Università di Urbino

Informatica Applicata

Programmazione Procedurale e Logica

Relazione

Progetto per la sessione invernale 2014/2015

Studente:
Marco Tamagno
matricola no: 261985

Studente:
Francesco Belacca
matricola no: 260492

Professore:
Marco Bernardo

May 30, 2015

Contents

1 Specifica del Problema

Scrivere un programma ANSI C che acquisisce da tastiera un insieme, una relazione binaria su quell'insieme ed un'operazione binaria su quell'insieme e poi verifica se l'insieme è chiuso rispetto all'operazione e se la relazione è una congruenza rispetto all'operazione.

2 Analisi del Problema

2.1 Input

Il problema prende in pasto come input un insieme, una relazione binaria su quell'insieme e un'operazione binaria su quell'insieme.

2.2 Output

Il problema ha come output il risultato della verifica della chiusura dell'insieme rispetto all'operazione e il risultato della verifica della congruenza della relazione rispetto all'operazione;

2.3 Relazioni tra input ed output

1)Chiusura:
Se due elementi qualsiasi, appartenenti all'insieme preso in considerazione vengono utilizzati come operandi per l'operazione immessa, si dice che l'operazione è chiusa rispetto all'insieme se e solo se anche il risultato dell'operazione appartiene all'insieme.

2)Congruenza:
Una relazione d'equivalenza su un insieme chiuso rispetto ad un'operazione è detta essere una congruenza rispetto a quell'operazione sse, ogni volta che si sostituisce un operando con un altro operando equivalente al primo, si ottiene un risultato equivalente a quello originario.

3 Progettazione dell'algoritmo

3.1 Scelte di progetto

La principale scelta di progetto è quella di restringere l'insieme degli input ai soli numeri.

3.2 Strutture utilizzate

I singoli elementi dell'insieme – acquisibili solo in modo sequenziale – debbono essere salvati in una struttura dati che agevoli la verifica delle proprietà. A tale scopo, risulta particolarmente adeguata una struttura dati che contenga un array unidimensionale e un intero che definisca quanti elementi sono stati acquisiti in totale. Chiameremo questa struttura Insieme, dato che è proprio ciò che deve rappresentare.
Per la relazione binaria invece, risulta più adeguata una struttura dati che contenga due array unidimensionali(uno contenete tutti i primi termini e uno tutti i secondi) insieme ad un altro intero che denoti il numero totale di coppie binarie acquisite. Chiameremo questa struttura relBin.
Infine per l'operazione, non c'è bisogno di salvare gli operandi, sapendo che devono appartenere all'insieme acquisito, perciò abbiamo deciso di chiedere all'utente ogni risultato delle operazioni possibili all'interno dell'insieme acquisito, in un semplice array unidimensionale, dicendogli di inserire 999 nel caso il risultato sia impossibile o indeterminato.

3.3 Passi del programma

-Acquisire e comunicare un insieme.
-Acquisire e comunicare una relazione binaria su quell'insieme.
-Acquisire e comunicare un operazione binaria su quell'insieme.
-Verificare e comunicare la chiusura dell'insieme rispetto all'operazione.
-Verificare e comunicare se la congruenza della relazione rispetto all'operazione.

