

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

<b>1</b>	<b>Specifica del Problema</b>	<b>1</b>
<b>2</b>	<b>Analisi del Problema</b>	<b>2</b>
2.1	Input . . . . .	2
2.2	Output . . . . .	2
<b>3</b>	<b>Progettazione dell'Algoritmo</b>	<b>3</b>
3.1	Teoria . . . . .	3
3.2	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
<b>4</b>	<b>Implementazione dell'Algoritmo</b>	<b>11</b>
4.1	Libreria (file .h) . . . . .	11
4.2	Libreria (file .c) . . . . .	12
4.3	Test . . . . .	50
4.4	Makefile . . . . .	53
<b>5</b>	<b>Testing del programma</b>	<b>54</b>
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
<b>6</b>	<b>Verica del programma</b>	<b>64</b>

## 1 Specifica del Problema

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

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

## 2 Analisi del Problema

### 2.1 Input

1. Per l'acquisizione come input abbiamo una relazione binaria del tipo  $(a,b); (a1,b1); (a2,b2); \dots$  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  $(a,b); (a1,b1); (a2,b2); \dots$  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 forzatamente 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 "lib_rel_bin.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");
50         printf("_inserire_stringhe\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,
86 (relazione.dimensione+1) * sizeof (double));
87      relazione.secondo_termine = (double *)
           realloc (relazione.secondo_termine,
88 (relazione.dimensione+1) * sizeof (double));
89      risultato_lettura = 0;
90
91
92      do
93      {
94          /*Acquisisco il primo termine*/
95          if(primo_termine_acquisito == 0)
96          {
97              printf("_Primo_Termine:_");
98              risultato_lettura = scanf("%lf",&
           relazione.primo_termine[
           relazione.dimensione - 1]);
99          }
100
101          if(risultato_lettura == 1)
102              primo_termine_acquisito = 1;
103
104          /*Acquisisco il secondo termine*/
105          if(primo_termine_acquisito == 1)
106          {
107              printf("_Secondo_Termine:_");
108              risultato_lettura = 0;
109              risultato_lettura = scanf("%lf",&
           relazione.secondo_termine[
           relazione.dimensione - 1]);
110          }
111          /*Controllo che i valori siano stati
           letti correttamente e nel caso non
           sia cosi svuoto il buffer*/
112          if(risultato_lettura != 1)
113              do
114                  carattere_non_letto = getchar
           ();

```



```

115         while(carattere_non_letto != '\n')
116             ;
117         if(risultato_lettura == 0 &&
118            primo_termine_acquisito == 0)
119             printf("\nC'e' un errore, \n
120                reinserire il primo termine\n");
121             ;
122         if(risultato_lettura == 0 &&
123            primo_termine_acquisito == 1)
124             printf("\nC'e' un errore, \n
125                reinserire il secondo termine\n
126                ");
127     }
128     while(risultato_lettura != 1);
129
130     /* Chiedo all'utente se ci sono altre
131        coppie*/
132
133     do
134     {
135         printf("\nVuoi acquisire un'altra \n
136            coppia? immetti 1 per uscire, 0 per \n
137            continuare\n");
138         printf("\nscelta: ");
139         risultato_lettura = scanf("%d",
140            &
141            acquisizione_finita
142            );
143         if(acquisizione_finita < 0 ||
144            acquisizione_finita > 1 ||
145            risultato_lettura != 1)
146             do
147             {
148                 carattere_non_letto = getchar
149                     ();
150                 while(carattere_non_letto != '\n')
151                     ;
152             }
153         while(acquisizione_finita < 0 ||
154            acquisizione_finita > 1 ||
155            risultato_lettura != 1);
156
157
158
159
160

```

```

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
169 (relazione.prima_stringa[relazione.dimensione
                -1],
170             (i+1) * sizeof (char*));
171             relazione.prima_stringa[relazione.
                dimensione - 1] [i] = temporaneo;
172             i++;
173         }
174
175         /*Imposto ora il carattere di terminazione
                a \0 dato che adesso \n*/

```

```

176
177     relazione.prima_stringa[relazione.
        dimensione - 1] [i - 1] = '\0';
178
179     /*Acquisisco il secondo termine della
        coppia*/
180
181     printf("___Secondo_Termine:_");
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("\n_Vuoi_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("\n_Ci_sono_elementi_del_secondo_insieme\n_
        che_non_fanno_parte_di_nessuna_coppia_ordinata
        ?\n");
213     printf("\n_1_si_2_no\n\n_scelta:_");
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_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("\nLa 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
263     if(stampa.controllo == 2)
264     {
265         while(i < stampa.dimensione)
266         {
267             printf("(%s,%s)", stampa.prima_stringa[i],
                stampa.seconda_stringa[i]);
268             if(i+1 != stampa.dimensione)
269                 printf(" ; ");
270             i++;
271         }
272     }
273
274     /****** Fine Stampa *****/
275     */
276
277     printf("}\n");
278     printf("\n\n... Stampa Terminata ... \n\n");

```

```

278
279 }
280
281 /******FUNZIONE DI VERIFICA DI RELAZIONI D
'ORDINE******/
282
283 int ordine_parziale(rel_bin verifica)
284 {
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
le propriet ******/
296     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");
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
348     while((i < verifica.dimensione) && (k < verifica.dimensione))
349     {
350

```

```

351      /* Verifica riflessivit per numeri */
352
353      if(verifica.controllo == 1)
354      {
355          riscontro = 0;
356          secondo_riscontro = 0;
357          if(verifica.primo_termine[i] == verifica.
              secondo_termine[i])
358              riscontro++; /* Controllo se c'
                           stato un riscontro a, a */
359          secondo_riscontro++;
360          if(riscontro != 0)
361          {
362              i++;
363              k++;
364          }
365          /**/
366          else
367          {
368              j=0;
369              riscontro = 0;
370              secondo_riscontro = 0;
371
372              /* Controllo la
                 riflessivit per gli elementi del
                 primo insieme
                 *****/
373
374              while(j < verifica.dimensione)
375              {
376                  if(j == i)
377                      j++;
378                  else
379                  {
380                      if(verifica.primo_termine[i]
                          == verifica.primo_termine[j]
                          )
381                          if(verifica.primo_termine[
                              j] == verifica.
                                  secondo_termine[j])
382                              riscontro++;
383
384                      j++;
385                  }

```



