

UNIVERSITY OF URBINO

APPLIED COMPUTER SCIENCE

PROCEDURAL AND LOGIC PROGRAMMING

Report

PROJECT FOR THE 2014/2015 WINTER SESSION

Student:

Marco TAMAGNO
matric no: 260324

Student:

Francesco BELACCA
matric no: 260324

Lecturer:

Marco BERNARDO

January 10, 2015

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 di 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 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 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, 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, più precisamente in teoria degli ordini, una relazione d'ordine o ordine su di un insieme è una relazione binaria tra elementi appartenenti all'insieme che gode delle seguenti proprietà:

riflessiva * antisimmetrica * transitiva.

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

Funzioni per lacquisizione:

acquisizione() : per acquisire la relazione.

Funzioni per la verifica delle propriet:

check_iniettivita() : per controllare se liniettivit rispettata o meno (0 non c, 1 c).

check_transitivita() : per controllare se la transitivit viene rispettata o meno (0 non c, 1 c).

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

check_riflessivita() : per controllare se la riflessivit viene rispettata o meno (0 non c, 1 c).

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

check_suriettivita(): verifica se la funzione gode della propriet di suriettivit, in questo caso sar sempre settata a 1 in quanto tutti gli elementi del codominio (presi come gli elementi dei vari secondi termini digitati durante lacquisizione) avranno sempre un elemento del dominio associato .

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

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

relazione_equivalenza() : richiama le funzioni delle propriet e controlla se c una relazione dequivalenza(stampa a video se c o meno una relazione dequivalenza, e nel caso non c stampa quali propriet non vengono rispettate).

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

3.3 Input

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

L input dovr essere dotato di diversi controlli, se lutente 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 dovr essere data per dare la possibilit allutente nel caso scelga di fare uninput di tipo numerico di poter effettuare operazioni non legate alle funzioni della libreria, (esempio : lutente vuole decidere di moltiplicare linput per due, e vedere se mantiene le propriet, con uninput di tipo numerico lutente pu farlo e ci avrebbe un senso, con uninput di tipo stringa meno).

La scelta dellinput di tipo stringa dovr 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 verr data come input la struttura dati salvata in precedenza dalla funzione Acquisizione, per poterne verificare le varie propriet.

3.4 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 così perché non essendo permesso l'utilizzo di variabili globali, il modo più semplice di passare i dati inseriti da una funzione all'altra è quello di creare una struttura dinamica. Una volta restituito l'indirizzo della struttura, a seconda della funzione lanciata nel file Test.c si lanceranno le altre 5 funzioni, dato che queste prendono tutte in pasto l'output della prima (cioè l'indirizzo della struttura della relazione binaria) e la utilizzano per verificarne varie proprietà.

3.5 Output - stampa

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

3.6 Output - ordine_parziale

La funzione ordine_parziale avrà come output la stampa a video del risultato della verifica delle proprietà di riflessività, antisimmetria e transitività. Nel caso in cui siano tutte verificate si stamperà che la relazione è una relazione di ordine parziale, mentre nel caso in cui non siano verificate si stamperà che non lo è e il perché (cioè quale proprietà non è o non sono verificate).

3.7 Output - ordine_totale

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

3.8 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 dequivalenza.

3.9 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

```
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_innmettere_solo_numeri,_2_
    per_altro\n");
46 scan = scanf("%d",&relazione.controllo);
47 }
48
49 /** resetto 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'è 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 /* riassetto 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("\n Inserisci il primo termine della coppia\n
        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("\n Inserisci il secondo termine della coppia
111             \n");
112      relazione.seconda_stringa[relazione.dimensione -
113                                1] = (char *) malloc(50);
114      scan = scanf("%[^\\n]s", relazione.seconda_stringa[
115                  relazione.dimensione - 1]);
116
117  /* Chiedo all'utente se ci sono altre coppie */
118
119      while(acquisizione_finita < 0 ||
120            acquisizione_finita > 1 || scan != 1){
121
122          printf("\n Vuoi acquisire un'altra coppia?
123                 immetti 1 per uscire, 0 per continuare\n");
124          scan = scanf("%d",&acquisizione_finita);
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
146             ], stampa.secondo_termine[i]);
147     if(i+1 != stampa.dimensione)
148         printf("_;");
149     i++;
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],
157                 stampa.seconda_stringa[i]);
158             if(i+1 != stampa.dimensione)
159                 printf(";");
160             i++;
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
172 'ORDINE******/
173
174 int ordine_parziale(struct relBin verifica){
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
185     propriet ******/