4 Implementazione dell'algoritmo

Questa è la traduzione dei passi in C:

```
1  /************************************************/
2  /* Progetto per la sessione estiva del 2014/2015 */
3  /************************************************/
4
5  /*****************************/
6  /* inclusione delle librerie */
7  /*****************************/
8
9  #include<stdio.h>
10 #include<stdlib.h>
11 #include<string.h>
12
13 /*********************************/
14 /* dichiarazione delle strutture */
15 /*********************************/
16 typedef struct Operazione
17 {
18     double   *operando_a;
19     double   *operando_b;
20     double   *risultati;
21
22 } operazione_t;
23
24 typedef struct RelBin
25 {
26     /* coppia numerica */
27
28     double  *primo_termine;
29     double *secondo_termine;
30
31     /* variabile per sapere il numero delle coppie */
32
33     int dimensione;
34 } rel_bin;
35
36 typedef struct Insieme
37 {
38     double* elementi_insieme;
39     int numero_elementi;
40 } insieme_t;
41
42 /*********************************/
43 /* dichiarazione delle funzioni */
44 /*********************************/
45
46 int controllo_simmetria (rel_bin);
```

```
47 int controllo_riflessivita (rel_bin);
48 int controllo_transitivita (rel_bin);
49 int relazione_equivalenza (rel_bin);
50 insieme_t acquisisci_insieme(void);
51 rel_bin acquisisci_rel_bin(insieme_t);
52 insieme_t crea_insieme_vuoto(void);
53 int acquisisci_elemento(insieme_t);
54 void stampa(rel_bin);
55 operazione_t acquisisci_operazione(insieme_t);
56 int controllo_chiusura(insieme_t,operazione_t);
57 void controllo_congruenza(rel_bin,insieme_t,operazione_t,int);
58
59 /*****************/
60 /* funzione main */
61 /*****************/
62
63 int main()
64 {
65     operazione_t operazione;
66     char carattere_non_letto;
67     int scelta;
68     int lettura_effettuata;
69     int ripeti;
70     int chiusura;
71
72     /* variabili per insieme e relazione */
73
74     insieme_t insieme;
75     rel_bin relazione;
76
77     /*inizializzo le variabili*/
78     ripeti = 1;
79     scelta = 0;
80     lettura_effettuata = 0;
81     chiusura = 1;
82
83     while(ripeti == 1)
84     {
85     printf("\n ************************************");
86     printf("************************\n");
87     printf("\n Questo programma acquisisce nel seguente");
88     printf(" ordine:\n");
89     printf("\n 1) Un insieme;\n 2) Una relazione binaria su ");
90     printf("quell'insieme;\n 3) Un'operazione binaria su quell");
91     printf("'insieme.\n\n Poi verifica se l'insieme e' chiuso ");
92     printf("rispetto all'operazione \n ");
93     printf(" e se la relazione e' una");
94     printf(" congruenza rispetto all'operazione.\n");
95     printf("\n *********************************");
```

```
96      printf("****************************\n");
97      printf("\n\n Digitare:\n 1 - se si vuole iniziare con");
98      printf(" l'acquisizione dell'insieme,\n 2 - se si vuole ");
99      printf("inserire l'insieme vuoto,");
100     printf("\n 3 - terminare il programma: ");
101
102        do
103        {
104            lettura_effettuata = scanf("%d",&scelta);
105            if(lettura_effettuata != 1)
106            {
107                do
108                    carattere_non_letto = getchar();
109                while (carattere_non_letto != '\n');
110                scelta=4;
111            }
112        }
113        while((scelta != 1 && scelta != 2
114      && scelta != 3) || lettura_effettuata != 1);
115
116        if(scelta == 1)
117        {
118            insieme = acquisisci_insieme();
119            relazione = acquisisci_rel_bin(insieme);
120            stampa(relazione);
121            operazione = acquisisci_operazione(insieme);
122            chiusura = controllo_chiusura(insieme, operazione);
123            controllo_congruenza(relazione, insieme, operazione,
124                                 chiusura);
125        }
126        if(scelta == 2)
127        {
128      printf("\n\n ************** INSIEME");
129      printf("VUOTO **********************\n");
130      insieme = crea_insieme_vuoto();
131      printf("\n L'insieme che si e' scelto e' vuoto,");
132      printf(" quindi qualsiasi \n sia la relazione");
133      printf(", simmetria, riflessivita' e transitivita'\n");
134      printf(" sono sempre verificate.\n Per convenzione ");
135      printf("diciamo anche che qualsiasi sia\n l'operazione");