```

386     }
387
388     j = 0;
389
390     /****** Controllo la
391         riflessivit per gli elementi del
392         secondo insieme
393         ******/
394
395     while(j < verifica.dimensione)
396     {
397         if(j == k)
398             j++;
399         else
400         {
401             if(verifica.secondo_termine[k]
402                 == verifica.
403                 secondo_termine[j])
404                 if(verifica.primo_termine[
405                     j] == verifica.
406                     secondo_termine[j])
407                     secondo_riscontro++;
408
409             j++;
410         }
411     }
412     if(riscontro != 0)
413         i++;
414
415     /**** Se non c' stato un riscontro di
416         riflessivit esco e imposto la
417         riflessivit a 0 *****/
418
419     else
420     {
421         i=verifica.dimensione;
422         riflessivita = 0;
423     }
424
425     if(secondo_riscontro != 0)
426         k++;
427
428     else
429     {

```

```

421             k=verifica.dimensione;
422             riflessivita = 0;
423         }
424     }
425
426 }
427
428 /****** VERIFICA RIFLESSIVIT PER
429 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
445     else
446     {
447         j=0;
448         riscontro = 0;
449         secondo_riscontro = 0;
450
451         /****** Controllo la
452         riflessivita per gli elementi del
453         primo insieme
454         ******/
455
456         while(j < verifica.dimensione)
457         {
458             if(j == i)
459                 j++;
460             else
461             {
462                 if(strcmp(verifica.prima_stringa[i], verifica.prima_stringa[j]) == 0)

```

```

458             if(strcmp(verifica.
                        prima_stringa[j],
                        verifica.
                        seconda_stringa[j]) ==
                        0)
459                 riscontro++;
460
461             j++;
462         }
463     }
464
465     j = 0;
466
467     /****** Controllo la
        riflessivit per gli elementi del
        secondo insieme
        *****/
468
469     while(j < verifica.dimensione)
470     {
471         if(j == k)
472             j++;
473         else
474         {
475             if(strcmp(verifica.
                        seconda_stringa[k], verifica.
                        seconda_stringa[j]) == 0)
476                 if(strcmp(verifica.
                            prima_stringa[j],
                            verifica.
                            seconda_stringa[j]) ==
                            0)
477                     secondo_riscontro++;
478
479             j++;
480         }
481     }
482     if(riscontro != 0)
483         i++;
484
485     else
486     {
487         i=verifica.dimensione;
488         riflessivita = 0;

```

```

489         }
490
491         if(secondo_riscontro != 0)
492             k++;
493
494         else
495         {
496             k=verifica.dimensione;
497             riflessivita = 0;
498         }
499     }
500
501 }
502
503 }
504 /* Relazione vuota */
505
506 if(verifica.controllo == 3)
507     riflessivita=0;
508
509 /****** Controllo se riflessiva
510 *****/
511
512 if(riflessivita == 1)
513     printf(" _ _ _ e 'riflessiva\n");
514 else
515     printf(" _ _ _ non _ e 'riflessiva\n");
516
517 /****** Fine riflessivita
518 *****/
519
520 return(riflessivita);
521 }
522
523 /****** FUNZIONE PER CONTROLLARE
524 LA SIMMETRIA *****/
525
526 int controllo_simmetria(rel_bin verifica)
527 {
528
529     int i ,

```

```

530         j ,
531         riscontro ,
532         simmetria ;
533
534     simmetria = 1;
535
536
537     i = 0;
538     j = 0;
539     riscontro = 0;
540
541     /* controllo della simmetria per numeri */
542
543     if(verifica.controllo == 1)
544     {
545
546         while( i < verifica.dimensione)
547         {
548
549             j = 0;
550             while( j < verifica.dimensione)
551             {
552
553                 if(verifica.primo_termine[i] ==
554                    verifica.secondo_termine[j])
555                     if(verifica.primo_termine[j] ==
556                        verifica.secondo_termine[i])
557                         riscontro++;
558
559                 j++;
560             }
561
562             if(riscontro == 0)
563             {
564                 j = verifica.dimensione;
565                 i = verifica.dimensione;
566                 simmetria = 0;
567             }
568             riscontro = 0;
569             i++;
570         }
571
572     }
573
574     /* controllo della simmetria per stringhe */

```

```

572
573     if(verifica.controllo == 2)
574     {
575
576         while( i < verifica.dimensione)
577         {
578
579             j = 0;
580             while( j < verifica.dimensione)
581             {
582
583                 if(strcmp(verifica.prima_stringa[i],
584                             verifica.seconda_stringa[j]) == 0 )
585                     riscontro++;
586
587                 j++;
588             }
589
590             if(riscontro == 0)
591             {
592                 j = verifica.dimensione;
593                 i = verifica.dimensione;
594                 simmetria = 0;
595             }
596             riscontro = 0;
597             i++;
598         }
599
600     }
601     /* Relazione Vuota */
602
603     if(verifica.controllo == 3)
604     {
605         printf("La stringa e' simmetrica\n");
606         simmetria = 1;
607     }
608
609     /****** Controllo se la simmetria e' stata
610                verificata *****/
611     if(verifica.controllo != 3)
612     {

```

```

612         if(simmetria == 1)
613             printf("Luce'simmetrica\n");
614         else
615             printf("Luce'asimmetrica\n");
616     }
617     /****** Fine controllo simmetria *****/
618
619     return(simmetria);
620 }
621
622
623
624 /* FUNZIONE PER CONTROLLARE LA TRANSITIVIT */
625
626
627 int controllo_transitivita(rel_bin verifica)
628 {
629
630     int i ,
631         j ,
632         k,
633         transitivita;
634
635     /*IMPOSTO LA TRANSITIVITA INIZIALMENTE COME VERA E
        AZZERO I CONTATORI*/
636     transitivita = 1;
637     i = 0;
638     j = 0;
639     k = 0;
640
641     /*VERIFICA TRANSITIVIT PER NUMERI*/
642
643
644     if(verifica.controllo == 1)
645     {
646
647         while(i < verifica.dimensione)
648         {
649             j = 0;
650
651             while(j < verifica.dimensione)
652             {
653                 k=0;
654

```

