University of Urbino

applied computer science

Procedural and logic programming

Report

Project for the 2014/2015 winter session

Studente:
Marco Tamagno
matricola no:

Studente:
Francesco Belacca
matricola no:

Lecturer:
Marco Bernardo

January 10, 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 37
4.3 Makefile 38

5 Testing the program **39**

6 Verifying the program **42**

1 Specifica del Problema

Write an ANSI C library that manages binary relations by exporting the following functions. The rst 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 fth 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 rst-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 ‘e una relazione d′ ordine parziale, stampando a video quale propriet‘a 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 ‘e una relazione d′ ordine totale, stampando a video quale propriet‘a 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 ‘e una relazione d′ equivalenza, stampando a video quale proprieta‘ 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 ‘e una funzione matematica; se non lo ‘e, allora si stampera‘ a video quale elemento violi la propriet‘a, altrimenti si stamper‘a a video un messaggio che indica se la funzione ‘e iniettiva, suriettiva o biiettiva. [Il progetto pu‘o 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 propietà della funzione, per poter mostrare a schermo quali popietà 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 proprieta′ 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 propietà della funzione, per poter mostrare a schermo quali popietà 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 propietà.

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 propietà 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 -¿ 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 acquisice 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 propietà 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 potra 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 propietà, 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 forzatamente 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 propietà.

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 e′ 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 e′ una relazione di ordine parziale, mentre nel caso in cui non siano verificate si stampera′ che non lo e′ e il perche′ (cioe′ quale proprieta′ non e′ 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 e′ di ordine totale, mentre se non lo e′ 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 e′ 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 e′ 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 e′ 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  /***** Creo una struttura dove salvare le coppie
       appartenenti alla Relazione*******/
7
8  struct relBin{
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
         coppia numerica o no e il numero delle coppie****
         */
18   int controllo,
19     dimensione;
20 };
21
22 /*DICHIARO LE FUNZIONI*/
23 int check_simmetria(struct relBin);
24 int check_riflessivita(struct relBin);
25 int check_transitivita(struct relBin);
26 int check_suriettivita(struct relBin);
27 void check_biiettivita(struct relBin);
28
29 /******************Funzione di acquisizione
       ******************/
30
31 struct relBin acquisizione(struct relBin relazione){
32
33 int acquisizione_finita = 0;
34 int scan = 0;
35
```

```
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 immettere solo numeri, 2
     per altro\n ");
46 scan = scanf("%d",&relazione.controllo);
47 }
48
49 /** risetto scan a 0 **/
50 scan=0;
51
52 /*Acquisizione Numerica*/
53
54 if(relazione.controllo == 1){
55   while(acquisizione_finita == 0){
56     relazione.dimensione++;
57     acquisizione_finita = 2;
58
59  /*Acquisisco il primo termine della coppia*/
60
61     printf("\n Inserisci il primo termine della coppia
         \n ");
62     relazione.primo_termine = (double *) realloc(
         relazione.primo_termine, (relazione.dimensione
         +1) * sizeof(double));
63
64  /*Check del primo termine della coppia*/
65
66   while((scanf("%lf",&relazione.primo_termine[
        relazione.dimensione - 1])) != 1){
67     fflush(stdin);
68     printf("\n C'e' un errore, reinserire il primo
         termine\n");
69     }
70
71  /*Acquisisco il secondo termine della coppia*/
72
```