136      printf(" e' chiusa rispetto all'insieme");
137        }
138
139        printf("\n\n Digitare:\n 1 - se si vuole acquisire");
140        printf(" un altro insieme,\n 2 - se si vuole uscire: ");
141
142        do
143        {
144            lettura_effettuata = scanf("%d",&ripeti);
```

```
145             if(lettura_effettuata != 1)
146             {
147                 do
148                     carattere_non_letto = getchar();
149                 while (carattere_non_letto != '\n');
150                 ripeti = 1;
151             }
152         }
153         while(lettura_effettuata != 1 || ripeti != 1 && ripeti != 2);
154     }
155 
156     return 0;
157 }
158 
159 
160 /*****************************/
161 /* acquisizione dell'insieme */
162 /*****************************/
163 
164 insieme_t acquisisci_insieme()
165 {
166     /*dichiaro la struttura insieme*/
167 
168     insieme_t insieme;
169 
170   /*variabile contatore */
171     int  i;
172     /*variabile contatore*/
173   int  j;
174     /*variabile per terminare l'acquisizione*/
175   int  finisci_di_acquisire;
176     /*variabile per l'acquisizione dell'elemento 0*/
177   int  zeri;
178     /*variabile per verificare che la
179   acquisizione vada a buon fine*/
180     int  elemento_acquisito;
181   /*variabile necessaria allo
182   svuotamento del buffer*/
183     char carattere_non_letto;
184   /*variabile per acquisire ogni
185   elemento temporaneamente*/
186     double temporaneo;
187 
188     /*inizializzo le variabili*/
189 
190     elemento_acquisito = 0;
191     j = 0;
192     i = 0;
193     zeri = 0;
```

```
194     temporaneo = 1;
195     insieme.numero_elementi = 50;
196     finisci_di_acquisire = 0;
197
198     /*alloco memoria*/
199     insieme.elementi_insieme = (double *)
200               malloc (insieme.numero_elementi);
201
202     /*inizio la vera e propria acquisizione*/
203
204     printf("\n\n Si e' scelto di acquisire un'insieme\n");
205
206     /*chiedo se l'utente vuole inserire lo 0*/
207
208   printf("\n\n ************** ACQUISIZIONE DELL'");
209   printf("INSIEME **********************\n");
210   printf("\n\n Digitare:\n 1 - se l'elemento 0");
211   printf("appartiene all insieme");
212   printf("\n 2 - nel caso non gli appartiene: ");
213     do
214     {
215         elemento_acquisito = scanf("%d",&zeri);
216         if(elemento_acquisito != 1)
217         {
218             do
219                 carattere_non_letto = getchar();
220             while (carattere_non_letto != '\n');
221         }
222     }
223     while(elemento_acquisito != 1 || zeri != 1 && zeri != 2);
224     if (zeri == 1)
225     {
226     insieme.elementi_insieme = (double *)
227       realloc (insieme.elementi_insieme, (i+1) * sizeof (double));
228     insieme.elementi_insieme[i] = 0;
229     i = 1;
230     }
231
232     /*faccio partire i i+1 se c'e' lo zero*/
233
234     if(zeri == 2)
235         i = 0;
236
237   printf("\n\n Per terminare l'acquisizione digitare 0\n\n");
238
239     while(finisci_di_acquisire != 1)
240     {
241     insieme.elementi_insieme = (double *)
242       realloc (insieme.elementi_insieme, (i+1) * sizeof (double));
```

```
243     printf("\n Digitare ora il %d elemento: ",i+1);
244     elemento_acquisito = scanf("%lf",&temporaneo);
245
246     if(temporaneo == 0)
247     {
248       finisci_di_acquisire = 1;
249       insieme.numero_elementi = i;
250     }
251
252     if(i >= 0)
253       insieme.elementi_insieme[i] = temporaneo;
254
255     for(j = i - 1; j >= 0; j--)
256     {
257       if(elemento_acquisito != 1 ||
258       temporaneo == insieme.elementi_insieme[j])
259       {
260         do
261         carattere_non_letto = getchar();
262         while (carattere_non_letto != '\n');
263         i--;
264         j = 0;
265       }
266
267     }
268     i++;
269   }
270
271
272     /***********************/
273     /* stampa dell'insieme */
274     /***********************/
275   printf("\n\n ************** STAMPA DELL'");
276   printf("INSIEME **************************\n");
277   printf("\n\n L'insieme acquisito e':");
278   printf("\n\n { ");
279     i=0;
280     while(i < insieme.numero_elementi)
281     {
282     printf("%.2lf",insieme.elementi_insieme[i]);
283         if(i+1 < insieme.numero_elementi)
284             printf(" ; ");
285         i++;
286     }
287   printf(" }\n\n");
288
289
290
291     return insieme;
```

```
292 }
293
294 insieme_t crea_insieme_vuoto()
