

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 $(a,b); (a1,b1); (a2,b2); \dots$ 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 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");
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 (primi_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              primi_termine_acquisito = 1;
103
104          /*Acquisisco il secondo termine*/
105          if (secondi_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
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 ("\nInserisci i termini della
159             coppia\n");
160         relazione.prima_stringa = (char **)
161             realloc (relazione.prima_stringa, (
162                 relazione.dimensione+1) * sizeof (char
163                 *));
164
165         /*Acquisisco i termini della coppia*/
166         relazione.prima_stringa[relazione.
167             dimensione - 1] = (char *) malloc (2);
168         fflush(stdin);
169         printf (" _Primo Termine: ");
170         while (temporaneo != '\n')
171         {
172             temporaneo = getc (stdin);
173             relazione.prima_stringa [relazione.
174                 dimensione - 1] = (char*) realloc
175                 (relazione.prima_stringa[relazione.dimensione
176                 -1],
177                 (i+1) * sizeof (char*));
178             relazione.prima_stringa [relazione.
179                 dimensione - 1] [i] = temporaneo;
180             i++;
181         }
182
183         /*Imposto ora il carattere di terminazione
184             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 = (double **)
        realloc (relazione.seconda_stringa ,
183 (relazione.dimensione+1) * sizeof (double*));
184     relazione.seconda_stringa [relazione.
        dimensione - 1] = (char *) malloc (2);
185     fflush (stdin);
186     temporaneo='a';
187     i=0;
188     while (temporaneo != '\n')
189     {
190         temporaneo = getc (stdin);
191         relazione.seconda_stringa [relazione.
            dimensione - 1] = (char*) realloc (
            relazione.seconda_stringa [relazione
                .dimensione-1],
192             (i+1) * sizeof (char*));
193         relazione.seconda_stringa [relazione.
            dimensione - 1] [i] = temporaneo;
194         i++;
195     }
196
197     /*Imposto ora il carattere di terminazione
        a \0 dato che adesso \n*/
198     relazione.seconda_stringa [relazione.
        dimensione - 1] [i - 1] = '\0';
199
200     /*Chiedo all'utente se ci sono altre
        coppie*/
201
202     while (acquisizione_finita < 0 ||
        acquisizione_finita > 1 ||
        risultato_lettura != 1)
203     {
204
205         printf ("\nVuoi_acquisire_un'altra_
            coppia?_immetti_1_per_uscire,_0_per

```

```

        _continuare\n");
206         risultato_lettura = scanf ("%d",&
            acquisizione_finita);
207     }
208 }
209 }
210
211 relazione.insieme_b = -1;
212 risultato_lettura = 0;
213
214 printf ("\n_Ci_sono_elementi_del_secondo_insieme\n
        _che_non_fanno_parte_di_nessuna_coppia_ordinata
        ?\n");
215 printf ("\n_1_si_2_no\n\n_scelta:_");
216 while ((relazione.insieme_b < 0) || (relazione.
        insieme_b > 2) || risultato_lettura != 1)
217 {
218     fflush (stdin);
219     risultato_lettura = scanf("%d",&relazione.
        insieme_b);
220 }
221
222 relazione.insieme_a = -1;
223 risultato_lettura = 0;
224
225 printf ("\n_Ci_sono_elementi_del_primo_insieme\n
        _che_non_fanno_parte_di_nessuna_coppia_ordinata
        ?\n");
226 printf ("\n_1_si_2_no\n\n_scelta:_");
227 while ((relazione.insieme_a < 0) || (relazione.
        insieme_a > 2) || risultato_lettura != 1)
228 {
229     fflush (stdin);
230     risultato_lettura = scanf("%d",&relazione.
        insieme_a);
231
232 }
233
234 printf ("\n\n.....Acquisizione_Terminata....\n\n
        ");
235 return (relazione);
236 }
237

```

```

238  /*****FUNZIONE DI STAMPA
      *****/
239
240  void stampa (rel_bin stampa)
241  {
242
243      int i = 0;
244
245      printf ("\nLa relazione binaria e ");
246      printf ("\n\n{");
247
248      /*****Stampa per coppie numeriche *****/
249
250      if (stampa.controllo == 1)
251      {
252          while (i < stampa.dimensione)
253          {
254
255              printf ("(%.2lf,%.2lf)", stampa.
                  primo_termine[i], stampa.secondo_termine
                  [i]);
256              if (i+1 != stampa.dimensione)
257                  printf (" ";
258              i++;
259          }
260      }
261
262      /*****Stampa per coppie non numeriche *****/
          */
263
264      if (stampa.controllo == 2)
265      {
266          while (i < stampa.dimensione)
267          {
268              printf ("(%s,%s)", stampa.prima_stringa[i],
                  stampa.seconda_stringa[i]);
269              if (i+1 != stampa.dimensione)
270                  printf (" ";
271              i++;
272          }
273      }
274  }
275

```

```

276      /****** Fine Stampa *****/
277      */
278      printf ("}\n");
279      printf ("\n\n... _Stampa_Terminata_...\n\n");
280
281  }
282
283  /******FUNZIONE DI VERIFICA DI RELAZIONI D
'ORDINE******/
284
285  int ordine_parziale (rel_bin verifica)
286  {
287
288      int riflessivita ,
289          transitivita ,
290          antisimmetria ,
291          parziale;
292
293      /*STAMPO LE PROPIETA 'DELLA RELAZIONE*/
294
295      printf ("\n\nLa relazione:\n\n");
296
297      /****** Chiamo le funzioni per poter stabilire
le propriet ******/
298      riflessivita = controllo_riflessivita (verifica);
299      controllo_simmetria(verifica);
300      antisimmetria = controllo_antisimmetria (verifica)
301      ;
302      transitivita = controllo_transitivita (verifica);
303
304      /****** Controllo se rispetta le propriet
per essere una relazione d'ordine parziale
******/
305      if (transitivita == 1 && antisimmetria == 1 &&
306          riflessivita == 1)
307      {
308          parziale = 1;
309          printf ("\n_Quindi_e 'una_relazione_d'ordine_
310              parziale\n\n");
311      }
312      else
313      {

```

