

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 12, 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	Funzioni per l'acquisizione:	5
3.3	Funzioni per la verifica delle proprietà:	5
3.4	Funzioni principali:	6
3.5	Input	7
3.6	Output - Acquisizione	8
3.7	Output - stampa	8
3.8	Output - ordine_parziale	8
3.9	Output - ordine_totale	8
3.10	Output - relazione_equivalenza	9
3.11	Output - check_funzione	9
4	Implementazione dell' algoritmo	10
4.1	Libreria	10
4.2	Test	38
4.3	Makefile	40
5	Testing del programma	41
5.1	Test 1:	41
5.2	Test 2:	42
5.3	Test 3:	43
5.4	Test 4:	44
5.5	Test 5:	45
5.6	Test 6:	46
5.7	Test 7:	47
5.8	Test 8:	48
5.9	Test 9:	49
5.10	Test 10:	50
6	Verica del programma	51

1 Specifica del Problema

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

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

2 Analisi del Problema

2.1 Input

1. Per l' acquisizione come input abbiamo una relazione binaria del tipo (a,b) che viene acquisita da tastiera;
2. Come input per le altre 5 funzioni abbiamo una relazione (precedentemente esportata dalla prima).

2.2 Output

1. La prima funzione (Acquisizione) restituisce una funzione binaria acquisita da tastiera;
2. La seconda funzione (Stampa) non restituisce nulla, ma stampa a video la relazione che aveva in ingresso; //
3. La terza funzione "ordine parziale" non restituisce nulla, ma stampa a video se la Relazione binaria acquisita è di ordine parziale o meno e stampa a video le proprietà della funzione, per poter mostrare a schermo quali proprietà non vengono rispettate;
4. La quarta funzione (ordine totale) non restituisce nulla, ma stampa a video se la Relazione binaria acquisita è di ordine totale o meno, stampando a video quale proprietà non vale nel caso la relazione non sia tale;
5. La quinta funzione (relazione equivalenza) non restituisce nulla, ma stampa a video se la relazione binaria acquisita è una relazione di equivalenza o meno e stampa a video le proprietà della funzione, per poter mostrare a schermo quali proprietà non vengono rispettate;
6. la sesta funzione (check funzione) non restituisce nulla, ma stampa a video se la relazione binaria acquisita è una funzione, e in caso contrario stampa a video quale coppia non fa rispettare le proprietà.

3 Progettazione dell' Algoritmo

3.1 Teoria

Per lo sviluppo di questo programma si necessita di alcuni cenni di Teoria degli insiemi quali:

Concetto di Relazione Binaria : In matematica, una relazione binaria definita su di un insieme, anche detta relazione o corrispondenza tra due oggetti, è un elenco di coppie ordinate di elementi appartenenti all'insieme. In modo equivalente, una relazione binaria è un sottoinsieme del prodotto cartesiano di un insieme con se stesso.

Concetto di Relazione d' Ordine Parziale: In matematica, pi precisamente in teoria degli ordini, una relazione d'ordine o ordine su di un insieme è una relazione binaria tra elementi appartenenti all'insieme che gode delle seguenti proprietà:

riflessiva

antisimmetrica

transitiva.

Concetto di Relazione d' Ordine Totale: Una relazione d' ordine si dice Totale, quando oltre a essere parziale soddisfa anche la proprietà di Dicotomia (tutti gli elementi devono essere in relazione tra di loro).

Concetto di riflessività : In logica e in matematica, una relazione binaria R in un insieme X è detta riflessiva se ogni elemento di X è in tale relazione con se stesso.

Concetto di transitività: In matematica, una relazione binaria R in un insieme X è transitiva se e solo se per ogni a, b, c appartenenti ad X , se a è in relazione con b e b è in relazione con c , allora a è in relazione con c .

Concetto di simmetricità: In matematica, una relazione binaria R in un insieme X è simmetrica se e solo se, presi due elementi qualsiasi a e b , vale che se a è in relazione con b allora anche b è in relazione con a .

Concetto di funzione: In matematica, una funzione, anche detta applicazione, mappa o trasformazione, è definita dai seguenti oggetti:

Un insieme X detto dominio della funzione. * Un insieme Y detto codominio della funzione. * Una relazione $f : X \rightarrow Y$ che ad ogni elemento dell'insieme X associa uno ed un solo elemento dell'insieme Y ; l'elemento assegnato a x appartenente ad X tramite f viene abitualmente indicato con $f(x)$.

Concetto di Iniettività: Una funzione si dice iniettiva quando a ogni elemento del dominio è assegnato uno e uno solo elemento del codominio.

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

3.2 Funzioni per l'acquisizione:

`acquisizione()` : per acquisire la relazione.

3.3 Funzioni per la verifica delle proprietà:

`check_iniettivita()` : per controllare se l' iniettività è rispettata o meno (0 non c' è, 1 c' è).

`check_transitivita()` : per controllare se la transitività viene rispettata o meno (0 non c' è, 1 c' è).

`check_simmetria()` : per controllare se la simmetria viene rispettata o meno (0 non c' è, 1 c' è).

`check_riflessivita()` : per controllare se la riflessività viene rispettata o meno (0 non c' è, 1 c' è).

`check_dicotomia()` : per verificare se la dicotomia viene rispettata o meno (0 non c' è, 1 c' è).

`check_suriettivita()`: verifica se la funzione gode della proprietà di suriettività, in questo caso sarà sempre settata a 1 in quanto tutti gli elementi del codominio (presi come gli elementi dei vari secondi termini digitati durante l' acquisizione) avranno sempre un elemento del dominio associato(dato che non si può acquisire il secondo termine se non se ne acquisisce prima il relativo primo, o arrivare alla funzione `check_suriettivita()` avendo acquisito solo il primo).

3.4 Funzioni principali:

`ordine_parziale()` : richiama le funzioni delle proprietà e controlla se c' è un ordine parziale(stampa a video se c' è o meno un ordine parziale, e nel caso non c' è stampa quali proprietà non vengono rispettate).

`ordine_totale()`: richiama la funzione `ordine_parziale` e `check_dicotomia` e controlla se c' è un ordine totale(stampa a video se esiste o meno un ordine totale, e nel caso non c' è stampa quali proprietà non vengono rispettate).