```

655         if(verifica.secondo_termine[i] ==
656             verifica.primo_termine[j])
657         {
658             transitivita = 0;
659             while(k < verifica.dimensione)
660             {
661                 if(verifica.primo_termine[i]
662                     == verifica.primo_termine[k
663                     ])
664                 {
665                     if(verifica.secondo_termine[k]==
666                         verifica.secondo_termine[j])
667                     {
668                         transitivita = 1;
669                         k = verifica.
670                             dimensione;
671                     }
672                 }
673             }
674             k++;
675         }
676         if(transitivita==0)
677         {
678             j=verifica.dimensione;
679             i=verifica.dimensione;
680         }
681         j++;
682     }
683     i++;
684 }
685 }
686
687
688 /***** VERIFICA TRANSITIVIT PER
689     STRINGHE *****/
690 if(verifica.controllo == 2)

```



```

691     {
692
693
694         while(i < verifica.dimensione)
695         {
696             j = 0;
697
698             while(j < verifica.dimensione)
699             {
700                 k=0;
701
702                 if(strcmp(verifica.seconda_stringa[i],
703                     verifica.prima_stringa[j]) == 0)
704                 {
705                     transitivita = 0;
706
707                     while(k < verifica.dimensione)
708                     {
709                         if(strcmp(verifica.prima_stringa[i], verifica.prima_stringa[k]) == 0)
710                         {
711                             if(strcmp(verifica.seconda_stringa[k],
712                                 verifica.seconda_stringa[j]) ==
713                                 0)
714                             {
715                                 transitivita = 1;
716                                 k = verifica.dimensione;
717                             }
718                         }
719                     }
720                     k++;
721                 }
722             }
723             if(transitivita==0)
724             {
725                 j=verifica.dimensione;
726                 i=verifica.dimensione;
727             }
728         }
729     }

```

```

727             j++;
728         }
729
730         i++;
731     }
732
733 }
734 /* Relazione Vuota */
735
736 if(verifica.controllo == 3)
737 {
738     transitivita = 1;
739 }
740
741 /****** Controllo se la relazione  Transitiva
       *****/
742
743 if(transitivita == 1)
744     printf("L' e' transitiva\n");
745
746 else
747     printf("L non e' transitiva\n");
748
749 /****** Fine controllo Transitivit
       *****/
750
751 return(transitivita);
752
753 }
754
755 /****** Dicotomia *****/
756
757 int controllo_dicotomia(rel_bin verifica)
758 {
759
760     int i,j,k;
761     int numero_elementi;
762     int dicotomia = 0;
763     int dimensione;
764     int riscontro;
765     int secondo_riscontro;
766     i=0;
767     j=0;
768     k=i-1;

```

```

769     riscontro = 0;
770     dimensione = verifica.dimensione;
771
772     /****** Dicotomia per numeri *****/
773
774     if(verifica.controllo == 1)
775     {
776
777         /****** Conto il numero delle coppie
778         esistenti (scarto le coppie uguali)
779         *****/
780
781         while( i < verifica.dimensione)
782         {
783             k = i-1;
784             j = i+1;
785             secondo_riscontro = 0;
786
787             if(i>0)
788             {
789                 while( k >= 0 )
790                 {
791                     if(verifica.primo_termine[i] ==
792                        verifica.primo_termine[k])
793                     {
794                         if(verifica.secondo_termine[i]
795                            == verifica.
796                            secondo_termine[k])
797                             secondo_riscontro = 1;
798                     }
799                     k--;
800                 }
801             }
802
803             if(secondo_riscontro != 1)
804             {
805                 while( j < verifica.dimensione)
806                 {
807                     if(verifica.primo_termine[i] ==
808                        verifica.primo_termine[j])
809                     if(verifica.secondo_termine[i]
810                        == verifica.
811                        secondo_termine[j])
812                     {

```

```

805                                     dimensione--;
806                                     }
807                                     j++;
808                                 }
809                            }
810                            i++;
811                    }
812
813
814                    i=0;
815                    j=0;
816                    k=0;
817                    numero_elementi=0;
818                    riscontro = 0;
819                    /****** Conto il numero degli
                        elementi distinti esistenti *****
                        */
820
821                    while(i<verifica.dimensione)
822                    {
823                        k=i-1;
824                        secondo_riscontro = 0;
825
826                        while(k >= 0)
827                        {
828                            if(verifica.primo_termine[i] ==
                                verifica.primo_termine[k])
829                                secondo_riscontro = 1;
830                            k--;
831                        }
832                        if(secondo_riscontro != 1)
833                        {
834                            if(verifica.primo_termine[i] ==
                                verifica.secondo_termine[i])
835                                riscontro++;
836
837                        }
838                        i++;
839                    }
840
841                    numero_elementi = riscontro;
842
843                    /****** Conto quanti dovrebbero essere
                        gli elementi per avere la dicotomia

```

```

844         *****/
845     while(numero_elementi > 0)
846     {
847         numero_elementi--;
848         riscontro = riscontro + numero_elementi;
849     }
850 }
851
852 /***/ VERIFICA DICOTOMICA PER
853 STRINGHE *****/
854 if(verifica.controllo == 2)
855 {
856
857     /***/ Conto il numero delle coppie
858     esistenti (scarto le coppie uguali)
859     *****/
860
861     while( i < verifica.dimensione)
862     {
863         k = i-1;
864         j = i+1;
865         secondo_riscontro = 0;
866         if(i>0)
867         {
868             while( k >= 0 )
869             {
870                 if((strcmp(verifica.prima_stringa[
871                     i],verifica.prima_stringa[k]))
872                     == 0)
873                 {
874                     if((strcmp(verifica.
875                         seconda_stringa[i],verifica
876                         .seconda_stringa[k])) == 0)
877                         secondo_riscontro = 1;
878                 }
879                 k--;
880             }
881         }
882     }
883
884     if(secondo_riscontro != 1)
885     {
886         while( j < verifica.dimensione)

```