```

312         printf ("\nNon e' una relazione d'ordine
           parziale in quanto non rispetta tutte le
           proprieta'\n");
313     parziale = 0;
314 }
315 if (transitivita == 0)
316     printf ("\nmanca la proprieta' di transitivita
           '\n");
317 if (antisimmetria == 0)
318     printf ("\nmanca la proprieta' di antisimmetria
           '\n");
319 if (riflessivita == 0)
320     printf ("\nmanca la proprieta' di riflessivita
           '\n");
321 /****** Fine controllo Ordine Parziale
           *****/
322
323     printf ("\n\n... Controllo Ordine Parziale
           Terminato...\n\n\n");
324     return (parziale);
325 }
326
327
328 /******FUNZIONE PER CONTROLLARE LA RIFLESSIVIT
           *****/
329
330 int controllo_riflessivita (rel_bin verifica)
331 {
332
333     int i,
334         j,
335         k,
336         riscontro,
337         secondo_riscontro,
338         riflessivita;
339
340     riflessivita = 1;
341     i = 0;
342     j = 0;
343     k = 0;
344     riscontro = 0;
345     secondo_riscontro = 0;
346
347     /* Verifica riflessivit */

```

```

348
349      /*Definizione: una relazione per la quale esiste
           almeno un elemento che non e'in relazione con
           s stesso non soddisfa la definizione di
           riflessivit */
350
351      while ( (i < verifica.dimensione) && (k < verifica
           .dimensione))
352      {
353
354          /* Verifica riflessivit per numeri*/
355
356          if (verifica.controllo == 1)
357          {
358              riscontro = 0;
359              secondo_riscontro = 0;
360              if (verifica.primo_termine[i] == verifica.
           secondo_termine[i])
361                  riscontro++; /**** Controllo se c'
           stato un riscontro a,a****/
362              secondo_riscontro++;
363              if (riscontro != 0)
364              {
365                  i++;
366                  k++;
367              }
368              /**/
369              else
370              {
371                  j=0;
372                  riscontro = 0;
373                  secondo_riscontro = 0;
374
375                  /****** Controllo la
           riflessivit per gli elementi del
           primo insieme
           ******/
376
377                  while (j < verifica.dimensione)
378                  {
379                      if (j == i)
380                          j++;
381                      else
382                      {

```



```

383         if (verifica.primo_termine[i]
              == verifica.primo_termine[j]
              )
384         if (verifica.primo_termine
              [j] == verifica.
              secondo_termine[j])
385             riscontro++;
386
387         j++;
388     }
389 }
390
391 j = 0;
392
393 /****** Controllo la
              riflessivit per gli elementi del
              secondo insieme
              *****/
394
395 while (j < verifica.dimensione)
396 {
397     if (j == k)
398         j++;
399     else
400     {
401         if (verifica.secondo_termine[k]
              == verifica.
              secondo_termine[j])
402             if (verifica.primo_termine
                  [j] == verifica.
                  secondo_termine[j])
403                 secondo_riscontro++;
404
405         j++;
406     }
407 }
408 if (riscontro != 0)
409     i++;
410
411 /**** Se non c' stato un riscontro di
              riflessivit esco e imposto la
              riflessivit a 0 *****/
412
413 else

```

```

414         {
415             i=verifica.dimensione;
416             riflessivita = 0;
417         }
418
419         if (secondo_riscontro != 0)
420             k++;
421
422         else
423         {
424             k=verifica.dimensione;
425             riflessivita = 0;
426         }
427     }
428
429 }
430
431 /****** VERIFICA RIFLESSIVIT PER
432 STRINGHE *****/
433
434 if (verifica.controllo == 2)
435 {
436     riscontro = 0;
437     secondo_riscontro = 0;
438     if (strcmp (verifica.prima_stringa[i],
439                 verifica.seconda_stringa[i]) == 0)
440         riscontro++;
441     secondo_riscontro++;
442     if (riscontro != 0)
443     {
444         i++;
445         k++;
446     }
447
448     else
449     {
450         j=0;
451         riscontro = 0;
452         secondo_riscontro = 0;
453
454         /****** Controllo la
455         riflessivita per gli elementi del
456         primo insieme
457         ******/

```

```

453
454 while (j < verifica.dimensione)
455 {
456     if (j == i)
457         j++;
458     else
459     {
460         if (strcmp (verifica .
                     prima_stringa[i], verifica .
                     prima_stringa[j]) == 0)
461             if (strcmp (verifica .
                         prima_stringa[j],
                         verifica .
                         seconda_stringa[j]) ==
                         0)
462                 riscontro++;
463
464         j++;
465     }
466 }
467
468 j = 0;
469
470 /****** Controllo la
    riflessivit per gli elementi del
    secondo insieme
    ******/
471
472 while (j < verifica.dimensione)
473 {
474     if (j == k)
475         j++;
476     else
477     {
478         if (strcmp (verifica .
                     seconda_stringa[k], verifica .
                     seconda_stringa[j]) == 0)
479             if (strcmp (verifica .
                         prima_stringa[j],
                         verifica .
                         seconda_stringa[j]) ==
                         0)
480                 secondo_riscontro++;
481

```

```

482             j++;
483         }
484     }
485     if (riscontro != 0)
486         i++;
487
488     else
489     {
490         i=verifica.dimensione;
491         riflessivita = 0;
492     }
493
494     if (secondo_riscontro != 0)
495         k++;
496
497     else
498     {
499         k=verifica.dimensione;
500         riflessivita = 0;
501     }
502 }
503
504 }
505
506 }
507 /* Relazione vuota */
508
509 if(verifica.controllo == 3)
510     riflessivita=0;
511
512 /****** Controllo se riflessiva
513      *****/
514
515 if (riflessivita == 1)
516     printf ("L' e' riflessiva\n");
517 else
518     printf ("L non e' riflessiva\n");
519
520 /****** Fine riflessivita
521      *****/
522
523 return (riflessivita);
524 }
525

```