`relazione_equivalenza()` : richiama le funzioni delle proprietà e controlla se c' è una relazione d'equivalenza(stampa a video se c' è o meno una relazione d'equivalenza, e nel caso non c' è stampa quali proprietà non vengono rispettate).

`check_funzione()`:verifica se la relazione è una funzione(stampa a video se c' è o non c' è una funzione e nel caso non ci sia dice quale coppia non soddisfa le proprietà).

3.5 Input

Per l' input abbiamo necessità di usare una struttura dati dinamica, nella quale andiamo a salvare la Relazione Binaria dataci dall' utente, il numero delle coppie e il tipo di input (numerico o per stringhe).

L input dovrà essere dotato di diversi controlli, se l' utente sceglie di inserire un input di tipo numerico allora non potrà digitare stringhe e/o caratteri speciali etc.

La scelta di due tipi di input differente dovrà essere data per dare la possibilità all' utente nel caso scelga di fare un' input di tipo numerico di poter effettuare operazioni non legate alle funzioni della libreria, (esempio : l' utente vuole decidere di moltiplicare l' input per due, e vedere se mantiene le proprietà, con un' input di tipo numerico l' utente pu farlo e ci avrebbe un senso, con un' input di tipo stringa meno).

La scelta dell' input di tipo stringa dovrà essere data per aver maggior completezza, una relazione binaria non deve essere forzosamente numerica ma pu essere anche tra cose, oggetti, animali, colori e qualsiasi altra cosa possa venire in mente.

Alle varie funzioni verrà data come input la struttura dati salvata in precedenza dalla funzione Acquisizione, per poterne verificare le varie proprietà.

3.6 Output - Acquisizione

Durante l' acquisizione avremo diversi output video (printf) che guideranno l' utente nell' inserimento dei dati, e che segnaleranno eventuali errori commessi. Finita l' acquisizione dovremo restituire l' indirizzo della struttura, che all' interno quindi conterra' i dati inseriti dall' utente. Abbiamo scelto di fare ci perchè non essendo permesso l' utilizzo di variabili globali, il modo pi semplice di passare i dati inseriti da una funzione all' altra è quello di creare una struttura dinamica. Una volta restituito l' indirizzo della struttura, a seconda della funzione lanciata nel file Test.c si lanceranno le altre 5 funzioni, dato che queste prendono tutte in pasto l' output della prima (cioè l' indirizzo della struttura della relazione binaria) e la utilizzano per verificarne varie proprieta' .

3.7 Output - stampa

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

3.8 Output - ordine_parziale

La funzione ordine_parziale avra' come output la stampa a video del risultato della verifica delle proprieta' di riflessivita' antisimmetria e transitivita' . Nel caso in cui siano tutte verificate si stampera' che la relazione è una relazione di ordine parziale, mentre nel caso in cui non siano verificate si stampera' che non lo è e il perchè (cioè quale proprieta' non è o non sono verificate).

3.9 Output - ordine_totale

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

3.10 Output - relazione_equivalenza

La funzione `relazione_equivalenza` avra' come output la stampa a video del risultato della verifica delle proprieta' di riflessivita' simmetria e transitivita' e nel caso in cui siano tutte positive si stampera' che la relazione è una relazione di equivalenza, mentre nel caso in cui qualcosa non sia verificato si stampera' cio' che impedisce alla relazione di essere una relazione d'equivalenza.

3.11 Output - check_funzione

La funzione `check_funzione` avra' come output la stampa a video della verifica della proprieta' che rende la relazione binaria una funzione, e in caso lo sia anche se questa è suriettiva (che poi spiegheremo essere sempre verificata) e iniettiva, e in caso sia entrambe si stampera' che la relazione binaria oltre ad essere una funzione è una funzione biiettiva.

4 Implementazione dell' algoritmo

4.1 Libreria

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4
5  /*****STRUTTURA relBin
6  *****/
7  /***** Creo una struttura dove salvare le coppie
8  *****/
9  /***** Coppia Numerica *****/
10 double *primo_termine ,
11        *secondo_termine ;
12
13 /***** Coppia Qualsiasi *****/
14 char **prima_stringa ,
15       **seconda_stringa ;
16
17 /**** Variabili per salvare se ho acquisito una
18 coppia numerica o no e il numero delle coppie****
19 */
20 int controllo ,
21    dimensione ;
22 };
23
24 /*DICHIARO LE FUNZIONI*/
25 int check_simmetria(struct relBin);
26 int check_riflessivita(struct relBin);
27 int check_transitivita(struct relBin);
28 int check_suriettivita(struct relBin);
29 void check_biiettivita(struct relBin);
30
31 /***** Funzione di acquisizione
32 *****/
33 struct relBin acquisizione(struct relBin relazione){
34     int acquisizione_finita = 0;
35     int scan = 0;
```

```

36
37 relazione.dimensione = 0;
38 relazione.primo_termine = (double *) malloc(2);
39 relazione.secondo_termine = (double *) malloc(2);
40 relazione.prima_stringa = (char **) malloc(100);
41 relazione.seconda_stringa = (char **) malloc(100);
42
43 while((relazione.controllo < 1) || (relazione.
    controllo > 2) || scan != 1){
44     fflush(stdin);
45     printf("\n_Premi_1_se_vuoi_inmettere_solo_numeri,_2_
        per_altro\n");
46     printf("\n_scelta:_");
47     scan = scanf("%d",&relazione.controllo);
48 }
49
50 /** resetto scan a 0 **/
51 scan=0;
52
53 /*Acquisizione Numerica*/
54
55 if(relazione.controllo == 1){
56     while(acquisizione_finita == 0){
57         relazione.dimensione++;
58         acquisizione_finita = 2;
59
60         /*Acquisisco il primo termine della coppia*/
61
62         printf("\n_Inserisci_il_primo_termine_della_coppia
            _\n");
63         relazione.primo_termine = (double *) realloc(
            relazione.primo_termine, (relazione.dimensione
            +1) * sizeof(double));
64         scan = 0;
65         /*Check del primo termine della coppia*/
66
67         while(scan != 1){
68             printf("_Primo_Termine:_");
69             fflush(stdin);
70             scan = scanf("%lf",&relazione.primo_termine[
                relazione.dimensione - 1]);
71         if(scan == 0)
72             printf("\n_C'e'_un_errore,_reinserire_il_primo_
                termine\n");

```