```
73        printf("\n Inserisci il secondo termine della
              coppia \n ");
74        relazione.secondo_termine = (double *) realloc(
              relazione.secondo_termine, (relazione.
              dimensione+1) * sizeof(double));
75
76  /*Check del secondo termine della coppia*/
77
78      while((scanf("%lf",&relazione.secondo_termine[
            relazione.dimensione - 1])) != 1){
79        fflush(stdin);
80        printf("\n C'e' un errore, reinserire il secondo
              termine\n");
81        }
82
83    /*Chiedo all'utente se ci sono altre coppie*/
84
85      while(acquisizione_finita < 0 || acquisizione_finita
            > 1 || scan != 1){
86        printf("\n Vuoi acquisire un'altra coppia? immetti
              1 per uscire, 0 per continuare\n ");
87        fflush(stdin);
88        scan = scanf("%d",&acquisizione_finita);
89        }
90      }
91  }
92
93  /*risetto scan a 0*/
94  scan = 0;
95
96  /*Acquisizione con stringhe*/
97  if(relazione.controllo == 2){
98    while(acquisizione_finita == 0){
99      relazione.dimensione++;
100     acquisizione_finita = 2;
101
102   /*Acquisisco il primo termine della coppia*/
103
104     printf(" Inserisci il primo termine della coppia \
            n ");
105     relazione.prima_stringa[relazione.dimensione - 1]
            = (char *) malloc(50);
106     scan = scanf(" %[^\n]s",relazione.prima_stringa[
            relazione.dimensione - 1]);
```

```
107
108  /*Acquisisco il secondo termine della coppia*/
109
110      printf("␣Inserisci␣il␣secondo␣termine␣della␣coppia
             ␣\n␣");
111      relazione.seconda_stringa[relazione.dimensione -
             1] = (char *) malloc(50);
112      scan = scanf("␣%[^\n]s",relazione.seconda_stringa[
             relazione.dimensione - 1]);
113
114 /*risetto scan a 0*/
115 scan = 0;
116
117  /*Chiedo all'utente se ci sono altre coppie*/
118
119      while(acquisizione_finita < 0 ||
             acquisizione_finita > 1 || scan != 1){
120
121        printf("\n␣Vuoi␣acquisire␣un'altra␣coppia?␣
             immetti␣1␣per␣uscire,␣0␣per␣continuare\n");
122        scan = scanf("%d",&acquisizione_finita);
123        }
124      }
125    }
126
127 printf("\n\n␣␣␣...␣Acquisizione␣Terminata␣...\n\n");
128 return relazione;
129 }
130
131 /*****************************FUNZIONE DI STAMPA
      ******************************/
132
133 void stampa(struct relBin stampa){
134
135  int i = 0;
136
137  printf("\n␣La␣relazione␣binaria␣e':");
138  printf("\n\n␣␣{");
139
140 /******Stampa per coppie numeriche *****/
141
142     if(stampa.controllo == 1){
143       while(i < stampa.dimensione){
144
```

```
145           printf("␣(%.2lf,%.2lf)",stampa.primo_termine[i
                  ],stampa.secondo_termine[i]);
146         if(i+1 != stampa.dimensione)
147             printf("␣;");
148         i++;
149       }
150     }
151
152   /********Stampa  per coppie non numeriche *********/
153
154     if(stampa.controllo == 2){
155         while(i < stampa.dimensione){
156           printf("(%s,%s)",stampa.prima_stringa[i],
                  stampa.seconda_stringa[i]);
157          if(i+1 != stampa.dimensione)
158           printf(";");
159       i++;
160
161       }
162     }
163
164 /***************** Fine Stampa ******************/
165
166     printf("␣}\n");
167     printf("\n\n␣␣␣...␣Stampa␣Terminata␣...\n\n");
168
169 }
170
171 /*****************FUNZIONE DI VERIFICA DI RELAZIONI D
      'ORDINE*****************/
172
173 int ordine_parziale(struct relBin verifica){
174
175     int riflessivita,
176       transitivita,
177       simmetria,
178       parziale;
179
180     /*STAMPO LE PROPIETA' DELLA RELAZIONE*/
181
182     printf("\n\n␣La␣relazione:\n\n");
183
184 /********* Chiamo le funzioni per poter stabilire le
       propiet *****************/
```