295 {
296     insieme_t insieme;
297     insieme.elementi_insieme = (double *) malloc (1);
298     insieme.numero_elementi = 0;
299     return insieme;
300 }
301
302 rel_bin acquisisci_rel_bin(insieme_t insieme)
303 {
304   printf("\n\n ********* ACQUISIZIONE DELLA");
305   printf("RELAZIONE BINARIA ***************\n");
306     rel_bin relazione;
307
308     int acquisizione_finita,
309         risultato_lettura,
310         i,
311         primo_termine_acquisito;
312
313     char carattere_non_letto;
314
315     acquisizione_finita = 1;
316     primo_termine_acquisito = 0;
317
318     relazione.dimensione = 0;
319     relazione.primo_termine = (double *) malloc (2);
320     relazione.secondo_termine = (double *) malloc (2);
321
322     while (acquisizione_finita == 1)
323     {
324         primo_termine_acquisito = 0;
325         relazione.dimensione++;
326         acquisizione_finita = 2;
327
328         /*Acquisisco i termini della coppia*/
329
330     printf ("\n\n Inserisci i termini della coppia \n ");
331     relazione.primo_termine = (double *)
332       realloc (relazione.primo_termine,
333         (relazione.dimensione+1) * sizeof (double));
334     relazione.secondo_termine = (double *)
335       realloc (relazione.secondo_termine,
336         (relazione.dimensione+1) * sizeof (double));
337         risultato_lettura = 0;
338
339
340         /*Acquisisco il primo termine*/
```

```
341         if (primo_termine_acquisito == 0)
342         {
343             printf (" Primo Termine: ");
344             relazione.primo_termine[relazione.dimensione - 1] =
345         acquisisci_elemento(insieme);
346         }
347         primo_termine_acquisito = 1;
348
349         /*Acquisisco il secondo termine*/
350         if (primo_termine_acquisito == 1)
351         {
352       printf (" Secondo Termine: ");
353       relazione.secondo_termine[relazione.dimensione - 1]
354       = acquisisci_elemento(insieme);
355       for(i=relazione.dimensione-2; i>=0; i--)
356         if(relazione.primo_termine[relazione.dimensione - 1]
357           == relazione.primo_termine[i])
358           if(relazione.secondo_termine[relazione.dimensione -1] ==
                 relazione.secondo_termine[i])
359                   {
360                       relazione.dimensione--;
361                       i = 0;
362                   }
363         }
364
365         /*Chiedo all'utente se ci sono altre coppie*/
366
367         do
368         {
369       printf("\n\n Digitare:\n 0 - per");
370       printf("terminare l'acquisizione,");
371       printf("\n 1 - se si vuole acquisire un altra coppia: ");
372       risultato_lettura = scanf ("%d",
373                                  &acquisizione_finita);
374             if (acquisizione_finita < 0 ||
375         acquisizione_finita > 1 || risultato_lettura != 1)
376                 do
377                     carattere_non_letto = getchar();
378                 while (carattere_non_letto != '\n');
379         }
380         while (acquisizione_finita < 0 || acquisizione_finita > 1 );
381     }
382     return relazione;
383 }
384
385 /************FUNZIONE DI STAMPA****************/
386
387 void stampa (rel_bin stampa)
388 {
```

```
389
390     int i = 0;
391     printf("\n\n ******** STAMPA DELLA RELAZIONE BINARIA *****");
392     printf ("***************\n\n La relazione binaria e':");
393     printf ("\n\n {");
394
395     /******Stampa per coppie numeriche *****/
396
397     while (i < stampa.dimensione)
398     {
399     printf ("(%.2lf,%.2lf)",
400       stampa.primo_termine[i],
401       stampa.secondo_termine[i]);
402         if (i+1 != stampa.dimensione)
403             printf (" ; ");
404         i++;
405     }
406     printf("}\n");
407     return ;
408 }
409
410 int acquisisci_elemento(insieme_t insieme)
411 {
412     /* dichiaro le variabili */
413     char carattere_non_letto;
414
415     int lettura_corretta,
416         i,
417         elemento_trovato;
418
419     double elemento;
420     /* inizializzo le variabili */
421     elemento = 0;
422     lettura_corretta = 1;
423     do
424     {
425         /* controllo che i valori siano stati letti correttamente */
426         /* e nel caso non sia cosi svuoto il buffer */
427         if(lettura_corretta != 1)
428         {
429             do
430                 carattere_non_letto = getchar();
431             while (carattere_non_letto != '\n');