```

524
525
526  /****** FUNZIONE PER CONTROLLARE
      LA SIMMETRIA *****/
527
528  /****** Definizione: In matematica, una
      relazione binaria R in un insieme X **/
529  /****** simmetrica se e solo se, presi due
      elementi qualsiasi a e b, vale che **/
530  /****** se a in relazione con b allora anche
      b in relazione con a. *****/
531
532  int controllo_simmetria (rel_bin verifica)
533  {
534
535      int i ,
536          j ,
537          riscontro ,
538          simmetria;
539
540      simmetria = 1;
541
542
543      i = 0;
544      j = 0;
545      riscontro = 0;
546
547      /*controllo della simmetria per numeri*/
548
549      if (verifica.controllo == 1)
550      {
551
552          while ( i < verifica.dimensione)
553          {
554
555              j = 0;
556              while ( j < verifica.dimensione)
557              {
558
559                  if (verifica.primo_termine[i] ==
                      verifica.secondo_termine[j])
560                  if (verifica.primo_termine[j] ==
                      verifica.secondo_termine[i])
561                      riscontro++;

```

```

562         j++;
563     }
564
565     if (riscontro == 0)
566     {
567         j = verifica.dimensione;
568         i = verifica.dimensione;
569         simmetria = 0;
570     }
571     riscontro = 0;
572     i++;
573 }
574
575 }
576
577 /* controllo della simmetria per stringhe */
578
579 if (verifica.controllo == 2)
580 {
581
582     while ( i < verifica.dimensione)
583     {
584
585         j = 0;
586         while ( j < verifica.dimensione)
587         {
588
589             if (strcmp (verifica.prima_stringa[i],
590                         verifica.seconda_stringa[j]) == 0 )
591                 if (strcmp (verifica.prima_stringa
592                             [j],verifica.seconda_stringa[i
593                             ]) == 0 )
594                     riscontro++;
595
596             j++;
597         }
598
599         if (riscontro == 0)
600         {
601             j = verifica.dimensione;
602             i = verifica.dimensione;
603             simmetria = 0;
604         }
605         riscontro = 0;

```

```

603         i++;
604     }
605
606 }
607 /* Relazione Vuota */
608
609 if (verifica.controllo == 3)
610 {
611     printf ("true'simmetrica\n");
612     simmetria = 1;
613 }
614
615 /***** Controllo se la simmetria stata
        verificata *****/
616 if(verifica.controllo != 3)
617 {
618     if (simmetria == 1)
619         printf ("true'simmetrica\n");
620     else
621         printf ("true'asimmetrica\n");
622 }
623 /***** Fine controllo simmetria *****/
624
625 return (simmetria);
626 }
627
628
629
630 /* FUNZIONE PER CONTROLLARE LA TRANSITIVIT */
631
632 /***** Definizione: In matematica, una relazione
        binaria  $R$  in un insieme  $X$  transitiva se e solo se
633     per ogni  $a, b, c$  appartenenti ad  $X$ , se  $a$  in
        relazione con  $b$  e  $b$  in relazione con  $c$ ,
        allora
634      $a$  in relazione con  $c$ .*****/
635
636
637 int controllo_transitivita (rel_bin verifica)
638 {
639
640     int i,
641         j,
642         k,

```

```

643         transitivita;
644
645     /*IMPOSTO LA TRANSITIVITA INIZIALMENTE COME VERA E
        AZZERO I CONTATORI*/
646     transitivita = 1;
647     i = 0;
648     j = 0;
649     k = 0;
650
651     /*VERIFICA TRANSITIVIT PER NUMERI*/
652
653
654     if (verifica.controllo == 1)
655     {
656
657         while (i < verifica.dimensione)
658         {
659             j = 0;
660
661             while (j < verifica.dimensione)
662             {
663                 k=0;
664
665                 if (verifica.secondo_termine[i] ==
                    verifica.primo_termine[j])
666                 {
667                     transitivita = 0;
668
669                     while (k < verifica.dimensione)
670                     {
671                         if (verifica.primo_termine[i]
                            == verifica.primo_termine[k
672
673                             {
674                                 if (verifica .
                                    secondo_termine[k]==
                                    verifica .
                                    secondo_termine[j])
675                                 {
676                                     transitivita = 1;
677                                     k = verifica .
678                                     dimensione;
679                                 }
680                             }
681                         }
682                     }
683                 }
684             }
685         }
686     }

```



```

679
680         k++;
681     }
682
683     if (transitivita==0)
684     {
685         j=verifica.dimensione;
686         i=verifica.dimensione;
687     }
688 }
689
690     j++;
691 }
692
693     i++;
694 }
695 }
696
697
698 /****** VERIFICA TRANSITIVITA PER
699 STRINGHE *****/
700
701 if (verifica.controllo == 2)
702 {
703
704     while (i < verifica.dimensione)
705     {
706         j = 0;
707
708         while (j < verifica.dimensione)
709         {
710             k=0;
711
712             if (strcmp (verifica.seconda_stringa[i]
713                        ,verifica.prima_stringa[j]) == 0)
714             {
715                 transitivita = 0;
716
717                 while (k < verifica.dimensione)
718                 {
719                     if (strcmp (verifica.
720                                prima_stringa[i],verifica.
721                                prima_stringa[k]) == 0)

```

```

719         {
720             if (strcmp (verifica .
                        seconda_stringa[k] ,
                        verifica .
                        seconda_stringa[j]) ==
721                 0)
722             {
723                 transitivita = 1;
724                 k = verifica .
725                     dimensione;
726             }
727         }
728     }
729     k++;
730 }
731     if (transitivita==0)
732     {
733         j=verifica . dimensione;
734         i=verifica . dimensione;
735     }
736 }
737     j++;
738 }
739 }
740     i++;
741 }
742 }
743 }
744 /* Relazione Vuota */
745
746 if (verifica . controllo == 3)
747 {
748     transitivita = 1;
749 }
750
751 /****** Controllo se la relazione    Transitiva
       *****/
752
753 if (transitivita == 1)
754     printf ("L' e' transitiva\n");
755
756 else

```