```

```

185
186     riflessivita = check_riflessivita(verifica);
187     simmetria = check_simmetria(verifica);
188     transitivita = check_transitivita(verifica);
189
190     /****** Controllo se rispetta le propriet 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     ;
204     if(simmetria == 1)
205         printf("\n_manca_la_propieta'_di_asimmetria'\n");
206     if(riflessivita == 0)
207         printf("\n_manca_la_propieta'_di_asimmetria'\n");
208     /****** Fine controllo Ordine Parziale
        ******/
209
210     printf("\n\n..._Controllo_Ordine_Parziale_
        Terminato...\n\n\n");
211     return(parziale);
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.
312             seconda_stringa[i]) == 0)
313         riscontro++;
314         secondo_riscontro++;
315     if(riscontro != 0){
316         i++;
317         k++;
318     }
319
320     else{
321         j=0;
322         riscontro = 0;
323         secondo_riscontro = 0;
324
325 /****** Controllo la riflessivita per gli
elementi del primo insieme
******/
326
327         while(j < verifica.dimensione){
328             if(j == i)
329                 j++;
330             else{
331                 if(strcmp(verifica.prima_stringa[i], verifica
332                         .prima_stringa[j]) == 0)
333                     if(strcmp(verifica.prima_stringa[j],
334                             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],
347                     verifica.seconda_stringa[j]) == 0)
348                 if(strcmp(verifica.prima_stringa[j],
349                     verifica.seconda_stringa[j]) == 0)
350                     secondo_riscontro++;
351             j++;
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("L' e' riflessiva\n");
378     else
379         printf("L 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         while( i < verifica.dimensione){
412             j = 0;
413             while( j < verifica.dimensione){
414                 if(verifica.primo_termine[i] == verifica .
                     secondo_termine[j])
415                     if(verifica.primo_termine[j] == verifica .
                         secondo_termine[i])
416                         riscontro++;
417
418                 j++;
419             }
420
421             if(riscontro == 0){
422                 j = verifica.dimensione;
423                 i = verifica.dimensione;
424                 simmetria = 0;
425             }
426             riscontro = 0;
427             i++;
428         }
429
430     }
431
432     /* Check della simmetria per stringhe */
433
434     if(verifica.controllo == 2){
435         while( i < verifica.dimensione){
436             j = 0;
437             while( j < verifica.dimensione){
438                 if(strcmp(verifica.prima_stringa[i], verifica .
                     seconda_stringa[j]) == 0 )
439                     if(strcmp(verifica.prima_stringa[j], verifica
                     .seconda_stringa[i]) == 0 )
440                         riscontro++;
441
442                 j++;
443             }
444
445             if(riscontro == 0){
446                 j = verifica.dimensione;
447                 i = verifica.dimensione;
448                 simmetria = 0;
449             }

```

```

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

```

```

489
490  /*VERIFICA TRANSITIVIT PER NUMERI*/
491
492
493  if(verifica.controllo == 1){
494
495      while(i < verifica.dimensione){
496          j = 0;
497
498          while(j < verifica.dimensione){
499              k=0;
500
501              if(verifica.secondo_termine[i] == verifica.
                    primo_termine[j]){
502                  transitivita = 0;
503
504                  while(k < verifica.dimensione){
505                      if(verifica.primo_termine[i] == verifica.
                    primo_termine[k]){
506                          if(verifica.secondo_termine[k]==verifica
                    .secondo_termine[j]){
507                              transitivita = 1;
508                              j = verifica.dimensione;
509                              k = verifica.dimensione;
510                          }
511                      }
512
513                      k++;
514                  }
515
516              }
517
518              j++;
519          }
520
521          i++;
522      }
523  }
524
525
526  /****** VERIFICA TRANSITIVIT PER STRINGHE
        *****/
527
528  if(verifica.controllo == 2){

