

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 (file .h)	11
4.2	Libreria (file .c)	12
4.3	Test	50
4.4	Makefile	53
5	Testing del programma	54
5.1	Test 1:	54
5.2	Test 2:	55
5.3	Test 3:	56
5.4	Test 4:	57
5.5	Test 5:	58
5.6	Test 6:	59
5.7	Test 7:	60
5.8	Test 8:	61
5.9	Test 9:	62
5.10	Test 10:	63
6	Verica del programma	64

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          */
19      int controllo ,
20           dimensione ,
21           insieme_a ,
22           insieme_b;
23 } rel_bin;
24
25 extern rel_bin acquisizione (rel_bin);
26 extern int controllo_simmetria (rel_bin);
27 extern int controllo_riflessivita (rel_bin);
28 extern int controllo_transitivita (rel_bin);
29 extern int controllo_suriettivita (rel_bin);
30 extern void controllo_biiettivita (rel_bin);
31 extern int controllo_antisimmetria (rel_bin);
32 extern void controllo_funzione (rel_bin);
33 extern void relazione_equivalenza (rel_bin);
34 extern void ordine_totale (rel_bin);
35 extern int ordine_parziale (rel_bin);
36 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
9 int controllo_simmetria (rel_bin);
10 int controllo_riflessivita (rel_bin);
11 int controllo_transitivita (rel_bin);
12 int controllo_suriettivita (rel_bin);
13 int controllo_antisimmetria (rel_bin);
14 int ordine_parziale (rel_bin);
15
16 void controllo_biiettivita (rel_bin);
17 void controllo_funzione (rel_bin);
18 void relazione_equivalenza (rel_bin);
19 void ordine_totale (rel_bin);
20 void stampa (rel_bin);
21
22
23 /***** Funzione di acquisizione
      *****/
24
25 rel_bin acquisizione (rel_bin relazione)
26 {
27
28     int acquisizione_finita ,
29         risultato_lettura ,
30         primo_termine_acquisito ,
31         i;
32
33     char temporaneo ,
34         carattere_non_letto;
35
36     acquisizione_finita = 0;
37     risultato_lettura = 0;
38     primo_termine_acquisito = 0;
39     i=0;
40
41     relazione.dimensione = 0;
```

```

42     relazione.primo_termine = (double *) malloc (2);
43     relazione.secondo_termine = (double *) malloc (2);
44     relazione.prima_stringa = (char **) malloc (2);
45     relazione.seconda_stringa = (char **) malloc (2);
46
47     do
48     {
49         printf ("\n_Premi\n\n1_se_vuoi_immettere_solo
           _numeri,\n2_per_inserire_stringhe_");
50         printf ("\n3_per_la_relazione_vuota\n");
51         printf ("\n_scelta:_");
52         risultato_lettura = scanf ("%d",
53                                     &relazione.
                                     controllo);
54         if (relazione.controllo < 1 || relazione.
           controllo > 3 || risultato_lettura != 1)
55             do
56                 carattere_non_letto = getchar();
57                 while (carattere_non_letto != '\n');
58     }
59     while (relazione.controllo < 1 || relazione.
           controllo > 3 || risultato_lettura != 1);
60
61     /** Imposto di nuovo risultato_lettura a 0 **/
62
63     risultato_lettura=0;
64
65     /* Relazione vuota */
66
67     if (relazione.controllo == 3)
68     {
69         printf("\n_Si_e'_scelto_di_inserire_una_
           relazione_vuota\n");
70     }
71
72     /* Acquisizione Numerica */
73
74     if (relazione.controllo == 1)
75     {
76         while (acquisizione_finita == 0)
77         {
78             primo_termine_acquisito = 0;
79             relazione.dimensione++;
80             acquisizione_finita = 2;

```

```

81
82      /*Acquisisco i termini della coppia*/
83
84      printf ("\n_Inserisci_i_termini_della_
            coppia_\n");
85      relazione.primo_termine = (double *)
            realloc (relazione.primo_termine, (
            relazione.dimensione+1) * sizeof (
            double));
86      relazione.secondo_termine = (double *)
            realloc (relazione.secondo_termine, (
            relazione.dimensione+1) * sizeof (
            double));
87      risultato_lettura = 0;
88
89
90      do
91      {
92          /*Acquisisco il primo termine*/
93          if (primo_termine_acquisito == 0)
94          {
95              printf ("_Primo_Termine:_");
96              risultato_lettura = scanf ("%lf",
97                                          &
                                                    relazione
                                                    .
                                                    primo_termine
                                                    [
                                                    relazione
                                                    .
                                                    dimensione
                                                    -
                                                    1]);
98          }
99
100         if (risultato_lettura == 1)
101             primo_termine_acquisito = 1;
102
103         /*Acquisisco il secondo termine*/
104         if (primo_termine_acquisito == 1)
105         {
106             printf ("_Secondo_Termine:_");
107             risultato_lettura = 0;
108             risultato_lettura = scanf ("%lf",

```



```

109                                     &
                                     relazione
                                     .
                                     secondo_termine
                                     [
                                     relazione
                                     .
                                     dimensione
                                     -
                                     1]);
110     }
111     /* Controllo che i valori siano stati
        letti correttamente e nel caso non
        sia così vuoto il buffer*/
112     if (risultato_lettura != 1)
113         do
114             carattere_non_letto = getchar
115             ();
116             while (carattere_non_letto != '\n'
117                 );
118             if (risultato_lettura == 0 &&
119                 primo_termine_acquisito == 0)
120                 printf ("\nC'e' un errore, \n
121                     reinserire il primo termine\n");
122                 ;
123             if (risultato_lettura == 0 &&
124                 primo_termine_acquisito == 1)
125                 printf ("\nC'e' un errore, \n
126                     reinserire il secondo termine\n
127                     ");
128         }
129     while (risultato_lettura != 1);
130     /* Chiedo all'utente se ci sono altre
        coppie*/
131     do
132     {
133         printf ("\nVuoi acquisire un'altra \n
134             coppia? immetti \n per uscire, \n per
135             continuare\n");
136     }
137     while (1);
138     printf ("\n scelta: \n");

```

```

131         risultato_lettura = scanf ("%d",
132                                     &
                                     acquisizione_finita
                                     );
133         if (acquisizione_finita < 0 ||
            acquisizione_finita > 1 ||
            risultato_lettura != 1)
134             do
135                 carattere_non_letto = getchar
                    ();
136                 while (carattere_non_letto != '\n'
                    );
137     }
138     while (acquisizione_finita < 0 ||
            acquisizione_finita > 1 ||
            risultato_lettura != 1);
139
140
141     }
142 }
143
144 /*imposto di nuovo risultato_lettura a 0*/
145 risultato_lettura = 0;
146
147 /*Acquisizione con stringhe*/
148 if (relazione.controllo == 2)
149 {
150     while (acquisizione_finita == 0)
151     {
152         primo_termine_acquisito = 0;
153         i=0;
154         temporaneo = 'a';
155         relazione.dimensione++;
156         acquisizione_finita = 2;
157
158         printf ("\n_Inserisci_i_termini_della_
                coppia_\n");
159         relazione.prima_stringa = (char **)
            realloc (relazione.prima_stringa, (
                relazione.dimensione+1) * sizeof (char
                *));
160
161         /*Acquisisco i termini della coppia*/

```