```

73     }
74
75     /* Acquisisco il secondo termine della coppia */
76     scan = 0;
77     printf("\n Inserisci il secondo termine della
       coppia\n");
78     relazione.secondo_termine = (double *) realloc(
       relazione.secondo_termine, (relazione.
       dimensione+1) * sizeof(double));
79
80     /* Check del secondo termine della coppia */
81
82     while(scan != 1){
83         printf("\n Secondo Termine: ");
84         fflush(stdin);
85         scan = scanf("%lf",&relazione.secondo_termine[
            relazione.dimensione - 1]);
86         if(scan == 0)
87             printf("\n C'è un errore, reinserire il secondo
            termine\n");
88     }
89
90     /* Chiedo all'utente se ci sono altre coppie */
91
92     while(acquisizione_finita < 0 || acquisizione_finita
       > 1 || scan != 1){
93         printf("\n Vuoi acquisire un'altra coppia? immetti
            1 per uscire, 0 per continuare\n");
94         printf("\n scelta: ");
95         fflush(stdin);
96         scan = scanf("%d",&acquisizione_finita);
97     }
98 }
99 }
100
101 /* resetto scan a 0 */
102 scan = 0;
103
104 /* Acquisizione con stringhe */
105 if(relazione.controllo == 2){
106     while(acquisizione_finita == 0){
107         relazione.dimensione++;
108         acquisizione_finita = 2;
109

```

```

110  /* Acquisisco il primo termine della coppia*/
111
112      printf("\nInserisci il primo termine della coppia\n"
              "\n");
113      printf("\nPrimo Termine: ");
114      relazione.prima_stringa[relazione.dimensione - 1]
              = (char *) malloc(50);
115      scan = scanf("%[^\\n]s", relazione.prima_stringa[
              relazione.dimensione - 1]);
116
117  /* Acquisisco il secondo termine della coppia*/
118
119      printf("\nInserisci il secondo termine della coppia"
              "\n");
120      printf("\nSecondo Termine: ");
121      relazione.seconda_stringa[relazione.dimensione -
              1] = (char *) malloc(50);
122      scan = scanf("%[^\\n]s", relazione.seconda_stringa[
              relazione.dimensione - 1]);
123
124  /* riassetto scan a 0*/
125      scan = 0;
126
127  /* Chiedo all'utente se ci sono altre coppie*/
128
129      while(acquisizione_finita < 0 ||
              acquisizione_finita > 1 || scan != 1){
130
131          printf("\nVuoi acquisire un'altra coppia? "
                  "immetti 1 per uscire, 0 per continuare\n");
132          scan = scanf("%d",&acquisizione_finita);
133      }
134  }
135  }
136
137  printf("\n\n.....Acquisizione Terminata...\n\n");
138  return relazione;
139  }
140
141  /******FUNZIONE DI STAMPA*****
142
143  void stampa(struct relBin stampa){
144

```

```

145  int i = 0;
146
147  printf("\nLa relazione binaria e'");
148  printf("\n\n{");
149
150  /******Stampa per coppie numeriche *****/
151
152      if(stampa.controllo == 1){
153          while(i < stampa.dimensione){
154
155              printf("_(%.2lf,%.2lf)",stampa.primo_termine[i
156                  ],stampa.secondo_termine[i]);
157              if(i+1 != stampa.dimensione)
158                  printf("_;");
159              i++;
160          }
161
162  /******Stampa per coppie non numeriche *****/
163
164      if(stampa.controllo == 2){
165          while(i < stampa.dimensione){
166              printf("(%s,%s)",stampa.prima_stringa[i],
167                  stampa.seconda_stringa[i]);
168              if(i+1 != stampa.dimensione)
169                  printf(";");
170              i++;
171          }
172      }
173
174  /****** Fine Stampa *****/
175
176      printf("}\n");
177      printf("\n\n..._Stampa_Terminata_...\n\n");
178
179  }
180
181  /******FUNZIONE DI VERIFICA DI RELAZIONI D
182      'ORDINE******/
183
184  int ordine_parziale(struct relBin verifica){
185
186      int riflessivita ,

```



```

186     transitivita ,
187     simmetria ,
188     parziale ;
189
190     /*STAMPO LE PROPIETA ' DELLA RELAZIONE*/
191
192     printf("\n\nLa relazione:\n\n");
193
194     /****** Chiamo le funzioni per poter stabilire le
        propriet ******/
195
196     riflessivita = check_riflessivita(verifica);
197     simmetria = check_simmetria(verifica);
198     transitivita = check_transitivita(verifica);
199
200     /****** Controllo se rispetta le propriet per
        essere una relazione d'ordine parziale******/
201
202     if(transitivita == 1 && simmetria == 0 &&
        riflessivita == 1){
203         parziale = 1;
204         printf("\n_Quindi_e'_una_relazione_d'ordine_
            parziale\n\n");
205     }
206     else{
207
208         printf("\n_Non_e'_una_relazione_d'ordine_parziale_
            in_quanto_non_rispetta_tutte_le_propieta'\n");
209         parziale = 0;
210     }
211     if(transitivita == 0)
212         printf("\n_manca_la_propieta'_di_transitivita'\n")
            ;
213     if(simmetria == 1)
214         printf("\n_manca_la_propieta'_di_asimmetria\n");
215     if(riflessivita == 0)
216         printf("\n_manca_la_propieta'_di_riflessivita'\n")
            ;
217     /****** Fine controllo Ordine Parziale
        ******/
218
219     printf("\n\n..._Controllo_Ordine_Parziale_
        Terminato...\n\n\n");
220     return(parziale);

```