```
185
186     riflessivita = check_riflessivita(verifica);
187     simmetria = check_simmetria(verifica);
188     transitivita = check_transitivita(verifica);
189
190 /************ Controllo se rispetta le propiet per
        essere una relazione d'ordine parziale***********/
191
192     if(transitivita == 1 && simmetria == 0 &&
          riflessivita == 1){
193       parziale = 1;
194       printf("\n Quindi e' una relazione d'ordine
            parziale\n\n");
195     }
196     else{
197
198       printf("\n Non e' una relazione d'ordine parziale
            in quanto non rispetta tutte le propieta'\n");
199       parziale = 0;
200     }
201       if(transitivita == 0)
202       printf("\n manca la propieta' di transitivita'\n")
            ;
203       if(simmetria == 1)
204       printf("\n manca la propieta' di asimmetria'\n");
205       if(riflessivita == 0)
206       printf("\n manca la propieta' di asimmetria'\n");
207 /************ Fine controllo Ordine Parziale
        *********************/
208
209     printf("\n\n    .... Controllo Ordine Parziale
          Terminato ...\n\n\n\n");
210     return(parziale);
211 }
212
213
214 /**********FUNZIONE PER CONTROLLARE LA RIFLESSIVIT
        **********/
215
216 int check_riflessivita (struct relBin verifica){
217
218     int i,
219       j,
220       k,
```

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

```
257
258           while(j < verifica.dimensione){
259             if(j == i)
260               j++;
261             else{
262               if(verifica.primo_termine[i] == verifica.
                    primo_termine[j])
263                 if(verifica.primo_termine[j] == verifica
                      .secondo_termine[j])
264                   riscontro++;
265
266             j++;
267           }
268         }
269
270         j = 0;
271
272 /***************** Controllo la riflessivit per gli
      elementi del secondo insieme
      ****************************/
273
274           while(j < verifica.dimensione){
275             if(j == k)
276               j++;
277             else{
278               if(verifica.secondo_termine[k] == verifica
                    .secondo_termine[j])
279                 if(verifica.primo_termine[j] == verifica
                      .secondo_termine[j])
280                   secondo_riscontro++;
281
282             j++;
283           }
284         }
285           if(riscontro != 0)
286             i++;
287
288 /**** Se non c' stato un riscontro di riflessivit
      esco e setto la riflessivit a 0 *****/
289
290           else{
291             i=verifica.dimensione;
292             riflessivita = 0;
293           }
```

```
294
295           if(secondo_riscontro != 0)
296             k++;
297
298           else{
299             k=verifica.dimensione;
300             riflessivita = 0;
301           }
302         }
303
304       }
305
306     /************** VERIFICA RIFLESSIVIT PER STRINGHE
          ****************/
307
308     if(verifica.controllo == 2){
309       riscontro = 0;
310       secondo_riscontro = 0;
311       if(strcmp(verifica.prima_stringa[i],verifica.
            seconda_stringa[i]) == 0)
312         riscontro++;
313         secondo_riscontro++;
314       if(riscontro != 0){
315         i++;
316         k++;
317       }
318
319       else{
320         j=0;
321         riscontro = 0;
322         secondo_riscontro = 0;
323
324   /***************** Controllo la riflessivit per gli
        elementi del primo insieme
        *****************************/
325
326         while(j < verifica.dimensione){
327           if(j == i)
328             j++;
329           else{
330             if(strcmp(verifica.prima_stringa[i],verifica
                  .prima_stringa[j]) == 0)
331               if(strcmp(verifica.prima_stringa[j],
                    verifica.seconda_stringa[j]) == 0)
```

```
332                 riscontro++;
333
334             j++;
335           }
336         }
337
338         j = 0;
339
340 /***************** Controllo la riflessivit per gli
      elementi del secondo insieme
      ****************************/
341
342         while(j < verifica.dimensione){
343           if(j == k)
344           j++;
345           else{
346             if(strcmp(verifica.seconda_stringa[k],
                  verifica.seconda_stringa[j]) == 0)
347               if(strcmp(verifica.prima_stringa[j],
                    verifica.seconda_stringa[j]) == 0)
348                 secondo_riscontro++;
349
350             j++;
351           }
352         }
353         if(riscontro != 0)
354           i++;
355
356         else{
357           i=verifica.dimensione;
358           riflessivita = 0;
359         }
360
361         if(secondo_riscontro != 0)
362           k++;
363
364         else{
365           k=verifica.dimensione;
366           riflessivita = 0;
367         }
368       }
369
370     }
371
```