```



```

529
530
531     while(i < verifica.dimensione){
532         j = 0;
533
534         while(j < verifica.dimensione){
535             k=0;
536
537             if(strcmp(verifica.seconda_stringa[i],verifica
                    .prima_stringa[j]) == 0){
538                 transitivita = 0;
539
540                 while(k < verifica.dimensione){
541                     if(strcmp(verifica.prima_stringa[i],
                            verifica.prima_stringa[k]) == 0){
542                         if(strcmp(verifica.seconda_stringa[k],
                            verifica.seconda_stringa[j]) == 0){
543                             transitivita = 1;
544                             j = verifica.dimensione;
545                             k = verifica.dimensione;
546                         }
547                     }
548
549                     k++;
550                 }
551             }
552
553             j++;
554         }
555
556         i++;
557     }
558
559 }
560
561 /***** Controllo se la relazione      Transitiva
        *****/
562
563 if(transitivita == 1)
564     printf("true 'transitiva\n");
565
566 else
567     printf("non e 'transitiva\n");
568

```

```

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

```

```

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

```

```

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

```

```

685         }
686         b++;
687     }
688 }
689 a++;
690 }
691
692
693 a=0;
694 b=0;
695 c=0;
696 numero_elementi = 0;
697
698 /****** Conto il numero degli elementi
distinti esistenti *****/
699
700 while(a<verifica.dimensione){
701     d=a-1;
702     secondo_riscontro = 0;
703
704     while(d >= 0){
705         if((strcmp(verifica.prima_stringa[a], verifica.
706             prima_stringa[d])) == 0)
707             secondo_riscontro = 1;
708         d--;
709     }
710     if(secondo_riscontro != 1){
711         if((strcmp(verifica.prima_stringa[a], verifica.
712             seconda_stringa[a])) == 0)
713             numero_elementi++;
714     }
715     a++;
716 }
717 c = numero_elementi;
718 /****** Conto quanti dovrebbero essere gli
elementi per avere la dicotomia *****/
719
720 while(numero_elementi > 0){
721
722     numero_elementi--;
723     c = c + numero_elementi;
724

```

```

725     }
726
727 }
728
729 /****** Verifico se la dicotomia verificata
       ******/
730
731     if(dimensione == c)
732         dicotomia = 1;
733
734     if(dicotomia == 1 && (check_riflessivita(verifica)
       == 1))
735         printf("L'ordine e' dicotomica\n\n");
736
737     else
738         printf("L'ordine non e' dicotomica\n\n");
739
740 /****** Fine verifica dicotomia
       ******/
741
742     return(dicotomia);
743 }
744
745 /*Funzione di verifica dell'ordine totale*/
746
747
748 void ordine_totale (struct relBin verifica){
749
750     int parziale ,
751         dicotomia;
752
753     parziale = ordine_parziale (verifica);
754     dicotomia = check_dicotomia (verifica);
755
756     if(parziale == 0)
757         printf("L'ordine non e' totale in quanto non e'
       'nemmeno parziale");
758
759     if(dicotomia == 0)
760         printf("L'ordine non e' totale in quanto non
       viene rispettata la proprieta' di dicotomia");
761
762     if(dicotomia == 1 && parziale == 1)

```

```

763     printf("\n_Quindi_e'_una_relazione_d'ordine_totale
        ");
764
765     printf("\n\n..._Controllo_Ordine_Totale_Terminato
        ... \n\n\n");
766 }
767
768 /*Funzione che stabilisce se e' una relazione di
        equivalenza o meno*/
769
770 void relazione_equivalenza(struct relBin verifica){
771     int riflessivita;
772     int simmetria;
773     int transitivita;
774
775     riflessivita = check_riflessivita(verifica);
776     simmetria = check_simmetria(verifica);
777     transitivita = check_transitivita(verifica);
778
779     if(riflessivita == 1 && simmetria == 1 &&
        transitivita == 1)
780     printf("\n_Quindi_e'_una_relazione_di_equivalenza\n"
        );
781
782     if(riflessivita == 0)
783     printf("\n_Quindi_non_e'_una_relazione_di_
        equivalenza_perche'_non_riflessiva\n");
784
785     if(simmetria == 0)
786     printf("\n_Quindi_non_e'_una_relazione_di_
        equivalenza_perche'_non_simmetrica\n");
787
788     if(transitivita == 0)
789     printf("\n_Quindi_non_e'_una_relazione_di_
        equivalenza_perche'_non_transitiva\n");
790 }
791
792 /*Funzione che stabilisce se la relazione binaria
        acquisita e' una funzione matematica*/
793
794 void check_funzione(struct relBin verifica){
795     int i;
796     int k;
797     int termini_diversi;