```

221 }
222
223
224 /******FUNZIONE PER CONTROLLARE LA RIFLESSIVIT
      ******/
225
226 int check_riflessivita (struct relBin verifica){
227
228     int i,
229         j,
230         k,
231         riscontro ,
232         secondo_riscontro ,
233         riflessivita;
234
235     riflessivita = 1;
236     i = 0;
237     j = 0;
238     k = 0;
239     riscontro = 0;
240     secondo_riscontro = 0;
241
242 /* Verifica riflessivit */
243
244 /* Definizione: una relazione per la quale esiste
      almeno un elemento che non e' in relazione con s
      stesso non soddisfa la definizione di riflessivit
      */
245
246     while((i < verifica.dimensione) && (k < verifica.
        dimensione)){
247
248 /* Verifica riflessivit per numeri*/
249
250         if(verifica.controllo == 1){
251             riscontro = 0;
252             secondo_riscontro = 0;
253             if(verifica.primo_termine[i] == verifica.
                secondo_termine[i])
254                 riscontro++; /**** Controllo se c' stato un
                    riscontro a,a****/
255             secondo_riscontro++;
256             if(riscontro != 0){
257                 i++;

```

```

258         k++;
259     }
260     /**/
261     else{
262         j=0;
263         riscontro = 0;
264         secondo_riscontro = 0;
265
266     /****** Controllo la riflessivit per gli
elementi del primo insieme
******/
267
268         while(j < verifica.dimensione){
269             if(j == i)
270                 j++;
271             else{
272                 if(verifica.primo_termine[i] == verifica.
                    primo_termine[j])
273                     if(verifica.primo_termine[j] == verifica
                        .secondo_termine[j])
274                         riscontro++;
275
276                 j++;
277             }
278         }
279
280         j = 0;
281
282     /****** Controllo la riflessivit per gli
elementi del secondo insieme
******/
283
284         while(j < verifica.dimensione){
285             if(j == k)
286                 j++;
287             else{
288                 if(verifica.secondo_termine[k] == verifica
                    .secondo_termine[j])
289                     if(verifica.primo_termine[j] == verifica
                        .secondo_termine[j])
290                         secondo_riscontro++;
291
292                 j++;
293             }

```

```

294     }
295     if(riscontro != 0)
296         i++;
297
298     /**** Se non c'è stato un riscontro di riflessivit
299     esco e setto la riflessivit a 0 *****/
300     else{
301         i=verifica.dimensione;
302         riflessivita = 0;
303     }
304
305     if(secondo_riscontro != 0)
306         k++;
307
308     else{
309         k=verifica.dimensione;
310         riflessivita = 0;
311     }
312 }
313
314 }
315
316 /****** VERIFICA RIFLESSIVIT PER STRINGHE
317 ******/
318 if(verifica.controllo == 2){
319     riscontro = 0;
320     secondo_riscontro = 0;
321     if(strcmp(verifica.prima_stringa[i], verifica.
322         seconda_stringa[i]) == 0)
323         riscontro++;
324     secondo_riscontro++;
325     if(riscontro != 0){
326         i++;
327         k++;
328     }
329
330     else{
331         j=0;
332         riscontro = 0;
333         secondo_riscontro = 0;

```

```

334  /***** Controllo la riflessivit per gli
      elementi del primo insieme
      *****/
335
336      while(j < verifica.dimensione){
337          if(j == i)
338              j++;
339          else{
340              if(strcmp(verifica.prima_stringa[i], verifica
                          .prima_stringa[j]) == 0)
341                  if(strcmp(verifica.prima_stringa[j],
                              verifica.seconda_stringa[j]) == 0)
342                      riscontro++;
343
344              j++;
345          }
346      }
347
348      j = 0;
349
350  /***** Controllo la riflessivit per gli
      elementi del secondo insieme
      *****/
351
352      while(j < verifica.dimensione){
353          if(j == k)
354              j++;
355          else{
356              if(strcmp(verifica.seconda_stringa[k],
                          verifica.seconda_stringa[j]) == 0)
357                  if(strcmp(verifica.prima_stringa[j],
                              verifica.seconda_stringa[j]) == 0)
358                      secondo_riscontro++;
359
360              j++;
361          }
362      }
363      if(riscontro != 0)
364          i++;
365
366      else{
367          i=verifica.dimensione;
368          riflessivita = 0;
369      }

```

```

370
371         if(secondo_riscontro != 0)
372             k++;
373
374         else{
375             k=verifica.dimensione;
376             riflessivita = 0;
377         }
378     }
379
380 }
381
382 }
383
384 /****** Controllo se riflessiva
******/
385
386     if(riflessivita == 1)
387         printf("e' riflessiva\n");
388     else
389         printf("non e' riflessiva\n");
390
391 /****** Fine riflessivita *****
*/
392
393     return(riflessivita);
394 }
395
396
397
398 /****** FUNZIONE PER CONTROLLARE
LA SIMMETRIA *****
399
400 /****** Definizione: In matematica, una
relazione binaria R in un insieme X */
401 /****** simmetrica se e solo se, presi due
elementi qualsiasi a e b, vale che */
402 /****** se a in relazione con b allora anche
b in relazione con a. *****
403
404 int check_simmetria(struct relBin verifica){
405
406     int i,
407         j,

```

```

408     riscontro ,
409     simmetria;
410
411     simmetria = 1;
412
413
414     i = 0;
415     j = 0;
416     riscontro = 0;
417
418     /* Check della simmetria per numeri*/
419
420     if(verifica.controllo == 1){
421
422         while( i < verifica.dimensione){
423
424             j = 0;
425             while( j < verifica.dimensione){
426
427                 if(verifica.primo_termine[i] == verifica .
428                     secondo_termine[j])
429                     if(verifica.primo_termine[j] == verifica .
430                         secondo_termine[i])
431                         riscontro++;
432
433             j++;
434         }
435
436         if(riscontro == 0){
437             j = verifica.dimensione;
438             i = verifica.dimensione;
439             simmetria = 0;
440         }
441         riscontro = 0;
442         i++;
443     }
444
445     /* Check della simmetria per stringhe*/
446
447     if(verifica.controllo == 2){
448
449         while( i < verifica.dimensione){

```