```
372 }
373
374 /********* Controllo se    riflessiva
        ******************/
375
376   if(riflessivita == 1)
377     printf("    e' riflessiva\n");
378   else
379     printf("    non e' riflessiva\n");
380
381 /********** Fine riflessivita **********************
        */
382
383   return(riflessivita);
384 }
385
386
387
388 /*************************** FUNZIONE PER CONTROLLARE
        LA SIMMETRIA ******************/
389
390 /************* Definizione: In matematica, una
        relazione binaria R in un insieme X   **/
391 /************* simmetrica se e solo se, presi due
        elementi qualsiasi a e b, vale che  **/
392 /************* se a   in relazione con b allora anche
        b   in relazione con a. *********/
393
394 int check_simmetria(struct relBin verifica){
395
396   int i,
397     j,
398     riscontro,
399     simmetria;
400
401   simmetria = 1;
402
403
404   i = 0;
405   j = 0;
406   riscontro = 0;
407
408 /*Check della simmetria per numeri*/
409
```

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

```
450
451         if(riscontro == 0){
452           j = verifica.dimensione;
453           i = verifica.dimensione;
454           simmetria = 0;
455         }
456         riscontro = 0;
457         i++;
458       }
459
460     }
461
462   /***** Controllo se la simmetria   stata verificata
        *********/
463
464     if(simmetria == 1)
465       printf("    e' simmetrica\n");
466     else
467       printf("    e' asimmetrica\n");
468
469   /****** Fine controllo simmetria ******/
470
471     return(simmetria);
472   }
473
474
475
476   /* FUNZIONE PER CONTROLLARE LA TRANSITIVIT */
477
478   /****** Definizione: In matematica, una relazione
        binaria R in un insieme X   transitiva se e solo se
479       per ogni a, b, c appartenenti ad X, se a   in
            relazione con b e b   in relazione con c,
            allora
480        a   in relazione con c.*******/
481
482
483   int check_transitivita(struct relBin verifica){
484
485     int i,
486       j,
487       k,
488       transitivita;
489
```

```
490 /*SETTO LA TRANSITIVITA INIZIALMENTE COME VERA E
      AZZERO I CONTATORI*/
491   transitivita = 1;
492   i = 0;
493   j = 0;
494   k = 0;
495
496 /*VERIFICA TRANSITIVIT PER NUMERI*/
497
498
499   if(verifica.controllo == 1){
500
501     while(i < verifica.dimensione){
502       j = 0;
503
504       while(j < verifica.dimensione){
505         k=0;
506
507         if(verifica.secondo_termine[i] == verifica.
              primo_termine[j]){
508           transitivita = 0;
509
510           while(k < verifica.dimensione){
511             if(verifica.primo_termine[i] == verifica.
                  primo_termine[k]){
512               if(verifica.secondo_termine[k]==verifica
                    .secondo_termine[j]){
513                 transitivita = 1;
514                 j = verifica.dimensione;
515                 k = verifica.dimensione;
516               }
517             }
518
519              k++;
520           }
521
522         }
523
524         j++;
525       }
526
527       i++;
528     }
529   }
```

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

```
569     if(transitivita == 1)
570       printf("   e' transitiva\n");
571
572     else
573       printf("   non e' transitiva\n");
574
575 /*********** Fine controllo Transitivit **********
        */
576
577     return(transitivita);
578
579 }
580
581 /********** Dicotomia ************/
582
583 int check_dicotomia(struct relBin verifica){
584
585     int a,b,c,d;
586     int numero_elementi;
587     int dicotomia = 0;
588     int dimensione;
589     int riscontro;
590     int secondo_riscontro;
591     a=0;
592     b=0;
593     c=0;
594     d=a-1;
595     dimensione = verifica.dimensione;
596
597 /********* Dicotomia per numeri *********/
598
599     if(verifica.controllo == 1){
600
601 /********** Conto il numero delle coppie esistenti (
        scarto le coppie uguali) *********/
602
603       while( a < verifica.dimensione){
604         d = a-1;
605         b = a+1;
606         secondo_riscontro = 0;
607
608       if(a>0){
609         while ( d >= 0 ){
```

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

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

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