```

880         {
881             if((strcmp(verifica.prima_stringa[
                        i], verifica.prima_stringa[j]))
                        == 0)
882                 if((strcmp(verifica.
                        seconda_stringa[i], verifica
                        .seconda_stringa[j])) == 0)
883                     {
884                         dimensione--;
885                     }
886                 j++;
887             }
888         }
889         i++;
890     }
891
892     i=0;
893     k=0;
894     j=0;
895     numero_elementi = 0;
896     /****** Conto il numero degli
897     elementi distinti esistenti *****
898     */
899     while(i<verifica.dimensione)
900     {
901         k=i-1;
902         secondo_riscontro = 0;
903
904         while(k >= 0)
905         {
906             if((strcmp(verifica.prima_stringa[i],
                        verifica.prima_stringa[k])) == 0)
907                 secondo_riscontro = 1;
908             k--;
909         }
910         if(secondo_riscontro != 1)
911         {
912             if((strcmp(verifica.prima_stringa[i],
                        verifica.seconda_stringa[i])) == 0)
913                 numero_elementi++;
914         }
915     }

```

```

916         i++;
917     }
918     riscontro = numero_elementi;
919
920     /****** Conto quanti dovrebbero essere
gli elementi per avere la dicotomia
******/
921
922     while(numero_elementi > 0)
923     {
924
925         numero_elementi--;
926         riscontro = riscontro + numero_elementi;
927
928     }
929
930 }
931
932 /****** Verifico se la dicotomia
verificata ******/
933
934 if(dimensione == riscontro)
935     dicotomia = 1;
936
937 if(dicotomia == 1 )
938     printf(" _ _ _ e 'dicotomica\n\n");
939
940 else
941     printf(" _ _ _ non e 'dicotomica\n\n");
942
943 /****** Fine verifica dicotomia
******/
944
945     return(dicotomia);
946 }
947
948 /*Funzione di verifica dell'ordine totale*/
949
950
951 void ordine_totale(rel_bin verifica)
952 {
953
954     int parziale ,
955         dicotomia;

```

```

956
957     dicotomia=2;
958     parziale = ordine_parziale(verifica);
959     if(parziale == 1)
960         dicotomia = controllo_dicotomia(verifica);
961
962     if(parziale == 0)
963         printf("\n_l'ordine_non_e'totale_in_quanto_
          non_e'nemmeno_parziale");
964
965     if(dicotomia == 0)
966         printf("\n_l'ordine_non_e'totale_in_quanto_
          non_viene_rispettata_la_propieta'di_
          dicotomia");
967
968     if(dicotomia == 1 && parziale == 1)
969         printf("\n_Quindi_e'una_relazione_d'ordine_
          totale");
970
971     printf("\n\n..._Controllo_Ordine_Totale_
          Terminato...\n\n\n\n");
972 }
973
974 /*Funzione che stabilisce se e'una relazione di
          equivalenza o meno*/
975
976 void relazione_equivalenza(rel_bin verifica)
977 {
978
979     int riflessivita;
980     int simmetria;
981     int transitivita;
982
983     riflessivita = controllo_riflessivita(verifica);
984     simmetria = controllo_simmetria(verifica);
985     controllo_antisimmetria(verifica);
986     transitivita = controllo_transitivita(verifica);
987
988     if(riflessivita == 1 && simmetria == 1 &&
          transitivita == 1)
989         printf("\n_Quindi_e'una_relazione_di_
          equivalenza\n");
990
991     if(riflessivita == 0)

```



```

992         printf("\n_Quindi_non_e'una_relazione_di_
           equivalenza_perche'non_riflessiva\n");
993
994     if(simmetria == 0)
995         printf("\n_Quindi_non_e'una_relazione_di_
           equivalenza_perche'non_simmetrica\n");
996
997     if(transitivita == 0)
998         printf("\n_Quindi_non_e'una_relazione_di_
           equivalenza_perche'non_transitiva\n");
999 }
1000
1001 /*Funzione che stabilisce se la relazione binaria
      acquisita e'una funzione matematica*/
1002
1003 void controllo_funzione(rel_bin verifica)
1004 {
1005
1006     int i;
1007     int k;
1008     int termini_diversi;
1009     int termini_uguali_prima;
1010     int termini_uguali_dopo;
1011     int errore;
1012
1013     if(verifica.controllo == 1)
1014     {
1015
1016         i=0;
1017         errore=0;
1018         termini_diversi=0;
1019         termini_uguali_dopo=0;
1020         termini_uguali_prima=0;
1021         while(i < verifica.dimensione)
1022         {
1023             k=verifica.dimensione-1;
1024             termini_uguali_dopo=termini_uguali_prima;
1025             while(k > i)
1026             {
1027                 if(verifica.primo_termine[i] ==
                   verifica.primo_termine[k])
1028                 {
1029                     if(verifica.secondo_termine[i] !=
                       verifica.secondo_termine[k])

```

```

1030         {
1031             errore=1;
1032             printf("\n_Nel_%d_elemento_c'e
                'un_errore_che_impedisce_
                alla_relazione_binaria\n",k
                +1);
1033             printf("_di_essere_una_
                funzione\n");
1034             k=i;
1035             i=verifica.dimensione;
1036         }
1037         if(verifica.secondo_termine[i] ==
            verifica.secondo_termine[k])
1038             termini_uguali_dopo++;
1039     }
1040     k--;
1041 }
1042 if(errore == 0 && termini_uguali_dopo ==
    termini_uguali_prima)
1043     termini_diversi++;
1044
1045     termini_uguali_prima = termini_uguali_dopo
        ;
1046     i++;
1047 }
1048 if(errore == 0 && (termini_diversi == (
    verifica.dimensione - termini_uguali_prima)
    ))
1049 {
1050     if(verifica.insieme_a == 2)
1051         printf("\n_La_relazione_binaria_e'una_
            funzione_totale\n");
1052     else
1053         printf("\n_La_relazione_binaria_ _una_
            funzione_parziale\n");
1054     controllo_biiettivita(verifica);
1055 }
1056 else
1057     printf("\n_La_relazione_binaria_non_e'una_
        funzione\n");
1058 }
1059
1060 /****** Controllo se c' una funzione per
        stringhe (le stringhe sono considerate come

```