```

450
451     j = 0;
452     while( j < verifica.dimensione){
453
454         if(strcmp(verifica.prima_stringa[i],verifica.
            seconda_stringa[j]) == 0 )
455             if(strcmp(verifica.prima_stringa[j],verifica
                .seconda_stringa[i]) == 0 )
456                 riscontro++;
457
458         j++;
459     }
460
461     if(riscontro == 0){
462         j = verifica.dimensione;
463         i = verifica.dimensione;
464         simmetria = 0;
465     }
466     riscontro = 0;
467     i++;
468 }
469
470 }
471
472 /****** Controllo se la simmetria stata verificata
    ******/
473
474     if(simmetria == 1)
475         printf("L' e simmetrica\n");
476     else
477         printf("L' e asimmetrica\n");
478
479 /****** Fine controllo simmetria ******/
480
481     return(simmetria);
482 }
483
484
485
486 /* FUNZIONE PER CONTROLLARE LA TRANSITIVITA' */
487
488 /****** Definizione: In matematica, una relazione
    binaria R in un insieme X transitiva se e solo se

```



```

489      per ogni  $a, b, c$  appartenenti ad  $X$ , se  $a$  in
        relazione con  $b$  e  $b$  in relazione con  $c$ ,
        allora
490       $a$  in relazione con  $c$ .*****/
491
492
493 int check_transitivita(struct relBin verifica){
494
495     int i,
496         j,
497         k,
498         transitivita;
499
500     /*SETTO LA TRANSITIVITA INIZIALMENTE COME VERA E
        AZZERO I CONTATORI*/
501     transitivita = 1;
502     i = 0;
503     j = 0;
504     k = 0;
505
506     /*VERIFICA TRANSITIVITA PER NUMERI*/
507
508
509     if(verifica.controllo == 1){
510
511         while(i < verifica.dimensione){
512             j = 0;
513
514             while(j < verifica.dimensione){
515                 k=0;
516
517                 if(verifica.secondo_termine[i] == verifica.
                    primo_termine[j]){
518                     transitivita = 0;
519
520                     while(k < verifica.dimensione){
521                         if(verifica.primo_termine[i] == verifica.
                            primo_termine[k]){
522                             if(verifica.secondo_termine[k]==verifica
                                .secondo_termine[j]){
523                                 transitivita = 1;
524                                 k = verifica.dimensione;
525                             }
526                         }

```

```

527
528         k++;
529     }
530
531     if (transitivita==0){
532         j=verifica.dimensione;
533         i=verifica.dimensione;
534     }
535 }
536
537     j++;
538 }
539
540     i++;
541 }
542 }
543
544
545  ***** VERIFICA TRANSITIVITA PER STRINGHE
546  *****
547  if(verifica.controllo == 2){
548
549
550      while(i < verifica.dimensione){
551          j = 0;
552
553          while(j < verifica.dimensione){
554              k=0;
555
556              if(strcmp(verifica.seconda_stringa[i],verifica
557                  .prima_stringa[j]) == 0){
558                  transitivita = 0;
559
560                  while(k < verifica.dimensione){
561                      if(strcmp(verifica.prima_stringa[i],
562                          verifica.prima_stringa[k]) == 0){
563                          if(strcmp(verifica.seconda_stringa[k],
564                              verifica.seconda_stringa[j]) == 0){
565                              transitivita = 1;
566                              k = verifica.dimensione;
567                          }
568                      }
569                  }
570              }
571          }
572      }
573  }

```

```

567         k++;
568     }
569
570     if( transitivita==0){
571         j=verifica.dimensione;
572         i=verifica.dimensione;
573     }
574 }
575
576     j++;
577 }
578
579     i++;
580 }
581
582 }
583
584 /****** Controllo se la relazione Transitiva
585 ******/
586
587     if(transitivita == 1)
588         printf("true 'transitiva\n");
589
590     else
591         printf("non true 'transitiva\n");
592
593 /****** Fine controllo Transittivit *****
594 */
595
596     return(transitivita);
597 }
598 /****** Dicotomia ******/
599
600 int check_dicotomia(struct relBin verifica){
601
602     int a,b,c,d;
603     int numero_elementi;
604     int dicotomia = 0;
605     int dimensione;
606     int riscontro;
607     int secondo_riscontro;
608     a=0;

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

1024 return(suriettivita);
1025 }
1026
1027 /******FUNZIONE PER IL CHECK DELLA
BIETTIVITA'******/
1028
1029 void check_biettivita(struct relBin verifica){
1030
1031     int    suriettivita ,
1032           iniettivita;
1033
1034     suriettivita = check_suriettivita(verifica);
1035     iniettivita = check_iniettivita(verifica);
1036
1037
1038     if( suriettivita == 1 && iniettivita == 1)
1039         printf("\nla funzione e' biettiva");
1040     else
1041         printf("\nla funzione non e' biettiva");
1042     return;
1043 }

```

4.2 Test

```
1 #include<stdio.h>
2 #include"librerie/Progetto.h"
3
4 int main(void){
5     struct relBin RelazioneBinaria;
6     int scelta;
7     int scan;
8
9     printf("\nProgramma per effettuare i Test sulla
        libreria\n");
10
11
12     printf("\n\nDigita il numero corrispondente all'
        azione che si vuole svolgere\n");
13     printf("\n1) Test Acquisizione\n2) Esci\n");
14
15
16     while((scelta < 1) || (scelta > 2) || scan != 1){
17         printf("\nscelta:");
18         fflush(stdin);
19         scan = scanf("%d",&scelta);
20     }
21     if(scelta == 1)
22         RelazioneBinaria = acquisizione(RelazioneBinaria);
23     if(scelta == 2){
24         printf("\n\n..... Test terminati ..... \n\n");
25         return(0);
26     }
27     scelta = -1;
28     while(scelta != 7){
29         printf("\n\nDigita il numero corrispondente all'
            azione che si vuole svolgere\n");
30         printf("\n1) Test Acquisizione\n2) Test Stampa\n
            3) Test verifica ordine parziale\n4) Test
            verifica ordine totale");
31         printf("\n5) Test verifica relazione d'equivalenza\
            n6) Test funzione\n7) Esci\n");
32         scelta = -1;
33         while((scelta < 1) || (scelta > 7) || scan != 1){
34             printf("\nscelta:");
35             fflush(stdin);
36             scan = scanf("%d",&scelta);
```