```
724 /************ Conto quanti dovrebbero essere gli
       elementi per avere la dicotomia **********/
725
726     while(numero_elementi > 0){
727
728       numero_elementi--;
729       c = c + numero_elementi;
730
731     }
732
733   }
734
735 /************ Verifico se la dicotomia   verificata
       ***************/
736
737   if(dimensione == c)
738     dicotomia = 1;
739
740   if(dicotomia == 1 && (check_riflessivita(verifica)
       == 1))
741     printf("    e' dicotomica\n\n");
742
743   else
744     printf("    non e' dicotomica\n\n");
745
746 /***************** Fine verifica dicotomia
       *****************/
747
748   return(dicotomia);
749 }
750
751 /*Funzione di verifica dell'ordine totale*/
752
753
754 void ordine_totale (struct relBin verifica){
755
756   int parziale,
757     dicotomia;
758
759   parziale = ordine_parziale (verifica);
760   dicotomia = check_dicotomia (verifica);
761
762   if(parziale == 0)
```

```
763       printf("\n l'ordine non e' totale in quanto non e
              ' nemmeno parziale");
764
765     if(dicotomia == 0)
766       printf("\n l'ordine non e' totale in quanto non
              viene rispettata la propieta' di dicotomia");
767
768     if(dicotomia == 1 && parziale == 1)
769       printf("\n Quindi e' una relazione d'ordine totale
              ");
770
771     printf("\n\n..... Controllo Ordine Totale Terminato
          ...\n\n\n\n");
772   }
773
774   /*Funzione che stabilisce se e' una relazione di
        equivalenza o meno*/
775
776   void relazione_equivalenza(struct relBin verifica){
777
778     int riflessivita;
779     int simmetria;
780     int transitivita;
781
782     riflessivita = check_riflessivita(verifica);
783     simmetria = check_simmetria(verifica);
784     transitivita = check_transitivita(verifica);
785
786     if(riflessivita == 1 && simmetria == 1 &&
            transitivita == 1)
787     printf("\n Quindi e' una relazione di equivalenza\n"
            );
788
789     if(riflessivita == 0)
790     printf("\n Quindi non e' una relazione di
            equivalenza perche' non riflessiva\n");
791
792     if(simmetria == 0)
793     printf("\n Quindi non e' una relazione di
            equivalenza perche' non simmetrica\n");
794
795     if(transitivita == 0)
796     printf("\n Quindi non e' una relazione di
            equivalenza perche' non transitiva\n");
```

```
797 }
798
799 /*Funzione che stabilisce se la relazione binaria
        acquisita e' una funzione matematica*/
800
801 void check_funzione(struct relBin verifica){
802
803   int i;
804   int k;
805   int termini_diversi;
806   int termini_uguali_prima;
807   int termini_uguali_dopo;
808   int errore;
809
810 if(verifica.controllo == 1){
811
812   i=0;
813   errore=0;
814   termini_diversi=0;
815   termini_uguali_dopo=0;
816   termini_uguali_prima=0;
817   while(i < verifica.dimensione){
818     k=verifica.dimensione-1;
819     termini_uguali_dopo=termini_uguali_prima;
820     while(k > i){
821       if(verifica.primo_termine[i] == verifica.
            primo_termine[k]){
822         if(verifica.secondo_termine[i] != verifica.
              secondo_termine[k]){
823           errore=1;
824           printf("\n Nel %d elemento c'e' un errore
                che impedisce alla realzione binaria\n",k
                +1);
825           printf("di essere una funzione\n");
826           k=i;
827           i=verifica.dimensione;
828         }
829         if(verifica.secondo_termine[i] == verifica.
              secondo_termine[k])
830           termini_uguali_dopo++;
831       }
832       k--;
833     }
```