```

757         printf ( " _ _ _ non e' transitiva\n" );
758
759         /* ***** Fine controllo Transittivit
760         ***** */
761     return (transitivita);
762
763 }
764
765 /* ***** Dicotomia ***** */
766
767 int controllo_dicotomia (rel_bin verifica)
768 {
769
770     int i,j,k;
771     int numero_elementi;
772     int dicotomia = 0;
773     int dimensione;
774     int riscontro;
775     int secondo_riscontro;
776     i=0;
777     j=0;
778     k=i-1;
779     riscontro = 0;
780     dimensione = verifica.dimensione;
781
782     /* ***** Dicotomia per numeri ***** */
783
784     if (verifica.controllo == 1)
785     {
786
787         /* ***** Conto il numero delle coppie
788         esistenti (scarto le coppie uguali)
789         ***** */
790
791         while ( i < verifica.dimensione)
792         {
793             k = i-1;
794             j = i+1;
795             secondo_riscontro = 0;
796
797             if (i>0)
798             {
799                 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         }
808         k--;
809     }
810 }
811
812 if (secondo_riscontro != 1)
813 {
814     while ( j < verifica.dimensione)
815     {
816         if (verifica.primo_termine[i] ==
817             verifica.primo_termine[j])
818         {
819             if (verifica.secondo_termine[i
820                 ] == verifica.
821                     secondo_termine[j])
822             {
823                 dimensione--;
824             }
825         }
826         j++;
827     }
828     i++;
829 }
830
831 i=0;
832 j=0;
833 k=0;
834 numero_elementi=0;
835 riscontro = 0;
836
837 /****** Conto il numero degli
838 elementi distinti esistenti *****
839 */
840
841 while (i<verifica.dimensione)
842 {
843     k=i-1;

```

```

834         secondo_riscontro = 0;
835
836         while (k >= 0)
837         {
838             if (verifica.primo_termine[i] ==
839                 verifica.primo_termine[k])
840                 secondo_riscontro = 1;
841             k--;
842         }
843         if (secondo_riscontro != 1)
844         {
845             if (verifica.primo_termine[i] ==
846                 verifica.secondo_termine[i])
847                 riscontro++;
848         }
849         i++;
850     }
851     numero_elementi = riscontro;
852
853     /****** Conto quanti dovrebbero essere
854         gli elementi per avere la dicotomia
855         *****/
856
857     while (numero_elementi > 0)
858     {
859         numero_elementi--;
860         riscontro = riscontro + numero_elementi;
861     }
862
863     /****** VERIFICA DICOTOMICA PER
864         STRINGHE *****/
865
866     if (verifica.controllo == 2)
867     {
868
869         /****** Conto il numero delle coppie
870             esistenti (scarto le coppie uguali)
871             *****/
872
873         while ( i < verifica.dimensione)
874         {

```

```

871         k = i-1;
872         j = i+1;
873         secondo_riscontro = 0;
874         if (i>0)
875         {
876             while ( k >= 0 )
877             {
878                 if ( (strcmp (verifica .
                        prima_stringa[i], verifica .
                        prima_stringa[k])) == 0)
879                 {
880                     if ( (strcmp (verifica .
                                seconda_stringa[i], verifica
                                .seconda_stringa[k])) == 0)
881                         secondo_riscontro = 1;
882                 }
883                 k--;
884             }
885         }
886
887         if (secondo_riscontro != 1)
888         {
889             while ( j < verifica.dimensione)
890             {
891                 if ( (strcmp (verifica .
                        prima_stringa[i], verifica .
                        prima_stringa[j])) == 0)
892                 {
893                     if ( (strcmp (verifica .
                                seconda_stringa[i], verifica
                                .seconda_stringa[j])) == 0)
894                         dimensione--;
895                 }
896                 j++;
897             }
898         }
899         i++;
900     }
901
902     i=0;
903     k=0;
904     j=0;
905     numero_elementi = 0;

```

```

907      /****** Conto il numero degli
          elementi distinti esistenti *****
          */
908
909      while (i < verifica.dimensione)
910      {
911          k = i - 1;
912          secondo_riscontro = 0;
913
914          while (k >= 0)
915          {
916              if ( (strcmp (verifica.prima_stringa[i]
917                          ], verifica.prima_stringa[k])) == 0)
918                  secondo_riscontro = 1;
919              k--;
920          }
921          if (secondo_riscontro != 1)
922          {
923              if ( (strcmp (verifica.prima_stringa[i]
924                          ], verifica.seconda_stringa[i])) ==
925                  0)
926                  numero_elementi++;
927          }
928          i++;
929      }
930      riscontro = numero_elementi;
931
932      /****** Conto quanti dovrebbero essere
          gli elementi per avere la dicotomia
          ******/
933
934      while (numero_elementi > 0)
935      {
936          numero_elementi--;
937          riscontro = riscontro + numero_elementi;
938      }
939
940  }
941
942      /****** Verifico se la dicotomia
          verificata ******/

```

```

943
944     if (dimensione == riscontro)
945         dicotomia = 1;
946
947     if (dicotomia == 1 )
948         printf ("L'ordine e' dicotomica\n\n");
949
950     else
951         printf ("L'ordine non e' dicotomica\n\n");
952
953     /****** Fine verifica dicotomia
          *****/
954
955     return (dicotomia);
956 }
957
958 /*Funzione di verifica dell'ordine totale*/
959
960
961 void ordine_totale (rel_bin verifica)
962 {
963
964     int parziale ,
965         dicotomia;
966
967     dicotomia=2;
968     parziale = ordine_parziale (verifica);
969     if (parziale == 1)
970         dicotomia = controllo_dicotomia (verifica);
971
972     if (parziale == 0)
973         printf ("\nL'ordine non e' totale in quanto
          non e' nemmeno parziale");
974
975     if (dicotomia == 0)
976         printf ("\nL'ordine non e' totale in quanto
          non viene rispettata la proprieta' di
          dicotomia");
977
978     if (dicotomia == 1 && parziale == 1)
979         printf ("\nQuindi e' una relazione d'ordine
          totale");
980

```



```

981     printf ("\n\n.....Controllo Ordine Totale _
          Terminato...\n\n\n\n");
982 }
983
984 /*Funzione che stabilisce se e'una relazione di
          equivalenza o meno*/
985
986 void relazione_equivalenza (rel_bin verifica)
987 {
988
989     int riflessivita;
990     int simmetria;
991     int transitivita;
992
993     riflessivita = controllo_riflessivita (verifica);
994     simmetria = controllo_simmetria (verifica);
995     controllo_antisimmetria (verifica);
996     transitivita = controllo_transitivita (verifica);
997
998     if (riflessivita == 1 && simmetria == 1 &&
          transitivita == 1)
999         printf ("\nQuindi e'una relazione di _
                  equivalenza\n");
1000
1001     if (riflessivita == 0)
1002         printf ("\nQuindi non e'una relazione di _
                  equivalenza perche'non riflessiva\n");
1003
1004     if (simmetria == 0)
1005         printf ("\nQuindi non e'una relazione di _
                  equivalenza perche'non simmetrica\n");
1006
1007     if (transitivita == 0)
1008         printf ("\nQuindi non e'una relazione di _
                  equivalenza perche'non transitiva\n");
1009 }
1010
1011 /*Funzione che stabilisce se la relazione binaria
          acquisita e'una funzione matematica*/
1012
1013 void controllo_funzione (rel_bin verifica)
1014 {
1015
1016     int i;

```