```

        costanti di diverso valore) *****/
1061
1062     if(verifica.controllo == 2)
1063     {
1064
1065         i=0;
1066         errore=0;
1067         termini_diversi=0;
1068         termini_uguali_dopo=0;
1069         termini_uguali_prima=0;
1070         while(i < verifica.dimensione)
1071         {
1072             k=verifica.dimensione-1;
1073             termini_uguali_dopo=termini_uguali_prima;
1074             while(k > i)
1075             {
1076                 if((strcmp(verifica.prima_stringa[i],
1077                             verifica.prima_stringa[k])) == 0)
1078                 {
1079                     if((strcmp(verifica.
1080                                 seconda_stringa[i], verifica.
1081                                 seconda_stringa[k])) != 0)
1082                     {
1083                         errore=1;
1084                         printf("\n_Nel_%d_elemento_c'e
1085                             'un_errore_che_impedisce_
1086                             alla_relazione_binaria\n",k
1087                             +1);
1088                         printf("_di_essere_una_
1089                             funzione\n");
1090                         k=i;
1091                         i=verifica.dimensione;
1092                     }
1093                     else
1094                         termini_uguali_dopo++;
1095                 }
1096                 k--;
1097             }
1098             if(errore == 0 && termini_uguali_dopo ==
1099                 termini_uguali_prima)
1100                 termini_diversi++;
1101             termini_uguali_prima = termini_uguali_dopo
1102             ;

```

```

1095         i++;
1096     }
1097     if(errore == 0 && (termini_diversi == (
        verifica.dimensione - termini_uguali_prima)
        ))
1098     {
1099         if(verifica.insieme_a == 2)
1100             printf("\nLa relazione binaria e' una
                funzione totale\n");
1101         else
1102             printf("\nLa relazione binaria e' una
                funzione parziale\n");
1103         controllo_biiettivita (verifica);
1104     }
1105     else
1106         printf("\nLa relazione binaria non e' una
                funzione\n");
1107 }
1108 /* Relazione Vuota */
1109 if(verifica.controllo == 3)
1110     printf("\nLa relazione vuota non e' una
            funzione\n");
1111 printf("\n\n... Controllo Funzione Terminato
        ... \n\n\n");
1112
1113 }
1114
1115 /******FUNZIONE PER IL controllo DELL'INIETTIVITA
        '******/
1116
1117 int controllo_iniettivita (rel_bin verifica)
1118 {
1119
1120     int i;
1121     int k;
1122     int termini_diversi;
1123     int termini_uguali_prima;
1124     int termini_uguali_dopo;
1125     int errore;
1126     int iniettivita;
1127
1128     iniettivita = 0;
1129
1130     if(verifica.controllo == 1)

```

```

1131     {
1132
1133         i=0;
1134         errore=0;
1135         termini_diversi=0;
1136         termini_uguali_dopo=0;
1137         termini_uguali_prima=0;
1138
1139         while(i < verifica.dimensione)
1140         {
1141
1142             k=verifica.dimensione-1;
1143             termini_uguali_dopo=termini_uguali_prima;
1144             while(k > i)
1145             {
1146
1147                 if(verifica.secondo_termine[i] ==
1148                     verifica.secondo_termine[k])
1149                 {
1150
1151                     if(verifica.primo_termine[i] !=
1152                         verifica.primo_termine[k])
1153                     {
1154
1155                         errore=1;
1156                         printf("\n_Nel_%d_elemento_c'e
1157                             'un_errore_che_impedisce_
1158                             alla_funzione\n",k+1);
1159                         printf("_di_essere_iniettiva\n
1160                             ");
1161                         k=i;
1162                         i=verifica.dimensione;
1163                     }
1164                     if(verifica.primo_termine[i] ==
1165                         verifica.primo_termine[k])
1166                         termini_uguali_dopo++;
1167                 }
1168             }
1169             k--;
1170         }
1171         if(errore == 0 && termini_uguali_dopo ==
1172             termini_uguali_prima)
1173             termini_diversi++;
1174     }

```

```

1167         termini_uguali_prima = termini_uguali_dopo
1168         ;
1169         i++;
1170     }
1171     if(errore == 0 && (termini_diversi == (
1172         verifica.dimensione - termini_uguali_prima)
1173     ))
1174     {
1175         printf("\nLa funzione e' iniettiva\n");
1176         iniettivita = 1;
1177     }
1178     else
1179         printf("\nLa funzione non e' iniettiva\n");
1180     ;
1181 }
1182
1183 /****** Controllo iniettivita' per stringhe
1184 ******/
1185
1186 if(verifica.controllo == 2)
1187 {
1188     i=0;
1189     errore=0;
1190     termini_diversi=0;
1191     termini_uguali_dopo=0;
1192     termini_uguali_prima=0;
1193
1194     while(i < verifica.dimensione)
1195     {
1196         k=verifica.dimensione-1;
1197         termini_uguali_dopo=termini_uguali_prima;
1198         while(k > i)
1199         {
1200             if((strcmp (verifica.seconda_stringa[i]
1201                 ,verifica.seconda_stringa[k])) ==
1202                 0)
1203             {
1204                 if((strcmp (verifica.prima_stringa
1205                     [i],verifica.prima_stringa[k]))
1206                     != 0)
1207                 {

```

```

1202         errore=1;
1203         printf("\n_Nel_%d_elemento_c'e
        'un_errore_che_impedisce_
        alla_funzione\n",k+1);
1204         printf("_di_essere_iniettiva\n
        ");
1205         k=i;
1206         i=verifica.dimensione;
1207     }
1208     if((strcmp (verifica.prima_stringa
        [i],verifica.prima_stringa[k]))
        == 0)
1209         termini_uguali_dopo++;
1210     }
1211
1212     k--;
1213 }
1214 if(errore == 0 && termini_uguali_dopo ==
        termini_uguali_prima)
1215     termini_diversi++;
1216
1217     termini_uguali_prima = termini_uguali_dopo
        ;
1218     i++;
1219 }
1220 if(errore == 0 && (termini_diversi == (
        verifica.dimensione - termini_uguali_prima)
        ))
1221 {
1222     printf("\n_La_funzione_e'iniettiva");
1223     iniettivita = 1;
1224 }
1225 else
1226     printf("\n_La_funzione_non_e'iniettiva");
1227 }
1228
1229 return(iniettivita);
1230 }
1231
1232 /*****FUNZIONE PER IL controllo DELLA
        SURIETTIVITA *****/
1233
1234 int controllo_suriettivita(rel_bin verifica)
1235 {

```