```
834       if(errore == 0 && termini_uguali_dopo ==
              termini_uguali_prima)
835       termini_diversi++;
836
837       termini_uguali_prima = termini_uguali_dopo;
838       i++;
839     }
840     if(errore == 0 && (termini_diversi == (verifica.
          dimensione - termini_uguali_prima))){
841     printf("\n La relazione binaria e' una funzione\n");
842     check_biiettivita(verifica);
843   }
844     else
845     printf("\n La relazione binaria non e' una funzione\
          n");
846   }
847
848   /********** Controllo se c' una funzione per stringhe
        (le stringhe sono considerate come costanti di
        diverso valore) ***********/
849
850   if(verifica.controllo == 2){
851
852     i=0;
853     errore=0;
854     termini_diversi=0;
855     termini_uguali_dopo=0;
856     termini_uguali_prima=0;
857     while(i < verifica.dimensione){
858       k=verifica.dimensione-1;
859       termini_uguali_dopo=termini_uguali_prima;
860       while(k > i){
861         if((strcmp(verifica.prima_stringa[i],verifica.
              prima_stringa[k])) == 0){
862           if((strcmp(verifica.seconda_stringa[i],
                verifica.seconda_stringa[k])) != 0){
863             errore=1;
864             printf("\n Nel %d elemento c'e' un errore
                  che impedisce alla realzione binaria\n",k
                  +1);
865             printf("di essere una funzione\n");
866             k=i;
867             i=verifica.dimensione;
868           }
```

```
869           else
870             termini_uguali_dopo++;
871         }
872         k--;
873       }
874       if(errore == 0 && termini_uguali_dopo ==
             termini_uguali_prima)
875       termini_diversi++;
876
877       termini_uguali_prima = termini_uguali_dopo;
878       i++;
879     }
880     if(errore == 0 && (termini_diversi == (verifica.
          dimensione - termini_uguali_prima))){
881     printf("\n La relazione binaria e' una funzione\n");
882     check_biiettivita(verifica);
883     }
884     else
885     printf("\n La relazione binaria non e' una funzione\
          n");
886 }
887
888 printf("\n\n .... Controllo Funzione Terminato ... \n\
       n\n\n");
889
890 }
891
892 /**********FUNZIONE PER IL CHECK DELL'INIETTIVITA
       '************/
893
894 int check_iniettivita(struct relBin verifica){
895
896   int i;
897   int k;
898   int termini_diversi;
899   int termini_uguali_prima;
900   int termini_uguali_dopo;
901   int errore;
902   int iniettivita;
903
904   iniettivita = 0;
905
906 if(verifica.controllo == 1){
907
```

```
908     i=0;
909     errore=0;
910     termini_diversi=0;
911     termini_uguali_dopo=0;
912     termini_uguali_prima=0;
913
914     while(i < verifica.dimensione){
915
916       k=verifica.dimensione-1;
917       termini_uguali_dopo=termini_uguali_prima;
918       while(k > i){
919
920         if(verifica.secondo_termine[i] == verifica.
              secondo_termine[k]){
921
922           if(verifica.primo_termine[i] != verifica.
                primo_termine[k]){
923
924             errore=1;
925             printf("\n Nel %d elemento c'e' un errore
                    che impedisce alla realzione binaria\n",k
                    +1);
926             printf("di essere una funzione\n");
927             k=i;
928             i=verifica.dimensione;
929           }
930           if(verifica.primo_termine[i] == verifica.
                primo_termine[k])
931             termini_uguali_dopo++;
932         }
933         k--;
934       }
935       if(errore == 0 && termini_uguali_dopo ==
            termini_uguali_prima)
936       termini_diversi++;
937
938       termini_uguali_prima = termini_uguali_dopo;
939       i++;
940     }
941     if(errore == 0 && (termini_diversi == (verifica.
          dimensione - termini_uguali_prima))){
942     printf("\n La relazione binaria e' iniettiva\n");
943     iniettivita = 1;
944   }
```