432             printf ("\n C'e'un errore, reinserire il termine e
                    verificare\n");
433             printf(" che appartenga all'insieme precedentemente
                    inserito: \n ");
434         }
435         lettura_corretta = scanf("%lf",&elemento);
```

```
436         /* verifico se l'elemento che si vuole utilizzare nella
                relazione */
437         /* e' presente nell'insieme inserito */
438         elemento_trovato = 0;
439         for(i=0; i < insieme.numero_elementi; i++)
440             if(elemento == insieme.elementi_insieme[i])
441                 elemento_trovato = 1;
442
443         if(elemento_trovato == 0)
444             lettura_corretta = 0;
445     }
446     while(lettura_corretta == 0);
447
448     return elemento;
449 }
450
451
452 /* Acquisisco l'operazione*/
453
454 operazione_t acquisisci_operazione(insieme_t insieme)
455 {
456     operazione_t operazione;
457     int i,
458         j,
459         dimensione;
460     double *risultati;
461
462     i = 0;
463     j = 0;
464     dimensione = 0;
465
466     operazione.risultati = (double *) malloc (2);
467     operazione.operando_a = (double *) malloc (2);
468     operazione.operando_b = (double *) malloc (2);
469     printf("\n\n ********* ACQUISIZIONE DELL'OPERAZIONE
            **********************\n");
470     printf(" \n\n Inserire ora i risultati dell'operazioni: \n");
471     printf(" \n Digitare 999 per risultati impossibili o
            indeterminati. \n");
472     for(i = 0; i < insieme.numero_elementi; i++)
473     {
474         for(j = 0; j < insieme.numero_elementi; j++)
475         {
476             operazione.risultati = (double *) realloc (operazione.
                    risultati, (dimensione+1) * sizeof (double));
477             operazione.operando_a = (double *) realloc (operazione.
                    operando_a, (dimensione+1) * sizeof (double));
478             operazione.operando_b = (double *) realloc (operazione.
                    operando_b, (dimensione+1) * sizeof (double));
```

```
479             operazione.operando_a[dimensione] = insieme.
                    elementi_insieme[i];
480             operazione.operando_b[dimensione] = insieme.
                    elementi_insieme[j];
481             printf("\n %f * %f = ",insieme.elementi_insieme[i],
                    insieme.elementi_insieme[j]);
482             scanf("%lf",&operazione.risultati[dimensione]);
483             dimensione++;
484         }
485     }
486     return operazione;
487 }
488
489 int controllo_chiusura(insieme_t insieme,operazione_t operazione)
490 {
491     int i,
492         j,
493         chiusura;
494
495     i = 0;
496     j = 0;
497     chiusura = 0;
498
499     for(i = 0; i<(insieme.numero_elementi*insieme.numero_elementi);
            i++)
500     {
501         chiusura = 0;
502         if(operazione.risultati[i] != 999)
503             for(j=0; j<insieme.numero_elementi; j++)
504                 if(operazione.risultati[i] == insieme.elementi_insieme
                        [j])
505                 {
506                     chiusura = 1;
507                     j = insieme.numero_elementi+1;
508                 }
509         if(chiusura == 0)
510             i = (insieme.numero_elementi*insieme.numero_elementi);
511     }
512     printf("\n\n *********** CHIUSURA
            ****************************************\n");
513     if(chiusura == 0)
514         printf("\n\n La chiusura non e' verificata\n");
515     if(chiusura == 1)
516         printf("\n\n La chiusura e' verificata\n");
517
518     return chiusura;
519 }
520
521 int controllo_riflessivita (rel_bin verifica)
```

```
522 {
523
524    int i,
525        j,
526        k,
527        riscontro,
528        secondo_riscontro,
529        riflessivita;
530
531    riflessivita = 1;
532    i = 0;
533    j = 0;
534    k = 0;
535    riscontro = 0;
536    secondo_riscontro = 0;
537
538    /*Verifica riflessivita'*/
539
540    /*Definizione: una relazione per la quale esiste almeno un
           elemento che non e'in relazione
541   con se' stesso non soddisfa la definizione di riflessivita'*/
542
543    while ( (i < verifica.dimensione) && (k < verifica.dimensione))
544    {
545
546        /*Verifica riflessivita' per numeri*/
547
548        riscontro = 0;
549        secondo_riscontro = 0;
550        if (verifica.primo_termine[i] == verifica.secondo_termine[i])
551            riscontro++; /****Controllo se c'e' stato un riscontro a,