```

1017     int k;
1018     int termini_diversi;
1019     int termini_uguali_prima;
1020     int termini_uguali_dopo;
1021     int errore;
1022
1023     if (verifica.controllo == 1)
1024     {
1025
1026         i=0;
1027         errore=0;
1028         termini_diversi=0;
1029         termini_uguali_dopo=0;
1030         termini_uguali_prima=0;
1031         while (i < verifica.dimensione)
1032         {
1033             k=verifica.dimensione-1;
1034             termini_uguali_dopo=termini_uguali_prima;
1035             while (k > i)
1036             {
1037                 if (verifica.primo_termine[i] ==
1038                     verifica.primo_termine[k])
1039                 {
1040                     if (verifica.secondo_termine[i] !=
1041                         verifica.secondo_termine[k])
1042                     {
1043                         errore=1;
1044                         printf (" \n_Nel_%d_elemento_c'
1045                             e'un_errore_che_impedisce_
1046                             alla_relazione_binaria\n",k
1047                             +1);
1048                         printf ("_di_essere_una_
1049                             funzione\n");
1050                         k=i;
1051                         i=verifica.dimensione;
1052                     }
1053                     if (verifica.secondo_termine[i] ==
1054                         verifica.secondo_termine[k])
1055                         termini_uguali_dopo++;
1056                 }
1057                 k--;
1058             }
1059         }
1060         if (errore == 0 && termini_uguali_dopo ==
1061             termini_uguali_prima)

```

```

1053             termini_diversi++;
1054
1055             termini_uguali_prima = termini_uguali_dopo
1056             ;
1057             i++;
1058         }
1059         if (errore == 0 && (termini_diversi == (
1060             verifica.dimensione - termini_uguali_prima)
1061         ))
1062         {
1063             if(verifica.insieme_a == 2)
1064                 printf ("\nLa relazione binaria e' una
1065                     _funzione_totale\n");
1066             else
1067                 printf ("\nLa relazione binaria _una
1068                     _funzione_parziale\n");
1069             controllo_biiettivita (verifica);
1070         }
1071     else
1072         printf ("\nLa relazione binaria non e' una
1073             _funzione\n");
1074 }
1075
1076 /****** Controllo se c' e' una funzione per
1077 stringhe (le stringhe sono considerate come
1078 costanti di diverso valore) *****/
1079
1080 if (verifica.controllo == 2)
1081 {
1082     i=0;
1083     errore=0;
1084     termini_diversi=0;
1085     termini_uguali_dopo=0;
1086     termini_uguali_prima=0;
1087     while (i < verifica.dimensione)
1088     {
1089         k=verifica.dimensione-1;
1090         termini_uguali_dopo=termini_uguali_prima;
1091         while (k > i)
1092         {
1093             if ( (strcmp (verifica.prima_stringa[i
1094                 ],verifica.prima_stringa[k])) == 0)
1095                 {

```

```

1088         if ( (strcmp (verifica .
1089                 seconda_stringa[i], verifica .
1090                 seconda_stringa[k])) != 0)
1091         {
1092             errore=1;
1093             printf ("\n_Nel_%d_elemento_c'
1094                     e'un_errore_che_impedisce_
1095                     alla_relazione_binaria\n",k
1096                     +1);
1097             printf ("_di_essere_una_
1098                     funzione\n");
1099             k=i;
1100             i=verifica.dimensione;
1101         }
1102         else
1103             termini_uguali_dopo++;
1104     }
1105     k--;
1106 }
1107 if (errore == 0 && termini_uguali_dopo ==
1108     termini_uguali_prima)
1109     termini_diversi++;
1110
1111 termini_uguali_prima = termini_uguali_dopo
1112     ;
1113 i++;
1114 }
1115 if (errore == 0 && (termini_diversi == (
1116     verifica.dimensione - termini_uguali_prima)
1117 ))
1118 {
1119     if(verifica.insieme_a == 2)
1120         printf ("\n_La_relazione_binaria_e'una
1121                 _funzione_totale\n");
1122     else
1123         printf ("\n_La_relazione_binaria_ _una
1124                 _funzione_parziale\n");
1125     controllo_biiettivita (verifica);
1126 }
1127 else
1128     printf ("\n_La_relazione_binaria_non_e'una
1129             _funzione\n");
1130 }
1131 /* Relazione Vuota*/

```

```

1119     if (verifica.controllo == 3)
1120         printf ("\nLa relazione vuota non e' una
                funzione\n");
1121     printf ("\n\n... Controllo Funzione Terminato
                ... \n\n\n");
1122
1123 }
1124
1125 /*****FUNZIONE PER IL controllo DELL'INIETTIVITA
                *****/
1126
1127 int controllo_iniettivita (rel_bin verifica)
1128 {
1129
1130     int i;
1131     int k;
1132     int termini_diversi;
1133     int termini_uguali_prima;
1134     int termini_uguali_dopo;
1135     int errore;
1136     int iniettivita;
1137
1138     iniettivita = 0;
1139
1140     if (verifica.controllo == 1)
1141     {
1142
1143         i=0;
1144         errore=0;
1145         termini_diversi=0;
1146         termini_uguali_dopo=0;
1147         termini_uguali_prima=0;
1148
1149         while (i < verifica.dimensione)
1150         {
1151
1152             k=verifica.dimensione-1;
1153             termini_uguali_dopo=termini_uguali_prima;
1154             while (k > i)
1155             {
1156
1157                 if (verifica.secondo_termine[i] ==
                        verifica.secondo_termine[k])
1158                 {

```