```

37     }
38
39
40     if(scelta == 1)
41         RelazioneBinaria = acquisizione(RelazioneBinaria);
42     if(scelta == 2)
43         stampa(RelazioneBinaria);
44     if(scelta == 3)
45         ordine_parziale(RelazioneBinaria);
46     if(scelta == 4)
47         ordine_totale(RelazioneBinaria);
48     if(scelta == 5)
49         relazione_equivalenza(RelazioneBinaria);
50     if(scelta == 6)
51         check_funzione(RelazioneBinaria);
52     if(scelta == 7){
53         printf("\n\n..... Test_terminati.....\n\n");
54         return(0);
55     }
56 }
57 return(0);
58
59 }

```

4.3 Makefile

```
Test.exe: Test.c Makefile
gcc -ansi -Wall -O Test.c -o Test.exe
pulisci:
rm -f Test.o
pulisci_tutto:
rm -f Test.exe Test.o
```

5 Testing del programma

5.1 Test 1:

Test di Relazione d'ordine Totale.

Inputs: (a,a)(a,b)(b,b)

Outputs: checkriflessività : 1, checksimmetria : 0, checktransitività : 1
checkdicotomia : 1, la relazione è una relazione d'ordine totale in quanto è
rispetta anche la proprietà di Dicotomia.

```
6> Test funzione
7> Esci
scelta: 2
La relazione binaria e':
<(a,a);(a,b);(b,b) >

... Stampa Terminata ...

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

```
La relazione:
e' riflessiva
e' asimmetrica
e' transitiva
Quindi e' una relazione d'ordine parziale

... Controllo Ordine Parziale Terminato ...

e' dicotomica
Quindi e' una relazione d'ordine totale
... Controllo Ordine Totale Terminato ...
```

5.2 Test 2:

Test di Relazione d'ordine Parziale.

Inputs:(a,a)(b,b)(a,b)(c,c)

Outputs:checkriflessività : 1, checksimmetria : 0, checktransitività : 1 la relazione è una relazione d'ordine parziale in quanto rispetta le proprietà.

```
6> Test funzione
7> Esci

scelta: 2

La relazione binaria e':
<(a,a);(a,b);(b,b);(c,c) >

... Stampa Terminata ...

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

scelta:
```

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

scelta: 3

La relazione:
e' riflessiva
e' asimmetrica
e' transitiva

Quindi e' una relazione d'ordine parziale

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

5.3 Test 3:

Test di Relazione d'ordine non Parziale.

Inputs:(a,a)(b,b)(c,c)(d,d)(e,e)(a,b)(b,c)

Outputs:checkriflessività : 1, checksimmetria : 0, checktransitività : 0 la relazione non è una relazione d'ordine parziale in quanto non rispetta le proprietà.

```
6> Test funzione
7> Esci

scelta: 2

La relazione binaria e':

<(a,a);<(b,b);<(c,c);<(d,d);<(e,e);<(a,b);<(b,c) >

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere

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

scelta:
```

```
6> Test funzione
7> Esci

scelta: 3

La relazione:

e' riflessiva
e' asimmetrica
non e' transitiva

Non e' una relazione d'ordine parziale in quanto non rispetta tutte le proprietà
, manca la proprietà di transitività

... Controllo Ordine Parziale Terminato ...

Digita il numero corrispondente all'azione che si vuole svolgere
```

5.4 Test 4:

Test di Relazione d'equivalenza.

Inputs:(a,a)(a,b)(b,a)(b,b)

Outputs:checkriflessività : 1, checksimmetria : 1, checktransitività : 1 checkdicotomia : 0, la relazione è una relazione d'equivalenza in quanto rispetta le proprietà.

```
6> test funzione
7> Esci

scelta: 2

La relazione binaria e':
<(a,a);(a,b);(b,a);(b,b)>

... Stampa Terminata ...

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

scelta:
```

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

scelta: 5
e' riflessiva
e' simmetrica
e' transitiva

Quindi e' una relazione di equivalenza

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

scelta:
```

5.5 Test 5:

Test di Relazione non d'equivalenza.

Inputs:(a,a)(a,b)(b,c)

Outputs:checkriflessività : 0, checksimmetria : 0, checktransitività : 0 la relazione non è una relazione d'ordine d'equivalenza in quanto non rispetta le proprietà.

```
6> Test funzione
7> Esci

scelta: 2

La relazione binaria e':
<(a,a);(a,b);(b,c)>

... Stampa Terminata ...

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

scelta:
```

```
7> Esci

scelta: 5
non e' riflessiva
e' asimmetrica
non e' transitiva

Quindi non e' una relazione di equivalenza perche' non riflessiva
Quindi non e' una relazione di equivalenza perche' non simmetrica
Quindi non e' una relazione di equivalenza perche' non transitiva