```

```

798     int termini_uguali_prima;
799     int termini_uguali_dopo;
800     int errore;
801
802     if(verifica.controllo == 1){
803
804         i=0;
805         errore=0;
806         termini_diversi=0;
807         termini_uguali_dopo=0;
808         termini_uguali_prima=0;
809         while(i < verifica.dimensione){
810             k=verifica.dimensione-1;
811             termini_uguali_dopo=termini_uguali_prima;
812             while(k > i){
813                 if(verifica.primo_termine[i] == verifica.
                        primo_termine[k]){
814                     if(verifica.secondo_termine[i] != verifica.
                        secondo_termine[k]){
815                         errore=1;
816                         printf("\n_Nel_%d_elemento_c'e'_un_errore_
                                che_impedisce_alla_realzione_binaria\n",k
                                +1);
817                         printf("di_essere_una_funzione\n");
818                         k=i;
819                         i=verifica.dimensione;
820                     }
821                     if(verifica.secondo_termine[i] == verifica.
                        secondo_termine[k])
822                         termini_uguali_dopo++;
823                 }
824                 k--;
825             }
826             if(errore == 0 && termini_uguali_dopo ==
                        termini_uguali_prima)
827                 termini_diversi++;
828
829             termini_uguali_prima = termini_uguali_dopo;
830             i++;
831         }
832         if(errore == 0 && (termini_diversi == (verifica.
                        dimensione - termini_uguali_prima))){
833             printf("\n_La_relazione_binaria_e'_una_funzione\n");
834             check_biiettivita(verifica);

```



```

835 }
836     else
837         printf("\nLa relazione binaria non e' una funzione\n");
838 }
839
840 /****** Controllo se c' e' una funzione per stringhe
    (le stringhe sono considerate come costanti di
    diverso valore) *****/
841
842 if(verifica.controllo == 2){
843     i=0;
844     errore=0;
845     termini_diversi=0;
846     termini_uguali_dopo=0;
847     termini_uguali_prima=0;
848     while(i < verifica.dimensione){
849         k=verifica.dimensione-1;
850         termini_uguali_dopo=termini_uguali_prima;
851         while(k > i){
852             if((strcmp(verifica.prima_stringa[i],verifica.
                        prima_stringa[k])) == 0){
853                 if((strcmp(verifica.seconda_stringa[i],
                        verifica.seconda_stringa[k])) != 0){
854                     errore=1;
855                     printf("\nNel %d elemento c'e' un errore _
                        che impedisce alla realzione binaria\n",k
                        +1);
856                     printf("di essere una funzione\n");
857                     k=i;
858                     i=verifica.dimensione;
859                 }
860                 else
861                     termini_uguali_dopo++;
862             }
863             k--;
864         }
865         if(errore == 0 && termini_uguali_dopo ==
            termini_uguali_prima)
866             termini_diversi++;
867
868         termini_uguali_prima = termini_uguali_dopo;
869         i++;
870     }

```

```

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

```

```

909         if(verifica.primo_termine[i] != verifica.
           primo_termine[k]){
910             errore=1;
911             printf("\n_Nel_%d_elemento_c'e'_un_errore_
                che_impedisce_alla_realzione_binaria\n",k
                +1);
912             printf("di_essere_una_funzione\n");
913             k=i;
914             i=verifica.dimensione;
915         }
916         if(verifica.primo_termine[i] == verifica.
           primo_termine[k])
917             termini_uguali_dopo++;
918     }
919     k--;
920 }
921 if(errore == 0 && termini_uguali_dopo ==
   termini_uguali_prima)
922     termini_diversi++;
923
924     termini_uguali_prima = termini_uguali_dopo;
925     i++;
926 }
927 if(errore == 0 && (termini_diversi == (verifica.
   dimensione - termini_uguali_prima))){
928     printf("\n_La_relazione_binaria_e'_iniettiva\n");
929     iniettivita = 1;
930 }
931 else
932     printf("\n_La_relazione_binaria_non_e'_iniettiva\n")
        ;
933
934
935 }
936
937 /****** Controllo iniettivita ' per stringhe
        *****/
938
939 if(verifica.controllo == 2){
940     i=0;
941     errore=0;
942     termini_diversi=0;
943     termini_uguali_dopo=0;
944     termini_uguali_prima=0;