```
945     else
946     printf("\n La relazione binaria non e' iniettiva\n")
            ;
947
948
949 }
950
951 /********** Controllo iniettivita' per stringhe
        ***********/
952
953 if(verifica.controllo == 2){
954
955     i=0;
956     errore=0;
957     termini_diversi=0;
958     termini_uguali_dopo=0;
959     termini_uguali_prima=0;
960
961     while(i < verifica.dimensione){
962       k=verifica.dimensione-1;
963       termini_uguali_dopo=termini_uguali_prima;
964       while(k > i){
965         if((strcmp(verifica.seconda_stringa[i],verifica.
              seconda_stringa[k])) == 0){
966           if((strcmp(verifica.prima_stringa[i],verifica.
                prima_stringa[k])) != 0){
967             errore=1;
968             printf("\n Nel %d elemento c'e' un errore
                  che impedisce alla realzione binaria\n",k
                  +1);
969             printf("di essere una funzione\n");
970             k=i;
971             i=verifica.dimensione;
972           }
973           if((strcmp(verifica.prima_stringa[i],verifica.
                prima_stringa[k])) == 0)
974             termini_uguali_dopo++;
975         }
976
977         k--;
978       }
979       if(errore == 0 && termini_uguali_dopo ==
            termini_uguali_prima)
980       termini_diversi++;
```

```
981
982       termini_uguali_prima = termini_uguali_dopo;
983       i++;
984     }
985     if(errore == 0 && (termini_diversi == (verifica.
          dimensione - termini_uguali_prima))){
986     printf("\n La relazione binaria e' iniettiva");
987     iniettivita = 1;
988     }
989     else
990     printf("\n La relazione binaria non e' iniettiva");
991   }
992
993   return(iniettivita);
994   }
995
996   /***************FUNZIONE PER IL CHECK DELLA
        SURIETTIVITA '***********/
997
998   int check_suriettivita(struct relBin verifica){
999
1000  /******* La suriettivit    sempre verificata in quanto
          il dominio e il codominio **********/
1001  /** sono entrambi i rispettivi x,y acquisiti, quindi
        non ho elementi y non associati a x **/
1002  int suriettivita;
1003
1004  suriettivita = 1;
1005  return(suriettivita);
1006  }
1007
1008  /***************FUNZIONE PER IL CHECK DELLA
        BIIETTIVITA '***********/
1009
1010  void check_biiettivita(struct relBin verifica){
1011
1012    int   surriettivita,
1013        iniettivita;
1014
1015  surriettivita = check_suriettivita(verifica);
1016  iniettivita = check_iniettivita(verifica);
1017
1018
1019    if( surriettivita == 1 && iniettivita == 1)
```

```
1020        printf("\n la funzione e' biiettiva");
1021      else
1022        printf("\n la funzione non e' biiettiva");
1023  return;
1024  }
```

4.2 Test

```
1  #include<stdio.h>
2  #include"Progetto.h"
3
4  int main(void){
5    struct relBin RelazioneBinaria;
6
7
8    RelazioneBinaria = acquisizione(RelazioneBinaria);
9
10   stampa(RelazioneBinaria);
11   ordine_totale(RelazioneBinaria);
12   relazione_equivalenza(RelazioneBinaria);
13   check_funzione(RelazioneBinaria);
14   return(0);
15 }
```

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 the program

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 propietà di Dicotomia.

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.

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.

Test 4:
Test di Relazione d′ equivalenza.

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

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

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.

Test 6:
Test di Funzione.

Inputs:(a,a) Outputs:La relazione binaria e′ una funzione.
La relazione binaria e′ iniettiva.
La relazione binaria e′ biiettiva.

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 contenti degli spazi.

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.

Test 9:
Inputs:(1,2)(1,1)

Outputs: La relazione binaria non è una funzione

Test 10:
Inputs:(1,1)(2,1)

Outputs:La relazione binaria e$'$ una funzione
La funzione binaria non e$'$ iniettiva
La funzione binaria non e$'$ biiettiva

6 Verifying the program