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

scelta:
```

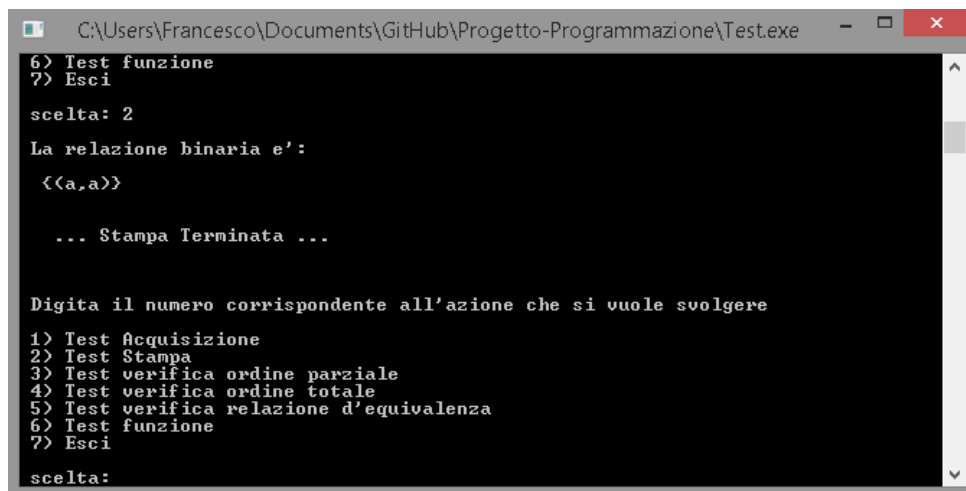
5.6 Test 6:

Test di Funzione.

Inputs:(a,a) Outputs:La relazione binaria è una funzione.

La relazione binaria è iniettiva.

La relazione binaria è biiettiva.



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

scelta: 2

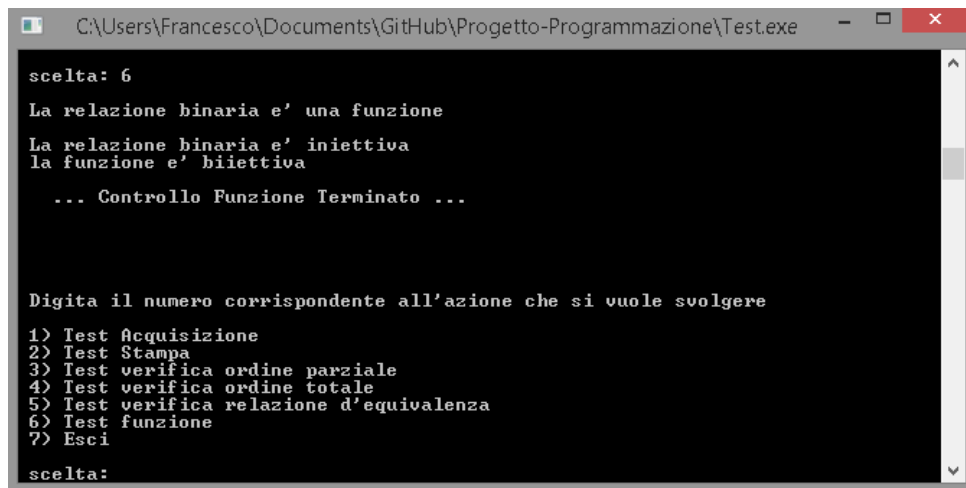
La relazione binaria e':

  <<a,a>>

... Stampa Terminata ...

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

scelta:
```



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

scelta: 6

La relazione binaria e' una funzione
La relazione binaria e' iniettiva
la funzione e' biiettiva

... Controllo Funzione Terminato ...

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

scelta:
```

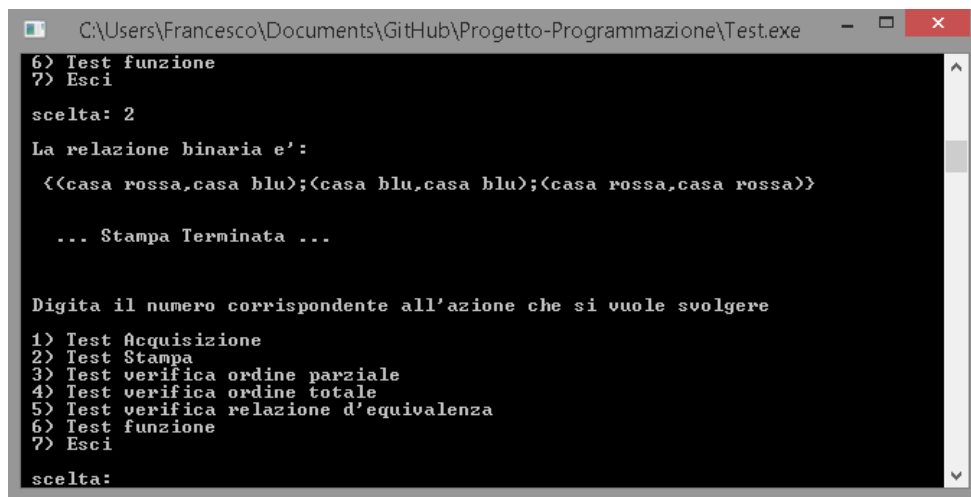

5.7 Test 7:

Test per verificare il controllo degli inputs.

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

Outputs:check_riflessività : 1,check_simmetria : 1, check_transitività : 1
dicotomia :1 la relazione è una relazione d'ordine totale in quanto rispetta le proprietà.

le funzioni funzionano anche con input contenuti degli spazi.



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

scelta: 2

La relazione binaria e':

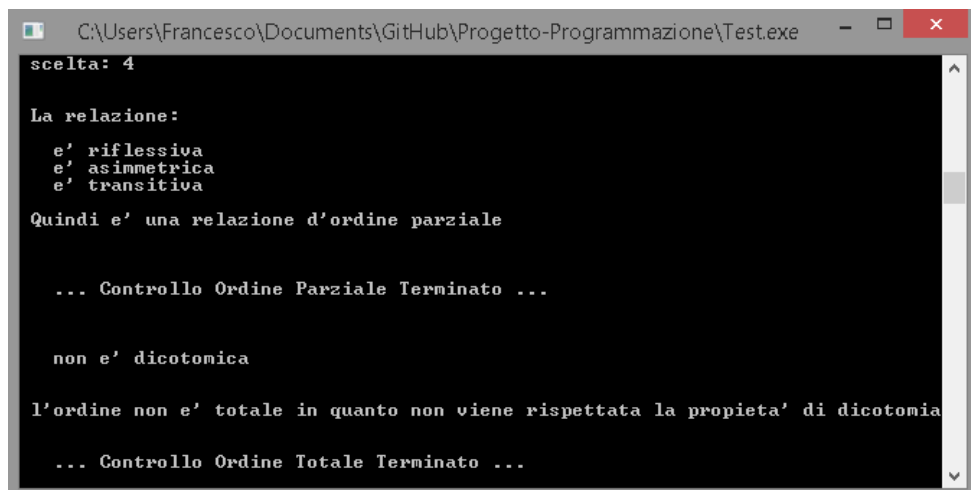
<(casa rossa,casa blu);(casa blu,casa blu);(casa rossa,casa rossa)>

... Stampa Terminata ...

Digita il numero corrispondente all'azione che si vuole svolgere

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

scelta:
```



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

scelta: 4

La relazione:

e' riflessiva
e' asimmetrica
e' transitiva

Quindi e' una relazione d'ordine parziale

... Controllo Ordine Parziale Terminato ...