```

1236     int suriattivita;
1237
1238     if(verifica.insieme_b == 2)
1239     {
1240         suriattivita = 1;
1241         printf("\n_la_funzione_e'_suriattivita");
1242     }
1243
1244     else
1245     {
1246         suriattivita = 0;
1247         printf("\n_la_funzione_non_e'_suriattivita");
1248     }
1249
1250     return(suriattivita);
1251 }
1252
1253 /******FUNZIONE PER IL controllo DELLA
BIETTIVITA'******/
1254
1255 void controllo_biattivita(rel_bin verifica)
1256 {
1257
1258     int    suriattivita ,
1259           iniattivita;
1260
1261     suriattivita = controllo_suriattivita(verifica);
1262     iniattivita = controllo_iniattivita(verifica);
1263
1264
1265     if( suriattivita == 1 && iniattivita == 1)
1266         printf("\n_la_funzione_e'_biattivita");
1267     else
1268         printf("\n_la_funzione_non_e'_biattivita");
1269     return;
1270 }
1271
1272
1273 int controllo_antisimmetria(rel_bin verifica)
1274 {
1275
1276     int i ,
1277         j ,
1278         riscontro ,

```



```

1279         antisimmetria;
1280
1281     antisimmetria = 1;
1282
1283
1284     i = 0;
1285     j = 0;
1286     riscontro = 1;
1287
1288     /* controllo della antisimmetria per numeri*/
1289
1290     if(verifica.controllo == 1)
1291     {
1292
1293         while( i < verifica.dimensione)
1294         {
1295
1296             j = 0;
1297             while( j < verifica.dimensione)
1298             {
1299
1300                 if(verifica.primo_termine[i] ==
1301                    verifica.secondo_termine[j]){
1302                     if(verifica.primo_termine[j] == verifica.
1303                        secondo_termine[i])
1304                         if(verifica.primo_termine[i]
1305                            == verifica.
1306                               primo_termine[j])
1307                             riscontro++;
1308                 }
1309
1310                 else
1311                     riscontro = 0;
1312             }
1313             j++;
1314         }
1315
1316         if(riscontro == 0)
1317         {
1318             j = verifica.dimensione;
1319             i = verifica.dimensione;
1320             antisimmetria = 0;
1321         }
1322         i++;
1323     }
1324 }

```

```

1319     }
1320
1321     /* controllo della antisimmetria per stringhe */
1322
1323     if(verifica.controllo == 2)
1324     {
1325
1326         while( i < verifica.dimensione)
1327         {
1328
1329             j = 0;
1330             while( j < verifica.dimensione)
1331             {
1332
1333                 if(strcmp(verifica.prima_stringa[i],
1334                             verifica.seconda_stringa[j]) == 0 )
1335                 {
1336                     if(strcmp(verifica.prima_stringa[j],
1337                             verifica.prima_stringa[i]) == 0 )
1338                     {
1339                         riscontro++;
1340                     }
1341                     else
1342                     {
1343                         riscontro=0;
1344                     }
1345                     j++;
1346                 }
1347             }
1348             if(riscontro == 0)
1349             {
1350                 j = verifica.dimensione;
1351                 i = verifica.dimensione;
1352                 antisimmetria = 0;
1353             }
1354             i++;
1355         }
1356     }
1357
1358     /* ***** Controllo se la simmetria stata verificata ***** */

```

```

1356
1357     if(antisimmetria == 1)
1358         printf("...e'antisimmetrica\n");
1359     else
1360         printf("...non_e'antisimmetrica\n");
1361
1362     /****** Fine controllo simmetria *****/
1363
1364     return(antisimmetria);
1365 }

```

### 4.3 Test

```
1 #include<stdio.h>
2 #include"librerie/lib_rel_bin.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 lib_rel_bin.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'antisimmetrica
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:
```

```
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'antisimmetrica
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
e'antisimmetrica
non e'transitiva

Non e'una relazione d'ordine parziale in quanto non rispetta tutte le propieta'
manca la propieta'di transitivita'

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

## 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:
```

```
4) Test verifica ordine totale
5) Test verifica relazione d'equivalenza
6) Test funzione
7) Esci

scelta: 5
e'riflessiva
e'simmetrica
non e'antisimmetrica
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:
```