```

162     relazione.prima_stringa[relazione.
        dimensione - 1] = (char *) malloc (2);
163     fflush(stdin);
164     printf ("___Primo_Termine:_");
165     while (temporaneo != '\n')
166     {
167         temporaneo = getc (stdin);
168         relazione.prima_stringa [relazione.
            dimensione - 1] = (char*) realloc (
                relazione.prima_stringa[relazione.
                    dimensione-1],
169                 (i+1) * sizeof (char*));
170         relazione.prima_stringa [relazione.
            dimensione - 1] [i] = temporaneo;
171         i++;
172     }
173
174     /*Imposto ora il carattere di terminazione
        a \0 dato che adesso  \n*/
175
176     relazione.prima_stringa [relazione.
        dimensione - 1] [i - 1] = '\0';
177
178     /*Acquisisco il secondo termine della
        coppia*/
179
180     printf ("___Secondo_Termine:_");
181     relazione.seconda_stringa = (double **)
        realloc (relazione.seconda_stringa, (
            relazione.dimensione+1) * sizeof (
                double*));
182     relazione.seconda_stringa [relazione.
        dimensione - 1] = (char *) malloc (2);
183     fflush (stdin);
184     temporaneo='a';
185     i=0;
186     while (temporaneo != '\n')
187     {
188         temporaneo = getc (stdin);
189         relazione.seconda_stringa [relazione.
            dimensione - 1] = (char*) realloc (
                relazione.seconda_stringa[relazione
                    .dimensione-1],
190                 (i+1) * sizeof (char*));

```

```

191         relazione.seconda_stringa [relazione.
           dimensione - 1] [i] = temporaneo;
192         i++;
193     }
194
195     /*Imposto ora il carattere di terminazione
           a \0 dato che adesso \n*/
196     relazione.seconda_stringa [relazione.
           dimensione - 1] [i - 1] = '\0';
197
198     /*Chiedo all'utente se ci sono altre
           coppie*/
199
200     while (acquisizione_finita < 0 ||
           acquisizione_finita > 1 ||
           risultato_lettura != 1)
201     {
202
203         printf ("\nVuoi acquisire un'altra
           coppia? immetti 1 per uscire, 0 per
           continuare\n");
204         risultato_lettura = scanf ("%d",&
           acquisizione_finita);
205     }
206 }
207 }
208
209 relazione.insieme_b = -1;
210 risultato_lettura = 0;
211
212 printf ("\nCi sono elementi del secondo insieme\n
           che non fanno parte di nessuna coppia ordinata
           ?\n");
213 printf ("\n1) si\n2) no\n\nscelta: ");
214 while ((relazione.insieme_b < 0) || (relazione.
           insieme_b > 2) || risultato_lettura != 1)
215 {
216     fflush (stdin);
217     risultato_lettura = scanf ("%d",&relazione.
           insieme_b);
218 }
219
220 relazione.insieme_a = -1;
221 risultato_lettura = 0;

```

```

222
223     printf ("\n_Ci_sono_elementi_del_primo_insieme\n_
        che_non_fanno_parte_di_nessuna_coppia_ordinata
        ?\n");
224     printf ("\n_1_si\n_2_no\n\n_scelta:_");
225     while ((relazione.insieme_a < 0) || (relazione.
        insieme_a > 2) || risultato_lettura != 1)
226     {
227         fflush (stdin);
228         risultato_lettura = scanf("%d",&relazione.
            insieme_a);
229
230     }
231
232     printf ("\n\n_..._Acquisizione_Terminata_...\n\n
        ");
233     return (relazione);
234 }
235
236 /******FUNZIONE DI STAMPA
        *****/
237
238 void stampa (rel_bin stampa)
239 {
240
241     int i = 0;
242
243     printf ("\n_La_relazione_binaria_e ':");
244     printf ("\n\n_{");
245
246     /******Stampa per coppie numeriche *****/
247
248     if (stampa.controllo == 1)
249     {
250         while (i < stampa.dimensione)
251         {
252
253             printf ("(%.2lf,%.2lf)",stampa.
                primo_termine[i],stampa.secondo_termine
                [i]);
254             if (i+1 != stampa.dimensione)
255                 printf ("_;");
256             i++;
257         }

```

```

258     }
259
260     /******Stampa per coppie non numeriche *****/
261
262     if (stampa.controllo == 2)
263     {
264         while (i < stampa.dimensione)
265         {
266             printf ("%s,%s", stampa.prima_stringa[i],
267                     stampa.seconda_stringa[i]);
268             if (i+1 != stampa.dimensione)
269                 printf (" ";
270             i++;
271         }
272     }
273
274     /****** Fine Stampa *****/
275
276     printf ("}\n");
277     printf ("\n\n... _Stampa_Terminata_...\n\n");
278
279 }
280
281 /******FUNZIONE DI VERIFICA DI RELAZIONI D
282 'ORDINE'******/
283
284 int ordine_parziale (rel_bin verifica)
285 {
286     int riflessivita ,
287         transitivita ,
288         antisimmetria ,
289         parziale;
290
291     /*STAMPO LE PROPIETA' DELLA RELAZIONE*/
292
293     printf ("\n\nLa relazione:\n\n");
294
295     /****** Chiamo le funzioni per poter stabilire
296 le propiet ******/
297     riflessivita = controllo_riflessivita (verifica);

```

```

297     controllo_simmetria(verifica);
298     antisimmetria = controllo_antisimmetria (verifica)
        ;
299     transitivita = controllo_transitivita (verifica);
300
301     /****** Controllo se rispetta le propriet
        per essere una relazione d'ordine parziale
        ******/
302     if (transitivita == 1 && antisimmetria == 1 &&
        riflessivita == 1)
303     {
304         parziale = 1;
305         printf ("\n_Quindi_e'una_relazione_d'ordine_
            parziale\n\n");
306     }
307     else
308     {
309
310         printf ("\n_Non_e'una_relazione_d'ordine_
            parziale_in_quanto_non_rispetta_tutte_le_
            propieta'\n");
311         parziale = 0;
312     }
313     if (transitivita == 0)
314         printf ("\n_manca_la_propieta'di_transitivita
            '\n");
315     if (antisimmetria == 0)
316         printf ("\n_manca_la_propieta'di_antisimmetria
            \n");
317     if (riflessivita == 0)
318         printf ("\n_manca_la_propieta'di_riflessivita
            '\n");
319     /****** Fine controllo Ordine Parziale
        ******/
320
321     printf ("\n\n_..._Controllo_Ordine_Parziale_
        Terminato_... \n\n\n\n");
322     return (parziale);
323 }
324
325
326 /******FUNZIONE PER CONTROLLARE LA RIFLESSIVIT
        ******/
327

```

```

328 int controllo_riflessivita (rel_bin verifica)
329 {
330
331     int i ,
332         j ,
333         k ,
334         riscontro ,
335         secondo_riscontro ,
336         riflessivita ;
337
338     riflessivita = 1;
339     i = 0;
340     j = 0;
341     k = 0;
342     riscontro = 0;
343     secondo_riscontro = 0;
344
345     /* Verifica riflessivit */
346
347     /* Definizione: una relazione per la quale esiste
        almeno un elemento che non e' in relazione con
        s stesso non soddisfa la definizione di
        riflessivit */
348
349     while ( (i < verifica.dimensione) && (k < verifica
        .dimensione))
350     {
351
352         /* Verifica riflessivit per numeri */
353
354         if (verifica.controllo == 1)
355         {
356             riscontro = 0;
357             secondo_riscontro = 0;
358             if (verifica.primo_termine[i] == verifica.
                secondo_termine[i])
359                 riscontro++; /**** Controllo se c'
                    stato un riscontro a,a*****/
360             secondo_riscontro++;
361             if (riscontro != 0)
362             {
363                 i++;
364                 k++;
365             }

```