```

1192
1193     if (verifica.controllo == 2)
1194     {
1195
1196         i=0;
1197         errore=0;
1198         termini_diversi=0;
1199         termini_uguali_dopo=0;
1200         termini_uguali_prima=0;
1201
1202         while (i < verifica.dimensione)
1203         {
1204             k=verifica.dimensione-1;
1205             termini_uguali_dopo=termini_uguali_prima;
1206             while (k > i)
1207             {
1208                 if ( (strcmp (verifica.seconda_stringa
1209                             [i],verifica.seconda_stringa[k]))
1210                     == 0)
1211                 {
1212                     if ( (strcmp (verifica.
1213                             prima_stringa[i],verifica.
1214                             prima_stringa[k])) != 0)
1215                     {
1216                         errore=1;
1217                         printf ("\n_Nel_%d_elemento_c'
1218                                 e'un_errore_che_impedisce_
1219                                 alla_funzione\n",k+1);
1220                         printf ("_di_essere_iniettiva\
1221                                 n");
1222                         k=i;
1223                         i=verifica.dimensione;
1224                     }
1225                     if ( (strcmp (verifica.
1226                             prima_stringa[i],verifica.
1227                             prima_stringa[k])) == 0)
1228                         termini_uguali_dopo++;
1229                 }
1230             }
1231             k--;
1232         }
1233         if (errore == 0 && termini_uguali_dopo ==
1234             termini_uguali_prima)
1235             termini_diversi++;

```

```

1226
1227         termini_uguali_prima = termini_uguali_dopo
1228             ;
1229         i++;
1230     }
1231     if (errore == 0 && (termini_diversi == (
1232         verifica.dimensione - termini_uguali_prima)
1233     ))
1234     {
1235         printf ("\nLa funzione e' iniettiva");
1236         iniettivita = 1;
1237     }
1238     else
1239         printf ("\nLa funzione non e' iniettiva");
1240 }
1241
1242 /******FUNZIONE PER IL controllo DELLA
1243 SURIETTIVITA'******/
1244 int controllo_suriettivita (rel_bin verifica)
1245 {
1246     int suriettivita;
1247
1248     if (verifica.insieme_b == 2)
1249     {
1250         suriettivita = 1;
1251         printf ("\nla funzione e' suriettiva");
1252     }
1253
1254     else
1255     {
1256         suriettivita = 0;
1257         printf ("\nla funzione non e' suriettiva");
1258     }
1259
1260     return (suriettivita);
1261 }
1262
1263 /******FUNZIONE PER IL controllo DELLA
1264 BIIETTIVITA'******/

```



```

1265 void controllo_biiettivita (rel_bin verifica)
1266 {
1267
1268     int    surriettivita ,
1269           iniiettivita;
1270
1271     surriettivita = controllo_suriettivita (verifica);
1272     iniiettivita = controllo_iniettivita (verifica);
1273
1274
1275     if ( surriettivita == 1 && iniiettivita == 1)
1276         printf ("\nla funzione e' biiettiva");
1277     else
1278         printf ("\nla funzione non e' biiettiva");
1279     return;
1280 }
1281
1282
1283 int controllo_antisimmetria (rel_bin verifica)
1284 {
1285
1286     int i ,
1287         j ,
1288         riscontro ,
1289         antisimmetria;
1290
1291     antisimmetria = 1;
1292
1293
1294     i = 0;
1295     j = 0;
1296     riscontro = 1;
1297
1298     /* controllo della antisimmetria per numeri*/
1299
1300     if (verifica.controllo == 1)
1301     {
1302
1303         while ( i < verifica.dimensione)
1304         {
1305
1306             j = 0;
1307             while ( j < verifica.dimensione)
1308             {

```

```

1309
1310         if (verifica.primo_termine[i] ==
1311             verifica.secondo_termine[j])
1312             if (verifica.primo_termine[j] ==
1313                 verifica.secondo_termine[i])
1314                 if (verifica.primo_termine[i]
1315                     == verifica.primo_termine[j]
1316                     )
1317                     riscontro++;
1318             else
1319                 riscontro = 0;
1320         j++;
1321     }
1322
1323     if (riscontro == 0)
1324     {
1325         j = verifica.dimensione;
1326         i = verifica.dimensione;
1327         antisimmetria = 0;
1328     }
1329     i++;
1330 }
1331
1332 }
1333
1334 /*controllo della antisimmetria per stringhe*/
1335
1336 if (verifica.controllo == 2)
1337 {
1338     while ( i < verifica.dimensione)
1339     {
1340         j = 0;
1341         while ( j < verifica.dimensione)
1342         {
1343             if (strcmp (verifica.prima_stringa[i],
1344                         verifica.seconda_stringa[j]) == 0 )
1345                 if (strcmp (verifica.prima_stringa
1346                             [j],verifica.seconda_stringa[i]
1347                             ) == 0 )
1348                     if (strcmp (verifica.
1349                                 prima_stringa[j],verifica.

```

```

1345         prima_stringa[i]) == 0 )
1346             riscontro++;
1347     else
1348         riscontro=0;
1349
1350         j++;
1351     }
1352     if (riscontro == 0)
1353     {
1354         j = verifica.dimensione;
1355         i = verifica.dimensione;
1356         antisimmetria = 0;
1357     }
1358     i++;
1359 }
1360
1361 }
1362
1363 /****** Controllo se la simmetria stata
1364 verificata *****/
1365
1366 if (antisimmetria == 1)
1367     printf ("_le'antisimmetrica\n");
1368 else
1369     printf ("_non_le'antisimmetrica\n");
1370
1371 /****** Fine controllo simmetria *****/
1372
1373 return (antisimmetria);
1374 }

```

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 ("\nProgramma per effettuare i Test sulla
        libreria\n");
15
16
17    printf ("\nDigita il numero corrispondente all'
        azione che si vuole svolgere\n");
18    printf ("\n1) Test Acquisizione\n2) Esci\n");
19
20    do
21    {
22        printf ("\nscelta:");
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..... 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 Progetto.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'dicotonica

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:
```