```
scelta: 5
non e'riflessiva
e'asimmetrica
e'antisimmetrica
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:
```

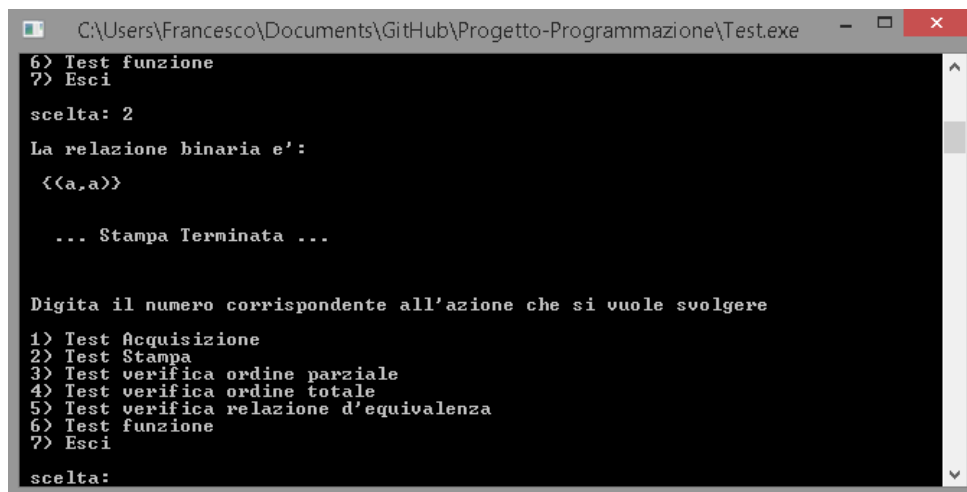
## 5.6 Test 6:

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

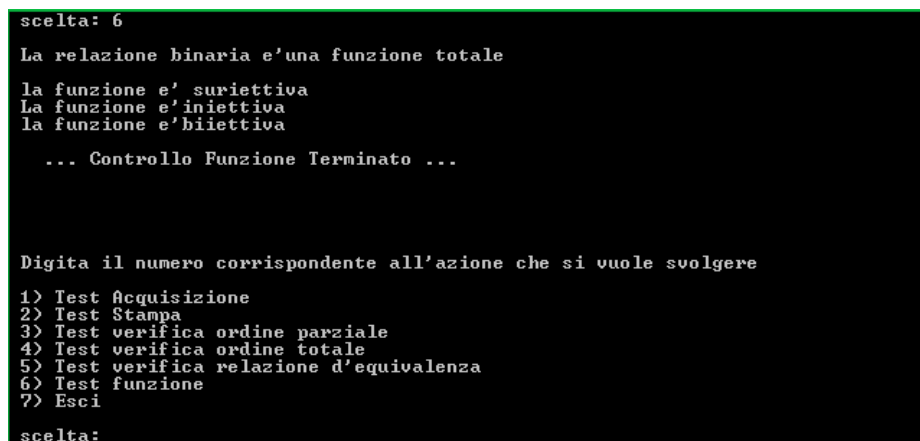
scelta: 2

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:
```



```
scelta: 6

La relazione binaria e'una funzione totale
la funzione e' suriettiva
La funzione 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:
```

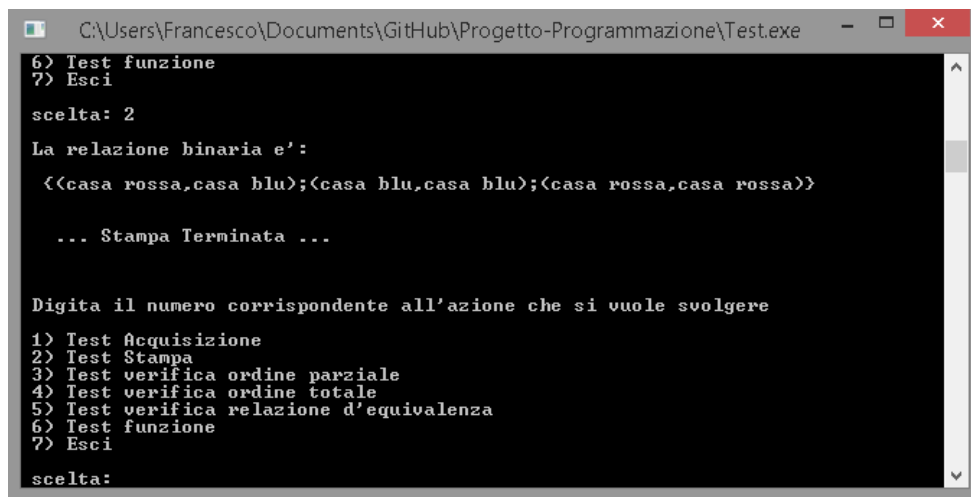
## 5.7 Test 7:

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

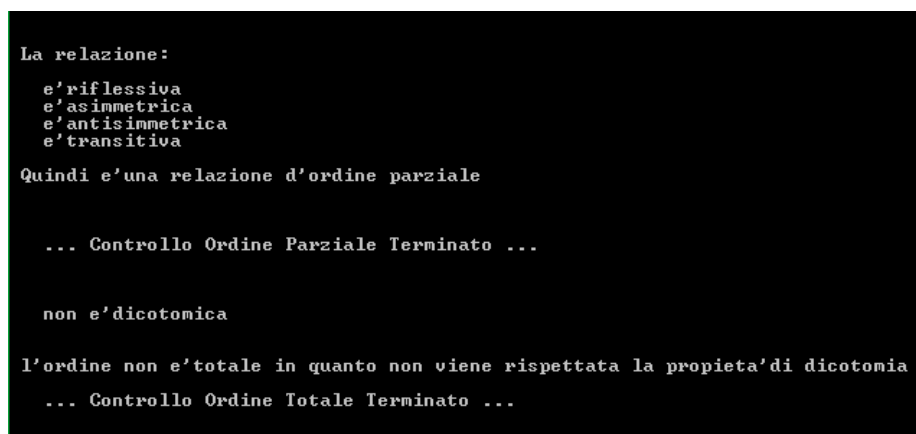
La relazione binaria e':