```

366      /**/
367      else
368      {
369          j=0;
370          riscontro = 0;
371          secondo_riscontro = 0;
372
373          /****** Controllo la
riflessivit per gli elementi del
primo insieme
******/
374
375          while (j < verifica.dimensione)
376          {
377              if (j == i)
378                  j++;
379              else
380              {
381                  if (verifica.primo_termine[i]
382                      == verifica.primo_termine[j])
383                      if (verifica.primo_termine
384                          [j] == verifica.
385                          secondo_termine[j])
386                          riscontro++;
387
388                  j++;
389              }
390          }
391
392          j = 0;
393
394          /****** Controllo la
riflessivit per gli elementi del
secondo insieme
******/
395
396          while (j < verifica.dimensione)
397          {
398              if (j == k)
399                  j++;
400              else
401              {

```

```

399         if (verifica.secondo_termine[k
                ] == verifica.
                    secondo_termine[j])
400             if (verifica.primo_termine
                    [j] == verifica.
                        secondo_termine[j])
                            secondo_riscontro++;
401
402
403             j++;
404         }
405     }
406     if (riscontro != 0)
407         i++;
408
409     /**** Se non c'è stato un riscontro di
        riflessivit esco e imposto la
        riflessivit a 0 *****/
410
411     else
412     {
413         i=verifica.dimensione;
414         riflessivita = 0;
415     }
416
417     if (secondo_riscontro != 0)
418         k++;
419
420     else
421     {
422         k=verifica.dimensione;
423         riflessivita = 0;
424     }
425 }
426
427 }
428
429 /****** VERIFICA RIFLESSIVIT PER
        STRINGHE *****/
430
431 if (verifica.controllo == 2)
432 {
433     riscontro = 0;
434     secondo_riscontro = 0;

```

```

435         if (strcmp (verifica.prima_stringa[i],
436                     verifica.seconda_stringa[i]) == 0)
437             riscontro++;
438         secondo_riscontro++;
439         if (riscontro != 0)
440         {
441             i++;
442             k++;
443         }
444     else
445     {
446         j=0;
447         riscontro = 0;
448         secondo_riscontro = 0;
449
450         /****** Controllo la
451             riflessivit per gli elementi del
452             primo insieme
453             ******/
454
455         while (j < verifica.dimensione)
456         {
457             if (j == i)
458                 j++;
459             else
460             {
461                 if (strcmp (verifica.
462                             prima_stringa[i],verifica.
463                             prima_stringa[j]) == 0)
464                     if (strcmp (verifica.
465                                 prima_stringa[j],
466                                 verifica.
467                                 seconda_stringa[j]) ==
468                                 0)
469                         riscontro++;
470
471                 j++;
472             }
473         }
474
475         j = 0;

```

```

468      /****** Controllo la
          riflessivit per gli elementi del
          secondo insieme
          ******/
469
470      while (j < verifica.dimensione)
471      {
472          if (j == k)
473              j++;
474          else
475          {
476              if (strcmp (verifica .
                          seconda_stringa[k], verifica
                          .seconda_stringa[j]) == 0)
477                  if (strcmp (verifica .
                              prima_stringa[j],
                              verifica .
                              seconda_stringa[j]) ==
                              0)
478                      secondo_riscontro++;
479
480                      j++;
481              }
482          }
483          if (riscontro != 0)
484              i++;
485
486          else
487          {
488              i=verifica.dimensione;
489              riflessivita = 0;
490          }
491
492          if (secondo_riscontro != 0)
493              k++;
494
495          else
496          {
497              k=verifica.dimensione;
498              riflessivita = 0;
499          }
500      }
501
502  }

```

```

503
504     }
505     /* Relazione vuota */
506
507     if(verifica.controllo == 3)
508         riflessivita=0;
509
510     /****** Controllo se    riflessiva
511                *****/
512
513     if (riflessivita == 1)
514         printf ("L_e'riflessiva\n");
515     else
516         printf ("L_non_e'riflessiva\n");
517
518     /****** Fine riflessivita
519                *****/
520
521     return (riflessivita);
522 }
523
524 /****** FUNZIONE PER CONTROLLARE
525                LA SIMMETRIA *****/
526
527 /****** Definizione: In matematica, una
528                relazione binaria R in un insieme X **/
529 /****** simmetrica se e solo se, presi due
530                elementi qualsiasi a e b, vale che **/
531 /****** se a    in relazione con b allora anche
532                b    in relazione con a. *****/
533
534 int controllo_simmetria (rel_bin verifica)
535 {
536
537     int i,
538         j,
539         riscontro,
540         simmetria;
541
542     simmetria = 1;
543
544

```

```

541     i = 0;
542     j = 0;
543     riscontro = 0;
544
545     /*controllo della simmetria per numeri*/
546
547     if (verifica.controllo == 1)
548     {
549
550         while ( i < verifica.dimensione)
551         {
552
553             j = 0;
554             while ( j < verifica.dimensione)
555             {
556
557                 if (verifica.primo_termine[i] ==
558                     verifica.secondo_termine[j])
559                     if (verifica.primo_termine[j] ==
560                         verifica.secondo_termine[i])
561                         riscontro++;
562
563                 j++;
564             }
565
566             if (riscontro == 0)
567             {
568                 j = verifica.dimensione;
569                 i = verifica.dimensione;
570                 simmetria = 0;
571             }
572             riscontro = 0;
573             i++;
574         }
575     }
576
577     /*controllo della simmetria per stringhe*/
578
579     if (verifica.controllo == 2)
580     {
581
582         while ( i < verifica.dimensione)
583         {

```

```

583         j = 0;
584         while ( j < verifica.dimensione)
585         {
586
587             if (strcmp (verifica.prima_stringa[i],
588                         verifica.seconda_stringa[j]) == 0 )
589                 if (strcmp (verifica.prima_stringa
590                             [j],verifica.seconda_stringa[i
591                             ]) == 0 )
592                     riscontro++;
593
594             j++;
595         }
596
597         if (riscontro == 0)
598         {
599             j = verifica.dimensione;
600             i = verifica.dimensione;
601             simmetria = 0;
602         }
603         riscontro = 0;
604         i++;
605     }
606
607     /* Relazione Vuota */
608
609     if (verifica.controllo == 3)
610     {
611         printf ("L' e' simmetrica\n");
612         simmetria = 1;
613     }
614
615     /****** Controllo se la simmetria  stata
616     verificata *****/
617
618     if(verifica.controllo != 3)
619     {
620         if (simmetria == 1)
621             printf ("L' e' simmetrica\n");
622         else
623             printf ("L' e' asimmetrica\n");
624     }
625
626     /****** Fine controllo simmetria *****/

```