                   a****/
552        secondo_riscontro++;
553        if (riscontro != 0)
554        {
555            i++;
556            k++;
557        }
558        /**/
559        else
560        {
561            j = 0;
562            riscontro = 0;
563            secondo_riscontro = 0;
564
565            /***************** Controllo la riflessivita' per gli
                   elementi del primo insieme
                   ****************************/
566
```

```
567             while (j < verifica.dimensione)
568             {
569                 if (j == i)
570                     j++;
571                 else
572                 {
573                     if (verifica.primo_termine[i] == verifica.
                            primo_termine[j])
574                         if (verifica.primo_termine[j] == verifica.
                                secondo_termine[j])
575                             riscontro++;
576 
577                     j++;
578                 }
579             }
580 
581             j = 0;
582 
583             /***************** Controllo la riflessivita' per gli
                    elementi del secondo insieme
                    ****************************/
584 
585             while (j < verifica.dimensione)
586             {
587                 if (j == k)
588                     j++;
589                 else
590                 {
591                     if (verifica.secondo_termine[k] == verifica.
                            secondo_termine[j])
592                         if (verifica.primo_termine[j] == verifica.
                                secondo_termine[j])
593                             secondo_riscontro++;
594 
595                     j++;
596                 }
597             }
598             if (riscontro != 0)
599                 i++;
600 
601             /**** Se non c'e' stato un riscontro di riflessivita'
                    esco e imposto la riflessivita' a 0 *****/
602 
603             else
604             {
605                 i = verifica.dimensione;
606                 riflessivita = 0;
607             }
608 
```

```
609             if (secondo_riscontro != 0)
610                 k++;
611 
612             else
613             {
614                 k = verifica.dimensione;
615                 riflessivita = 0;
616             }
617         }
618 
619     }
620 
621 
622     /********** Fine riflessivita **********************/
623     return (riflessivita);
624 }
625 
626 int controllo_transitivita (rel_bin verifica)
627 {
628 
629     int i,
630         j,
631         k,
632         transitivita;
633 
634     /*IMPOSTO LA TRANSITIVITA' INIZIALMENTE COME VERA E AZZERO I
            CONTATORI*/
635     transitivita = 1;
636     i = 0;
637     j = 0;
638     k = 0;
639 
640     /*VERIFICA TRANSITIVITa' PER NUMERI*/
641 
642 
643     while (i < verifica.dimensione)
644     {
645         j = 0;
646 
647         while (j < verifica.dimensione)
648         {
649             k = 0;
650 
651             if (verifica.secondo_termine[i] == verifica.primo_termine
                    [j])
652             {
653                 transitivita = 0;
654 
655                 while (k < verifica.dimensione)
```

```
656                 {
657                     if (verifica.primo_termine[i] == verifica.
                            primo_termine[k])
658                     {
659                         if (verifica.secondo_termine[k]==verifica.
                                secondo_termine[j])
660                         {
661                             transitivita = 1;
662                             k = verifica.dimensione;
663                         }
664                     }
665 
666                     k++;
667                 }
668 
669                 if (transitivita==0)
670                 {
671                     j = verifica.dimensione;
672                     i = verifica.dimensione;
673                 }
674             }
675 
676             j++;
677         }
678 
679         i++;
680     }
681 
682     /************ Fine controllo Transitivita' ************/
683 
684     return (transitivita);
685 
686 }
687 
688 
689 int relazione_equivalenza (rel_bin verifica)
690 {
691 
692     int riflessivita,
693         simmetria,
694         transitivita,
695         equivalenza;
696 
697     equivalenza = 0;
698     riflessivita = controllo_riflessivita(verifica);
699     simmetria = controllo_simmetria(verifica);