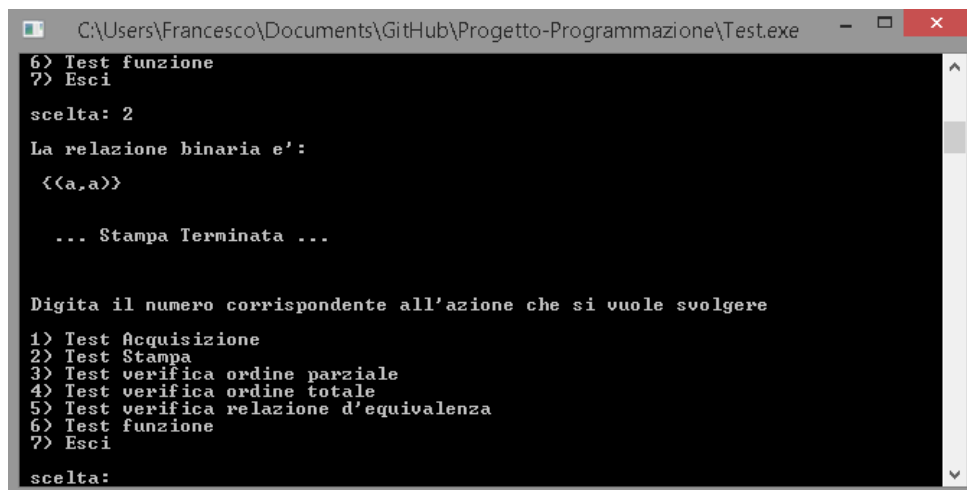
5.6 Test 6:

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

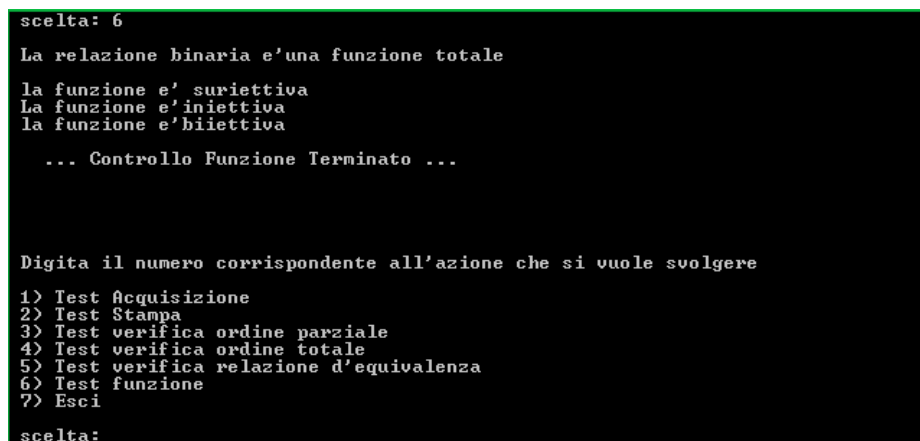
scelta: 2

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:
```

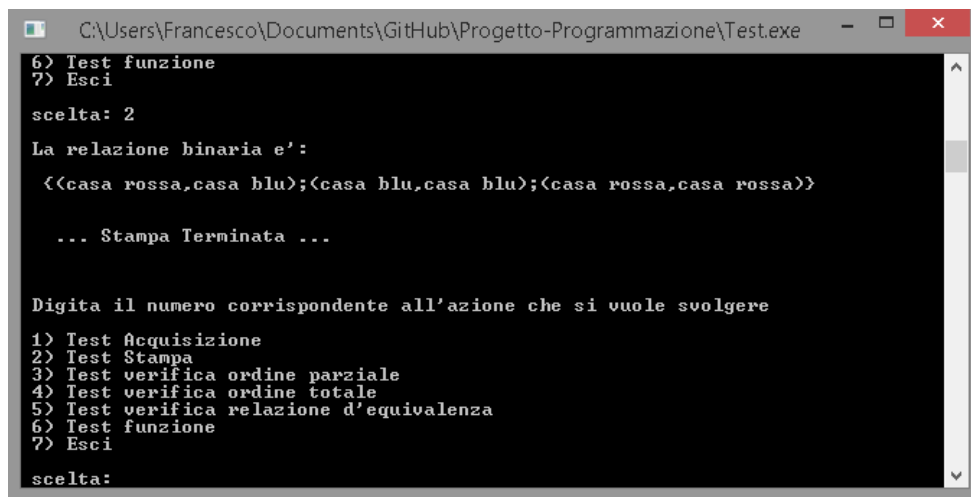
5.7 Test 7:

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

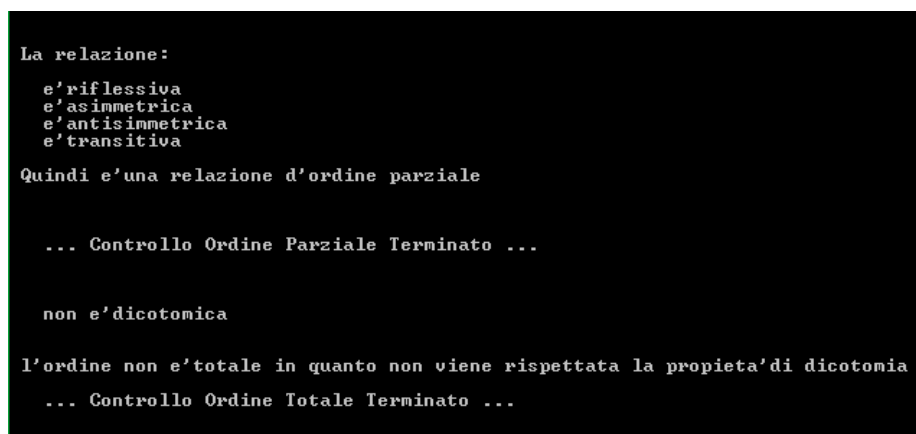
La relazione binaria e':

<(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:
```