```

623     return (simmetria);
624 }
625
626
627
628 /* FUNZIONE PER CONTROLLARE LA TRANSITIVIT */
629
630 /****** Definizione: In matematica, una relazione
        binaria R in un insieme X transitiva se e solo se
631 per ogni a, b, c appartenenti ad X, se a in
        relazione con b e b in relazione con c,
        allora
632 a in relazione con c.*****/
633
634
635 int controllo_transitivita (rel_bin verifica)
636 {
637
638     int i,
639         j,
640         k,
641         transitivita;
642
643     /*IMPOSTO LA TRANSITIVITA INIZIALMENTE COME VERA E
        AZZERO I CONTATORI*/
644     transitivita = 1;
645     i = 0;
646     j = 0;
647     k = 0;
648
649     /*VERIFICA TRANSITIVIT PER NUMERI*/
650
651
652     if (verifica.controllo == 1)
653     {
654
655         while (i < verifica.dimensione)
656         {
657             j = 0;
658
659             while (j < verifica.dimensione)
660             {
661                 k=0;
662

```



```

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

```

```

699     {
700
701
702         while (i < verifica.dimensione)
703         {
704             j = 0;
705
706             while (j < verifica.dimensione)
707             {
708                 k=0;
709
710                 if (strcmp (verifica.seconda_stringa[i
711                             ],verifica.prima_stringa[j]) == 0)
712                 {
713                     transitivita = 0;
714
715                     while (k < verifica.dimensione)
716                     {
717                         if (strcmp (verifica .
718                                     prima_stringa[i],verifica .
719                                     prima_stringa[k]) == 0)
720                         {
721                             if (strcmp (verifica .
722                                         seconda_stringa[k],
723                                         verifica .
724                                         seconda_stringa[j]) ==
725                                         0)
726                             {
727                                 transitivita = 1;
728                                 k = verifica .
729                                     dimensione;
730                             }
731                         }
732                     }
733                 }
734             }
735         }
736     }

```

```

735             j++;
736         }
737
738         i++;
739     }
740
741 }
742 /* Relazione Vuota */
743
744 if (verifica.controllo == 3)
745 {
746     transitivita = 1;
747 }
748
749 /****** Controllo se la relazione  Transitiva
       *****/
750
751 if (transitivita == 1)
752     printf ("L' e' transitiva\n");
753
754 else
755     printf ("L non e' transitiva\n");
756
757 /****** Fine controllo Transittivit
       *****/
758
759 return (transitivita);
760
761 }
762
763 /****** Dicotomia *****/
764
765 int controllo_dicotomia (rel_bin verifica)
766 {
767
768     int i,j,k;
769     int numero_elementi;
770     int dicotomia = 0;
771     int dimensione;
772     int riscontro;
773     int secondo_riscontro;
774     i=0;
775     j=0;
776     k=i-1;

```

```

777     riscontro = 0;
778     dimensione = verifica.dimensione;
779
780     /****** Dicotomia per numeri *****/
781
782     if (verifica.controllo == 1)
783     {
784
785         /****** Conto il numero delle coppie
786         esistenti (scarto le coppie uguali)
787         *****/
788
789         while ( i < verifica.dimensione)
790         {
791             k = i-1;
792             j = i+1;
793             secondo_riscontro = 0;
794
795             if (i>0)
796             {
797                 while ( k >= 0 )
798                 {
799                     if (verifica.primo_termine[i] ==
800                     verifica.primo_termine[k])
801                     {
802                         if (verifica.secondo_termine[i
803                         ] == verifica.
804                         secondo_termine[k])
805                             secondo_riscontro = 1;
806                     }
807                     k--;
808                 }
809             }
810
811             if (secondo_riscontro != 1)
812             {
813                 while ( j < verifica.dimensione)
814                 {
815                     if (verifica.primo_termine[i] ==
816                     verifica.primo_termine[j])
817                     if (verifica.secondo_termine[i
818                     ] == verifica.
819                     secondo_termine[j])
820                     {

```

```

813                                     dimensione--;
814                                     }
815                                     j++;
816                                 }
817                            }
818                            i++;
819                    }
820
821
822                    i=0;
823                    j=0;
824                    k=0;
825                    numero_elementi=0;
826                    riscontro = 0;
827                    /****** Conto il numero degli
                        elementi distinti esistenti *****
                        */
828
829                    while (i<verifica.dimensione)
830                    {
831                        k=i-1;
832                        secondo_riscontro = 0;
833
834                        while (k >= 0)
835                        {
836                            if (verifica.primo_termine[i] ==
                                verifica.primo_termine[k])
837                                secondo_riscontro = 1;
838                            k--;
839                        }
840                        if (secondo_riscontro != 1)
841                        {
842                            if (verifica.primo_termine[i] ==
                                verifica.secondo_termine[i])
843                                riscontro++;
844
845                        }
846                        i++;
847                    }
848
849                    numero_elementi = riscontro;
850
851                    /****** Conto quanti dovrebbero essere
                        gli elementi per avere la dicotomia

```

```

852          *****/
853      while (numero_elementi > 0)
854      {
855          numero_elementi--;
856          riscontro = riscontro + numero_elementi;
857      }
858  }
859
860  /***** VERIFICA DICOTOMICA PER
      STRINGHE *****/
861
862  if (verifica.controllo == 2)
863  {
864
865      /***** Conto il numero delle coppie
          esistenti (scarto le coppie uguali)
          *****/
866
867      while ( i < verifica.dimensione)
868      {
869          k = i-1;
870          j = i+1;
871          secondo_riscontro = 0;
872          if (i>0)
873          {
874              while ( k >= 0 )
875              {
876                  if ( (strcmp (verifica.
                        prima_stringa[i], verifica.
                        prima_stringa[k])) == 0)
877                  {
878                      if ( (strcmp (verifica.
                                seconda_stringa[i], verifica
                                .seconda_stringa[k])) == 0)
879                          secondo_riscontro = 1;
880                  }
881                  k--;
882              }
883          }
884
885          if (secondo_riscontro != 1)
886          {
887              while ( j < verifica.dimensione)

```

```

888         {
889             if ( (strcmp (verifica .
                        prima_stringa[i], verifica .
                        prima_stringa[j])) == 0)
890                 if ( (strcmp (verifica .
                        seconda_stringa[i], verifica .
                        seconda_stringa[j])) == 0)
891                     {
892                         dimensione--;
893                     }
894                 j++;
895             }
896         }
897         i++;
898     }
899
900     i=0;
901     k=0;
902     j=0;
903     numero_elementi = 0;
904     /****** Conto il numero degli
905     elementi distinti esistenti *****
906     */
907     while (i<verifica.dimensione)
908     {
909         k=i-1;
910         secondo_riscontro = 0;
911
912         while (k >= 0)
913         {
914             if ( (strcmp (verifica.prima_stringa[i]
                        ], verifica.prima_stringa[k])) == 0)
915                 secondo_riscontro = 1;
916             k--;
917         }
918         if (secondo_riscontro != 1)
919         {
920             if ( (strcmp (verifica.prima_stringa[i]
                        ], verifica.seconda_stringa[i])) ==
                        0)
921                 numero_elementi++;
922

```