<(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:
```



```
La relazione:
e'riflessiva
e'asimmetrica
e'antisimmetrica
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.

```
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 inserire stringhe
3 per la relazione vuota

scelta: 1

Inserisci i termini della coppia
Primo Termine: 1
Secondo Termine: a

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

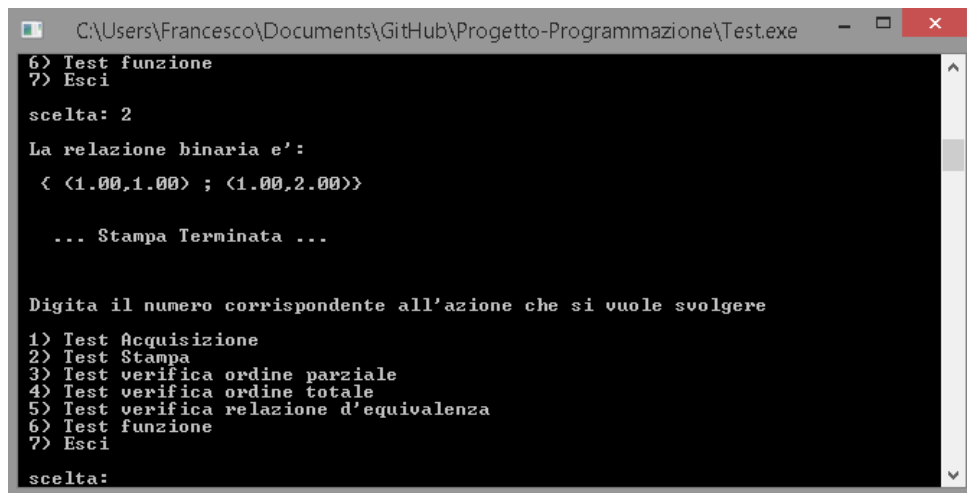
## 5.9 Test 9:

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;



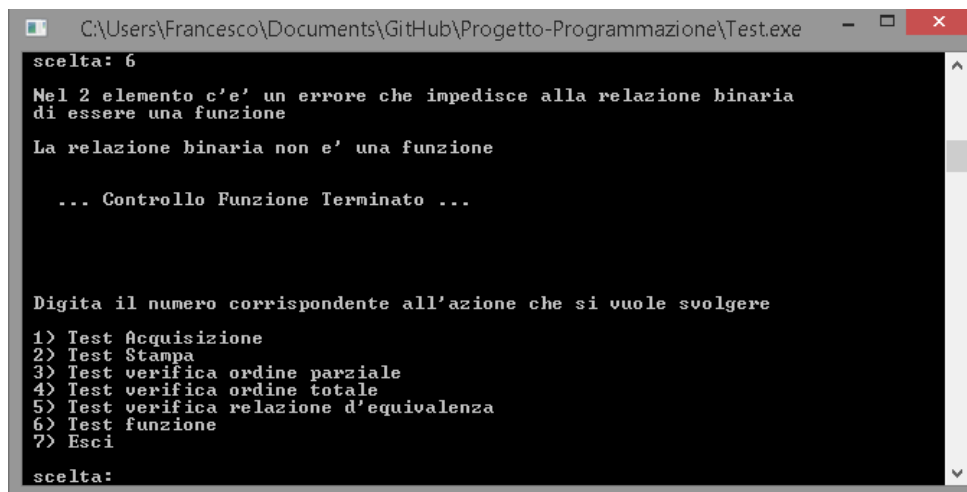
```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

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:

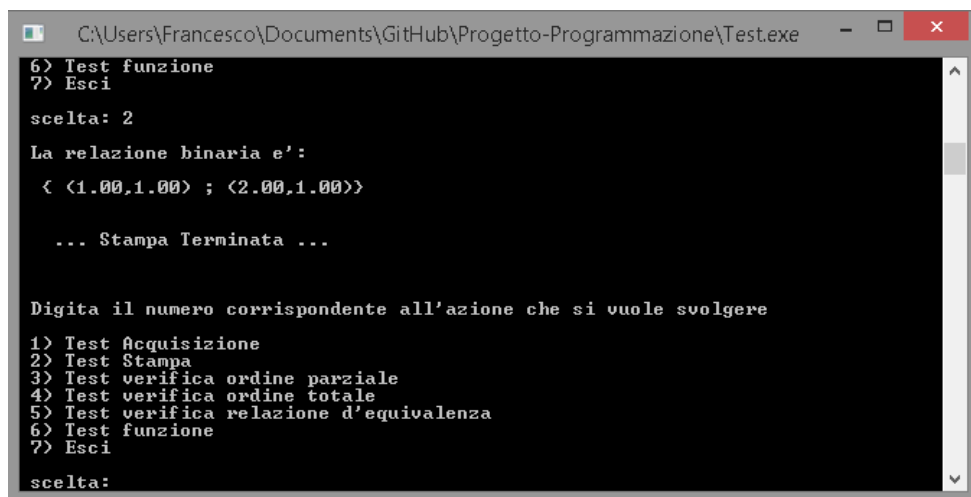
Inputs: (1,1) (2,1)

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



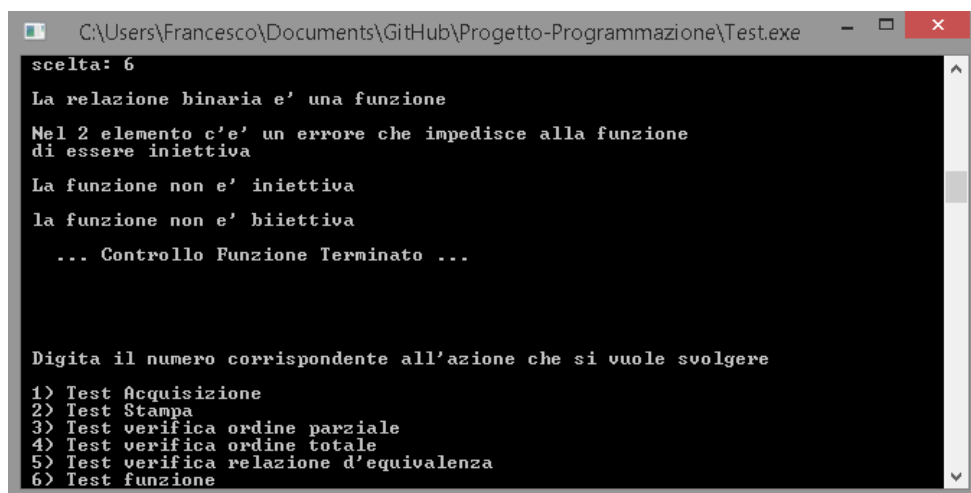
```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

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:

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

| = numero\_elementi > numero\_elementi - 1  
Poichè:

$$(P \wedge \text{numero\_elementi} < 1) = (\text{numero\_elementi} > 0 \wedge \text{riscontro} = (\text{numero\_elementi} > 1) = (\text{numero\_elementi} > 0 \wedge \text{riscontro} =$$

$$= \sum_{j=0}^{\text{numero\_elementi}-1} \text{numero\_elementi} - j \wedge \text{numero\_elementi} < 1)$$

$$\equiv (\text{numero\_elementi} = 1 \wedge \text{riscontro} =$$

$$= \sum_{j=0}^{\text{numero\_elementi}-1} \text{numero\_elementi} - j \wedge \text{numero\_elementi} < 1)))$$

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:

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