```

```

945
946 while(i < verifica.dimensione){
947     k=verifica.dimensione-1;
948     termini_uguali_dopo=termini_uguali_prima;
949     while(k > i){
950         if((strcmp(verifica.seconda_stringa[i], verifica.
951             seconda_stringa[k])) == 0){
952             if((strcmp(verifica.prima_stringa[i], verifica.
953                 prima_stringa[k])) != 0){
954                 errore=1;
955                 printf("\n_Nel_%d_elemento_c'e'_un_errore_
956                     che_impedisce_alla_realzione_binaria\n", k
957                     +1);
958                 printf("di_essere_una_funzione\n");
959                 k=i;
960                 i=verifica.dimensione;
961             }
962             if((strcmp(verifica.prima_stringa[i], verifica.
963                 prima_stringa[k])) == 0)
964                 termini_uguali_dopo++;
965         }
966     }
967     k--;
968 }
969 if(errore == 0 && termini_uguali_dopo ==
970     termini_uguali_prima)
971     termini_diversi++;
972 termini_uguali_prima = termini_uguali_dopo;
973 i++;
974 }
975 if(errore == 0 && (termini_diversi == (verifica.
976     dimensione - termini_uguali_prima))){
977     printf("\n_La_relazione_binaria_e'_iniettiva");
978     iniettivita = 1;
979 }
980 else
981     printf("\n_La_relazione_binaria_non_e'_iniettiva");
982 }
983 return(iniettivita);
984 }
985

```

```

981  /*****FUNZIONE PER IL CHECK DELLA
      SURIETTIVITA'*****/
982
983  int check_suriettivita(struct relBin verifica){
984  /***** La suriattivita' sempre verificata in quanto
      il dominio e il codominio *****/
985  /** sono entrambi i rispettivi x,y acquisiti, quindi
      non ho elementi y non associati a x **/
986  int suriattivita;
987
988  suriattivita = 1;
989  return(suriattivita);
990  }
991
992  /*****FUNZIONE PER IL CHECK DELLA
      BIETTIVITA'*****/
993
994  void check_biattivita(struct relBin verifica){
995
996      int      suriattivita ,
997              inattivita;
998
999  suriattivita = check_suriattivita(verifica);
1000  inattivita = check_inattivita(verifica);
1001
1002
1003      if( suriattivita == 1 && inattivita == 1)
1004          printf("\nla funzione e' biattiva");
1005      else
1006          printf("\nla funzione non e' biattiva");
1007  return;
1008  }

```

```

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 }

```

5 Testing the program

Test 1:

Test di Relazione dordine Totale.

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

Outputs: checkriflessivit : 1, checksimmetria : 0, checktransitivit : 1 checkdicotomia : 1, la relazione una relazione dordine totale in quanto rispetta anche la propriet di Dicotomia.

Test 2:

Test di Relazione dordine Parziale.

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

Outputs:checkriflessivit : 1, checksimmetria : 0, checktransitivit : 1 la relazione una relazione dordine parziale in quanto rispetta le propriet.

Test 3:

Test di Relazione dordine non Parziale.

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

Outputs:checkriflessivit : 1, checksimmetria : 0, checktransitivit : 0 la relazione non una relazione dordine parziale in quanto non rispetta le propriet.

Test 4:

Test di Relazione dequivalenza.

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

Outputs:checkriflessivit : 1, checksimmetria : 1, checktransitivit : 1 checkdicotomia : 0, la relazione una relazione dequivalenza in quanto rispetta le propriet.

Test 5:

Test di Relazione non dequivalenza.

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

Outputs:checkriflessivit : 0, checksimmetria : 0, checktransitivit : 0 la relazione

non una relazione d'ordine di 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 è iniettiva.

La relazione binaria è 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_riflessivit : 1,check_simmetria : 1, check_transitivit : 1 dicotomia

:1 la relazione è una relazione d'ordine totale in quanto rispetta le proprietà.

le funzioni funzionano anche con input contenuti negli spazi.

Test 8:

Test per inserire stringhe in una relazione numerica.

Inputs:(1,a)

Outputs: c 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 è iniettiva

La funzione binaria non è biiettiva

6 Verifying the program