```

923         }
924         i++;
925     }
926     riscontro = numero_elementi;
927
928     /****** Conto quanti dovrebbero essere
gli elementi per avere la dicotomia
******/
929
930     while (numero_elementi > 0)
931     {
932
933         numero_elementi--;
934         riscontro = riscontro + numero_elementi;
935
936     }
937
938 }
939
940 /****** Verifico se la dicotomia
verificata ******/
941
942 if (dimensione == riscontro)
943     dicotomia = 1;
944
945 if (dicotomia == 1 )
946     printf ("L'ordine totale e' dicotomica\n\n");
947
948 else
949     printf ("L'ordine totale non e' dicotomica\n\n");
950
951 /****** Fine verifica dicotomia
******/
952
953 return (dicotomia);
954 }
955
956 /*Funzione di verifica dell'ordine totale*/
957
958
959 void ordine_totale (rel_bin verifica)
960 {
961
962     int parziale ,

```



```

963         dicotomia;
964
965     dicotomia=2;
966     parziale = ordine_parziale (verifica);
967     if (parziale == 1)
968         dicotomia = controllo_dicotomia (verifica);
969
970     if (parziale == 0)
971         printf ("\n_l'ordine_non_e'totale_in_quanto_
          non_e'nemmeno_parziale");
972
973     if (dicotomia == 0)
974         printf ("\n_l'ordine_non_e'totale_in_quanto_
          non_viene_rispettata_la_propieta'di_
          dicotomia");
975
976     if (dicotomia == 1 && parziale == 1)
977         printf ("\n_Quindi_e'una_relazione_d'ordine_
          totale");
978
979     printf ("\n\n..._Controllo_Ordine_Totale_
          Terminato...\n\n\n");
980 }
981
982 /*Funzione che stabilisce se e'una relazione di
          equivalenza o meno*/
983
984 void relazione_equivalenza (rel_bin verifica)
985 {
986
987     int riflessivita;
988     int simmetria;
989     int transitivita;
990
991     riflessivita = controllo_riflessivita (verifica);
992     simmetria = controllo_simmetria (verifica);
993     controllo_antisimmetria (verifica);
994     transitivita = controllo_transitivita (verifica);
995
996     if (riflessivita == 1 && simmetria == 1 &&
          transitivita == 1)
997         printf ("\n_Quindi_e'una_relazione_di_
          equivalenza\n");
998

```

```

999     if (riflessivita == 0)
1000         printf ("\nQuindi non e' una relazione di
                equivalenza perche' non riflessiva\n");
1001
1002     if (simmetria == 0)
1003         printf ("\nQuindi non e' una relazione di
                equivalenza perche' non simmetrica\n");
1004
1005     if (transitivita == 0)
1006         printf ("\nQuindi non e' una relazione di
                equivalenza perche' non transitiva\n");
1007 }
1008
1009 /*Funzione che stabilisce se la relazione binaria
        acquisita e'una funzione matematica*/
1010
1011 void controllo_funzione (rel_bin verifica)
1012 {
1013
1014     int i;
1015     int k;
1016     int termini_diversi;
1017     int termini_uguali_prima;
1018     int termini_uguali_dopo;
1019     int errore;
1020
1021     if (verifica.controllo == 1)
1022     {
1023
1024         i=0;
1025         errore=0;
1026         termini_diversi=0;
1027         termini_uguali_dopo=0;
1028         termini_uguali_prima=0;
1029         while (i < verifica.dimensione)
1030         {
1031             k=verifica.dimensione-1;
1032             termini_uguali_dopo=termini_uguali_prima;
1033             while (k > i)
1034             {
1035                 if (verifica.primo_termine[i] ==
                        verifica.primo_termine[k])
1036                     {

```

```

1037         if (verifica.secondo_termine[i] !=
1038             verifica.secondo_termine[k])
1039         {
1040             errore=1;
1041             printf ("\n_Nel_%d_elemento_c'
1042                 e'un_errore_che_impedisce_
1043                 alla_relazione_binaria\n",k
1044                 +1);
1045             printf ("_di_essere_una_
1046                 funzione\n");
1047             k=i;
1048             i=verifica.dimensione;
1049         }
1050         if (verifica.secondo_termine[i] ==
1051             verifica.secondo_termine[k])
1052             termini_uguali_dopo++;
1053     }
1054     k--;
1055 }
1056 if (errore == 0 && termini_uguali_dopo ==
1057     termini_uguali_prima)
1058     termini_diversi++;
1059
1060 termini_uguali_prima = termini_uguali_dopo
1061     ;
1062 i++;
1063 }
1064 if (errore == 0 && (termini_diversi == (
1065     verifica.dimensione - termini_uguali_prima)
1066 ))
1067 {
1068     if(verifica.insieme_a == 2)
1069         printf ("\n_La_relazione_binaria_e'una
1070             _funzione_totale\n");
1071     else
1072         printf ("\n_La_relazione_binaria_ _una
1073             _funzione_parziale\n");
1074     controllo_biiettivita (verifica);
1075 }
1076 else
1077     printf ("\n_La_relazione_binaria_non_e'una
1078         _funzione\n");
1079 }

```

```

1068      /****** Controllo se c'è una funzione per
           stringhe (le stringhe sono considerate come
           costanti di diverso valore) *****/
1069
1070      if (verifica.controllo == 2)
1071      {
1072
1073          i=0;
1074          errore=0;
1075          termini_diversi=0;
1076          termini_uguali_dopo=0;
1077          termini_uguali_prima=0;
1078          while (i < verifica.dimensione)
1079          {
1080              k=verifica.dimensione-1;
1081              termini_uguali_dopo=termini_uguali_prima;
1082              while (k > i)
1083              {
1084                  if ( (strcmp (verifica.prima_stringa[i]
1085                      ],verifica.prima_stringa[k])) == 0)
1086                  {
1087                      if ( (strcmp (verifica.
1088                          seconda_stringa[i],verifica.
1089                          seconda_stringa[k])) != 0)
1090                      {
1091                          errore=1;
1092                          printf ("\\n_Nel_%d_elemento_c'
1093                              e'un_errore_che_impedisce_
1094                              alla_relazione_binaria\\n",k
1095                              +1);
1096                          printf ("_di_essere_una_
1097                              funzione\\n");
1098                          k=i;
1099                          i=verifica.dimensione;
1100                      }
1101                      else
1102                          termini_uguali_dopo++;
1103                  }
1104                  k--;
1105              }
1106          }
1107          if (errore == 0 && termini_uguali_dopo ==
1108              termini_uguali_prima)
1109              termini_diversi++;
1110
1111

```

```

1102         termini_uguali_prima = termini_uguali_dopo
1103         ;
1104         i++;
1105     }
1106     if (errore == 0 && (termini_diversi == (
1107         verifica.dimensione - termini_uguali_prima)
1108     ))
1109     {
1110         if(verifica.insieme_a == 2)
1111             printf ("\nLa relazione binaria e' una
1112                 _funzione totale\n");
1113         else
1114             printf ("\nLa relazione binaria e' una
1115                 _funzione parziale\n");
1116         controllo_biiettivita (verifica);
1117     }
1118     else
1119         printf ("\nLa relazione binaria non e' una
1120             _funzione\n");
1121 }
1122
1123 /* Relazione Vuota */
1124 if (verifica.controllo == 3)
1125     printf ("\nLa relazione vuota non e' una
1126         _funzione\n");
1127 printf ("\n\n... _Controllo _Funzione _Terminato _
1128     ... \n\n\n\n");
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500

```