non e' dicotomica

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

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

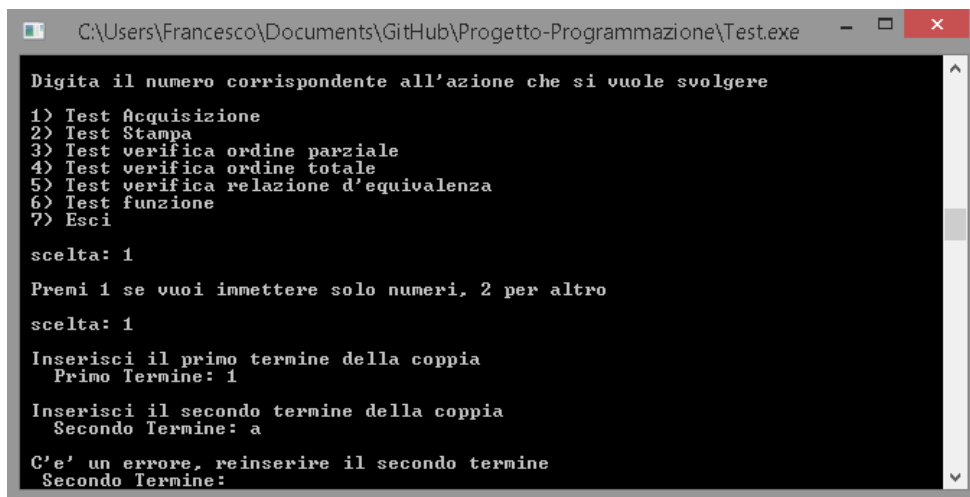
5.8 Test 8:

Test per inserire stringhe in una relazione numerica.

Inputs:(1,a)

Outputs: c' è un errore reinserisci il valore.

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



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

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

scelta: 1

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

Inserisci il primo termine della coppia
Primo Termine: 1

Inserisci il secondo termine della coppia
Secondo Termine: a

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

5.9 Test 9:

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

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

Outputs: La relazione binaria non è una funzione

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

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

scelta: 2

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

... Stampa Terminata ...

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

scelta:
```

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

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

La relazione binaria non e' una funzione

... Controllo Funzione Terminato ...

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

scelta:
```

5.10 Test 10:

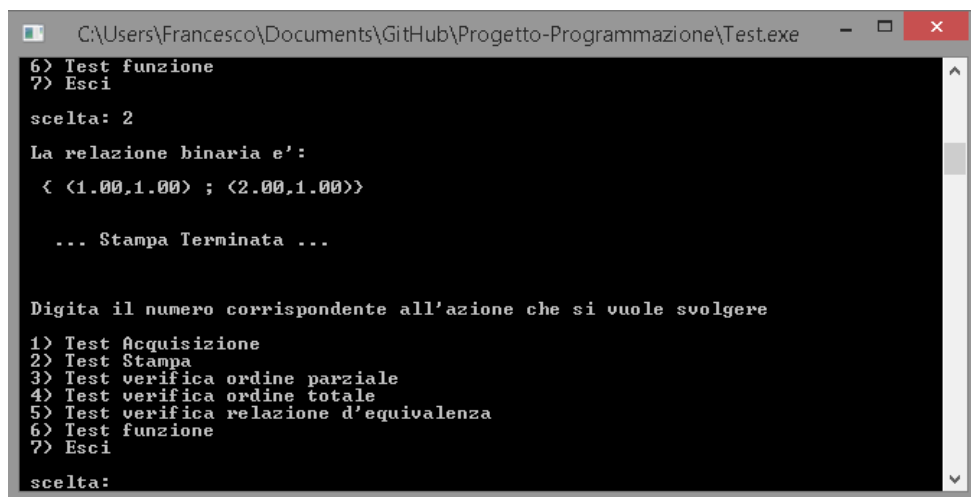
Inputs:(1,1)(2,1)

Outputs: La relazione binaria è una funzione

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

La funzione non è iniettiva

La funzione non è biiettiva



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

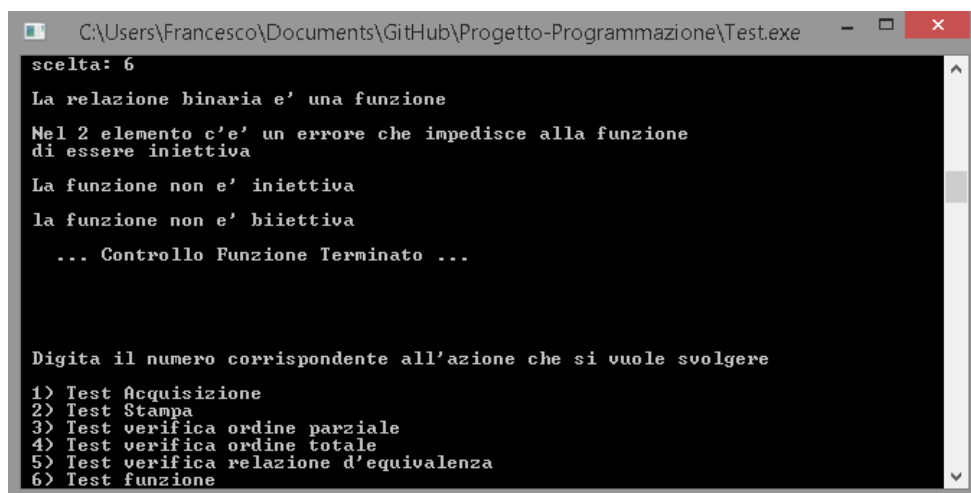
scelta: 2

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

... Stampa Terminata ...

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

scelta:
```



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

scelta: 6

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

... Controllo Funzione Terminato ...

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

6 Verica del programma

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

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

La postcondizione è

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

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

-Il predicato

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

e la funzione :

$$tr(numero_elementi) = numero_elementi - 1)$$

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

$$*\{P \wedge numero_elementi > 0\} c = c + numero_elementi; numero_elementi = numero_elementi - -; \{P\}$$

segue da :

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

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

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

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

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

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

* La limitatezza segue da:

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

$$\begin{aligned} &| = numero_elementi > numero_elementi - 1 \\ &\text{Poichè:} \end{aligned}$$

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

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

-Proseguendo infine a ritroso si ottiene prima:

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

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

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