700     transitivita = controllo_transitivita(verifica);
701 
702     if (riflessivita == 1 && simmetria == 1 && transitivita == 1)
```

```
703     {
704         printf ("\n e' una relazione di equivalenza\n");
705         equivalenza=1;
706     }
707 
708     if (riflessivita == 0)
709         printf ("\n non e'una relazione di equivalenza perche' e' non
                riflessiva\n");
710 
711     if (simmetria == 0)
712         printf ("\n non e'una relazione di equivalenza perche' e' non
                simmetrica\n");
713 
714     if (transitivita == 0)
715         printf ("\n non e'una relazione di equivalenza perche' e' non
                transitiva\n");
716 
717     return equivalenza;
718 }
719 
720 int controllo_simmetria (rel_bin verifica)
721 {
722 
723     int i,
724         j,
725         riscontro,
726         simmetria;
727 
728     simmetria = 1;
729 
730 
731     i = 0;
732     j = 0;
733     riscontro = 0;
734 
735     /*controllo della simmetria per numeri*/
736 
737     while ( i < verifica.dimensione)
738     {
739 
740         j = 0;
741         while ( j < verifica.dimensione)
742         {
743 
744             if (verifica.primo_termine[i] == verifica.secondo_termine
                    [j])
745                 if (verifica.primo_termine[j] == verifica.
                        secondo_termine[i])
746                     riscontro++;
```

```
747             j++;
748         }
749 
750         if (riscontro == 0)
751         {
752             j = verifica.dimensione;
753             i = verifica.dimensione;
754             simmetria = 0;
755         }
756         riscontro = 0;
757         i++;
758     }
759 
760     return (simmetria);
761 }
762 
763 
764 void controllo_congruenza(rel_bin relazione, insieme_t insieme,
        operazione_t operazione,int chiusura)
765 {
766     printf("\n\n ********* CONTROLLO LA CONGRUENZA
            **************************\n");
767     int equivalenza,
768         controllo,
769         i,
770         j,
771         k;
772 
773     equivalenza = relazione_equivalenza(relazione);
774 
775     i = 0;
776     j = 0;
777     k = 0;
778     controllo=1;
779 
780     for(i = 0; i<relazione.dimensione; i++)
781     {
782         for(j=0; j<(insieme.numero_elementi*insieme.numero_elementi);
                j++)
783         {
784             if(relazione.primo_termine[i] == operazione.operando_a[j
                    ])
785                 for(k = 0; k<(insieme.numero_elementi*insieme.
                        numero_elementi); k++)
786                     if(relazione.secondo_termine[i] == operazione.
                            operando_a[k] && operazione.operando_b[j] ==
                            operazione.operando_b[k])
787                         if(operazione.risultati[j] != operazione.
                                risultati[k])
```

```
788                     {
789                         controllo = 0;
790                         k = (insieme.numero_elementi*insieme.
                                numero_elementi);
791                         j = (insieme.numero_elementi*insieme.
                                numero_elementi);
792                         i = relazione.dimensione;
793                     }
794             if(relazione.primo_termine[i] == operazione.operando_b[j
                    ])
795                 for(k = 0; k<(insieme.numero_elementi*insieme.
                        numero_elementi); k++)
796                     if(relazione.secondo_termine[i] == operazione.
                         operando_b[k] && operazione.operando_a[j] ==
                         operazione.operando_a[k])
797                         if(operazione.risultati[j] != operazione.
                             risultati[k])
798                         {
799                             controllo = 0;
800                             k = (insieme.numero_elementi*insieme.
                                    numero_elementi);
801                             j = (insieme.numero_elementi*insieme.
                                    numero_elementi);
802                             i = relazione.dimensione;
803                         }
804         }
805     }
806
807
808     if(equivalenza == 0 || controllo == 0 || chiusura == 0)
809         printf("\n\n La cogruenza non e' verificata\n");
810     else
811         printf("\n\n La congruenza e' verificata\n");
812
813     return;
814 }
```