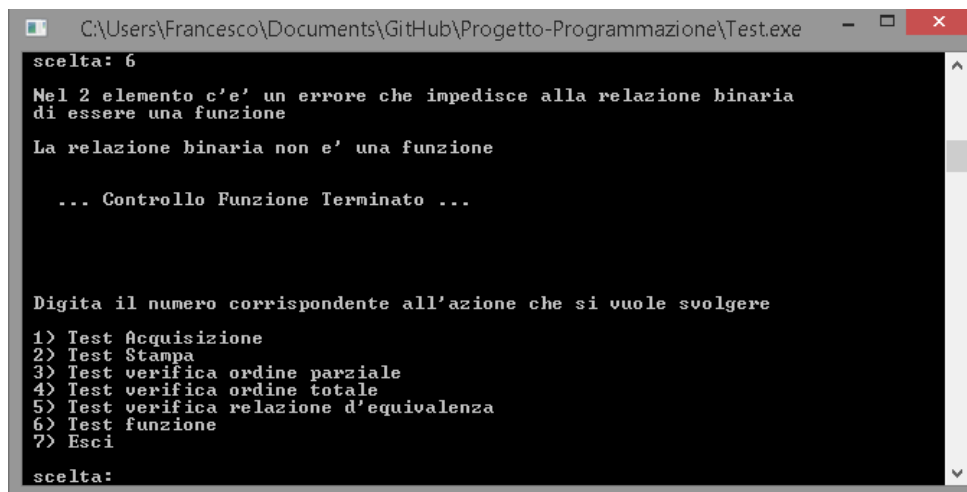
```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

scelta: 2

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

... Stampa Terminata ...

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



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

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

... Controllo Funzione Terminato ...

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


5.10 Test 10:

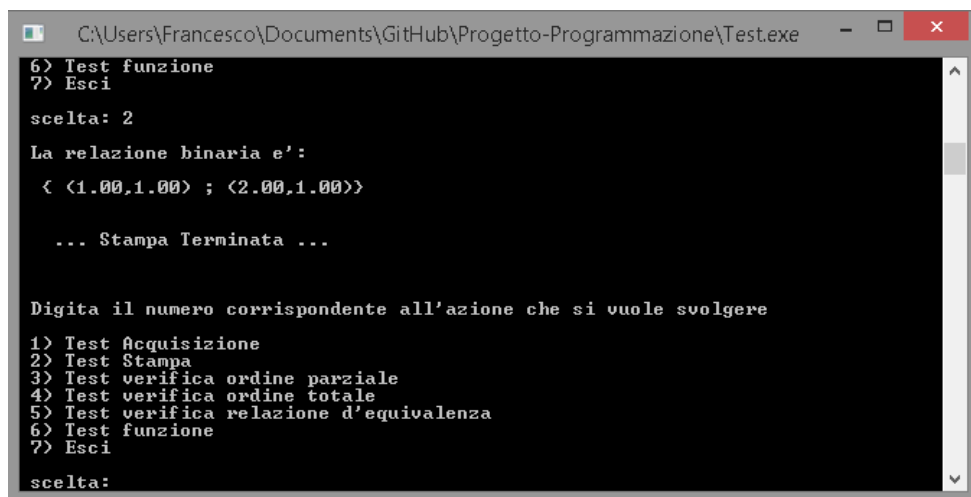
Inputs:(1,1)(2,1)

Outputs: La relazione binaria è una funzione

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

La funzione non è iniettiva

La funzione non è biiettiva



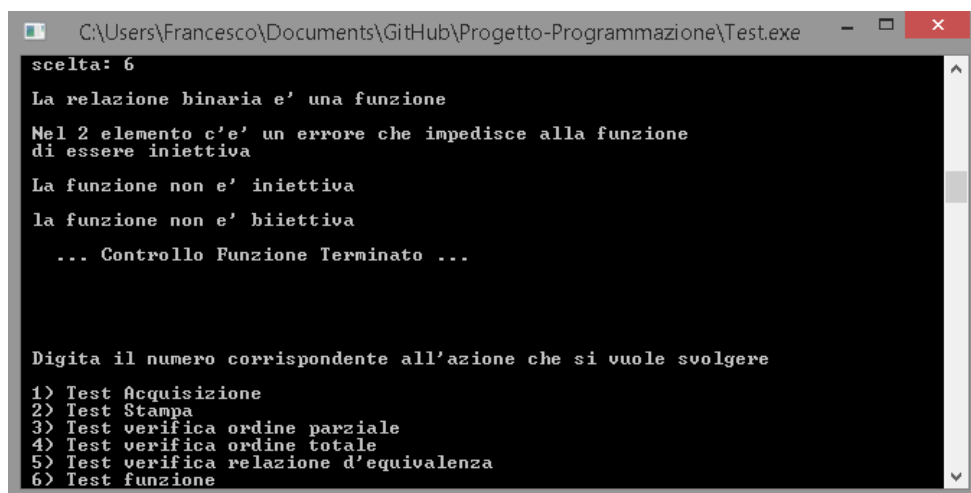
```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

scelta: 2

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

... Stampa Terminata ...

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



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

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

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

6 Verica del programma

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

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

La postcondizione è

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

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

-Il predicato

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

e la funzione:

$$\text{tr}(\text{numero_elementi}) = \text{numero_elementi} - 1)$$

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

$$\{P \wedge \text{numero_elementi} > 0\} \text{riscontro} = \text{riscontro} + \text{numero_elementi}; \text{numero_elementi} = \text{numero_elementi} - -; \{P\}$$

segue da:

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

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

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

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

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

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

* La limitatezza segue da:

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

$| = \text{numero_elementi} > \text{numero_elementi} - 1$
 Poichè:

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

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

-Proseguendo infine a ritroso si ottiene prima:

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

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

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