```

1137
1138     if (verifica.controllo == 1)
1139     {
1140
1141         i=0;
1142         errore=0;
1143         termini_diversi=0;
1144         termini_uguali_dopo=0;
1145         termini_uguali_prima=0;
1146
1147         while (i < verifica.dimensione)
1148         {
1149
1150             k=verifica.dimensione-1;
1151             termini_uguali_dopo=termini_uguali_prima;
1152             while (k > i)
1153             {
1154
1155                 if (verifica.secondo_termine[i] ==
1156                     verifica.secondo_termine[k])
1157                 {
1158
1159                     if (verifica.primo_termine[i] !=
1160                         verifica.primo_termine[k])
1161                     {
1162
1163                         errore=1;
1164                         printf (" \n_Nel_%d_elemento_c'
1165                             e' un_errore_che_impedisce_
1166                             alla_funzione\n", k+1);
1167                         printf (" _di_essere_iniettiva\
1168                             n");
1169                         k=i;
1170                         i=verifica.dimensione;
1171                     }
1172                     if (verifica.primo_termine[i] ==
1173                         verifica.primo_termine[k])
1174                         termini_uguali_dopo++;
1175                 }
1176                 k--;
1177             }
1178             if (errore == 0 && termini_uguali_dopo ==
1179                 termini_uguali_prima)
1180                 termini_diversi++;

```

```

1174
1175         termini_uguali_prima = termini_uguali_dopo
1176         ;
1177         i++;
1178     }
1179     if (errore == 0 && (termini_diversi == (
1180         verifica.dimensione - termini_uguali_prima)
1181         ))
1182     {
1183         printf ("\nLa funzione e' iniettiva\n");
1184         iniettivita = 1;
1185     }
1186     else
1187         printf ("\nLa funzione non e' iniettiva\n"
1188             );
1189
1190
1191     }
1192
1193     /****** Controllo iniettivita' per stringhe
1194     ******/
1195
1196     if (verifica.controllo == 2)
1197     {
1198         i=0;
1199         errore=0;
1200         termini_diversi=0;
1201         termini_uguali_dopo=0;
1202         termini_uguali_prima=0;
1203
1204         while (i < verifica.dimensione)
1205         {
1206             k=verifica.dimensione-1;
1207             termini_uguali_dopo=termini_uguali_prima;
1208             while (k > i)
1209             {
1210                 if ( (strcmp (verifica.seconda_stringa
1211                     [i], verifica.seconda_stringa[k]))
1212                     == 0)
1213                 {
1214                     if ( (strcmp (verifica.
1215                         prima_stringa[i], verifica.
1216                         prima_stringa[k])) != 0)

```

```

1209         {
1210             errore=1;
1211             printf ("\n_Nel_%d_elemento_c'
                    e'un_errore_che_impedisce_
                    alla_funzione\n",k+1);
1212             printf ("_di_essere_iniettiva\
                    n");
1213             k=i;
1214             i=verifica.dimensione;
1215         }
1216         if ( (strcmp (verifica.
                    prima_stringa[i],verifica.
                    prima_stringa[k])) == 0)
                    termini_uguali_dopo++;
1217     }
1218 }
1219
1220     k--;
1221 }
1222 if (errore == 0 && termini_uguali_dopo ==
    termini_uguali_prima)
    termini_diversi++;
1223
1224     termini_uguali_prima = termini_uguali_dopo
    ;
1225     i++;
1226 }
1227
1228 if (errore == 0 && (termini_diversi == (
    verifica.dimensione - termini_uguali_prima)
    ))
1229 {
1230     printf ("\n_La_funzione_e'iniettiva");
1231     iniettivita = 1;
1232 }
1233 else
1234     printf ("\n_La_funzione_non_e'iniettiva");
1235 }
1236
1237 return (iniettivita);
1238 }
1239
1240 /*****FUNZIONE PER IL controllo DELLA
    SURIETTIVITA *****/
1241
1242 int controllo_suriettivita (rel_bin verifica)

```



```

1243 {
1244     int suriattivita;
1245
1246     if (verifica.insieme_b == 2)
1247     {
1248         suriattivita = 1;
1249         printf ("\nla funzione e' suriattivita");
1250     }
1251
1252     else
1253     {
1254         suriattivita = 0;
1255         printf ("\nla funzione non e' suriattivita");
1256     }
1257
1258     return (suriattivita);
1259 }
1260
1261 /******FUNZIONE PER IL controllo DELLA
BIETTIVITA'******/
1262
1263 void controllo_biattivita (rel_bin verifica)
1264 {
1265
1266     int    suriattivita ,
1267           iniattivita;
1268
1269     suriattivita = controllo_suriattivita (verifica);
1270     iniattivita = controllo_iniattivita (verifica);
1271
1272
1273     if ( suriattivita == 1 && iniattivita == 1)
1274         printf ("\nla funzione e' biattivita");
1275     else
1276         printf ("\nla funzione non e' biattivita");
1277     return;
1278 }
1279
1280
1281 int controllo_antisimmetria (rel_bin verifica)
1282 {
1283
1284     int i ,
1285         j ,

```

```

1286         riscontro ,
1287         antisimmetria;
1288
1289     antisimmetria = 1;
1290
1291
1292     i = 0;
1293     j = 0;
1294     riscontro = 1;
1295
1296     /* controllo della antisimmetria per numeri*/
1297
1298     if (verifica.controllo == 1)
1299     {
1300
1301         while ( i < verifica.dimensione)
1302         {
1303
1304             j = 0;
1305             while ( j < verifica.dimensione)
1306             {
1307
1308                 if (verifica.primo_termine[i] ==
1309                     verifica.secondo_termine[j])
1310                     if (verifica.primo_termine[j] ==
1311                         verifica.secondo_termine[i])
1312                         if (verifica.primo_termine[i]
1313                             == verifica.primo_termine[j]
1314                             )
1315                             riscontro++;
1316                 else
1317                     riscontro = 0;
1318             j++;
1319         }
1320
1321         if (riscontro == 0)
1322         {
1323             j = verifica.dimensione;
1324             i = verifica.dimensione;
1325             antisimmetria = 0;
1326         }
1327         i++;
1328     }

```

```

1326     }
1327
1328     /* controllo della antisimmetria per stringhe */
1329
1330     if (verifica.controllo == 2)
1331     {
1332
1333         while ( i < verifica.dimensione)
1334         {
1335
1336             j = 0;
1337             while ( j < verifica.dimensione)
1338             {
1339
1340                 if (strcmp (verifica.prima_stringa[i],
1341                             verifica.seconda_stringa[j]) == 0 )
1342                     if (strcmp (verifica.prima_stringa
1343                                 [j],verifica.seconda_stringa[i
1344                                 ]) == 0 )
1345                         if (strcmp (verifica.
1346                                     prima_stringa[j],verifica.
1347                                     prima_stringa[i]) == 0 )
1348                             riscontro++;
1349                         else
1350                             riscontro=0;
1351
1352                 j++;
1353             }
1354
1355             if (riscontro == 0)
1356             {
1357                 j = verifica.dimensione;
1358                 i = verifica.dimensione;
1359                 antisimmetria = 0;
1360             }
1361             i++;
1362         }
1363     }
1364
1365     /***** Controllo se la simmetria stata
1366             verificata *****/
1367
1368     if (antisimmetria == 1)

```

```

1364         printf ("_ee'antisimmetrica\n");
1365     else
1366         printf ("_non_ee'antisimmetrica\n");
1367
1368     /****** Fine controllo simmetria *****/
1369
1370     return (antisimmetria);
1371 }

```

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_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:
```