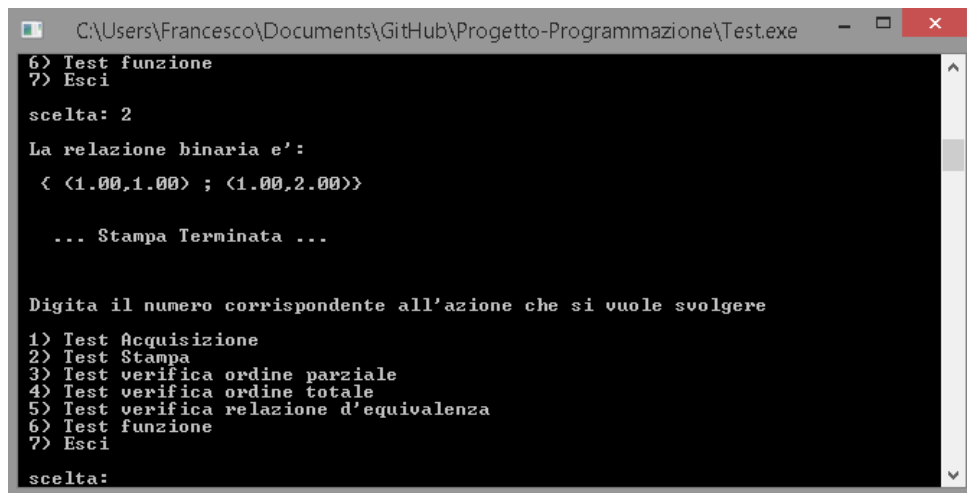
5.9 Test 9:

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':

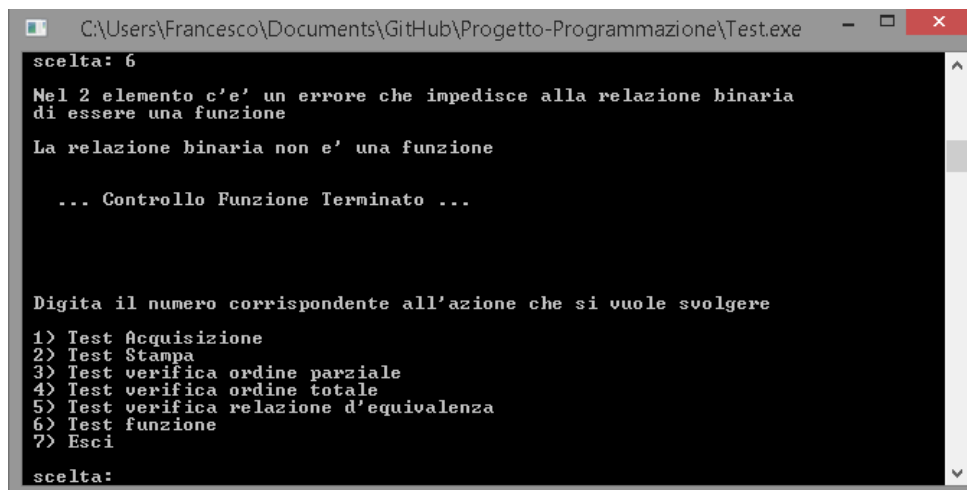
< (1.00,1.00) ; (1.00,2.00)>

... 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:

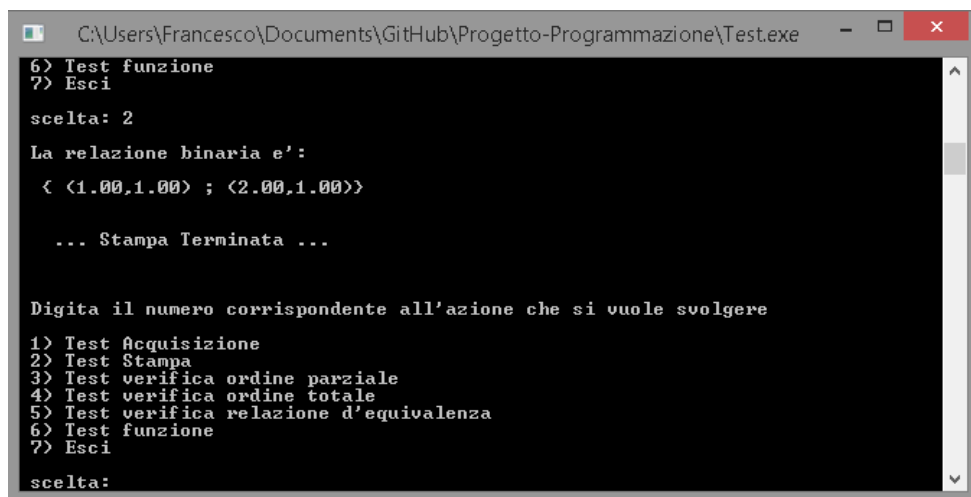
Inputs: (1,1) (2,1)

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



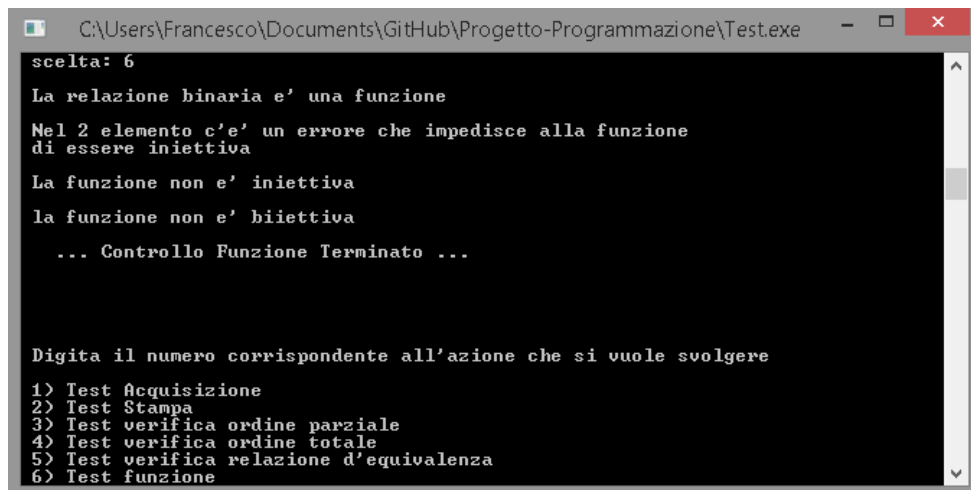
```
C:\Users\Francesco\Documents\GitHub\Progetto-Programmazione\Test.exe
6> Test funzione
7> Esci

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 $\text{tr}(\text{numero_elementi})$ decresce di un unità ad ogni iterazione in quanto numero_elementi viene decrementata di un'unità ad ogni iterazione.

* La limitatezza segue da:

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

| = numero_elementi > numero_elementi - 1
Poichè:

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

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

$$\equiv (\text{numero_elementi} = 1 \wedge \text{riscontro} =$$

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

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:

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