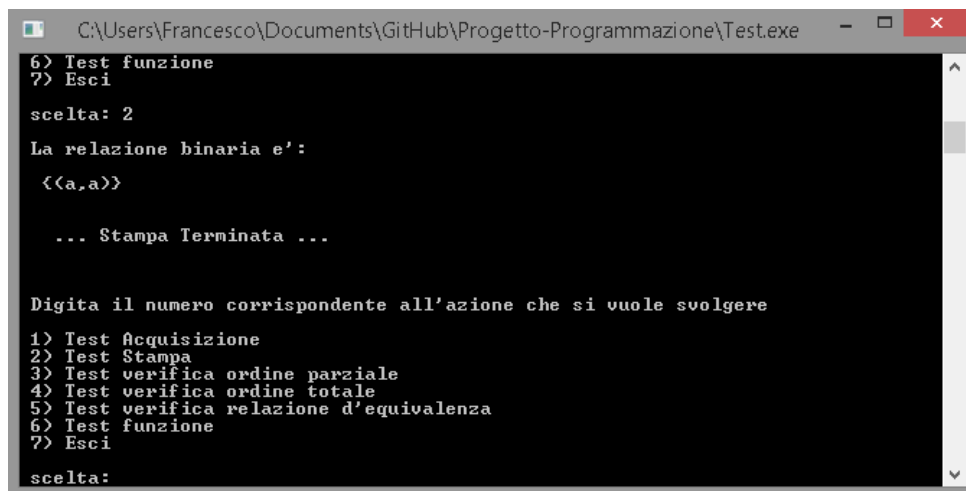
5.6 Test 6:

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

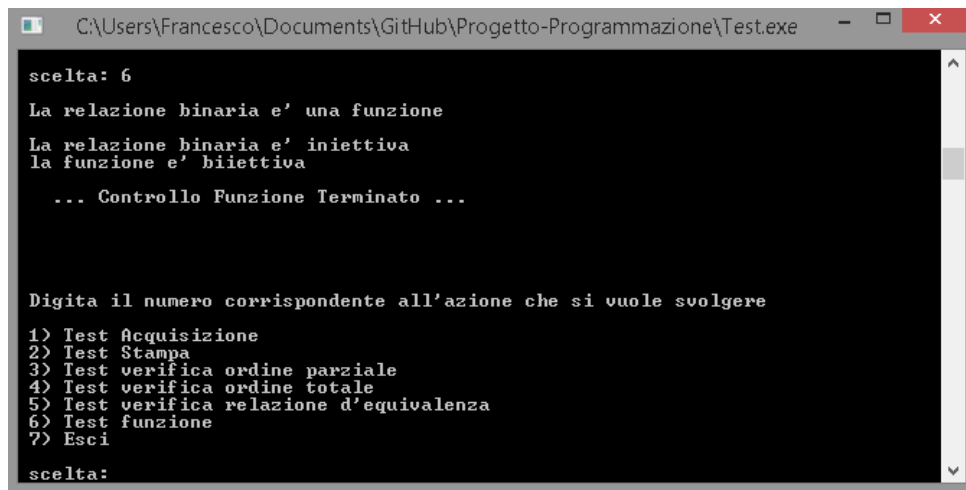
La relazione binaria e':

<<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:
```

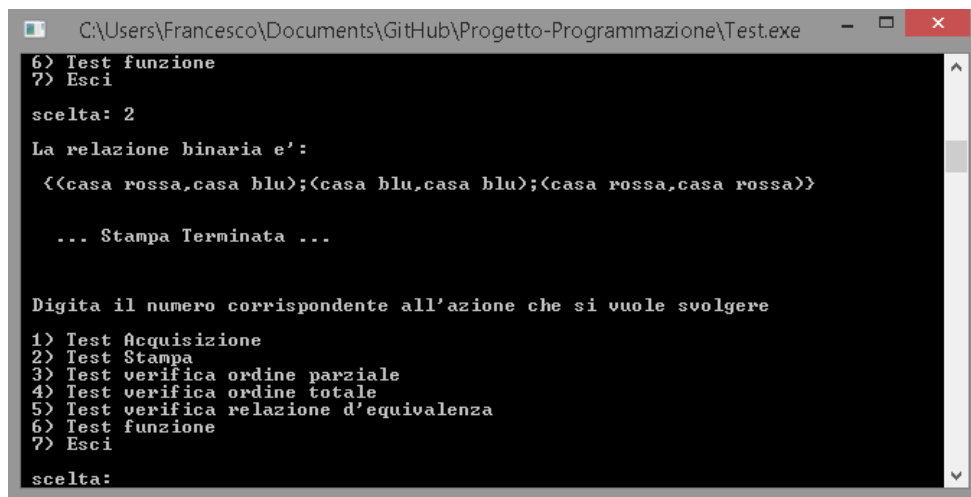
5.7 Test 7:

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':

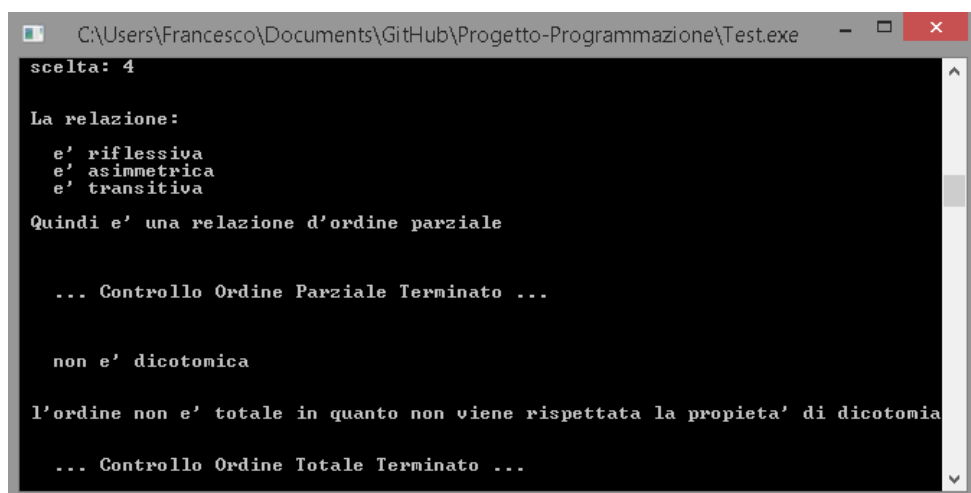
<(casa rossa,casa blu);(casa blu,casa blu);(casa rossa,casa rossa)>

... 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 ...
```

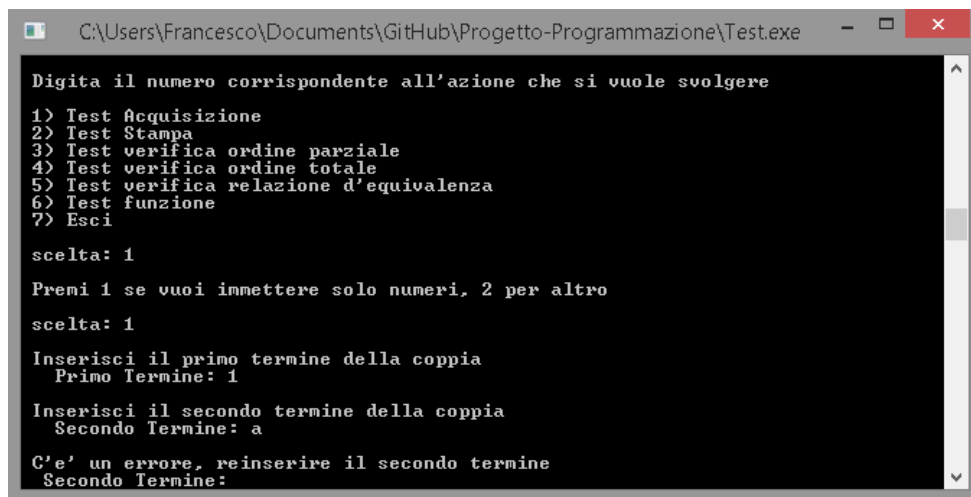
5.8 Test 8:

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:
```

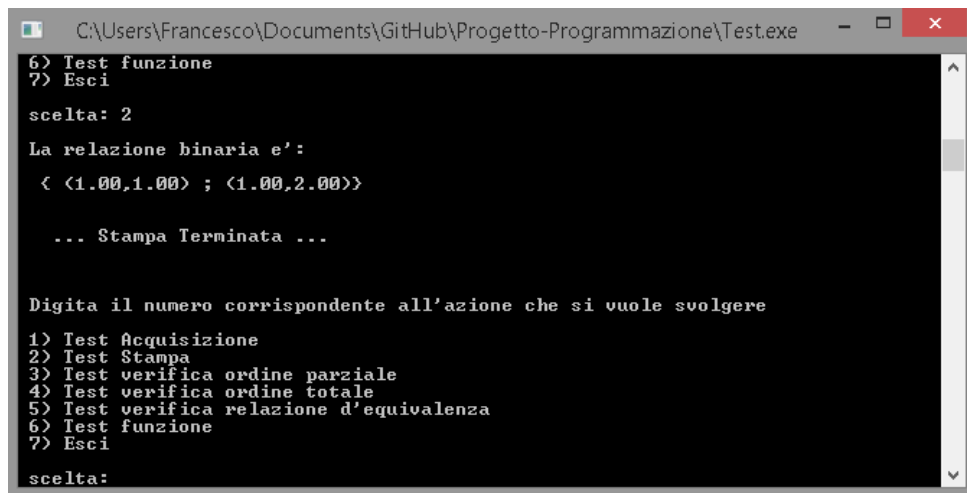

5.9 Test 9:

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;



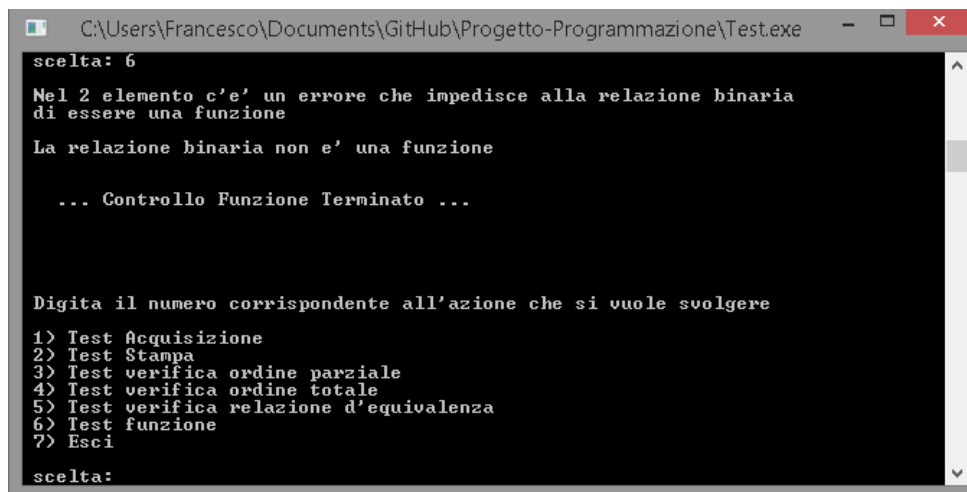
```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

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:

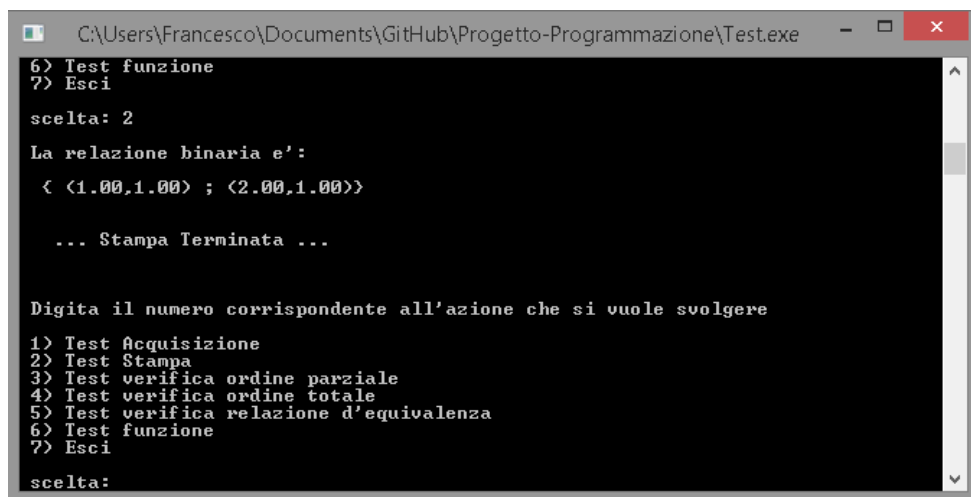
Inputs: (1,1) (2,1)

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



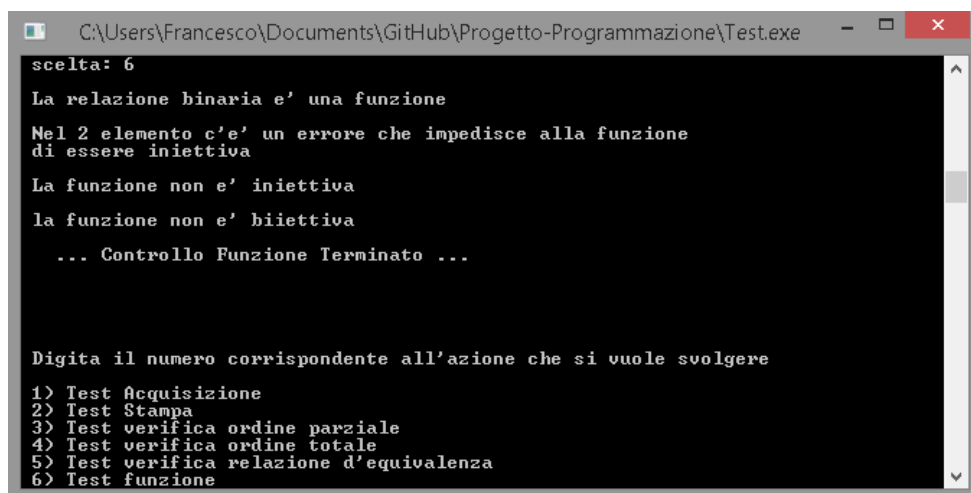
```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

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 tr (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:

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